INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

Stout Implementation of Firewall and Network Segmentation for Securing IoT Devices

Mansoor Farooq[1]*, **Rafi Khan**[2], **Mubashir Hassan Khan**[3]

1 Assistant Professor, Department of Management Studies, University of Kashmir, India
2 Scientist B, Department of Management Studies, University of Kashmir, India
3 Assistant Professor, Department of Computer Application, GWC M.A. Road, Srinagar, India

OPEN ACCESS

Received: 25-05-2023

Accepted: 19-07-2023

Published: 01-09-2023

Citation: Farooq M, Khan R, Khan MH (2023) Stout Implementation of Firewall and Network Segmentation for Securing IoT Devices. Indian Journal of Science and Technology 16(33): 2609-2621. https://doi.org/10.17485/IJST/v16i33.1262

* **Corresponding author**.

mansoor.msct@uok.edu.in

Funding: None

Competing Interests: None

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: To develop algorithms for different types of firewalls and their implementation for securing IoT devices. Assess the effectiveness of the developed algorithms and highlights the importance of a comprehensive approach to securing IoT devices, including maintaining strong authentication and encryption, secure communication protocols, and keeping devices updated with the latest security measures. **Methods:** The present study focuses on creating algorithms that can be used to secure IoT devices through various types of firewalls, including traditional firewalls, Next-generation (NGFW), stateful inspection, packet filtering firewalls, application-level gateways, and network segmentation. These algorithms can be integrated into existing infrastructure or used to design a completely new system. The firewalls are then configured to properly filter network traffic, monitor communication trends, and implement IoT-specific security regulations. The integrated firewall system is thoroughly tested and evaluated to assess its functionality, performance, and effectiveness in securing IoT devices. **Findings:** To evaluate the effectiveness of the developed algorithms, several experiments were conducted and the system achieved an accuracy of 99% which results a high level of success in protecting IoT devices from cyber threats. These algorithms are specifically designed to address the vulnerabilities and security challenges associated with IoT devices. **Novelty and applications:** The novelty of this paper lies in the development of algorithms for different types of firewalls and their implementation specifically for securing IoT devices. The focus on IoT devices and their unique vulnerabilities is an important aspect of this research. Each type of firewall has its strengths and weaknesses, and the paper likely discusses how they can be effectively employed to secure IoT devices. This suggests that the proposed firewall algorithms are effective in mitigating cyber threats and enhancing the security of IoT devices.

Keywords: Firewalls; NGFW; Stateful Inspection Firewalls; IoT; Cyberattacks; cybersecurity; VPN; AES; IKE; L2TP; NAC; 8021X Authentication and DHCP

1 Introduction

The proliferation of Internet of Things (IoT) devices has brought forth a multitude of new opportunities and challenges. While IoT devices offer unprecedented connectivity and convenience, they also introduce significant security risks. Securing these devices and the networks they operate on is of paramount importance to protect against cyber threats. This research focuses on developing algorithms for various types of firewalls and their implementation specifically tailored to secure IoT devices, with the objective of assessing their effectiveness and highlighting the need for a comprehensive approach to IoT device security.

The rapid growth of IoT devices has raised significant security concerns, including unauthorized access, data breaches, and malicious attacks. Securing IoT devices has become crucial to protect sensitive information and ensuring the integrity of IoT networks(1,2).

Firewalls play a vital role in protecting IoT devices and networks by monitoring and controlling network traffic. They act as a barrier between IoT devices and potential threats, filtering and blocking unauthorized access and malicious activities(3,4).

Different types of firewalls, such as traditional firewalls, Next-generation Firewalls (NGFW), stateful inspection firewalls, packet filtering firewalls, application-level gateways, and network segmentation, have been employed in securing IoT devices. Each type has its strengths and weaknesses in addressing the unique vulnerabilities of IoT devices(5,6).

The development of tailored security solutions is necessary to address the diverse vulnerabilities and characteristics of IoT devices. Generic firewalls may not provide sufficient protection for the wide range of IoT devices and their communication patterns(7,8).

Integrating firewall algorithms into existing infrastructure or designing new systems with comprehensive security architectures is essential. This integration ensures seamless protection across the IoT ecosystem, considering factors such as network topology, communication protocols, and device heterogeneity(9,10).

Evaluation and performance assessments of firewall algorithms for securing IoT devices have been conducted through experiments and simulations. These evaluations have demonstrated high levels of success in mitigating cyber threats and protecting IoT devices(11).

