Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.28, NO.2, Apr. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.2.347

랜섬웨어 탐지율을 높이기 위한 블록암호 알고리즘 식별 방법에 관한 연구

윤 세 원,† 전 문 석‡
숭실대학교

A Study on a Method of Identifying a Block Cipher Algorithm to Increase Ransomware Detection Rate

Se-won Yoon,† Moon-seog Jun‡
Soongsil University

요 약

랜섬웨어는 블록암호와 같은 대칭키 알고리즘을 사용해서 사용자의 파일을 불법적으로 암호화한다. 만약 특정 프로그램에서 블록암호의 흔적을 사전에 발견할 수 있다면, 랜섬웨어 탐지율을 높일 수 있을 것이다. 블록암호가 포함되어 있다는 것은 잠재적으로 암호화 기능을 수행할 것이라고 볼 수 있기 때문이다. 이 논문은 특정 프로그램에 블록암호가 포함되었는지 판단하는 방법을 제시한다. 대부분의 랜섬웨어가 사용하는 AES 뿐만 아니라 향후 사용될 수 있는 다양한 블록암호의 구현 특성을 살펴보는데, 그 특성을 기반으로 특정 프로그램에 해당 알고리즘이 포함되어 있는지 알 수 있다. 이 논문에서 제시하는 방법은 기존 탐지방법을 보완해서 더욱 높은 확률로 랜섬웨어를 탐지할 수 있을 것이다.

ABSTRACT

Ransomware uses symmetric-key algorithm such as a block cipher to encrypt users' files illegally. If we find the traces of a block cipher algorithm in a certain program in advance, the ransomware will be detected in increased rate. The inclusion of a block cipher can consider the encryption function will be enabled potentially. This paper proposes a way to determine whether a particular program contains a block cipher. We have studied the implementation characteristics of various block ciphers, as well as the AES used by ransomware. Based on those characteristics, we are able to find what kind of block ciphers have been contained in a particular program. The methods proposed in this paper will be able to detect ransomware with high probability by complementing the previous detection methods.

Keywords: Ransomware, Block Cipher, Optimization

I. 서 론

랜섬웨어(ransomware)는 특정 시스템에 침투 후 문서, 이미지 등과 같이 개인의 파일을 소유주가 열람할 수 없도록 불법적으로 암호화하는 악성 프로그램이다. 랜섬웨어 배포자는 파일 소유주에게 암호화된 파일을 복호화해주는 대가로 금품을 요구한다. 2016년에는 RaaS(Ransomware as a Service) 형태의 랜섬웨어가 등장했다. 특별한 지식과 기술이 없더라도 제작자에게 비용을 지불하면 원하는 형태로 랜섬웨어를 제작해준다. 누구나 손쉽게 랜섬웨어를 악용할 수 있게 된 것이다[1]. 랜섬웨어는 계속해서

Received(12. 06. 2017), Modified(02. 05. 2018), Accepted(03. 26. 2018)
† 주저자, sewon6555@naver.com
‡ 교신저자, mjun@ssu.ac.kr(Corresponding author)

확산되었고, 2017년 3분기에 집계된 악성코드 중 약 77%가 랜섬웨어였다. 활동 영역도 점차 확대되고 있는데, PC 뿐만 아니라 모바일환경도 랜섬웨어로부터 자유롭지 않다[2].

랜섬웨어를 탐지하기 위한 여러가지 방법들이 고안되었지만 계속해서 변형된 랜섬웨어가 발견되고 있고, 다음과 같은 상황에서는 탐지를 못할 수 있다.

- 파일의 일부가 암호화되어 엔트로피 증가율이 미미하다.
- 파일의 고유한 헤더를 유지한 채 나머지 일부 데이터만을 암호화한다.
- 랜섬웨어가 미끼 파일을 우회하고 사용자의 파일만을 암호화한다.
- 미끼 파일에 접근하기 전 암호화 행위는 감지하지 못한다.

랜섬웨어는 공통적으로 블록암호와 같은 대칭키 알고리즘을 사용해서 파일을 암호화한다. 위와 같이 탐지를 못할 수 있는 상황에서, 블록암호가 구현되어 있다는 사실을 사전에 알 수 있다면, 그것이 랜섬웨어일 것이라는 확률을 높일 수 있게 된다. 블록암호가 포함되어 있다는 것은 잠재적으로 암호화 기능을 수행할 것이라고 볼 수 있다.

