206 - Analyze Drone File

Team Information

Team Name : LuckyVicky

Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee

Email Address : dfc_luckyvicky@googlegroups.com

Instructions

Description A drone has been discovered flying around a specific area. For an unknown reason, the drone fell to the ground. Upon analyzing the drone, it was found to be storing encrypted files. Unable to determine the content of these files, it was identified that there is encryption code within a specific file. Analyze this file to decrypt the encrypted files and extract the FLAG value contained within, and then submit it.

Target	Hash (MD5)
arducopter	12e2ad65255e2813434d53356a877342
output.enc	92217d5773ef7f26869e622c85575994

Questions

- Decrypt the output.enc file. (200 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).
-

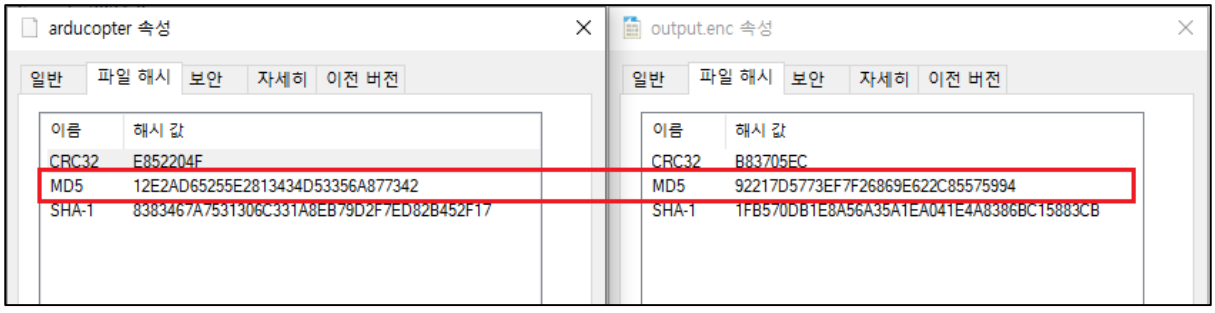
Tools used:

Name:	IDA PRO	Publisher:	hex-rays
Version:	8.2.230124		
URL:	https://hex-rays.com/		

Name:	HashTab	Publisher:	Implbits Software
Version:	6.0.0		
URL:	https://implbits.com		

Name:	Sublime Text 4	Publisher:	Sublime Text
Version:	Build 4180		
URL:	https://www.sublimetext.com/		

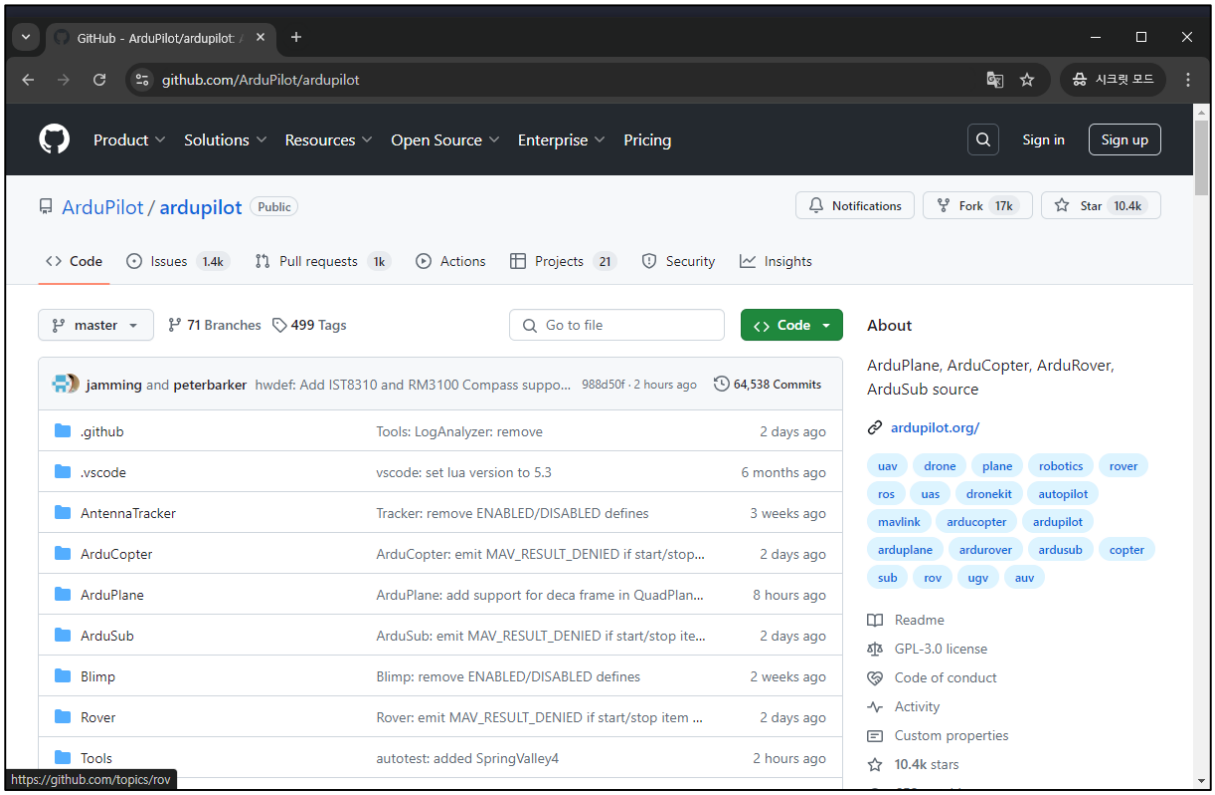
Step-by-step methodology:



[그림 1] md5 해시 값 확인

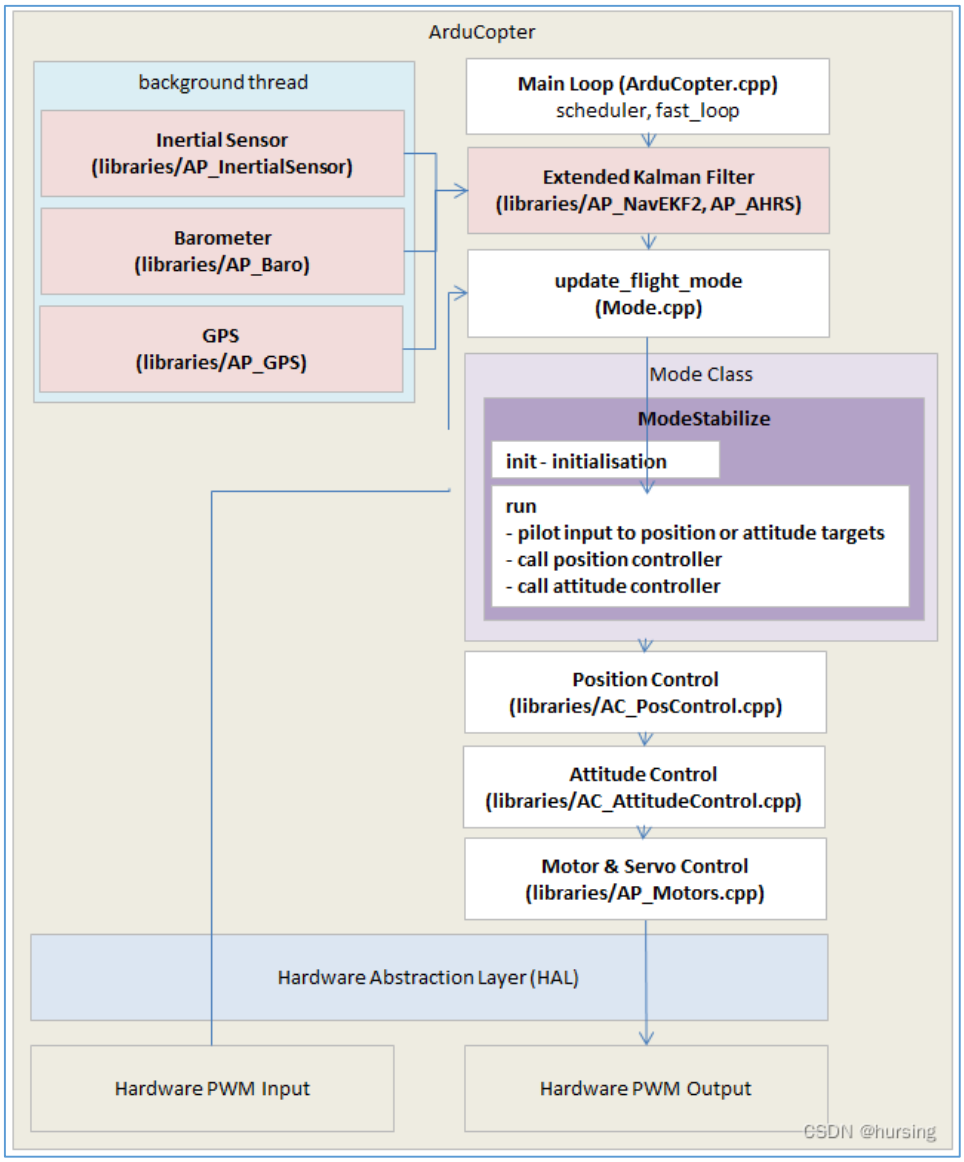
주어진 파일들의 MD5 해시 값이 일치하는 것을 확인하였습니다.

