ISSN(Print) : 1976-5304, ISSN(Online) : 2713-931X
https://kdfs.jams.or.kr, DOI : 10.22798/kdfs.2022.16.1.64

랜섬웨어에서 메모리 덤프를 통한 파일 복호화에 관한 연구

김 승 환*, 손 태 식**
아주대학교 사이버보안학과 (학부생)*, (교수)**

A research on file decryption through memory dumps at ransomware

Seunghwan Kim*, Taeshik Shon**
Dept. of Cybersecurity, Ajou University (Undergraduate Student)*, (Professor)**

요 약

랜섬웨어는 2013년을 기점으로 급격하게 늘어나고 있다. 랜섬웨어는 사용자의 파일을 암호화하고 복호화하는데 돈을 요구하고 있다. 이렇게 파일이 암호화된 경우 복호화를 하기 위해서는 공격자에게 돈을 송금하거나 공격자를 체포해 키를 알아내는 방법이 주로 사용된다. 하지만 두 가지 방법 모두 성공할 확률이 낮으며 복호화를 하기 위해 키를 무작위 대입으로 찾는 방법만 남게 되며 이 경우는 키를 찾는데 아주 오랜 시간이 걸린다는 단점이 존재한다. 그렇지만 랜섬웨어에 의해 암호화가 진행되고 있는 도중에는 암호화에 사용되는 키가 메모리에 저장되어 있을 확률이 높다. 따라서 메모리 덤프를 통해 복호화 키를 추출하고 암호화된 파일을 복호화할 수 있을 것이다. 본 연구에서는 2017년 유명한 랜섬웨어인 WannaCry를 인위적으로 감염시키고 메모리를 덤프해 분석하였다. 해당 연구를 위해 WannaCry의 암호화 방식과 감염 구조에 관해 사전 연구가 필요하다. 메모리 덤프를 분석해 복호화 키를 추출하고, 이를 응용해서 다른 랜섬웨어에서도 사용할 수 있는지에 대해 연구하였다.

주제어 : 랜섬웨어, WannaCry, 메모리 덤프

ABSTRACT

Ransomware has been on the rise since 2013. Ransomware demands money to encrypt and decrypt users' files. If the file is encrypted in this way, in order to decrypt it, sending money to the attacker or arresting the attacker and obtaining the key are mainly used. However, both methods have a low probability of success, and only the method of finding the key by random assignment for decryption is left. In this case, there is a disadvantage that it takes a very long time to find the key. However, there is a high probability that the key used for encryption is stored in memory while encryption is in progress by the ransomware. Therefore, you will be able to extract the decryption key from the memory dump and decrypt the encrypted file. In this research, WannaCry, a famous ransomware in 2017, was artificially infected and memory was dumped and analyzed. For this study, prior research on WannaCry's encryption method and infection structure is required. We analyzed the memory dump to extract the decryption key, and studied whether it can be used in other ransomware by applying it.

Key Words : Ransomware, WannaCry, Memory Dump

▪ Received 11 January 2022, Revised 12 January 2022, Accepted 31 March 2022
▪ 제1저자(First Author) : Seunghwan Kim (asdf7845120@naver.com)
▪ 교신저자(Corresponding Author) : Taeshik Shon (tsshon@ajou.ac.kr)

Ⅰ. 서 론

파일을 암호화해 피해를 주는 랜섬웨어는 1989년 처음 “에이즈”라는 이름으로 등장하였다[1]. 이 당시는 파일을 암호화해서 돈을 요구하더라도, 돈을 안정적으로 회수할 수 없었기 때문에 공격자는 파일이 암호화되어 사용할 수 없는 상태를 사용해 피해를 주려고 하였다. 해당 공격 방식은 대칭 키 암호화 방식이었기에 방어자가 암호화 키를 찾아 암호화를 해제하면 해결되어 공격자에게 인기가 없었고 해당 방식의 랜섬웨어는 사장되었었다. 하지만 중앙은행 없이 송금할 수 있고, 경찰의 추적을 회피할 수 있는 비트코인이 발명됨과 동시에 대칭 키 암호화가 아닌 공개 키 암호화 알고리즘을 사용하면서 랜섬웨어는 다시 유행하기 시작했다.

비트코인을 응용 최초의 랜섬웨어인 CryptoLocker는 2013년경 최초 감염 사례가 보고되었다[1]. 해당 악성 프로그램은 Rivest, Shamir, Adleman 3명이 만든 공개 키 암호화 알고리즘을(RSA) 사용해 파일들을 암호화하고 비트코인 주소를 보여주면서 돈을 요구하는 형태를 보여주었고, 이후 등장하는 랜섬웨어 들의 표준적인 모델이 되었다. CryptoLocker는 러시아와 중국에서 큰 피해를 주었고, 비트코인의 거래 내용을 통해 많은 돈을 벌었다는 것을 보고 많은 공격자가 CryptoLocker를 모방해서 여러 랜섬웨어 들을 만들었다. 그 이후로 2017년 미국 국가 안보국(NSA)에서 유출된 Windows의 파일 공유 프로토콜인 Server Message Block(SMB) 취약점을 이용한 WannaCry가 등장하면서 대중에게 랜섬웨어의 위험성과 심각성을 각인하였다[1].

랜섬웨어에 감염되어서 암호화된 파일을 복호화하기 위해서는 공격자에게 원하는 돈을 보내야 한다. 예전 극소수의 랜섬웨어는 돈을 보내면 파일을 복호화해주는 경우가 많았지만, 대다수는 돈만 받고는 복호화를 해주지 않는 경우가 많다. 따라서 파일 복호화를 하기 위해 RSA 공개키를 브루트포싱 공격을 통해 비밀키를 알아내거나, 공격자를 체포해서 RSA 비밀키를 알아내 복호화해야 한다. 하지만 앞서 말한 방법을 사용하게 되면 높은 확률로 RSA 비밀키를 알아내지 못해 데이터를 잃게 되거나, 알아내더라도 단기간에 파일을 복호화하기도 어렵다. 따라서 이러한 방법 이외에 다른 방법을 찾고자 한다.

