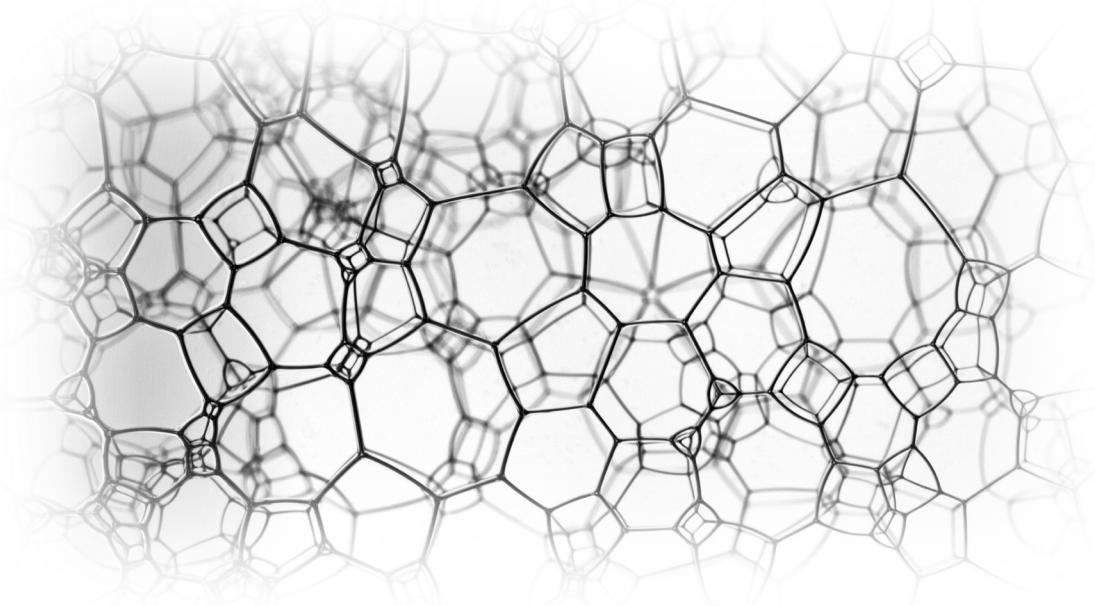

KDFS 2018

분석보고서

대회 제출용



"Every contact leaves a trace"

- Dr. Edmond Locard -

분석자 : 오동빈 (경찰대학 교무과)
이종찬 (PLAINBIT)
박재유 (LG전자)

인가된 자 외에 열람을 금함

목 차

1. 분석 개요	4
1.1. 개요.....	4
1.2. 분석 목표 확인	4
1.3. 증거물 개요	5
1.3.1. 증거물 확보 방안 및 확보 과정.....	5
1.3.2. 확보된 증거 이미지 정보	5
1.4. 이미지 분석	6
1.4.1. 분석 결과 요약	6
1.4.2. 사고 타임라인.....	7
2. 분석 상세	8
2.1 분석 도구 정보.....	8
2.2 분석 이미지 기본정보 확인	9
2.2.1 운영체제 정보 확인	9
2.2.2 악성코드 실행 시점 파악	10
2.2.3 악성코드 유입 경로 및 실행 원인 파악.....	15
2.2.4 악성코드 전파 흔적 확인	17
2.2.5 악성코드 지속 및 은닉 흔적 확인	18
2.3 증거 분석 결과	19
3. ZIP 파일 복구	20
3.1 개요	20
3.2 파일 카빙 시도	20
3.2.1 비활당영역에서의 카빙	20
3.2.2 볼륨 쉐도우 카피(VSC)를 이용한 카빙	24
3.2.3 페이지파일을 이용한 카빙	28

3.3 파일 메타데이터 카빙.....	30
3.3.1 Yara를 이용한 메타데이터 탐지.....	30
3.3.2 페이지파일 메타데이터 카빙.....	33
3.3.3 Zip 파일의 Central Directory 해석.....	35
3.4 OSINT를 이용한 asd.exe 파일 획득	37
3.5 파일 동일성 검증	39
3.5.1 ZIP Central Directory 메타데이터를 이용한 파일 동일성 검증	39
3.5.2 Zip 파일 압축 크기를 이용한 동일성 검증	39
3.5.3 asd.exe 행위 분석을 이용한 동일성 검증.....	41
3.5.4 파일 슬랙 부분을 이용한 동일성 검증.....	43
3.6 소결.....	46
4. 랜섬웨어 행위 분석 및 파일 복호화.....	47
4.1 악성코드 샘플 파일 입수.....	47
4.2 정적 분석	48
4.2.1 PE 바이너리 정보.....	48
4.2.2 Decompile 분석.....	50
4.3 동적 분석	52
4.3.1 가상머신 환경에서의 감염 실험.....	52
4.3.2 디버거를 통한 동적 분석	54
4.3.3 활성 메모리 분석	58
4.4 파일 암호화 작동 방식 분석	62
4.4.1 AES-256 대칭키 생성 및 파일 암호화.....	63
4.4.2 AES 키에 대한 RSA 암호화(하이브리드 방식).....	65
4.4.3 암호화된 Key Data를 파일에 저장하는 방식	66
4.4.4 파일 암호화 방식	67

4.5 파일 복호화 방안	68
4.5.1 Encrypted Key Data 복원 방안	68
4.5.2 Kaspersky RakhniDecryptor 를 통한 파일 복원	71
4.6 소결	80
5. 랜섬웨어 실행 차단 방안	81
5.1 일반적인 랜섬웨어 대책	81
5.2 기술적 측면의 연구 동향	82
5.2.1 UNVEIL	82
5.2.1 블록암호의 구현 특성에 따른 식별	83
5.3 Jaff 랜섬웨어에 특화된 대응 방안 제안	83

별 첨

[첨부 1] analyzeMFT 결과 (result.xlsx)

[첨부 2] 파일시스템 트랜잭션 로그 (UsnJrnl.csv,LogFile.xlsx,LogFile.csv)

[첨부 3] 파일시스템 내 파일 해쉬 목록 (File_Hash_List.csv)

[첨부 4] query_exe_md5_to_vt.py

[첨부 5] get_encrypted_file_lists.py

[첨부 6] 랜섬웨어 감염 파일 목록 (encrypted_file_list.xlsx)

[첨부 7] 작성한 YARA Rule (yara_rule.txt)

[첨부 8] 악성코드 샘플 (asd_pw_KDFS2018.zip)

암축 비밀번호 : KDFS2018

[첨부 9] 수정한 랜섬 노트 (!!!TO SAVE YOUR FILES!!!!_fixed.txt)

[첨부 10] 복호화 도구 로그 내역 (decryptor_log.txt)

별첨 내용은 제출 메일 내 “별첨.zip” 첨부파일을 통해 확인할 수 있음

1. 분석 개요

1.1. 개요

본 장에서는 주어진 분석 목표와 증거물을 확인하고 이에 따라 사고 분석의 방향성을 정립 후, 분석 결과 및 이후 진행된 과정인 파일 복구와 악성코드 분석의 필요성에 대해 제시한다.

따라서 분석 목표를 통해 상세 분석이 필요하다고 식별되는 포렌식 아티팩트를 먼저 정의한 후 확보할 수 있도록 체계를 구축하였으며, 증거 확보와 해당 증거의 무결성 및 원본성을 보장할 방안을 마련하고 이에 해당하는 절차를 준수하여 수집하였다. 이후 위의 관점을 바탕으로 증거 이미지에서 분석 대상 포렌식 아티팩트를 선별하였다. 해당 아티팩트를 추출한 후 각각의 아티팩트에 대한 상세 분석을 진행하였으며, 이를 통해 침해사고 발생 원인을 확인하였으며 사고 경위와 그 근거를 서술하였다. 이후 동일한 침해사고에 대해 대응할 수 있는 방안에 대해 제시하였다.

1.2. 분석 목표 확인

본 분석 대회의 공고[1]를 통해 목표를 확인하였다. 공고문에 제시되어 있는 내용 중 분석 목표 및 첨부를 요구하는 중점적 사항을 정리하면 다음과 같다.

목표	랜섬웨어 감염 시 침해사고 분석
분석 중점 1	랜섬웨어가 피해자 컴퓨터에서의 행위 (유입 경로, 행위 분석)
분석 중점 2	분석한 결과를 토대로 타임라인 생성

Table 1. 분석 중점 사항

제시된 목표를 통해 본 분석 케이스는 랜섬웨어 의해 발생한 침해사고이며, 분석 결과로 도출하고자 하는 것은 사건의 경위 파악 및 시간 별 행위 내역 파악으로 정리할 수 있다. 랜섬웨어 침해사고의 경우 가장 우선적으로 실시해야 하는 것은 추가적인 파일 감염을 막기 위해 시스템의 전원 차단 및 파일 암호화에 사용한 키 복구를 위해 메모리 덤프를 확보하는 것 등이며[2] 이는 침해사고 대응팀의 판단에 따라 수행한다. 최종적으로 가능하다면 암호화된 파일을 복구하는 것과 네트워크 등을 통해 타 시스템으로 전이되지 않도록 후속 대응책 마련에 목표가 있다.

"Table 1"의 분석 중점 1과 같이 악성코드가 PC에서의 행위를 밝히기 위해 악성코드의 유입 흔적, 실행 흔적, 전파 흔적, 은닉 및 지속 흔적 순으로 상세 분석을 진행하였으며, 이 결과를 토대로 분석 중점 2에 해당하는 타임라인을 생성하였다. 이후 악성코드의 실제 동작 원리 파악하기 위해 삭제된 악성코드 복구 및 원본을 확보, 악성코드 행위를 분석하였다. 이 결과를 통해 암호화된 파일에 대해 복호화를 진행하였다.

1.3. 증거물 개요

1.3.1. 증거물 확보 방안 및 확보 과정

디지털 포렌식에서 일반적으로 디스크 이미징 등을 통해 증거물을 확보할 경우 무결성을 보장하기 위한 방안으로 이미징 과정에서 해시(HASH)를 계산하며, 이후 계산한 해시가 이미징 결과 파일과 일치하는지에 대한 검증 과정을 거치게 된다. 또한 이 해시 검증의 결과가 증거물 확보 당시에 이루어졌음을 보증하기 위해 결과 화면을 사진기 등으로 촬영하고, 이후 이미징 파일의 해시값과 사진의 내용이 일치함을 확인하여 원본성을 확보할 수 있다.

대회 주최사에서는 증거 이미지를 1.2절에서 밝힌 공고 게시글을 통해 다운로드 할 수 있는 링크[3] 공유하였으며, 이를 다운로드하여 증거물을 확보할 수 있었다. 하지만 증거물에 대한 파일 해시 등을 제공하지 않아 무결성 및 원본성은 별도로 비교할 수 없었다.

1.3.2. 확보된 증거 이미지 정보

1.3.1절에서 밝힌 공고의 링크를 통해 받은 이미지의 정보는 다음과 같다.

파일명	2018 Digital Forensic Challenge.001
분석 종점 1	30GB (31,457,280 KB)
파일 형태	Raw Disk Image (DD)
파일해시(MD5)	D5DDDAE23E2203124566D0FE598AFA35
파일해시(SHA1)	5BB35A5A92FFCC7196D2FEA7EFED9A51FA63F6CB

Table 2. 증거 이미지 정보

이미징 파일의 해시를 확인하기 위해 사용한 도구의 정보는 다음과 같다.

도구명	HashTab64
버전정보	6.0.0.34
제조사	Implbits Software
다운로드 링크	http://implbits.com/products/hashtab
파일 해시(MD5)	6E6559AC4C7ABF6F7D60165E1C2F9B65
파일 해시(SHA1)	2D0FAF4D27680C9C971F8FFBF2B0152B8FB9C4C2

Table 3. 파일 해시 검증 도구 정보

1.4. 이미지 분석

1.4.1. 분석 결과 요약

1.2절에서 정의한 분석 중점인 악성코드의 유입 흔적, 실행 흔적, 전파 흔적, 지속 및 은닉 흔적
토대로 분석 결과를 요약하면 아래와 같다.

- 악성코드 유입 흔적

공격자는 피해자가 사용하는 네이버 메일 계정에 스파이어피싱 메일을 전송하였으며, 피해자는
메일에 포함된 첨부파일을 다운로드하였다.

- 악성코드 실행 흔적

메일의 첨부파일은 Zip 형태의 압축 파일이었으며, 해당 파일에는 랜섬웨어로 추정되는
"asd.exe"라는 실행파일과 그 파일을 실행하도록 유도하는 "이력서.doc.lnk" LNK 파일이 포함되어
있었으며, 사용자는 LNK 파일을 이력서를 작성한 문서 파일로 착각하여 실행하였다.

- 악성코드 전파 흔적

사용자가 설치한 원격 제어 프로그램은 찾을 수 없었으며, 추가적인 원격 제어 프로그램 역시
발견되지 않았다. 또한 Windows 원격 데스크톱 역시 설정이 꺼져있었으며, 이 프로그램을 통해
타 PC로 연결된 흔적 역시 발견되지 않았다.

- 악성코드 지속 및 은닉 흔적

파일명 은닉, 파일 경로 은닉, 사용하지 않는 영역에 숨김 등의 정황이 발견되지 않았다.

1.4.2. 사고 타임라인

상세 분석 결과 본 사고의 경위를 정리하면 다음과 같다.

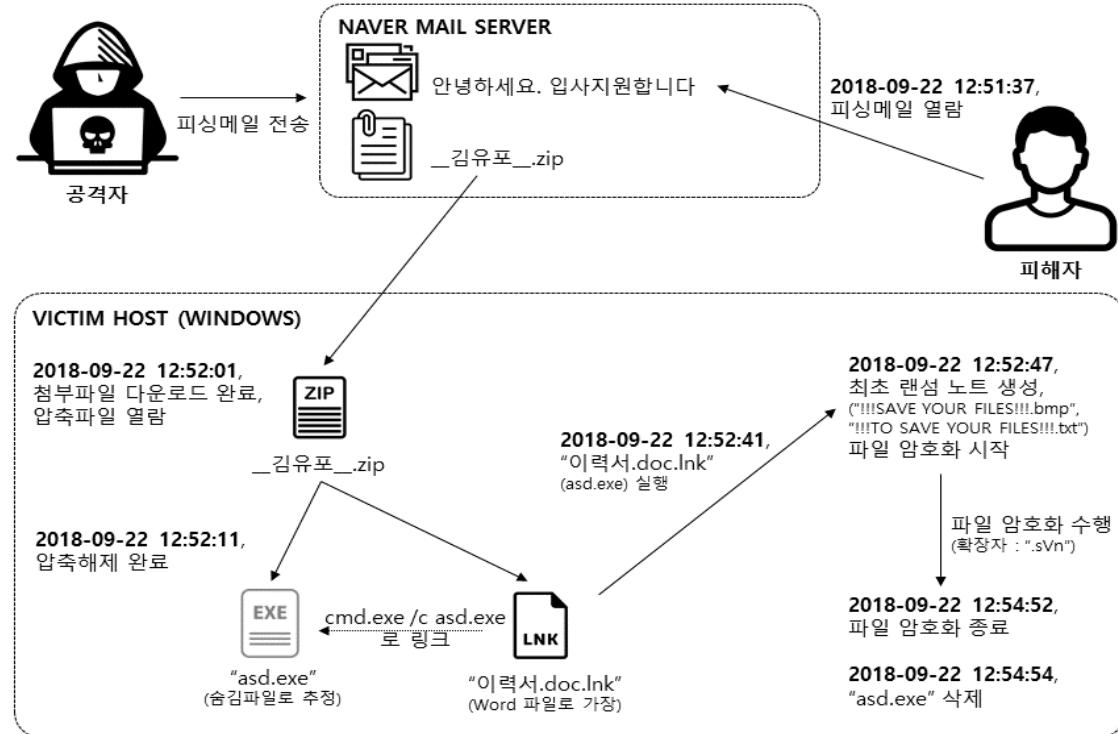


Fig 1. 분석 결과 요약

시간정보 (UTC+09:00)	행 위	출처
2018-09-20 20:28:37	운영체제 설치	레지스트리
2018-09-22 12:51:37	피해자, 피싱메일 열람	웹브라우저 히스토리
2018-09-22 12:52:01	PC에 피싱메일에 첨부된 것으로 추정되는 압축파일 다운로드	파일시스템 로그
2018-09-22 12:52:01	피해자, 압축파일 열람	웹브라우저 파일 열람 기록
2018-09-22 12:52:11	압축파일 압축해제	파일시스템 로그
2018-09-22 12:52:11	압축파일 내 "asd.exe" 파일 압축해제 완료	파일시스템 로그
2018-09-22 12:52:11	압축파일 내 "이력서.doc.lnk" 파일 압축해제 완료	파일시스템 로그
2018-09-22 12:52:41	"이력서.doc.lnk" 파일 실행, 파일 암호화 시작	레지스트리
2018-09-22 12:52:47	최초 랜섬노트 생성	파일시스템
2018-09-22 12:54:52	파일 암호화 종료	파일시스템
2018-09-22 12:54:54	악성코드 원본 "asd.exe" 삭제	파일시스템 로그

Table 4. 사고 타임라인

2. 분석 상세

본 장은 침해사고를 분석하는 논리적 흐름으로 기술되었다. 악성코드에 의한 침해사고 분석 시 악성코드 유입, 실행, 전파, 지속 및 은닉 중심으로 분석해야 한다. 이때, 최우선적으로 악성코드의 최초 실행 시점을 파악하는 것이 중요하며, 이후 유입 흔적 확인을 통해 해당 시스템 혹은 사설 네트워크 보안성 강화에 대한 방안을 마련한다. 또한 동일 네트워크 내 타 시스템으로 전파되었을 가능성을 확인하여 추가 피해를 차단하고, 추후 감염 PC 내에서 악성코드가 추가적인 동작을 수행하려는 흔적이 있는지 확인해야 한다.

2.1 분석 도구 정보

분석에서 사용한 도구의 정보를 요약하면 아래의 표와 같다.

도구명	버전정보	제조사/개발자	사용 용도
FTK Imager	4.2.0.13	AccessData	이미지 기본정보 확인 및 파일 추출
WinHex	15.8	X-Ways Software	이미지 및 파일시스템 탐색
analyzeMFT	1.15	dkovar	NTFS 파일시스템 구조 상세 분석
REGA	1.5.0.4	DFRC	Registry 분석
WinPrefetchViewer	1.35	NirSoft	Windows Prefetch 분석
AppCompatCacheParser	1.3.0.0	Eric Zimmerman	응용프로그램 호환성 캐시 분석
Strings	2.53	Mark Russinovich	문자열 추출
Notepad++	7.5.8	Notead++ team	문자열 확인
UFTLinkParser	0.9	UFT Team	LNK 파일 분석
HxD	2.0.0.0	Maël Hörz	파일 Hex 데이터 확인
WEFA	1.5.1	DFRC	웹 브라우저 캐시 및 히스토리 분석
Message Analyzer	1.4	Microsoft	Windows Event Log 분석
R-Studio	7.7.159851	R-tools Technology	파일 카빙 및 복구
plaso	20170930	log2timeline	통합 타임라인 분석
YARA	3.8.1	Yara Project	패턴 매칭
010 editor	6.0.2	Sweet Scape	ZIP 파일 구조 확인
Ollydbg	1.0.10.0	Oleh Yuschuk	악성코드 디버깅

Table 5. 분석에 사용된 도구 정보

2.2 분석 이미지 기본정보 확인

분석 이미지의 파티션 정보와 파일 시스템, 운영체제 종류를 파악하기 위해 FTK Imager를 사용하였으며, 그 결과는 다음과 같다.

종류	크기	볼륨 정보	포맷일자 (UTC+09:00)	볼륨시리얼번호	용도
NTFS	30GB	주 파티션	2018-09-20 20:23:47	1467-94CB	운영체제 설치

Table 6. 이미지 내 파일시스템 정보

이미지 내 파티션은 하나만 존재하였으며, 해당 파티션은 NTFS 형식으로 Windows 운영체제가 설치되어 있는 것을 확인할 수 있었다. 이하 분석은 Windows에서 발생할 수 있는 아티팩트를 분석하였다.

2.2.1 운영체제 정보 확인

Hive 파일명	경로	크기	저장 정보
BCD-Template	%systemroot%/System32/config/	28KB	부팅 관련 정보
COMPONENTS	%systemroot%/System32/config/	29.5MB	시스템 컴포넌트 정보
DEFAULT	%systemroot%/System32/config/	256KB	계정 생성 시 필요정보
SAM	%systemroot%/System32/config/	256KB	계정정보
SECURITY	%systemroot%/System32/config/	256KB	감사정책, 권한정보
SOFTWARE	%systemroot%/System32/config/	36.7MB	시스템 운영 관련 정보
SYSTEM	%systemroot%/System32/config/	12MB	시스템 부팅 관련 설정
Default.NTUSER.DAT	['Default' %userprofile%]	256KB	계정 관련 설정 정정
Kang_Lion.NTUSER.DAT	['Kang-Lion' %userprofile%]	1MB	계정 관련 설정 정보
Kang_Lion.UsrClass.dat	['Kang-Lion' %localappdata%]/Microsoft/Windows	512KB	계정 관련 프로그램 설정 정보

Table 7. 수집한 Registry Hive 파일 정보

Windows 시스템의 전반적인 정보를 확인하기 위해 FTK Imager로 위의 Table 7과 같은 Registry Hive 파일을 수집하였다. 이때 Windows는 사용자의 홈 디렉토리는 운영체제가 설치된 볼륨의 Root 디렉토리 하위의 'Users' 디렉토리에 사용자 이름별로 구성되며, 각 사용자의 Registry Hive 파일인 'NTUSER.DAT', 'UsrClass.dat'은 각 사용자의 홈 디렉토리 하위에 위치한다. 본 분석에서는 사용자의 이름을 Hive 파일 앞에 기술함으로써 각각을 구분하였다.

이후 Registry Hive 파일 분석 도구인 REGA를 통해 이미지에 설치된 운영체제의 정보를 다음과 같이 확인하였다.

제품명	Windows 7 Home Premium
소유자	Windows 사용자
제품 ID	00359-OEM-9818072-56929
제품 버전	Multiprocessor Free 6.1.7601.17514.x86fre.win7sp1_rtm.101119-1850
설치 날짜 (UTC+09:00)	2018-09-20 20:28:37
시스템 루트	C:/Windows

Table 8. 설치된 운영체제 정보

2.2.2 악성코드 실행 시점 파악

일반적으로 랜섬웨어 침해사고의 경우 파일이 감염되기 시작한 때를 악성코드가 실행 기점으로 판단할 수 있기 때문에 분석가 입장에서는 적어도 타 악성코드에 비해 실행 시점을 비교적 빠르게 파악할 수 있다. 하지만 악성코드 중 분석을 방해할 목적으로 시간 정보를 수정하는 경우도 있어 유의해야 한다.

NTFS 파일 시스템은 파일의 경로와 메타데이터 등을 \$MFT에 저장하므로 이 파일을 시간 기준으로 분석하면 파일 간 전후 상황을 파악할 수 있다. FTK Imager를 통해 위의 파일을 확보하였으며, 그 정보는 다음과 같다.

아티팩트명	경로	저장 정보
\$MFT	%systemdrive% /	NTFS 내 파일 경로 정보 및 메타데이터

Table 9. 수집한 파일시스템 구조 해석 관련 아티팩트 정보

이후 \$MFT 분석 도구인 analyzeMFT를 사용하여 좀 더 상세히 파일 시스템 내부 구조를 확인하였다. analyzeMFT는 CSV 결과 파일 저장 시 인코딩을 UTF-8로 처리한다. 이때 CSV 결과물을 확인할 때, 스프레드시트 프로그램인 Excel로 열 경우 파일 시스템 내의 경로에 포함된 한글이 인코딩이 깨져 출력된다. 해당 프로그램은 기본적으로 ANSI 인코딩 방식을 사용하기 때문이다. 따라서 본 분석에서는 해당 오류를 기존 소스코드에서 직접 수정해 사용하였으며[4], 그 결과 파일은 '첨부 1. analyzeMFT 결과'를 통해 확인할 수 있다.

'첨부 1'을 참고하면 확장자가 ".svn"인 파일을 다수 발견할 수 있다. svn 파일이 실제로 감염된

파일인지 확인하기 위해 WinHex로 sVn을 확장자로 갖는 파일 중 '[1]06-338.pdf.sVn'의 16진수 값의 윗부분만 살펴보면 다음과 같다.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
5A9067000	02	01	00	00	33	31	36	38	33	33	38	35	35	38	30	38	316833855808
5A9067010	33	35	32	36	32	38	33	31	36	31	36	34	33	38	31	36	3526283161643816
5A9067020	38	39	38	35	37	35	35	37	35	38	36	37	31	35	31	35	8985755758671515
5A9067030	30	33	38	36	32	34	32	37	37	38	38	31	30	34	33	30	0386242778810430
5A9067040	38	39	31	38	34	38	32	39	31	37	32	34	36	38	39	30	8918482917246890
5A9067050	35	37	38	39	32	37	35	37	38	32	35	38	30	38	38	30	5789275782580880
5A9067060	30	35	38	36	32	30	37	34	35	35	31	31	33	36	31	31	0586207455113611
5A9067070	31	38	30	37	34	35	36	37	36	32	39	33	39	31	39	32	1807456762939192
5A9067080	32	30	30	34	20	32	39	35	38	33	32	36	37	32	32	35	2004 29583267225
5A9067090	37	37	37	36	30	39	33	33	33	37	32	37	32	39	35	35	7776093333727295
5A90670A0	32	39	31	35	37	37	38	32	38	37	35	34	35	38	38	36	2915778287545886
5A90670B0	36	32	37	37	30	36	39	35	37	39	38	33	34	39	39	30	6277069579834990
5A90670C0	30	32	33	33	33	35	38	36	35	39	38	34	34	36	37	33	0233358659844673
5A90670D0	32	31	39	30	31	36	30	38	34	30	39	33	37	35	31	33	2190160840937513
5A90670E0	36	32	32	37	33	31	35	34	36	30	32	36	35	30	38	37	6227315460265087
5A90670F0	33	35	33	33	31	31	34	34	31	37	35	36	34	38	34	32	3533114417564842
5A9067100	39	30	33	39	33	20	A0	E6	19	00	EA	AE	73	40	54	06	90393 n è@s@T
5A9067110	90	B4	10	9F	3C	F3	E3	66	25	E8	81	79	3E	54	7C	D9	' I<éäf%è y>T Ù
5A9067120	D7	2E	28	CA	DF	E7	EB	7B	91	05	AC	47	F2	06	5A	BE	×, (Èççé{ ' ~Gò Zë
5A9067130	36	75	30	9B	EE	BE	56	80	D2	7C	62	6F	64	CC	FB	41	6u0 is%V Ø bodíúA
5A9067140	AD	AD	6D	FE	27	83	A8	EE	A5	B6	BE	4E	FD	D8	DC	2D	--mp' I"iÝMNYØÙ-
5A9067150	7A	42	BF	56	81	C6	BA	E6	75	40	A4	E6	E5	DA	94	30	zBéV Èøæu@xæåÙ O
5A9067160	6F	F9	51	73	E0	52	ED	A8	A8	C5	09	07	0F	8E	DC	FF	ouQsàRi " "À IÙy

Fig 2. [1]06-338.pdf.sVn 파일의 16진수 값

일반적으로 PDF 포맷의 문서 파일은 '0x25, 0x50, 0x44, 0x46'(ASCII 코드로 '%PDF')을 파일 헤더 부분에 시그니처로 가지게 된다. 하지만 이 파일 이외의 sVn을 확장자로 갖는 시스템 내의 파일들은 위 그림의 빨간색으로 표시한 것처럼 '0x02, 0x01, 0x00, 0x00'을 파일 헤더로 가지고, 일정 크기의 블록 후에 해석이 불가능한 데이터들이 위치한다. 이를 통해 sVn 파일은 랜섬웨어에 의해 암호화된 파일임을 유추할 수 있다.

또한 랜섬웨어에 의해 암호화된 것으로 추정되는 파일들과 같은 경로에 "!!!SAVE YOUR FILES!!!.bmp"와 "!!!TO SAVE YOUR FILES!!!.txt"의 이름을 가진 두 파일을 같이 확인할 수 있다. WinHex로 두 파일을 추출하였으며, 각각의 내용은 다음과 같다.



Fig 3. "!!!SAVE YOUR FILES!!!.bmp" 의 내용

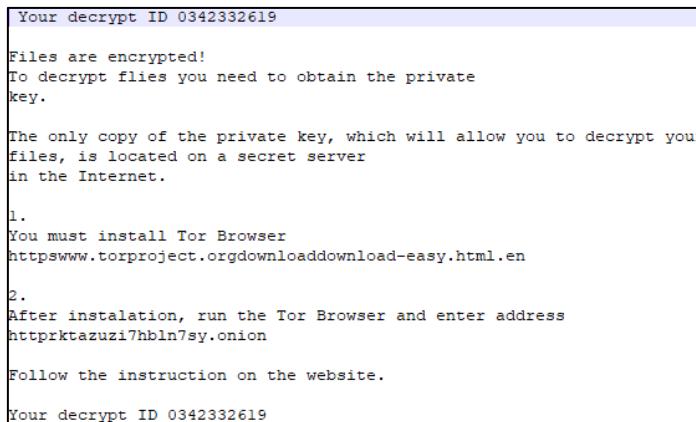


Fig 4. “!!!TO SAVE YOUR FILES!!!.txt” 의 내용

위의 두 파일은 랜섬웨어가 파일을 암호화한 후 피해자로부터 금품을 요구하는 랜섬 노트로, 토르 브라우저 설치 및 접속을 통해 공격자가 구축한 서버로 우회 접속해 지시에 따르도록 유도하고 있다.

암호화된 파일 및 랜섬 노트를 확인 후 최초 감염 시간을 파악하기 위해 NTFS의 Standard Information(이하 Std Info)의 수정 시간을 기준으로 정렬하였다. 하지만 파일이 암호화된 시간과 랜섬 노트의 생성시간의 순서가 일정하지 않았다. 다음은 Std Info 수정시간을 오름차순으로 정렬하였을 때 'sVn'을 확장자로 갖는 파일 중 가장 오래된 파일과 그 경로에 포함된 랜섬 노트의 시간 정보이다.

Std Info 수정 시간 (UTC+09:00)	Std Info 경로	Std Info 접근시간 (UTC+09:00)	Std Info Entry 시간 (UTC+09:00)
2018-09-20 21:30:12	['Kang-Lion' %appdata%]/ESTsoft /Cooperation/SwingBrowser_1_6.rtf.sVn	2018-09-20 21:09:28	2018-09-22 12:53:04
2018-09-22 12:53:04	['Kang-Lion' %appdata%]/ESTsoft /Cooperation/!!!SAVE YOUR FILES!!!.bmp	2018-09-22 12:53:04	2018-09-22 12:53:04
2018-09-22 12:53:04	['Kang-Lion' %appdata%]/ESTsoft /Cooperation/!!!TO SAVE YOUR FILES!!!.bmp	2018-09-22 12:53:04	2018-09-22 12:53:04

Table 10. 접근시간이 상이한 암호화된 파일과 랜섬노트 예시

랜섬웨어가 실행된 시점부터 파일이 암호화가 되었다면 특정 시간 이후로 일렬로 나열되어야 하지만 그렇지 않다. 이것으로 수정시간을 조작한 안티 포렌식이 이루어진 것을 확인 할 수 있다. 이렇게 NTFS 파일시스템에서 Std Info의 시간 값이 조작되어 있는 경우 Std Info의 MFT Entry 변경 시간을 참고하면 파일 시스템에서 일어난 각 행위에 대해 좀 더 구체적인 시간을 확인할 수 있다. Std Info의 MFT Entry 변경 시간을 기준 오름차순으로 정렬한 결과, 최초로 생성된 랜섬 노트는 다음과 같다.

Std Info Entry 시간 (UTC+09:00)	경로
2018-09-22 12:52:47	/fdfd0bf95c1dae9258c82b1350e8/!!!SAVE YOUR FILES!!!.bmp
2018-09-22 12:52:47	/fdfd0bf95c1dae9258c82b1350e8/!!!!TO SAVE YOUR FILES!!!!.txt

Table 11. 최초로 생성된 랜섬노트의 경로

이를 통해 파악한 랜섬웨어 실행 시점은 '2018-09-22 12:52:57' 이전으로 판단되며, 이하 분석은 위에서 파악한 시간과의 연관성을 분석하여 실제 랜섬웨어 실행파일을 유추하였다.

Windows 호스트 분석시 악성코드의 실행 흔적을 파악하기 위해서는 Prefetch, 응용 프로그램 호환성 캐시, Shellbag 등을 확인한다. 본 이미지에서 수집한 포렌식 아티팩트를 정리하면 다음과 같다.

아티팩트 명	경로	저장 정보
Windows Prefetch	%SystemRoot%/System32/config	Prefetch 관련 파일
RecentFileCache.bcf	%SystemRoot%/AppCompact/Programs	프로세스의 파일 경로 임시 저장
AppCompactCache	UsrClass.dat 하이브 파일 내 Key	SDB를 참조한 응용프로그램 목록
UserAssist Key	NTUSER.DAT 하이브 파일 내 Key	응용 프로그램 사용 흔적

Table 12. 수집한 응용프로그램 실행 관련 아티팩트 정보

악성코드의 실행을 파악하기 위해 흔히 살펴보는 흔적 중 Amcache의 경우 Windows 8 부터 등장한 개념으로, 'RecentFileCache.bcf'를 대체하는 Registry 하이브 형태의 파일이다. 본 분석 대상은 Windows 7이 설치되어 있으므로 이는 존재하지 않아 분석을 진행하지 않았다.

Windows Prefetch는 해석 도구인 WinPrefecthView를 이용하여 확인했으나 악성코드와 관련된 흔적을 확인할 수 없었다. RecentFileCache.bcf 파일은 strings를 이용하여 문자열 추출 후, Notepad++로 내용을 확인하였으나 동일하게 악성코드의 실행 판단 근거는 찾을 수 없었다.

AppCompactCache와 UserAssist Key는 Registry 분석도구인 REGA를 이용하여 분석하였다. 응용 프로그램 호환성 캐시에서는 별다른 흔적을 발견할 수 없었으나, 응용프로그램 사용 흔적에서는 다음의 내용이 악성코드 실행 관련 정보로 의심되었다. 첫번째 응용프로그램 사용 흔적은 정상 흔적으로, 비교를 위해 표기하였다.

계정명	최종실행시각 (UTC+09:00)	실행파일 경로	종류	실행횟수
Kang-Lion	2018-09-20 21:53:42	{7C5A40EF-A0FB-4BFC-874A-C0F2E0B9FA8E}/Microsoft Office/Office14/WINWORD.EXE	CTLSESSION	2

Kang-Lion	2018-09-22 12:52:41	/Users/Kang-Lion/Desktop/0 력서/_김유포_이력서.doc.lnk	CTLSESSION	1
-----------	---------------------	--	------------	---

Table 13. 응용프로그램 사용 로그 중 정상 실행 흔적과 악성코드 실행 의심 흔적

위 파일의 실행시간은 앞서 확인한 랜섬 노트 최초 생성 시간과 비교해 이전에 실행된 가장 가까운 실행파일이다. 일반적으로 LNK 파일이 해당 아티팩트에 남는 경우는 LNK로 연결된 파일이 실행파일인 경우이다. 이때, 상단의 표 두 번째 행에 표기된 “이력서.doc.lnk”의 파일 이름을 확인하면 확장자 “lnk” 이전에 “doc”가 있는 것을 확인 할 수 있다. 문서 파일은 실행되더라도 상단의 표 첫번째 행처럼 “WINWORD.EXE”를 통해 실행되어야 한다. 즉, 해당 흔적은 lnk의 원본 파일이 문서가 아닌 PE 형식의 실행파일이기 때문에 흔적이 남은 것으로 유추 가능하다. 시간 순서상 이 LNK 파일을 통해 실행한 응용 프로그램이 악성코드와 관련 있을 가능성이 높으므로, 해당 파일을 추가적으로 분석하여 “이력서.doc.lnk”의 실행 원본을 추적해야 한다. LNK 파일을 수집한 경로는 다음과 같다.