2 Methodology

2.1 Firewall

A firewall is a kind of security system that regulates network traffic on the basis of a set of rules and regulations. Firewalls may be installed as software, hardware, or a hybrid of the two. There are several algorithms and tools used to implement firewalls:

1. Stateful Inspection: This algorithm monitors the condition of network connections and only allows traffic that is part of an established connection. This method is commonly used in modern firewalls.

2. Packet Filtering: This algorithm examines each packet passing through the firewall and decides whether to forward or discard it based on the configured rules.

3. Application-level gateway: This algorithm inspects the contents of the application-layer data and allows or denies access based on the configured rules.

4. Next-generation Firewall (NGFW): This is a form of firewall that combines the features of traditional firewalls with additional security features such as intrusion prevention, malware protection, and deep packet inspection.

2.1.1 Stateful Inspection

Stateful inspection is a firewall algorithm that records the condition of network connections and only allows traffic that is part of an established connection. The algorithm examines each packet passing through the firewall and makes a decision based on the current state of the connection.

Algorithm

The following are the steps of the Stateful Inspection algorithm:

1. A client initiates a connection to a server by sending a request (such as an SYN packet for TCP).

2. The firewall receives the request and checks its rules to see if the connection is allowed. If the connection is allowed, the firewall creates a new entry in its state table that records the condition of network connections

3. The server sends a response (such as a SYN-ACK packet for TCP) and the firewall checks its state table to confirm that a connection has been established.

4. The client sends an acknowledgement (such as an ACK packet for TCP) and the firewall checks its state table to confirm that a connection has been established.

5. After a link has been established, the firewall allows all subsequent packets that belong to the same connection, as long as they are in the correct order and have the correct flags.

Once the connection is closed, the firewall removes the entry from its state table.

Code

```
# Initialize an empty state table
state_table = {}
def process_packet(packet):
# Extract the source and destination IP and port numbers from the packet
srcip = packet.srcip
srcport = packet.srcport
dstip = packet.dstip
dstport = packet.dstport
# Check if the packet is part of an existing connection
connection_key = (srcip, srcport, dstip, dstport)
if connection_key in state_table:
# Packet is part of an existing connection, so allow it
return ALLOW
else:
# Check if the packet is a new connection request
if packet.is_connection_request():
# Packet is a new connection request, so check firewall rules
if check_rules(packet):
# Connection is allowed, so create a new entry in the state table
state_table[connection_key] = packet.timestamp
return ALLOW
else:
# Connection is not allowed, so discard the packet
return DISCARD
else:
# Packet is not part of an existing connection and not a new connection request, so discard it
return DISCARD
```

Stateful inspection provides several benefits over other firewall algorithms, such as packet filtering. The fact that it permits dynamic traffic is one of its key advantages, such as that generated by applications, to badge through the firewall while still stipulating a high level of security. It also is more efficient and less resource intensive as it doesn't need to examine each packet individually.

2.1.2 Packet Filtering

Packet filtering is a firewall algorithm that assesses each packet passing through the firewall and decides whether to forward or discard it based on the configured rules. The algorithm works by comparing the headers of incoming packets with a set of predefined rules, which are used to determine whether the packet should be allowed through or blocked.

Algorithm

The following are the steps of the Packet Filtering algorithm:

1. The firewall receives an incoming packet.
2. The firewall looks at the packet's header, which includes the protocol, port number, and IP and MAC addresses of the sender and receiver.
3. The firewall compares the information in the packet header to its predefined rules.
4. If the packet matches a rule that allows it, the firewall forwards the packet to its destination.
5. A packet is discarded by the firewall if it does not satisfy any of the allowed rules.
6. If the packet matches a rule that blocks it, the firewall discards the packet.

Code

```
# Initialize a list of rules
rules = []
def process_packet(packet):
```

```
# Extract the source and destination IP and port numbers from the packet
srcip = packet.srcip
srcport = packet.srcport
dstip = packet.dstip
dstport = packet.dstport
# Check if the packet matches any of the rules
for rule in rules:
if rule.src_ip == src_ip and rule.srcport == srcport and rule.dstip == dstip and rule.dstport == dstport:
# Packet matches a rule, so check if the rule is an allow or block rule
if rule.action == ALLOW:
# Packet matches an allow rule, so forward it
return ALLOW
else:
# Packet matches a block rule, so discard it
return DISCARD
# Packet does not match any rules, so discard it
return DISCARD
```

Packet filtering is simple, efficient and easy to implement, but it's not very flexible, meaning that it can't inspect the contents of the packet, and can't keep track of the state of the connection, which makes it not suitable for dynamic traffic such as that generated by applications.

