드론해킹방어대회 문제풀이 보고서

팀명	LS2K
팀장	서민재
팀원	김슬기, 권정은, 이경림
제출일	2023-11-17

목차

1. Easy to audit code

1) 문제 분석

[key1]~[key4]까지의 힌트를 이용하여서 문제 파일에 숨겨져 있는 키 값을 찾아내야한다. 파일의 개수가 많기 때문에 문제의 조건을 토대로 문자열 검색, 디렉터리 검색, 단어 검색 등을 활용하여 적절한 키워드를 찾아 키값이 존재하는 파일을 찾아내어 키값을 추출한다.

2) 문제 풀이

[key1] : Search for source codes regarding "pulse with modulation" and controlling the motor from external sources (e.g., remote control). Look for "count status".

Count status를 찾으라는 문제 조건을 토대로 터미널에서 count status를 가진 문자열이 존재하는지 검색을 해보았으나 존재하지 않음을 확인하였다.

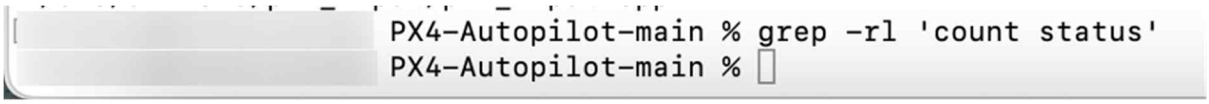


Figure 1. 터미널에서 "count status" 문자열을 검색하였을 때 결과

Figure 1은 grep –rl 'count status' 명령어를 사용하여 count status를 검색하였을 때 나온 결과 값이며 아무것도 존재 하지 않는다.

이에 "count" 문자열이 존재하는 파일이 있는지 재검색을 하였고, 해당 문자열이 사용되는 디렉터리 또는 파일들을 확인하였다. 해당 결과를 Figure 2에 첨부하였다.

```
./src/drivers/optical_flow/px4flow/px4flow.cpp
./src/drivers/optical_flow/px4flow/i2c_frame.h
./src/drivers/rpm/pcf8583/parameters.c
./src/drivers/rpm/pcf8583/PCF8583.cpp
./src/drivers/rpm/pcf8583/PCF8583.hpp
./src/drivers/transponder/sagetech_mxs/SagetechMXS.hpp
./src/drivers/transponder/sagetech_mxs/sg_sdk/LICENSE
./src/drivers/transponder/sagetech_mxs/sg_sdk/target.c
./src/drivers/transponder/sagetech_mxs/sg_sdk/target.h
./src/drivers/transponder/sagetech_mxs/SagetechMXS.cpp
./src/drivers/smart_battery/batmon/batmon.cpp
./src/drivers/rc/crsf_rc/QueueBuffer.hpp
./src/drivers/rc/crsf_rc/CrsfRc.hpp
./src/drivers/rc/crsf_rc/CrsfRc.cpp
./src/drivers/rc/crsf_rc/QueueBuffer.cpp
```

Figure 2. 터미널에서 "count" 문자열을 검색했을 때 결과

이제 다른 조건인 "pulse with modulation(PWM)" 키워드를 토대로 해당 키워드와 관련있는 디렉터리 또는 파일을 찾되, count 값이 16진수라는 문제 조건을 토대로 "0x"로 키값은 시작할 것이라고 판단하였기에 PWM모듈과 관련이 있고, count =0x라는 조건으로 새로 검색을 시도하였다. 해당 결과는 Figure 3에 첨부하였다.

```
PX4-Autopilot-main % grep -rl 'count=0x'
./src/drivers/pwm_input/pwm_input.cpp
```

Figure 3. 터미널에서 "count=0x" 문자열을 검색했을 때 결과

grep –rl 'count=0x' 토대로 pwm과 관련된 파일로 ***./src/drivers/pwm_input/pwm_input.cpp*** 를 찾아내었고, 해당 파일 안에서 count를 검색한 결과, 주석으로 키 값이 숨어있는 것을 발견하였다. 해당 키 값은 Figure 4에서 확인할 수 있다.

```
int
PWMIN::print_status()
{
        PX4_INFO("count=%u period=%u width=%u", //count=0xDA8DFC31, period= width=
                 static_cast<unsigned>(_pulses_captured),
                 static_cast<unsigned>(_last_period),
                 static_cast<unsigned>(_last_width));
        return 0;
}
```

Figure 4. [key1]의 키 값

이를 토대로 [key1]의 값은 **0xDA8DFC31**이다.

[key2] Look for collision prevention feature, using the Google Test framework. There will be a special comments indicating the expected index for the "set point".

