|  |
| --- |
| **HWP 악성코드 대응 프로그램**  **개발 보고서** |

|  |  |
| --- | --- |
| **수업명** | 악성코드 분석 및 침해사고 대응 시스템 개발 |
| **작성자** | SAYNOT |
| **작성일** | 2020년 02월 13일 |

목 차

[1. 준비 3](#_Toc32519503)

[1.1 사전 조사 3](#_Toc32519504)

[1.2툴 설계 3](#_Toc32519505)

[2. 진행 과정 5](#_Toc32519506)

[2.1 바이트 평균 5](#_Toc32519507)

[2.2 코드 설명 7](#_Toc32519508)

[2.3 사용법 11](#_Toc32519509)

[3. 결과 및 보완점 12](#_Toc32519510)

[3.1 결과 12](#_Toc32519511)

[3.2 보완점 및 아이디어 14](#_Toc32519512)

# 1. 준비

## 1.1 사전 조사

krCERT에서 발표한 한글 취약점을 통해 유포되는 악성코드 동향 및 분석 보고서에 따르면 HWP 악성코드 삽입 시 많이 사용되는 기술이 Heap Spray 기법이란 것을 알 수 있다. 이외에도 BinData 안에 PostStript(ps)로 코딩한 악성코드를 실행하고 원하는 파일을 가져오는 방법들 등 다양한 스크립트를 삽입하는 방법이 존재한다. 하지만 본 대응 프로그램에선 많은 공격을 막을 수 있는 Heap Spray 공격에 초점을 맞추어 방어를 진행하고자 하였다.

한글 문서 5.0의 구조를 살펴보면 다양한 섹션들이 존재한다. FileHeader, DocInfo, BodyText, HwpSummaryInformation, BinData, PrvText, PrvImage, Docoptions, Scripts, XML Template, DocHistory 정도로 크게 나뉜다. 여기서 참고해야 할 정보가 여럿 존재하는데 일단 이 중 레코드 구조를 띄는 형태는 DocInfo, BodyText, DocHistory가 있으며 향후 과정에서 중요한 zlib로 압축된 영역은 DocInfo, BodyText, BinData 등이 있다. 또, krCERT에서 발표한 보고서를 포함한 대다수의 악성코드에서 BodyText 부분과 BinData 부분에서 집중적인 쉘 코드, 스크립트 삽입이 일어난 것을 알 수 있어서 이부분을 집중적으로 보기로 하였다. 하지만 다른 영역에서도 충분히 코드 삽입이 일어날 가능성이 있기에 본 프로그램에선 모든 영역을 검사 할 것이다.

본 대응 프로그램에선 위에서 설명하였듯이 Heap spray를 통한 공격을 중심적으로 대응해 나갈 것이다. 진행을 위해선 NOP Sled의 특징에 대한 이해가 필요하다. NOP sled를 사용해 특정 영역에 할당되었을 때 NOP 행위를 연속적으로 하여 악성 행위자가 의도한 쉘 코드를 실행시키는 방식이다. 보통 NOP은 0x90으로 알고있지만 실제로 NOP 행위를 할 수 있는 다양한 코드 들이 존재한다. 그래서 단지 고정 값을 통해 대응하긴 힘들고 본 프로그램에선 바이트 빈도를 이용하여 Nop Sled를 탐지할 생각이다. 그렇기 위해선 기준이 필요할 것이다. 그렇기 때문에 약 2000개의 정상 HWP 문서(대부분 통계청사이트)를 수집하였고 2000개의 각각 영역에 대한 바이트 평균을 계산한 뒤 이에 크게 벗어나면 악성이라고 진단할 계획이다. NOP Sled를 진단한 뒤 해당 영역을 집중적으로 분석하여 쉘 코드를 찾고 이후 악성행위의 키가 되는 xor연산 행위를 차단하여 해당 악성코드 실행을 차단할 생각이다.

## 1.2툴 설계

. 일단 툴을 설계하기 전 기본적인 프로그램 동작 순서도를 작성하였다.