랜섬웨어는 파일을 암호화하기 위해 RSA 또는 고급 암호화 표준(AES)과 같은 암호화 알고리즘을 사용한다. 이런 암호화를 위해 랜섬웨어가 직접 암호화 알고리즘을 코딩하지는 않고 OpenSSL, WinCrypt와 같은 오픈소스나 내장 윈도우 암호화 함수를 사용하는 경우가 많다. 이러한 오픈소스나 내장함수들은 암복호화를 위해 힙 메모리를 사용하는 경우가 많고, 이 메모리를 사용한 뒤 그대로 남겨 놓는 경우가 많다. 특히 암복호화에 사용되는 키가 그대로 메모리에 남아있는 경우가 많다. 이렇게 메모리에 남은 정보들을 가지고 우리는 랜섬웨어의 RSA 비밀키를 알아내는 지표로 사용할 수 있다.

이번 복호화 방식으로 만들 랜섬웨어 복호화 도구는 감염된 즉시 메모리 덤프를 하고 복호화 도구를 실행시켜 덤프 된 메모리에서 암호화된 파일의 복호화 키를 찾고 복호화를 할 예정이다. 하지만 새로운 유형의 랜섬웨어일 경우 메모리에 어떻게 키 데이터가 저장되는지 모르므로 메모리 분석이 필요하다. 그래서 랜섬웨어에 감염된 직후 메모리 덤프하고, 이를 분석하여 복호화 키를 추출할 수 있는지 확인하고, 추출이 가능하다면 복호화 키를 추출한 뒤 복호화를 하는 시도를 할 것이다. 따라서 랜섬웨어에 감염되더라도 메모리 덤프해서 암호화된 파일을 복호화할 수 있는지를 판단해볼 것이다.

본 논문의 3절에서는 유명한 랜섬웨어인 WannaCry를 사용해 VMware를 사용한 가상 환경인 Windows 7에서 감염을 할 것이고, 메모리 덤프를 생성한 다음 WannaCry를 리버스 엔지니어링 하며 얻어낸 정보, 자체적으로 만든 프로그램을 사용해 분석 방법을 연구할 것이다. 그리고 4장에서는 3장에서 고안해낸 방안으로 복호화 정보를 찾는 자동화 도구를 제작하고, 이를 통해 암호화된 파일을 복호화하는 과정을 나타낼 것이다. 5장에서는 테스트 결과 분석과 복호화 방법의 한계와 의의에 관해 설명하고 6장에서는 결론으로 마무리한다.

Ⅱ. 관련 연구 및 배경 지식

본 연구에서 사용할 WannaCry 랜섬웨어의 구조와 암호화 방법은 여러 자료에서 이미 설명하고 있다[2]. 또한, WannaCry에 감염되었더라도 바로 복호화 도구를 사용하면 암호화된 파일을 복호화할 수 있는 도구 또한 인터넷에 제시되어 있다[3]. 하지만 우리가 원하는 내용인 랜섬웨어에 감염된 컴퓨터의 메모리에서 암호화 정보를 추출하고, 이를 분석해서 복호화키를 어떻게 추출하였는지에 대한 자료는 찾아보기 어렵고, 기술 문서에 있는 내용을 재현하더라도 같은 결과가 나타나지 않았다[4]. 따라서 해당 방식에 대해 더 구체적으로 접근하고자 연구 주제로 정하였고, 이를 위해서는 랜섬웨어가 어떤 암호화 방식을 사용하고 어떻게 키를 생성하고 처리하는지 랜섬웨어를 분석해서 랜섬웨어의 특징과 기능 구조를 명확하게 알고 있어야 한다.

이번 연구에서 실험 대상으로 선정된 WannaCry의 기능은 크게 두 가지로 나눌 수 있는데, 감염 전파와 파일 암호화이다. 첫 번째로 감염 전파는 랜섬웨어가 감염된 컴퓨터 주위에 있는 감염되지 않은 컴퓨터에 공격을 시도해서 감염시키는 기능을 의미한다. WannaCry가 감염을 전파하는 방식은 NSA에서 유출된 Microsoft의 SMB라 하는 파일 공유 시스템에서 원격으로 프로그램이 실행되는 취약점(EternalBlue)을 사용하고 있다[5]. WannaCry가 공격을 시작했을 당시에는 아직 Microsoft의 취약점 패치가 이루어지지 않았기 때문에 전파력이 강했고 많은 컴퓨터에서 큰 피해를 주었다. 해당 감염 전파 기능은 본 연구에서 연구하려는 암호화된 파일을 복호화하는 주제와는 관계가 없기에 이 논문에서는 해당 기능의 세세한 부분을 다루지 않는다.

두 번째 기능인 파일 암호화 및 복호화 기능은 공격자가 금전을 요구하기 위해 파일을 인질로 잡는 기능이다. 해당 기능의 핵심적 역할은 파일을 암호화해서 공격자 이외에 파일을 복호화할 수 없게 만들고, 공격자가 금전을 획득했다면 복호화를 통해 파일을 복원하여 돈을 주면 파일을 복원할 수 있다는 신용을 쌓는 것이다. 이 부분에서 핵심적인 부분은 암호화 부분인데 WannaCry는 아래와 같은 절차를 가지고 파일 암호화를 수행한다.