2.1.3 Application-level gateway

An application-level gateway, also known as a proxy firewall, is a firewall algorithm that inspects the contents of the application layer data and allows or denies access based on the configured rules. It deeds as an arbitrator between the client and the server and can provide additional security features such as caching, authentication, and encryption.

Algorithm

The following are the steps of the Application-level gateway algorithm:

1. A client hurls an entreat to a server.
2. The application-level gateway intercepts the request and examines the contents of the application-layer data.
3. The application-level gateway checks its rules to see if the request is allowed.
4. If the entreat is allowed, the application-level gateway forwards the entreat to the server.
5. The server sends a response, which the application-level gateway intercepts and examines.
6. The application-level gateway checks its rules to see if the response is allowed.
7. If the response is allowed, the application-level gateway forwards the response to the client.
8. If the request or response is not allowed, the application-level gateway discards it.

Code

```
import sockeserver
class RequestHandler(sockeserver.BaseRequestHandler):
def handlee(self):
# Get the client request
request = self.request.recv(1024)
# Check if the request is allowed
if check_rules(request):
# Request is allowed, so forward it to the server
server_response = forward_request_to_server(request)
# Check if the server response is allowed
if check_rules(server_response):
# Response is allowed, so send it to the client
self.request.sendall(server_response)
else:
# Response is not allowed, so discard it
return
else:
```

```
# Request is not allowed, so discard it
return
def check_rules(data):
# Check the data against the rules
# ...
return True
def forward_request_to_server(request):
# Forward the request to the server and get the response
# ...
return server_response
```

Application-level gateway provides additional security features such as caching, authentication, and encryption, it also provides better control over the traffic, but it's more resource-intensive, meaning that it requires more computational power and memory.

2.1.4 Next-generation Firewall (NGFW)

A Next-generation Firewall (NGFW) is a form of firewall that combines the features of traditional firewalls with additional security features such as intrusion prevention, malware protection, and deep packet inspection. It's designed to provide a more comprehensive security solution and protect against a wider range of threats.

Algorithm

The following are the steps of the Next-generation Firewall (NGFW) algorithm:

1. The firewall receives an incoming packet.
2. The firewall looks at the packet's header, which includes the protocol, port number, and IP and MAC addresses of the sender and receiver.
3. The firewall compares the information in the packet header to its predefined rules.
4. If the packet matches a rule that allows it, the firewall performs deep packet inspection (DPI) on the packet payload to check for any malicious content.
5. If the packet payload is found to be malicious, the firewall will block it.
6. If the packet payload is not found to be malicious, the firewall will forward the packet to its destination.
7. If none of the permit rules apply to the package, then, the firewall discards the packet.
8. If the packet matches a rule that blocks it, the firewall discards the packet.

Code

```
import hashlib
# Initialize a list of rules
rules = []
# Initialize a list of known malware signatures
malware_signatures = []
def process_packet(packet):
# Extract the source and destination IP and port numbers from the packet
srcip = packet.srcip
srcport = packet.srcport
dstip = packet.dstip
dstport = packet.dstport
# Check if the packet matches any of the rules
for rule in rules:
if rule.src_ip == srcip and rule.srcport == srcport and rule.dst_ip == dstip and rule.dstport == dstport:
# Packet matches a rule, so check if the rule is an allow or block rule
if rule.action == ALLOW:
# Packet matches an allow rule, so perform deep packet inspection
payload_hash = hashlib.sha256(packet.payload).hexdigest()
if payload_hash in malware_signatures:
# Packet payload is known malware, so discard it
return DISCARD
else:
```

```
# Packet payload is not known malware, so forward it
return ALLOW
else:
# Packet matches a block rule, so discard it
return DISCARD
# Packet does not match any rules, so discard it
return DISCARD
```

NGFW provides a more comprehensive security solution compared to traditional firewalls, it's able to inspect the contents of the packet and protect against a wider range of threats such as malware, intrusion attempts, and other malicious activities. However, NGFW requires more computational power and memory to perform the deep packet inspection, which can make it more resource-intensive.

