디지털 포렌식 연구 제17권 제3호
‘역공학을 통한 Bianlian 랜섬웨어 복호화 방안 연구’

BCGLAB 240131
IT정보공학과 박수빈

목차

table of contents

기존 랜섬웨어 대응 및 복호화 기술 연구 동향

랜섬웨어 탐지

- Yun Feng외 2인의 연구: Windows OS의 API(FindFirstFile, FindNextFile)을 후킹해 미끼용 파일에 먼저 접근하도록 유도하고, 미끼 파일의 엔트로피 변화를 모니터링함으로 랜섬웨어를 탐지하는 연구를 수행

- Daniel Morato외 3인의 연구: 네트워크 트래픽 분석모델을 통해 랜섬웨어의 암호화 수행 전 C&C 서버와의 키 교환 등의 징후를 포착하고, 랜섬웨어를 탐지하는 연구를 수행

- Iman Almomani외 6인의 연구: Android OS의 랜섬웨어 파일 데이터셋에서 권한과 API call 등의 특징을 추출해 머신러닝 모델을 학습시키는 방식을 통해 랜섬웨어를 탐지

기존 랜섬웨어 대응 및 복호화 기술 연구 동향

감염 파일의 복호화

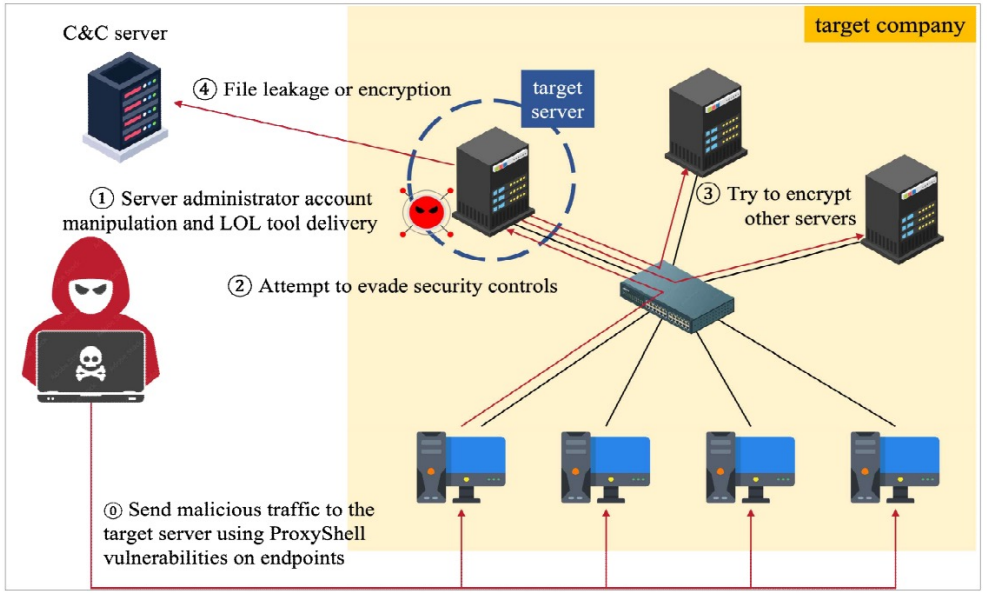
- 강수진 외 5인의 연구: 스트림암호 salsa20을 이용하는 Ragnar Locker의 키 재사용 취약점을 확인하여 이를 통해 감염 파일을 복호화하는 방법을 제안

- 김기윤 외 3인의 연구: Go 언어로 작성된 Hive 랜섬웨어의 분석을 통해 Key stream을 복구함으로써 높은 확률로 파일을 복호화 가능함을 증명

- 이세훈 외 5인의 연구: Hidden Tears를 기반으로 작성된 Donut 랜섬웨어의 분석하여 암호화 과정과 암호화 요소를 파악하고, 메모리 분석을 통해 암호화에 사용된 Key를 찾아낸 후 이를 이용해 감염 파일을 복호화하는 방법을 제시

- 백신 회사 Avast: Go 언어로 작성된 Bianlian 랜섬웨어의 감염 파일을 복호화하는 도구를 배포하였지만 Bianlian 랜섬웨어의 암호화 알고리즘에 대한 정보는 공개하지 않았으며 기존에 수집된 Key와 IV를 이용해 복호화를 수행하는 데 그침