1) 공격자는 미리 랜섬웨어에 RSA public key (SPUB.key)를 저장해 둔다.
2) 공격이 시작되면 랜섬웨어는 RSA private key (CPRIV.key)와 public key (CPUB.key)를 생성한다.
3) 생성한 RSA pulbic key (CPUB.key)는 파일 시스템에 저장하고, 생성한 RSA private key (CPRIV.key)는 공격자가 미리 저장해 둔 키(SPUB.key)로 암호화하고 파일 시스템에 저장한다.
4) 암호화할 파일마다 AES 키를 생성해 암호화하고 해당 키를 생성한 RSA public key (CPUB.key)로 암호화해서 저장한다.

해당 절차에서 가장 중요한 것은 2번인 것을 알 수 있다. 메모리에서 랜섬웨어가 생성한 RSA private key (CPRIV.key)를 찾기만 한다면 공격자의 RSA private key (SPRIV.key)를 찾지 않고도 모든 암호화된 파일을 복호화할 수 있다. 따라서 메모리에서 RSA private key (CPRIV.key)를 찾는 것이 우리의 목표이다.

WannaCry는 Windows에서 제공하는 WinCrypt API를 사용하고 있다[2]. 해당 API 내부에서는 RSA 키를 관리하기 위해 아래 사진과 같은 구조체를 사용하고 있다[6].

```
typedef struct _RSAPUBKEY {
  DWORD magic;
  DWORD bitlen;
  DWORD pubexp;
} RSAPUBKEY;
```

〈Figure 1〉 WinCrypt RSAPUBKEY structure

CPRIV.key를 찾는 것이 이 연구의 목적이므로, 메모리에서 최우선으로 발견해야 할 데이터의 구조가 되는 것이다.

하지만 WinCrypt API를 내부에서 여러 방식으로 사용하고 있으므로 〈Figure 1〉과 같은 구조는 많이 발견될 수 있다. 따라서 찾은 RSA 키값이 CPRIV.key인지 확인하기 위해 검증할 방법이 필요하다. 검증할 방법으로는 RSA의 특징인 Exponent와 Modulus가 Public key와 Private key 모두 같다는 것을 이용하는 것이다. CPUB.key는 컴퓨터에 저장되어 있으므로, CPUB.key에서 추출한 RSA의 Exponent와 Modulus를 추출한 RSA 키와 같은지 비교하고, CPRIV.key가 맞는지 확인하는 것이다.

```
PUBLICKEYSTRUC  publickeystruc;
RSAPUBKEY rsapubkey;
BYTE modulus[rsapubkey.bitlen/8];
BYTE prime1[rsapubkey.bitlen/16];
BYTE prime2[rsapubkey.bitlen/16];
BYTE exponent1[rsapubkey.bitlen/16];
BYTE exponent2[rsapubkey.bitlen/16];
BYTE coefficient[rsapubkey.bitlen/16];
BYTE privateExponent[rsapubkey.bitlen/8];
```

〈Figure 2〉 PRIVATEKEYBLOB structure

메모리에서 RSA 키들이 〈Figure 1〉과 같은 형태를 가지고 있지 않을 수 있다. RSA 키를 담고 있는 영역이 메모리 할당에서 해제된 상태라 RSA 키의 앞부분이 여러 값으로 덮어 씌워졌을 가능성도 존재한다. 〈Figure 2〉를 보면 CPRIV.key를 저장하기 위해 사용되는 PRIVATEKEY BLOB 구조체에는 RSA의 p, q 값이 저장되는 것을 알 수 있다[7]. 따라서 메모리에서 Prime 값이 존재하는지 확인하고, Modulus를 통해 올바른 값인지 확인할 수 있다.

Ⅲ. 감염된 컴퓨터의 메모리 덤프 및 분석 방안

3.1 실험 환경 및 계획

실제 컴퓨터에 랜섬웨어를 감염시키는 것은 하드웨어 드라이버를 건드려 하드웨어 오작동을 유도해 물질적인 피해를 받을 수 있다. 그래서 실사용 컴퓨터에서 랜섬웨어 감염 테스트를 진행하는 것보다는 가상 PC 환경에서 테스트하는 것이 좋다. 따라서 이번 문서에서는 VMware 프로그램을 사용한 가상 환경에서 랜섬웨어 감염을 진행할 것이다. 세부적인 가상 하드웨어는 4개의 CPU 코어와 4GB의 RAM, 30GB 저장공간으로 설정하였다. 그리고 랜섬웨어를 감염시킬 가상 PC의 OS는 Windows 7을 사용하기로 하였다. 이번 테스트에서 사용할 랜섬웨어인 WannaCry가 유행할 당시 주로 사용되었던 운영체제가 Windows 7이었기 때문이다[8].

이번 실험의 흐름도는 아래와 같은 순서로 이루어질 것이다.

1) 검증 파일과 WannaCry 랜섬웨어를 가상 PC에 저장하고 가상 PC를 감염시킨다.
2) 감염된 가상 PC에서 WannaCry 프로세스의 메모리를 덤프한다.
3) 해당 덤프 된 메모리에서 복호화 키가 존재하는지 분석한다.
4) 복호화 키가 존재한다면 해당 키로 검증 파일을 복호화하고, 복호화가 잘 되었는지 확인한다.

이 중 3번과 4번에서는 Python Script를 사용해 분석하고 복호화할 것이다.

3.2 PC 감염

WannaCry에 감염된 PC를 만들기 위해 랜섬웨어 실행 파일과 랜섬웨어에 의해 암호화되어 나중에 복호화해야 할 검증 파일이 필요하다. 랜섬웨어 실행 파일은 VirusShare에서 찾을 수 있었다. 그리고 검증 파일은 문서 파일을 만든 다음 내용을 임의로 적어 다른 문서와 혼동이 없게 하였고, HASH 값을 기록해 두어 후에 파일을 비교할 때 같은 파일인지 확인할 수 있게 하였다.