1. OLE 문서 헤더확인 : 0xD0 0xCF 0x11 0xE0 0xA1 0xB1 0x1A 0xE1

2. Root Entry를 검색하여 OLE 문서 구조를 띄는지 확인 : 섹터크기 \* 첫번째 섹션 ID 위치

3. FileHeader 스트림 시작 오프셋에서 HWP Document File 문자열 확인

4. 압축 여부 확인 후 Decompress 진행하여 분석 시작

5. 바이트 평균에 크게 벗어난 섹션 확인

6. 확인된 세션에서 F0 FF (Type부분에 size 확인 방식과 동일)를 탐색 : BinData에선 해당항목 제외

7. 뒤 4바이트 사이즈를 읽고 점프하여 해당 영역안에 있는 쉘 코드 함수 형태 확인

8. 확인 시 해당 주소부터 쉘 코드 무력화

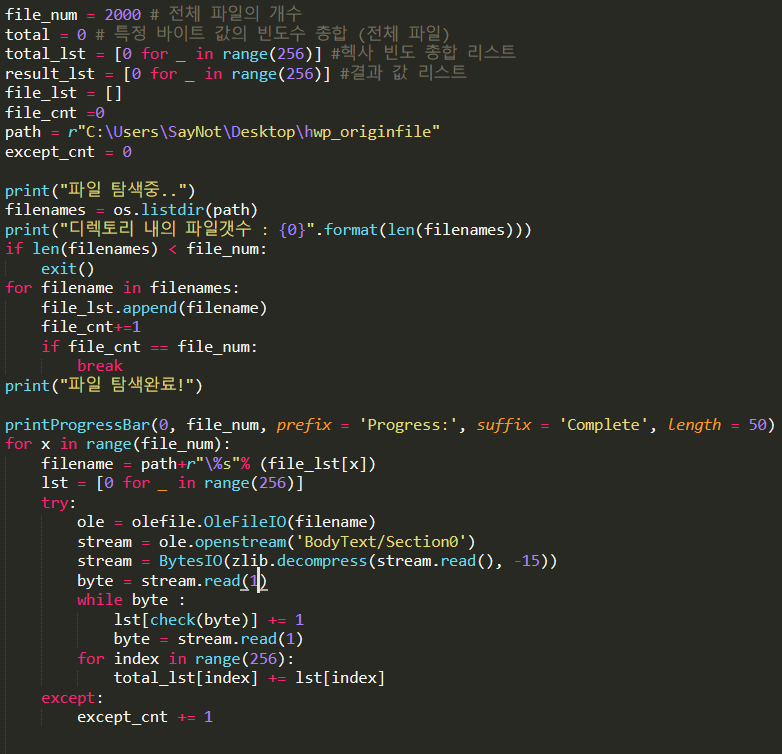
9. 만약 7번에서 함수가 없을 시 NOP Sled 00으로 덮기

이를 위하여 약 3가지 정도의 파일을 만들면 된다고 생각하였다. 하나는 행위를 탐색하고 막는 메인 프로그램 파일과 비교를 할 때 사용되는 파일, 그리고 바이트 평균을 계산하기 위한 프로그램까지 총 3개 파일을 코딩하기로 하였다.

# 2. 진행 과정

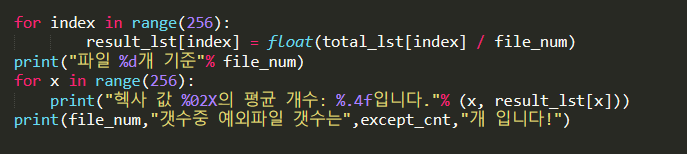
## 2.1 바이트 평균

설계 단계에서 설명하였던 바이트 평균을 계산하기 위한 툴을 개발하였다. 일단 한 폴더에 2000개의 파일을 넣어두고 하나씩 읽으며 OLE파서를 이용하여 바이트 평균을 계산하기 위한 영역 바이트를 추출할 것이다. 이후 00-FF까지 각각에 대한 개수를 더하고 파일 개수로 나누어 바이트 평균을 계산한다.