Bianlian 랜섬웨어의 암호화과정 및 암호화 요소 분석



〈Figure 1〉 Propagation and Infection Scenario of Bianlian Ransomware

〈Table 1〉 Propagation and Infection Scenario of Bianlian Ransomware

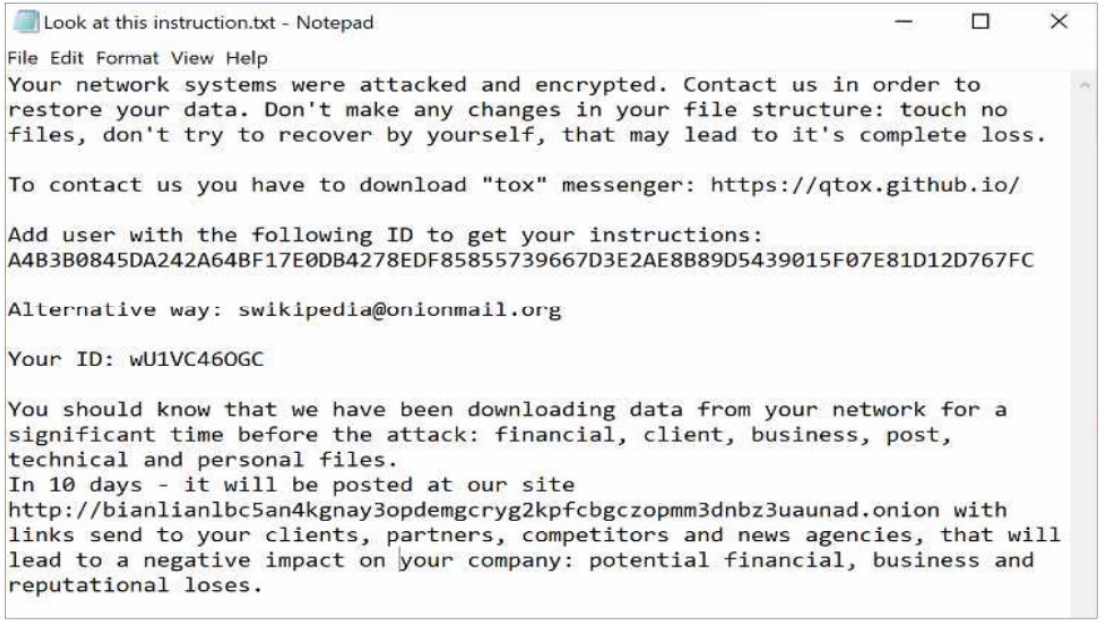
#	Time (Duration)	Event
0	Start to End (4.5 Hours)	공격 시간 동안 서로 다른 Endpoints들로 ProxyShell 취약점을 이용한 악성 트래픽 송신
1	Start + 1 Hours (1 Hour)	여러 서버에서 net.exe를 통한 계정 조작 수행 관리자 계정의 비밀번호를 변경함으로 공격에 대한 방어를 어렵게 만듦 파일 유출 및 원격 접속을 위한 LOL(Living on the Land)도구가 공격지의 여러 서버로 drop 되어 실행
2	Start + 2 Hours (2 Hours)	EDR(Endpoint Detection and Response)등 여러 보안제어 회피 시도
3	Start + 2 Hours (2 Hours)	공격지 내부의 여러 서버로 침투 및 암호화 시도
4	Start + 2 Hours (2.5 Hours)	미리 drop 해놓은 LOL 도구(Backdoor)를 이용해 파일 유출 및 암호화 수행

구성
- 백도어(침투 및 지속적인 공격), Encrypto(파일 암호화를 수행)