Count status를 찾으라는 문제 조건을 토대로 터미널에서 count status를 가진 문자열이 존재하는지 검색을 해보았으나 존재하지 않음을 확인하였다

Google Test가 무엇인지 모르기에, 이것이 무엇인지 구조를 검색을 통하여 살펴보았다.

```
TEST(TestSuiteName, TestName) {
  ...
}
```

```
// Tests factorial of 0.
TEST(FactorialTest, HandlesZeroInput) {
  EXPECT_EQ(Factorial(0), 1);
}

// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(Factorial(1), 1);
  EXPECT_EQ(Factorial(2), 2);
  EXPECT_EQ(Factorial(3), 6);
  EXPECT_EQ(Factorial(8), 40320);
}
```

Figure 5. Test 구조

Figure 5는 Test의 구조를 찾아본 결과이며, 이를 토대로 코드에는 TEST 혹은 TEST_F라는 문자열이 존재할 것이라고 판단하였다. 키 값은 해당 파일에 존재할 것이 기에,

*Find . –type f –print | xargs grep – I "Test"*를 이용하여 검색하였으나 많은 파일들이 출력되어 다른 조건을 더 찾아보았다. Collision prevent (충돌 방지) 라는 조건에 초점을 두어 파일 범위를 더 줄이기 위하여 이를 Figure 6과 같이 명령어를 이용하여 검색을 시도하였고, Figure 7과 같은 결과값을 확인하였다.

```
user@DESKTOP-DLU7MSD MINGW64 ~/Desktop/Hanbat.Univ/HackTheDrone/P1/PX4-Autopilot-main
$ find ./src/lib/collision_prevention -type f -print | xargs grep -i "TEST"
```

Figure 6. 충돌 방지 조건과 TEST 조건을 이용한 문자열 검색

Figure 7을 확인하였을 때, 해당 조건에 해당되는 것은 ***CollisionPreventionTest.cpp*** 에 작성되어있는 것을 확인할 수 있었다. 문제에서 "set point"라는 키워드의 조건이 있고, 이를 활용하면 키 값을 해당 파일에서 찾아낼 수 있을 거라 생각하였다.

```
$ find ./src/lib/collision_prevention -type f -print | xargs grep -i "TEST"
./src/lib/collision_prevention/CMakeLists.txt:px4_add_functional_gtest(SRC CollisionPreventionTest.cpp LINKLIBS CollisionPrevention)
./src/lib/collision_prevention/CollisionPrevention.hpp: //Timing functions. Necessary to mock time in the tests
./src/lib/collision_prevention/CollisionPreventionTest.cpp:#include <gtest/gtest.h>
./src/lib/collision_prevention/CollisionPreventionTest.cpp:// to run: make tests TESTFILTER=CollisionPrevention
./src/lib/collision_prevention/CollisionPreventionTest.cpp:class CollisionPreventionTest : public ::testing::Test
./src/lib/collision_prevention/CollisionPreventionTest.cpp:class TestCollisionPrevention : public CollisionPrevention
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestCollisionPrevention() : CollisionPrevention(nullptr) {}
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     void test_addDistanceSensorData(distance_sensor_s &distance_sensor, const matrix::Quatf &attitude)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     void test_addObstacleSensorData(const obstacle_distance_s &obstacle, const matrix::Quatf &attitude)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     void test_adaptSetpointDirection(matrix::Vector2f &setpoint_dir, int &setpoint_index,
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     bool test_enterData(int map_index, float sensor_range, float sensor_reading)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:class TestTimingCollisionPrevention : public TestCollisionPrevention
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestTimingCollisionPrevention() : TestCollisionPrevention() {}
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, instantiation) { CollisionPrevention cp(nullptr); }
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, behaviorOff)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestCollisionPrevention cp;
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, noSensorData)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestCollisionPrevention cp;
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, testBehaviorOnWithObstacleMessage)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestCollisionPrevention cp;
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, testBehaviorOnWithDistanceMessage)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestCollisionPrevention cp;
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, testPurgeOldData)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestTimingCollisionPrevention cp;
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, testNoRangeData)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestTimingCollisionPrevention cp;
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, noBias)
./src/lib/collision_prevention/CollisionPreventionTest.cpp:     TestCollisionPrevention cp;
./src/lib/collision_prevention/CollisionPreventionTest.cpp:TEST_F(CollisionPreventionTest, outsideFOV)
```

Figure 7. 충돌방지와 TEST 문자열 조건을 이용하여 출력된 결과값.