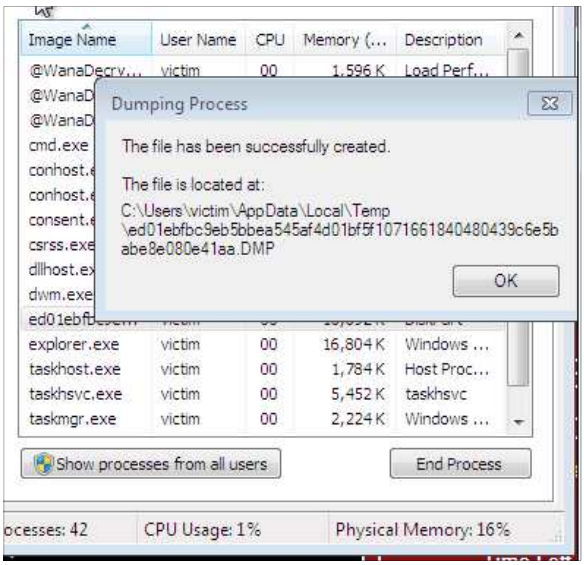
〈Table 1〉 File SHA-256 value

파일명	HASH
wannacry.exe	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
sample1.docx	269329fc7ae54b3f289b3ac52efde387edc2e566ef9a48d637e841022c7e0eab

〈Figure 3〉 PC infected with ransomware

랜섬웨어를 실행하게 되면 위 그림과 같은 금전을 요구하는 프로그램이 나오게 된다. 비트코인 지갑의 주소를 보여주면서 3일의 시간이 지나면 복구 가격을 인상한다고 되어 있고, 7일이 지나면 파일 복구가 불가능하다고 설명하고 있다.

3.3 메모리 덤프 생성



〈Figure 4〉 Memory Dumping

메모리 덤프 파일을 획득하는 방법에는 VMware에서 가상 컴퓨터 일시 정지를 통한 가상 컴퓨터의 전체 메모리 덤프를 획득하는 방법이 있다. 하지만 해당 테스트에서는 실사용 컴퓨터에서 메모리 덤프를 하는 것을 가정했기 때문에 가상 컴퓨터 일시 정지를 통한 메모리 덤프는 부적합하고, 전체 메모리 덤프를 사용하는 것은 분석할 정보가 너무 많아 분석에 시간이 많이 들 수 있다. 따라서 전체 프로세스 메모리 덤프를 사용하기로 했다.

3.4 메모리 덤프 분석 방안

```
BOOL __cdecl sub_10004350(int a1, int a2)
{
  return CryptGenKey(a1, 1, 0x8000001, a2) != 0;// CryptGenKey(hProv, AT_KEYEXCHANGE, 0x8000001, &hKey)
                                                // Create 2048 RSA
}
```

```
*pdwDataLen = 4096;
result = (BYTE *)CryptExportKey(hKey, 0, dwBlobType, 0, pbData, pdwDataLen);// CryptExportKey(hKey, 0, PUBLICKEYBLOB, 0, Data, &len);
                                                // Export RSA Private Key
```

〈Figure 5〉 RSA key generate and Private key extract by ransomware

메모리 덤프를 수행했으면 이번 테스트에서 찾아야 하는 포렌식 정보를 명확하게 알아야 한다. 2장의 관련 연구 및 배경 지식에 의하면 WannaCry 랜섬웨어는 자체적으로 RSA 키를 생성하는 기능이 있다. 해당 기능을 랜섬웨어 실행 파일에서 찾은 결과 〈Figure 5〉와 같이 CryptGenKey와 CryptExportKey를 통해 2048 RSA 키를 생성하고 특정한 형식으로 생성한 RSA Private Key(CPRIV.key)를 추출하는 기능이 있었다. 그리고 추출한 Private Key는 미리 저장되어 있던 SPUB.key로 암호화한 후 00000000.eky로 저장하게 된다.

랜섬웨어도 원본 CPRIV.key는 사용하지 않으므로 CryptDestroyKey를 호출해 키를 삭제한다. MSDN의 CryptDestroyKey 문서를 참고하면 public/private key pair로 생성된 키는 해당 함수에서 제거되지 않고, 오직 키의 핸들만 제거하는 것을 알 수 있다[9]. 따라서 메모리에는 CPRIV.key 원본 데이터가 남아있다는 것을 알 수 있고, 이것이 이번 테스트에서 찾아야 하는 포렌식 정보이다.

주소	Hex																ASCII
0018D584	07	02	00	00	00	A4	00	00	52	53	41	32	00	08	00	00	.....¤..RSA2....
0018D594	01	00	01	00	1F	72	E3	62	ED	33	B3	30	8C	EB	2A	20	.....rãbí3³0.ë*
0018D5A4	8E	76	69	A4	3F	12	1D	22	E5	56	B0	DA	6A	FA	15	C4	.vi¤?.."åV°Újú.Ä
0018D5B4	8F	51	E6	3A	DC	F1	09	E1	30	06	43	18	F4	7F	38	25	.Qæ:Üñ.á0.C.ô.8%
0018D5C4	2E	51	74	A1	6F	F5	4D	18	4D	34	25	97	4A	50	72	FD	.Qt¡oõM.M4%.JPrý
0018D5D4	55	41	5D	47	C0	37	6F	AA	3B	6D	84	A6	BC	FA	D1	5E	UA]GÀ7oª;m.¦¼úÑ^
0018D5E4	4A	80	49	DF	6A	A6	71	70	8B	81	A0	43	6B	7E	BF	A5	J.Ißj¦qp.. Ck~¿¥
0018D5F4	BA	7D	D8	A2	6E	4B	C6	47	19	D6	67	69	22	52	F0	AE	º}Ø¢nKÆG.Ögi"Rð®
0018D604	62	64	D4	C0	9C	06	D9	AA	FB	2F	69	C1	15	56	33	B6	bdÔÀ..Ùªû/iÁ.V3¶
0018D614	EB	A7	47	E2	0C	A0	AF	8D	C0	DC	2E	BC	A8	A2	5A	78	ë§Gâ. ¯.ÀÜ.¼¨¢Zx
0018D624	68	75	74	EF	29	FC	7C	26	86	0E	59	B3	45	AB	4C	4C	hutï)ü|&..Y³E«LL
0018D634	7A	7B	56	36	DA	BC	78	D4	14	29	CF	8F	75	22	48	43	z{V6Ú¼xÔ.)Ï.u"HC
0018D644	8F	2D	02	46	AC	2A	F0	83	67	C4	C4	52	B5	94	08	6F	.-.F¬*ð.gÄÄRµ..o
0018D654	92	10	3A	88	AF	6A	74	4C	68	6A	3D	6A	30	AC	2D	54	..:.¯jtLhj=j0¬-T