- Decrypt the output.enc file. (200 points)



[그림 2] 펌웨어 관련 오픈소스

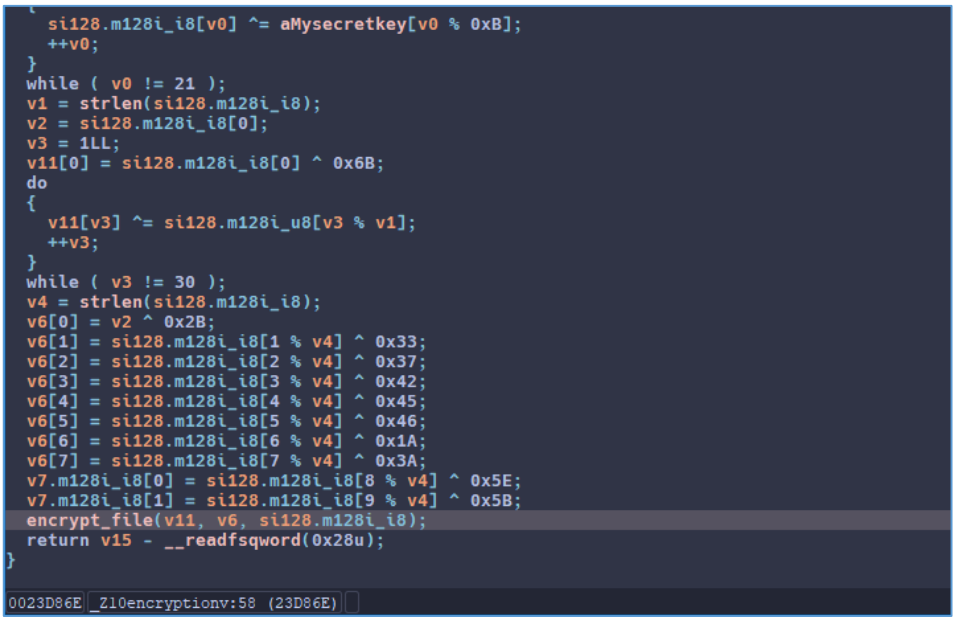
ArduCopter는 ArduPilot 프로젝트의 하위 프로젝트로써 멀티로터 드론을 위한 자동 조종 소프트웨어입니다. ArduCopter는 쿼드콥터, 헥사콥터, 옥토콥터 등 여러 종류의 멀티로터 비행체의 제어를 가능하게 합니다.



[그림 3] ArduCopter의 구조

ArduCopter의 구조는 위치 및 자세 제어, 비행 모드, 주요 제어 루프, 센서 데이터 처리, 그리고 하드웨어 추상화 계층(HAL)로 구성되어 있습니다. 이러한 구성 요소들은 드론의 비행을 제어하고 안정화하는 핵심 역할을 하며 이를 바탕으로 펌웨어를 분석했습니다.