전파 및 감염 과정
- 백도어가 공격 대상 서버를 감염시키고 장악하며,
마지막 단계에서 Encryptor를 내려받아 대상 서버의 파일들을 암호화

Bianlian 랜섬웨어의 암호화과정 및 암호화 요소 분석

특징
- 하드웨어적으로 가속화된 AES 256-CBC 암호화와 Go언어의 goroutine을 이용한 동시다발적인 스레드 생성 기능을 통해 빠른 파일 암호화를 수행
- goroutine기능은 정적분석 과정에서 분기되는 주소 숨김으로써 분석 저항성 가짐
- 감염된 파일은 이름 뒤에'.bianlian'확장자가 추가
- 'Look at this instruction.txt ' 이라는 이름의 랜섬노트를 생성



〈Figure 2〉 Ransomnote of Bianlian Ransomware

Bianlian 랜섬웨어의 암호화 과정 및 암호화 요소 분석–분석 환경

AlphaGolang: Go언어 프로그램 분석 보조 도구, Go언어로 작성된 x64 프로그램을 정적분석

Virtualbox 5.2.44 버전의 Windows 10 x64: 동적분석 및 정적분석

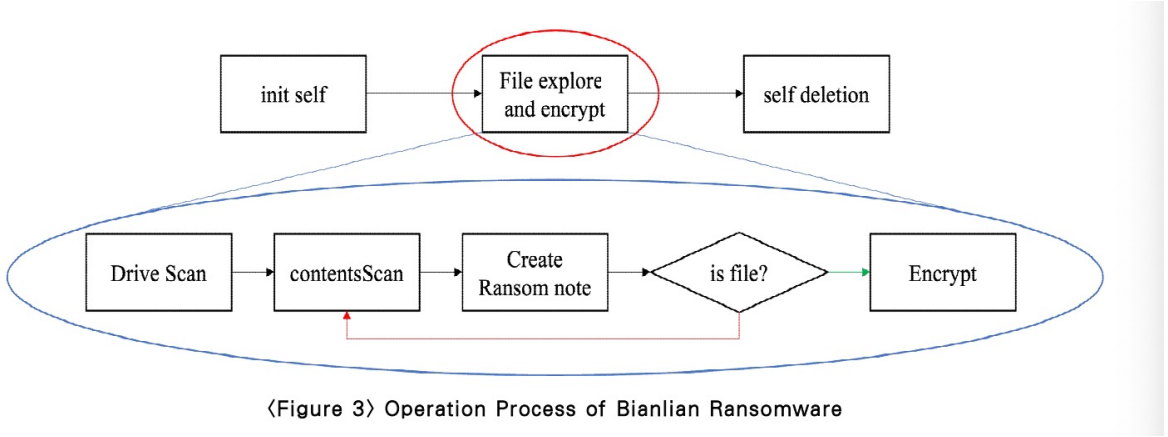
정적분석: Bianlian 랜섬웨어의 전체 흐름 및 세부 암호화 방식을 확인

동적분석: 전체 흐름 및 세부 암호화 방식의 교차검증 및 암호화 과정에 사용되는 Key, IV의 메모리상 위치를 확인

〈Table 2〉 Analysis Environment

Category		Information	Version	Note
Ransomware sample		SHA256 : 46d340eaf6b78207e24b6011422f1a5b4a566e493d72365c6a1cace11c36b28b SHA256 : eaf5e26c5e73f3db82cd07ea45e4d244ccb3ec3397ab5263a1a74add7bbcb6e2 SHA256 : 1fd07b8d1728e416f897bef4f1471126f9b18ef108eb952f4b75050da22e8e43		
HostPC	OS	Windows 10 x64	22H2 19045.2673	-
Guest	SW	Virtualbox	5.2.44	-
	OS	Windows 10 x64	1809 17763.379	4core CPU, 8GB RAM
Analysis Tool		IDA Pro	7.7	Decompiler
		AlphaGolang	-	Recover Golang symbol
		x64dbg	May 8 2021, 14:10:25	Debugger
		python	3.9.13	Detect bianlian Ransomware in memory, Extract key and IV
		Go	1.20.0	Decrypt infected file