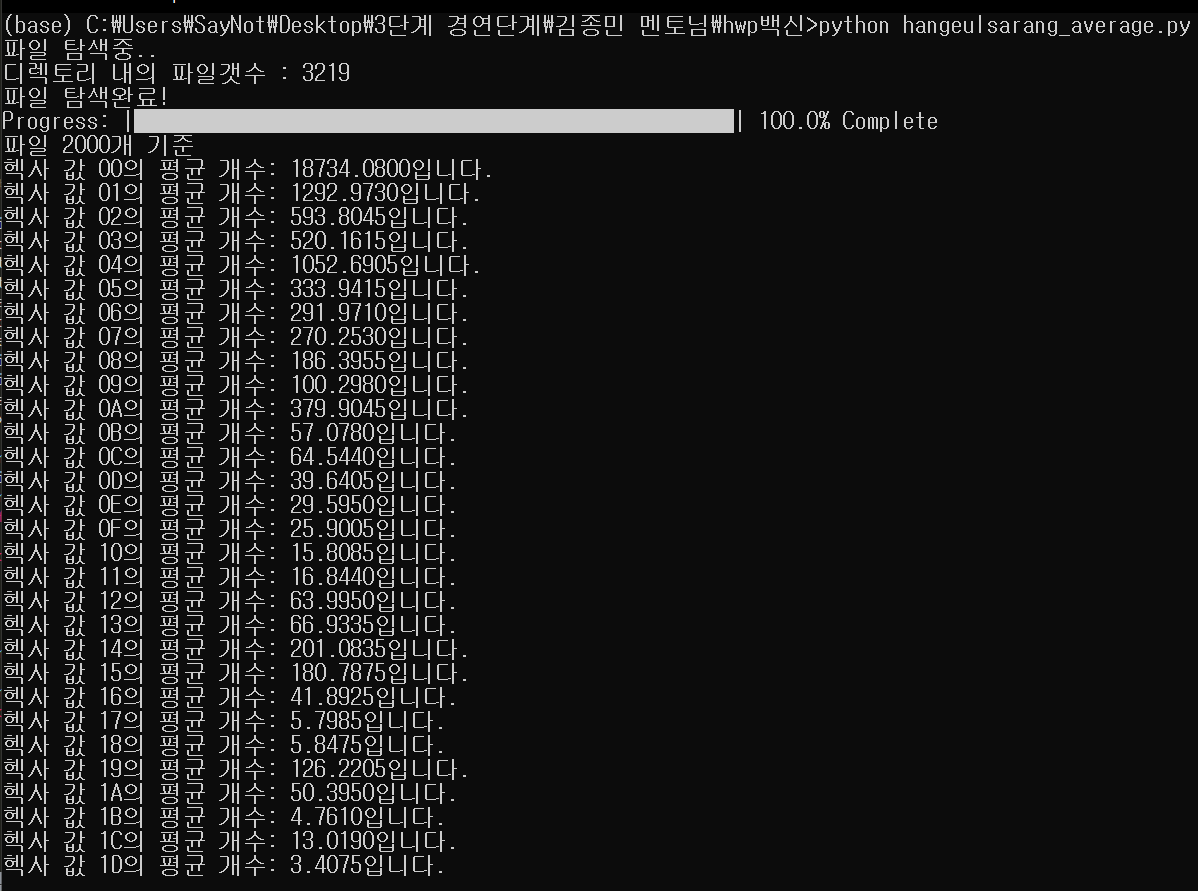
<사진1. hangeulsarang\_average.py>

일단 통계를 내기 위한 2000개 파일 수를 지정해주고 경로를 지정해준다. 이후 폴더에 있는 파일들을 하나씩 읽어온다. 이후 olefile 모듈을 이용하여 원하는 섹션의 스트림을 불러온다. 그리고 zlib으로 압축 되어있는 섹션일 때 zlib 라이브러리의 decompress를 이용하여 압축을 풀고 바이트를 읽는다. 그리고 각 바이트에 맞는 개수를 하나씩 카운트한다. 정확한 통계 결과를 위해 try, except을 이용하여 예외처리를 하고 처리가 안된 파일 역시 카운트한다.



<사진2. hangeulsarang\_average.py>

이후 모든 바이트 총합 개수에서 파일 개수로 나누어 평균을 계산하여 result\_lst라는 리스트에 저장한다. 그리고 2000개 한글 파일에 대한 각 헥사 값에 대한 평균 개수를 알려준다. 실행결과 화면은 다음과 같다.



<사진3. python hangeulsarang\_average.py>(사진아래 1E-FF까지 존재)

메인 코드를 설명하기 전에 간단하게 바이트평균 결과에 대한 세가지 특징을 서술할 것이다. 2000개의 정상 HWP 파일을 기준으로 Fileheader, DocInfo, Section0, Section1, BinData에 대한 결과들을 살펴보았다.

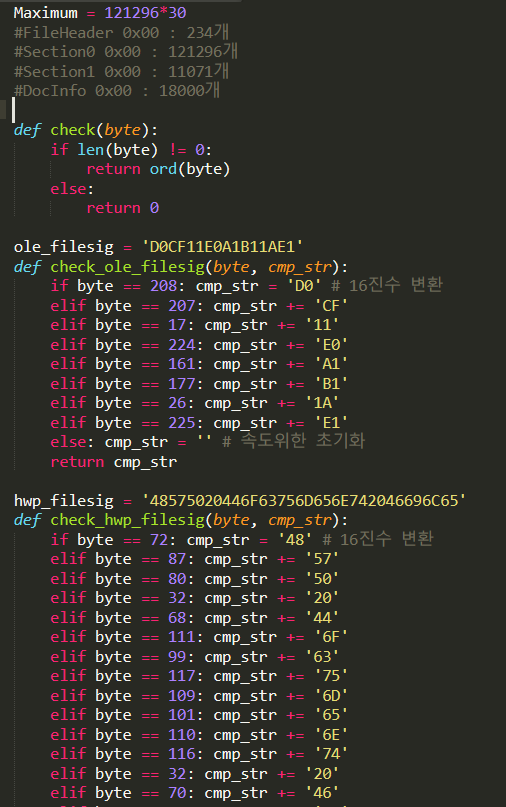
1. 0x00이 대부분 많은 개수를 차지하고 있다. : 바이트의 개수가 텍스트는 고르게 분포되는게 일반적이나 대부분의 섹션에서 0x00이 많은 개수를 차지하는 것을 볼 수 있다. 하지만 0x00을 제외하고 대체로 3-500개정도의 적은 바이트 개수들을 확인할 수 있다.

2. BinData에 바이트 평균 개수가 비교적 높으며 그중 0xFF 개수가 매우 많다. : BinData 부분엔 바이너리 데이터가 들어가며 bmp와 같은 사진 파일이 존재해 평균 바이트 개수가 높고 FF의 경우 100만-1000만개를 넘어가는 경우도 존재한다. 이경우 NOP Sled와 같이 평균 바이트 값을 크게 벗어나기 때문에 피처로 잡을 경우 정상인데 악성으로 오탐 할 수도 있으니 조심해야 한다.

3. 정상파일의 경우 상위 1, 2번을 제외하면 바이트 개수가 기하학적으로 크지 않다.

## 2.2 코드 설명

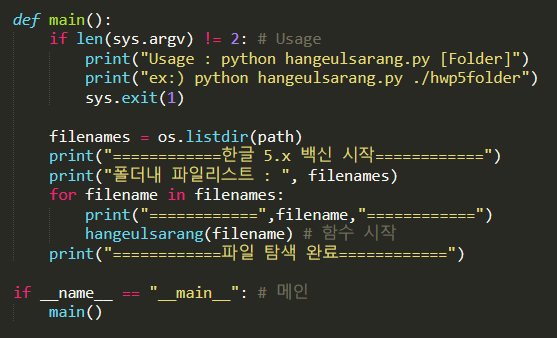
메인 코드는 hangeulsarang\_cmp.py와 hangeulsarang.py 두가지 파일로 나뉜다. 우선 hangeulsarang\_cmp.py 소스를 살펴볼 것이다.