경로	비고
[‘Kang-Lion’ %userprofile%]/Desktop	
[‘Kang-Lion’ %appdata%]/Microsoft/Windows/Recent	최근 실행 파일 관련
[‘Kang-Lion’ %appdata%]/Microsoft/Windows/Start	
[‘Kang-Lion’ %appdata%]/Microsoft/Internet Explorer/Quick Launch	빠른 시작 관련

Table 14. LNK 파일 수집 경로

수집한 파일 중에서 위의 “이력서.doc.lnk” 가 포함되어 있었으며, 자세한 정보는 다음과 같다. ‘Internet Explorer.lnk’, ‘곰플레이어.lnk’ 파일은 비교를 위해 같이 포함하였다.

접근시간 (UTC+09:00)	LNK 파일명	링크된 경로	실행 옵션	볼륨 시리얼 번호	MAC 주소
2018-09-20 20:30:12	Internet Explorer.lnk	C:/Program Files/Internet Explorer/iexplore.exe	-	1467-94CB	00:0C:29: 43:17:33
2018-09-20 21:52:14	곰플레이어.lnk	C:/Program Files/GRETECH /GOMPlayer/GOM.EXE	-	1467-94CB	00:0C:29: 42:17:33
2018-09-22 12:52:12	이력서.doc.lnk	C:/Windows/System32 /cmd.exe	/c asd.exe	D6D1-DC00	00:0C:29: A1:A9:40

Table 15. 정상 LNK 파일과 악성코드 관련 의심 LNK 파일

‘Internet Explorer.lnk’ 와 ‘곰플레이어.lnk’ 파일의 경우 볼륨 시리얼 번호와 MAC 주소가 일치하지만, “이력서.doc.lnk”의 경우 두 파일의 볼륨 시리얼 번호와 MAC 주소가 서로 다름을 확인할 수 있다. 이는 수집한 타 LNK 파일들과도 마찬가지로 해당 정보들이 서로 일치하는 경우가 있었으나 “이력서.doc.lnk” 만 유일하게 같은 정보를 가진 파일이 없다. 이는 “이력서.doc.lnk” 만 다른

호스트에서 생성되었음을 확인할 수 있다.

"이력서.doc.lnk" 가 링크된 원본 실행파일은 명령 프롬프트(Command Prompt, 이하 cmd.exe) 쉘 프로그램이다. 따라서 "이력서.doc.lnk" LNK 파일을 실행한다면 'cmd.exe' 를 '/c' 옵션을 통해 "asd.exe" 를 실행하게 되는데, 이 옵션은 "asd.exe" 를 실행하고 쉘을 바로 종료한다. 이를 통해 LNK 파일의 실행 옵션으로 포함된 "asd.exe" 가 랜섬웨어 실행파일의 원본으로 추정 할 수 있다. 따라서 랜섬웨어로 추정되는 실행파일의 실행시간은 '2018-09-22 12:52:41' 로 판단된다.

2.2.3 악성코드 유입 경로 및 실행 원인 파악

악성코드 유입 흔적을 파악하기 위해 Windows 호스트에서 분석하는 아티팩트는 웹 브라우저 접속 기록, USB 연결 기록, 원격 프로그램 연결 기록 등이 있다. 이때 '첨부 1'의 내용 중 다음의 내용을 근거로 웹 브라우저 연결 흔적을 우선적으로 분석하였다.

Std Info Entry 시간 (UTC+09:00)	아티팩트 존재 경로	비고
2018-09-22 12:52:12	/Users/Kang-Lion/AppData/Local/Microsoft/Windows/Temporary Internet Files/Low/Content.IE5/index.dat	Internet Explorer 사용시 웹사이트 연결흔적 기록
2018-09-22 12:52:31	/Users/Kang-Lion/Desktop/이력서/_김유포_/이력서.doc.lnk	랜섬웨어 의심 실행파일 실행옵션을 가진 LNK
2018-09-22 12:52:36	/Users/Kang-Lion/AppData/Local/Microsoft/Windows/Temporary Internet Files/Low/Content.IE5/JP8KS9YS	Internet Explorer 사용시 생성되는 임시파일 경로
2018-09-22 12:52:47	/fdfd0bf95c1dae9258c82b1350e8/!!!SAVE YOUR FILES!!!.bmp	최초로 생성된 랜섬노트

Table 16. '이력서.doc.lnk'와 웹브라우저 연결 흔적의 연관관계

'이력서.doc.lnk'의 Std Info Entry 시간을 앞뒤로 살펴봤을 때 Internet Explorer(IE) 사용 시 참조하는 파일(index.dat)과 임시로 생성되는 파일(Content.IE5 폴더 아래에 생성되는 파일)에 대한 흔적도 같이 발견되었다. 이를 근거로 IE와 관련된 웹브라우저 관련 흔적들을 상세히 확인할 필요가 있다. IE와 관련된 흔적을 분석하기 위하여 FTK Imager를 통해 아래의 파일을 수집하였다.

경로	비고
['Kang-Lion' %localappdata%]/Microsoft/Windows/Temporary Internet Files/Content.IE/*	
['Kang-Lion' %localappdata%]/Microsoft/Windows/Temporary Internet Files/Content.IE/index.dat	웹 캐시

['Kang-Lion' %localappdata%]/Microsoft/Windows/History/History.IE/*/index.dat	웹 히스토리
['Kang-Lion' %localappdata%]/Microsoft/Windows/History/History.IE/index.dat	
['Kang-Lion' %appdata%]/Microsoft/Windows/Cookies/*	쿠키
['Kang-Lion' %appdata%]/Microsoft/Windows/Coockies/index.dat	

Table 17. IE 웹브라우저 연결 흔적 수집 경로

위의 경로에서 수집한 파일은 웹브라우저 연결 흔적 분석 도구인 WEFA로 분석하였다. 이때 연결 기록 중 악성코드 유입과 관련된 의심 흔적을 발견하였으며 해당 항목은 다음과 같다.

방문시간 (UTC+09:00)	브라우저	행위	URL	페이지 제목
2018-09-22 12:51:19	IE	페이지 로그인	https://nid.naver.com/nidlogin.lo gin?mode=form&url=http%3A% 2F%2Fwww.naver.com	네이버 : 로그인
2018-09-22 12:51:37	IE	메일	http://mail.naver.com/read/12	안녕하세요. 입사 지원합니다
2018-09-22 12:52:01	IE	파일열람	C:/Users/Kang-Lion/Desktop/0 력서/_김유포_.zip	
2018-09-22 12:52:02	IE	다운로드	http://download.mail.naver.com/fi le/download/each/?attachType =normal&mailSN=12&attachInd ex=2&virus=0&domain=mail.nav er.com	

Table 18. 웹브라우저 연결 기록 중 악성코드 유입 관련 의심 흔적

이를 통해 네이버의 메일 서비스를 접속하여 메일을 확인하였고, 이 메일에 포함되어 있던 첨부파일을 다운로드 받았음을 알 수 있다. 다운로드한 파일은 “_김유포_.zip”으로 추정되지만, 다운로드 시간과 파일 열람 시간이 역전되어 있다. 이는 브라우저가 파일 다운로드 완료 후 해당 URL에 접근을 기록했을 가능성이 있으나, 정확한 시간 관계를 파악하기 ‘첨부1’의 내용을 확인하였다. 하지만 “_김유포_.zip”의 흔적은 남아있지 않았는데, 이 경우 파일시스템의 로그를 확인하면 실제 디스크의 IO 단위로 발생한 트랜잭션 등의 정확한 시간을 파악할 수 있다. FTK Imager를 통해 확보한 파일 시스템 로그는 다음과 같다.

경로	비고
%systemdrive%/\$MFT	NTFS 내 파일 경로 정보 및 메타데이터
%systemdrive%/\$LogFile	파일시스템 트랜잭션 로그
%systemdrive%/\$Extend/\$UsnJrnl/\$J	파일의 변경과 관련된 로그

Table 19. 수집한 파일시스템 로그 정보

파일 시스템 로그를 분석도구로는 NTFS Log Tracker를 사용하였으며, 해당 내용은 ‘첨부 2. 파일

시스템 트랜잭션 로그'에서 확인할 수 있다. '첨부 2'를 통해 "__김유포__.zip" 파일이 디스크 내에 저장된 시간 정보는 다음과 같다.

Event 발생 시간 (UTC+09:00)	행위	경로	비고
2018-09-22 12:51:41	파일 생성	/Users/Kang-Lion/AppData/Local/Microsoft/Windows/Temporary Internet Files/Low/Content.IE5/0B58SG5A/__김유포__[1].zip	IE 임시생성 파일
2018-09-22 12:51:59	이름 변경	/Users/Kang-Lion/Desktop/이력서	새 폴더 → 이력서
2018-09-22 12:52:01	파일 생성	/Users/Kang-Lion/Desktop/이력서/__김유포__.zip	
2018-09-22 12:52:11	파일 생성	/Users/Kang-Lion/Desktop/이력서/_김유포__asd.exe	
2018-09-22 12:52:11	파일 생성	/Users/Kang-Lion/Desktop/이력서/_김유포__/이력서.doc.lnk	
2018-09-22 12:54:54	파일 삭제	/Users/Kang-Lion/Desktop/이력서/_김유포__asd.exe	

Table 20. 파일시스템 로그에 기록되어 있는 메일첨부 추정 파일 흔적

"__김유포__.zip"의 파일 생성 시간이 웹 브라우저 흔적과 연관 있음에 따라 위의 내용들을 종합하면 네이버 메일을 통해 이력서를 가장한 피싱 메일이 전달되었으며, 피해자는 해당 메일에 포함되어 있던 첨부파일을 자신의 PC에 다운로드하게 된다. (웹 브라우저 연결 기록을 통해 확인) 이후 압축을 해제(로컬 파일 열람 기록으로 확인)하여 "이력서.doc.lnk"를 실행(응용프로그램 사용 로그로 확인)하였고, 이는 같이 압축되어 있던 "asd.exe"의 LNK 파일이었으며, LNK의 옵션을 통해 "asd.exe"를 명령 프롬프트를 통해 실행시켰다. "asd.exe"는 랜섬웨어 파일로 추정되며, 해당 파일이 실행된 이후 파일 시스템 내 파일들을 암호화하였다. 이때 추가적으로 피해자의 착각을 유도하기 위해 "asd.exe" 파일은 숨김 파일 속성을 가지고 있을 가능성에 대해 추정할 수 있다.

2.2.4 악성코드 전파 흔적 확인

악성코드가 사설망 내에 다른 호스트를 감염시키기 위해서는 네트워크 프로토콜의 취약점을 공격하거나 원격 제어와 관련된 도구를 사용해야 한다. 하지만 이 흔적은 네트워크 트래픽과 관련된 로그를 살펴보는 것이 더 정확하며, 원격 제어 도구의 사용 기록은 프로그램 자체에서 남기는 로그를 확인해야 한다. 피해자 PC에는 원격 제어 도구로 추정된 프로그램은 설치되어 있지 않았으며, 윈도우 자체 원격 제어 도구인 Windows Desktop의 흔적을 확인하기 위해 레지스트리에서

는 활성 여부, 이벤트 로그를 통해 연결 기록을 확인하였다. 하지만 다른 호스트를 감염시키기 위한 추가 전파 흔적은 발견되지 않았다.

2.2.5 악성코드 지속 및 은닉 흔적 확인

Windows에서 프로세스가 자동으로 실행되기 위해서는 작업 스케줄러, 서비스, 자동실행 등에 등록해야 한다. 일부 악성코드 역시 침해행위를 지속하기 위해 첫 실행 후 프로그램 자기 자신을 위의 항목들에 등록한다. 따라서 지속 흔적을 확인하기 위해서는 작업 스케줄러 등록, 서비스 등록, 자동 실행 등록 흔적 등을 확인해야 한다. 이를 확인하기 위해 다음과 같은 레지스트리 키를 레지스트리 분석 도구인 REGA로 확인하였다.

경로	비고
HKEY_CURRENT_USER/Software/Microsoft/Windows/CurrentVersion/Run	지속 등록
HKEY_CURRENT_USER/Software/Microsoft/Windows/CurrentVersion/RunOnce	1회성 등록
HKEY_CURRENT_USER/Software/Microsoft/Windows NT/CurrentVersion/Windows	
HKEY_CURRENT_USER/Software/Microsoft/Windows NT/CurrentVersion/Winlogon	
HKEY_LOCAL_MACHINE/Software/Microsoft/Windows/CurrentVersion/Run	지속 등록
HKEY_LOCAL_MACHINE/Software/Microsoft/Windows/CurrentVersion/RunOnce	1회성 등록
HKEY_LOCAL_MACHINE/Software/Microsoft/Windows NT/CurrentVersion/Winlogon	
HKEY_LOCAL_MACHINE/Software/Microsoft/Windows NT/CurrentVersion/Windows	
HKEY_LOCAL_MACHINE/System/ControlSet002/Services/	서비스 등록

Table 21. 자동 실행 흔적 확인을 위해 상세 분석한 레지스트리 키 목록

위의 표에 기재된 레지스트리 키에서는 시스템 서비스를 제외하고 침해사고와 연관된 지속 흔적은 발견되지 않았다.

악성코드가 호스트에 저장이 된 이후, 피해자 PC에서 악성 프로그램으로 발각되지 않기 위해 알려진 시스템 프로그램 이름과 비슷하게 위장하거나(파일 이름 은닉), 알려진 시스템 프로그램 이름과 동일한 파일명 인 채로 시스템 관련 경로에 저장(파일 경로 은닉), 사용하지 않은 영역 (MBR/VBR 슬랙, 비활당 영역 등)에 저장하는 경우 등이 있다.

따라서 전자인 파일 이름 은닉, 파일 경로 은닉을 확인하기 위해 FTK Imager의 Export File Hash List 기능을 통해 파일 시스템 내 전체 파일의 해시값을 확보하였다. 이는 '첨부 3'을 통해 확인 가능하다. 이 해시 목록 중 exe 파일의 MD5 해시값에 대해서 악성코드 OSINT 서비스인 'VirusTotal'에 질의하여 기존에 알려진 악성 프로그램이 있는지 확인하였다. 이 질의 과정을 자동

화 한 코드는 '첨부 4. query_exe_md5_to_vt.py'로 첨부하였으며, 질의 결과로 의심되는 프로그램은 발견되지 않았다. 또한 후자인 MBR/VBR 슬랙에는 PE 포맷을 가진 파일은 발견되지 않았다.

2.3 증거 분석 결과

본 침해사고의 경우 사용자의 부주의로 인해 발생되었을 가능성이 높다. LNK 파일의 타겟 파일인 악성코드 "asd.exe"가 만약 숨김 파일로 되어 있더라도 압축 해제 과정에서 압축 프로그램을 통해 실행 파일이 압축되어 있는 것을 확인할 수 있었을 것이다. 이러한 LNK 파일을 이용한 악성코드를 실행하는 방법은 최근에 유포되고 있는 "GandCrab" 랜섬웨어와 "VenusLocker" 랜섬웨어에서 주로 사용되고 있다.

최근의 메일 서비스는 압축한 아카이브 파일 내에 실행 파일과 관련된 확장자가 있다면 검출하여 전송을 차단한다. 공격자는 이를 우회하기 위해 압축파일을 암호화하여 메일 전송 서비스의 검출 서비스를 우회하였다. 이와 같이, 피싱 메일을 통한 공격은 증가하고 있는 추세이다. 기술 난이도가 높지 않고 기업 내 실무자를 직접 공격하기 가장 적합한 방법이기 때문이다. 따라서 사용자는 보안 의식을 좀 더 높게 가질 필요가 있으며 의심되는 이메일, 특히 첨부파일은 되도록 사용하는 PC에서 직접 열지 않고 백신을 이용한 악성코드 검사 후 사용해야한다. 기업 차원에서는 이메일 보안 솔루션을 도입하여 메일 내의 첨부 파일을 선제적으로 검사하고, 사용자에게는 메일 내용을 해석해 이미지 형태로 제공하는 등의 대응이 필요하다. 솔루션 도입 시, 사내 사설 네트워크와 외부 네트워크 중간을 솔루션이 연계하게 되며, 만일의 사고 시 회사 내부의 네트워크까지 악성코드가 전파되는 것을 차단할 수 있다.

사고의 피해 규모

랜섬웨어 의해 감염된 파일의 개수는 총 2701개이다. 이는 '첨부 3. 파일 시스템 내 파일 해시 목록'에서 'sVn' 확장자를 가진 파일의 개수를 계산한 것이며, 소스코드를 통해 자동화하였다. 소스코드는 '첨부 5. get_encrypted_file_lists.py'를 통해 확인할 수 있다. 또한 감염된 파일의 목록은 '첨부 6. 랜섬웨어 감염 파일 목록'으로 첨부하였다.

이후 진행 사항

악성코드의 정확한 동작원리와 파일 복호화 여부를 확인하기 위해서는 악성코드의 원본 확보가 필요하다. 하지만 파일이 파일 시스템 내에서 삭제되었으므로 원본을 복구하여야 한다. 악성코드 원본이 확보가 되면 이후 암호화된 파일에 대한 복호화 가능 여부에 대해서 확인해야 한다. 이후 보고서는 위의 두 가능성에 대해 작성하였다.

3. ZIP 파일 복구

3.1 개요

이미지 분석을 통해 “_김유포_.zip” 으로부터 “이력서.doc.lnk” 와 “asd.exe” 라는 파일이 압축 해제된 것을 확인하였다. “이력서.doc.lnk” 의 경우, 본래 파일이 존재하였던 위치인 “/Users/Kang-Lion/Desktop/이력서/_김유포_” 에 존재하여 그 내용을 확인한 결과, System32의 cmd.exe 를 이용해 “asd.exe” 를 실행시키는 것을 확인할 수 있었다. “asd.exe”가 한 행위를 파악하기 위해 파일 시스템 상의 “asd.exe”와 “_김유포_.zip” 를 찾아보려고 시도하였으나, 삭제되어 있어서 확인할 수 없었다. 삭제된 “asd.exe”와 “_김유포_.zip” 를 복구하기 위해 주어진 디스크 덤프에서 ‘비활당 영역’, ‘블룸 쉐도우 카피’, ‘페이지 파일’을 대상으로 카빙을 시도하였다.

3.2 파일 카빙 시도

3.2.1 비활당영역에서의 카빙

NTFS 파일 시스템에서 파일 삭제를 할 경우, 데이터를 실질적으로 지우기 이전에 MFT 엔트리의 Attribute를 Deleted로 변경하거나 제거되기도 한다. 효율적인 파일 시스템의 운영을 위해 이후 새로운 데이터가 유입되면 Deleted로 변경된 Attribute를 가진 클러스터도 덮어 쓰인다[5]. 이와 같이 비활당 영역 중 아직 덮어 쓰이지 않은 영역의 경우, 기존 파일의 데이터를 보존하고 있기 때문에 카빙을 통해 파일을 복구할 수 있다.

Winhex를 이용하여 다음과 같이 NTFS 파일 시스템의 비활당 영역을 확인할 수 있다.

\$Volume		0 B	2018-09-20 20:2...	2018-09-20 20:2...	2018-09-20 20:2...	ISH	
autoexec.bat	bat	24 B	2009-07-14 11:0...	2009-06-11 06:4...	2009-07-14 11:0...	A	6311230
bootmgr		375 KB	2018-09-20 20:2...	2010-11-21 06:2...	2018-09-20 20:2...	SHRA	111898...
BOOTSECT.BAK	BAK	8.0 KB	2018-09-20 20:2...	2018-09-20 20:2...	2018-09-20 20:2...	SHRA	112141...
config.sys	sys	10 B	2009-07-14 11:0...	2009-06-11 06:4...	2009-07-14 11:0...	A	6311236
pagefile.sys	sys	2.0 GB	2018-09-20 20:2...	2018-09-22 17:4...	2018-09-20 20:2...	SHA	143591...
Free space		4.0 GB					4102656
Idle space		211 MB					432807...
Misc non-resident attributes		36.0 KB					3838512
Volume slack		4.0 KB					629084...

Fig 5. Winhex를 이용한 비활당영역 확인

Winhex에서 확인할 수 있는 “Free Space”와 “Idle space”는 모두 NTFS Filesystem의 비활당 영역이다. “Free Space”的 경우, 파일 시스템에서 사용하지 않고 있다고 mark된 영역을 의미하고, “Idle Space”的 경우, 파일 시스템에서 사용하고는 있는데 정확한 할당이 되지 않은 부분을 의미한다[6]. 해당 영역을 Winhex의 “Recover/Copy” 기능을 이용하여 추출하였다. 추출한 영역으로부터 생성된 파일의 세부사항은 다음과 같다.

File Name	Free Space
File Size	3.96GB (4,261,527,55 byte)
MD5	F9E383D4A6C940919CD818C39BB03FDC
SHA-1	4F255587B7333B0B37BE9FACFFCB64AA90080D07

Table 22. "Free Space"의 세부 사항

File Name	Idle Space
File Size	211MB (221,409,280 byte)
MD5	B0432572F4DE307862B1C9133ACE06E4
SHA-1	AC65ED455DB31654D0D963C885B32AC18593D3CE

Table 23. "Idle Space"의 세부 사항

추출한 비활당 영역은 R-Studio를 이용해 카빙을 시도하였다. R-Studio는 파일 시그니처를 기반으로 데이터 복구와 카빙을 위해 주로 사용되는 도구이다[7]. 카빙을 통해 얻으려는 파일이 "asd.exe"라는 이름을 가진 실행 파일이나 "_김유포_.zip"라는 이름을 가진 ZIP 압축 파일이므로 추출하는 확장자를 ZIP과 EXE로 설정하고 카빙을 시도하였다.

"Free Space"을 대상으로 한 카빙 결과는 다음과 같다.

Name	Size	Created	Modified	Accessed
conhost.exe	271360 ...	2018-06-09 AM 12:...		
dnsocacheugc.exe	30720 B...	2018-06-09 AM 12:...		
wordpad.exe	4247040...	2018-06-09 AM 12:...		
tzupd.exe	40448 B...	2018-06-21 AM 11:...		
msisexec.exe	73216 B...	2018-06-28 AM 12:...		
consent.exe	105152 ...	2018-06-28 AM 12:...		
cleanupintcache.exe	14336 B...	2018-06-28 AM 12:...		
smss.exe	69632 B...	2018-08-11 AM 12:...		
lsass.exe	22016 B...	2018-08-11 AM 12:...		
ntkrnlmp.exe	3961440...	2018-08-11 AM 12:...		
ntkrpamp.exe	4054192...	2018-08-11 AM 12:...		
rstrui.exe	262656 ...	2018-08-11 AM 12:...		
auditpol.exe	50176 B...	2018-08-11 AM 12:...		
appidcertstorecheck.exe	16896 B...	2018-08-11 AM 12:...		
appidpolicyconverter.exe	97792 B...	2018-08-11 AM 12:...		
mrtstub.exe	4724008...	2018-08-31 AM 8:2...		

Fig 6. "Free space"에서 발견된 exe 실행파일

"Free Space"에서 발견한 EXE 파일의 경우, 카빙되는 파일 이름을 확인할 수 있으나, 추출 대상인 "asd.exe"는 발견되지 않았다.



Fig 7. "Free space"에서 발견된 ZIP 압축파일

"Free Space"에서 발견한 ZIP 파일의 경우, 카빙되는 원래 파일 이름을 확인할 수 없어서 별도로 추출하여 해당 파일의 정보를 확인하였다. 추출한 ZIP 파일의 파일 내용을 확인한 결과, 아래의 사진과 같이 토익 시험 관련 mp3 파일이 존재하였다. 그러나 복구하려는 대상인 "asd.exe"와 "__김유포_" 파일은 발견할 수 없었다.

이름	압축된 크기	유형	변경된 날짜
..		로컬 디스크	
RT_004_92-94_Am.mp3	1,972,571	MP3 파일	2013-01-16 9
RT_004_89-91_Br.mp3	2,218,403	MP3 파일	2013-01-16 9
RT_004_98-100_Br.mp3	2,289,643	MP3 파일	2013-01-16 9
RT_004_95-97_Am.mp3	2,571,904	MP3 파일	2013-01-16 9

Fig 8. "Free space"에서 발견된 ZIP 파일의 내용

"Idle Space"에서도 동일한 방법으로 카빙을 시도하였다. "Free Space" 영역과는 다르게 ZIP 파일과 관련된 것은 확인할 수 없었으나 "Free Space" 영역과 마찬가지로 EXE 파일은 파일명과 함께 확인할 수 있었다.

Name	Size	Created	Modified	Accessed
auditpol.exe	50176 B...	2014-04-12 AM 10:...		
NETFXSBS10.exe	87704 B...	2014-05-02 PM 4:...		
TabTip.exe	181760 ...	2014-06-18 AM 10:...		
osk.exe	646144 ...	2014-06-18 AM 10:...		
rstrui.exe	262656 ...	2015-07-15 AM 10:...		
appidpolicyconverter.exe	97792 B...	2015-07-15 AM 11:...		
lsass.exe	22528 B...	2015-07-23 AM 1:3...		
auditpol.exe	50176 B...	2015-07-23 AM 1:4...		
lsass.exe	22528 B...	2015-07-23 AM 5:1...		
rstrui.exe	262656 ...	2015-07-23 AM 5:2...		
auditpol.exe	50176 B...	2015-07-23 AM 5:3...		
sdbinst.exe	20992 B...	2015-10-30 AM 1:3...		
PDIALOG.exe	48640 B...	2016-01-07 AM 2:5...		
GWXWU.exe	78960 B...	2016-08-26 AM 5:0...		
mrtstub.exe	3947685...	2018-01-31 PM 1:2...		
PCIClearStaleCache.exe	30248 B...	2018-06-30 AM 12:...		

Fig 9. "Idle space"에서 발견한 EXE 실행 파일

"Idle Space"로부터 추출 가능한 EXE 파일의 이름을 확인해본 결과, 카빙의 목표가 되는 "asd.exe"라는 이름의 파일은 발견할 수 없었다.

윈도우의 기능을 이용해 삭제한 파일의 경우, R-Studio를 이용하여 복구할 수 경우가 있다. 이를 파악하기 위해서는 R-Studio의 스캔 대상을 디스크 이미지 파일 전체로 설정하고 분석을 진행하였다. 분석을 통해 "asd.exe"가 실제로 삭제된 흔적을 발견할 수 있었다.

Folders		Contents				
		Name	Size	Created	Modified	Accessed
▼	Partition1	☒ _김유포_[1].zip	131201 Bytes			
▼	Archive	☐ AboutBox.zip	35969 Bytes	2009-06...	2009-06-02 PM 9:47:18	2018-09...
☐	7-Zip Archive (.7z)	☐ AppConfig.zip	593 Bytes	2009-06...	2009-06-02 PM 9:47:18	2018-09...
☐	Microsoft Cabinet Archive (.cab)	☐ AppConfigInternal.zip	621 Bytes	2009-06...	2009-06-02 PM 9:47:18	2018-09...
☐	Quantum Archive (.pak)	☐ AppConfigurationInternal.zip	1249 Bytes	2009-06...	2009-06-02 PM 9:47:20	2018-09...
☐	Windows Installer Package (.msi)	☐ AssemblyInfo.zip	1291 Bytes	2009-06...	2009-06-02 PM 9:47:18	2018-09...
☒	ZIP Archive (.zip)	☐ ARREXE	20992 Bytes	2009-07...	2009-07-14 AM 10:14:12	2009-07...
▼	Executable, Library, DLL	☐ ARREXE	20992 Bytes	2009-07...	2009-07-14 AM 10:14:12	2009-07...
☐	Windows Device Driver (.sys)	☒ asd.exe	215040 Bytes			
☐	Windows DLL (.dll)	☐ aspnet_compiler.exe	36864 Bytes	2009-07...	2009-06-11 AM 6:22:45	2009-07...
☐	Windows Executable (.exe)	☐ aspnet_compiler.exe	36864 Bytes	2009-07...	2009-06-11 AM 6:22:45	2009-07...
☐	Windows OCX File (.ocx)					

Fig 10. 복구 대상인 "_김유포_[1].zip"

하지만 해당 파일의 내용을 확인한 결과, "asd.exe"의 경우는 0x00으로 쓰였고, "_김유포_[1].zip"의 경우는 Zip 파일 대신에 png 파일의 시그니처를 발견할 수 있었다.

Sector 0 (Parent: Partition1 Record: 40040760)	
0:	89 50 4E 47 0D 0A 1A 0A - 00 00 00 0D 49 48 44 52
10:	00 00 00 2B 00 00 00 18 - 08 02 00 00 00 EC D1 93
20:	94 00 00 00 09 70 48 59 - 73 00 00 0B 13 00 00 0B
30:	13 01 00 9A 9C 18 00 00 - 39 A6 69 54 58 74 58 4D
40:	4C 3A 63 6F 6D 2E 61 64 - 6F 62 65 2E 78 6D 70 00
50:	00 00 00 00 3C 3F 78 70 - 61 63 6B 65 74 20 62 65
60:	67 69 6E 3D 22 EF BB BF - 22 20 69 64 3D 22 57 35
70:	gin="□? id="W5 M0MpCehiHzreSzNT
80:	4D 30 4D 70 43 65 68 69 - 48 7A 72 65 53 7A 4E 54 czkc9d"?>.<x:xmp
90:	63 7A 6B 63 39 64 22 3F - 3E 0A 3C 78 3A 78 6D 70 meta xmlns:x="ad
A0:	6D 65 74 61 20 78 6D 6C - 6E 73 3A 78 3D 22 61 64 obe:ns:meta/" x:
B0:	6F 62 65 3A 6E 73 3A 6D - 65 74 61 2F 22 20 78 3A xmptk="Adobe XMP
C0:	78 6D 70 74 6B 3D 22 41 - 64 6F 62 65 20 58 4D 50 Core 5.6-c132 7
D0:	39 2E 31 35 39 32 38 34 - 2C 20 32 30 31 36 2F 30 9.159284, 2016/0 .□.□.□.

Fig 11. "_김유포_[1].zip"에서 발견된 PNG 시그니처

해당 PNG 파일의 내용은 다음과 같았다.



Fig 12. PNG 파일 내용

실제로 \$Logfile을 이용해 확인해본 결과 해당 “_김유포_[1].zip”이 있던 클러스터에 “nsd172615885[1].png” 하는 파일이 덮어 쓰였음을 확인할 수 있다.

LSN	Event	Detail	FileName
필터	필터	%5005095%	필터
269295543	Writing Content of Non-Resident File	Cluster Number : 5005095(14)	Cab93B5.tmp
269564885	Writing Content of Non-Resident File	Cluster Number : 5005095(14)	CabB380.tmp
270301387	Writing Content of Non-Resident File	Cluster Number : 5005095(33)	_김유포_[1].zip
270440380	Writing Content of Non-Resident File	Cluster Number : 5005095(5)	nsd172615885[1].png

Fig 13. \$logfile Cluster 로그를 통해 확인한 PNG 파일

3.2.2 볼륨 쉐도우 카피(VSC)를 이용한 카빙

볼륨 쉐도우 카피는 윈도우 7 이후 시스템 복원을 위해 도입되었다. 볼륨 쉐도우 카피를 이용하면 볼륨 쉐도우 카피를 이용해 저장되었던 특정 시점의 컴퓨터의 상황으로 복구할 수 있는 것이 대표적인 특징이다[8]. 볼륨 쉐도우 카피는 ShadowExplorer라는 프로그램을 이용하여 확인할 수 있다. 볼륨 쉐도우 카피가 존재하는지 확인을 위해 qemu-img를 이용해 제공받은 이미지 파일을 VMDK(Vmware Disk) 형식으로 변환하였다. Qemu는 대표적인 가상화 관련 도구이며[9], 변환을 위해 사용한 명령어는 아래와 같다.

```
Qemu-img.exe convert "2018 Digital Forensic Challenge.001" -O vmdk "2018KDFS.vmdk"
```

VMDK 파일이 생성되면 Vmware Workstation을 이용하여 라이브 부팅을 시도할 수 있다. 라이브 부팅을 위해서는 새로운 가상 머신을 생성하여야 한다. 가상 머신의 생성을 위해 필요한 옵션들은 아래와 같다.

- Custom (advanced)
- I will install the operating system later.
- Other (Operating System)
- Use an Existing Virtual Disk

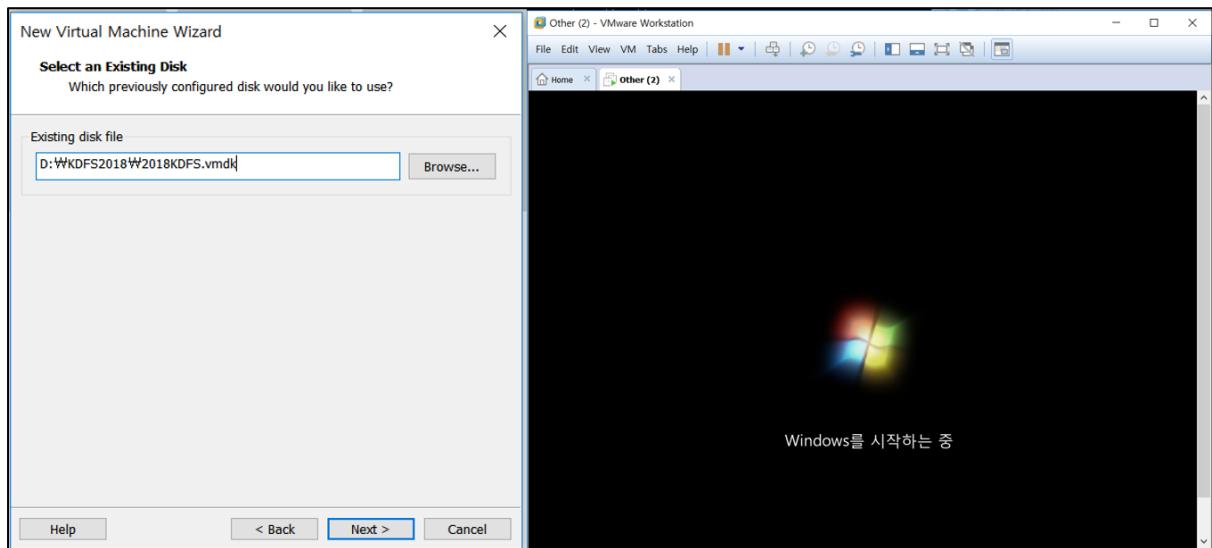


Fig 14. 이미지 라이브 부팅

이후 좌측 그림과 같이 VMDK 파일의 경로를 지정해주고, 생성한 가상 머신을 실행하면 우측 그림과 같이 Windows가 부팅되는 것을 확인할 수 있다. 부팅이 완료되면 다음과 같은 랜섬 노트가 적힌 바탕화면을 확인할 수 있다. 랜섬 노트에는 딥웹에 접속하기 위한 토르 브라우저 설치 방법과 복호화 키를 받기 위한 딥 웹 주소와 아이디를 알려주고 있다.