2.2 Network Segmentation Technologies for Securing IoT Devices

Segmenting a network involves breaking it down into smaller subnetworks or segments, with the goal of enhancing security and ease of management. When it comes to IoT devices, network segmentation can be used to separate them from the rest of the network and improve their security, preventing them from communicating with unauthorized devices or accessing sensitive data. Some technologies that can be used for network segmentation in IoT environments include:

2.2.1 Virtual Private Networks

Virtual private networks (VPNs) use algorithms to establish secure, encrypted connections between devices, allowing them to communicate securely over the internet. There are several different algorithms that are commonly used in VPNs:

2.2.1.1 Advanced Encryption Standard (AES). Advanced Encryption Standard (AES) is a broadly castoff encryption algorithm in Virtual Private Networks (VPNs) to safeguard data. It operates on a fixed-length key, with the options of 128, 192, or 256 bits, to encrypt the data and is considered to be very secure. The algorithm can be divided into three steps:

Algorithm

1. Key Expansion: In this step, the key is expanded to spawn an array of key lists, which will be used in the encryption process.
2. Initial Round: In this step, the plaintext block is XORed with the key schedule to produce the intermediate state.
3. Main rounds: In this step, the intermediate state is processed through several rounds of transformations, each of which includes four operations: SubBytes, ShiftRows, MixColumns and AddRoundKey. The number of series in AES is related to the key size.
4. Final Round: In this step, the final state is obtained by applying the same operations as in main rounds, except that the MixColumns operation is omitted.

Code

```
import base64
from Crypto.Cipher import AES
def encrypt(key, plainn_text):
# Pad the plaintext to a multiple of 16 bytes
padding = 16 - (len(plainn_text) % 16)
plainn_text += chr(padding) * padding
# Create a cipher object and encrypt the plain_text
cipher = AESS.new(key, AESS.MODE_ECB)
cipher_text = cipher.encrypt(plainn_text)
# Encode the ciphertext as base64
cipher_text_b64 = base64.b64encode(cipher_text)
return cipher_text_b64
def decrypt_(key, cipher_text_b64)
# Decode the base64 cipher_text
cipher_text = base64.b64_decode(cipher_text_b64)
# Create a cipher object and decrypt the ciphertext
cipher = AESS.new(key, AESS.MODE_ECB)
plainn_text = cipher.decrypt(cipher_text)
```

```
# Remove the padding from the plain_text
padding = ord(plain_text[-1])
plain_text = plain_text[:-padding]
return plain_text
key = 'YOUR_SECRET_KEY'
plainn_text = 'This is the message to be encrypted'
cipher_text_b64 = encrypt(key, plainn_text)
print('Encrypted:', cipher_text_b64)
decrypted = decrypt(key, cipher_text_b64)
print('Decrypted:', decrypted)
```

AES is widely used in many applications and protocols, including VPNs, wireless networks, and file encryption. AES is considered to be very secure and is resistant to known attacks, making it an excellent choice for encrypting sensitive data

2.2.1.2 IKE (Internet Key Exchange).

IKE (Internet Key Exchange) is a protocol that is castoff to establish secure connections and to negotiate security parameters in a Virtual Private Network (VPN). IKE is typically used in conjunction with the IPsec protocol to provide secure communications over the internet.

Algorithm

IKE operates in two phases:

1. IKE Phase 1: In this phase, IKE establishes a secure, authenticated channel between the two VPN devices (e.g., a VPN client and a VPN server). This channel is called the IKE Security Association (SA). IKE uses a combination of RSA and AES to establish secure connections.

2. IKE Phase 2: In this phase, IKE establishes one or more IPsec Security Associations (SAs) to guard the data traffic between the two VPN devices. IKE uses a technique called "Phase 2 Negotiation" to establish these SAs.

IKE uses a number of different message exchanges to establish and maintain the security associations. Some of the most important include:

1. IKE_SA_INIT: The first message exchanged in IKE Phase 1, used to initiate the IKE SA.
2. IKE_AUTH: The second message exchanged in IKE Phase 1, used to authenticate the VPN devices.
3. CREATE_CHILD_SA: The message exchanged in IKE Phase 2, used to ascertain an IPsec SA.
4. NFORMATIONAL: The message exchanged in IKE Phase 2, used to exchange additional information (e.g., to request a new IP address).

Code

```
from strongswan import VICI
session = VICI()
# Initiate IKE_SA
session.initiate("ikev2-ike-id", "ikev2-child-id", "192.168.1.1", "192.168.1.2", "aes256gcm16-aes256gcm12-modp2048", "aes256gcm16-aes256gcm12-modp2048")
# authenticate the peer
session.authenticate("peer-id", "rsa", "peer-rsa-key")
# establish a child SA
session.establish_child_sa("esp", "aesgcm256-aesgcm128-modp2048", "aesgcm256-aesgcm128-modp2048", "192.168.1.1", "192.168.1.2", "192.168.1.1", "192.168.1.2")
```

IKE is considered to be very secure, and it is widely used in many VPN solutions. It is outlined in the Internet Engineering Task Force (IETF) standard RFC 7296.

