r/ThingsYouWillNeed • 1 mo. ago
TheStocksGuy

Shield your cyberspace: Keep threats at bay with layers of security that play

How This Approach Helps Prevent Attacks

The provided code integrates TCP and UDP traffic handling with attack detection, offering a robust solution for monitoring and securing your network against potential threats. This approach helps in several ways:

Benefits

1. **Real-time Traffic Monitoring**:
   - Continuously monitors both TCP and UDP traffic.
   - Logs all incoming traffic with timestamps, providing a detailed record for analysis.
2. **Attack Detection**:
   - Identifies potential attacks based on patterns in the traffic data.
   - Uses a predefined list of attack signatures (e.g., DDoS, Phishing, Malware) to detect threats.
3. **Detailed Logging**:
   - Maintains a log file with information on all traffic, which can be used to trace and analyze suspicious activities.
   - Includes protocol type, client address, and message content in the logs.
4. **Immediate Alerts**:
   - Prints detected attacks to the console in real-time, allowing for quick identification and response.
   - Helps distinguish between legitimate traffic and potential threats.

Preventive Measures

- **Identifying True Attackers**:
  - By analyzing traffic patterns and content, the system distinguishes between normal and suspicious packets.
  - Ensures that only highly suspect packets, which match known attack signatures, are flagged for further action.
- **Avoiding False Positives**:
  - Minimizes the risk of blocking legitimate visitors by using specific criteria to identify attacks.
  - Employs a systematic approach to assess the likelihood of a packet being an attack before taking any drastic measures.
- **Blocking Abusive Traffic**:
  - Once a packet is identified as an attack, further steps can be implemented to block the source IP or route such traffic to null.
  - Prevents attackers from exploiting network vulnerabilities and causing harm.

- Integrate a functionality to route identified malicious traffic to null or block the source IP.
- Ensure thorough testing to fine-tune the detection mechanisms and minimize false positives.
- Continuously update the list of attack signatures to keep up with evolving threats.