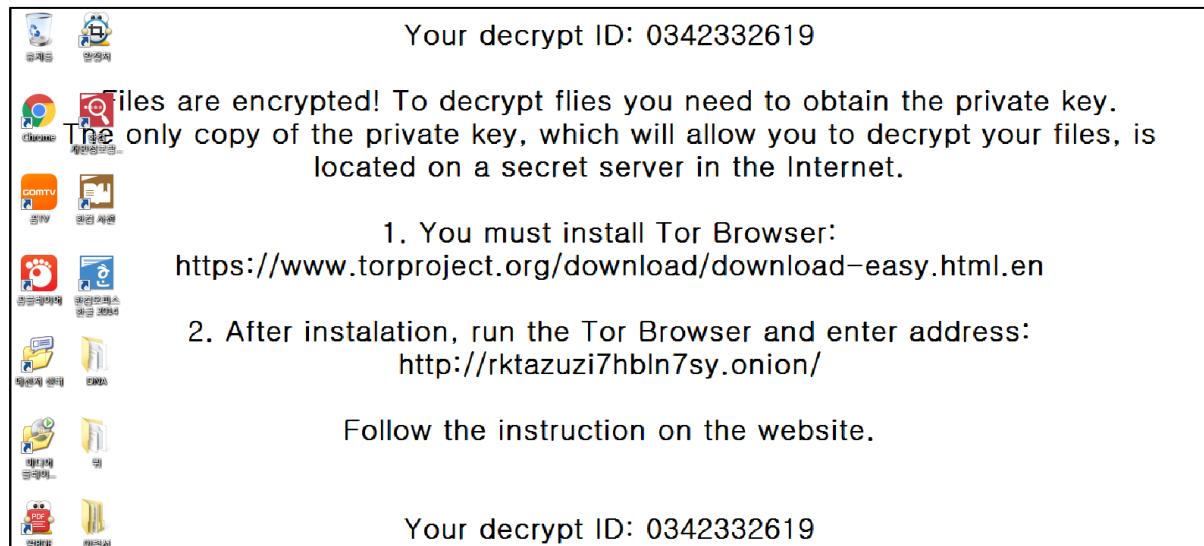


Fig 15. 라이브 부팅을 통해 발견한 바탕화면의 랜섬노트

볼륨 쉐도우 카피의 목록을 확인하기 위해서는 "vssadmin"이라는 명령어를 활용하여 확인할 수 있다. 관리자 권한으로 상승한 "cmd.exe"를 이용해 볼륨 쉐도우 카피를 확인하기 위한 명령어는 다음과 같다.

vssadmin List Shadows

명령어 실행 결과는 아래의 그림과 같이 볼륨 쉐도우 카피를 확인할 수 없었다.

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>vssadmin List Shadows
vssadmin 1.1 - 볼륨 샘플 복사본 서비스 관리 명령줄 도구
(C) Copyright 2001-2005 Microsoft Corp.

쿼리를 만족하는 항목이 없습니다.
```

Fig 16. vssadmin 명령어 실행 결과

ShadowExplorer를 이용한 확인 결과도 아래의 그림과 같이 볼륨 쉐도우 카피가 존재하지 않는 것으로 확인되었다.

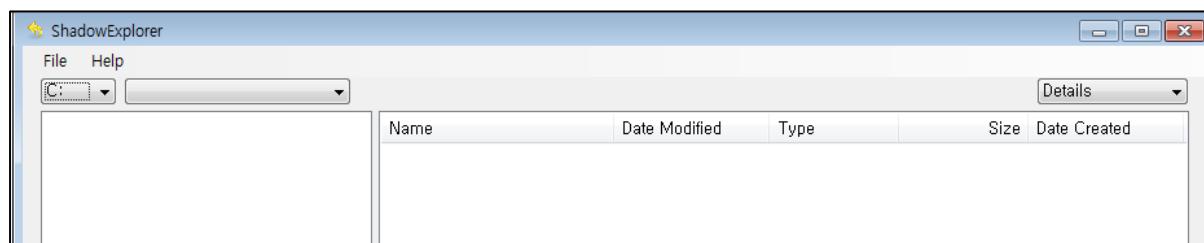


Fig 17. ShadowExplorer 결과

주어진 시스템에는 볼륨 쉐도우 카피가 남아있지 않은 것으로 확인되었다. 그러나, 악성코드 실행에 따라 볼륨 쉐도우 카피가 지워졌을 가능성이 존재한다. 이 경우, 볼륨 쉐도우 카피의 카빙을 통해 볼륨 쉐도우 카피를 복구할 수 있다[10]. 볼륨 쉐도우 카피를 카빙하기 위해 vss_carver.py라는 도구를 사용하였다. vss_carver.py를 사용하기 위해서는 NTFS 파일 시스템의 시작 오프셋을 알아야한다. NTFS 시작 오프셋은 Winhex를 통해 다음의 그림과 같이 파악할 수 있다.

2018 Digital For... 2018 Digital For..., P1																		
Partitioning style: MBR																		
Name	Ext.	Size	Created	Modified	Accessed	Attr.	1st sector											
Partition 1	NT...	30.0 GB					2048											
Partition gap		38.0 KB					629104...											
Start sectors		1.0 MB					0											
Unpartitionable space		2.0 MB					629105...											
[...]2018 Digital Forensic Challer		Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Default Edit Mode	original	00001048576	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00
State:		00001048592	00	00	00	00	00	F8	00	00	3F	00	FF	00	00	08	00	00
Undo level:	0	00001048608	00	00	00	00	80	00	80	00	FF	E7	BF	03	00	00	00	00
Undo reverses:	n/a	00001048624	00	00	0C	00	00	00	00	00	02	00	00	00	00	00	00	00
Total capacity:	30.0 GB	00001048656	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	68	C0	07
	32,212,254,720 bytes	00001048672	1F	1E	68	66	00	CB	88	16	0E	00	66	81	3E	03	00	4E
Bytes per sector:	512	00001048688	54	46	53	75	15	B4	41	BB	AA	55	CD	13	72	0C	81	FB
Surplus sectors at end:	4020	00001048704	55	AA	75	06	F7	C1	01	00	75	03	E9	DD	00	1E	83	EC
Partition:	1	00001048720	18	68	1A	00	B4	48	8A	16	0E	00	8B	F4	16	1F	CD	13
Relative sector No.:	0	00001048736	9F	83	C4	18	9E	58	1F	72	E1	3B	06	0B	00	75	DB	A3
		00001048752	0F	00	C1	2E	0F	00	04	1E	5A	33	DB	B9	00	20	2B	C8

Fig 18. NTFS 볼륨의 시작 오프셋 찾기

NTFS 파티션이 시작되는 VBR의 시그니처를 확인한 결과, NTFS 파티션은 00001048576부터 시작됨을 확인할 수 있었다. 이를 이용해 vss_carver.py에 적용할 명령어는 다음과 같다.

```
Python vss_carver.py -o 1048576 -i $image_path -c $catalog_path -s $store_path
```

명령어 수행 결과는 아래의 사진과 같다.

```
sin90@sinsin9090:~/Desktop/KDFS2018/vss_carver$ python vss_carver.py -o 1048576 -i '/home/sin90/Desktop/KDFS2018/2018 Digital Forensic Challenge.001' -c '/home/sin90/Desktop/KDFS2018/vss_carver/catalog_file' -s '/home/sin90/Desktop/KDFS2018/vss_carver/store_file'
=====
Stage 1: Checking if VSS is enabled.
Volume size: 0x77fd00000
Found VSS volume header.
0x1e00: b'6b87083876c1484eb7ae04046e6cc752'
Catalog offset: 0x0
=====
Stage 2: Reading catalog from disk image.
VSS snapshot was enabled. But all snapshots were deleted.
=====
Stage 3: Carving data blocks.
=====
Stage 4: Grouping store blocks by VSS snapshot.
=====
Stage 5: Checking next block offset lists.
=====
Stage 6: Deduplicating carved catalog entries.
=====
Stage 7: Writing store file.
=====
Stage 8: Writing catalog file.
```

Fig 19. vss_carver.py 의 명령어 수행 결과

위의 사진과 같이 vss_carver.py를 통해 볼륨 쉐도우 카피를 마운트하기 위한 Store 파일과 Catalog 파일이 생성된다. 볼륨 쉐도우 카피를 마운트 하기 이전에 카빙된 볼륨 쉐도우 카피 파일의 리스트를 확인하여야 한다. 리스트 확인에는 생성된 Catalog 파일을 이용하면, 필요한 명령어는 다음과 같다.

```
Python vss_catalog_manipulator.py list -c $catalog_path
```

```
sin90@sinsin9090:~/Desktop/KDFS2018/vss_carver$ python vss_catalog_manipulator.py list catalog_file
sin90@sinsin9090:~/Desktop/KDFS2018/vss_carver$ od catalog_file
0000000 103553 034010 140566 047110 127267 002004 066156 051307
0000020 000001 000000 000002 000000 000000 000000 000000 000000
0000040 000000 000000 000000 000000 040000 000000 000000 000000
0000060 000000 000000 000000 000000 000000 000000 000000 000000
*
0040000 103553 034010 140566 047110 127267 002004 066156 051307
0040020 000001 000000 000002 000000 040000 000000 000000 000000
0040040 040000 000000 000000 000000 100000 000000 000000 000000
0040060 000000 000000 000000 000000 000000 000000 000000 000000
*
0100000 103553 034010 140566 047110 127267 002004 066156 051307
0100020 000001 000000 000002 000000 100000 000000 000000 000000
0100040 100000 000000 000000 000000 140000 000000 000000 000000
0100060 000000 000000 000000 000000 000000 000000 000000 000000
*
0140000 103553 034010 140566 047110 127267 002004 066156 051307
0140020 000001 000000 000002 000000 140000 000000 000000 000000
0140040 140000 000000 000000 000000 000000 000000 000000 000000
0140060 000000 000000 000000 000000 000000 000000 000000 000000
*
0200000
```

Fig 20. Catalog file 내용

"vss_catalog_manipulator.py"를 이용한 리스트 확인 결과, 카탈로그 내에 볼륨 쉐도우 카피의 리스트가 존재하지 않았다. 상세한 분석을 위해 od를 이용해 Hex 값을 살펴보았으나, 의미있는 데 이터를 발견하지 못하고 대부분 0x00으로 씌워져 있음을 확인하였다.

3.2.3 페이지파일을 이용한 카빙

페이지 파일은 윈도우 운영체제에서 메모리 관리를 위해 생성하는 프로그램이다. 페이징(Paging)이라는 방법을 이용해 메모리의 일부를 저장하였다가 필요할 때 사용하는 방법을 사용한다[11]. 윈도우 운영체제에서는 Windows 7 이후로 C:/ 이하의 "pagefile.sys" 파일을 이용해 확인할 수 있다. 카빙에 사용한 "pagefile.sys"의 세부사항은 다음과 같다.

File Name	Pagefile.sys
File Size	1.99GB (2,146,951,168 byte)
MD5	10576a16291876ca84bb8c76f7187f9e
SHA-1	69D5094172CC962ACEC44FCEE4DB19204A556009

Table 24. "pagefile.sys" 파일의 세부사항

"pagefile.sys"의 경우도 비활당 영역에서 ZIP 파일과 EXE 파일을 카빙하는 방법과 동일하게 R-Studio를 이용하였다. R-Studio를 이용한 카빙 결과는 다음과 같다.

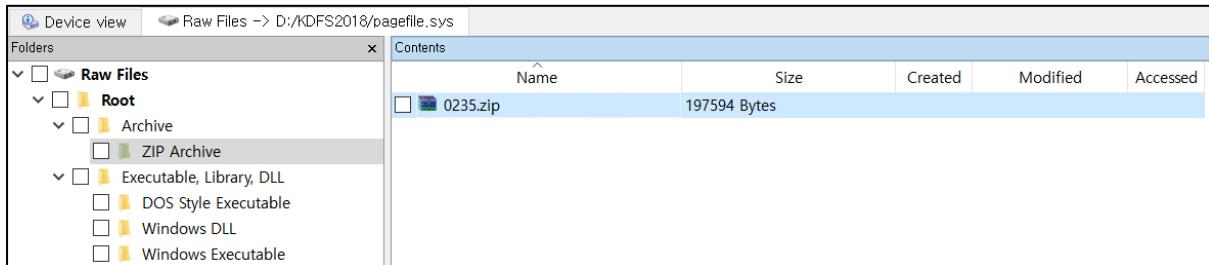


Fig 21. "Pagefile.sys"에서 발견한 ZIP 압축 파일

해당 파일은 파일 이름이 복구되지 않아서 추출을 통해 ZIP 파일의 내용을 확인을 시도하였다. 그러나 추출된 파일은 카빙이 제대로 되지 않아, 내용을 확인할 수 없었다.

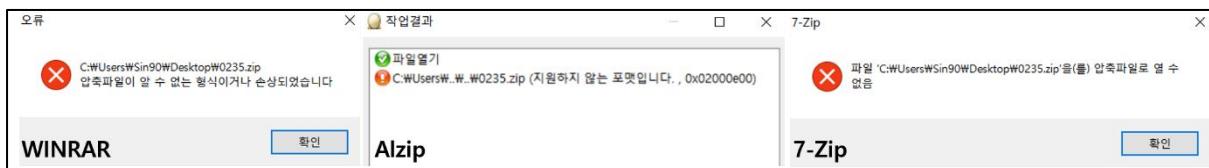


Fig 22. 카빙된 ZIP 파일 확인 결과

"pagefile.sys"안에 존재하는 EXE 파일도 동일한 방법으로 복구를 시도하였다.

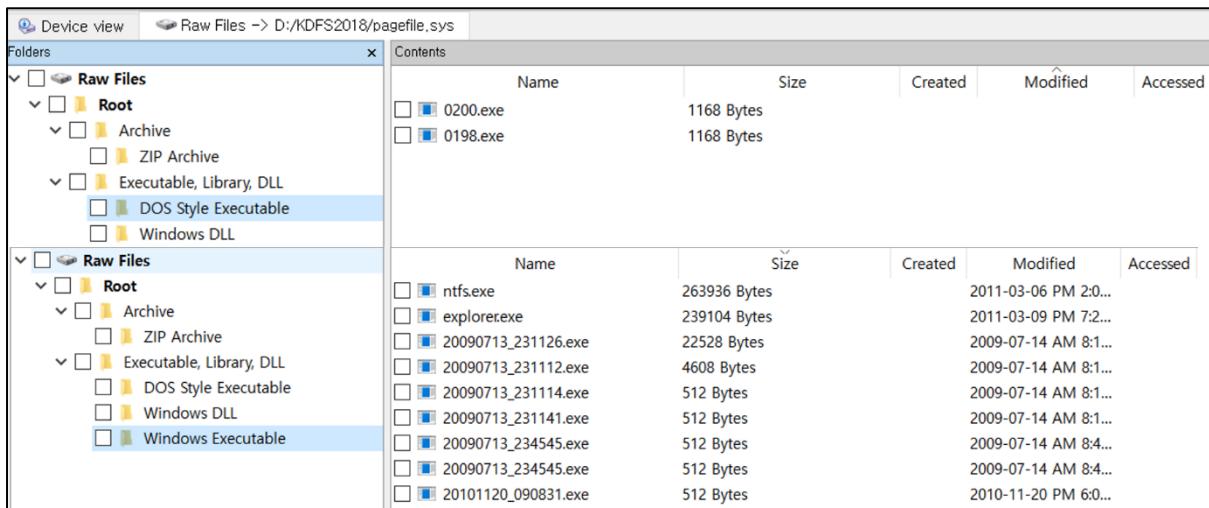


Fig 23. "Pagefile.sys"내의 EXE 파일 확인

EXE 파일의 경우 파일 이름까지 복구할 수 있는 경우와 그렇지 못해서 "Modified" 시간을 이용해 파일 이름이 결정된 경우가 존재한다. 파일 이름의 경우, 복구 대상인 "asd.exe"는 발견할 수 없고, 대부분이 512~1168 byte로 파일 크기가 작아 랜섬웨어 행위를 하기에 어려우며, 특히 "Modified" 시간의 경우 2009년으로 설정되어 있어 랜섬웨어로 인한 침해 행위가 일어난 2018년과 시간적 거리가 있다. 이를 근거로 "pagefile.sys"에서 복구할 수 있는 exe 파일은 본 침해사고

의 분석 대상인 랜섬웨어 악성코드와 관련이 적은 것으로 판단하였다.

비활당 영역, 볼륨 쉐도우 카피, 페이지 파일을 이용하여 파일 복구를 시도한 결과, 압축 파일이나 실행 파일 원본을 얻는 것은 어려운 것으로 판명되었다. 이후 분석에서는 실행 파일 원본을 얻는 것 대신, 원본 파일의 정보를 담고 있는 파일 메타데이터를 복구 대상으로 진행하였다.

3.3 파일 메타데이터 카빙

3.3.1 Yara를 이용한 메타데이터 탐지

1.2 를 통해 비활당영역, 볼륨 쉐도우 카피, 페이지 파일을 이용하여 파일 복구를 시도하였다. 그러나 원본 파일 복구는 어려운 일이라고 판단하였다. 이에 복구하려는 대상인 “asd.exe”와 “_김유포_.zip”에 대한 원본 파일 복구 대신에, 원본 파일의 정보를 가지고 있는 메타데이터를 복구 대상으로 하였다.

“asd.exe”的 경우 PE 구조를 가지는 실행 파일이고, “_김유포_.zip”은 PK 구조를 가진 압축 파일이다. PE 구조와 PK 구조는 각각 정의된 문서에 따라 일정한 시그니쳐와 포맷을 가지고 있다.

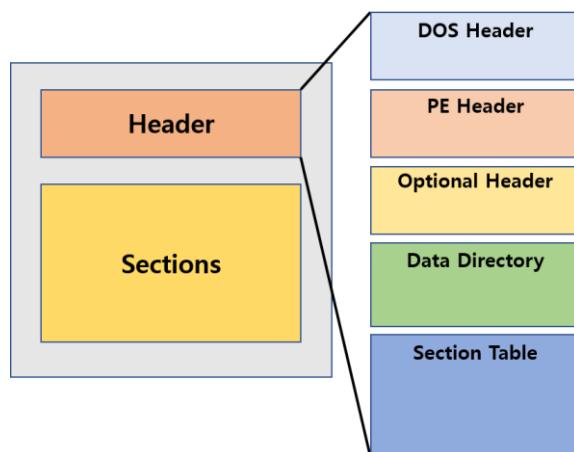


Fig 24. EXE 실행파일의 헤더 구조

PE 구조의 경우, PE Header 부분에 실행 파일과 관련된 정보를 담고 있는 메타데이터가 존재한다 [12]. PE Header에서 추출 가능한 정보는 다음이 대표적이다.

- 실행 파일 내의 Section의 개수
- 파일 실행을 위한 Entry point의 Offset 주소
- Header의 Size
- 실행을 위해 메모리에 필요한 공간

그러나 PE 구조로부터 얻을 수 있는 정보는 많은 EXE 파일의 메타데이터 중에서, 본 침해사고 분석에 필요한 “asd.exe”라는 실행 파일의 메타데이터라고 특정할 수 없다는 단점이 있다. 이에 파일 메타데이터 카빙 대상은 ZIP 파일로, “_김유포_.zip”이라는 파일을 특정하는 정보를 카빙하는 것을 목표로 설정하였다.

ZIP 파일의 경우 다음의 구조를 가지고 있다.

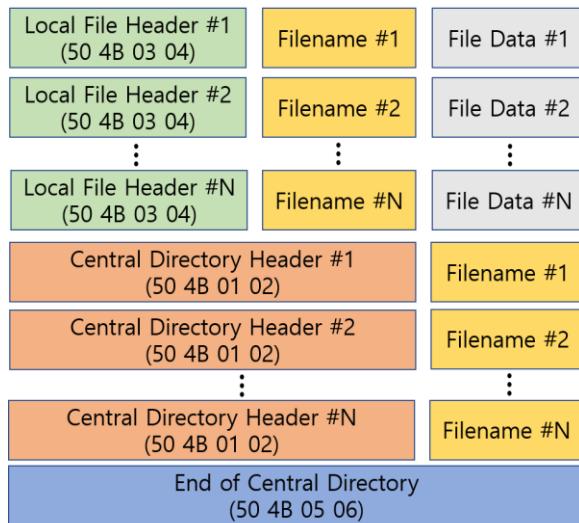


Fig 25. ZIP 압축 파일의 구조

ZIP 파일의 경우, 함께 압축되어 있는 파일 이름을 Local File Header와 Central Directory Header에 각각 가지고 있다. Local File Header가 손상되면 Central Directory Header와 End of Central Directory를 통해 복구할 수 있고, Central Directory Header가 손상되면 Local File Header의 정보를 기반으로 복구할 수 있다.

“_김유포_.zip”的 메타데이터를 확인하기 위해서는 YARA를 사용하여 탐지하였다. YARA는 패턴 매칭을 도와주는 도구로서, 파일 내부에 작성한 Rule에 따라 매칭되는 여부를 나타내어 준다.

LSN	날짜	Event	etc	FileName	FullPath
필터	필터	필터	필터	%김유포%	
270301387		Writing Content o...	...	_김유포__[1].zip	\\Users\\Kang-Lion\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet...
270472200		Move(Before)		_김유포__.zip	\\Users\\Kang-Lion\\Desktop\\미력서_김유포__.zip
270493074		Move(After)		asd.exe	\\Users\\Kang-Lion\\Desktop\\미력서_김유포__.asd.exe
270493669		Move(After)		미력서.doc.lnk	\\Users\\Kang-Lion\\Desktop\\미력서_김유포__.미력서.doc.lnk
270355922		File Deletion		_김유포__.zip	\\Users\\Kang-Lion\\Desktop\\미력서_김유포__.zip

Fig 26. “_김유포_.zip”과 관련된 파일시스템 로그 확인

“_김유포_.zip”的 경우 압축 해제 시, “_김유포_” 폴더에 “asd.exe”와 “미력서.doc.lnk” 파일이 생성되는 것이 특징이다. 이는 “_김유포_.zip” 안에 “_김유포_” 폴더가 존재하고, 해당 폴더 내부에 “asd.exe”와 “미력서.doc.lnk”가 압축되어 있음을 의미한다. ZIP 파일 구조 상, Local File Header에 “_김유포_/asd.exe”와 “_김유포_/미력서.doc.lnk”라는 파일 이름이 있고, Central Directory Header에 “_김유포_/asd.exe”와 “_김유포_/미력서.doc.lnk”가 존재하여야 한다. 이러한

ZIP 파일의 압축된 파일 이름 관련된 YARA Rule을 작성할 수 있다.

ZIP 파일 구조의 Local File Header 구조의 경우 시그니처 (50 4B 03 04)로부터 30바이트 이후에 파일 이름이 존재하는 구조이고, Central Directory Header의 경우 시그니처 (50 4B 01 02)로부터 46바이트에 파일 이름이 존재한다. 이러한 특징을 이용해서 작성한 Rule은 다음과 같다. 작성한 Rule은 '첨부 7 작성한 YARA Rule'을 통해 상세히 확인할 수 있다.

```
rule asdcentral
{
strings:
$central_file = { 50 4b 01 02 }
$b = { 5F 5F B1 E8 C0 AF C6 F7 5F 5F 2F 61 73 64 2E 65 78 65 }
condition:
$central_file and
for any i in (1..#central_file):
($b in (@central_file[i]+46..@central_file[i]+46+uint16(@central_file[i]+0x12)))
)
}

rule lnkcentral
{
strings:
$central_file = { 50 4b 01 02 }
$b = { 5F 5F B1 E8 C0 AF C6 F7 5F 5F 2F C0 CC B7 C2 BC AD 2E 64 6F 63 2E 6C 6E 6B }
condition:
$central_file and
for any i in (1..#central_file):
($b in (@central_file[i]+0x2E..@central_file[i]+0x2E+uint16(@central_file[i]+0x19)))
)
```

Fig 27. Rule of Central Directory Header

```
rule asdlocal
{
strings:
$local_file = { 50 4b 03 04 }
$b = { 5F 5F B1 E8 C0 AF C6 F7 5F 5F 2F 61 73 64 2E 65 78 65 }
condition:
$local_file and
for any i in (1..#local_file):
($b in (@local_file[i]+30..@local_file[i]+30+uint16(@local_file[i]+0x12)))
)
}

rule lnklocal
{
strings:
$local_file = { 50 4b 03 04 }
$b = { 5F 5F B1 E8 C0 AF C6 F7 5F 5F 2F C0 CC B7 C2 BC AD 2E 64 6F 63 2E 6C 6E 6B }
condition:
$local_file and
for any i in (1..#local_file):
($b in (@local_file[i]+30..@local_file[i]+30+uint16(@local_file[i]+0x19)))
)
```

Fig 28. Rule of Local File Header

작성한 Rule을 분석하는 디스크 전체에 적용시키기 위해 FTK Imager를 이용하여 마운트 하였다.

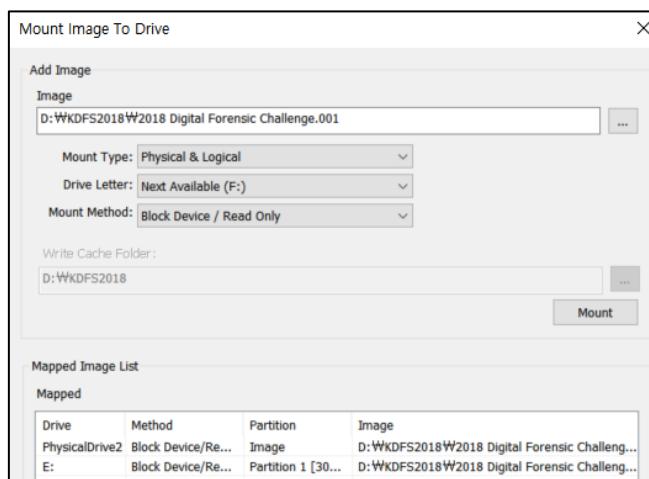


Fig 29. FTK Imager를 이용한 이미지 마운트

마운트를 이미지 한 후 YARA[13]를 이용해 시스템 전체에 대해 매칭되는 YARA rule이 존재하는지 확인하였다. Yara에 -r 옵션을 주고 검색을 시도하면 모든 파일에 대해 Recursive하게 검색을 수행할 수 있다.

```
Yara -rw $yara_path $drive_path
```

명령어 수행 결과, 권한 오류로 여러가 발생한 파일을 제외하고, "pagefile.sys"에 매칭된 Central

Directory Header 관련된 YARA rule을 확인할 수 있었다.

```
E:\>D:\KDFS2018\yara64.exe -rw D:\KDFS2018\yrar.txt .
error scanning .\fdfd0bf95c1dae9258c82b1350e8\!!!TO SAVE YOUR FILES!!!!.txt: could not open file
error scanning .\fdfd0bf95c1dae9258c82b1350e8\!!!SAVE YOUR FILES!!!.bmp: could not open file
ascentral .\pagefile.sys
Inkcentral .\pagefile.sys
error scanning .\Users\All Users\DELL\SARemediation\esp\EFI\Microsoft\Boot\BCD: could not open file
error scanning .\Users\All Users\DELL\SARemediation\esp\EF\Microsoft\Boot\BCD.LOG: could not open f
error scanning .\Users\All Users\Dropbox\Update\Log\DropboxUpdate.log-2018-10-23-59-36-182-9604: o
error scanning .\Users\All Users\Microsoft\Crypto\RSA\MachineKeys\2ae0248bfff9a91f828c352a796e04f99_b
error scanning .\Users\All Users\Microsoft\Crypto\RSA\MachineKeys\f686aace6942fb7f7ceb231212eef4a4_b
error scanning .\Users\All Users\Microsoft\Diagnosis\DownloadedScenarios\WINDOWS.DIAGNOSTICS.xml: co
error scanning .\Users\All Users\Microsoft\Diagnosis\DownloadedScenarios\WINDOWS.PERFTRACKESCALATIONS
error scanning .\Users\All Users\Microsoft\Diagnosis\DownloadedScenarios\WINDOWS.PERFTRACKPOINTDATA.
error scanning .\Users\All Users\Microsoft\Diagnosis\DownloadedScenarios\WINDOWS.SLUF.xml: could not
error scanning .\Users\All Users\Microsoft\Diagnosis\DownloadedSettings\telemetry.ASM-WindowsDefault
error scanning .\Users\All Users\Microsoft\Diagnosis\DownloadedSettings\telemetry.ASM-WindowsDefault
```

Fig 30. YARA Rule을 이용한 파일시스템의 패턴 매칭 결과

1.2.1에서와 같이 비활당 영역에 메타데이터가 존재할 가능성도 있어서 비활당 영역에서의 파일 카빙에 사용했던 “Free Space” 파일과 “Idle Space” 파일에 대해서도 Yara를 통한 패턴 매칭을 시도하였다. 사용한 명령어는 다음과 같다.

Yara \$yara_path “Free Space”
Yara \$yara_path “Idle Space”

명령어 수행 결과, 비활당 영역을 추출한 “Free Space” 파일과 “Idle Space” 파일에서는 얻고자 하는 ZIP 파일을 메타데이터를 수집할 수 없었다.