Bianlian 랜섬웨어의 암호화 과정 및 암호화 요소 분석–분석 결과



〈Figure 3〉 Operation Process of Bianlian Ransomware

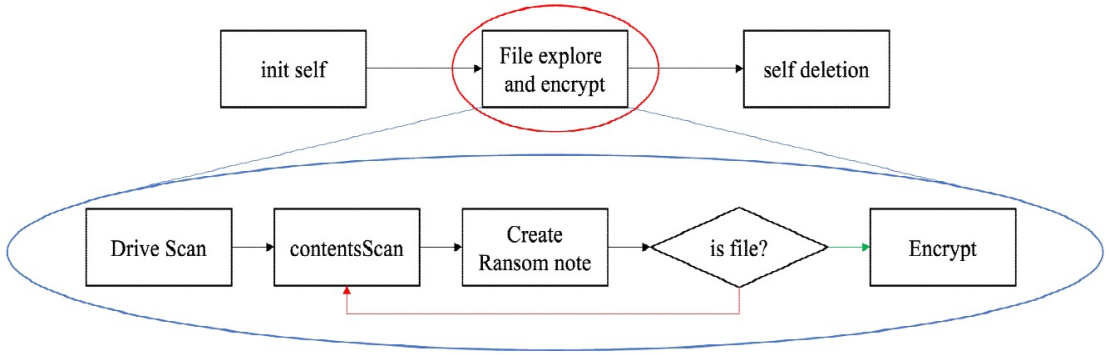
init self: 암호화 과정에서 암호화를 수행하지 않을 확장자와 경로를 지정해 메모리에 할당

File explore and encrypt: 피해자의 컴퓨터 내 모든 드라이브의 파일들을 AES 256-CBC 암호화하며 랜섬노트를 생성

self deletion: 암호화가 모두 끝난 후 자신을 스스로 삭제 및 메모리 제로화를 수행하여 추후 랜섬웨어 감염 사고에 대한 분석을 어렵게 함

Bianlian 랜섬웨어의 암호화 과정 및 암호화 요소 분석–분석 결과

암호화대상드라이브,파일및폴더탐색



〈Figure 3〉 Operation Process of Bianlian Ransomware

지정된 확장자를 가진 파일과 경로를 제외하고 재귀적으로 폴더를 탐색하며 탐색한 폴더마다 랜섬노트를 생성

〈Table 3〉 File Extensions and Paths Excluded in the Encryption Process

Extension	Path
.exe .sys .drv .dll .html .txt .mul .lnk bianlian	Appdata\Local\Microsoft Windows webroot Sophos

Bianlian 랜섬웨어의 암호화 과정 및 암호화 요소 분석–분석 결과

Bianlian랜섬웨어암호화과정

```
encryptFile(File){
    Data = File.read()
    Chaining_size = calculateChainingSize(Data.Length)
    Chaining_count = calculateChainingCount(Data.Length, Chaining_size)
    Idx = 0
    Encrypted_Data[0:0x34] = Data[0:0x34]
    for(Idx = 0x34; Idx = 0x34 + (Chaining_size * Chaining_count); Idx += Chaining_size){
        Encrypted_Data[Idx:Idx+Chaining_size] = AES_256_CBC.encrypt(Key, IV, Data[Idx:Idx+chaining_size])
    }
    Encrypted_Data[Idx:Data.length] = Data[Idx:Data.length]
    File.write(Encrypted_Data)
    File.rename(File.Name + ".bianlian")
}
```

〈Figure 4〉 Pseudocode of Encrypt Phase

Encrypt 단계

- 하드코딩 된 Key와 IV를 고정된 위치에서 사용하여 AES 256-CBC 모드 암호화 수행
- CBC 모드의 체이닝을 수행하는 블록의 크기(Chaining_size, bytes 단위)와 체이닝 블록 개수(Chaining_count)는 암호화 대상 파일의 길이에 따라 달라짐
- 암호화 대상 파일의 offset 0부터 0x 34(Data[0:0x34])는 암호화되지 않음
- 파일의 길이 내에서만 블록을 생성하므로 패딩 없이 암호화를 수행한 다는 특징
- 파일의 암호화가 완료되면 파일의 이름에 ".bianlian" 확장자를 추가