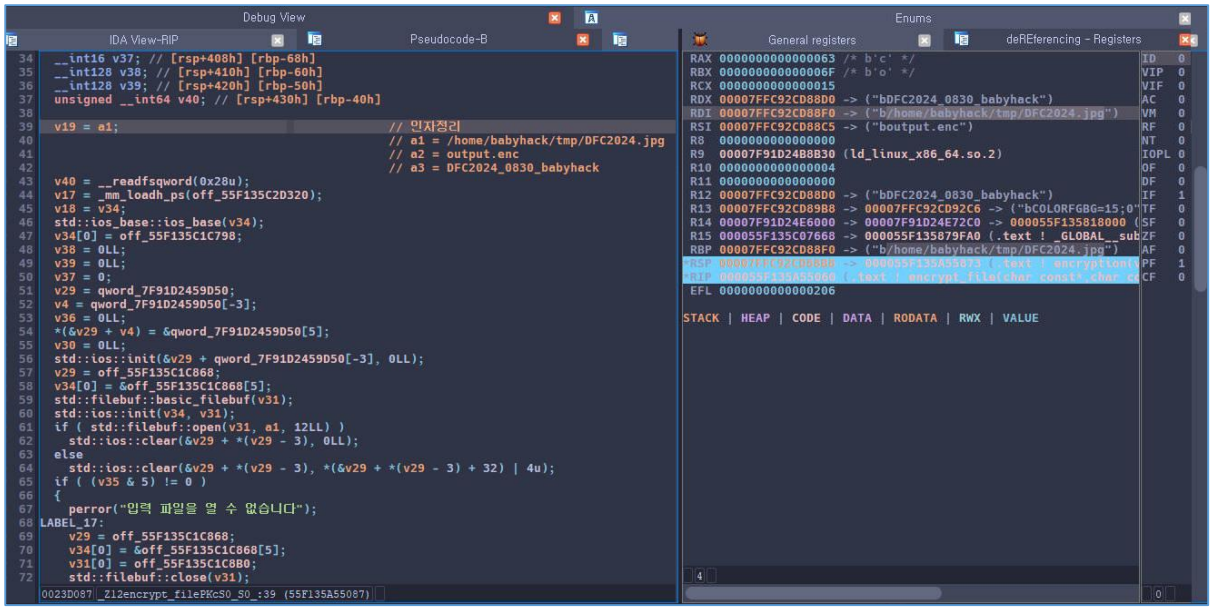
* 사진 출처: https://blog.csdn.net/hursing/article/details/128665474



```
    si128.m128i_i8[v0] ^= aMysecretkey[v0 % 0xB];
    ++v0;
  }
  while ( v0 != 21 );
  v1 = strlen(si128.m128i_i8);
  v2 = si128.m128i_i8[0];
  v3 = 1LL;
  v11[0] = si128.m128i_i8[0] ^ 0x6B;
  do
  {
    v11[v3] ^= si128.m128i_u8[v3 % v1];
    ++v3;
  }
  while ( v3 != 30 );
  v4 = strlen(si128.m128i_i8);
  v6[0] = v2 ^ 0x2B;
  v6[1] = si128.m128i_i8[1 % v4] ^ 0x33;
  v6[2] = si128.m128i_i8[2 % v4] ^ 0x37;
  v6[3] = si128.m128i_i8[3 % v4] ^ 0x42;
  v6[4] = si128.m128i_i8[4 % v4] ^ 0x45;
  v6[5] = si128.m128i_i8[5 % v4] ^ 0x46;
  v6[6] = si128.m128i_i8[6 % v4] ^ 0x1A;
  v6[7] = si128.m128i_i8[7 % v4] ^ 0x3A;
  v7.m128i_i8[0] = si128.m128i_i8[8 % v4] ^ 0x5E;
  v7.m128i_i8[1] = si128.m128i_i8[9 % v4] ^ 0x5B;
  encrypt_file(v11, v6, si128.m128i_i8);
  return v15 - __readfsqword(0x28u);
}
0023D86E _Z10encryptionv:58 (23D86E)
```

[그림 4] 암호화 관련 함수

펌웨어 구조를 분석하는 과정에서 관련 함수들을 제외하고 암호 함수를 찾던 중 encrypt() 함수를 발견했습니다. 이 함수의 내부 로직을 조사한 결과 encrypt_file() 함수가 핵심 역할을 한다는 것을 파악하였습니다.