Figure 8. 처럼VSCode 의 문자열 찾기 기능을 이용하여 [key2]값의 키 값을 확인하였다.

```
//define setpoint
matrix::Vector2f setpoint_dir(1, 0);
float sp_angle_body_frame = atan2f(setpoint_dir(1), setpoint_dir(0)) - vehicle_yaw_angle_rad;
float sp_angle_with_offset_deg = matrix::wrap(math::degrees(sp_angle_body_frame) - cp.getObstacleMap().angle_offset,
                                 0.f, 360.f);
int sp_index = floor(sp_angle_with_offset_deg / cp.getObstacleMap().increment); // sp_index=0x8E2615E0
```

```
sp_index=0x8E2615E0
```

Figure 8. [key2]의 키 값

이를 토대로 [key2]의 값은 **0x8E2615E0**이다.

[key3] Search for "laser distance sensor related codes" via the serial part in the driver. There is a security check to prevent heap buffer overflow bug(hint : find PX4 git commit history to see the security patch).

Driver 디렉터리 안에 "laser distance sensor"와 관련된 코드를 검색하라고 하였으므로, 해당 해당 단어와 관련된 디렉터리를 찾은 결과

/PX4-Autopilot-main/src/drivers/distance_sencsor/lightware_laser_serial/ 파일 경로를 찾을 수 있었고, 해당 경로에서 ***lightware_laser_serial.cpp***을 문제 조건대로 바로 찾을 수 있었다.

남은 조건대로 heap, buffer, overflow에 관한 단어가 해당파일에 존재하는지 VSCode를 이용하여 검색하였다. Figure 9, Figure 10, Figure 11은 각각 heap, buffer, overflow를 해당 파일에서 검색했을 때 나온 결과이다.

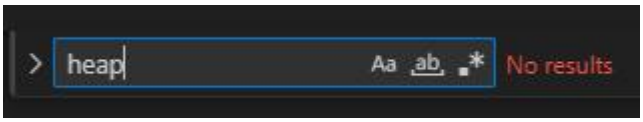


Figure 9. "heap" 검색 결과

```
perf_begin(_sample_perf);

/* clear buffer if last read was too long ago */
int64_t read_elapsed = hrt_elapsed_time(&_last_read);

/* the buffer for read chars is buflen minus null termination */
char readbuf[sizeof(_linebuf)];
unsigned readlen = sizeof(readbuf) - 1;

/* read from the sensor (uart buffer) */
const hrt_abstime timestamp_sample = hrt_absolute_time();
int ret = ::read(_fd, &readbuf[0], readlen);
```

Figure 10. "buffer" 검색 결과

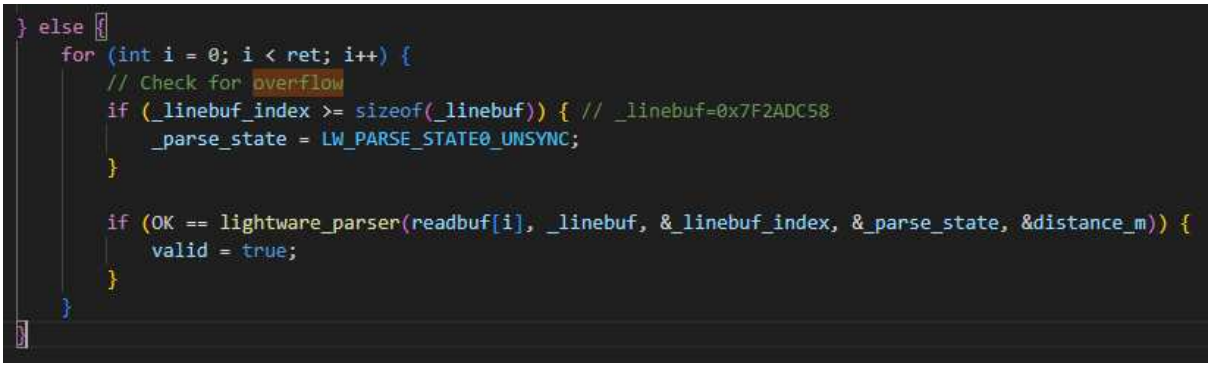


Figure 11. overflow 검색 결과

Figure 11번에 _linebuf=0x7F2ADC58이라는 값이 주석으로 처리되어 숨겨져 있었고, 이를 이용하여 [key3]의 키값은 **0x7F2ADC58**이다.

[key4] In remote control system, there is CRSF protocol. Search for a hexadecimal value for the **COMMAND** offset in the 'type' field of the CRSF packets.