Bianlian 랜섬웨어의 암호화 과정 및 암호화 요소 분석–분석 결과

Bianlian랜섬웨어암호화과정

```
calculateChainingSize(Data_length){
    Chaining_size = (Data_length >> 12) << 12
    if(Chaining_size > 0x400000) {
        Chaining_size = (Chaining_size >>14) << 12
        if(Chaining_size >= 0x400000{
                Chaining_size = 0x400000
        }
    } else if(Chaining_size < 0x10){
        Chaining_size = 0x10
    }
    return Chaining_size
}

calculateChainingCount(Data_length, Chaining_size){
    temp1 = Data_length - 0x34
    Chaining_count = temp1/Chaining_size
    temp1 = temp1 >> 10
    if(temp1 > 0x400){
        temp2 = Chaining_size * 0.0009765625 * 0.0009765625
        Chaining_count = temp1 * 0.2 / temp2
        if(temp1 > 0x3ff){
                Chaining_count = temp1 >> 10
                if(Chaining_count < 500) ratio = 0.00015
                else if(Chaining_count == 500) ratio = 0.001
                else ratio = 0.00005
                Chaining_count = ratio * Chaining_count / (temp2 * 0.0009765625)
        }
    }
    return Chaining_count
}
```

〈Figure 5〉 Pseudocode of Calculating Chaining Size and Chaining Count

(체이닝을 수행할 블록의 크기와 체이닝 블록 개수 구하기)

- 체이닝을 수행할 블록의 크기: 암호화 대상 파일의 크기(Data_length, bytes)에 의해 결정

  파일 크기가 0x400000 bytes보다 작은 경우: 파일의 크기의 하위 12 bits를 0으로 만들어 이를 체이닝 블록 크기로 이용

  파일 크기가 0x400000보다 큰 경우: 파일의 크기를 4로 나눈 후 하위 12 bits를 0으로 만들어 이를 체이닝 블록 크기로 이용

  체이닝 블록의 크기가 0x10(16)보다 작아진다면: AES 256-CBC의 블록 크기인 0x10 bytes로 설정해 블록 간의 체이닝 없이 AES 256 암호화한 결괏값과 IV의 xor 연산을 수행

Bianlian 랜섬웨어의 암호화 과정 및 암호화 요소 분석–분석 결과

Bianlian랜섬웨어암호화과정

```
calculateChainingSize(Data_length){
    Chaining_size = (Data_length >> 12) << 12
    if(Chaining_size > 0x400000) {
        Chaining_size = (Chaining_size >>14) << 12
        if(Chaining_size >= 0x400000{
                Chaining_size = 0x400000
        }
    } else if(Chaining_size < 0x10){
        Chaining_size = 0x10
    }
    return Chaining_size
}

calculateChainingCount(Data_length, Chaining_size){
    temp1 = Data_length - 0x34
    Chaining_count = temp1/Chaining_size
    temp1 = temp1 >> 10
    if(temp1 > 0x400){
        temp2 = Chaining_size * 0.0009765625 * 0.0009765625
        Chaining_count = temp1 * 0.2 / temp2
        if(temp1 > 0x3ff){
                Chaining_count = temp1 >> 10
                if(Chaining_count < 500) ratio = 0.00015
                else if(Chaining_count == 500) ratio = 0.001
                else ratio = 0.00005
                Chaining_count = ratio * Chaining_count / (temp2 * 0.0009765625)
        }
    }
    return Chaining_count
}
```

〈Figure 5〉 Pseudocode of Calculating Chaining Size and Chaining Count

(체이닝을 수행할 블록의 크기와 체이닝 블록 개수 구하기)

- 체이닝 블록의 개수: 암호화 대상 파일의 크기와 체이닝 수행 블록의 크기에 의해 결정

먼저 암호화 대상 파일의 크기로부터 0x34를 빼고 나머지 값을 체이닝 블록의 크기로 나눠 횟수 계산

(체이닝 블록 개수 up -> 파일 암호화 시간 up)

-> 암호화 대상 파일의 크기가 클수록 암호화되는 비율은 줄어들도록 ratio 변수를 설정하고, 체이닝 블록의 개수를 연산 해 암호화 소요시간을 감소