```
E:\>D:\KDFS2018\yara64.exe -D:\KDFS2018\yrar.txt "D:\KDFS2018\Free space"
E:\>D:\KDFS2018\yara64.exe -D:\KDFS2018\yrar.txt "D:\KDFS2018\Idle space"
```

Fig 31. YARA Rule을 이용한 비활당영역의 패턴 매칭 결과

ZIP 파일이 가지고 있는 Local File Header, Central Directory Header의 구조를 이용하여 “_김유포_.zip” 파일의 메타데이터를 유추하였다. 유추한 메타데이터는 Yara Rule로 작성하여 실제 분석 대상인 이미지에 패턴 매칭을 시도한 결과, “Pagefile.sys”에 Central Directory Header와 관련된 메타데이터가 존재함을 알 수 있었다. “Pagefile.sys” 내의 “_김유포_.zip”的 Local File Header의 경우 페이징이라는 메모리 관리 방법으로 인해 덮어 쓰였음을 유추할 수 있다.

3.3.2 페이지파일 메타데이터 카빙

“Pagefile.sys” 안에는 “_김유포_.zip”的 Central Directory Header가 존재하는 것을 확인하였다. Central Directory Header의 경우, 압축된 파일 이름 외에도 파일의 특징을 알 수 있는 메타데이터들이 존재한다. “Pagefile.sys” 안의 “_김유포_.zip”的 Central Directory Header를 확인하기 위해 Yara

rule에서 사용하였던 "5F 5F B1 E8 C0 AF C6 F7 5F 5F 2F C0 CC B7 C2 BC AD 2E 64 6F 63 2E 6C 6E 6B" ("_김유포_이력서.doc.lnk")를 Hex 검색으로 검색하였다. 검색 결과 총 3개가 검색되었으며, 그중 2개는 ZIP 파일 구조 중에서 End of Central Directory 구조를 확인할 수 있었다.

3A5C:4430h:	FC 01 00 5F 5F B1 E8 C0 AF C6 F7 5F 5F 2F C0 CC	ü.. ±èÀ È÷ /ÀÌ
3A5C:4440h:	B7 C2 BC AD 2E 64 6F 63 2E 6C 6E 6B 0A 00 20 00	·À¼-.doc.lnk.. .
3A5C:4450h:	00 00 00 00 01 00 18 00 A1 F5 05 EF E9 50 D4 01;õ.iéPô.
3A5C:4460h:	10 D0 F4 45 E2 50 D4 01 10 D0 F4 45 E2 50 D4 01	.ĐôEâPô..ĐôEâPô.
3A5C:4470h:	75 70 24 00 01 CC 3F 4E CD 5F 5F EA B9 80 EC 9C	up\$..ì?Ní _ ê¹eiœ
3A5C:4480h:	A0 ED 8F AC 5F 5F 2F EC 9D B4 EB A0 A5 EC 84 9C	i.¬ /i. 'ë ¥i,,œ
3A5C:4490h:	2E 64 6F 63 2E 6C 6E 6B 50 4B 05 06 00 00 00 00	.doc.lnkPK.....
3A5C:44A0h:	03 00 03 00 89 01 00 00 E2 FE 01 00 00 00 46 1B%...âp....F.
3A5C:44B0h:	8F 8A 84 AE 62 21 B0 6C F7 A3 36 51 F8 E0 E9 CF	.Š.,@b!°l÷f6QøáéÍ
3A5C:44C0h:	31 AD F0 75 E1 C7 B2 8F A8 14 C1 7D 31 77 5A 62	1-ðuáç². ..Á}1wZb
3A5C:44D0h:	2F C4 1B 0E 86 56 EA 64 DE 4F 2C A2 92 E0 1E 9F	/Ä..+VêdPô, ç' à.Ý
3A5C:44E0h:	A6 F7 F4 80 20 3C 42 55 7D E3 43 B8 5A 56 E0 07	;÷ô€ <BU}ãC, Zvà.
3A5C:44F0h:	F6 E7 0E 62 B9 E6 8D 58 31 F4 BA 71 65 51 8A 5E	öç.b¹æ.X1ô°qeQš^

Fig 32. End of Central Directory 발견 내역

A0 ED 8F AC	5F 5F 2F 61	73 64 2E 65	78 65 50 4B	i.¬ /asd.exePK
01 02 14 00	14 00 09 00	08 00 3D B7	34 4D D3 6A=·MÓj
C9 BF 8F 01	00 00 B1 02	00 00 19 00	4C 00 00 00	É¿....±.....L....
00 00 00 00	20 00 00 00	F4 FC 01 00	5F 5F B1 E8ôü.. ±è
C0 AF C6 F7	5F 5F 2F C0	CC B7 C2 BC	AD 2E 64 6F	À~È÷ /ÀÌ ·À¼-.do
63 2E 6C 6E	6B 0A 00 20	00 00 00 00	00 01 00 18	c.lnk..
00 A1 F5 05	EF E9 50 D4	01 10 D0 F4	45 E2 50 D4	.;õ.iéPô..ĐôEâPô
01 10 D0 F4	45 E2 50 D4	01 75 70 24	00 01 CC 3F	..ĐôEâPô.up\$..ì?
4E CD 5F 5F	EA B9 80 EC	9C A0 ED 8F	AC 5F 5F 2F	Ní _ ê¹eiœ i.¬ /
EC 9D B4 EB	A0 A5 EC 84	9C 2E 64 6F	63 2E 6C 6E	i. 'ë ¥i,,œ.doc.ln
6B 00 35 00	39 00 35 00	F1 D9 88 30	34 00 00 80	k.5.9.5.ñÙ^04 .€

Fig 33. End of Central Directory 발견하지 못함

존재하는 End of Central Directory를 영역을 해석해 본 결과 다음과 같은 정보를 얻을 수 있었다.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	50 4B 05 06	00 00 00 00	03 00 03 00	89 01 00 00	PK.....%....											
0010h:	E2 FE 01 00	00														âp...
Template Results - ZIPTemplate.bt																
Name																
struct ZIPENDLOCATOR endLocator																
> char elSignature[4]																
Value																
Start																
Size																
Color																

Fig 34. End of Central Directory의 상세 내용

"_김유포_.zip" 이라고 예상되는 파일의 메타데이터는 아래의 특징을 가지고 있음을 알 수 있다.

- 압축된 파일 수 (3개)
- Central Directory의 크기 (393 byte)
- Local File Header의 크기, Central Directory 시작 Offset (130786 byte)

위의 정보를 이용해 "_김유포_.zip" 이라고 예상되는 파일의 Central Directory 영역을 추출할 수 있다.

3.3.3 Zip 파일의 Central Directory 해석

추출할 Central Directory의 크기는 총 393 bytes로 "50 4B 01 02"(PK..)라는 시그니처를 가지고 있으며, 디렉토리와 파일을 포함해서 총 3개가 압축되어 있고, "Pagefile.sys"에 존재한다. "Pagefile.sys"에서 언급한 특징을 모두 만족시키는 영역을 추출한 것은 아래와 같다.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	50	4B	01	02	14	00	14	00	00	00	38	B7	34	4D		PK.....8'4M	
0010h:	00	00	00	00	00	00	00	00	00	00	0B	00	3B	00	;..	
0020h:	00	00	00	00	00	10	00	00	00	00	00	00	5F	5F		
0030h:	B1	E8	C0	AF	C6	F7	5F	5F	2F	0A	00	20	00	00	00	íèÀ~E=/_/..	
0040h:	00	01	00	18	00	22	F6	9E	E9	E9	50	D4	01	22	F6	9E	
0050h:	E9	E9	50	D4	01	42	BF	6F	38	E2	50	D4	01	75	70	13	
0060h:	00	01	5B	B5	55	6F	5F	5F	EA	B9	80	EC	9C	A0	ED	8F	
0070h:	AC	5F	5F	2F	50	4B	01	02	14	00	14	00	09	00	08	00	
0080h:	F5	71	0A	4D	69	CE	E6	D7	66	FC	01	00	04	48	03	00	
0090h:	12	00	42	00	00	00	00	00	00	00	20	00	00	00	40	00	
00A0h:	00	00	5F	5F	B1	E8	C0	AF	C6	F7	5F	5F	2F	61	73	64	
00B0h:	2E	65	78	65	0A	00	20	00	00	00	01	00	18	00		.exe.....	
00C0h:	52	F2	7C	2E	69	30	D4	01	5A	72	65	2E	69	30	D4	01	
00D0h:	5A	72	65	2E	69	30	D4	01	75	70	1A	00	01	47	19	05	
00E0h:	7D	5F	5F	EA	B9	80	EC	9C	A0	ED	8F	AC	5F	5F	2F	61	
00F0h:	73	64	2E	65	78	65	50	4B	01	02	14	00	14	00	09	00	
0100h:	08	00	3D	B7	34	4D	D3	6A	C9	BF	8F	01	00	00	B1	02	
0110h:	00	00	19	00	4C	00	00	00	00	00	00	20	00	00	00		
0120h:	F4	FC	01	00	5F	5F	B1	E8	C0	AF	C6	F7	5F	5F	2F	C0	
0130h:	CC	B7	C2	BC	AD	2E	64	6F	63	2E	6C	6B	6B	0A	00	20	
0140h:	00	00	00	00	00	01	00	18	00	A1	F5	05	EF	E9	50	D4	
0150h:	01	10	D0	F4	45	E2	50	D4	01	10	D0	F4	45	E2	50	D4	
0160h:	01	75	70	24	00	01	CC	3F	4E	CD	5F	5F	EA	B9	80	EC	
0170h:	9C	A0	ED	8F	AC	5F	5F	2F	EC	9D	B4	EB	A0	A5	EC	84	
0180h:	9C	2E	64	6F	63	2E	6C	6E	6B							æ.doc.lnk	

Template Results - ZIP Template.bt							
Name	Value	Start	Size				
struct ZIPENTRY dirEntry[0]	íèÀ~E=/_/	0h	74h				
struct ZIPENTRY dirEntry[1]	íèÀ~E=/_/asd.exe	74h	82h				
struct ZIPENTRY dirEntry[2]	íèÀ~E=/_/Al-Á,do...	F6h	93h				

Fig 35. 추출된 Central Directory 확인

"Pagefile.sys"에서 추출한 393바이트의 Central Directory 영역은 3개의 Central Directory으로 구성되어 있으며, 각각 가지고 있는 파일 이름은 "_김유포_/", "_김유포_/asd.exe", "_김유포_/이력서.doc.lnk"임을 확인할 수 있다. 총 3개의 Central Directory를 각각 해석하여 유의미한 정보를 추출하였다.

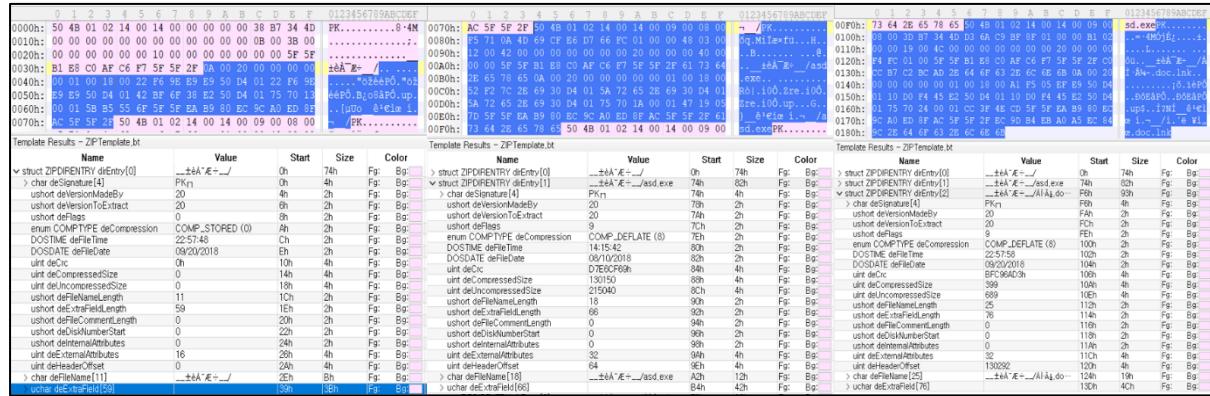


Fig 36. 압축 파일 내부의 각 Central Directory 데이터

'Fig 36' 의 좌측은 "_김유포_" 로서, "asd.exe"와 "이력서.doc.lnk" 파일을 포함하고 있는 폴더이다. 2018년 09월 20일 22시 57분 48초에 수정된 시간으로 나오고 "STORED" 형식으로 무압축 저장되었음을 알 수 있다. 또한 압축되기 전의 파일 크기와 압축된 파일 크기도 0이며 CRC32 값도 0으로 표시되어 있다. 이는 "_김유포_"가 단순히 폴더이고 파일이 아니기 때문이다.

'Fig 36' 의 중간은 "asd.exe"로 "_김유포_"의 하위에 존재한다. 2018년 08월 10일 14시 15분 42초에 수정된 것으로 나타나고, "DEFLATE"라는 압축 알고리즘에 의해 압축되어 있음을 알 수 있다. "asd.exe"의 Central Directory 같은 경우, General Bit Flag 가 9의 값을 나타내는데, 9는 이진법으로 0b1001로 나타낼 수 있다.

General Bit Flag의 경우 최하위 Bit 가 0이면 암호화되어 있지 않음을 나타내고, 1이면 암호화되어 있다는 것을 의미한다. 또한 Deflate 알고리즘의 경우 하위 두 번째 비트와 세 번째 Bit에 의해 압축 당시에 속도를 알아낼 수 있다[14]. 정리하면 다음과 같이 표현할 수 있다.

Bit 2	Bit 1	Bit 0	Description
Not use	Not use	0	Not Encrypted
Not use	Not use	1	Encrypted
0	0	Not use	Normal Compression
0	1	Not use	Maximum Compression
1	0	Not use	Fast Compression
1	1	Not use	Super Fast Compression

Table 25. General Bit Flag의 특징 (Deflate 알고리즘 사용 시)

위의 규칙에 따라 "asd.exe"는 "Normal Compression" 방법을 이용해 압축되어 있고 암호화되어 있음을 알 수 있다. 또한 CRC32 값이 "D7E6CF69"이며 압축 전의 원본 파일의 크기는 215040 바이트이고 압축 후에는 130150 바이트로 나타났다.

'Fig 36' 의 우측은 "이력서.doc.lnk" 파일로서 "_김유포_"의 하위에 존재한다. 2018년 09월 20일 22시 57분 58초에 수정된 것으로 나타나고, "DEFLATE"라는 압축 알고리즘에 의해 압축되어 있

음을 알 수 있다. "asd.exe"와 마찬가지로 General Bit Flag 가 9의 값을 나타내는데 이는 "Normal Compression" 방법으로 압축 되어있고 암호화되어 있는 것을 알 수 있다. 파일이 압축되기 전에는 689 바이트였고 압축 후에는 399 바이트로 나타났다. CRC32의 경우 BFC96AD3 이다. "이력서.doc.Ink"의 경우 "asd.exe"와는 다르게 삭제되지 않고 디스크 이미지 상에 남아있는 파일이다. 해당 파일을 복구하여 확인 가능한 파일 크기와 CRC 32가 일치함을 확인할 수 있었다.

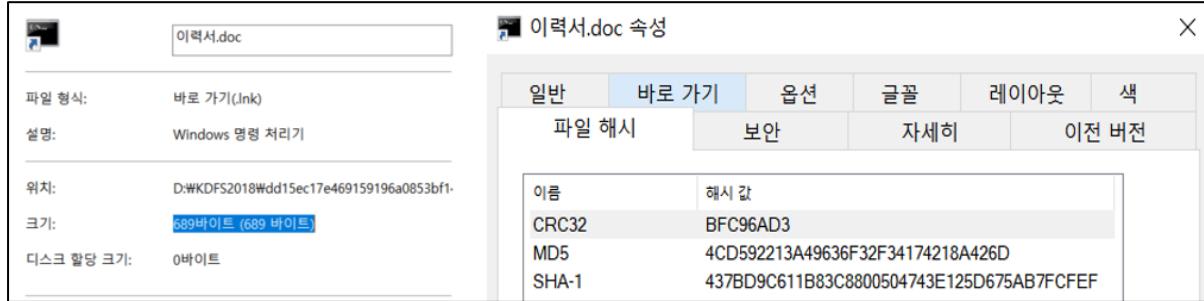


Fig 37. "이력서.doc.Ink"와의 CRC32 값 일치

이러한 메타데이터를 종합하여 유실된 Local File Header의 일부를 복구하여 압축해제는 불가하지만 메타데이터는 볼 수 있는 ZIP 파일을 구성할 수 있다.

pkpk.zip - ZIP 볼륨, 압축을 풀 파일 크기는 215,729 byte(s)입니다				
이름	압축된 크기	유형	변경된 날짜	CRC32
로컬 디스크				
이력서.doc.Ink *	399	바로 가기	2018-09-20 오후 10:57	BFC96AD3
asd.exe *	130,150	응용 프로그램	2018-08-10 오후 2:15	D7E6CF69

Fig 38. Local File Header 재구성한 결과

Zip 파일의 Central Directory에서 얻을 수 있는 정보들을 다음의 표로 최종 정리하였다.

File Name	Compress Type	Encrypted	Compress Speed	Original Size	Compressed Size	CRC32
김유포	STORED	X	NORMAL	0 (Directory)	0 (Directory)	0
asd.exe	DEFLATE	O	NORMAL	689 bytes	399 bytes	D7E6CF69
이력서.doc.Ink	DEFLATE	O	NORMAL	215040 bytes	130150 bytes	BFC96AD3

Table 26. Data from Zip Central Directory

3.4 OSINT를 이용한 asd.exe 파일 획득

ZIP Central Directory의 Header를 해석하여 asd.exe의 CRC32라는 메타데이터를 얻을 수 있었다. CRC32의 경우 비 암호학적 해시함수로서, 주로 파일의 체크섬을 검사하는 용도로 사용된다.

CRC32 값을 구글 검색을 통해 검색한 결과, 다음의 결과를 얻을 수 있었다.

[Detailed Analysis - Trojan/Ransom-ENZ - Viruses and Spyware ... - Sophos](https://www.sophos.com/en-us/threat.../detailed-analysis.aspx)
<https://www.sophos.com/en-us/threat.../detailed-analysis.aspx> ▾ 이 페이지 번역하기
 2017. 6. 15. - Size: 210K; SHA-1: 69d5094172cc962acec44fceee4db19204a556009; MD5:
 dea5cd9dcf444d6107b14cabefbb1774; CRC-32: d7e6cf69; File ...

Fig 39. CRC32 값을 이용한 구글 검색

구글 검색을 통해 발견된 사이트는 Sophos라는 악성코드 리포팅 사이트로, MD5 해시와 SHA-1 해시를 포함하고 파일 크기와 첫 발견 날짜를 나타내고 있다. 또한 "Ransom-ENZ"라는 진단명을 나타내고 있다[15].

The screenshot shows a detailed analysis page for the Trojan/Ransom-ENZ threat. Key details include:

- Category:** Viruses and Spyware
- Type:** Trojan
- Prevalence:** [Progress bar]
- Protection available since:** 15 Jun 2017 17:28:08 (GMT)
- Last Updated:** 15 Jun 2017 17:28:08 (GMT)
- File Information:**
 - Size: 210K
 - SHA-1: 69d5094172cc962acec44fceee4db19204a556009
 - MD5: dea5cd9dcf444d6107b14cabefbb1774
 - CRC-32: d7e6cf69
 - File type: Windows executable
 - First seen: 2017-06-14
- Download Sophos Home:** Free business-grade security for the home.
- Free Mac Anti-Virus:** Download our free Anti-Virus for Mac OS X

Fig 40. 동일한 CRC32 값으로부터 얻은 정보

검색된 MD5 해시값을 이용해 Hybrid-Analysis라는 악성코드 분석 사이트에 질의해 본 결과 다음의 결과를 얻을 수 있었다[16].

The screenshot shows the Hybrid-Analysis report for the MD5 hash dd15ec17e469159196a0853bf14edb45a86054c71bc555e2cd0afc1c410917b2. Key findings include:

- malicious**
- Threat Score:** 76/100
- AV Detection:** 13%
- Labeled as:** Mal_SageCrypt #ransomware
- Incident Response:**
 - Risk Assessment:**
 - Ransomware:** Detected indicator that file is ransomware
 - Spyware:** Accesses potentially sensitive information from local browsers
 - Fingerprint:** Reads the active computer name
 - Network Behavior:** Reads the cryptographic machine GUID
 - Network Behavior:** Contacts 1 domain and 1 host. View the network section for more details.

Fig 41. "Hybrid-Analysis"에서 발견한 악성코드 샘플

해당 사이트를 통해 "asd.exe"라고 추정할 수 있는 악성코드 샘플을 구할 수 있다. Hybrid-Analysis를 이용해 다운로드한 악성코드 샘플의 정보는 다음과 같다.

File Name	dd15ec17e469159196a0853bf14edb45a86054c71bc555e2cd0afc1c410917b2.bin
File Size	210KB (215,040 byte)
MD5	DEA5CD9DCF444D6107B14CABEFBB1774
SHA-1	69D5094172CC962ACEC44FCEE4DB19204A556009

Fig 42. Hybrid Analysis를 이용해 구한 악성코드의 세부 사항

'첨부 8 악성코드 샘플'을 통해 악성코드 샘플을 확인할 수 있다. 압축된 파일의 비밀번호는 "KDFS2018"로 별도 설정하여 압축하였다.

3.5 파일 동일성 검증

3.5.1 ZIP Central Directory 메타데이터를 이용한 파일 동일성 검증

인터넷 검색을 통하여 얻은 악성코드 샘플이 실제로 시스템에서 랜섬웨어의 행위를 한 "asd.exe"와 동일한지 파악하기 위해 실험을 진행하였다. 1.3.3 의 Zip Central Directory의 해석을 통하여, "_김유포_.zip"에 압축된 상태로 존재하는 "asd.exe"의 CRC32 값을 알 수 있었다. 실제로 인터넷 검색을 통해 얻은 악성코드도 동일한 CRC32와 파일 길이를 가지고 있는지 실험해 보았다.

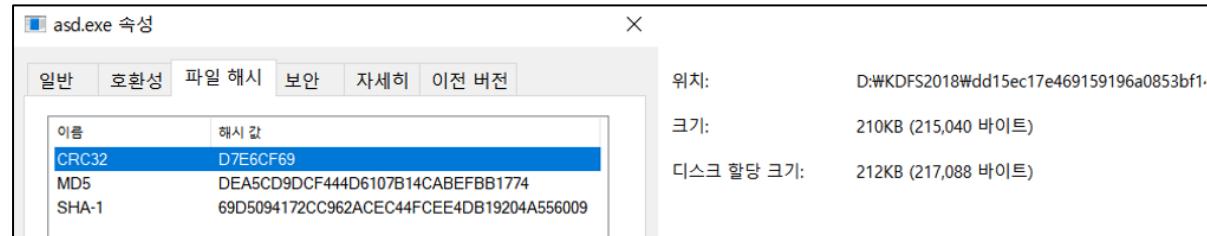


Fig 43. "asd.exe"와 수집한 악성코드 샘플의 파일 비교

확인 결과, 압축 파일 내부에 있던 "asd.exe"의 CRC32 값과 인터넷을 통해 찾은 악성코드의 CRC32의 값이 동일한 것으로 나타났다. 그러나 CRC32의 경우 비 암호학적 함수로 파일의 동일성을 입증하는 데는 한계가 있다. 또한 "_김유포_.zip"의 Central Directory 영역을 해석하면서 알 수 있었던, 압축되기 전 원본 파일의 크기 또한 215040 바이트로 일치함을 확인하였다.

아래의 추가적인 실험을 통해 "asd.exe"와 다운로드 받은 파일의 동일성 여부를 검증하였다.

3.5.2 Zip 파일 압축 크기를 이용한 동일성 검증

Zip 파일은 공개된 압축 알고리즘을 사용하고 있다. ZIP 형식에서 사용하는 대표적인 압축 알고리즘은 Deflate, LZMA가 대표적이다. 1.3.3을 통해 "_김유포_.zip" 내부의 2개의 파일, "이력

서.doc.lnk"와 "asd.exe"가 Deflate라는 알고리즘을 이용해 압축하고 있고, 또한 "Normal"의 속도로 암호화를 적용해서 압축했음을 알 수 있었다.

만약 압축된 파일의 동일하다면, 동일한 압축 알고리즘, 동일한 압축 속도, 동일한 암호화 여부를 적용한다면 동일한 압축된 파일 크기가 나와야 한다. 그러나 압축 프로그램 내부적으로 동일한 압축 알고리즘을 사용하더라도, 세부적인 로직에는 차이가 있어 압축 프로그램 별로 압축된 크기에 차이가 있을 수 있다. 즉, 압축 파일의 동일성을 증명하기 위해서는 동일한 압축 알고리즘, 동일한 압축 속도, 동일한 암호화 여부뿐만 아니라 사용한 압축 프로그램도 동일하게 설정하고 압축된 파일 사이즈를 비교하여야 한다.

압축된 파일 사이즈를 비교하기 위해서 4가지의 흔히 사용되는 압축 프로그램 (Alzip, 7-zip, WinRAR, Windows 기본 압축)을 이용하였다. 또한 압축의 속도 역시 Fast와 Normal 두 가지를 적용하였고 암호화 여부도 실험 대상으로 추가하였다. 적용할 수 있는 모든 경우의 수를 이용하여, 디스크 이미지에서 추출할 수 있었던 "이력서.doc.lnk" 파일과 인터넷으로 다운로드한 악성코드 "asd.exe"를 압축한 뒤 압축된 파일 크기를 확인하였다. 실험 결과는 아래의 표와 같다.

Archiver	Speed	Encryption	Filename	File size
Alzip	Fast	O	이력서.doc.lnk	408 bytes
			asd.exe	132989 bytes
	Normal	X	이력서.doc.lnk	396 bytes
			asd.exe	132977 bytes
	Normal	O	이력서.doc.lnk	399 bytes
			asd.exe	130150 bytes
		X	이력서.doc.lnk	387 bytes
			asd.exe	130138 bytes
7-zip	Fast	O	이력서.doc.lnk	410 bytes
			asd.exe	131259
		X	이력서.doc.lnk	398 bytes
			asd.exe	131247 bytes
	Normal	O	이력서.doc.lnk	404 bytes
			asd.exe	128149 bytes
		X	이력서.doc.lnk	392 bytes
			asd.exe	128137 bytes
WinRAR	Fast	O	이력서.doc.lnk	404 bytes
			asd.exe	131675 bytes
		X	이력서.doc.lnk	392 bytes
			asd.exe	131663 bytes
	Normal	O	이력서.doc.lnk	399 bytes
			asd.exe	130432 bytes

Windows	X	이력서.doc.lnk	387 bytes
		asd.exe	130420 bytes
	Not Usable	이력서.doc.lnk	389 bytes
		asd.exe	130419 bytes

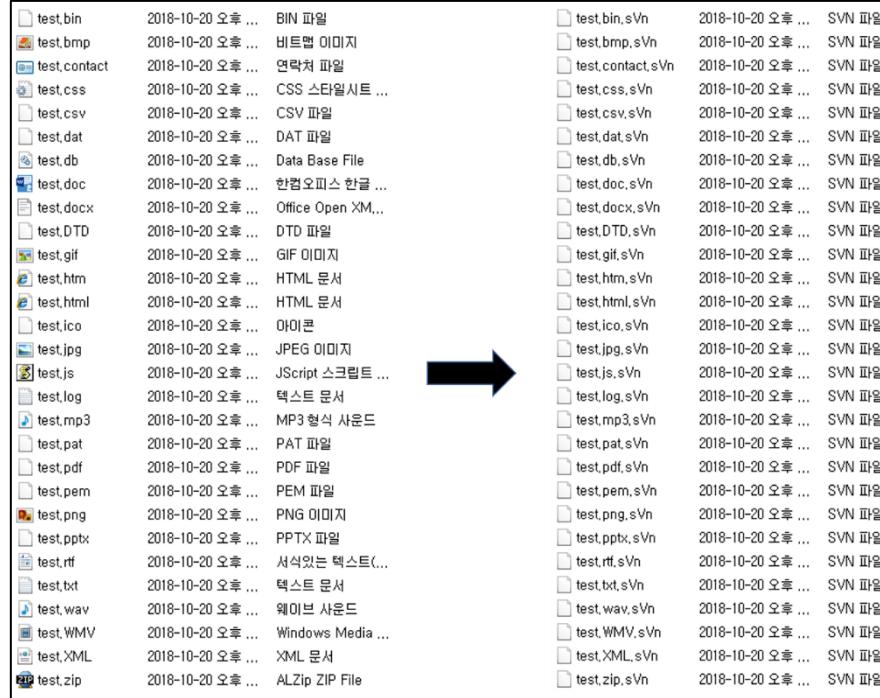
Table 27. ZIP 압축 테스트 결과

실험 결과, “_김유포__.zip”이라고 추정되는 파일의 Zip Central Directory를 해석해서 알 수 있었던 압축된 각각의 파일 크기 (“이력서.doc.lnk”의 압축된 파일 크기는 399 바이트, “asd.exe”의 압축된 파일 크기는 130150 바이트)는 알집을 이용해 “Normal”의 속도로 비밀번호를 압축한 상태에서만 나타났다. 압축된 파일 크기의 비교를 통하여 알집이라는 압축 프로그램을 이용해 “보통” 속도로 암호화를 설정하여 “_김유포__.zip”을 압축했음을 알 수 있고, 동일한 방법으로 압축하여 동일한 압축된 크기가 나온 “asd.exe”的 동일성도 검증할 수 있다.

3.5.3 asd.exe 행위 분석을 이용한 동일성 검증

“asd.exe”는 Jaff 랜섬웨어로 추정되는 악성코드이다. 랜섬웨어는 기본적으로 파일을 암호화하고 시그니처 같은 확장자를 추가하는 행위를 한다. 랜섬웨어 종류 별로 파일을 암호화하고 시그니처 확장자를 추가하는 행위는 상이하다. 만약 검색을 통해 구한 악성코드가 동일하다면, 랜섬웨어 행위로 인해 암호화하는 확장자 역시 동일해야 한다.

“\$LogFile”과 “\$MFT”를 기반으로 “.sVn”이라는 확장자를 가진 2701개의 파일 중 중복되는 확장자를 제거하여 총 암호화된 29개의 확장자를 선별할 수 있었다. 검색을 통해 구한 악성코드가 “asd.exe”와 동일하다면, 선별한 29개의 확장자 역시 암호화될 때 “.sVn”이라는 확장자 추가되어야 한다. 이를 실험해보기 위해 선별한 29개의 확장자를 가진 더미 파일을 생성하여 구한 가상 머신에서 악성코드를 실행해보았다. 실행 결과는 더미 파일로 생성한 29개의 확장자를 가진 파일 모두 “.sVn”이라는 확장자가 추가됨을 확인할 수 있었다.



test.bin	2018-10-20 오후 ...	BIN 파일	test.bin.sVn	2018-10-20 오후 ...	SVN 파일
test.bmp	2018-10-20 오후 ...	비트맵 이미지	test.bmp.sVn	2018-10-20 오후 ...	SVN 파일
test.contact	2018-10-20 오후 ...	연락처 파일	test.contact.sVn	2018-10-20 오후 ...	SVN 파일
test.css	2018-10-20 오후 ...	CSS 스타일시트 ...	test.css.sVn	2018-10-20 오후 ...	SVN 파일
test.csv	2018-10-20 오후 ...	CSV 파일	test.csv.sVn	2018-10-20 오후 ...	SVN 파일
test.dat	2018-10-20 오후 ...	DAT 파일	test.dat.sVn	2018-10-20 오후 ...	SVN 파일
test.db	2018-10-20 오후 ...	Data Base File	test.db.sVn	2018-10-20 오후 ...	SVN 파일
test.doc	2018-10-20 오후 ...	한컴오피스 한글 ...	test.doc.sVn	2018-10-20 오후 ...	SVN 파일
test.docx	2018-10-20 오후 ...	Office Open XM...	test.docx.sVn	2018-10-20 오후 ...	SVN 파일
testDTD	2018-10-20 오후 ...	DTD 파일	testDTD.sVn	2018-10-20 오후 ...	SVN 파일
test.gif	2018-10-20 오후 ...	GIF 이미지	test.gif.sVn	2018-10-20 오후 ...	SVN 파일
test.htm	2018-10-20 오후 ...	HTML 문서	test.htm.sVn	2018-10-20 오후 ...	SVN 파일
test.html	2018-10-20 오후 ...	HTML 문서	test.html.sVn	2018-10-20 오후 ...	SVN 파일
test.ico	2018-10-20 오후 ...	아이콘	test.ico.sVn	2018-10-20 오후 ...	SVN 파일
test.jpg	2018-10-20 오후 ...	JPEG 이미지	test.jpg.sVn	2018-10-20 오후 ...	SVN 파일
test.js	2018-10-20 오후 ...	JScript 스크립트 ...	test.js.sVn	2018-10-20 오후 ...	SVN 파일
test.log	2018-10-20 오후 ...	텍스트 문서	test.log.sVn	2018-10-20 오후 ...	SVN 파일
test.mp3	2018-10-20 오후 ...	MP3 형식 사운드	test.mp3.sVn	2018-10-20 오후 ...	SVN 파일
test.pat	2018-10-20 오후 ...	PAT 파일	test.pat.sVn	2018-10-20 오후 ...	SVN 파일
test.pdf	2018-10-20 오후 ...	PDF 파일	test.pdf.sVn	2018-10-20 오후 ...	SVN 파일
test.pem	2018-10-20 오후 ...	PEM 파일	test.pem.sVn	2018-10-20 오후 ...	SVN 파일
test.png	2018-10-20 오후 ...	PNG 이미지	test.png.sVn	2018-10-20 오후 ...	SVN 파일
test.pptx	2018-10-20 오후 ...	PPTX 파일	test.pptx.sVn	2018-10-20 오후 ...	SVN 파일
test.rtf	2018-10-20 오후 ...	서식있는 텍스트(...)	test.rtf.sVn	2018-10-20 오후 ...	SVN 파일
test.txt	2018-10-20 오후 ...	텍스트 문서	test.txt.sVn	2018-10-20 오후 ...	SVN 파일
test.wav	2018-10-20 오후 ...	웨이브 사운드	test.wav.sVn	2018-10-20 오후 ...	SVN 파일
test.WMV	2018-10-20 오후 ...	Windows Media ...	test.WMV.sVn	2018-10-20 오후 ...	SVN 파일
test.XML	2018-10-20 오후 ...	XML 문서	test.XML.sVn	2018-10-20 오후 ...	SVN 파일
test.zip	2018-10-20 오후 ...	ALZip ZIP File	test.zip.sVn	2018-10-20 오후 ...	SVN 파일

Fig 44. 파일 확장자 암호화 여부 테스트

또한 암호화하는 파일의 폴더마다 랜섬 노트 파일인 "TO SAVE YOUR FILES!!!!.txt" 와 "!!!SAVE YOUR FILES!!!.bmp"를 생성하였다. 파일 내용으로는 복호화를 위한 ID와 Tor 설치법, Deep Web 주소를 알려주는 것을 확인할 수 있었다.

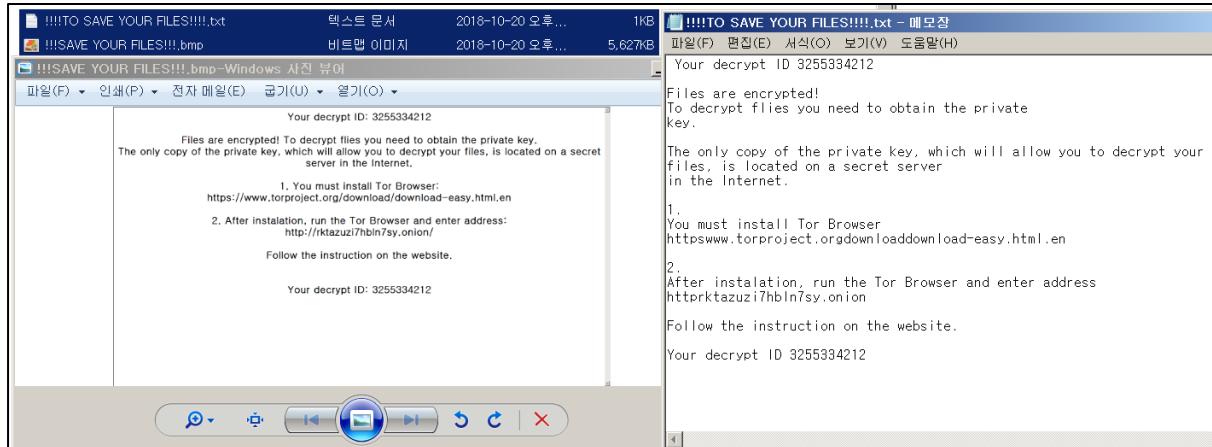


Fig 45. 랜섬 노트 생성 확인

그러나 파일 복호화에 사용되는 Decrypt ID는 시스템 환경에 따라 달라짐을 확인할 수 있었다. 추출한 랜섬웨어에 대한 세부적인 행위는 본 보고서의 "랜섬웨어 행위 분석 및 복호화" 부분에서 다루도록 하겠다.

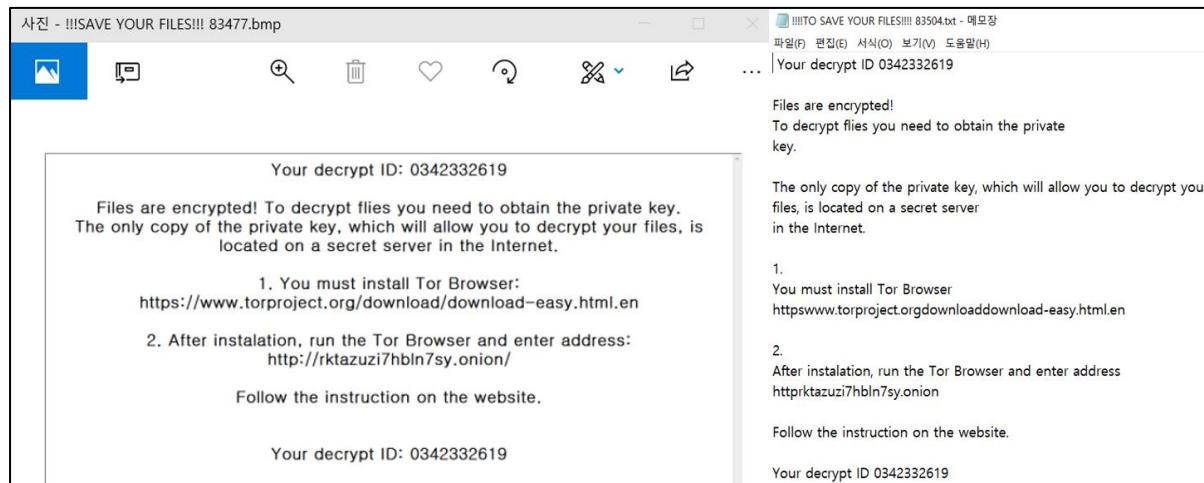


Fig 46. Decrypt ID의 차이

3.5.4 파일 슬랙 부분을 이용한 동일성 검증

슬랙 영역은, 물리적인 구조와 논리적인 구조의 차이로 인하여 발생한다. 물리적으로는 파일에 할당되어 있지만 논리적으로는 사용할 수 없는 공간으로 파일 복구와 삭제된 파일의 일부를 조사할 때 사용할 수 있다[17].

슬랙 영역은 램 슬랙과 파일 슬랙, 파일 시스템 슬랙, 볼륨 슬랙으로 분류되며 이 중 파일 슬랙은, 클러스터 사용으로 낭비되는 공간 중 램 슬랙을 제외한 부분으로 특정 파일이 존재하였는지 규명할 수 있으며, 0x00으로 기록되는 램 슬랙과 다르게 이전의 데이터가 그래도 남아있다는 특징이 있다. 클러스터 사용 내역은 \$logfile을 이용해 확인할 수 있다.

실험을 위해 NTFS Log Tracker를 이용해 “_김유포_.zip”와 관련된 클러스터 사용 내역을 조사하였다. 해당 내역은 다음과 같다.

270301387	Writing Content of Non-Resident File	Cluster Number : 5005095(33)	_김유포_[1].zip
270316605	Writing Content of Non-Resident File	Cluster Number : 5005128(5)	nsd173618176[1].png
270317477	Writing Content of Non-Resident File	Cluster Number : 5005133(4)	nsd15453589[1].png
270320108	Writing Content of Non-Resident File	Cluster Number : 5005161(102)	
270387190	Writing Content of Non-Resident File	Cluster Number : 5005137(1)	
270432055	Writing Content of Non-Resident File	Cluster Number : 5005139(1)	C__Program_Files_ESTsoft_ALZip_ALZip_exe[1]
270440380	Writing Content of Non-Resident File	Cluster Number : 5005095(5)	nsd172615885[1].png
270440972	Writing Content of Non-Resident File	Cluster Number : 5005100(5)	nsd144944309[1].png
270441639	Writing Content of Non-Resident File	Cluster Number : 5005105(5)	nsd145032565[1].png
270445498	Writing Content of Non-Resident File	Cluster Number : 5005110(2)	nsd192546763[1].png
270446090	Writing Content of Non-Resident File	Cluster Number : 5005112(4)	nsd151929775[1].png
270449550	Writing Content of Non-Resident File	Cluster Number : 5005120(5)	nsd144020188[1].png

Fig 47. “_김유포_.zip”的 클러스터 내역

클러스터 사용 내역을 그림으로 표현하면 다음과 같다.

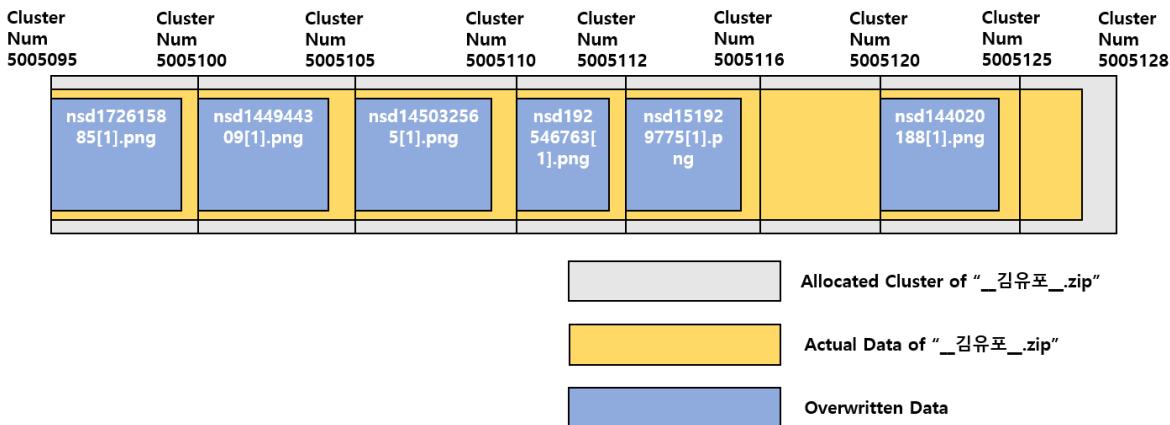


Fig 48. "_김유포_.zip"과 관련된 슬랙 영역

\$LogFile의 내역을 기준으로, 다른 파일 데이터로 덮이지 않은 "_김유포_.zip"의 데이터가 일부 존재함을 알 수 있었다. 존재하는 일부의 "_김유포_.zip"의 데이터는 두 가지 종류로 분류할 수 있다. 우선 클러스터 내에 이미 다른 파일이 존재하나, 파일의 끝 부분 이후에 원래의 데이터가 존재할 수 있다. 다른 한 가지는 클러스터에 다른 파일로 인한 데이터가 덮이지 않은 경우이다.

전자의 경우 그림에서 확인할 수 있는 것과 같이, 6개의 부분이 존재한다. 각 부분에 대한 정보를 나타내면 다음과 같다.

Index	Cluster Number	Logical Sector Number	Description
1	5005099	40040792	PNG Footer + 0x00 filled
		40040793 ~ 40040799	"_김유포_.zip" Data (추정)
2	5005104	40040832	PNG Footer + 0x00 filled
		40040833 ~ 40040839	"_김유포_.zip" Data (추정)
3	5005109	40040872	PNG Footer + 0x00 filled
		40040873 ~ 40040879	"_김유포_.zip" Data (추정)
4	5005111	40040890	PNG Footer + 0x00 filled
		40040891 ~ 40040895	"_김유포_.zip" Data (추정)
5	5005115	40040920 ~ 40040924	0x20 filled
		40040924 ~ 40040927	PNG Data
		40040927	PNG Footer + 0x00 filled
6	5005124	40040992	PNG Footer + 0x00 filled
		40040993 ~ 40040999	"_김유포_.zip" Data (추정)

Table 28. "_김유포_.zip" 관련 클러스터 사용 내역

그러나 "_김유포_.zip"의 경우 압축된 파일 내용이 암호화되어 있다. 이는 압축 파일을 생성할 때 생성한 비밀번호를 알지 못하는 이상 동일성 검증이 불가능하고 위에 같이 "_김유포_.zip"의 데이터라고 추정만 가능하다.

후자의 경우 그림에서 확인할 수 있는 것과 같이, 2개의 부분이 존재하며 내용은 다음과 같다.

Index	Cluster Number	Logical Sector Number	Description
1	5005116 ~ 5005119	40040928 ~ 40040932	4AHSWCCU Data
		40040933 ~ 40040959	0x00 filled
2	5005125 ~ 5005128	40041000 ~ 40041023	{F00D1A16-BE1A-11E8-A01B-000C29421733}.dat.sVn Data

Table 29. "asd.exe" 관련 클러스터 사용 내역

"_김유포_.zip"과 동일한 방법으로 \$logfile의 정보를 이용해 "asd.exe"와 관련된 클러스터 사용 내역을 조사해보았다.

270004110	Writing Content of Non-Resident File	Cluster Number : 5024566(8)	TarE422.tmp
270162006	Writing Content of Non-Resident File	Cluster Number : 5024489(14)	CabFEF5.tmp
270178887	Writing Content of Non-Resident File	Cluster Number : 5024567(14)	Cab183.tmp
270179305	Writing Content of Non-Resident File	Cluster Number : 5024581(8)	Tar193.tmp
270428730	Writing Content of Non-Resident File	Cluster Number : 5024489(8)	asd.exe

Fig 49. "asd.exe" 관련 클러스터 내역

조사한 결과, "asd.exe"가 클러스터에 기록된 이후로 \$logfile에는 Cluster에 해당하는 데이터가 존재하지 않으나 해당 영역을 살펴본 결과 다음과 같은 데이터가 발견되었다.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4CAAE9000	20	20	20	20	20	20	3C	78	73	64	3A	65	6C	65	6D	
4CAAE9010	65	6E	74	20	64	65	66	61	75	6C	74	3D	22	53	72	69
4CAAE9020	20	4C	61	6E	6B	61	20	53	74	61	6E	64	61	72	64	20
4CAAE9030	54	69	6D	65	22	20	6E	61	6D	65	3D	22	53	72	69	4C
4CAAE9040	61	6E	6B	61	53	74	64	56	61	6C	75	65	22	20	74	79
4CAAE9050	70	65	3D	22	78	73	64	3A	73	74	72	69	6E	67	22	20
4CAAE9060	77	63	6D	3A	68	61	6E	64	6C	65	72	3D	22	72	65	67
4CAAE9070	6B	65	79	28	27	48	4B	45	59	5F	4C	4F	43	41	4C	5F
4CAAE9080	4D	41	43	48	49	4E	45	5C	53	4F	46	54	57	41	52	45
4CAAE9090	5C	4D	69	63	72	6F	73	6F	66	74	5C	57	69	6E	64	6F
4CAAE90A0	77	73	20	4E	54	5C	43	75	72	72	65	6E	74	56	65	72
4CAAE90B0	73	69	6F	6E	5C	54	69	6D	65	20	5A	6F	6E	65	73	5C
4CAAE90C0	53	72	69	20	4C	61	6E	6B	61	20	53	74	61	6E	64	61

Fig 50. "asd.exe" 클러스터의 실제 데이터

데이터는 XML 형식의 파일로, 검색을 통해 구한, "asd.exe"에는 존재하지 않는다. 파일의 내용도 "Free Space"로 확인되었으며 실제 데이터로 보아, 해당 클러스터에는 이미 지워진 "asd.exe"가 아닌 파일이 내용이 덮어 씌워진 것으로 추정된다.

"asd.exe"와 관련된 클러스터 내용을 그림으로 표현하면 다음과 같다.

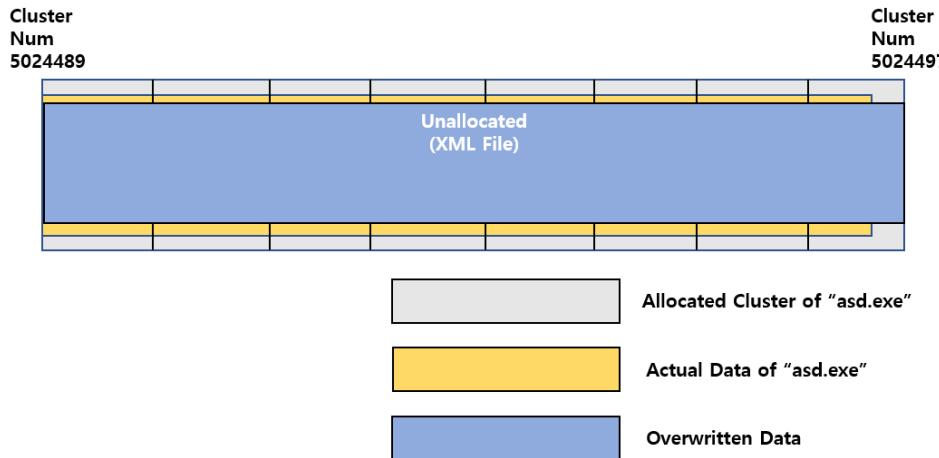


Fig 51. "asd.exe" 관련 클러스터 내역

슬랙 데이터를 이용하여, 파일 일부분을 추출해 검색을 통해 얻은 "asd.exe"나 "asd.exe"와 "이력서.doc.lnk"를 압축해서 만든 "_김유포_.zip"의 일치 여부를 통해 동일성을 검증하려고 시도하였다. 그러나 "_김유포_.zip"의 경우 암호화가 적용된 압축 파일로 압축된 데이터가 압축 파일을 만들 때 사용된 비밀번호로 암호화되어 있어, 실제 비밀번호를 알지 못하는 한 해당 슬랙 부분이 실제 "_김유포_.zip"과 동일한지 여부는 알 수 없었다. 일부 데이터는 \$LogFile에는 반영되지 않았지만, 다른 파일의 데이터로 덮어 쓰인 것도 확인할 수 있었다. "asd.exe"의 경우, \$LogFile에서 해당 클러스터에 추가적인 사용 내역은 없으나, 실제로 클러스터를 확인해보니 "asd.exe"가 아닌 다른 파일이 이미 덮어 쓰인 것을 확인할 수 있었다.

3.6 소결

이미지 분석을 통해 파악한 "_김유포_.zip"과 "asd.exe"의 관련된 흔적에 관하여 면밀히 살펴보았다. 특히 비활당영역에서의 파일 복구, 볼륨 쉐도우 카피를 이용한 파일 복구, 페이지 파일에서의 파일 복구를 통해 "_김유포_.zip"과 "asd.exe"의 원본 파일을 획득하려 시도하였지만 이미 삭제가 되어있는 상태이거나, 다른 데이터가 덮어 쓰여 있어 복구하지는 못하였다. 그럼에도 불구하고 ZIP 파일의 구조를 활용해 YARA Rule을 작성 후 이를 이용해 "Pagefile.sys" 내에 ZIP Central Directory Data가 존재함을 확인하였다. ZIP Central Directory Data에는 CRC32와 파일 이름, 원본 데이터의 크기와 압축된 데이터의 크기를 가지고 있다. ZIP Central Directory 중 "asd.exe"의 CRC32 정보를 검색을 통해 MD5 해시값, 악성코드 원본을 차례로 구할 수 있었다.

그러나 CRC32 정보만 이용해 본 케이스에 적용된 "asd.exe"와 구한 악성코드 원본이 동일하다고 판단하기에 근거가 부족하여 동일성에 대한 실험을 추가로 진행하였다. 실험 결과, ZIP Central Directory 메타데이터 비교, ZIP 파일 내의 압축 크기 비교, 악성코드 행위 비교를 통해 검색을 통해 구한 악성코드와 본 케이스에 적용된 "asd.exe"가 동일함을 입증하였다. 이하 보고서에서는 "asd.exe"의 상세 분석을 통해 악성코드가 시스템에 어떠한 행위를 하는지 확인해보도록 하겠다.

4. 랜섬웨어 행위 분석 및 파일 복호화

4.1 악성코드 샘플 파일 입수

본 문제 상황에서 사용된 랜섬웨어는 이미 알려진 바 있는 악성코드의 일종으로 파악되었다. 특히 감염 시 아래와 같은 메시지를 표출하는 것에서 단서를 확보하였다.

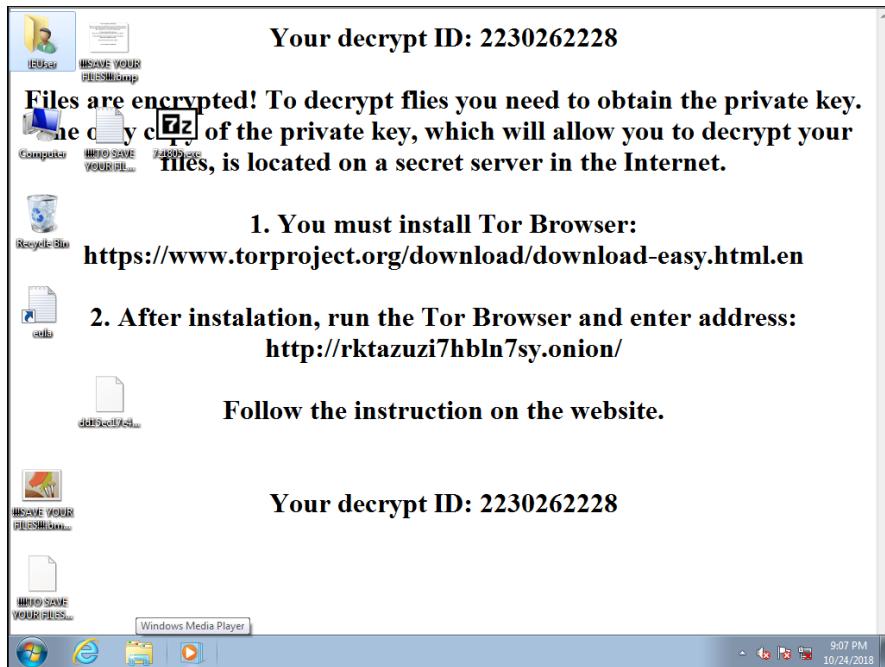


Fig 52. 악성코드에 감염된 화면

랜섬웨어의 경우 공격자의 궁극적 목표는 금전적 이득이므로 자신들에게 돈을 입금할 방법을 설명해두곤 하는데, 본 상황에서의 악성코드 유포자는 익명성을 확보하기 위해 Tor Web Browser를 사용하여 <http://rktazuzi7hbln7sy.onion> 사이트에 접속하도록 유도하고 있다. 본 URL은 현재로는 차단되어 접속이 불가능한 상황이지만, 구글 검색을 통해 Jaff Ransomware라는 키워드를 획득할 수 있었다. 재프 랜섬웨어는 2017년 5월경 유행했던 문서형 악성코드로 알려져 있다[18].

해당 랜섬웨어의 구체적인 작동방식을 파악하기 위해 <https://www.hybrid-analysis.com/>에서 샘플을 획득하였다. 입수한 악성코드의 정보는 'Table 30과 같다.

파일명 (SHA-256 Hash)	파일크기
dd15ec17e469159196a0853bf14edb45a86054c71bc555e2cd0afc1c410917b2.zip	215,040 bytes

Table 30. 문제에서 사용된 것과 동일한 종류의 Jaff Ransomware 샘플

안전한 가상 머신 환경을 구축하여 내부에서 악성코드를 정적/동적으로 분석함으로써 어떤 과정을 거쳐 피해자의 파일들을 암호화하였는지를 확인하기로 하였다.

4.2 정적 분석

4.2.1 PE 바이너리 정보

우선 악성코드 바이너리가 컴파일될 때 기록된 정보들을 통해서 대략적인 단서들을 수집하였다. 'Table 31'은 PE explorer 도구를 통해 얻은 정보이다.

Info	Contents
Subsystem	0x2(WINDOWS)
Image Base	0x400000
Characteristics	0x103(RELOCS_STRIPPED, EXECUTABLE_IMAGE, 32BIT_MACHINE)
PE File Type	PE32
Stored Checksum	0x419EE
File Alignment	0x200
Entry Point	0x7E89
Section Alignment	0x1000

Table 31. PE Binary 정보

또한 PE 바이너리 내부에 Import 된 라이브러리의 목록이 'Table 32'에 있다.

Name	Description
KERNEL32.dll	Windows NT 기본 API 클라이언트 DLL
USER32.dll	다중 사용자 Windows 사용자 API 클라이언트 DLL
GDI32.dll	GDI Client DLL
COMDLG32.dll	Common Dialogs DLL
SHELL32.dll	Windows 셸 공용 DLL
ole32.dll	Windows용 Microsoft OLE
OLEAUT32.dll	Windows용 Microsoft OLE
WinSCard.dll	Microsoft 스마트 카드 API
COMCTL32.dll	사용자 경험 관리 라이브러리
RPCRT4.dll	원격 프로시저 호출 런타임
WINHTTP.dll	Windows HTTP Services
setupapi.dll	Windows 설치 API
SensApi.dll	SENS Connectivity API DLL
mscms.dll	Microsoft 색 일치 시스템 DLL
ADVAPI32.dll	고급 Windows32 기본 API

Table 32. Import 정보

다양한 종류의 라이브러리가 포함되어 있지만, 정상적인 프로그램에서도 기본적으로 호출하고 있는 라이브러리를 제외하고 그중 특이하게 주목해야 할 항목이 있다면 "ADVAPI32.dll" 이다.

특히 해당 API 가 지원하고 있는 다양한 기능을 살펴보면 아래 'Fig 53'와 같이 cryptsp.dll 이나 bcrypt.dll 을 내포하고 있다. 이들은 Microsoft에서 제공하는 Windows 기반의 암호화 (Cryptographic) 관련 서비스를 제공하는 API이다.

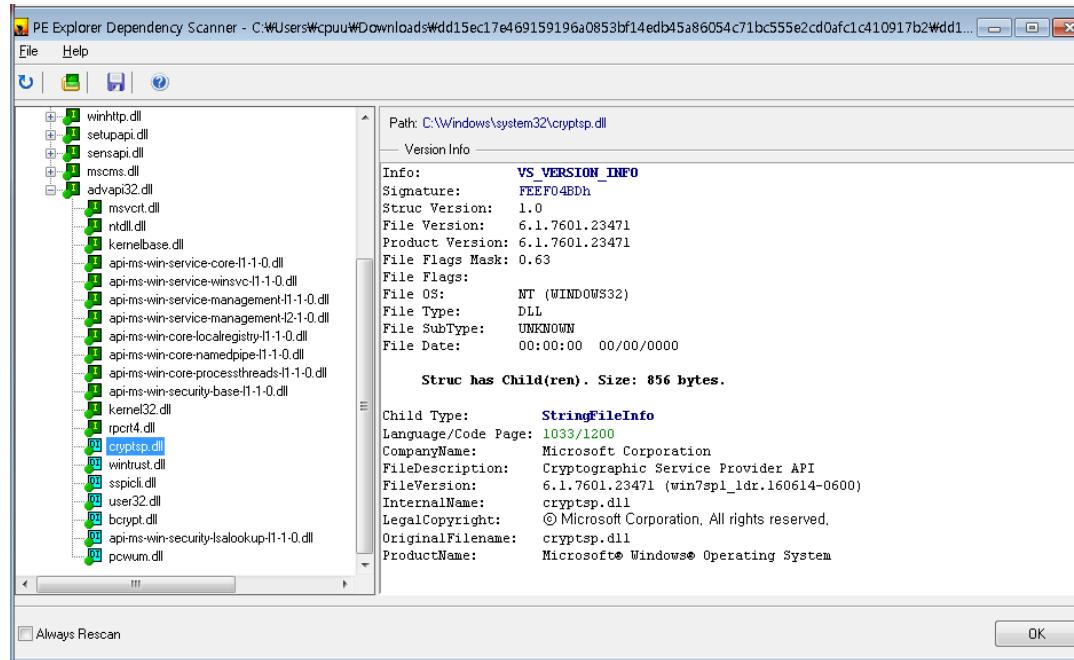


Fig 53. 암호화 관련 API가 포함된 라이브러리

랜섬웨어의 특성상 사용자의 파일을 암호화하기 위해서는 현대에서 사용되는 다양한 암호화 기법을 프로그래밍적으로 구현할 수 있는 방법이 필요한데, 악성코드 개발자가 직접 개발하는 경우도 있겠지만 대부분의 경우 이미 잘 설계되어 있는 OpenSSL 등의 라이브러리를 활용하는 경우가 많다. 본 사례에서도 마찬가지로 Windows에서 제공하는 암호화 라이브러리를 이용한 것으로 보이며 이 부분은 동적 분석에서 더욱 살펴보도록 하겠다.

그 밖의 파일 메타데이터 정보를 EXIF 추출 프로그램을 사용하여 'Table 33'와 같이 정리하였다.

Name	Description
MIME Type	application/octet-stream
Subsystem	Windows GUI
Machine Type	Intel 386 or later, and compatibles
Filetype Extension	exe
Timestamp	2017:06:14 11:13:01+01:00
Filetype	Win32 EXE
PEType	PE32
Code Size	101376
Linker Version	14.0
ImageFileCharacteristics	No relocs, Executable, 32-bit

EntryPoint	0x7e89
InitializedDataSize	112640
Subsystem Version	6.0
Image Version	0.0
OS Version	6.0
UninitializedDataSize	0

Table 33. File Metadata

요약하면 Intel 기반의 32bit 환경 Windows 운영체제에서 작성된 PE 바이너리 파일이며, 2017년 6월 14일 11시 13분 1초에 컴파일된 것으로 보인다.

4.1.2 Decompile 분석

악성코드 샘플의 정적 분석을 위해 IDA Pro를 활용하였다. 기본적으로 본 샘플은 C/C++ 을 혼용해서 사용한 것으로 보이며, GUI 기반 환경을 개발하기 위해 Windows 프로그래밍을 적용하였다.

'Fig 54'은 IDA Pro를 사용하여 WinMain함수를 디컴파일 한 화면이다.