문제 조건을 CRSF 프로토콜 내부에 "COMMAND" 문자열이 존재한다 하였으므로, CRSF를 가진 디렉터리를 검색하였다. 이를 위해 *find . –type d –name "*crsf*"* 를 이용하였다.

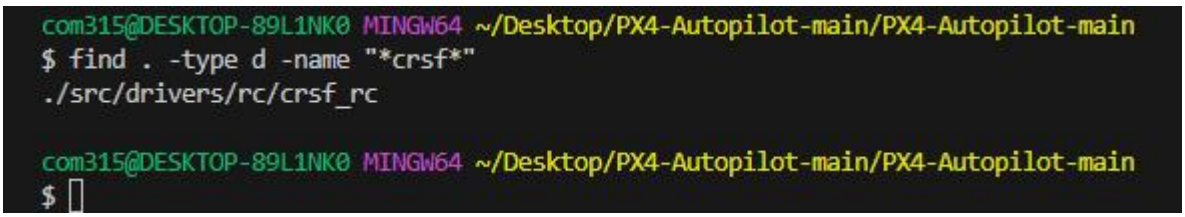


Figure 12. crsf의 이름을 포함하는 디렉터리 검색 결과

Figure 12는 해당 문자열을 이용하여 찾은 결과이며, 단 하나의 파일만 존재하였고, 파일 내부

에서 Figure 13과 같이 COMMAND를 검색한 결과 0x32 하나가 존재하였다.

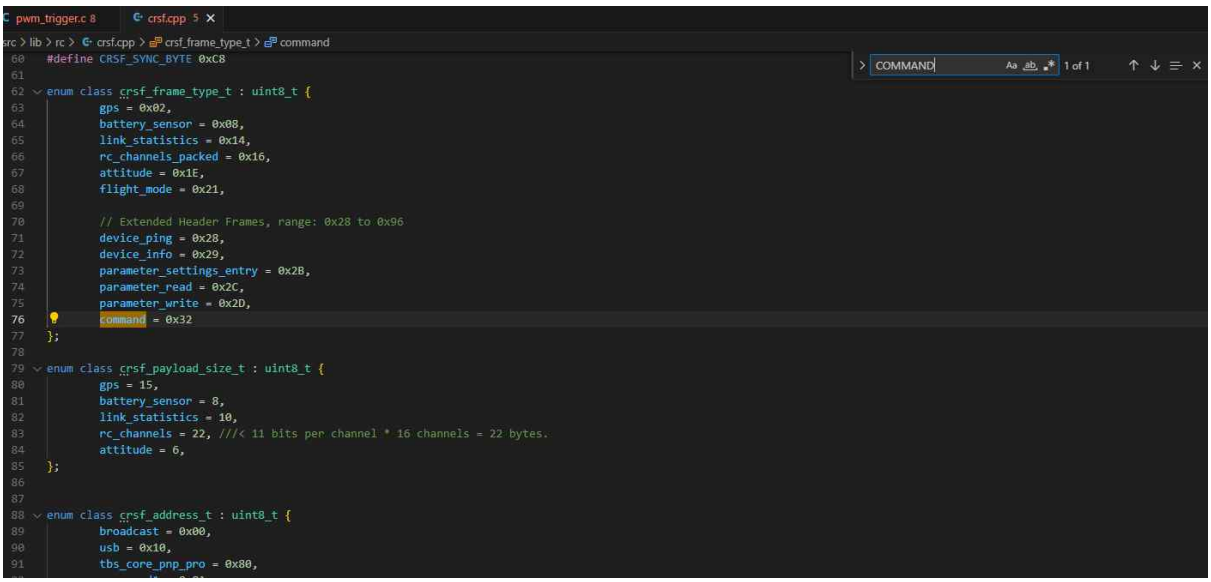


Figure 13. ./src/drivers/rc/crsf_rc 에서 COMMAND 검색 결과

생각보다 많은 파일이 나오지 않아, 모든 디렉터리에서 "CRSF" 문자열이 존재하는지 재 검색하였고, Figure 14와 같은 결과를 확인하였다.