Part 3

Bianlian 랜섬웨어 복호화 방안 연구

암호화가 완료된 후에는 랜섬웨어가 자체적으로 삭제되고 메모리도 제로화 -> Key와 IV를 미리 획득하지 않는 한 감염된 파일의 복원이 제한적

3장의 분석 내용을 통해 복호화 알고리즘을 설계

Bianlian 랜섬웨어가 작동 중인 경우 메모리상에서 Key와 IV 를 자동으로 추출해 감염된 파일을 복원할 수 있는 도구를 제안

검증: 본 연구에서 제안한 도구와 기존의 도구를 실험하여 비교, 기존 도구의 한계 극복 방법, 새로운 도구의 성능을 어떻게 검증하였는지

Bianlian 랜섬웨어 복호화 방안 연구

감염파일복호화알고리즘

```
decryptFile(Encrypted_File){
    Encrypted_Data = Encrypted_File.read()
    Chaining_size = calculateChainingSize(Encrypted_File.Length)
    Chaining_count = calculateChainingCount(Encrypted_Data.Length, Chaining_size)
    Idx = 0
    Decrypted_Data[0:0x34] = Encrypted_Data[0:0x34]
    for(Idx = 0x34; Idx = 0x34 + (Chaining_size * Chaining_count); Idx += Chaining_size){
        Decrypted_Data[Idx:Idx+Chaining_size] = AES_256_CBC.decrypt(Key, IV, Encrypted_Data[Idx:Idx+chaining_size])
    }
    Decrypted_Data[Idx:Data.length] = Data[Idx:Data.length]
    Encrypted_File.write(Decrypted_Data)
    Encrypted_File.rename(Encrypted_File.Name[0:-9])
}
```

〈Figure 6〉 Pseudocode of Decrypting Infected Files

복호화

- AES 256-CBC 복호화 알고리즘 이용

- 파일의 길이에 따라 CBC 모드의 체이닝을 수행할 블록 크기와 블록의 개수를 정하는 알고리즘을 적용

- 패딩이 수행되지 않았기 때문에 암호화 전후의 파일 크기는 동일

- 이러한 특성으로 암호화된 파일의 크기를 통해 원본 파일의 크기를 구할 수 있으며, 체이닝을 수행할 블록의 크기와 블록 개수를 <Figure 5>의 의사코드를 이용해 구할 수 있음

Bianlian 랜섬웨어 복호화 방안 연구

Key및IV추출

〈Table 4〉 Keys and IVs Extracted from Bianlian Ransomware Samples

Sample Hash	Entity	Captured Key & IV		
		RVA	Size	Value (Hex)
46d340eaf6b78207e24b6011422f1a5b4a566e493d72365c6a1cace11c36b28b	Key	0x1D82A0	0x20 (32 bytes)	97 94 12 FF 08 F7 D2 F0 BD 5F 7E 7E 2A 49 19 E9 BF 68 CC 7A AB AB 49 98 72 EC 82 2D DC DA 53 07
	IV	0x1D7910	0x10 (16 bytes)	0F C1 43 23 F7 A1 3C CA 05 69 EB CF AE 28 39 96
1fd07b8d1728e416f897bef4f1471126f9b18ef108eb952f4b75050da22e8e43	Key	0x1D82A0	0x20 (32 bytes)	63 3A 56 D0 38 88 69 23 7C 8D A6 B7 FC 09 F5 5D F3 54 08 C3 32 61 4F 69 2D 11 22 25 83 B3 6B 62
	IV	0x1D7910	0x10 (16 bytes)	FC 55 A6 0A 25 B0 47 2C FC 2C 6E BC 3D D8 9D AB
eaf5e26c5e73f3db82cd07ea45e4d244ccb3ec3397ab5263a1a74add7bbcb6e2	Key	0x1D82A0	0x20 (32 bytes)	27 CF AE 34 A8 3C 9A 7E 48 06 0E 18 A6 8A 23 39 17 27 1D B7 D5 14 C8 38 Fb 1E AE AE A8 9E 5C DE
	IV	0x1D7910	0x10 (16 bytes)	22 3B 67 AC 53 4F 99 38 CC 7B 1F 97 77 A9 58 40