<사진 4. hangeulsarang\_cmp.py>

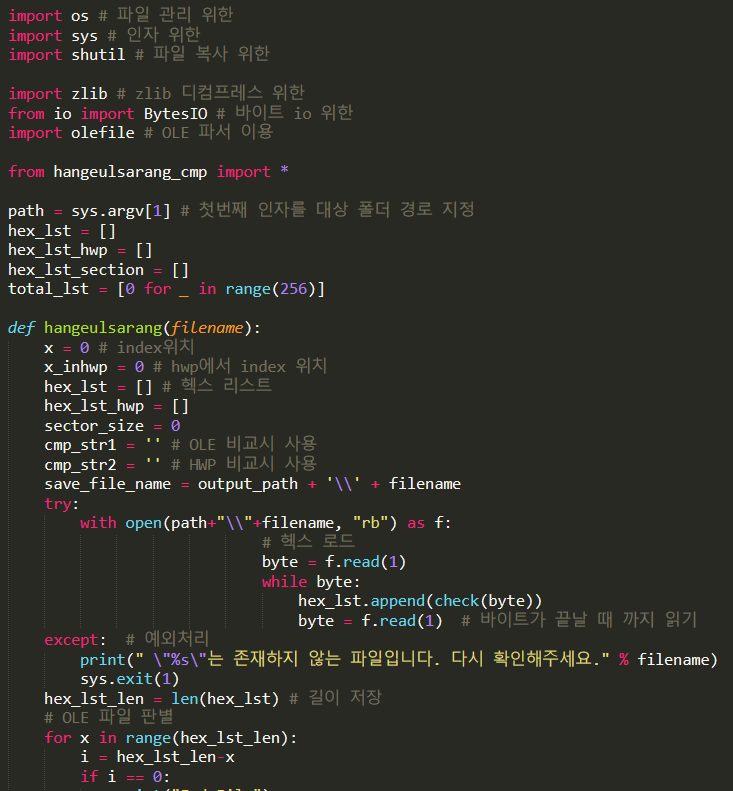
hangeulsarang\_cmp.py파일은 hangeulsarang.py에서 비교를 하기위해 사용할 변수들과 함수를 정의 해 놓은 파일이다. 바이트평균 결과를 토대로 평균보다 많이 떨어진 기준을 정하기 위한 Maximum이란 변수와 바이트 체크를 위한 check 함수가 존재한다. 그리고 ole 파일 시그니쳐 8바이트를 확인하기 위한 check\_ole\_filesig함수, FileHeader 섹션에서 hwp파일을 확인하기 위한 check\_hwp\_filesig 함수가 있다. 이후 heap spray 기술 사용시 사용되는 F0 FF를 탐색하는 함수와 쉘 코드 함수 시작부분 90 57 56 52 53 55 51을 역으로 탐색하기 위한 함수를 정의해 두었다.

여기서 주의해야할 점은 heap spray 탐색 시 BinData에서 오탐을 일으킬 확률이 높으며 Maximum변수를 넘는 값이 존재할 수 있으므로 Maximum 기준 선정 시 신중해야 한다. 그리고 쉘 코드 패턴으로 정의한 해당 바이트가 존재하지 않는 경우도 있을 것이다.