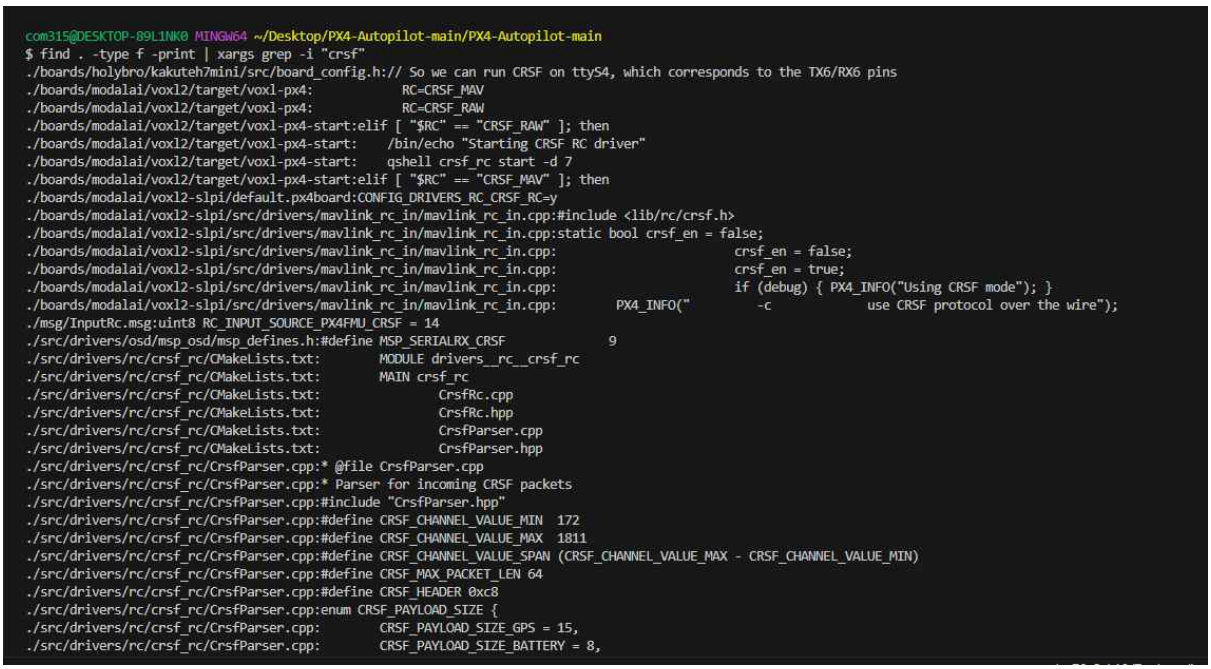


Figure 14 crsf 문자열이 존재하는 파일 검색 결과

CRSF과 관련된 디렉터리를 찾는 중 대부분의 파일의 ***./src/drivers/rc/crsf_rc/CrsfParser.cpp***

에 존재한다는 것을 확인하였고, 해당 파일 내부에서 COMMAND 단어를 figure 15와 같이 검색한 결과 CRSF_PACKET_TYPE_COMMAND =0x32 옆에 주석으로 키값이 숨어있는 것을 발견하였다. 문제 조건대로 CRSF_PACKET 과 관련된 내용이므로 키 값이 확실하다.

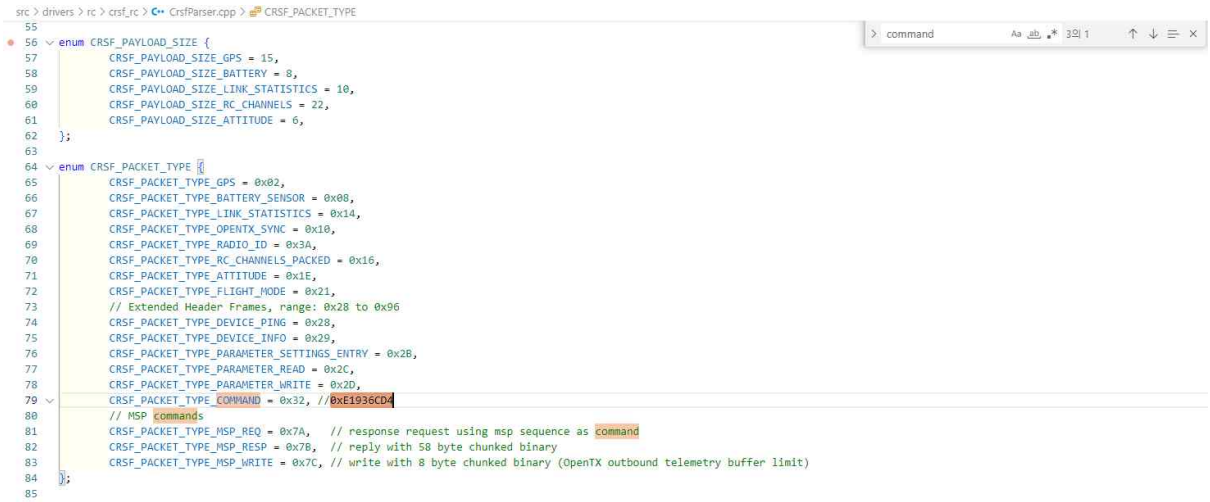


Figure 15 ./src/drivers/rc/crsf_rc/CrsfParser.cpp 에서 command 검색 결과

이를 토대로 [key4]는 **0xE1936CD4**이다.