〈Figure 6〉 pbData contents of CryptExportKey

〈Figure 6〉은 x64dbg 디버깅 도구로 알아본 〈Figure 5〉의 CryptExportKey의 pbData 내용이다. 데이터 내용을 보면 〈Figure 1〉의 구조가 포함된 것을 보여주고 있다. 빨간 박스는 Microsoft의 파일 형식인 공개키 BLOB의 헤더 부분이고 파란 박스는 RSA의 Exponent 값이며 나머지 부분은 RSA의 Modulus 값이다[10]. 헤더 부분의 RSA1은 Public Key를 의미하고 RSA2는 Private key를 의미한다[6]. RSA의 특징상 Exponent 값과 Modulus 값은 Public Key와 Private key 모두 같으므로 저장된 CPUB.key에서 추출해 키워드로 사용할 수 있다. 따라서 메모리 덤프에서 CPRIV.key를 찾기 위해서 "RSA2"라는 키워드와 앞서 말한 Exponent와 Modulus를 사용하면 된다.

해당 키워드가 없다면 〈Figure 2〉로 인해 메모리에 P, Q가 저장된다는 것을 이용해 메모리 덤프에서 0x80 크기를 읽어 Modulus에 나머지 연산을 하여 해당 값이 P, Q가 맞는지 확인하고 이를 사용해 RSA Prviate key를 찾아내야 한다. 해당 방법은 Script 작성을 통해 구현할 수 있다.

Ⅳ. 메모리 분석 및 암호화된 파일 복구

4.1 메모리 분석

프로세스 메모리 덤프에서는 "RSA2"라는 키워드가 존재했지만 모두 다른 Private key였고, Modulus 키워드는 존재하지 않았다. 따라서 마지막 방안인 메모리 덤프에서 RSA P, Q를 찾는 것으로 결정하였다. 해당 방식은 Python Script를 사용하였다.

```python
# Parse N and E
E = unpack("<I",pky_Data[0x10:0x14])[0]
N = int.from_bytes(pky_Data[0x14:0x114],byteorder='little')

found_p = False
for i in range(len(DMP_Data)-0x80):
    P = int.from_bytes(DMP_Data[i:i+0x80],byteorder='little')
    if P == 0 or P == 1:
        continue
    if N % P == 0:
        found_p = True
        break
    del P

if found_p == False:
    print("Failed to Find Prime number!")
    sys.exit()
```

〈Figure 7〉 Find Prime at Dump

먼저 3.4절에서 말한 CPUB.key인 00000000.pky에서 E와 N을 추출한 다음, N은 P와 Q의 곱셈으로 만드는 RSA의 원리를 사용해 P와 Q를 검증할 것이다. RSA 키의 길이가 0x80이고, 메모리의 어느 부분에 P 또는 Q가 저장되어 있을지 모르므로 메모리 덤프 파일에서 0x80만큼 Sliding Windows 알고리즘을 통해 P를 찾는다.

```python
# Calculate Q, D
Q = N // P
phi = (P-1)*(Q-1)
gcd, a, b = egcd(E, phi)
D = a

print("Found P, Q")
print("P:",hex(P))
print("Q:",hex(Q))

# Save RSA Private key as PEM
key_params = (N, E, D, P, Q)
RSA_Private_key = RSA.construct(key_params)
RSA_Private_key_PEM = RSA_Private_key.export_key()

with open("decrypt_key.pem","wb") as f:
    f.write(RSA_Private_key_PEM)

print("Save Private key at decrypt_key.pem!")
```

〈Figure 8〉 Calculate E and D

만약 P를 찾았다면 Q는 N을 P로 나눔으로서 알아낼 수 있고, D는 〈Figure 8〉과 같이 RSA에 나온 연산식을 통해 알 수 있다[10].

```
PS C:\Users\asdf7\Desktop\3-1\소융보\과제\논둠> python .\parser.py .\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.DMP .\00000000.pky
Found P, Q
P: 0xe4dbbc36655809a2bab2e64a39a0be9871d689bc69fcc69b8ae7ddd10ae872e27e29edeff9c5e4e8e704739876ad35f4218ffe6ed9f2ee69db76521e0949abb0f3fab71013169edd8dfe44c590e80fee0e9f67cf26826afa3a5cac34d9f263f7e6a8bb110b3b17b2b7e15d26829d7fb9505804c4ac18a4f328a5076d3e11ebc3
Q: 0xd8d55ff686e951de7eb847411066a5ab69abf7e649409ba9d0c3dae01ee9a3121c1839da2d10f66caf1cb57473c6719e791f98f217829ffc90617b6326e4dd354afa7869b07c161727753e510024cadb8961b04d248f38336f3f41f5ad4219c57b531f939a5ec84a95025bf2660514b5ae0aceeeef2aad6b4ac019c91e5a8353
Save Private key at decrypt_key.pem!
```