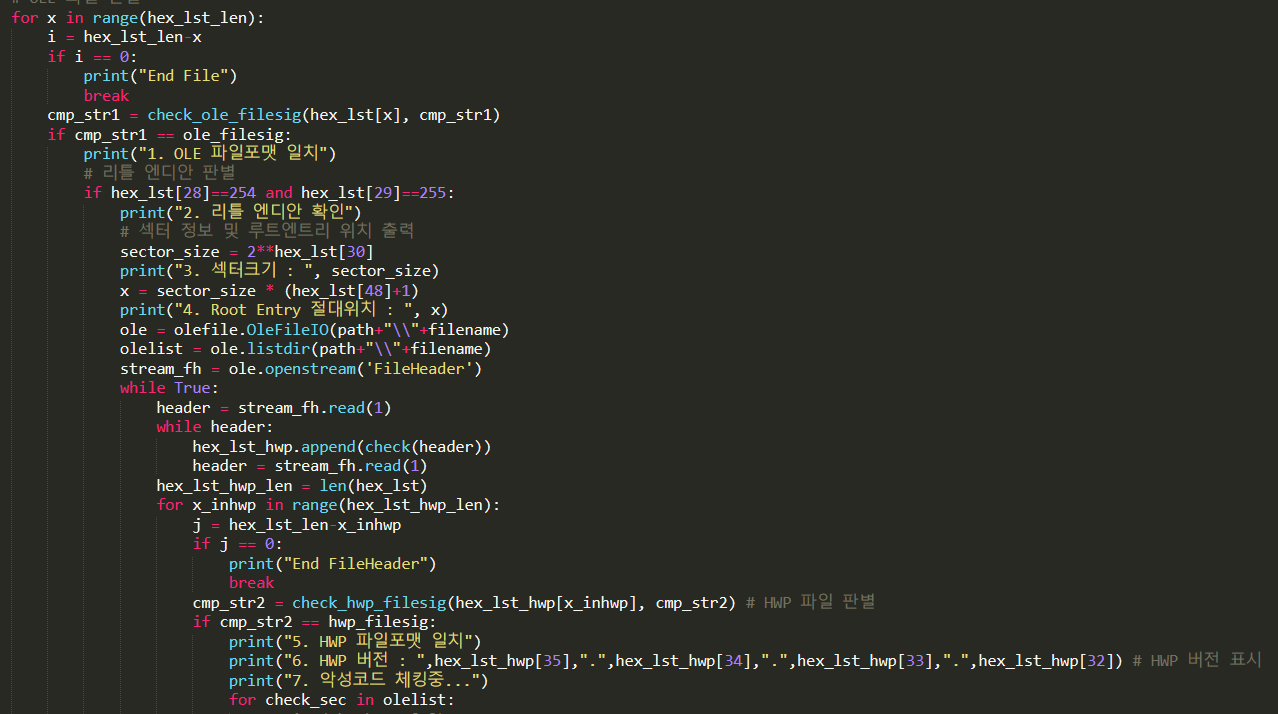
<사진 5. hangeulsarang.py>

다음은 main이 존재하는 hangeulsarang.py 파일이다. 인자를 통해 진단을 할 폴더 경로를 입력 받는다. 이후 폴더에 있는 파일들을 하나하나 읽어 hangeulsarang 이라는 함수에 집어넣는다.



<사진 6. hangeulsarang.py>

최상단엔 main과 hangeulsarang함수를 실행시키기 위한 모듈 라이브러리를 import한다. 그리고 헥사 리스트를 저장할 리스트들을 변수로 선언해 둔다. hangeulsrang 함수에선 파일을 입력 받아 바이트 단위로 파일을 읽는다.



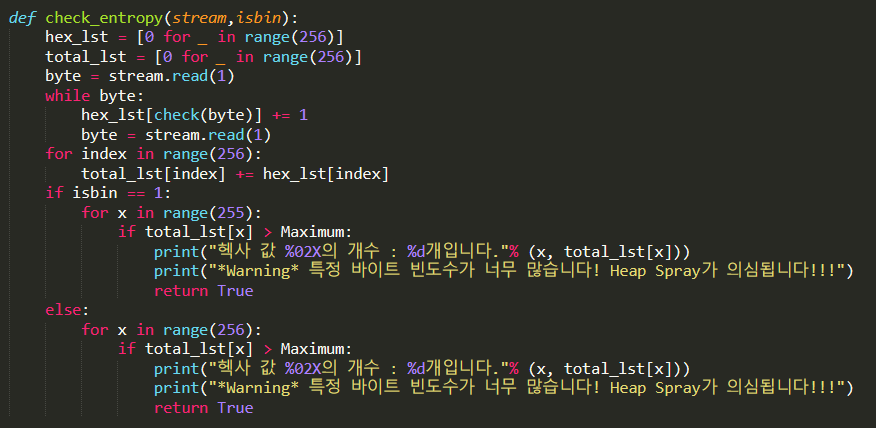
<사진 7. hangeulsarang.py>

파일을 읽고 위에서 1.2 툴설계에서 정의해둔 순서대로 동작을 시작한다. 일단 처음 8바이트를 읽어 시그니쳐 확인를 통해 ole 파일을 검증한다. 이후 리틀 엔디안 방식을 확인후 섹터 크기 정보를 출력한다. 이후 Root Entry 절대위치를 알려주며 olefile 모듈 사용을 시작하게 된다. 이후 FileHeader 스트림을 읽어 hwp string과 버전 필드를 읽어 hwp 파일 여부와 hwp 버전을 출력한다. 이제 이런 검증과정들을 마치고 악성코드 체크를 시작하게 된다.