최종적으로 정답은 [key1][key2][key3][key4]를 조합하여

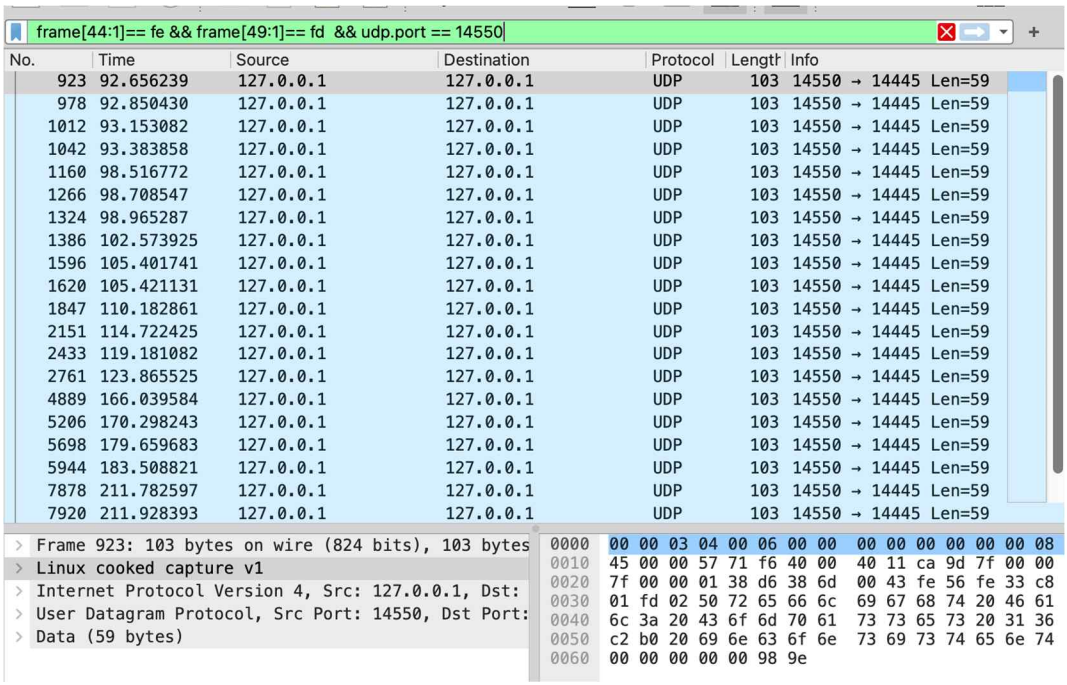
DA8DFC318E2615E07F2ADC58E1936CD4이다.

2. Unraveling the Packet Mystery

1) 문제 분석

MAVLink가 OSI 5계층임을 알고, 해당 문제를 풀기 위하여 이 패킷의 구조를 이해하여야 한다. MAVLink의 패킷을 WireShark를 이용하여 분석하고, 이를 토대로 키 값을 추출한다.

2) 문제 풀이



네트워크 패킷 분석 도구를 사용하여 패킷 분석을 시도한다. MAVLink 프로토콜은 기본적으로 UDP(User Datagram Protocol)를 사용하며, 일반적으로 14550 포트를 사용하여 통신하므로 udp.port == 14550라는 명령어를 사용하면 MAVlink 프로토콜을 사용하는 패킷들을 얻을 수 있다. MAVlink는 0번 째 byte에 Start marker인 fe, 1번 째 byte에 Payload length를 나타내는 값, 2번 째 byte에 SEQ, 3번 째 byte에 SYS ID, 4번 째 byte에 COMP ID, 5번 째 byte에 MSG ID, 6번 째 byte부터 payload의 데이터로 이루어져 있다.

fe라는 데이터로 시작해야하므로 frame[44:1] == fe 라는 명령어를 사용한다.

5번 째 byte인 MSG ID의 값이 10진수로 253(16진수로 fd)인 패킷이 조작된 패킷이므로 시작비트+5인 49번 째 byte가 fd인 패킷을 찾기 위해 frame[49:1] == fd 명령어를 사용한다.