〈Figure 9〉 Finding P and Q through the parser

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAwdgna21fP/WVCpJ94JxfL54/lBIdbnLn9gBhR+nhMILZXSb5
5YFSmfFa086RsSy7vMPw3GN00VoFOpyYsFBmY48zCnYYX0KOj0MlVoKdrPBjLi7v
Kb/YyPTBH+UvPfV+V8q8W78hFnGPt7g2BBfxy/YvQWitlc1n8EIvrV7grd4mcoBI
Q95rADsXftOXEZzdw1BV+6GeUkoKsPlBCp/p9qr+WmUHHlx4hVjqddeUokS19pH2
3XBeapnOMgnxsYfLv2n3i754NcdtxWBfXBvkoeFtqpu1q/z32UxApAtzPMXfnlIv
KBIlmy4ImuoJLUqE77j2kdqgFnO2VGI3CAI5OQIDAQABAoIBAAva1E5VmgWnVz4W
R2x9I7Ud0HWVxgcDL/xzeX+UZhk5Z2GzCIVoq0yO3icl8vMK8VBSC2DvFC3Ze4hi
G1RtFlXwZAa3wjXRNDa+vIHV6x6wdOBWLno3KWLg1Q7GTWVUpi1oUKyru7JLr6Zk
taj6beczXyB47mTC8yBWa+QGcWcwm5d4L9Pj3xa9W3R9H3kIjSEIDXyC15rWLn37
vNDaxaR6sYfGwQCefSx0FHf+SI4MZm6o1JJh/GbFCNSNKIiCXjbE7Zdr6K1ANdWL
2u/QM6Hum505/aMC6sKP3ftggxXbxxMwjF/+SwOkyUSmLX9OM3lW8zr0VRAI3NcU
+p2Dw6ECgYEA5Nu8NmVYCaK6suZKOaC+mHHWibxp/MabiufdQrocuJ+Ke3v+cXk
6OcEc5h2rTX0IY/+btny7mnbdlIeCUmrsPP6txATFp7djf5ExZDoD+4On2fPJoJq
+jpcrDTZ8mP35qi7EQs7F7K34V0mgp1/uVBYBMSsGKTzKKUHbT4R68MCgYEA2NVf
9obpUd5+uEdBEGalq2mr9+ZJQJup0MPa4B7poxIcGDnaLRD2bK8ctXRzxnGeeR+Y
8heCn/yQYXtjJuTdNUr6eGmwfBYXJ3U+UQAkytuJYbBNJI84M28/QfWtQhnFe1Mf
k5peyEqVAlvyZgUUta4Kzu7vKq1rSsAZyR5ag1MCgYAVo+V/K0QQ3S9UPUxleJqt
JY6IxaUr/se0ccpauvJR8rLFWjdn4duMXlPgbllCdaiZB9miN7+t5ZqX7JnkfO/c
MqXRa5aye+EtTQ7Js7MN1o/RaOqAlLJy6NPcgL7fGa7WQ7w0l2BGFGMozzsV/LiY
GE3KXaJ8ThSkEj7KiKT/KwKBgH5qwJ0ZgPRjVNwZZtyqC9LJNCK+11xcpP/7ACQE
wdB6IXkaWZJGvnUVDxbXwxwiZ/NUMbaLcHPYK1E6P8+QRUvp3+fOLr/ZnTw7Bc7x
14GRxyH2SLOU8H5EUMwi8RWFsvY4u0MyECqUGRYcB1HlinJ1+wU0lqn6G4CkSSRR
lxHTAoGAKNabSDciLJO9XgfYNQWS4cRg9o7d0yvpw4t4akyzEJK88Ob3xsEVmAWQ
vJaSgT/N5v6ijIzhg8m0bkfI2HZoIYtL8oraAkTve6y/+5pCTuM/glQ6we4s6bvX
udgb3Y4GA/FrtSiDA0CwAtDTtxGzw2dNb3iXl4KN7ygQD/5JGnY=
-----END RSA PRIVATE KEY-----
```

〈Figure 10〉 Private key parsed from memory dump

Python Script를 실행해 보니 CPUB.key과 정보가 일치하는 P를 구할 수 있었고, 결과적으로 CPRIV.key를 메모리 덤프에서 찾아내었다. 그리고 CPRIV.key의 정보를 〈Figure 2〉와 같은 Key BLOB 형식이 아닌 PEM 형식으로 저장하였는데, 이는 Key BLOB 형식은 Python 기본 암호화 라이브러리에서 지원하지 않기 때문에 편의성을 위해 PEM으로 저장하였다.

4.2 암호화된 파일 복호화 도구

WannaCry에서는 추출한 Private key를 Windows의 Key BLOB 형식인 00000000.dky라는 파일로 저장하고, Decrypt 버튼을 누르면 자동으로 암호화를 해제해 준다. 하지만 암호화된 파일과 4.1절에서 찾은 Private key만 가지고 있다는 가정하에 Python Script를 작성하였다.