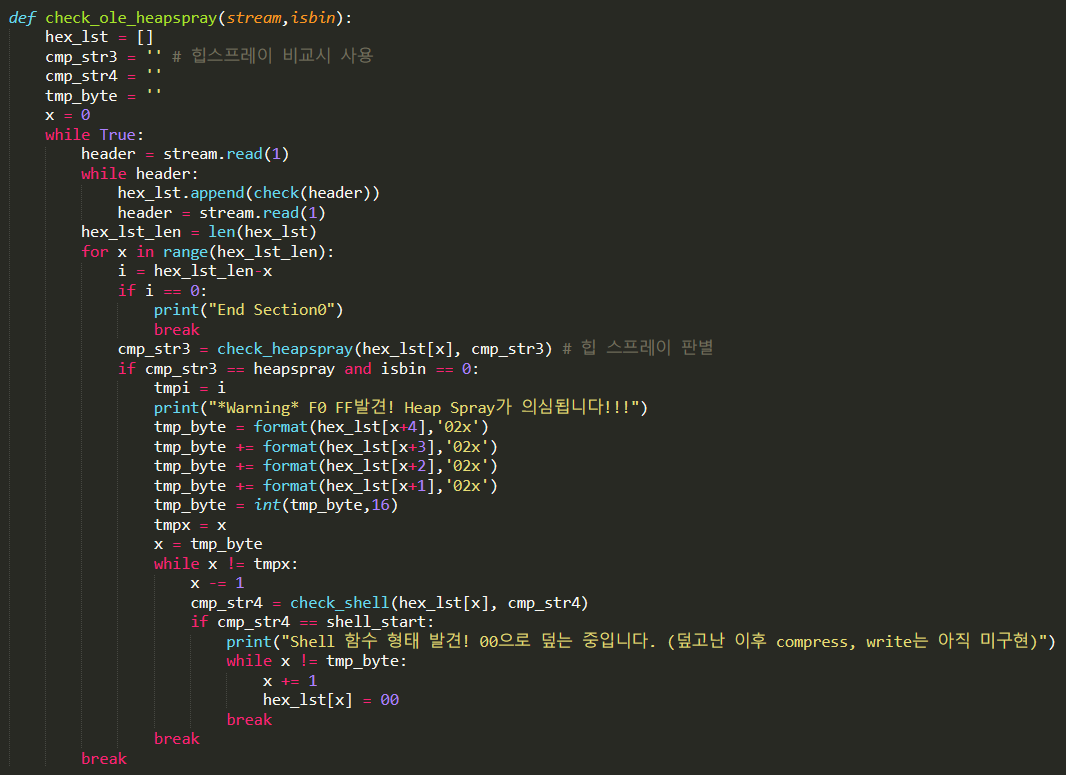
<사진 8. hangeulsarang.py>

일단 olelist를 사용하여 ole 파일안에 존재하는 섹션리스트를 불러오고 하나하나 검증을 시작한다. zlib으로 압축되어 있는 섹션과 압축 되어있지 않은 섹션으로 1차 분류하였다. 이후 DocInfo와 BinData를 예외 상황이 많으므로 2차로 분류하고 check를 시작한다. 현재 로직은 check\_entrop함수를 거쳐 평균 바이트평균에서 크게 벗어나는 바이트가 존재하면 heap spray를 의심하고 이후 check\_ole\_heapspray 함수를 수행하여 F0 FF를 찾음으로써 Size를 통해 heap spray를 재 판별한다.



<사진 9. hangeulsarang.py>

check\_entropy 함수이다. 바이트 평균 구하는 코드와 유사하며 해당 코드에선 한 파일에 대한 바이트 빈도수를 구하고 바이트 평균과 크게 벗어나게 정의한 Maximum과 비교하여 heap spray을 판별한다. 여기서 isbin 인자를 넣어 BinData에서 일어나는 FF의 많은 빈도수를 고려해 BinData일 경우 range를 255까지로 설정하여 FF는 검사하지 않도록 하여 오탐을 줄였다. 그리고 여기서 Maximum보다 큰 바이트가 발생할 경우 다음check\_ole\_heapspray함수를 수행하기 위하여 return으로 True를 준다.



<사진 10. hangeulsarang.py>

check\_ole\_heapspreay 함수이다. 해당 함수에선 tag id, level, size 중 size 부분을 통해 heap spray를 의심할 수 있으므로 F0 FF를 탐색한다. 여기서 해당 바이트가 존재할 경우 뒤에 size를 의미하는 4바이트를 읽어 그 크기만큼 이동한다. 이후 x를 1바이트씩 감소시키며 쉘 코드형태를 역으로 탐색한다. 쉘 코드 시작부분에 주로 존재하는 nop push push push~~부분을 탐색하고 쉘 코드가 의심되면 해당 바이트부터 위에서 도착했던 바이트까지 x가 다시 역으로 증가하며 해당 바이트들을 00으로 덮어버린다. 이론적으로 원래 수행하던 쉘 코드를 0으로 덮어버리므로 실행하지 못하게 된다. 계획상 덮은 이후 다시 압축하여 해당 파일에 write해야하는데 이 부분은 추후 보완할 예정이다.