```

    f sub_401260 ,text 0 678 SelectObject(v73, v75);   "
    f sub_401266 ,text 0 679 lstrcat(String1, "Alloc");
    f sub_401292 ,text 0 680
    f sub_4012C4 ,text 0 681 Rectangle(v73, Rect.left, Rect.top, Rect.right, Rect.bottom);
    f sub_40131C ,text 0 682 ReleaseDC((HWnd)String1, v73);
    f sub_401364 ,text 0 683 DeleteObject(v74);
    f sub_40559A ,text 0 684 DeleteObject(v75);
    f WinMain(x,x,x,x) ,text 0 685 v70 = (HWNd)colorref[0];
    f sub_4074D5 ,text 0 686 }
    f sub_407524 ,text 0 687 v76 = GetModuleHandleA("kernel32.dll");
    f sub_40759E ,text 0 688 v77 = GetProcAddress(v76, String1);
    f sub_4075F5 ,text 0 689 sub_401292((int)&x1, (int)"%s %d", (int)"Message");
    f sub_40767F ,text 0 690 v78 = ((int) __stdcall *) (DWORD, _DWORD, _HDC) v77)(VolumeSerialNumber[0], 0, v166);
    f sub_4076B4 ,text 0 691 v157 = v78;
    f sub_4076E1 ,text 0 692 SendMessageA(v78, 0xCu, 0, (LPARAM)&x1);
    f sub_40772F ,text 0 693 DispatchMessageA((const MSG *)&pv);
    f sub_40786C ,text 0 694 SendMessageA(v78, 0xDu, 0x3E8u, (LPARAM)&x1);
    f sub_407900 ,text 0 695 v79 = sub_40BCC0((int)v217);
    f sub_407992 ,text 0 696 sub_401266((int)"Total Messages : %d\n", v79);
    f sub_407A06 ,text 0 697 DestroyWindow(v70);
    f sub_407A4A ,text 0 698 if (v78 / 25 != 1901 )
    f sub_407AB0 ,text 0 699 v165 = v157;
    f sub_407AD1 ,text 0 700 String1_CSTR = (RPC_CSTR)(unsigned __int16)lParam;
    f std::exception::exception(std::exception&) ,text 0 701 left = (unsigned __int16)lParam + v157;
    f std::exception::exception(char const * c...) ,text 0 702 v80 = left;
    f sub_401364 ,text 0 703 y += (unsigned __int8)h / 10;
    f sub_407AAC ,text 0 704 sub_401364();

```

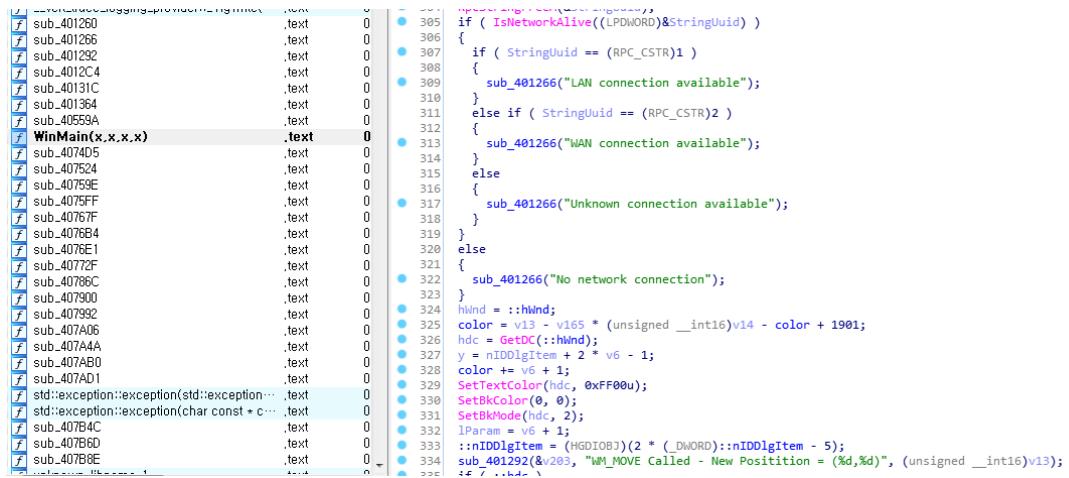
Fig 54. GetModuleHandleA()를 이용하여 kernel32.dll을 호출하는 구문

해당 프로그램은 Main 이 시작하게 되면 GetModuleHandleA()함수를 이용하여 kernel32.dll을 호출한다. 이는 메모리 관리, 입출력 명령, 프로세스와 스레드 생성, 그리고 동기화 등 대부분의 Windows 운영체제 레벨의 많은 작업들을 관리한다. 특히 그중 일부인 COMCTL32.DLL 이 파일 오픈, 저장 등의 원도우 표준 파일 처리를 담당하고 있는데 본 악성코드가 사용자의 파일에 대한 접근 및 처리, 삭제 등을 반복하는 속성을 가졌음을 기억할 때 이렇게 API를 사용할 준비를 하는 점은 상당히 유의미하다.

결국 랜섬웨어의 가장 핵심적인 기능은 특정 디스크의 파일 시스템을 순차적으로 순회하면서 파일을 검색하고, 몇몇 확장자 등 목적에 부합하는 파일인 경우 암호화 함수를 호출하여 암호화된 파일을 만든 후 기존 원본 파일을 삭제하는 과정을 반복문을 통해 지속 수행하는 것이다. 본 랜

섬웨어도 이와 크게 다르지 않을 것으로 가정하고 관련된 함수 구문들을 위주로 살펴보았으나, 정적 분석만으로는 해당 구문을 발견할 수 없었다. 이 이유는 악성코드가 리버스 엔지니어링을 회피하기 위해 주요 구문들을 스스로 인코딩한 상태로 보유하고 있다가, 실행되는 시점에서 메모리에 코드를 올리고 디코딩하여 나온 결과물을 실행하는 방식을 취하는 것으로 추정된다. 이러한 내용은 동적 분석 및 실행 시점에서의 메모리 분석을 통해 확인할 수 있을 것이다.

우선 본 절에서는 정적 분석 만으로 얻을 수 있는 정보들을 위주로 살펴본다. 아래 'Fig 55'은 랜섬웨어의 WinMain 함수에서 프로그램이 작동하는 초기에 외부 네트워크와의 연결 상태를 확인하는 구문이다.



```

    if ( IsNetworkAlive((LPDWORD)&StringUuid) )
    {
        if ( StringUuid == (RPC_CSTR)1 )
        {
            sub_401266("LAN connection available");
        }
        else if ( StringUuid == (RPC_CSTR)2 )
        {
            sub_401266("WAN connection available");
        }
        else
        {
            sub_401266("Unknown connection available");
        }
    }
    else
    {
        sub_401266("No network connection");
    }

    hWind = ::hWind;
    color = v13 - v165 * (unsigned __int16)v14 - color + 1901;
    hdc = GetDC(::hWind);
    y = hIDObjItem + 2 * v6 - 1;
    color += v6 + 1;
    SetTextColor(hdc, 0xFF00u);
    SetBkColor(0, 0);
    SetBkMode(hdc, 2);
    lParam = v6 + 1;
    :::IDDlItem = (HGDIOBJ)(2 * (_DWORD)::nIDDlItem - 5);
    sub_401292(&v203, "WM_MOVE Called - New Position = (%d,%d)", (unsigned __int16)v13);

```

Fig 55. 네트워크 연결 탐지 기능

해당 악성코드가 토르 브라우저를 통해 사용자에게 접속을 유도하고 몇 가지 정보를 공유하는 기능을 가졌으므로, 사용자의 대상 PC 가 네트워크와 연결이 되어있는지 확인하는 작업이 우선적으로 필요했던 것 같다. 현재는 서버가 닫혀 있으므로 큰 의미는 없는 상태이다.

'Fig 56'는 GetSystemInfo 함수를 호출한 후 그 내부에서 볼륨 시리얼 넘버와 그 밖의 시스템 시간 정보 등을 추출하는 과정의 일부이다. 아직 정적 분석만으로 볼륨 시리얼 넘버가 어디에 사용되는지 정확히 파악할 수는 없지만, 해당 악성코드의 동작에 영향을 미치는 변수 중 하나로 이용되는 것으로 추정할 수 있다. 사용자의 PC의 디스크 장치에 따라 고유의 UUID가 부여되는데, 실제 악성코드 내부에서 이러한 정보를 추출하고 모종의 연산을 거쳐 구별하기 쉬운 ID 형태로 부여하는 것으로 추정되며, 이 역시 동적 분석을 통해 구체적인 과정을 살펴볼 수 있을 것이다.

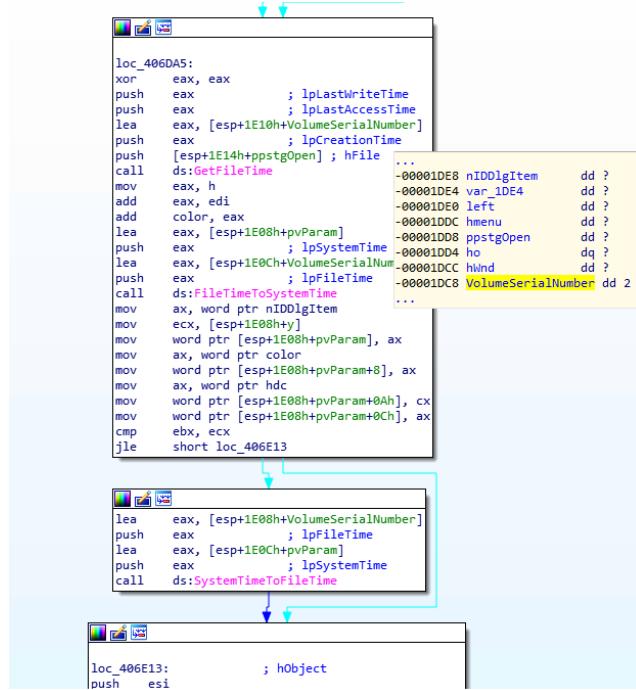


Fig 56. 볼륨 시리얼 넘버 추출 구문

4.3 동적 분석

악성코드를 실제 구동하여 어떠한 행위를 수행하는지를 관찰하기 위해 동적 분석을 실시하였다. 이를 통해 정적 분석만으로는 얻을 수 없는 정보를 획득할 수 있었고 일부 가설들에 대한 입증을 할 수 있었다.

4.3.1 가상머신 환경에서의 감염 실험

먼저 안전한 실험 환경을 구축하는 것이 필요하다. 해당 샘플은 실제로도 여전히 악성 행위를 하는 기능을 내포하고 있는 것이므로, 잘못하면 분석자의 PC가 감염될 우려가 있다. 본 실험에서는 가상 머신 소프트웨어인 VMware® Workstation 12 Pro를 이용하였으며, 실제 감염 PC의 조건과 최대한 비슷하게 구축하기 위해 MS Windows 7로 'Fig 57' 과 같이 구축하였다.

그리고 혹시 모를 사태를 방지하기 위하여 VMware 설정에서 네트워크 연결을 차단하고, 공유 폴더 등에 대한 설정을 모두 해제하였다. 이는 Guest 환경에서의 감염이 Host PC까지 전이되는 것을 방지하기 위함이다.

또한, 반복적인 작업 수행을 지속하여야 하므로, Snapshot 기능을 활용하였다. 예를 들어, 감염 당하기 이전 상태에서 Clean 스냅샷을 캡처하고, 랜섬웨어 파일을 실행하는 중간 단계와 모든 작업이 종료된 상태 모두 별도로 스냅샷을 지정해 두었다. 이렇게 함으로써 같은 작업을 반복할 때 분석의 효율성을 높였다.

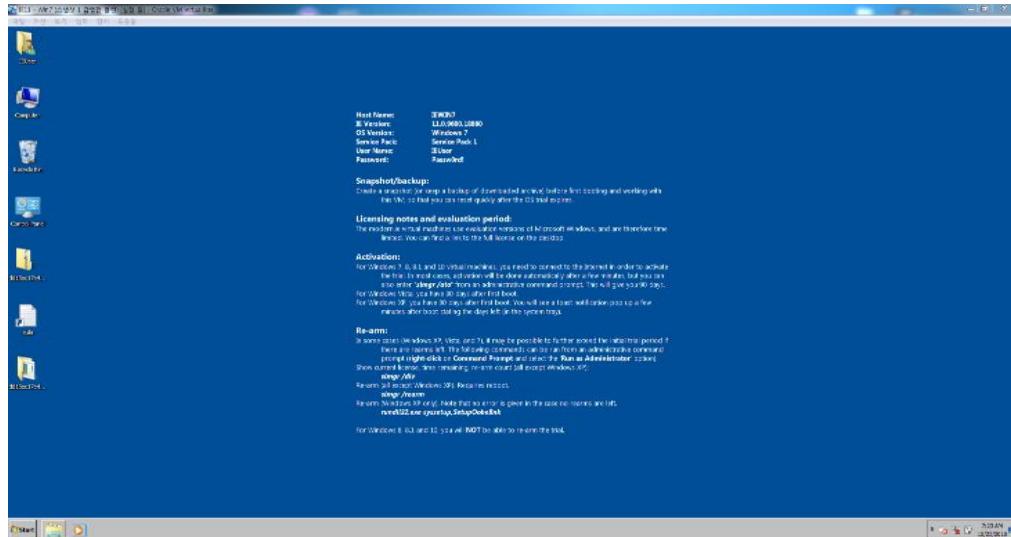


Fig 57. 감염되기 전 정상 부팅된 Win7 VM 환경

'Fig 58'은 가상 머신 내부에서 랜섬웨어 수행이 종료되었을 때의 화면이다.

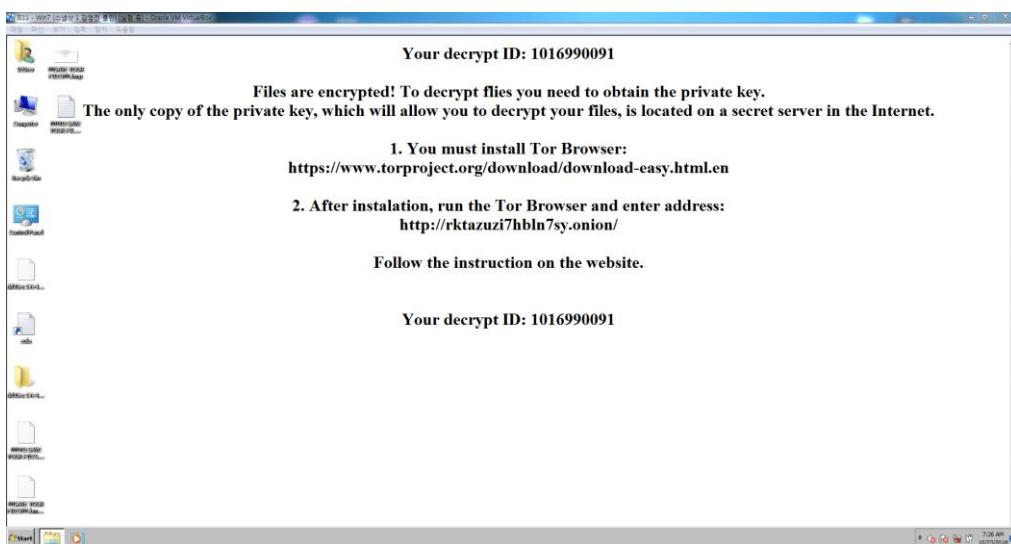


Fig 58. 감염된 후의 Win7 VM 환경

눈에 띄는 증상으로는, 바탕화면의 배경이 랜섬웨어 메시지로 변경되어 있고, 여기에는 Your decrypt ID라는 고유번호가 숫자 형식을 1016990091로 표기되고 있다. 이 번호는 같은 실험 환경에서 여러 번 테스트했을 때 매번 동일한 값이 표출되었으며, 다른 시스템에서 시행하면 또 다른 고유의 값이 지속 표출되었다. 여기에서 추론할 수 있는 사실은 감염 PC의 특정 고유 값을 seed로 사용하여 일련의 숫자 형태로 만드는 것이며 이 값은 동일한 장비에서는 계속 일치하게 표출 된다는 점이다. 이는 앞서 언급한 GetSystemInfo()에서의 볼륨 시리얼 넘버 등의 정보를 활용하였음으로 추정할 만한 합리적인 근거가 된다.

이어서 다양한 종류의 파일을 sample로 생성하여 시스템 내부에 위치해 두고, 악성코드의 영향을 받는지 확인하였다.

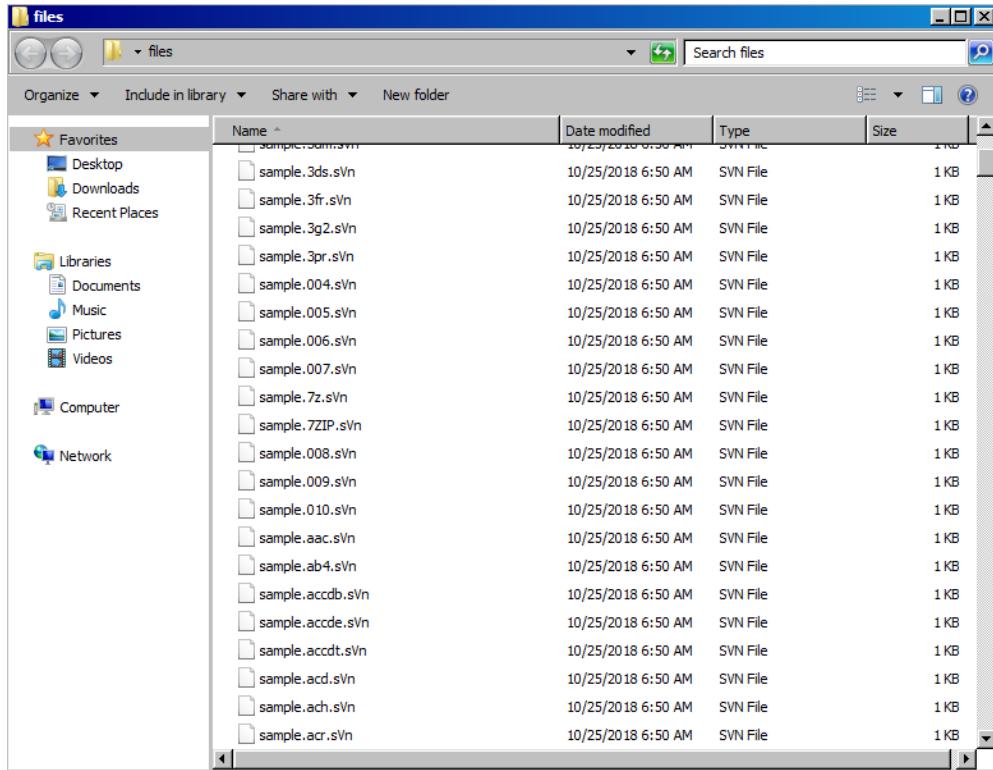


Fig 59. 감염된 파일들

대다수의 파일들이 암호화되었는데, 이 파일들은 기존의 이름의 끝에 *.svn 이라는 별도의 확장자가 첨부되어 있다. 당연히 이 내용들은 암호화 결과물이기 때문에 사용자는 기존의 파일 정보를 얻을 수 없고, 가용성을 심각히 침해당하게 되는 것을 관찰하였다.

4.3.2 디버거를 통한 동적 분석

가상 머신 내부에서 악성코드가 실행될 때 어떠한 행위를 하는지를 동적으로 분석하기 위해서 디버깅 도구를 활용하였다.

대부분의 악성코드들은 디버깅당하는 환경을 스스로 탐지하고 이를 강제로 종단시키는 등의 우회방법을 사용하고 있다. 본 프로그램도 일부 보호 기법이 적용되어 있으나, 백신 업계의 Jaff 랜섬웨어 관련 분석 문서를 참고하여 디버깅을 통한 동적 분석을 시도하였다. 먼저 'Fig 60'은 기존 정적 분석에서 발견할 수 없었던 인코딩 되었던 구문이다. Jaff 랜섬웨어는 우선 A부터 Z 드라이브 까지를 순회하며 암호화할 대상을 특정한다.