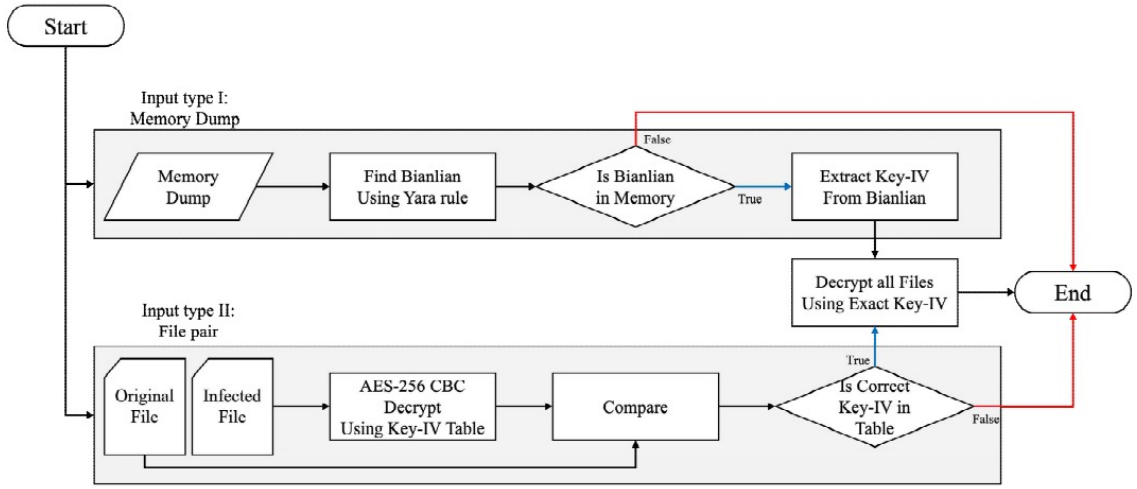
역공학을 통해 AES 256-CBC 알고리즘이 참조하는 값을 분석한 결과 Bianlian 랜섬웨어의 AES 256-CBC는 특정 위치에 있는 Key와 IV를 이용해 암호화를 수행하는 것을 확인

메모리 내에서 BaseImage 주소와 떨어진 거리인 RVA(Relative Virtual Address)가 샘플마다 동일하고, 이를 이용해 암호화를 수행

이 점을 이용해 메모리에서 실행 중인 Bianlian 랜섬웨어를 식별하고, 자동으로 Key와 IV를 추출하여 암호화된 파일들을 복호화 가능

Bianlian 랜섬웨어 복호화 방안 연구

Bianlian랜섬웨어복호화도구



〈Figure 7〉 Design of Bianlian Ransomware Decryption Tool Proposed in This Paper

본 논문의 분석 결과를 토대로 개발한 Bianlian 랜섬웨어 복호화 도구의 구조

기존도구(Avast의 'Decryption Tool for Bianlian): 이미 분석된 샘플에서 획득한 Key와 IV를 사용하여 감염파일을 복호화하는 'Input type II' 방식만을 채택

본 연구에서 개발한 도구: 추가적으로 'Input type I' 방식을 적용

이 방식은 Blac kberry의 Bianlian 랜섬웨어 탐지 Yara rule을 수정하여 메모리에서 Bianlian 랜섬웨어의 실행을 탐지하고, 실행 중인 랜섬웨어로부터 자동으로 올바른 Key와 IV를 추출하여 감염파일을 복호화하는 작업을 수행

Bianlian 랜섬웨어 복호화 방안 연구

기존도구와의비교(Avast의'DecryptionToolforBianlian)

〈Table 5〉 Experiment Environment

Category		Information	Version	Note
Ransomware sample		SHA256 : 46d340eaf6b78207e24b6011422f1a5b4a566e493d72365c6a1cace11c36b28b SHA256 : eaf5e26c5e73f3db82cd07ea45e4d244ccb3ec3397ab5263a1a74add7bbcb6e2 SHA256 : 1fd07b8d1728e416f897bef4f1471126f9b18ef108eb952f4b75050da22e8e43		
Guest	SW	Virtualbox	5.2.44	-
	OS	Windows 10 x64	1809 17763.379	4core CPU, 8GB RAM
Infected File		Windows default file	-	24,871 files were used
		Document files	-	50 files of various sizes were used (16 bytes ~ 2GB)