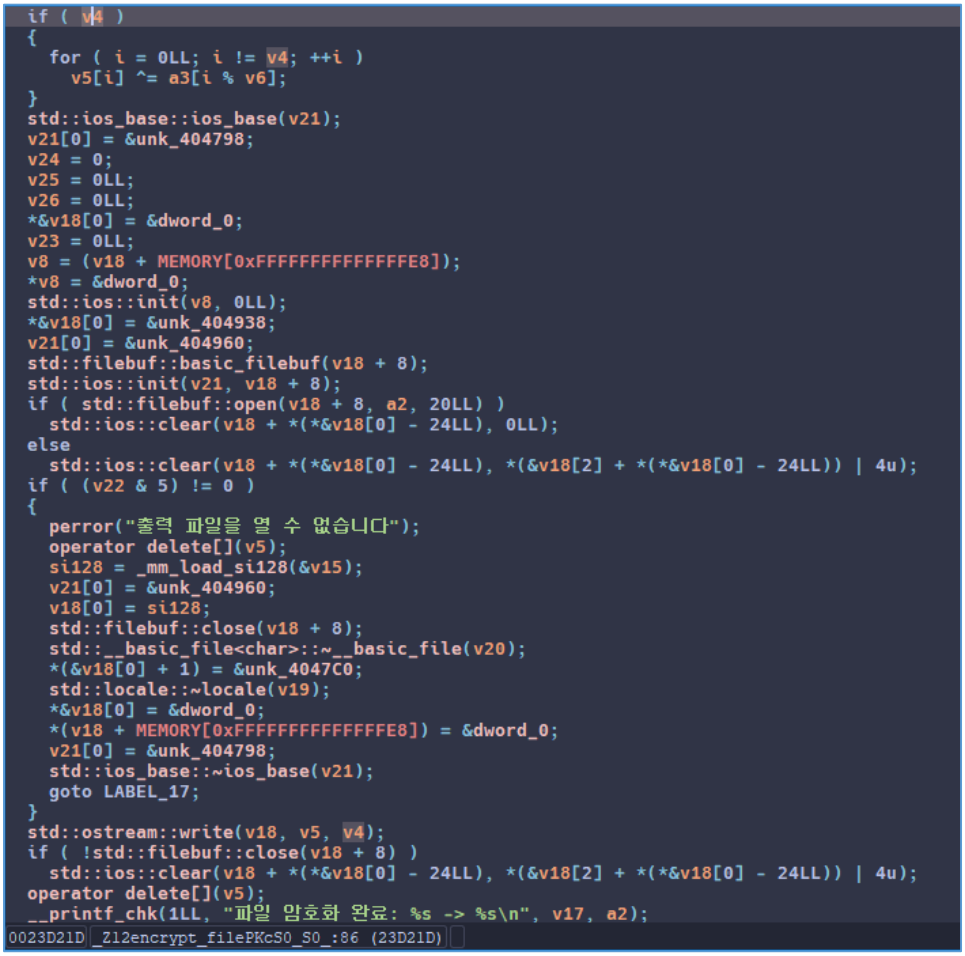


[그림 5] 암호화 함수 디버깅

Args	Data
Input file (a1)	/home/babyhack/tmp/DFC2024.jpg
Output file (a2)	output.enc
Key (a3)	DFC2024_0830_babyhack

[표 1] encrypt_file 함수 인자 값

디버깅을 통해 함수에서 활용되는 인자 값들을 파악하였습니다.