```

75855870 8BFF MOV EDI,EDI
75855872 55 PUSH EBP
75855873 8BEC MOV EBX,ESP
75855875 81EC SUB ESP,0x480
75855876 01 39028875 MOV EBX, DWORD PTR DS:[0x75800230]
75855880 33C5 XOR EBX,EBP
75855882 9945 FC MOV DWORD PTR SS:[EBP-0x4],EAX
75855885 53 PUSH EBX
75855886 56 PUSH ESI
75855887 8B35 0C120425 MOV ESI,DWORD PTR DS:[<0xntdll.RtlUpcaseUnicodeCI
7585588D 57 PUSH EDI
7585588E 8B7D 08 MOV EDI,DWORD PTR SS:[EBP+0x8]
75855891 89BD D8FBFFF MOV DWORD PTR SS:[EBP-0x28],EDI
75855897 85FF TEST EDI,EDI
75855899 75 1B JNZ SHORT KernelBa.758558B6
7585589A 809D F4FDFFF LEA EBX, DWORD PTR SS:[EBP-0x28C]
758558A1 8BC3 MOU EBX,EBX
758558A3 50 PUSH EAX
758558A4 68 08020000 PUSH 0x208
758558A9 FF15 DC138425 CALL DWORD PTR DS:[<0xntdll.RtlGetCurrentDirec
758558B2 83F8 06 CMW EBX,0x6
758558B4 76 71 JBE SHORT KernelBa.75855925
758558B4 EB 66 JME SHORT KernelBa.7585591C
758558B6 83FF FF CMP EDI,-0x1
ED1-7705E026 <ntdll.RtlAllocateHeap>

```

Fig 60. 암호화할 디스크 드라이브를 순회하는 구문

또한 특이한 점은, 'Fig 61'과 같이 총 3번에 걸쳐 API를 호출하여 GetUserDefaultUILanguage, GetUserDefaultLangID, GetSystemDefaultLangID 값들을 가져온다는 것이다. 그리고는 해당 레지스터의 값이 0x19인지 점검한다. 참고로 0x0019 값에 해당하는 언어는 러시아어인데, 특이하게도 이 경우에 해당되면 감염 작업을 더 이상 수행하지 않고 자기 자신을 종료한다. 이는 곧 러시아는 공격 대상이 아님을 암시하는데, 악성코드 제작자가 러시아인 이거나, 최소한 러시아에 대한 피해는 주지 않고 싶은 것으로 보인다.

```

004042C1 ? FF15 B4704000 CALL DWORD PTR DS:[0x4070B4]
004042C7 ? B9 FF030000 MOV ECX,0x3FF
004042CC ? 66:23C1 AND AX,CX
004042CF . 66:83F8 19 CMP AX,0x19
004042D3 ?~ 74 2D JE SHORT asd.00404302
004042D5 . 004042D5 B0704000 CALL DWORD PTR DS:[0x4070B0]
004042DB ? BA FF030000 MOV EDX,0x3FF
004042E0 . 66:23C2 AND AX,DX
004042E3 ? 66:83F8 19 CMP AX,0x19
004042E7 ?~ 74 19 JE SHORT asd.00404302
004042E9 ? FF15 0C704000 CALL DWORD PTR DS:[0x4070C1]
004042EF . B9 FF030000 MOV ECX,0x3FF
004042F4 ? 66:23C1 AND AX,CX
004042F7 ? 66:83F8 19 CMP AX,0x19
004042FB ?~ 74 05 JE SHORT asd.00404302
004042FD . E8 6E090000 CALL asd.00404C70

```

Fig 61. 대상 시스템의 언어가 러시아어인지 확인하는 구문

또한 안랩에서 분석한 Jaff 랜섬웨어의 특징에 따르면[19], 노트북 컴퓨터의 경우 배터리 절약 모드가 설정되는 상황이 있는데, 이를 체크하는 SystemStatusFlag 의 상태를 가져와서 만약 절약 모드가 1로 세팅되어 있다면, 감염을 진행하지 않고 종료한다. 이는 암호화가 수행되는 시간이 굉장히 오래 걸릴 경우 작업이 완료되기 이전에 전원 공급 불안정으로 인해 시스템이 강제로 종료되는 경우 암호화 작업 역시 불완전하게 중단되게 되고, 문제가 심각한 경우 파일 시스템에 전체적인 문제가 발생할 수 있다는 것을 염두 해 둔 것 같다.

이어서 Jaff 랜섬웨어는 자신이 암호화를 수행할 파일의 확장자 목록을 하나씩 호출한다. 이때 기존 바이너리에는 내용이 인코딩 되어 있어서 분별되지 않으나 실행 중에 XOR 연산을 통해 문자열의 내용이 하나씩 드러나면서 로드되는 방식이다. 'Fig 62' 은 427종의 확장자 목록을 보여주고 있다.

Address	Hex dump	ASCII		041CEF90	76A538FE	RETURN to kernel32.76A538FE from kernel32.76A53914
02F783FC	2E 00 74 00 78 00 74 00 00 00 2E 00 64 00 6F 00 ..r.x.t.....d.o.			041CEFP4	041CEFP8	
02F7840C	63 00 00 00 2E 00 77 00 62 00 6B 00 00 00 2E 00 c.....w.b.k.....			041CEFP8	76A73102	kernel32.lstrcpyW
02F7841C	6D 00 64 00 62 00 00 00 2E 00 76 00 63 00 66 00 n.d.b.....v.c.f.			041CEFP9C	00000030	
02F7842C	00 00 2E 00 64 00 6F 00 63 00 78 00 00 00 2E 00 ...d.o.c.x.....			041CEFP0	76A5390B	RETURN to kernel32.76A5390B from kernel32.76A51472
02F7843C	69 00 63 00 73 00 00 00 2E 00 76 00 73 00 63 00 i.c.s.....v.s.c.			041CEFP4	02F783FC	UNICODE ".txt"
02F7844C	00 00 2E 00 6D 00 64 00 66 00 00 00 2E 00 64 00 ...n.d.f....d.			041CEFP8	006240C8	
02F7845C	73 00 72 00 00 00 2E 00 6D 00 64 00 69 00 00 00 s.r.....m.d.i...			041CEFPAC	00000000	
02F7846C	2E 00 6D 00 73 00 67 00 00 00 2E 00 78 00 6C 00 ...n.s.g.....x.l...			041CEFPB0	00000000	
02F7847C	73 00 00 00 2E 00 70 00 70 00 00 00 2E 00 s.....p.p.t.....			041CEFB4	00000000	
02F7848C	78 00 70 00 73 00 00 00 2E 00 67 00 62 00 64 00 p.p.s.....o.b.d.			041CEFPB8	00000000	
02F7849C	00 00 2E 00 6D 00 70 00 64 00 00 00 2E 00 64 00 ...n.p.d....d.			041CEFPBC	00000000	
02F7840C	6F 00 24 00 00 00 2E 00 78 00 6C 00 74 00 00 00 o.t.....x.l.t...			041CEFC0	00000000	
02F784BC	2E 00 70 00 6F 00 74 00 00 00 2E 00 6F 00 62 00 ...p.o.t.....o.h.			041CEFC4	00000000	

Fig 62. 암호화할 확장자 목록

이 목록을 정리하여 아래 'Table 34'에 정렬하여 표기하였다.

'001', '002', '003', '004', '005', '006', '007', '008', '009', '010', '1cd', '3dm', '3ds', '3fr', '3g2', '3pr', '7ZIP', '7z', 'MPEG', 'aac', 'ab4', 'accdb', 'accde', 'accdt', 'acd', 'ach', 'acr', 'act', 'adb', 'adp', 'ads', 'agdl', 'ai', 'aif', 'aiff', 'ait', 'al', 'aoi', 'apj', 'arw', 'as4', 'ASF', 'asm', 'asp', 'asp', 'aspx', 'asx', 'avi', 'awg', 'back', 'backup', 'backupdb', 'bak', 'bank', 'bay', 'bdb', 'bgt', 'bik', 'bin', 'bcp', 'blend', 'bmp', 'bpw', 'c', 'cad', 'cbr', 'cdf', 'cdr', 'cdr3', 'cdr4', 'cdr5', 'cdr6', 'cdrw', 'cdx', 'ce1', 'ce2', 'cer', 'cfg', 'cgm', 'cib', 'class', 'cls', 'cmt', 'config', 'contact', 'cpi', 'cpp', 'cr2', 'craw', 'crt', 'crw', 'cs', 'csh', 'csl', 'css', 'csv', 'dac', 'dat', 'db', 'db3', 'db_journal', 'dbf', 'dbx', 'dc2', 'dcr', 'dcs', 'ddd', 'ddoc', 'ddrw', 'dds', 'deb', 'der', 'des', 'design', 'dgc', 'dit', 'djvu', 'dng', 'doc', 'docm', 'docx', 'dot', 'dotm', 'dotx', 'drf', 'drw', 'dsr', 'dtd', 'dwg', 'dxp', 'dxf', 'dgg', 'edb', 'eml', 'eps', 'erbsql', 'erd', 'exf', 'fdb', 'ffd', 'fff', 'fh', 'fhd', 'fif', 'fla', 'flac', 'flv', 'flvv', 'fp', 'fxg', 'gif', 'gray', 'grey', 'groups', 'gry', 'gz', 'gz', 'h', 'hbk', 'hdd', 'hdr', 'hpp', 'htm', 'html', 'ibank', 'ibd', 'ibz', 'ico', 'ics', 'idf', 'idx', 'iff', 'iif', 'iiq', 'incpas', 'indd', 'iso', 'java', 'jnt', 'jpe', 'jpeg', 'jpg', 'js', 'kc2', 'kdbx', 'kdc', 'key', 'kpdx', 'kwm', 'laccdb', 'lit', 'log', 'lua', 'm', 'm2ts', 'm3u', 'm4a', 'm4p', 'm4v', 'mapimail', 'max', 'mbx', 'md', 'mdb', 'mdc', 'mdf', 'mdi', 'mef', 'mfw', 'mid', 'mix', 'mkv', 'mlb', 'mmw', 'mny', 'moneywell', 'mos', 'mov', 'mp3', 'mp4', 'mpd', 'mpg', 'mrw', 'msg', 'myd', 'nd', 'ndd', 'ndf', 'nef', 'nk', 'nop', 'nrw', 'ns2', 'ns3', 'ns4', 'nsd', 'nsf', 'nsg', 'nsh', 'nvram', 'nwb', 'nx2', 'nxl', 'nyf', 'oab', 'obd', 'obj', 'obt', 'odb', 'odc', 'odf', 'odg', 'odm', 'odp', 'ods', 'odt', 'ogg', 'oil', 'ord', 'ost', 'otg', 'oth', 'otp', 'ots', 'ott', 'ova', 'p12', 'p7b', 'p7c', 'pab', 'pages', 'par', 'pas', 'pat', 'pcd', 'pct', 'pdb', 'pdd', 'pdf', 'pef', 'pem', 'pfx', 'php', 'pif', 'pl', 'plc', 'plus_muhd', 'png', 'pot', 'potm', 'potx', 'ppam', 'pps', 'ppsm', 'ppsx', 'ppt', 'pptm', 'pptx', 'prf', 'prn', 'ps', 'psafe3', 'psd', 'pspimage', 'pst', 'ptx', 'pub', 'pwm', 'py', 'qba', 'qbb', 'qbm', 'qbr', 'qbw', 'qbx', 'qby', 'qcow', 'qcow2', 'qed', 'r3d', 'raf', 'rar', 'rat', 'raw', 'rdbrwl', 'rm', 'rpm', 'rtf', 'rvt', 'rw2', 'rwz', 's3db', 'safe', 'sas7bdat', 'sav', 'save', 'say', 'sd0', 'sda', 'sdf', 'sitx', 'sldm', 'sldx', 'sql', 'sqlite', 'sqlite3', 'sqlitedb', 'sr', 'srf', 'srt', 'srw', 'st4', 'st5', 'st6', 'st7', 'st8', 'stc', 'std', 'sti', 'stl', 'stm', 'stw', 'stx', 'svg', 'swf', 'swm', 'sxc', 'sxd', 'sxn', 'sxi', 'sxm', 'sxw', 'tar', 'tar', 'tex', 'tga', 'thm', 'tib', 'tif', 'tlg', 'txt', 'vbox', 'vcf', 'vdi', 'veg', 'vhdx', 'vib', 'vmdk', 'vmsd', 'vmx', 'vmxf', 'vob', 'vsc', 'vsd', 'wab', 'wad', 'wallet', 'wav', 'waw', 'wb2', 'wbk', 'wda', 'wma', 'wmv', 'wpd', 'wps', 'x11', 'x3f', 'xis', 'xla', 'xlam', 'xlk', 'xlm', 'xls', 'xlsb', 'xlm', 'xlsx', 'xlt', 'xltm', 'xlw', 'xml', 'xmod', 'ycbcra', 'zip', 'zipx', 'zpf'
--

Table 34. 암호화 대상 확장자 정보

이어서 악성코드는 대상 디스크를 순회하며 파일 목록을 가져온 후 'Table 34'의 확장자 정보와 대조하여 해당되는 포맷의 파일인 경우 CryptEncrypt API 함수를 통해 파일 암호화를 수행한다.

암호화를 모두 마친 후에는 바탕화면의 이미지를 랜섬웨어 메시지로 교체하는 작업을 수행한다.

0018D158	0018D17C	return to user32.75064AAC from user32.75064A26
0018D15C	75064AAC	L"C:\\\\ProgramData\\\\Rondo\\\\Wallpaper.bmp"
0018D160	00000000	
0018D164	0018D1CC	
0018D168	00000000	
0018D16C	00000000	
0018D170	00020810	
0018D174	0018D5F8	
0018D178	00000000	
0018D17C	0018D5E0	
0018D180	7509FB81	return to user32.7509FB81 from user32.7505FBFC
0018D184	00000000	
0018D188	0018D1CC	L"C:\\\\ProgramData\\\\Rondo\\\\Wallpaper.bmp"
0018D18C	00000000	
0018D190	00000000	
0018D194	00000000	
0018D198	00020810	
0018D19C	00000000	

Fig 63. 바탕화면 변경을 위한 이미지 파일 생성

'Fig 63'는 바탕화면 변경을 수행하는 과정을 보여주고 있는데, "C:/ProgramData/사용자ID" 하위의 디렉토리에 Wallpaper.bmp라는 파일을 생성한다. 그리고 해당 이미지가 'Fig 64'처럼 바탕화면 배경으로 변경된다.

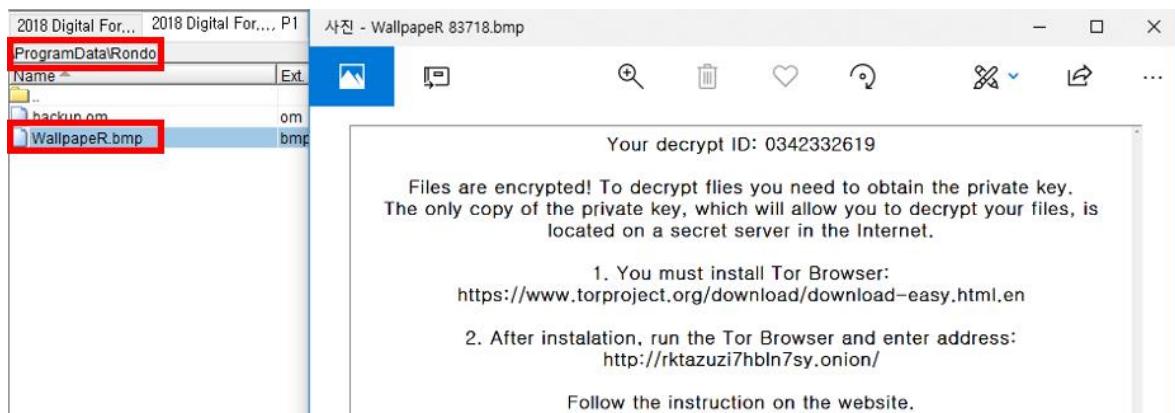


Fig 64. 랜섬웨어 메시지로 변경된 바탕화면 이미지

한편 사용자의 decrypt ID 부분의 고유 숫자 정보는 암호화 대상 드라이브의 볼륨 일련번호를 이용하는 것으로 확인되었다. 아래 'Fig 65'를 보면 일련번호 C208-7544인 C드라이브에서 감염된 경우 decrypt ID가 3255334212로 표기되며 이는 동일한 숫자에 대한 각각 16진수와 10진수 표기 법인 것으로 확인하였다.



Fig 65. Decrypt ID 값 확인

바탕화면 변경 작업까지 마무리되면 랜섬웨어는 자신의 모든 임무를 완수한 후 마지막으로 자기 자신을 삭제하는 cmd 명령어를 'Fig 66'와 같이 호출하고 종료된다.

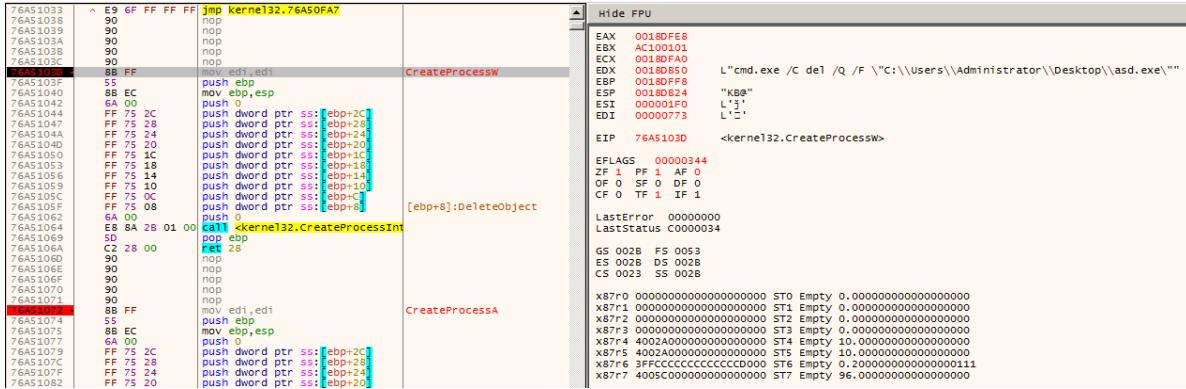


Fig 66. 작업 완료 후 자기 자신을 삭제하는 cmd 명령어 호출

동적 분석을 통해 핵심 과정을 관찰할 수는 있었지만, 실제로 해당 프로그램에 의해 시스템에 전역적으로 발생하는 모든 이벤트를 수동으로 찾기에는 한계가 있었다. 이에 다음 방법으로 메모리 포렌식 분석을 이용하기로 하였다.

4.3.3 활성 메모리 분석

메모리 포렌식 분석으로는 오픈소스 도구인 Volatility를 활용하였다. 이미 앞서 VMware 환경에서 스냅샷을 캡처해두었는데, Snapshot 파일은 그 자체로 VM에 대한 Virtual Memory Dump로써 활용 가능하다.

우선 imageinfo 플러그인을 이용하여 얻은 메모리 덤프의 정보는 'Table 35'과 같다.

Name	Description
KDBG	0xf80002a08110L
DTB	0x187000L
KUSER_SHARED_DATA	0xfffff78000000000L
Suggested Profile(s)	Win7SP1x86, Win7SP0x86, Win7SP1x86_23418
Image Type	1
Number of Processors	1
KPCR for CPU 0	0xfffff80002a09d00L
Image date and time	2018-10-25 04:11:17 UTC+0000
Image local date time	2018-10-24 21:11:17 -0700
PAE type	No PAE
AS Layer1	WindowsPagedMemory (Kernel AS)
AS Layer2	FileAddressSpace (/hdd1/df/memory/IEWIN7-ing.raw)

Table 35. imageinfo 플러그인

가장 중요한 정보는 Suggested Profile인데, Win 7에서 자체 구축하였으므로 이미 Win7SP1x86임을 알고 있는 상황이다. 이후에 진행될 모든 플러그인에 프로파일 지정이 선행되어야 하며, --profile=Win7SP1x86의 형태로 입력하였다.

아래 'Fig 67' 은 악성코드가 실행된 후 추가적인 프로세스를 생성하는지 여부를 보여주고 있으며, 다음 명령어로 확인한 내용 중 불필요한 다른 프로세스를 생략하고 요약하였다.

```
python vol.py -f /hdd1/df/memory/IEWIN7-ing.raw --profile=Win7SP1x86 pslist
```

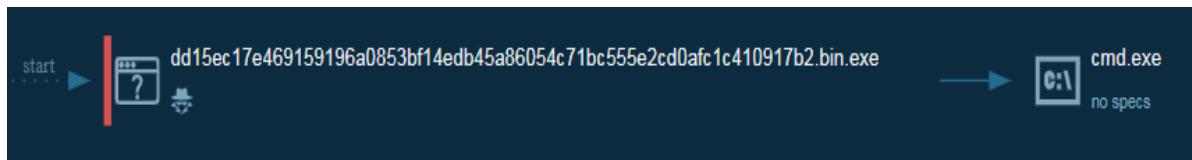


Fig 67. 프로세스 도식화

2692번 프로세스의 악성 행위를 구체적으로 파악하기 위해 malfind 플러그인을 통해 확인해보았다.

```
python vol.py -f /hdd1/df/memory/IEWIN7-ing.raw --profile=Win7SP1x86 malfind -p 2692
```

```
Volatility Foundation Volatility Framework 2.6
Process: dd15ec17e46915 Pid: 2692 Address: 0x29a0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 40, PrivateMemory: 1, Protection: 6

0x029a0000 c5 e5 5b e9 72 04 00 01 ee ff ee ff 00 00 00 00 00 ...[.r.....
0x029a0010 a8 00 9a 02 a8 00 9a 02 00 00 9a 02 00 00 9a 02 .....v...
0x029a0020 9e 00 00 00 88 05 9a 02 00 e0 a3 02 76 00 00 00 .....v...
0x029a0030 01 00 00 00 00 00 f0 7f 9c 02 f0 7f 9c 02 .....v...

0x029a0000 c5          DB 0xc5
0x029a0001 e55b       IN EAX, 0x5b
0x029a0003 e972040001 JMP 0x39a047a
0x029a0008 ee          OUT DX, AL
0x029a0009 ff          DB 0xff
0x029a000a ee          OUT DX, AL
0x029a000b ff00       INC DWORD [EAX]
0x029a000d 0000       ADD [EAX], AL
0x029a000f 00a8009a02a8 ADD [EAX-0x57fd6600], CH
0x029a0015 009a0200009a ADD [EDX-0x65fffffe], BL
0x029a001b 0200       ADD AL, [EAX]
0x029a001d 009a029e0000 ADD [EDX+0x9e02], BL
0x029a0023 0088059a0200 ADD [EAX+0x29a05], CL
0x029a0029 e0a3       LOOPNZ 0x299ffce
0x029a002b 027600     ADD DH, [ESI+0x0]
0x029a002e 0000       ADD [EAX], AL
0x029a0030 0100       ADD [EAX], EAX
0x029a0032 0000       ADD [EAX], AL
0x029a0034 0000       ADD [EAX], AL
0x029a0036 0000       ADD [EAX], AL
0x029a0038 f07f9c    JG 0x299ffd7
0x029a003b 02f0       ADD DH, AL
0x029a003d 7f9c       JG 0x299ffdb
0x029a003f 02          DB 0x2

sep@siftworkstation -> ~/volatility
```

Fig 68. malfind 플러그인을 통한 인젝션 코드 확인

이 과정에서 랜섬웨어의 메모리 영역 0x29a0000의 주소에서 특정 코드가 인젝션 된 것이 발견되었다.

그리고 암호화 관련 API 를 호출하는데 필요한 crypt 관련 dll 파일들이 해당 프로세스에서 사용되는 것 또한 확인하였다.

```
python vol.py -f /hdd1/df/memory/IEWIN7-ing.raw --profile=Win7SP1x86 ldrmodules
-p 2692 |grep crypt
```

Volatility Foundation Volatility Framework 2.6						
2692	dd15ec17e46915	0x00000000752a0000	True	True	True	\Windows\SysWOW64\cryptbase.dll
2692	dd15ec17e46915	0x0000000074dd0000	True	True	True	\Windows\SysWOW64\cryptsp.dll
2692	dd15ec17e46915	0x0000000076760000	True	True	True	\Windows\SysWOW64\crypt32.dll

Fig 69. ldrmodules 플러그인을 통해 암호화 관련 라이브러리 확인

이러한 일련의 악성 행위를 마치면 프로세스는 자기 자신을 종료하면서 cmd.exe라는 프로세스를 추가적으로 한번 더 생성한다. 그리고 해당 cmd는 아래와 같은 명령어를 수행하는 것으로 확인하였다.

```
cmd.exe /C del /Q /F
"C:/Users/admin/AppData/Local/Temp/dd15ec17e469159196a0853bf14edb45a86054c71b
c555e2cd0afc1c410917b2.bin.exe"
```

이는 곧 자기를 호출한 악성코드의 원본 바이너리를 자가 삭제하는 명령어이다. 대부분의 악성코드들이 흔적을 지우기 위해 사용하는 방법과 유사하다.

또한 아래와 같이 netscan 명령어를 통해 악성코드가 네트워크 관련 행위를 수행하는지를 점검할 수 있었으나, 의도적으로 네트워크를 단절시킨 상황에서 테스트했으므로 정확한 결과를 얻을 수 없었다. 하지만 실제로도 해커의 url 자체가 막혀있는 상황이어서, DNS 질의는 성공할 수 없었을 것으로 판단된다.

```
python vol.py -f /hdd1/df/memory/IEWIN7-ing.raw --profile=Win7SP1x86 netscan
```

결과 : DNS Requests : toronadrouuyrt5wwf.com (응답받지 못함)

또한 파일 관련 조작이 2000회 이상 발생하였는데, 이는 랜섬웨어에 의한 모든 파일 변경으로 인한 것이다. 앞서 'Table 34'에서 동적 분석을 통해 암호화 대상 확장자 목록을 확인하였는데, 실제 랜섬웨어 프로세스의 메모리를 덤프하여 확인한 'Fig 70' 와도 정확히 일치하는 목록들을 추출함으로써 교차 검증할 수 있었다.