```
typedef struct _wana_hdr {
    uint8_t magic[8];           // "WANACRY!"
    uint32_t key_len;           // Key length 0x0100 (256) bytes
    uint8_t key[256];           // Encrypted key
    uint32_t file_code;         // File type code, 0x03 or 0x04
    uint64_t file_length;       // Length of original file
    uint8_t *encrypted_data;    // Ciphertext
}  wana_hdr_t;
```

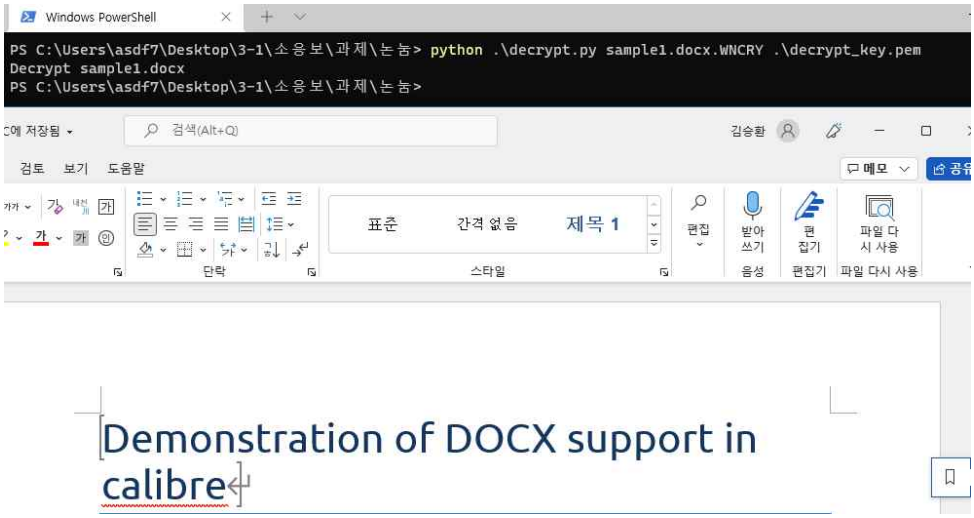
〈Figure 11〉 Wannacry encrypted file header structure

WannaCry는 암호화된 파일을 〈Figure 10〉와 같은 구조로 저장하고 있다[2]. key는 AES-128-CBC로 암호화된 파일을 복호화하는 key이나 CPUB.key로 암호화되어 있다. 하지만 4.1절에서 찾은 CPRIV.key를 가지고 있으므로 복호화할 수 있다.

```
# Load private key
rsa_private_key = RSA.import_key(decrypt_key_pem)
cipher = PKCS1_v1_5.new(rsa_private_key)

# Decrypt aes key
decrypted_aes_key = cipher.decrypt(encrypted_aes_key[::-1],b'error')

# Load AES 128 CBC key
crypto = AES.new(decrypted_aes_key, AES.MODE_CBC,b'\x00'*16)
# Decrypt data
decrypted_data = crypto.decrypt(encrypted_data)