By combining real-time traffic monitoring, detailed logging, and precise attack detection, this approach helps secure your network against potential threats while minimizing the risk of false positives. It ensures that you are alerted to highly suspect packets, allowing you to take appropriate action against malicious attempts to abuse your network.

```
import socket
import select
import json
from datetime import datetime

def combined_server(host='0.0.0.0', port=3000, log_file='server_log.txt'):
    """
    Starts a server that handles both TCP and UDP connections.
    Parameters:
    host (str): The hostname or IP address to bind the server to.
    port (int): The port number to bind the server to.
    log_file (str): The file where traffic logs will be saved.
    """

    # Create TCP socket
    tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_socket.bind((host, port))
    tcp_socket.listen(5)
    print(f"Server is listening on {host}:{port}...")

    # Create UDP socket
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_socket.bind((host, port))

    # List of sockets to monitor
    sockets_list = [tcp_socket, udp_socket]

    while True:
        # Use select to wait for activity on any socket
        read_sockets, _, _ = select.select(sockets_list, [], [])

        for notified_socket in read_sockets:
            if notified_socket == tcp_socket:
                # Handle TCP connections
                connection, client_address = tcp_socket.accept()
                try:
```

```
                        log_traffic('TCP', client_address, data, log_file)
                        # Process the request
                        result = process_request(data)
                        response = json.dumps(result).encode('utf-8')
                        connection.sendall(response)
                except Exception as e:
                    print(f"TCP error: {e}")
                finally:
                    connection.close()
            elif notified_socket == udp_socket:
                # Handle UDP messages
                try:
                    data, address = udp_socket.recvfrom(4096)
                    if data:
                        # Log the traffic
                        log_traffic('UDP', address, data, log_file)
                        # Process the request
                        result = process_request(data)
                        response = json.dumps(result).encode('utf-8')
                        udp_socket.sendto(response, address)
                except Exception as e:
                    print(f"UDP error: {e}")

def log_traffic(protocol, address, data, log_file):
    """
    Logs the traffic data with a timestamp.
    Parameters:
    protocol (str): The protocol used (TCP or UDP).
    address (tuple): The address of the client.
    data (bytes): The data received from the client.
    log_file (str): The file where the log will be saved.
    """
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    log_entry = f"{timestamp} {protocol} from {address}: {data.decode('utf-8')}\n"

    with open(log_file, 'a') as file:
        file.write(log_entry)

    print(log_entry.strip())

def detect_attack(log_file):
    """
    Detects potential attacks by analyzing the log file.
    Parameters:
    log_file (str): The file where the log is saved.
    """
```

```
    attack_ips = set()
    attack_methods = {
        'DDoS': [],
        'Phishing': [],
        'Malware': [],
        'Unauthorized Access': []
        # Add more attack methods as necessary
    }

    for line in lines:
        if "attack detected" in line:
            parts = line.split(' ')
            ip = parts[-1].strip()  # Assuming the IP is at the end of the line
            method = identify_attack_method(line)
            attack_ips.add(ip)
            if method in attack_methods:
                attack_methods[method].append(ip)

    for ip in attack_ips:
        print(f"Possible attack from IP: {ip}")

    for method, ips in attack_methods.items():
        print(f"\n{method} Attacks:")
        for ip in ips:
            print(f" - {ip}")

def identify_attack_method(log_entry):
    """
    Identifies the type of attack based on the log entry.
    Parameters:
    log_entry (str): A single line from the log file.
    Returns:
    str: The type of attack detected.
    """
    if "DDoS" in log_entry:
        return 'DDoS'
    elif "Phishing" in log_entry:
        return 'Phishing'
    elif "Malware" in log_entry:
        return 'Malware'
    elif "Unauthorized access" in log_entry:
        return 'Unauthorized Access'
    else:
        return 'Unknown'

def process_request(data):
```

```
    data (bytes): The data received from the client.
    Returns:
    dict: A dictionary with a message indicating successful processing.
    """
    # Example function to process incoming data
    return {"message": "Request processed successfully"}

if __name__ == "__main__":
    combined_server()
    detect_attack('server_log.txt')
```

Starting the Server

1. **Run the Server**:
   - Execute the script to start the server.
   - The server will listen for both TCP and UDP connections on the specified host and port.
   - You should see a message like: `Server is listening on 0.0.0.0:3000...`
2. **Monitoring Traffic**:
   - The server logs traffic for both TCP and UDP protocols.
   - Traffic data, including timestamps, protocols, client addresses, and messages, is logged to the specified log file (e.g., `server_log.txt`).

Viewing Traffic

1. **Console Output**:
   - For each connection, the console displays logged traffic.
   - For TCP connections, you will see something like: `2024-12-05 17:07:12 TCP from ('127.0.0.1', 12345): Normal TCP traffic`
   - For UDP messages, the output will be similar: `2024-12-05 17:07:14 UDP from ('127.0.0.1', 12345): Normal UDP traffic`
2. **Log File**:
   - The log file records all traffic data.
   - Each entry includes a timestamp, protocol, client address, and received message.
   - Example log entry: `2024-12-05 17:07:12 TCP from ('127.0.0.1', 12345): Normal TCP traffic`

Detecting Attacks

1. **Running Detection**:
   - The `detect_attack` function analyzes the log file to identify potential attacks.
   - This function checks for known attack patterns (e.g., DDoS, Phishing, Malware).

- The type of attack is also indicated, e.g., `DDoS Attacks: - 127.0.0.1`

Normal Operation

- **No Attacks**:
  - When no attacks are occurring, the server continuously logs and processes incoming traffic.
  - The console and log file will show normal traffic entries without any attack alerts.

Ensuring Proper Monitoring

- **Static Connections**:
  - The server maintains and monitors static connections efficiently.
  - By analyzing the logged data, you can ensure the server is online and capable of detecting potential threats.