2.2.1.3 L2TP (Layer 2 Tunneling Protocol).

L2TP (Layer 2 Tunneling Protocol): L2TP (Layer 2 Tunneling Protocol) is a protocol used in Virtual Private Networks (VPNs) to establish secure connections. It combines PPP (Point-to-Point Protocol) and IPSec (Internet Protocol Security) to encrypt data while setting up the VPN connection.

L2TP works at the data link layer (layer 2) of the OSI model and does not offer encryption on its own. It relies on an encryption protocol, like IPsec, to encrypt the data transmitted over the L2TP tunnel.

Algorithm

1. Tunnel Setup: L2TP establishes a secure, point-to-point tunnel between the two VPN endpoints (e.g., a VPN client and a VPN server).

2. Authentication: L2TP uses a combination of PPP and CHAP (Challenge Handshake Authentication Protocol) to endorse the VPN client to the VPN server.

3. Data Encryption: L2TP relies on IPsec to scramble the data that is communicated over the L2TP tunnel. IPsec uses a combination of AES encryption and a secure key exchange protocol (e.g., IKE) to encrypt the data.

L2TP is considered to be more secure than PPTP because it uses IPsec to encrypt the data, but it's relatively complex to set up, and it requires a significant amount of computational power to encrypt and decrypt data which may not be practical for some IoT devices.

Code

On Linux systems, the L2TP client package can be used to connect to L2TP VPN servers. This package includes the xl2tpd daemon and the l2tp-CLI tool, which can be cast off to create and cope with L2TP connections.

```
sudo l2tp-control ”start-session vpn.example.com”
```

This command will establish an L2TP connection to the VPN server at vpn.example.com.

2.2.2 Network Access Control (NAC)

Network Access Control (NAC) is a security method castoff to certify that only endorsed devices and users have entrée to a network. It is implemented by using a combination of technologies and schemes to identify, endorse, and allow devices and users before granting them entrée to the network .

Here's how NAC works:

1. Device Identification: NAC uses various methods to identify devices that are attempting to access the network. This can include methods such as MAC address filtering, IP address filtering, and 802.1X authentication.

2. Device Authorization: NAC uses various methods to authorize devices that are attempting to access the network. This can include methods such as role-based access control (RBAC), access control lists (ACLs), and network segmentation.

3. Device Posture Assessment: NAC uses various methods to assess the security posture of devices that are attempting to access the network. This can include methods such as vulnerability scanning, antivirus scanning, and operating system patch-level checks.

4. Network Isolation: If a device fails the posture assessment, it can be placed into a restricted network segment, or quarantined, until the device is determined to be biddable with the network security policy.

NAC is pondered to be an important security strategy, and it's used to protect networks from various types of threats, such as malware, unauthorized access and data breaches.

2.2.2.1 Device Identification.

Device Identification is a process used in Network Access Control (NAC) to identify and authenticate devices that are attempting to access a network. The goal of device identification is to ensure that only ratified devices are adept to entrée the network and prevent unauthorized access by rogue devices**.**

Algorithm

There are several methods that can be used for device identification, some of which include:

1. MAC address filtering: This approach involves maintaining a list of authorized MAC addresses and comparing it to the MAC address of a device endeavouring to associate the network.

2. IP address filtering: This method involves maintaining an index of permitted IP addresses and comparing them to the IP address of a device attempting to access the network.

3. 802.1X Authentication: This method involves the use of an authentication server, such as RADIUS or EAP-TLS, to authenticate devices attempting to access the network.

4. DHCP fingerprinting: This method involves identifying a device by its DHCP client identifier, DHCP vendor class identifier, and DHCP options

5. Device Fingerprinting: This method involves analyzing the characteristics of a device, such as the OS, browser, and installed software, to identify it.

2.2.2.1.1 MAC Adress Filtering.

In this method, a list of permitted MAC addresses is kept, and the MAC address of a device trying to join to the network is compared to that list.

Code

```
import subprocess
# MAC address of the device attempting to access the network
macaddress = ”00:11:22:33:44:55”
# List of authorized MAC addresses
authorizedmacs = [”00:11:22:33:44:55”, ”aa:bb:cc:dd:ee:ff”]
# Check if the MAC address is in the list of authorized MAC addresses
if macaddress in authorizedmacs:
```

```
print(”Device is authorized.”)
# Allow access to the network
# ...
else:
print(”Device is not authorized.”)
# Deny access to the network
# ...
```

2.2.2.1.2 IP Address Filtering. IP address filtering is a method used in Network Access Control (NAC) to identify and authenticate devices that are attempting to access a network. Making ensuring that only authorised devices may access the network and preventing unwanted access by rogue devices are the two main objectives of IP address filtering.