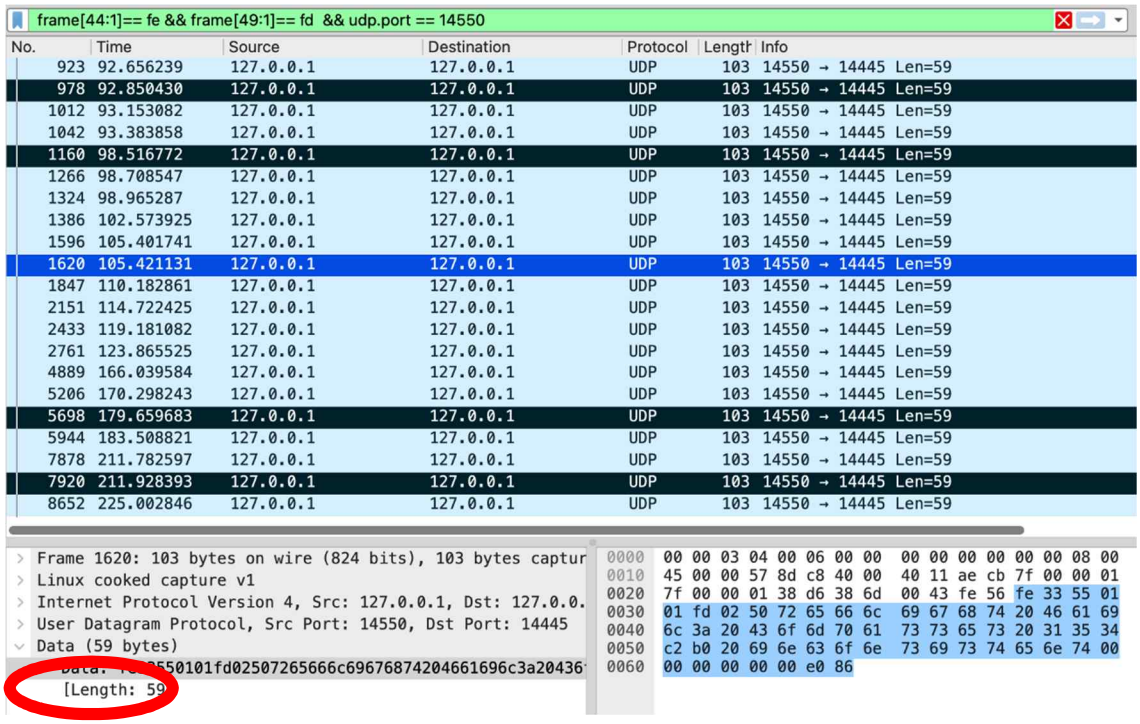


Figure 16 조작되지 않은 패킷

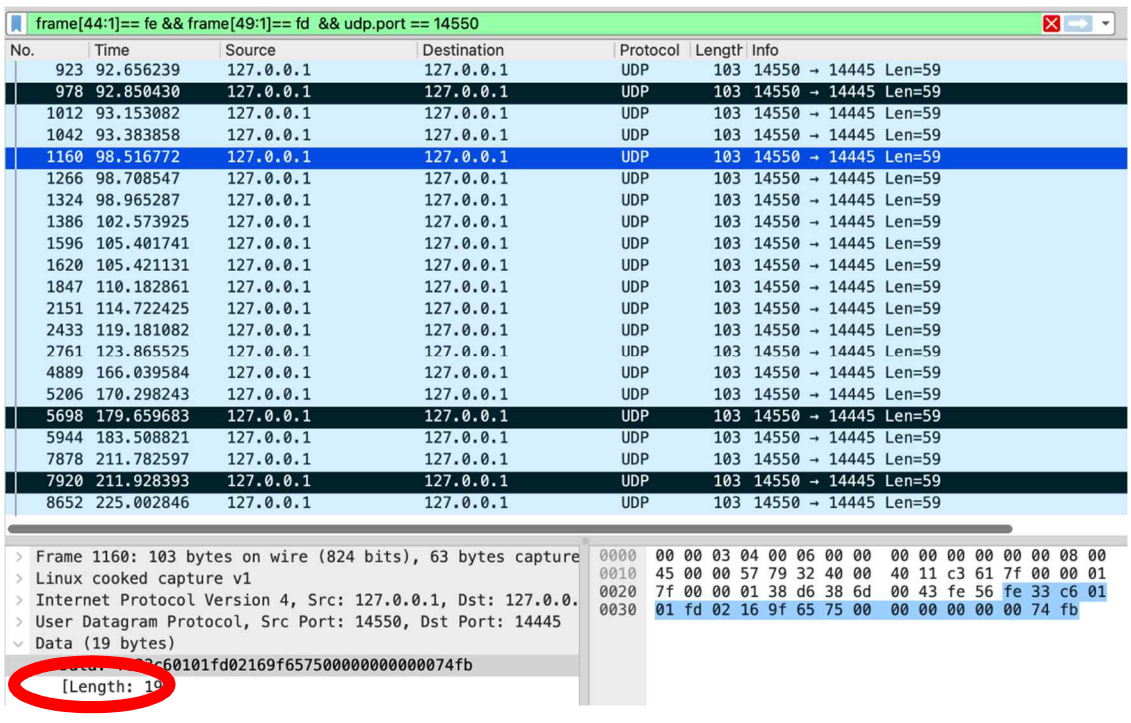


Figure 17 조작된 패킷

조작되지 않은 패킷은 length가 59이고, 조작된 패킷을 살펴보면 페이로드의 길이 차이가 확연하다. 조작된 패킷들을 mark한 결과 4개의 패킷이 조작됐다는 것을 알 수 있었다.

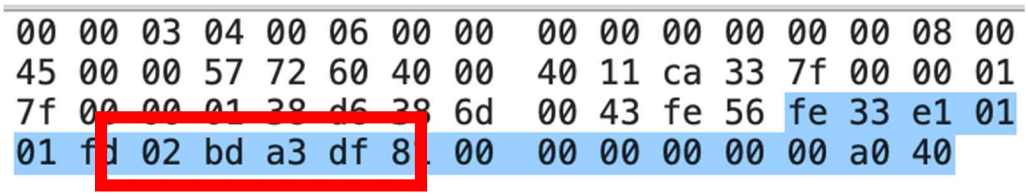


Figure 18 No.978

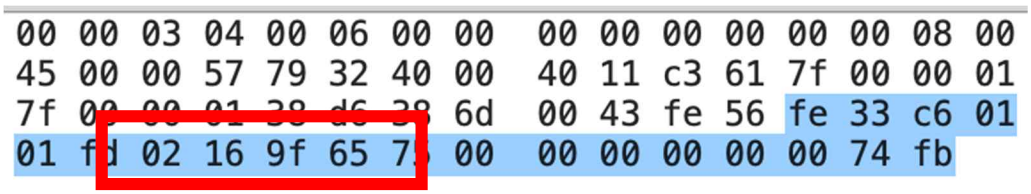