Avast의 'Decryption Tool for Bianlian': 이미 분석된 샘플의 Key와 IV만을 이용해 감염파일 복호화 가능
-> 이전에 분석되지 않은 새로운 샘플로 인한 감염 시 복호화는 제한적

Bianlian 랜섬웨어 복호화 방안 연구

복호화범위비교

HxD를 이용하여 기존 Bianlian 랜섬웨어 샘플 3개의 Key와IV를 수정하여 새로운 샘플 3개를 추가로 제작하고 가상환경에서 기존 샘플과 새로운 샘플을 실행시켜 파일 암호화를 진행

실험 결과

기존 도구

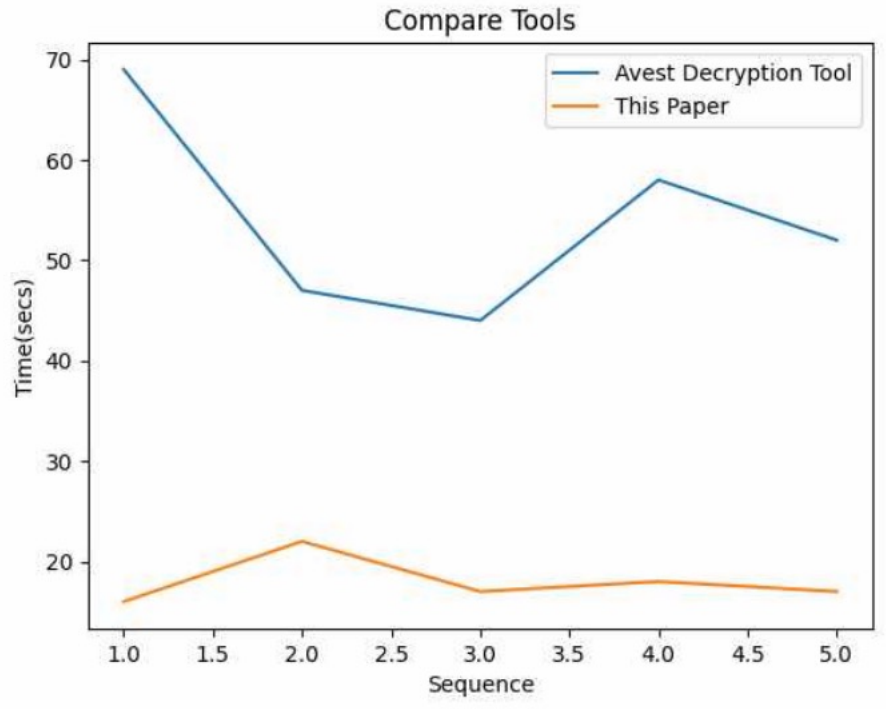
- Bianlian 랜섬웨어의 분석을 통해 기존에 획득한 9개의 Key-IV조합을 통해 암호화된 파일 복호화를 시도
- 기존 샘플에 의해 감염된 파일의 복호화는 정확히 수행 가능
- 새롭게 생성된 샘플에 의해 감염된 파일의 복호화 수행 불가

본 연구의 도구

- 메모리에서 실시간으로 Key와 IV를 성공적으로 추출
- 기존 샘플에 의해 감염된 파일과 새롭게 생성된 파일에 의해 감염된 파일 모두 효과적으로 복호화

Bianlian 랜섬웨어 복호화 방안 연구

복호화속도비교



〈Figure 8〉 Compare Execution Complete Times for Decryption Tools

복호화 작업 소요 시간을 5번에 걸쳐 측정

평균적으로 3배 차이 (기존 도구 57초, 본 연구의 도구 19초)

Part 4

결론 및 향후 연구

Bianlian 랜섬웨어의 동작 과정 및 암호화 메커니즘을 상세히 분석

효율적인 복호화 도구를 제시 (기존 도구보다 범위, 속도 증가)

한계점: Bianlian 랜섬웨어의 동작이 종료된 후에는 메모리에서 Yara rule을 이용한 식별이 불가능

-> 향후 연구 방향으로 제안됨

감사합니다

차해성, 서승희, 이창훈. (2023). 역공학을 통한 Bianlian 랜섬웨어 복호화 방안 연구.
디지털포렌식연구, 17(3), 135-145.