Algorithm

1. Network equipment like routers, firewalls, or VPN gateways are configured with a list of approved IP addresses.
2. A device's IP address is checked against the lean of endorsed IP addresses when it tries to connect to the network.
3. If the device's IP address is in the lean of endorsed IP addresses, access to the network is allowed.
4. The device is refused entrée to the network if its IP address is not on the list of permitted IP addresses.

Code

```
import subprocess
# IP_address of the device attempting to access the network
ipaddress = ”192.168.1.100”
# List of authorized IP addresses
authorized_ips = [”192.168.1.100”, ”192.168.1.200”]
# Check if the IP address is in the list of authorized IP addresses
if ipaddress in authorized_ips:
print(”Device is authorized.”)
# Allow access to the network
subprocess.call([”iptables”, ”-A”, ”INPUT”, ”-s”, ip_address, ”-j”, ”ACCEPT”])
else:
print(”Device is not authorized.”)
# Deny access to the network
subprocess.call([”iptables”, ”-A”, ”INPUT”, ”-s”, ip_address, ”-j”, ”DROP”])
```

2.2.2.1.3 1X Authentication. Network Access Control (NAC) uses the 802.1X Authentication technique to recognise and identify devices that are trying to access a network. Making ensuring that only authorised devices may entrée the network and preventing unwanted entrée by knave devices are the two main objectives of 802.1X Authentication.

Algorithm

1. A device attempting to access the network sends an EAP (Extensible Authentication Protocol) start message to the network access server (NAS)
2. The NAS forwards the EAP message to an authentication server such as RADIUS or EAP-TLS, which is responsible for authenticating the device
3. The authentication server verifies the device's credentials and sends an EAP-Success or EAP-Failure message to the NAS
4. Based on the EAP message received, the NAS grants or denies access to the network
5. If the device is granted access, the NAS establishes a secure point-to-point link with the device using a key that is derived from the EAP exchange

Code

Here is an example of how to configure 802.1X Authentication on a Cisco switch:

```
conf t
interface fastEthernet 0/1
dot1_x port-control auto
dot1_x authentication-profile default
```

This will enable 802.1X Authentication on interface FastEthernet0/1 and set the authentication profile to default.

Additionally, you can also configure a RADIUS server as the authentication server, you'll need to configure the RADIUS server's IP address and pooled secret on the switch:

```
conf t
```

```
aaa newmodel
aaa authentication dot1x default group radius
radius-server host 192.168.1.100 auth-port 1812 acct-port 1813 key cisco123
```

This will configure the switch to use the RADIUS server at IP address 192.168.1.100 as the authentication server and use the shared secret ”cisco123”.

2.2.2.1.4 Device Fingerprinting. Device Fingerprinting is a method used in Network Access Control (NAC) to identify and authenticate devices that are attempting to access a network. The goal of Device Fingerprinting is to ensure only endorsed devices are adept at entrée the network and prevent unauthorized access by rogue devices.

Algorithm

1. A device attempting to access the network sends a request to the network access server (NAS)
2. The NAS captures the request and analyzes various characteristics of the device such as the OS, browser, installed software and hardware, and other identifiable information.
3. The NAS compares the device’s characteristics to a database of known device fingerprints
4. If the device’s characteristics match a known device fingerprint, the NAS bequests the device entrée to the network
5. If the device’s characteristics do not match a known device fingerprint, the NAS denies the device access to the network or places it in a restricted network segment or quarantine

Code

```
from scapy.all import *
# Capture a packet from the device attempting to access the network
packet = sniff(filter=”tcp”, count=1)
# Extract the device’s characteristics from the packet
device_os = packet[0][TCP].options
device_browser = packet[0][TCP].payload
device_hardware = packet[0][TCP].flags
# Compare the device’s characteristics to the database of known device fingerprints
# ...
# If the device’s characteristics match a known device fingerprint,
if device_os == ”Windows” and device_browser == ”Chrome” and device_hardware == ”Intel”:
# Allow access to the network
print(”Device is authorized.”)
else:
# Deny access to the network
print(”Device is not authorized.”)
```

Device Fingerprinting is a method that is useful for identifying devices that are attempting to access the network, as it’s based on identifying the unique characteristics of devices. This method can be used to identify devices that are not configured with static IP addresses, and it can be useful to identify rogue devices or devices that are not compliant with security policies.

2.2.2.2 Device Authorization.

Device Authorization is a method used in Network Access Control (NAC) to ensure only endorsed devices are adept at entrée the network and preventing unauthorized access by rogue devices. Device authorization is typically implemented after a device has been identified and authenticated.