## 2.3 사용법

* **hangeulsarang\_average.py**

- 파일에 있는 진단 폴더 및 진단 파일 개수 수정 후 실행

- python hangeulsarang\_average.py

* **hangeulsarang.py**

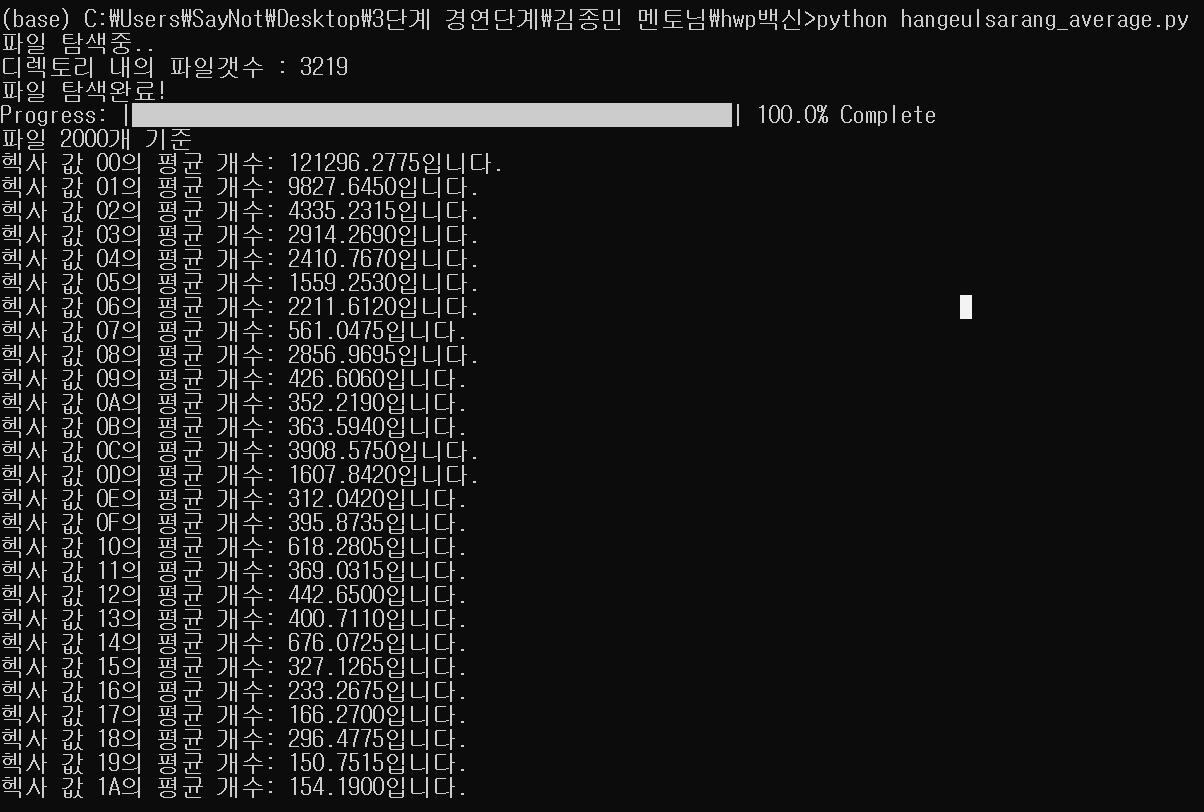
- 인자로 진단을 하고 싶은 hwp가 들어있는 폴더 경로 지정

- python hangeulsarang.py ./malhwp/malhwp

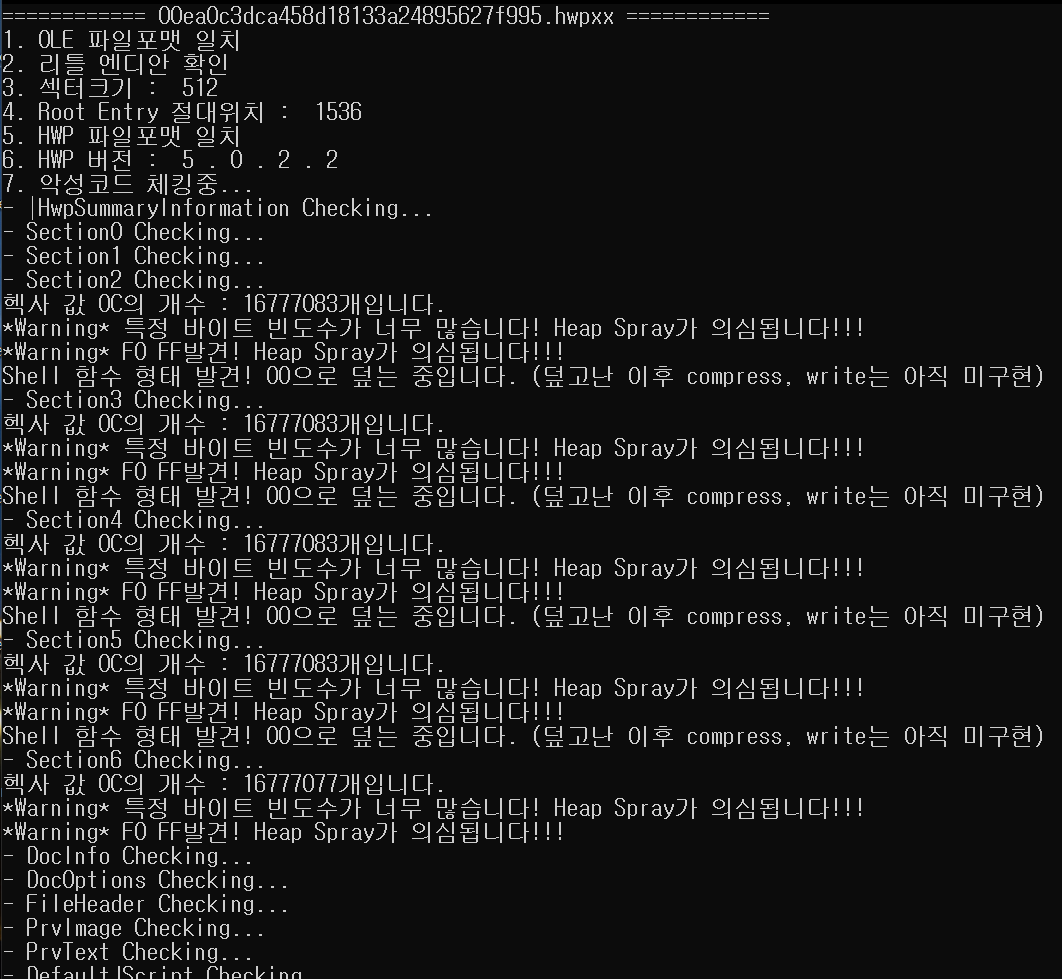
# 3. 결과 및 보완점

## 3.1 결과

1. 다음은 hangeulsarang\_cmp.py를 실행한 화면이다.