Figure 19 No.1160

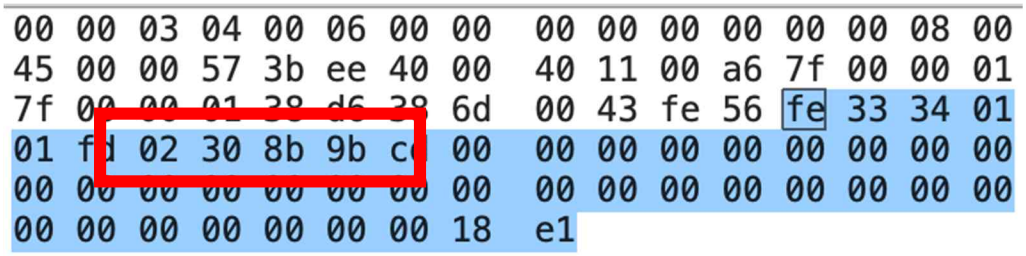


Figure 20 No.5698

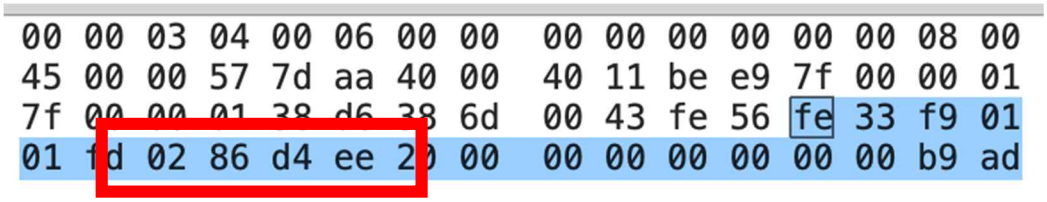


Figure 21 No.7920

위 4개의 사진은 조작된 패킷의 내용이다.

6번째 byte부터 전송하려고 하는 조작된 데이터이므로 payload의 값을 찾아주면 된다. Flag가 16byte라는 점을 고려하여 앞에 중복되는 02를 제외하고 나머지 4byte씩 순서대로 이어 붙여 입력하면 답을 찾을 수 있었다.