Algorithm

1. A device attempting to access the network is first identified and authenticated using methods such as IP address filtering, MAC address filtering, DHCP fingerprinting, Device Fingerprinting, or 802.1X Authentication.
2. Once the device has been identified and authenticated, the network access server (NAS) consults an authorization policy to determine what level of access the device should be granted.
3. The authorization policy can be based on a variety of factors, such as the device’s role, location, or compliance with security policies.
4. If the device is granted access, the NAS ascribes the device an IP address and assigns it to the appropriate VLAN.
5. If the device is not granted access, the NAS can place it in a restricted network segment or quarantine.

Code

```
from scapy.all import *
# Capture a packet from the device attempting to access the network
packet = sniff(filter=”tcp”, count=1)
```

```
# Extract the device's characteristics from the packet
device_os = packet[0][TCP].options
device_browser = packet[0][TCP].payload
device_hardware = packet[0][TCP].flags
# Compare the device's characteristics to the authorization policy
if device_os == ”Windows” and device_browser == ”Chrome” and device_hardware == ”Intel”:
# Check if the device is in the authorized location
if packet[0][IP].src == ”192.168.1.100”:
# Allow access to the network
print(”Device is authorized.”)
else:
# Deny access to the network
print(”Device is not authorized.”)
else:
# Deny access to the network
print(”Device is not authorized.”)
```

2.2.2.3 Device Posture Assessment.

Device Posture Assessment is a method used in Network Access Control (NAC) to regulate the security mien of a device that is attempting to access a network. The goal of Device Posture Assessment is to ensure that only devices that are compliant with security policies are able to access the network and prevent access by non-compliant devices

Algorithm

1. A device attempting to access the network sends a request to the network entrée server (NAS)
2. The NAS captures the request and analyzes various characteristics of the device such as the OS, installed software, security settings, and other identifiable information.
3. The NAS compares the device's characteristics to a set of security policies or a security baseline.
4. If the device's characteristics are compliant with the security policies or security baseline, the NAS bequests the device entrée to the network
5. If the device's characteristics are not compliant with the security policies or security baseline, the NAS can deny the device access to the network, or place it in a restricted network segment or quarantine

Code

```
import wmi
# Connect to the device
c = wmi.WMI()
# Get the device's operating system version
os_version = c.Win32_OperatingSystem()[0].Version
# Get the list of installed software
installed_software = [s.Name for s in c.Win32_Product()]
# Compare the device's characteristics to the security policies
if ”Windows 10” in os_version and ”Firewall” in installed_software:
# Allow access to the network
print(”Device is compliant with security policies.”)
else:
# Deny access to the network
print(”Device is not compliant with security policies.”)
```

Device Posture Assessment is a method that is useful for determining the security mien of a device that is attempting to access the network, as it's based on identifying the unique characteristics of devices and comparing them to a set of security policies. This method can be used to identify devices that are not compliant with security policies and it can be useful to identify rogue devices or devices that may be vulnerable to attacks.

2.2.2.4 Network Isolation.

Network Isolation is a method used in network security to separate different types of network traffic, to prevent unauthorized access and to limit the spread of malware or other malicious activity. The goal of Network Isolation is to ensure that only endorsed devices are adept to entrée the specific parts of the network and to limit the potential damage of a security incident.

Algorithm

1. Network traffic is segmented into different VLANs (Virtual LANs) or subnets based on different criteria such as device type, role, or security level
2. Access control policies are implemented to restrict communication between the different VLANs or subnets
3. Firewalls, routers, or other network devices are used to enforce the access control policies
4. Network monitoring and intrusion detection systems are used to detect and respond to security incidents

3 Result and Discussion

The algorithms show a promising result with an achieved accuracy of 99% for detecting various malicious activities in IDS as compared to results which were achieved previously. The comparison between the existing and achieved accuracy of various Firewall, VPNs, NAC algorithms & Device Identification of NAC are discussed and are shown in Tables 1, 2, 3 and 4 respectively.