Decrypted_File_Name = Encrypted_File_Name.replace(".WNCRY","")
with open(Decrypted_File_Name,"wb") as f:
    f.write(decrypted_data[:file_length])
```

〈Figure 12〉 File decrypt script

decrypt_key_pem은 4.1에서 찾은 CPRIV.key이고, AES에는 key와 IV 값이 필요한데, WannaCry에서 IV를 설정하지 않았기 때문에 00 값으로 설정하면 된다.



〈Figure 13〉 Decrypted file

Script로 암호화된 검증 파일을 복호화하였고 원본 파일과 같은 HASH 값을 가지고 있었다.

Ⅴ. 토 의

5.1 종합적 결과 분석

이번 테스트에서는 WannaCry에 감염된 PC의 암호화된 문서들을 메모리 덤프를 통해 성공적으로 복호화할 수 있었다. 메모리 덤프를 통한 랜섬웨어 복호화를 하기 위해서는 랜섬웨어의 암호화 방식에 취약한 부분이 있어야 하는데, 이번 실험 대상이었던 WannaCry에서는 CryptDestroyKey에서 키 데이터를 제거하지 않는 문제로 인해 메모리 덤프에서 Private key를 추출하는 시도를 할 수 있었다[9]. 하지만 해당 취약점은 Private key 데이터를 지우지만 않는다는 의미이지, 키 데이터 자체를 저장하고 있는 메모리 구역을 사용하지 않고 다른 용도로 사용될 수 있는 상태인 메모리 해제 상태로 만든다는 것이다. 따라서 해제된 메모리는 언제라도 프로세스의 메모리 할당 명령으로 인해 다시 사용될 수 있고, 이는 Private key가 다른 값으로 덮어씌워질 수 있다[9]. 그래서 메모리 덤프에서 Private key를 찾지 못할 수 있고, 이는 이번 실험에 사용된 방식이 항상 성공하는 것이 아닌 확률적으로 성공할 수 있다는 것을 의미한다.

이러한 확률에 영향을 미치는 변수 중 하나가 가상 PC의 CPU 코어 수와 연관이 있었다. CPU 코어 수가 하나일 경우 대부분의 메모리 덤프에서 Private key를 찾지 못하였고, 둘이나 넷 이상부터는 메모리에서 Private key를 찾을 수 있었다. 이러한 차이에 대해서는 싱글코어와 멀티코어 간의 Windows의 메모리 관리 차이점에서 오는 현상이라고 추측되고 있으나 더 자세한 부분은 추후 연구가 필요하다. 그리고 이러한 차이가 실사용 PC에서도 일어나는 현상인지도 알아볼 필요가 있다.

5.2 복호화 방식의 한계와 의의

메모리 덤프를 통한 Private key를 추출하기 위해서는 랜섬웨어가 어떤 암호화 도구를 사용하는지, Private key를 다루는 행동에서 잘못되거나 취약한 부분이 있는지 리버스 엔지니어링을 통해 알아내야 하는 선행 과정이 필요하다. 랜섬웨어는 이러한 분석을 막기 위해 리버스 엔지니어링에 필요한 디버깅 도구들을 탐지하거나, 프로그램의 코드를 분석하기 어렵게 난독화를 하는 등 여러 기법을 사용하고 있어 랜섬웨어의 암호화 모듈의 취약한 부분을 찾기 어려워지고 있다[11].

이번 실험에서 예시로 설정한 WannaCry과 같은 경우는 초창기에 만들어진 랜섬웨어였기에 앞서 말한 Wincrypt API를 사용해서 메모리에 Private key가 남는 등 데이터 보안에 신경을 쓰지 않았다. 하지만 최근에 발견되는 랜섬웨어는 보안성이 뛰어난 OpenSSL이나 자체적인 암호화 도구를 사용하며, 중요한 데이터는 값을 0이나 아무 값으로 덮어씌운 다음 메모리에서 해제한다. 따라서 이러한 방식을 사용한 랜섬웨어는 일반적인 메모리 덤프로 Private key를 추출할 가능성은 작다.

결과적으로 메모리 덤프를 통한 Private key 추출 방식이 가능한 경우는 한정적이지만 랜섬웨어에 감염된 이상 암호화된 데이터를 복호화하는 시도를 할 수 있다는 점에서 큰 의의가 있다. 데이터 복호화를 할 수 있는 다른 방법인 RSA public key 공격을 시도하는 방법이나, 랜섬웨어 공격자를 검거하는 방법 중 메모리 덤프에서 Private key를 찾는 방법은 랜섬웨어에 감염되었을 때 가장 빠르게 시도할 수 있다.

Ⅵ. 결 론

본 연구에서는 랜섬웨어에 감염된 상황에서 RSA public key 공격, 랜섬웨어 공격자를 검거하는 방법 외에 피해자가 단기간에 암호화된 파일을 복호화할 방법이 있는지에 대해 알아보았다. 랜섬웨어 공격자는 하나의 암호화 키로 여러 감염된 PC의 암호화된 파일을 복호화하는 것을 막기 위해 랜섬웨어가 감염된 다음 RSA 키를 만들게 되고 이는 복호화 키 정보가 메모리에 남게 되는 것이다. 따라서 복호화 키 정보가 메모리에 남은 것을 메모리 덤프와 분석을 통해 키를 추출하고 암호화된 파일을 복호화하리라 추측하였다. 따라서 이 추측이 맞는지에 대해 실험을 통해 알아보고자 하였고, 실험 대상으로 2017년에 유행한 WannaCry와 해당 랜섬웨어에 가장 많이 감염된 OS인 Windows 7을 선정하여 실험하였다.

실사용 PC에서의 실험은 PC 감염으로 인해 물리적 피해가 있을 수 있고, 테스트마다 같은 실험 환경 상태인지 모호하며 실험 환경 변수를 변경하기가 어렵다는 점에서 문제가 있었고, 가상 PC를 사용하기로 하였다. 가상 PC에 랜섬웨어를 감염시킨 뒤 랜섬웨어 프로세서의 메모리를 덤프하였고, Python Script와 RSA의 특징을 사용해 RSA key의 핵심 값인 P와 Q를 메모리 덤프에서 찾아내었다. P와 Q를 사용해 복호화 키를 만들어 내었고 랜섬웨어에 의해 암호화된 파일을 복호화하는 실험까지 수행하였다.

이번 실험에서는 성공적으로 암호화된 파일을 복호화했지만, 최신 랜섬웨어에서는 이러한 방법을 회피하고자 여러 방지 기법들을 사용하고 있다. 따라서 이러한 방법은 최근에는 성공 확률이 낮지만 다른 방법에 비해 단기간에 시도할 수 있으므로 의미 있는 방법이라 생각한다. 다만 메모리에 복호화 키가 남을 확률이 CPU 코어 수에 의해 조절되는 것을 발견하였는데 이는 추가적인 연구가 필요하다고 생각한다.

참 고 문 헌 (References)

[1] Crowdstrike, "History of Ransomware", Available: https://www.crowdstrike.com/cybersecurity-101/ransomware/history-of-ransomware/, 2022.01.10. confirmed

[2] Secureworks, "WCry Ransomware Analysis", Available: https://www.secureworks.com/research/wcry-ransomware-analysis, 2022.01.10. confirmed

[3] Github, "wanakiwi", Available: https://github.com/gentilkiwi/wanakiwi, 2022.01.10. confirmed

[4] Jaime C Asocsta, Adrian J Belmontes, Salamah Salamah, "Hands-on Cybersecurity Studies: Ransomware Key Recovery", Available: https://apps.dtic.mil/sti/pdfs/AD1097107.pdf, 2022.03.09. confirmed

[5] CVE, "CVE-2017-0144", Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2017-0144, 22.03.09. confirmed

[6] MSDN, "RSAPUBKEY structure (wincrypt.h)", Available: https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/ns-wincrypt-rsapubkey, 2022.01.10. confirmed

[7] MSDN, "Base Provider Key BLOBs", Available: https://docs.microsoft.com/en-us/windows/win32/seccrypto/base-provider-key-blobs, 2022.01.10. confirmed

[8] statcounter, "Operating System Market Share Worldwide", Available: https://gs.statcounter.com/os-market-share, 2022.01.10. confirmed

[9] MSDN, "CryptDestroyKey function (wincrypt.h)", Available: https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptdestroykey, 2022.01.10. confirmed

[10] A Method for Obtaining Digital Signatures and Public-Key Cryptosystems R. Rivest, A. Shamir, L. Adleman, Communications of the ACM, Vol. 21 (2), 1978, pages 120-126.

[11] Bitdefender, "A Technical Look into Maze Ransomware", Available: https://www.bitdefender.com/blog/labs/a-malware-researchers-guide-to-reversing-maze/, 2022.01.10. confirmed

저 자 소 개

김 승 환 (Seunghwan Kim)
준회원

2018년 ~ 현재 : 아주대학교 정보통신대학 사이버보안학과 학사과정
관심분야 : Reverse Engineering, IoT Security

손 태 식 (Taeshik Shon)
평생회원

2000년 : 아주대학교 정보및컴퓨터공학부 졸업(학사)
2002년 : 아주대학교 정보통신전문대학원 졸업(석사)
2005년 : 고려대학교 정보보호대학원 졸업(박사)
2004년 ~2005년 : University of Minnesota 방문연구원
2005년 ~2011년 : 삼성전자 통신· DMC 연구소 책임연구원
2017년 ~2018년 : Illinois Institute of Technology 방문교수
2011년 ~ 현재 : 아주대학교 정보통신대학 사이버보안학과 교수
관심분야 : Digital Forensics, ICS/Automotive Security