```

00b03bd0: 1a4a ac04 0a2e 0078 006c 0073 0078 0000 .J.....x.l.s.x..
00b03be0: 002e 0061 0063 0064 0000 002e 0070 0064 ...a.c.d.....p.d
00b03bf0: 0066 0000 002e 0070 0066 0078 0000 002e .f.....p.f.x....
00b03c00: 0063 0072 0074 0000 002e 0064 0065 0072 .c.r.t.....d.e.r
00b03c10: 0000 002e 0063 0061 0064 0000 002e 0064 .....c.a.d.....d
00b03c20: 0077 0067 0000 002e 004d 0050 0045 0047 .w.g.....M.P.E.G
00b03c30: 0000 002e 0072 0061 0072 0000 002e 0076 .....r.a.r.....v
00b03c40: 0065 0067 0000 002e 007a 0069 0070 0000 .e.g.....z.i.p..
00b03c50: 002e 0074 0078 0074 0000 002e 0064 006f ...t.x.t.....d.o
00b03c60: 0063 0000 002e 0077 0062 006b 0000 002e .c.....w.b.k.....
00b03c70: 006d 0064 0062 0000 002e 0076 0063 0066 .m.d.b.....v.c.f
00b03c80: 0000 002e 0064 006f 0063 0078 0000 002e .....d.o.c.x....
00b03c90: 0069 0063 0073 0000 002e 0076 0073 0063 .i.c.s.....v.s.c
00b03ca0: 0000 002e 006d 0064 0066 0000 002e 0064 .....m.d.f.....d
00b03cb0: 0072 0072 0000 002e 0061 0060 0000 .....m.d.f.....d

```

Fig 70. 랜섬웨어 프로세스 메모리 덤프에서 발견된 확장자 목록

메모리 분석을 통해 마지막으로 발견한 유의미한 정보로는 Windows의 핵심 아티팩트라고 할 수 있는 레지스트리 변화 정보이다. Volatility의 hivescan 과 hivelist 플러그인을 통해 정보를 획득할 수 있었으며, 핵심 내용만 요약하면 아래 'Table 36'과 같다.

Key	Name	Value
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion /Internet Settings/ZoneMap	UNCAsIntranet	0
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion /Internet Settings/ZoneMap	AutoDetect	1
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASAPI32	EnableFileTracing	0
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASAPI32	EnableConsoleTracing	0
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASAPI32	FileTracingMask	4294901760
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASAPI32	ConsoleTracingMask	4294901760
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASAPI32	MaxFileSize	1048576
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASAPI32	FileDirectory	%windir%\tracing
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASMNCs	EnableFileTracing	0
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASMNCs	EnableConsoleTracing	0
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\RASMNCs	FileTracingMask	42949017

	ask	60
HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Tracing/RASMANCS	ConsoleTracingMask	4294901760
HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Tracing/RASMANCS	MaxFileSize	1048576
HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Tracing/RASMANCS	FileDirectory	%windir%/tracing
HKEY_CURRENT_USER/Software/Microsoft/Windows/CurrentVersion/Internet Settings	ProxyEnable	0

Table 36. 레지스트리 변화 정보 (요약)

변경된 레지스트리 관련 Key 값들이 의미하는 바와 지금까지 파악한 동적 행위들을 토대로 악성 코드가 시스템에 가한 행위를 아래와 같이 종합 요약할 수 있다.

- 랜섬웨어 명령을 수행하여 다수의 파일에 대한 읽기/쓰기 작업 수행
- 크롬 및 인터넷 익스플로러 브라우저의 환경설정 파악
- 인터넷 캐쉬 환경설정 파악
- 시스템 관련 디렉토리 임의 수정
- 파일 및 콘솔에 대한 tracing 정보 임의 수정
- Tor URL 이 포함되어 있는 파일 생성
- CMD.exe 프로세스 호출

4.4 파일 암호화 작동 방식 분석

암호학은 기본적으로 정보보호의 3요소 중 데이터의 기밀성을 유지하기 위해 사용하는 기술이다. 그러나 정당하지 않은 권한에 의해 잘못 오남용 될 경우 오히려 가용성을 침해하는 무기가 되기도 한다. 랜섬웨어가 바로 그러한 전형적인 사례이며, 사용자가 원하지 않는 암호화를 강제로 적용하여 피해를 입힌다.

고전적인 암호 방식에서는 암호화하는 방식 자체를 숨기는 방식을 사용했는데, 현대에는 암호화 알고리즘의 설계가 모두 공개되어 있더라도 Key의 안전성만 유지된다면 보안을 확보할 수 있는 방법을 사용한다. 이러한 원리를 Kirchhoff의 원리라고 하는데, 오늘날 사용되는 대부분의 암호화 알고리즘은 NIST 등의 표준기관의 승인을 거치고 오랜 시간 동안 암호학 연구자들에 의해 공격이 시도되었음에도 Key에 대한 충분한 안전성이 확보되었다. 가장 대표적인 방식은 AES 대칭키 암호화 방식과 RSA 공개키 방식이 있다. 본 랜섬웨어는 이 두 가지 방식을 혼합한 하이브리드 암호화 방식을 사용하는 것으로 알려져 있다.

따라서 본 랜섬웨어에 의해 암호화된 파일들을 직접 복원하기 위해 암호 알고리즘 자체를 직접 분석할 필요는 없다. AES 및 RSA 알고리즘의 복호화 방법 자체는 이미 표준으로 공개되어 있으므로, Key 만 찾아낼 수 있다면 파일을 복구할 수 있다. 이어지는 장에서 해당 랜섬웨어의 암호화 과정을 분석하고, Key 가 어떻게 생성되어 처리되는지를 확인하여 반대로 Key 를 추출할 수 있는 방안을 모색하였다.

4.4.1 AES-256 대칭키 생성 및 파일 암호화

먼저 Ahnlab의 보고서[19]에 따르면, Jaff Ransomware의 암호화 과정은 'Fig 71' 의 방식으로 도식화할 수 있다.

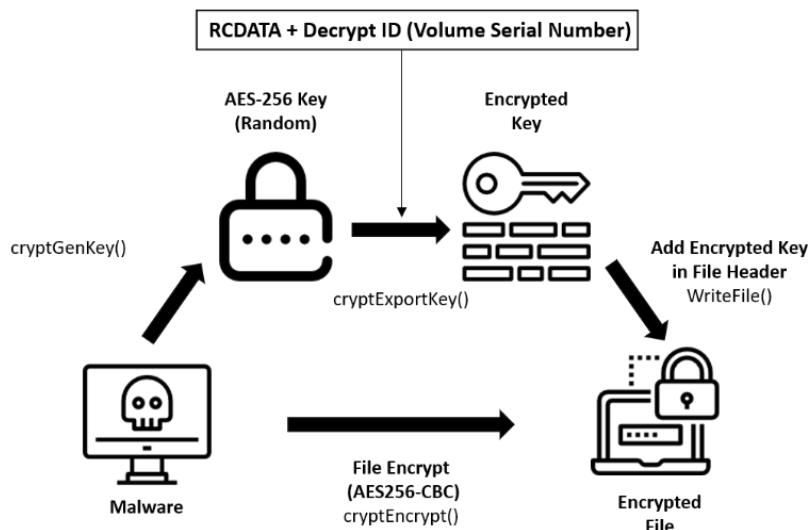


Fig 71. 랜섬웨어 암호화 절차 도식화

Jaff 랜섬웨어는 우선 AES-256 알고리즘을 사용하기 위해 필요한 키를 생성한다. 이때 암호화 관련 라이브러리는 악성코드 제작자가 직접 구현한 것이 아니라, 이미 구현되어 있는 라이브러리를 활용하였는데, 아래와 같이 랜섬웨어 프로세스에서 string 정보를 추출함으로써 파악할 수 있었다.

```
Strings dd15ec17e469159196a0853bf14edb45a86054c71bc555e2cd0afc1c410917b2.bin.DMP! grep
Crypto
```

그 결과 얻어지는 문자열 중 아래와 같은 정보가 단서가 되었다.

- Microsoft Base Cryptographic Provider v1.0
- Microsoft Enhanced RSA and AES Cryptographic Provider

구글 검색을 통해 'Fig 72' 와 같이 Microsoft에서 지원하는 관련 라이브러리 Documents 페이지에 접근할 수 있었다. 여기에서는 실제 암호화 관련 라이브러리에 대한 스펙과 각 알고리즘을 사용

할 수 있는 예제 코드를 제공하고 있다. 보통 암호화 라이브러리로는 OpenSSL이 많이 사용되는 경향이 있으나 본 랜섬웨어는 MS Windows 시스템을 주 타깃으로 삼으며 윈도우 관련 라이브러리를 최대한 활용한 것으로 보인다.

The screenshot shows the Microsoft Windows Dev Center page for the Microsoft AES Cryptographic Provider. The URL is [Docs / Windows / Desktop / Cryptography / About Cryptography / Cryptographic Service Providers / Microsoft Cryptographic Service Providers / Microsoft AES Cryptographic Provider](https://docs.microsoft.com/windows/desktop/Cryptography/Cryptographic-Service-Providers/Microsoft-Cryptographic-Service-Providers/Microsoft-AES-Cryptographic-Provider). The page title is "Microsoft AES Cryptographic Provider". It includes a sidebar with navigation links for Microsoft Cryptographic Service Providers, Microsoft Base Cryptographic Provider, Microsoft Strong Cryptographic Provider, Microsoft Enhanced Cryptographic Provider, and Microsoft AES Cryptographic Provider. The main content area contains a summary of the provider's capabilities, provider type (PROV_RSA_AES), and provider name (MS_ENH_RSA_AES_PROV). The page is dated 05/31/2018 and has a reading time of 2 minutes.

Fig 72. 마이크로소프트에서 제공하는 암호화 라이브러리 문서[20]

해당 문서 내용 중 ALG_ID 라는 항목에 대한 설명을 보면, 암호화 알고리즘을 결정할 때 'Table 37'과 같은 값을 기준으로 구분한다.

Identifier	Value	Description
CALG_3DES	0x000006603	Triple DES encryption algorithm.
CALG_3DES_112	0x000006609	Two-key triple DES encryption.
CALG_AES	0x000006611	Advanced Encryption Standard (AES).
CALG_AES_128	0x00000660e	128 bit AES.
CALG_AES_192	0x00000660f	192 bit AES.
CALG_AES_256	0x000006610	256 bit AES.

Table 37. 암호화 알고리즘 선택 시 사용하는 값 예시

그리고 실제 동적 분석을 통해 해당 함수의 인수로 전달된 값을 보면 'Fig 73' 와 같이 0x6610 이므로 AES-256 알고리즘을 통해 암호화를 수행한다는 것을 알 수 있다.

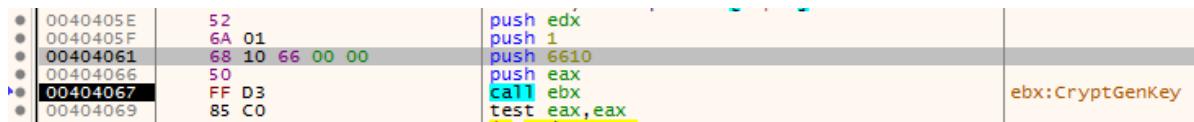


Fig 73. CryptGenKey API의 CryptID

랜섬웨어가 Microsoft Cryptographic Service Providers 에서 지원하는 AES 암호화 API를 활용했을 것으로 가정할 수 있다. 본 암호화 알고리즘 API의 작동원리는 아래와 같은 순서를 따른다.

1. CryptGenKey() 함수를 이용하여 세션 키를 생성한다. 이 방식은 사용자에게 패스워드를 입력하도록 하는 방식과 무작위로 생성하는 방법이 있는데, 여기에서는 랜덤한 방식을

채택하였다.

2. CryptAcquireContextW() 함수를 이용하여 암호화에 사용할 모드와 초기화 벡터 등의 요소를 지정한다.
3. CryptEncrypt() 함수를 이용하여 파일 데이터를 암호화한다. 이때 1에서 생성한 세션키를 이용하여 대칭키 방식으로 빠르게 암호화한다. 이때 블록 암호 알고리즘의 특성상 암호화되는 블록의 크기를 균일하게 조정해야 할 필요가 있는데, 이때 패딩 등의 방법이 동원된다. 그래서 암호화된 결과물은 원본과 비교했을 때 사이즈가 다소 변동될 우려가 있는데, 추후 복호화를 안정적으로 하기 위해서는 해당 데이터의 크기에 대한 정보를 반드시 기억해 두어야 한다.
4. 추후 복호화에 사용하기 위해서 1에서 랜덤하게 생성된 세션 키 관련 정보를 별도로 기록해 두어야 할 필요가 있다. 이때 CryptExportKey() 함수가 사용된다. 본 함수는 복호화에 사용할 세션 키를 다시 공개키 알고리즘으로 암호화하여 일명 key BLOB 형태로 저장하는데, 이를 해제하기 위해서는 별도의 개인키가 요구된다.

이를 도식화한 그림이 'Fig 74'이다. 결론적으로 요약하면 파일을 암호화할 때는 대칭키 알고리즘인 AES 알고리즘을 사용하여 고속으로 처리하고, 그 대칭키를 다시 한번 암호화하기 위해 다른 종류의 알고리즘인 RSA 공개키 방식을 이용한다. 이 알고리즘은 비교적 느린 방식이지만 1회만 수행하기 때문에 효율성 측면에서 우수한 하이브리드 방식이다.

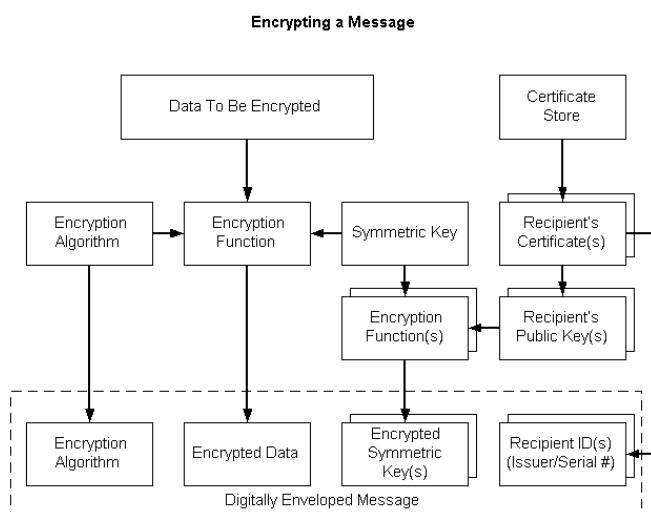


Fig 74. 암호화 과정 도식화

4.4.2 AES 키에 대한 RSA 암호화(하이브리드 방식)

'Fig 75'는 API를 통해 하이브리드 암호화를 수행하는 소스코드의 일부분이다.

```

if (CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, NULL, &dwKeyBlobLen)) {
    _tprintf(TEXT("The key BLOB is %d bytes long. %n"), dwKeyBlobLen);
}

```

```

} else {
    MyHandleError(TEXT("Error computing BLOB length! \n"), GetLastError());
    goto Exit_MyEncryptFile;
}

if (pbKeyBlob = (BYTE * ) malloc(dwKeyBlobLen)) {
    _tprintf(TEXT("Memory is allocated for the key BLOB. \n"));
} else {
    MyHandleError(TEXT("Out of memory. \n"), E_OUTOFMEMORY);
    goto Exit_MyEncryptFile;
}

if (CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, pbKeyBlob, &dwKeyBlobLen)) {
    _tprintf(TEXT("The key has been exported. \n"));
} else {
    MyHandleError(TEXT("Error during CryptExportKey!\n"), GetLastError());
    goto Exit_MyEncryptFile;
}

if (hXchgKey) {
    if (!(CryptDestroyKey(hXchgKey))) {
        MyHandleError(TEXT("Error during CryptDestroyKey.\n"), GetLastError());
        goto Exit_MyEncryptFile;
    }
}

hXchgKey = 0;
}

```

Fig 75. 하이브리드 암호화 과정의 C 소스코드 예시

이 코드에 따르면 공개키를 통해 session key를 암호화하고, 그 결과를 pbKeyBlob에 저장한다. 그러기 위해 key BLOB의 사이즈를 계산하고 적절한 메모리 공간을 할당한다. 그 뒤 session key를 public key를 이용해서 암호화한 후 그 결과물을 pbKeyBlob에 저장하였다. 이 작업이 완료되면 hKey와 hXchgKey를 destroy하는 함수를 호출하여 메모리 공간상에서 해제해버린다.

4.4.3 암호화된 Key Data를 파일에 저장하는 방식

'Fig 76'은 암호화된 Key Data인 일명 KEY BLOB 정보를 파일에 저장하는 과정을 나타내고 있다.

```

if (!WriteFile(hDestinationFile, &dwKeyBlobLen, sizeof(DWORD), &dwCount, NULL)) {
    MyHandleError(TEXT("Error writing header.\n"), GetLastError());
    goto Exit_MyEncryptFile;
}

}

```

```

_tprintf(TEXT("A file header has been written. \n"));
}

if (!WriteFile(hDestinationFile, pbKeyBlob, dwKeyBlobLen, &dwCount, NULL)) {
    MyHandleError(TEXT("Error writing header.\n"), GetLastError());
    goto Exit_MyEncryptFile;
} else {
    _tprintf(TEXT("The key BLOB has been written to the ") TEXT("file. \n"));
}

free(pbKeyBlob);

```

Fig 76. Key Blob Data 저장 과정 C 소스코드 예시

WriteFile 함수는 목적 파일에 내용을 기록한다. 가장 먼저 key BLOB 데이터의 크기를 기록하고, 이후 pbKeyBlob 데이터를 기록하고 있다. 이 작업이 완료되면 메모리 상에서 pbKeyBlob를 해제하여 찾을 수 없도록 만든다.

4.4.4 파일 암호화 방식

'Fig 77'는 키를 이용하여 파일들을 암호화하는 부분을 보여주고 있다.

```

bool fEOF = FALSE;
do {
    if (!ReadFile(hSourceFile, pbBuffer, dwBlockLen, &dwCount, NULL))
    {
        MyHandleError(TEXT("Error reading plaintext!\n"), GetLastError());
        goto Exit_MyEncryptFile;
    }
    if (dwCount < dwBlockLen)
    {
        fEOF = TRUE;
    }
    if (!CryptEncrypt(hKey, NULL, fEOF, 0, pbBuffer, &dwCount, dwBufferLen))
    {
        MyHandleError(TEXT("Error during CryptEncrypt. \n"), GetLastError());
        goto Exit_MyEncryptFile;
    }

    if (!WriteFile(hDestinationFile, pbBuffer, dwCount, &dwCount, NULL))
    {
        MyHandleError(TEXT("Error writing ciphertext.\n"), GetLastError());
        goto Exit_MyEncryptFile;
    }
}

```

```

    }
} while (!fEOF);

```

Fig 77. 파일 암호화 과정 C 소스코드 예시

먼저 ReadFile 함수를 통해 암호화할 파일의 내용을 버퍼에 저장한 후, 이어 CryptEncrypt 함수에서 hKey(AES-256 세션키)를 통해 암호화 작업을 수행한다. 이 암호화는 메모리 영역에서 이루어지는 것이며, 암호화된 내용을 WriteFile 함수를 통해 DestinationFile에 저장한다. 이 과정을 do-while 루프를 이용하여 지속 반복하며, 암호화가 완료되면 해당 버퍼 메모리를 비우고, 기존 파일도 삭제함으로써 사용자가 원본 파일에 접근할 수 없게 만든다.

이로써 Microsoft Cryptographic Service Providers 를 사용하여 파일을 암호화한 랜섬웨어의 행위를 파악할 수 있었다. 본 API는 여러 가지 알고리즘을 지원하지만, Jaff 랜섬웨어는 대칭키 암호화에서는 AES-256의 CBC 모드를 채택하였으며, 공개키 암호화 과정에서는 RSA 알고리즘을 선택하였다는 사실을 전제로 하여 복호화 방안을 마련해 보자.

4.5 파일 복호화 방안

기본적으로 복호화는 암호화 과정의 역순으로 진행한다. 전체적으로 Key 가 생성되고 어디에 저장되는지를 확인하였으므로 그 역순으로 진행해보고자 한다.

4.5.1 Encrypted Key Data 복원 방안

파일을 복호화하기 위해서는 먼저 공개키 암호화를 해제할 수 있는 개인키가 필요하고, 그 개인키를 통해 Key Blob 데이터를 복호화하면 session key를 얻을 수 있다. 이 세션 키가 바로 파일 복호화를 할 수 있는 열쇠이다.

우선 암호화된 파일을 Hex 에디터로 열어보면 원본에는 없었던 블록이 추가적으로 첨가되어서 파일의 크기가 살짝 늘어난 것을 알 수 있다.

'Fig 78' 은 파일 Head 부분에 262 Bytes의 내용을 보여주고 있다. 여기에서 맨 앞 4 Bytes는 Encrypted Key Data의 길이를 나타내는데, 02010000으로 표기된 내용을 리틀 엔디언 기준의 16진수로 표현하면 0x00000102이고, 이는 10진수로 258이 된다. 따라서 그 뒤에 연달아 나오는 258 Bytes의 데이터가 바로 Key BLOB 데이터이며, session 키가 공개키로 암호화된 값이다.

리틀 엔디안(little endian) 기준 → 0x00000102 → 258	
00000000h:	02 01 00 00 32 35 31 33 38 32 35 30 35 33 32 36 ; ... 251302505326
00000010h:	56 35 30 30 35 35 39 33 34 32 38 39 38 32 32 ; 6500559342890822
00000020h:	33 35 35 38 35 31 36 32 37 39 30 30 35 39 38 31 ; 358516279005981
00000030h:	30 36 36 35 30 33 33 33 30 38 30 30 30 30 33 36 ; 8665033308000036
00000040h:	38 34 32 35 35 37 33 38 35 36 33 37 31 35 37 31 ; 8425573856371571
00000050h:	39 33 31 37 37 36 31 33 33 36 30 36 35 36 33 34 ; 9317761336656534
00000060h:	34 39 39 38 31 33 38 31 38 34 34 31 33 32 33 31 ; 4998138184413231
00000070h:	31 37 30 38 36 34 38 39 35 31 31 32 32 36 39 38 ; 1788640951122698
00000080h:	33 38 35 39 20 31 34 38 34 39 31 31 33 35 36 36 ; 3859 14849113560
00000090h:	34 36 31 33 33 32 36 34 37 38 35 33 36 36 32 35 ; 4013326478536625
000000a0h:	38 36 38 33 33 36 35 32 37 32 34 33 32 36 32 34 32 ; 8803032724326242
000000b0h:	35 31 35 31 37 32 34 39 30 36 37 38 30 35 36 34 ; 5151724906780564
000000c0h:	36 36 33 33 32 36 32 37 31 31 38 39 33 34 33 37 ; 6833202711893437
000000d0h:	39 33 38 34 31 34 31 38 30 37 38 32 36 37 35 38 ; 9384141807826758
000000e0h:	33 37 35 39 35 33 36 37 36 32 35 36 35 31 38 32 ; 3759536762565102
000000f0h:	32 34 39 37 32 37 36 38 39 37 36 38 38 36 31 ; 2497276889760861
00000100h:	36 38 39 36 35 20 E0 00 00 00 00 2B E5 B1 8F C3 19 ; 68965 ?..+猿.
00000110h:	48 E1 29 53 6D 8E E2 CE 1E EB 80 78 B1 A9 56 F4 ; K?Sm是?/?x吸V?
00000120h:	BA D8 7A 93 93 E0 88 80 3D 64 A5 CC 44 A4 E0 42 ; 長z吸?dHD么B
00000130h:	E4 DA 34 BC 18 41 2C FD 67 C2 E3 BC AF 7C 0C 86 ; 喵?4A.?音吸 .?
00000140h:	44 E2 E7 7D D0 01 07 30 43 27 FD 4F CB BC 31 79 ; D詞)今,C=?.調1y
00000150h:	5A B5 37 5E 94 A5 56 0A C3 53 A3 1F 97 38 40 ; Z?^音V.翻吸.?M
00000160h:	EE D2 65 28 8F 66 AC D7 4F 39 84 24 14 82 F0 42 ; ?. (音809?:音B
00000170h:	FB D8 92 86 EF 88 47 68 B4 59 CA 98 88 11 D0 AD ; 聲吸?Gh吸??集
00000180h:	02 AB C1 59 89 C3 35 68 F9 30 66 92 F8 96 75 06 ; ?. ?Y吸5? f吸.
00000190h:	A4 44 15 C8 DC 44 4F E2 2D E5 80 E2 99 AC 69 ; 韶,聲D0???.吸1
000001a0h:	B0 42 DD BE 33 8A 58 9C EB 25 98 9C 83 DA 50 91 ; 防防3音吸%吸%吸]
000001b0h:	EE E3 54 71 C0 F9 58 FA B2 78 5E E3 11 57 FF F9 ; 調Tq吸X吸(^W ?
000001c0h:	E8 91 70 BC 33 F8 B2 FB CD 08 9F 74 E1 51 D1 74 ; ?p??.吸,聲??
000001d0h:	F1 BC BD 9F 6E B1 CC AF 10 31 2C FD 72 8D C4 90 ; 注歌n吸?1,?聲?
000001e0h:	E9 36 E7 35 16 7A 5F F5 A0 C6 ; ??_z_??

Fig 78. Encrypted Key Data 부분

Key BLOB 데이터만 별도로 추출해서 확인해보면 아래와 같이 숫자로만 이루어진 String이 추출된다. 길이가 258 Bytes이지만 한 글자씩의 공백 문자(), 아스키코드 0x20이 끝에 하나씩 포함되어 있어서 사실상 $128 + () + 128 + ()$ 의 형태이며, 결국 256 Bytes이 유의미한 값으로 보인다.

316833855808352628316164381689857557586715150386242778810430891848291724689 0578927578258080058620745511361118074567629391922004 295832672257776093333727295291577828754588662770695798349900233358659844673 21901608409375136227315460265087353311441756484290393

이어서 'Fig 79'는 실제 원본 파일의 데이터가 암호화된 형태로 저장되어 있는 부분이다. 이 역시 전반부에는 파일의 크기가 나오고 뒤에 연이어 파일 내용이 기록되는데, AES 알고리즘의 특성상 패딩으로 인해 크기가 변동될 우려가 있으므로 크기 부분을 명시해서 작업 완료 후 검증하기 위한 수단으로 사용한다.

```

00000000h: 02 01 00 00 32 35 31 33 30 32 35 30 35 33 32 36 ; ....251302505326
00000010h: 36 35 30 30 35 35 39 33 34 32 38 39 30 38 32 32 ; 6500559342898822
00000020h: 33 35 35 38 35 31 36 32 37 39 30 30 35 39 38 31 ; 3558516279005981
00000030h: 30 36 36 35 30 33 33 30 38 30 30 30 33 36 ; 0665033308000036
00000040h: 38 34 32 35 35 37 33 38 35 36 33 37 31 35 37 31 ; 8425573856371571
00000050h: 39 33 31 37 37 36 31 33 33 36 30 36 35 36 33 34 ; 9317761336065634
00000060h: 34 39 39 38 31 33 38 31 38 34 34 31 33 32 33 31 ; 4998138184413231
00000070h: 31 37 30 38 36 34 30 39 35 31 31 32 32 36 39 38 ; 1708640951122698
00000080h: 33 38 35 39 26 31 34 38 34 39 31 31 33 35 36 38 ; 3859 14849113560
00000090h: 34 36 31 33 33 32 36 34 37 38 35 33 36 36 32 35 ; 40131326478536625
000000a0h: 38 38 30 33 36 33 32 37 32 34 33 32 36 32 34 32 ; 8803032724326242
000000b0h: 35 31 35 31 37 32 34 39 30 36 37 38 30 35 36 34 ; 51517249067880564
000000c0h: 36 38 33 33 32 30 32 37 31 31 38 39 33 34 33 37 ; 6833202711893437
000000d0h: 39 33 38 34 31 34 31 38 30 37 38 32 36 37 35 38 ; 9384141807826758
000000e0h: 33 37 35 39 35 33 36 37 36 32 35 36 35 31 36 32 ; 3759536762565182
000000f0h: 32 34 39 37 37 32 37 36 38 38 39 37 36 38 36 31 ; 2497276889760861
00000100h: 36 38 39 36 35 26 F0 00 00 00 B E5 B1 8F C3 19 ; 68965 ? .+拱?
00000110h: 4B E1 29 53 60 8E E2 CE 1E EB 80 78 B1 A9 56 F4 ; ?S是?X吸V?
00000120h: BA DB 7A 93 93 E8 88 80 3D 64 A5 CC 44 A4 E0 42 ; 長z缺?AMDISH
00000130h: E4 DA 34 BC 18 41 2C F6 67 C2 E3 BC AF 7C 0C 86 ; 電4A.?脣歎|?
00000140h: 44 E2 E7 7D D0 D1 07 3D 43 27 FD 4F CB BC 31 79 ; D(固)今.+C?鏡1Y
00000150h: 5A B5 37 5E 94 A5 56 0A C3 53 83 A3 1F 97 38 40 ; Z?^부V. 脊?
00000160h: EE 62 05 28 8F 66 AC D7 4F 39 B4 2A 14 82 F0 42 ; ?. (脳?G??
00000170h: FB D8 92 86 EF 88 47 68 59 CA 98 BB 11 D8 AD ; 声翻?G翌??莫?
00000180h: 02 AB C1 59 89 C3 35 60 F9 30 62 F8 96 75 06 ; .手Y脳??.?
00000190h: A4 44 15 C8 DC 44 AF E2 2D E5 80 E2 99 AC 98 69 ; 韓.脳DO??.?
000001a0h: B0 42 0D BE 33 8A 58 9C EB 25 98 9C 83 DA 50 91 ; 鏡附3云脳%?
000001b0h: EE E3 54 71 C0 F9 58 FA B2 7B 5E E3 11 57 FF F9 ; 鏡T??
000001c0h: E8 92 80 33 F8 B2 FB CD 08 9F 74 E1 51 D1 74 ; P?烹.要.?
000001d0h: F1 BC 8D 9F 61 B1 CC AF 10 31 2C FD 72 8D C4 96 ; 住観n?.?
000001e0h: E9 36 E7 35 16 7A 5F F5 A0 06 ; ??-z ?E

```

→ 리틀 엔디언(little endian) 기준 → 0x000000E0 → 224 Byte

Fig 79. 암호화된 파일 부분

이 부분 역시 처음 4자리인 E0 00 00 00을 리틀 엔디언 기준 16진수로 변환하면 0x000000E0이 되어 10진수 224 Byte가 된다. 다만 실제 원본의 크기는 212 Byte였는데, 이는 패딩의 영향이다.

Ahnlab의 보고서에 따르면, Jaff Ransomware의 변종들은 리소스 영역의 RCData Section에 고정된 0x104 길이의 데이터를 사용한다고 한다. 'Fig 80'은 PE Explorer 도구를 사용하여 해당 바이너리에 포함된 리소스 정보를 확인한 화면이다.

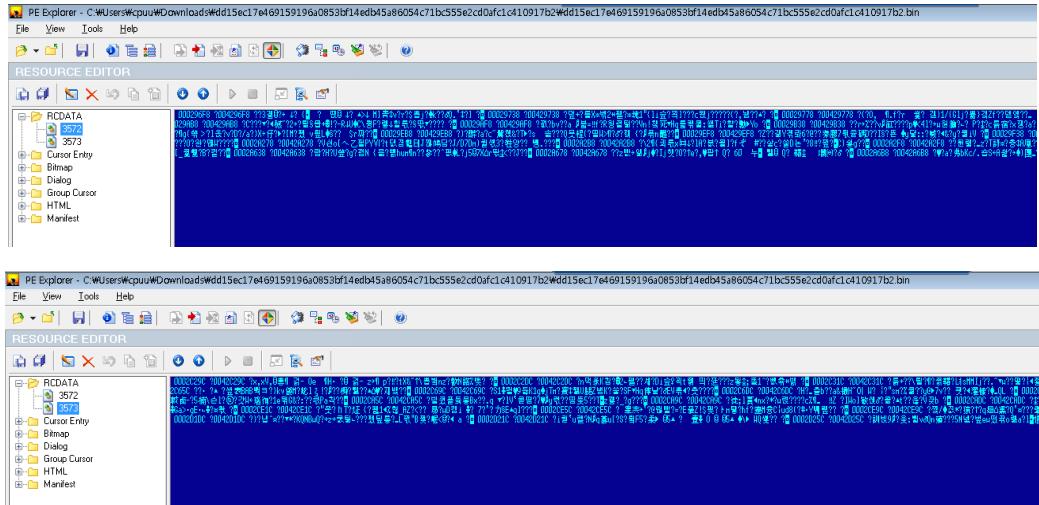


Fig 80. RCData에 포함된 블록

그러나 이 정보는 인코딩 되어 있으며, 실제 런타임에 디코딩되는데 메모리 분석을 통해 Yara 검색으로 해당 값을 'Fig 81'과 같이 추출하여 확인할 수 있었다.

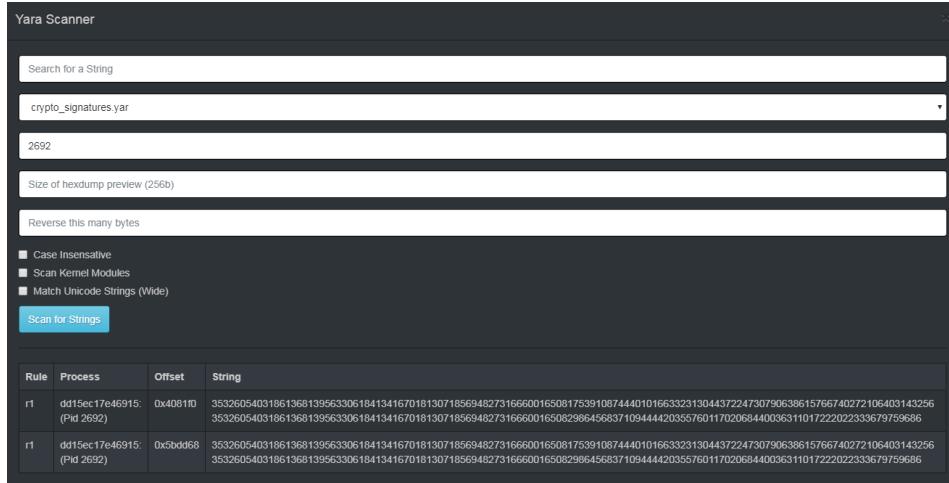


Fig 81. 메모리에서 추출된 개인키로 추정되는 값

이 값은 0x104 즉 십진수로 260이지만, 두 글자씩의 개행 문자(/n), 아스키 코드 0x0D 0x0A가 끝에 하나씩 포함되어 있어서 사실상 128 + () + 128 + ()의 형태이며, 결국 256 Bytes이 유의미한 값으로 보인다.

```
353260540318613681395633061841341670181307185694827316660016508175391087444
01016633231304437224730790638615766740272106403143256
353260540318613681395633061841341670181307185694827316660016508298645683710
9444420355760117020684400363110172220223367975968667
```

이 스트링으로 된 숫자 값이 앞서 'Fig 78'에서 사용된 Key BLOB DATA를 생성하는 데 사용되는 것으로 보아 RSA 알고리즘의 개인키에 해당하는 것으로 추정된다.

이상의 내용을 종합하면, 암호화 과정에서 최초 생성된 AES-256 대칭 키 데이터는 모든 파일 암호화에 이용된다. 이때 암호화된 파일에 저장할 키 데이터를 생성하기 위해 사용자 decrypt ID 정보, AES-256 키 블롭 데이터 정보, RCDATA 영역에 저장된 고정된 0x104 크기의 데이터가 사용된다.

RCDATA 영역에서 추출되는 데이터는 Jaff 랜섬웨어의 모든 변종에서 동일한 것으로 확인되었으므로, 사용자의 Decrypt ID만 얻을 수 있다면, 암호화 과정을 거꾸로 연산하여 키 블롭 데이터로부터 파일 암호화에 사용된 AES-256 키를 얻을 수 있고, 암호화된 파일들을 복원할 수 있게 된다.

4.5.2 Kaspersky RakhniDecryptor 를 통한 파일 복원

복호화는 암호화의 역순으로 해결할 수 있다. 기본적으로 암호화에 사용한 라이브러리인

Microsoft Cryptographic Service Providers에서 제공하는 복호화 함수의 대략적인 코드는 아래와 같다.

```

if (!ReadFile(hSourceFile, &dwKeyBlobLen, sizeof(DWORD), &dwCount, NULL))
{
    MyHandleError(TEXT("Error reading key BLOB length!\\n"), GetLastError());
    goto Exit_MyDecryptFile;
}

if (!(pbKeyBlob = (PBYTE) malloc(dwKeyBlobLen)))
{
    MyHandleError(TEXT("Memory allocation error.\\n"), E_OUTOFMEMORY);
}

if (!ReadFile(hSourceFile, pbKeyBlob, dwKeyBlobLen, &dwCount, NULL))
{
    MyHandleError(TEXT("Error reading key BLOB length!\\n"), GetLastError());
    goto Exit_MyDecryptFile;
}

if (!CryptImportKey(hCryptProv, pbKeyBlob, dwKeyBlobLen, 0, 0, &hKey))
{
    MyHandleError(TEXT("Error during CryptImportKey!\\n"), GetLastError());
    goto Exit_MyDecryptFile;
}

if (pbKeyBlob)
{
    free(pbKeyBlob);
}

```

Fig 82. key BLOB에서 세션키를 복원하는 C 소스코드 예시

'Fig 82'은 keyBLOB에서 세션 키를 복원하는 코드의 진행 과정이다. 먼저 암호화된 파일로부터 key BLOB 데이터의 길이를 읽어오고, 그것을 저장할 만한 크기의 버퍼를 동적으로 할당한 후 key BLOB 데이터를 읽어와서 pbKeyBlob 변수에 저장한다. 읽어온 내용은 CryptImportKey() 함수를 호출하여 처리된다.

이렇게 획득한 pbKeyBlob으로부터 실제 파일 암호화에 사용한 AES-256의 세션 키를 복원하였다면, 다음으로는 세션 키로부터 파일을 복원하는 과정이 필요하다.