With this setup, you can effectively monitor and detect potential attacks on your network. The detailed logging and real-time console alerts help in maintaining a secure and responsive network environment.

```
import socket
import select
import json
import logging
from datetime import datetime
import re

def setup_logging(log_file):
    logging.basicConfig(filename=log_file, level=logging.INFO,
                        format='%(asctime)s %(levelname)s %(message)s')

def combined_server(host='0.0.0.0', port=80, log_file='server_log.txt'):
    """
    Starts a server that handles both TCP and UDP connections.
    Parameters:
    host (str): The hostname or IP address to bind the server to.
    port (int): The port number to bind the server to.
    log_file (str): The file where traffic logs will be saved.
    """
    setup_logging(log_file)
    logging.info("Server starting...")

    tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_socket.bind((host, port))
    tcp_socket.listen(5)
    logging.info(f"TCP server is listening on {host}:{port}...")

    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_socket.bind((host, port))
    logging.info(f"UDP server is listening on {host}:{port}...")

    sockets_list = [tcp_socket, udp_socket]

    while True:
        read_sockets, _, _ = select.select(sockets_list, [], [])

        for notified_socket in read_sockets:
            if notified_socket == tcp_socket:
                handle_tcp_connection(tcp_socket)
            elif notified_socket == udp_socket:
                handle_udp_connection(udp_socket)

def handle_tcp_connection(tcp_socket):
    connection, client_address = tcp_socket.accept()
```

```
            log_traffic('TCP', client_address, data)
            result = process_request(data)
            response = json.dumps(result).encode('utf-8')
            connection.sendall(response)
    except Exception as e:
        logging.error(f"TCP error: {e}")
    finally:
        connection.close()

def handle_udp_connection(udp_socket):
    try:
        data, address = udp_socket.recvfrom(4096)
        if data:
            log_traffic('UDP', address, data)
            result = process_request(data)
            response = json.dumps(result).encode('utf-8')
            udp_socket.sendto(response, address)
    except Exception as e:
        logging.error(f"UDP error: {e}")

def log_traffic(protocol, address, data):
    logging.info(f"{protocol} from {address}: {data.decode('utf-8')}")
    if detect_packet_injection(data):
        logging.warning(f"Possible packet injection from {address}")

def detect_packet_injection(data):
    packet_injection_pattern = re.compile(r'suspicious_pattern')  # Replace with actual patter
    return bool(packet_injection_pattern.search(data.decode('utf-8')))

def detect_attack(log_file):
    with open(log_file, 'r') as file:
        lines = file.readlines()

    attack_ips = set()
    attack_methods = {
        'DDoS': [],
        'Phishing': [],
        'Malware': [],
        'Unauthorized Access': []
    }

    for line in lines:
        if "attack detected" in line:
            parts = line.split(' ')
            ip = parts[-1].strip()
            method = identify_attack_method(line)
            attack_ips.add(ip)
```

```
    for ip in attack_ips:
        print(f"Possible attack from IP: {ip}")

    for method, ips in attack_methods.items():
        print(f"\n{method} Attacks:")
        for ip in ips:
            print(f" - {ip}")

def identify_attack_method(log_entry):
    if "DDoS" in log_entry:
        return 'DDoS'
    elif "Phishing" in log_entry:
        return 'Phishing'
    elif "Malware" in log_entry:
        return 'Malware'
    elif "Unauthorized access" in log_entry:
        return 'Unauthorized Access'
    else:
        return 'Unknown'

def process_request(data):
    return {"message": "Request processed successfully"}

if __name__ == "__main__":
    combined_server()
    detect_attack('server_log.txt')
```





Create


Updated again but reddit wouldn't let me update anything or comment it. So here is the github to the update version.

https://github.com/BadNintendo/AntiAttacker/tree/main


1



Approved 16 days ago