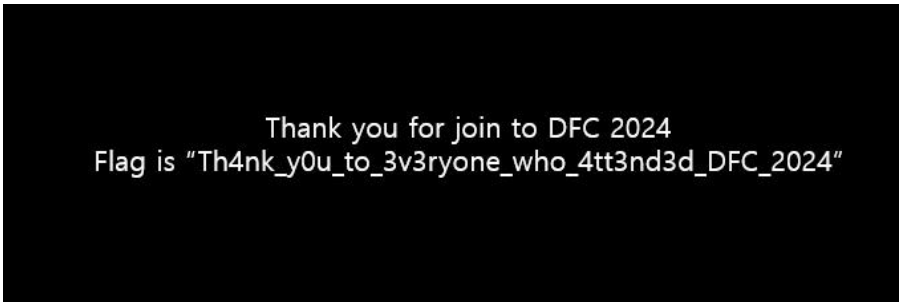
```
if ( v4 )
{
  for ( i = 0LL; i != v4; ++i )
    v5[i] ^= a3[i % v6];
}
std::ios_base::ios_base(v21);
v21[0] = &unk_404798;
v24 = 0;
v25 = 0LL;
v26 = 0LL;
*&v18[0] = &dword_0;
v23 = 0LL;
v8 = (v18 + MEMORY[0xFFFFFFFFFFFFFFE8]);
*v8 = &dword_0;
std::ios::init(v8, 0LL);
*&v18[0] = &unk_404938;
v21[0] = &unk_404960;
std::filebuf::basic_filebuf(v18 + 8);
std::ios::init(v21, v18 + 8);
if ( std::filebuf::open(v18 + 8, a2, 20LL) )
  std::ios::clear(v18 + *(*&v18[0] - 24LL), 0LL);
else
  std::ios::clear(v18 + *(*&v18[0] - 24LL), *(&v18[2] + *(*&v18[0] - 24LL)) | 4u);
if ( (v22 & 5) != 0 )
{
  perror("출력 파일을 열 수 없습니다");
  operator delete[](v5);
  si128 = _mm_load_si128(&v15);
  v21[0] = &unk_404960;
  v18[0] = si128;
  std::filebuf::close(v18 + 8);
  std::__basic_file<char>::~__basic_file(v20);
  *(&v18[0] + 1) = &unk_4047C0;
  std::locale::~locale(v19);
  *&v18[0] = &dword_0;
  *(v18 + MEMORY[0xFFFFFFFFFFFFFFE8]) = &dword_0;
  v21[0] = &unk_404798;
  std::ios_base::~ios_base(v21);
  goto LABEL_17;
}
std::ostream::write(v18, v5, v4);
if ( !std::filebuf::close(v18 + 8) )
  std::ios::clear(v18 + *(*&v18[0] - 24LL), *(&v18[2] + *(*&v18[0] - 24LL)) | 4u);
operator delete[](v5);
__printf_chk(1LL, "파일 암호화 완료: %s -> %s\n", v17, a2);
0023D21D _Z12encrypt_filePKcS0_S0_:86 (23D21D)
```

[그림 6] 암호화 과정

인자로 전달 받은 KEY를 활용해 XOR하여 데이터를 output.enc로 저장하게 됩니다. 따라서 아래와 같은 소스코드를 통해 원본 데이터를 획득 할 수 있습니다.

```cpp
#include <iostream>
#include <fstream>
#include <vector>


void decrypt_file(const std::string& input_file, const std::string& output_file,
    const std::string& key) {
    std::ifstream infile(input_file, std::ios::binary | std::ios::ate);

    std::streamsize size = infile.tellg();
    infile.seekg(0, std::ios::beg);

    std::vector<unsigned char> buffer(size);
    if (!infile.read(reinterpret_cast<char*>(buffer.data()), size)) {
        std::cerr << "Error reading input file!" << std::endl;
        return;
    }

    infile.close();

    size_t key_len = key.size();
    std::vector<unsigned char> decrypted_data(buffer.size());

    // 암호화 때와 동일한 XOR 연산을 역으로 수행
    for (size_t i = 0; i < buffer.size(); ++i) {
        decrypted_data[i] = buffer[i] ^ key[i % key_len];
    }

    std::ofstream outfile(output_file, std::ios::binary);
    if (!outfile) {
        std::cerr << "Cannot open output file!" << std::endl;
        return;
    }

    outfile.write(reinterpret_cast<const char*>(decrypted_data.data()),
        decrypted_data.size());
    outfile.close();

    std::cout << "Decryption complete: " << input_file << " -> " << output_file <<
         std::endl;
}

void decryption() {
    // encrypt_file() 함수에 활용된 3개의 인자를 그대로 활용
    std::string key = "DFC2024_0830_babyhack";
    std::string input_file = "output.enc";
    std::string output_file = "DFC2024.jpg";
    decrypt_file(input_file, output_file, key);
}

```

[그림 7] 복호화 소스코드



[그림 8] 원본 데이터

소스코드를 통해 output.enc를 원본 데이터 JPG 파일로 복호화 하였습니다.

답: Th4nk_y0u_to_3v3ryone_who_4tt3nd3d_DFC_2024