```

bool fEOF = false;
do {
    if (!ReadFile(hSourceFile, pbBuffer, dwBlockLen, &dwCount, NULL))
    {

```

```

MyHandleError(TEXT("Error reading from source file!\n"), GetLastError());
goto Exit_MyDecryptFile;
}

if (dwCount <= dwBlockLen)
{
    fEOF = TRUE;
}

if (!CryptDecrypt(hKey, 0, fEOF, 0, pbBuffer, &dwCount))
{
    MyHandleError(TEXT("Error during CryptDecrypt!\n"), GetLastError());
    goto Exit_MyDecryptFile;
}

if (!WriteFile(hDestinationFile, pbBuffer, dwCount, &dwCount, NULL))
{
    MyHandleError(TEXT("Error writing ciphertext.\n"), GetLastError());
    goto Exit_MyDecryptFile;
}

} while (!fEOF);
fReturn = true;

```

Fig 83. 파일 내용을 복호화하여 저장하는 C 소스코드 예시

'Fig 83' 은 do while loop를 순회하며 해당 경로의 파일들을 하나씩 읽어드린 후 CryptDecrypt 함수를 통해 hKey(세션키)로 파일 내용을 복호화하는 과정을 보여주고 있다. 복호화가 완료되면 WriteFile 함수를 통해 버퍼의 내용을 hDestinationFile 에 저장함으로써 파일 작업을 완료한다.

이상의 내용을 종합적으로 요약하자면, 먼저 암호화된 데이터에 접근하고, RSA 암호화를 해결할 certificate store를 개통한다. 그리고 식별자로 사용한 User Decrypt ID(숫자) 정보를 토대로 개인키 정보를 얻어낸다. 이 개인키를 통해 Key BLOB DATA를 복호화하면 파일 대칭 암호에 사용된 세션 키를 복원할 수 있다. 이제부터는 AES-256 알고리즘의 복호화 기능으로 세션 키를 이용하여 모든 파일을 복원하면 된다.

해당 API의 암호화 및 복호화 함수 C/C++ 구현 프로그램은 그 구조가 거의 유사하므로, 단순히 암호화 함수(CryptEncrypt)를 복호화 함수인 (CryptDecrypt)로 변경하기만 해도 파일을 복원할 수 있다. 왜냐하면 암호화 시에 CryptEncrypt() 함수에서 사용하는 파라미터 중 6개가 복호화 시 CryptDecrypt() 함수에서 요구하는 것과 일치하기 때문이다. 따라서 PE 바이너리의 IAT를 PE 바이

너리 수정 도구를 이용해 강제로 패치하거나, 아니면 실행시간에 디버거를 사용하여 호출되는 함수를 변조하여도 해법이 될 수 있다.

한편 복호화 기능을 구현한 사례가 바로 2018년 1월 9일에 Kaspersky 업체에서 공개한 Rakhni Descriptor 프로그램이다. <https://noransom.kaspersky.com/> 에서 무료로 다운로드할 수 있으며, 현재 최신 버전인 1.21.19.2 를 보면 Jaff ransomware 에 대한 복구를 지원하고 있다.

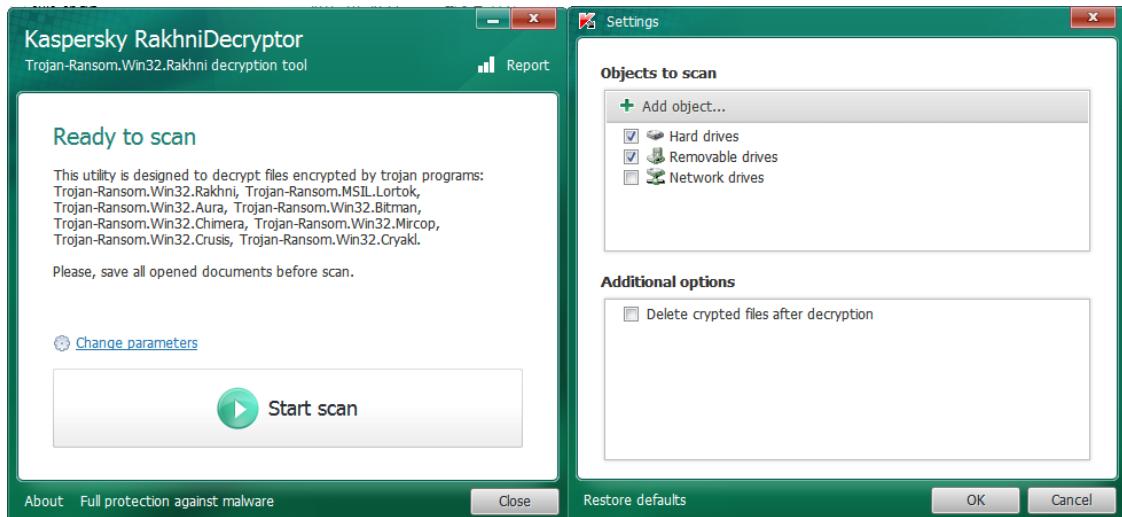


Fig 84. Kaspersky RakhniDecryptor를 이용한 파일 복원

이 프로그램이 이미 4.5.2절에서 설명한 복호화 알고리즘을 충실히 구현하고 있으므로 이를 활용하였다. 'Fig 84' 과 같이 실행하여 검색 대상 드라이브를 선택하고 스캐닝을 시작하면 된다.

먼저 AccessData의 FTK Imager 를 통해 감염된 상태의 대회 이미지 파일을 마운트 하였으며, 복호화 작업에는 파일 수정 작업이 필요하므로 읽기 전용 모드가 아닌 쓰기 가능 모드를 선택하였다. 그리고 복호화 프로그램에서 감염자 PC 바탕화면의 암호화된 sVn 확장자 파일을 선택하였다.

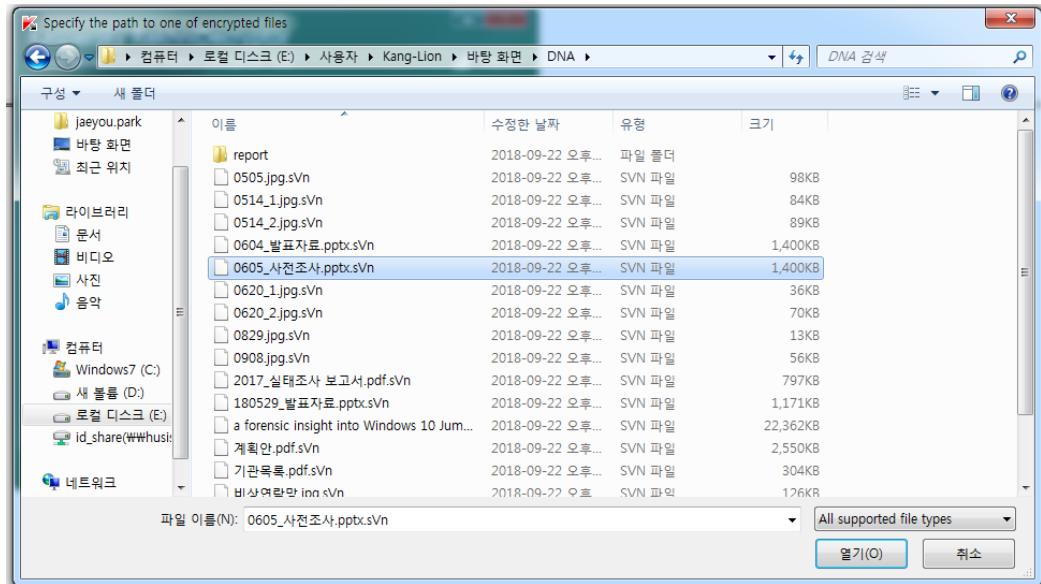


Fig 85. 대회 이미지 내부 파일 복원 시도

이어서 프로그램이 자동으로 sVn 파일임을 인식하여 Jaff ransomware인 것을 확인 후 해당 악성 코드에서 생성한 랜섬웨어 노트 파일을 지정하는 대화 상자가 'Fig 86' 와 같이 팝업 된다.

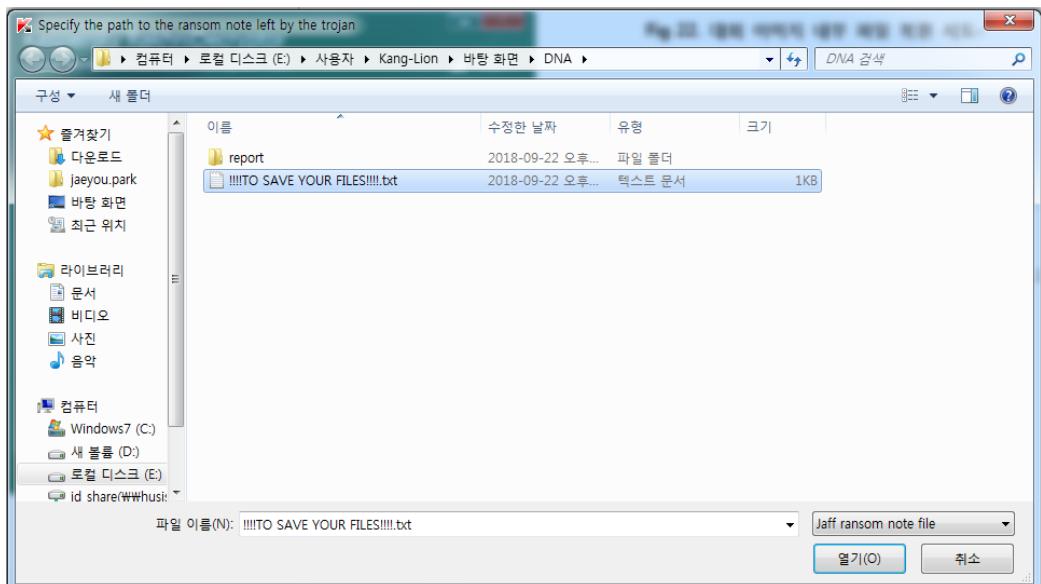


Fig 86. Ransom note 파일 지정

"!!!!TO SAVE YOUR FILES!!!!.txt" 파일을 지정하면 순조롭게 해결될 것으로 기대하였으나 아쉽게도 'Fig 87' 와 같이 실패했다는 메시지가 출력되었다.



Fig 87. 복원 실패 메시지 표출

처음에는 복구 프로그램이 온전하지 못하거나, 랜섬웨어의 일부 변종에는 적절히 대응하지 못하는 것으로 보여 복구가 불가능할 것으로 짐작하였다.

그러나 추가적인 관찰을 통해 몇 가지 특이사항을 발견하였다. 먼저 바탕화면에 보이는 bmp 파일에는 사용자의 decrypt ID를 표기하는 곳에 구분자로 콜론(:)이 존재하는데, 각 폴더 내부에 있는 텍스트 파일에는 ":"이 없는 상황이다. 이는 'Fig 88'와 'Fig 89' 을 비교해보면 알 수 있다.

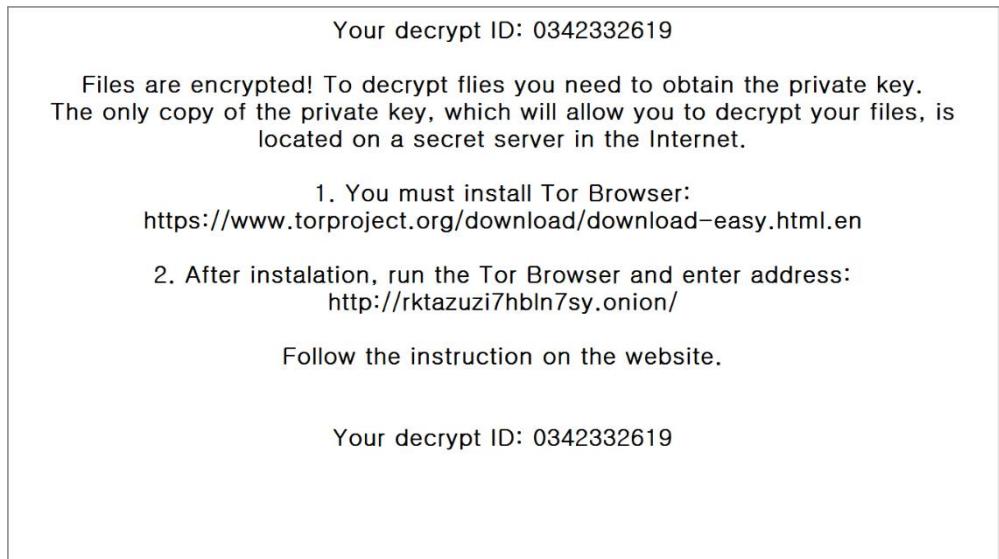


Fig 88. Your decrypt ID 부분에 ":" 문자로 구분됨

특히 랜섬 노트 파일에 기재된 url 주소를 표기하는 곳에서도 http:// 프로토콜에 대한 : 구분 문자가 생략되어 있다. 구체적인 이유는 확인할 수 없지만 특수문자 처리 과정에서 오류가 발생했

거나, 혹은 랜섬웨어가 의도적으로 복호화 프로그램의 작동 로직을 회피하기 위한 목적으로 : 문자를 출력하지 않는 것으로 보인다.

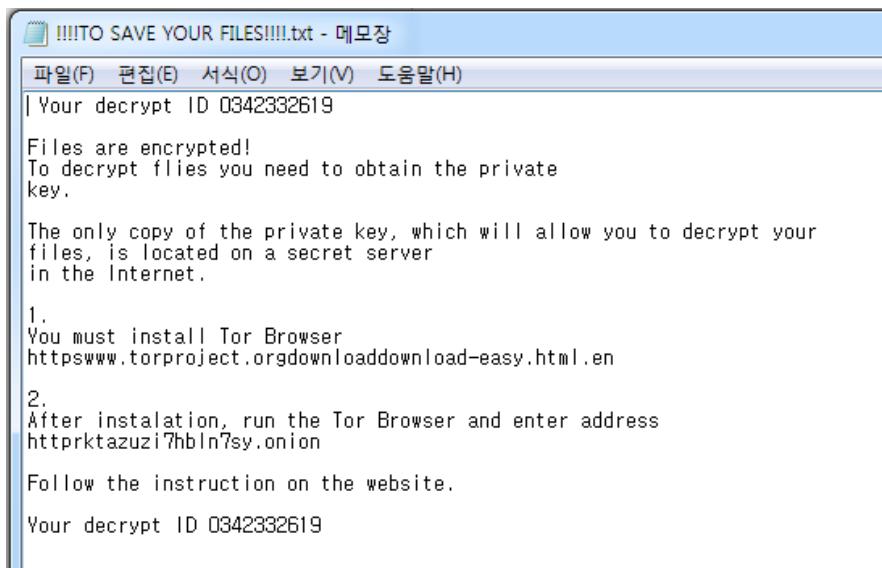


Fig 89. Your decrypt ID 부분에 : 구분 문자가 없음

이를 해결하기 위한 방법으로는 복호화 도구에서 : 문자를 기준으로 텍스트 파싱하는 부분을 수정하는 것이 근본적인 해결책이지만, 본 상황에서는 대회 이미지에 대한 복원이 시급하므로 도구에 대한 수정이 아닌 랜섬 노트 파일에 대한 수정으로 문제를 단순화시키는 것이 효율적이라고 판단하였다. 'Fig 90'은 랜섬웨어 노트 파일을 텍스트 에디터로 수정하여 decrypt ID 부분에 구분 문자인 콜론(:)을 삽입하여 저장한 화면이다.

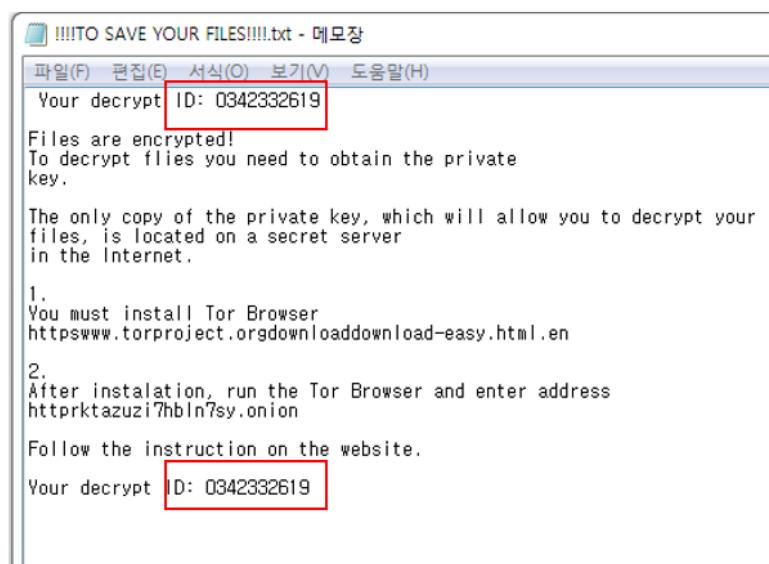


Fig 90. Your decrypt ID 부분에 ":" 구분 문자 수동으로 추가

이를 통해 복호화 도구를 재수행한 결과 'Fig 91'과 같이 Password recover에 성공했다는 로그 메시지가 확인되었으며, 파일 복호화 작업이 시작되었다. 수정된 랜섬 노트는 '첨부 9 수정한 랜섬

'노트'에서 확인할 수 있고, 동일하게 실험할 수 있다.

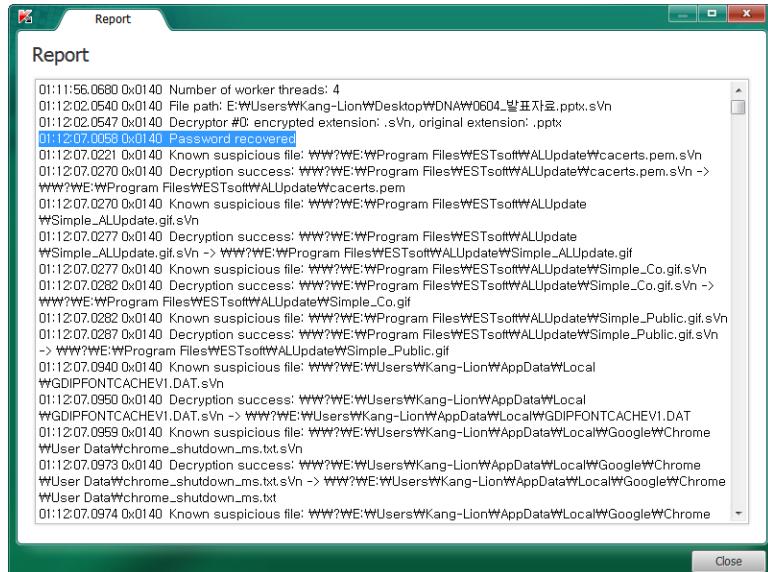


Fig 91. Password 복원 성공 로그 메시지

전체 파일의 양이 방대하여, 복구 작업이 진행되는 동안 시간이 많이 소요되었다.

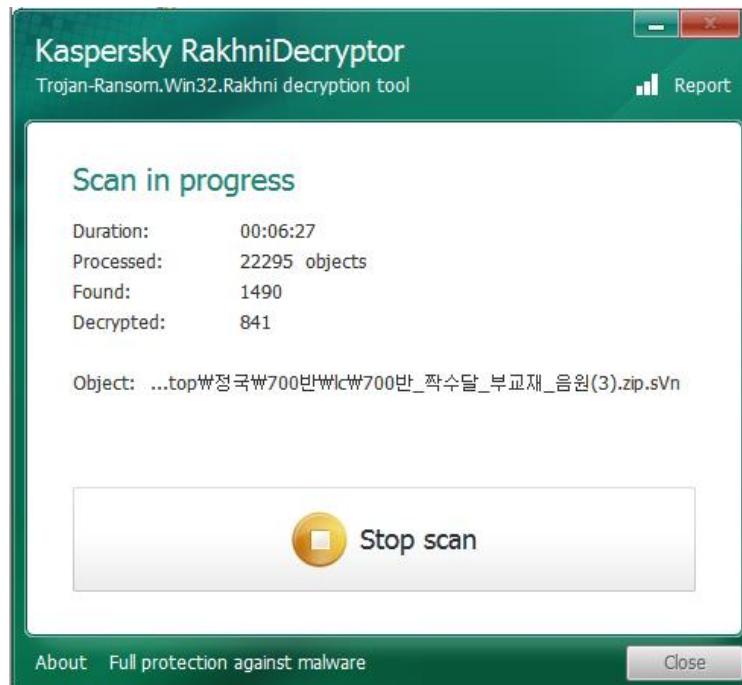


Fig 92. 파일 복호화 진행 메세지

복호화 작업이 완료된 후에는 'Fig 93'과 같이 확인 보고서가 표출된다. 바탕화면에 존재하였던 다수의 PDF 문서 파일들이 정상적으로 Decrypted 되었다는 메시지가 보인다. 총 2701개의 svn 확장자 중 2676개가 복구되었다. 복구 작업 관련 로그는 '첨부 10 복호화 도구 로그 내역'을 통해 확인할 수 있다.

Event	Object
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2013\session4 New forensic environments#D
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2013\session4 New forensic environments#D
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2013\session4 New forensic environments#D
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2013\session4 New forensic environments#D
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2013\session5 Memory Forensics#DFRWS201
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014#DFRWS 2014 Agenda.html.s\n
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014#DFRWS 2014 Agenda_files#analytics.js
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014#DFRWS 2014 Agenda_files#calendar-64
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014#DFRWS 2014 Agenda_files#dfrrwsGloba
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014#DFRWS 2014 Agenda_files#dfrrwsNav.js
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014#DFRWS 2014 Agenda_files#searchButt
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014\session1 Memory#DFRWS2014-1.pdf.s
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014\session1 Memory#DFRWS2014-2.pdf.s
Decrypted	WW?WE:WUsers#Kang-Lion#Desktop#W\b WDFRWS#2014\session1 Memory#DFRWS2014-3.pdf.s

Fig 93. 복원 성공 파일 목록

실제로 복호화에 성공한 몇 가지 원본 파일들의 내용을 'Fig 94'과 'Fig 95' 처럼 확인할 수 있다.

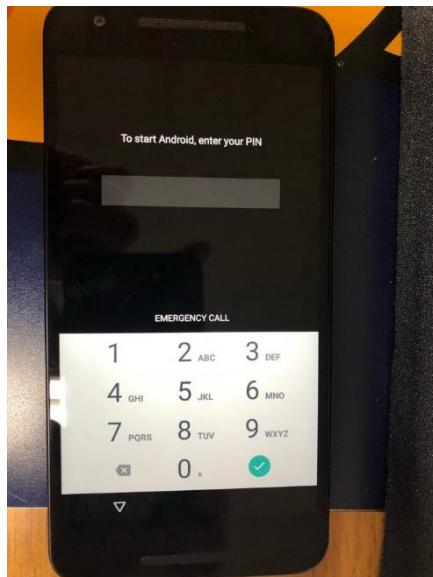


Fig 94. "/Users/Kang-Lion/Desktop/DNA/0514_2.jpg" 파일의 복호화된 내용



Fig 95. "/Users/Kang-Lion/Desktop/부/한국정보보호학회지/2014/악성코드 탐지를 위한 물리 메모리 분석 기술.pdf" 파일의 복호화된 내용

4.6 소결

본 대회에서 출제된 시스템은 2017년 2분기에 유행하였던 Jaff 랜섬웨어에 의해 파일이 암호화된 상태이다. 이에 본 장에서는 해당 랜섬웨어의 샘플을 입수하여 해당 악성코드에 대한 정적 분석과 동적 분석 그리고 메모리 포렌식 분석을 수행하였다.

그 결과 랜섬웨어가 전체적으로 작동하는 방식을 파악할 수 있었으며, 특히 파일 암호화를 수행할 때 사용하는 Microsoft Cryptographic Service Providers 의 API를 심층적으로 분석함으로써, 실제 파일 암호화가 이루어지는 과정과 그에 따른 암호화 키 생성 및 저장법에 관한 정보를 얻을 수 있었다.

또한 해당 API에서 암호화된 파일을 다시 복호화하여 원본 파일로 변경하는 방법이 가능함을 확인하였으며, 이 과정을 이미 구현한 사례인 Kaspersky 업체의 RakhniDecrypt 도구를 통해 대회 시스템 디스크 이미지의 파일 복원을 성공하였다. 이때 기존 복구 도구에서 약간의 버그 혹은 미흡한 오류가 발생하였으나, 랜섬웨어 노트 파일의 Decrypt ID에 대한 구분자를 직접 삽입함으로써 정상적으로 활용이 가능함을 입증하였다.

5. 랜섬웨어 실행 차단 방안

5.1 일반적인 랜섬웨어 대책

일반적으로 랜섬웨어를 포함한 대부분의 악성코드는 시스템의 취약점을 이용하는 등의 고도의 기법을 사용하기도 하지만, 때로는 가장 약한 고리를 표적으로 삼는 것에서 더 큰 수확을 얻기도 하는데, 그 지점은 바로 '사람'이다. 언뜻 보면 가장 단순한 것 같아도 실제로 가장 많은 비율을 차지하는 것이 이메일을 통한 첨부파일 전송이다. 이글루시큐리티 보고서의 통계에 따르면 메일을 통해 개인뿐만 아니라 기업을 대상으로 한 공격이 가장 활개를 치고 있다.



Fig 96. 이메일을 통한 랜섬웨어 등 악성코드 피해 통계

따라서 출처가 불분명한 곳으로부터 전달된 메일의 링크를 클릭하거나, 첨부파일을 열어볼 때에는 반드시 신중한 판단이 필요하다. 되도록이면 메일 클라이언트에서 제공하는 안티 스팸 필터를 사용하시기를 권장하며, 아웃룩이나 구글 Gmail 등등 대부분의 메일 서비스 업체는 안티 스팸 기능을 기본으로 탑재되어 있다.

또한 기업차원에서는 정보보호팀의 주관으로 주기적으로 모의훈련을 실시하여 직원들을 대상으로 보안의식을 고취시킬 수 있다. 실제로 이러한 훈련을 통해 횟수가 늘어날수록 대상자들의 학습효과가 뛰어난 것으로 나타났다.

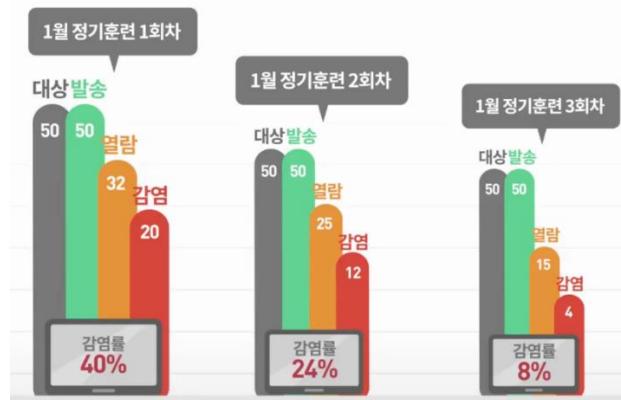


Fig 97. 기업 내부 모의 훈련을 통한 경각심 고취

이러한 훈련을 통해 사람에 의한 결점을 보완하는 것이 가장 효과가 크다고 볼 수 있다.

또한 사내 보안 정책 등을 마련하여, 불필요하게 개방되어 있는 부분들을 식별하여 권한을 축소하는 것도 중요하다. 예를 들어 문서형 악성코드는 대부분 오피스 문서 파일의 매크로 기능을 악용하는 방식을 취하는데, 기업 내부 보안 정책에 의거하여 매크로 기능의 사용을 지양할 수 있다. 꼭 사용해야 하는 경우라면 디지털 서명으로 검증된 매크로의 경우에만 한정적으로 허용할 수도 있다. 그 외에도 Windows script Host, Power Shell 등의 기능이 조직의 특성상 꼭 필요한 경우가 아니라면 이를 비활성화하는 것이 좋다.

그리고 안티 바이러스 및 백신 제품, 방화벽 솔루션 등을 사용하고 있다면 최신 상태로 업데이트 하여 운용하는 것이 중요하다. 이러한 솔루션들은 대부분 알려진 위협에 대해 시그니처 기반으로 탐지를 하고 있기 때문에, 아직 알려지지 않은 유형의 공격에 대해서는 속수무책으로 당할 수밖에 없다. 정기적인 업데이트를 통해 최신 취약점 대응책이 적용되도록 하는 것이 좋으며 이는 윈도우 운영체제 역시 예외가 아니다.

마지막으로 강조할 수 있는 것은 데이터 백업이다. 중요한 파일은 반드시 정기적으로 백업하여야 한다. 그리고 백업이 완료되면 해당 저장매체는 별도로 분리하여 보관하여야 한다. 특히 클라우드에 자동으로 동기화하는 방식의 백업의 경우 깊은 주의가 요구된다. 랜섬웨어에 감염되었다는 사실을 깨닫게 되는 즉시 네트워크/인터넷 연결을 분리하고, 컴퓨터 시스템을 꺼버려야 한다. 그렇게 해야 암호화된 파일이 백업본마저 덮어써버리는 참사를 방지할 수 있다.

5.2 기술적 측면의 연구 동향

본 절에서 먼저 기술적 측면에서의 랜섬웨어 차단 관련 연구동향을 살펴보고, 이를 종합하여 다음 절에서 Jaff 랜섬웨어에 특화된 대응 방안을 제시한다.

5.2.1 UNVEIL

USENIX Security 2016에서 발표된 논문[21]은 소니 픽쳐스가 영화 "The Interview"를 제작하여 배급하려던 중 광범위한 랜섬웨어 공격을 받았던 사건에 착안하여, 동적 분석을 통해 랜섬웨어를 탐지하는 일명 UNVEIL이라는 시스템을 제안하였다. 이 시스템이 사용하는 핵심 기술은 다음으로 요약할 수 있다.

- 자동으로 시스템의 특정 영역에 일명 '미끼' 파일을 만들어 둔 뒤, 그 파일에 대한 접근이 발생하면 탐지
- 대상 시스템의 바탕화면의 변화를 추적하여 랜섬웨어로 의심되는 행위 감시
- 파일 시스템 활동 추적 (입출력 버퍼의 엔트로피 및 접근 패턴 분석)

이 해법은 일부 안티 바이러스업체의 성능과 비교했을 때 아직 알려지지 않은 악성코드를 탐지

할 수 있다는 장점이 있었지만, 오탐이 일부 발생할 수 있다는 단점도 존재하였다. 또한, UNVEIL 자체가 Kernel Mode로 작동해야 하기 때문에 지나친 권한을 부여해야 하는 것이 아니냐는 지적 또한 존재하며, 악성코드가 Kernel까지 장악한 경우에는 효과를 발휘할 수 없다는 한계점이 존재한다.

5.2.1 블록암호의 구현 특성에 따른 식별

윤세원 등[22]에 따르면 기존 연구와 차별화된 방안의 일환으로 블록 암호의 구현 특성에 따른 식별을 제시한다.

AES 알고리즘은 국제 표준 블록 암호로서 정보보호 시스템 뿐만 아니라 랜섬웨어들도 애용하는 알고리즘이다. 이 알고리즘의 구현은 S-Box 등을 통해 다차원 행렬을 빠르게 연산하는 방식으로 이루어지는데, 이런 상수값들의 곱셈 결과가 메모리에 저장되므로, 해당 값이 인지된다면 AES가 작동 중임을 유추할 수 있다는 것을 골자로 한다. 실험을 통해 WannaCry나 Mamba 등의 랜섬웨어에서 AES의 작동을 탐지할 수 있었으며, 이러한 기법은 AES뿐만 아니라 LEA, HIGHT 등의 암호 알고리즘에도 동일하게 적용할 수 있다는 것을 보였다.

본 방법의 경우 기존의 탐지 방법들이 가질 수 있는 미탐 현상에 대비하여 '현재 암호 알고리즘이 동작하고 있음' 등과 같은 추가적인 단서를 제공하는데 유용하다. 하지만 본 방법만으로는 오탐이 발생할 수 있어 기존 탐지방법들과 적절한 결합이 필요할 것으로 보인다.

5.3 Jaff 랜섬웨어에 특화된 대응 방안 제안

본 문제 상황에서 사용된 Jaff 랜섬웨어의 특징에 착안하여, 다음과 같은 탐지 및 대응 방안들을 제안할 수 있다.

- Jaff 랜섬웨어는 디스크 볼륨을 순회하면서 해당 드라이브 중 특정 확장자를 가진 파일을 암호화하는 특징이 있다. 이러한 특성을 이용해서 드라이브의 최상위 디렉토리에 알파벳 순서로 가장 빠른 단어를 제목으로 한 미끼(bait) 파일을 배치하고 어떤 프로세스가 파일에 수정 행위를 가하는지 모니터링을 수행한다. 해당 파일이 가장 먼저 목록에 투입된다 고 가정한다면 그 이후의 작업을 조기에 중단시키는 효과가 있다.
- 암호화 시에 Microsoft Cryptographic Service Providers 의 API 을 이용한다는 점에 착안하여, 특정 API Call 을 Hooking 한다. 본 악성코드는 CryptGenKey, CryptEncrypt 같은 함수들을 암호화 과정에서 호출하는데, 이때 Hooking을 통해 암호화 키의 내용을 확인할 수 있다.[23]

참고 문헌

- [1] <https://kdfs.jams.or.kr>
- [2] TTA 표준화 위원회, 디지털 증거 수집·보존 가이드라인, 2017, p.10-11
- [3] <https://drive.google.com/open?id=12Kvi2pDMkCy5p0MVVMRr5WMvSUHRzvCT>
- [4] chester777, (<https://github.com/chester777/analyzeMFT>)
- [5] <http://www.ntfs.com/disk-scan.htm>
- [6] Winhex Maual, (https://www.fss.jp/wp-content/uploads/WinHex_fss_soft0502.pdf)
- [7] <https://www.r-studio.com/ko/>
- [8] <http://forensic-proof.com/archives/2854>
- [9] <https://cloudbase.it/qemu-img-windows/>
- [10] Recovering Deleted VSS Snapshots, Minoru Kobayashi, Blackhat US 2018
- [11] Gruhn, Michael. "Windows nt pagefile. sys virtual memory analysis." IT Security Incident Management & IT Forensics (IMF), 2015 Ninth International Conference on. IEEE, 2015.
- [12] Wikipedia, (https://en.wikipedia.org/wiki/Portable_Executable)
- [13] securityintelligence, (<https://securityintelligence.com/signature-based-detection-with-yara/>)
- [14] ZIP File Specification, (<https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT>)
- [15] <https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Troj~Ransom-ENZ/detailed-analysis.aspx>
- [16] <https://www.hybrid-analysis.com/sample/dd15ec17e469159196a0853bf14edb45a86054c71bc555e2cd0afc1c410917b2>
- [17] 김진국, 박정흠, & 이상진. (2010). 디지털 포렌식 관점의 데이터 복구 및 분석 프레임워크. 정보처리학회논문지, 17(5), 393.
- [18] "Necurs" Botnet Fuels Massive Spam Campaigns Spreading "Jaff" Ransomware (<https://mdb-dev.es/page/52/?cat=-1>)
- [19] ASEC REPORT Vol.87, 2017, Ahnlab (<https://www.ahnlab.com/kr/site/securityinfo/asec/asecReportList.do>)
- [20] MSDN, <https://docs.microsoft.com/en-us/windows/desktop/seccrypto/microsoft-aes-cryptographic-provider>
- [21] Kharraz, Amin, et al. "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware." USENIX Security Symposium. 2016.
- [22] 윤세원, 전문석. (2018). 랜섬웨어 탐지율을 높이기 위한 블록암호 알고리즘 식별 방법에 관한 연구. 정보보호학회논문지, 28(2), 347-355.
- [23] 송정환, 황인태. (2011). 윈도우즈 Crypto API를 이용한 악성코드 무력화 방안 연구 및 도구 구현. 정보보호학회논문지, 21(2), 111-117.