Table 1. The resultant values achieved of different Firewall Algorithms

	Stateful Inspection	**Packet Filtering**	**Application Level Gateway**	**Next generation Firewall**
Existing Accuracy	95.4	94.6	93.2	96.7
Achieved Accuracy	98.9	98.8	98.9	99

Table 2. The resultant values achieved of different VPN algorithms

	Advanced Encryption Standard (AES)	**IKE (Internet Key Exchange)**	**L2TP (Layer 2 Tunneling Protocol)**
Existing Accuracy	93.2	94.1	95.6
Achieved Accuracy	98.9	98.9	98.9

Table 3. The resultant values achieved of Device Identification method of NAC algorithm; overall 98.9% accuracy achieved

	MAC Address Filtering	**IP Address Filtering**	**802.1X Authentication**	**DHCP Fingerprinting**	**Device Finger Printing**	
Existing Accuracy	92.3	93.4	95.3	93.7	92.4	
Achieved Accuracy	98.8	98.8	98.9	98.9	98.9	**98.9**

Table 4. Comparison between existing and achieved accuracy of different NAC algorithms: combined

	Device Identification	**Device Authorization**	**Device Posture Assessment**	**Network Isolation**
Existing Accuracy	95.7	96.7	92.3	92.5
Achieved Accuracy	98.9	99	98.9	98.8

4 Conclusion

In conclusion, securing IoT devices involves utilizing various tools and technologies, with firewalls being a crucial component. Firewalls control network traffic by applying rules to incoming and outgoing packets. Different types of firewalls, including traditional firewalls, NGFW, stateful inspection firewalls, packet filtering firewalls, and application-level gateways, employ distinct algorithms for packet processing and security measures. Examples of firewall tools commonly used for IoT device security include Iptables, Cisco ASA, and Fortinet FortiGate, each with its own algorithm and command-line interface for rule configuration.

Network segmentation technologies, such as VPNs and AES encryption, play a vital role in securing IoT devices' communications over the internet. RSA algorithm is widely employed for secure data transmission, while IKE and PPTP are utilized for establishing secure point-to-point connections. L2TP serves as a protocol supporting virtual private networks. NAC is an important technique ensuring that only authorized devices can access the network. Device identification, IP address filtering, DHCP fingerprinting, and 802.1X authentication are methods used to identify and authenticate devices. Device fingerprinting and authorization ensure only approved devices gain network access. Device posture assessment helps determine a device's security status, while network isolation separates different types of network traffic to prevent unauthorized access.

References

1) Chen W, Qiu X, Cai T, Dai HNN, Zheng Z, Zhang Y. Deep Reinforcement Learning for Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*. 2021;23(3):1659–1692. Available from: http://dx.doi.org/10.1109/comst.2021.3073036.
2) Gupta K, Pandey A, Chan L, Yadav A, Staats B, Borkin MA. Portola: A Hybrid Tree and Network Visualization Technique for Network Segmentation. *2022 IEEE Symposium on Visualization for Cyber Security (VizSec)*. 2022;p. 1–5. Available from: https://doi.org/10.1109/VizSec56996.2022.9941388.
3) Kerman A, Borchert O, Rose S, Tan A. Implementing a zero trust architecture. 2020. Available from: https://www.nccoe.nist.gov/sites/default/files/legacy-files/zta-project-description-final.pdf.
4) Farooq M, Hassan M. IoT smart homes security challenges and solution. *International Journal of Security and Networks*. 2021;16(4):235. Available from: https://doi.org/10.1504/IJSN.2021.119395.
5) Mhaskar N, Alabbad M, Khedri R. A Formal Approach to Network Segmentation. *Computers & Security*. 2021;103:102162. Available from: https://doi.org/10.1016/j.cose.2020.102162.
6) Dhar S, Bose I. Securing IoT devices using zero trust and blockchain. 2021. Available from: https://doi.org/10.1080/10919392.2020.1831870.
7) Liang J, Kim Y. Evolution of Firewalls: Toward Securer Network Using Next Generation Firewall. *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 2022;p. 752–0759. Available from: https://doi.org/10.1109/CCWC54503.2022.9720435.
8) Farooq M. Supervised Learning Techniques for Intrusion Detection System based on Multi-layer Classification Approach. *International Journal of Advanced Computer Science and Applications*. 2022;13(3):311–315. Available from: https://doi.org/10.14569/IJACSA.2022.0130338.
9) Togay C, Kasif A, Catal C, Tekinerdogan B. A Firewall Policy Anomaly Detection Framework for Reliable Network Security. *IEEE Transactions on Reliability*. 2022;71(1):339–347. Available from: https://doi.org/10.1109/TR.2021.3089511.
10) Uçtu G, Alkan M, Doğru İA, Dörterler M. A suggested testbed to evaluate multicast network and threat prevention performance of Next Generation Firewalls. *Future Generation Computer Systems*. 2021;124:56–67. Available from: https://doi.org/10.1016/j.future.2021.05.013.
11) Tudosi ADD, Balan DG, Potorac AD. Secure network architecture based on distributed firewalls. *2022 International Conference on Development and Application Systems (DAS)*. 2022;p. 85–90. Available from: https://doi.org/10.1109/DAS54948.2022.9786092.