2. 다음은 hangeulsarang.py ./malhwp (김종민 멘토님 5개 hwp 악성파일 담긴 폴더)를 실행한 화면이다. (사진은 분량상 중략)



다음과 같이 OLE 파일 포맷, 리틀 엔디안, 섹터 크기, 루트 엔트리 위치, HWP 파일 포맷, 버전을 체크하고 빈도수를 체크한다. 이후 의심되는 size를 잡아 이동하여 쉘 코드를 탐색하고 00으로 덮어쓴다. 수행한 결과는 다음과 같다.

* **00ea0c3dca458d18133a24895627f995.hwpxx**

- Secion1~Section5까지 Heap Spray 탐지 성공, 쉘 코드 찾아 0으로 덮음.

- Secion6은 Heap Spray 탐지 성공, 쉘 코드 찾지 못함.

* **2.\_기술료\_지급대상자\_명단(12.09.06\_현재).hwp**

- Section0에서 Heap Spray 탐지 성공, 쉘 코드 찾지 못함.

* **2013년 대구세계에너지총회.hwp**

- Section0에서 Heap Spray 탐지 성공, 쉘 코드 찾지 못함.

* **조어도 영유권 분쟁.hwp**

- Section1~Section5까지 Heap Spray 탐지 성공, 쉘 코드 찾아 0으로 덮음.

* **한반도통일대토론회.hwp**

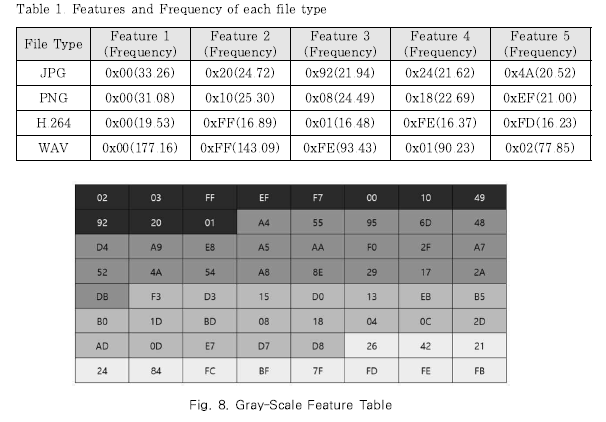
- Section1~Section5까지 Heap Spray 탐지 성공, 쉘 코드 찾아 0으로 덮음.

**결과 : Heap Spray는 100% 탐지 하였고, 쉘 코드는 약 84% 탐지하였다 (18개 Section 중 3개는 탐지 못함).**

## 3.2 보완점 및 아이디어

해당 프로그램은 많이 미약하다. 아직 압축/쓰기가 구현되지 않았으며, BinData를 제대로 탐색하고 분류하지 못한다. Maximum도 결국엔 직접 정한 기준이라 파해 당할 가능성이 존재한다. 그래서 Maximum은 각각 섹션과 영역마다 각각의 Maxmimum을 부여 받아야 하며 평균보다 ‘많다’라는 기준을 정확한 근거로 명시해야 할 것이다. 그리고 BinData에 처음 설명하였듯이 FF값이 많고 그 외의 바이트 개수 양도 많으므로 Heap Spray로 오탐이 날 수 있는 가능성이 있어 보완이 필요하다. 아직 완벽한 코드가 아니므로 미비점이 여럿 존재해 과제 제출 이후에도 지속적으로 보완 해 나갈 예정이다.

아이디어로 한가지 방식을 제안하려고 한다. 프로젝트때 이용한 방법을 통해 해당 부분도 탐지가 가능할 것이라 예측한다. CNN을 사용하여 AI로 분류하는 방법이다. 이미 파일을 그대로 Gray-scale변환하여 정상파일과 악성파일을 학습시키고 악성파일을 분류하는 방법은 여럿 존재한다. 하지만 악성HWP 분류에 사용한다고 가정했을 때 NOP Sled, 혹은 쉘코드 영역이 바뀌었다고 Gray-scale에 있어서 큰 차이를 보이지 않을 것이라 생각한다. 제안하는 방법은 바이트 평균을 계산하여 바이트 중 큰 특징을 보이는 바이트들을 추출하고 이 바이트들로 테이블을 만들어 테이블을 Gray-scale화 시킨다. 이렇게 하면 큰 특징을 보이는 바이트들로만 Gray-scale이 형성되어 여기서 쉘코드로 많이 쓰이는 바이트들을 테이블에 삽입하여 일반 HWP 파일들과 큰 차이를 만들어 정상/악성 분류 정확도가 높을 것이라 생각한다.

[해당 논문은 확정이 팀이 2차 프로젝트때 작성한 논문 (디지털 포렌식 공모전 우수작 논문)]

참고한 레퍼런스1. krCERT의 한글 취약점을 통해 유포되는 악성코드 동향 및 분석 보고서

참고한 레퍼런스 2. 한글 文書파일의 靜的分析을 通한 惡性코드 探知에 관한 硏究

참고한 레퍼런스 3. 바이트 평균의 Gray-Scale화를 통한 Signature가 존재하지않는 멀티미디어 데이터 조각 파일 타입 분류 연구