이 논문은 기존의 랜섬웨어 탐지방법들을 보완하고자 특정 프로그램에 블록암호가 포함되었는지 판단하는 방법을 제시한다. 랜섬웨어에서 널리 사용되고 있는 AES 뿐만 아니라 향후 사용될 수 있는 다양한 블록암호 알고리즘을 식별할 수 있다. 이 논문을 기반으로 블록암호가 포함된 프로그램을 선별하고, 기존의 탐지 방법을 추가적으로 적용한다면, 랜섬웨어 탐지율을 높일 수 있을 것이다.

2장에서는 기존 탐지방법을 소개하고 3장에서는 각 블록암호의 구현에 따른 고유한 특성을 살펴본다. 4장에서는 제안하는 방법대로 블록암호를 식별할 수 있는지 랜섬웨어 WannaCry, Mamba, Rex와 정상 암호모듈을 대상으로 실험하고, 5장에서는 제안하는 방법의 한계점과 향후 연구, 6장에서 결론을 내린다.

II. 기존 탐지방법

이번 장에서는 알려진 랜섬웨어 탐지방법에 대해서 살펴본다. 랜섬웨어들은 각각 고유한 특징이 있기 때문에 어느 하나의 탐지방법으로는 랜섬웨어를 식별하기 힘들다. 그래서 백신 제조사들은 복수의 탐지방법을 혼용한다.

2.1 표식기반 탐지

이미 발견된 랜섬웨어의 고유하고 정적인 특징을 표식(signature)으로 제작하고 비교해서 랜섬웨어를 탐지한다. 표식과 다른 신종 랜섬웨어를 탐지하지 못하는 단점이 있다.

2.2 행위기반 탐지

랜섬웨어를 여러 컴퓨터에서 실행하고 살펴본 후 공통된 행위를 찾는다. 그 공통된 행위를 하는 프로세스를 랜섬웨어로 인식하는 방법이다. 또한 특정 API를 호출하는 패턴을 기반으로 탐지한다. 예를 들어 Windows에서 제공하는 암호기능인 WinCrypt 관련 API를 호출하는 것을 관찰한다. 하지만 랜섬웨어 자체에 암호기능이 구현되어 있다면 탐지하기 힘들다. 행위기반은 정확도가 떨어진다는 단점이 있다.

2.3 파일 엔트로피 검사

파일이 암호화되면 엔트로피(entropy)가 높아지게 된다. 파일에 대한 쓰기 행위 전후의 엔트로피를 검사하고, 일정 수준 이상의 엔트로피가 발생됐을 경우 그 파일이 암호화된 것으로 간주한다[3]. 단, 파일의 일부분만 암호화됐을 경우 엔트로피의 변화량이 미미하므로 암호화 여부를 판단하는 것이 힘들다.

2.4 미끼 탐지

랜섬웨어는 저장장치를 순회하면서 특정 확장자를 가진 파일을 암호화하는 특징이 있다. 이러한 특징을 이용해서 미끼(bait) 파일을 배치하고 어떤 프로세스가 파일에 수정 행위를 가하는지 감시하는 것이 미끼기반(decoy) 탐지 방법이다[4]. 랜섬웨어가 미끼 파일에 접근하기 전까지는 탐지할 수 없고, 미끼파일을 우회할 수도 있다.

2.5 실시간 백업 및 복구

파일 수정 이벤트가 발생되면, 먼저 파일을 백업 후 이벤트를 처리한다. 이벤트 처리 완료 후 파일의

헤더 또는 포맷을 검사 후 고유한 양식이 아니라면, 파일이 암호화된 것으로 간주하고, 백업된 파일을 이용해서 복구한다. 파일의 포맷이 손상되지 않는 선에서 일부 데이터가 암호화 될 경우 혹은 포맷의 양식을 알지 못하면 암호화 여부를 판단하기 힘들다[5].

위에서 설명한 탐지방법들은 특정 상황에서 랜섬웨어를 탐지하지 못할 수 있다. 만약 랜섬웨어라고 의심할 수 있는 추가 단서를 발견한다면, 탐지율을 높일 수 있을 것이다.

III. 블록암호의 구현 특성에 따른 식별

랜섬웨어는 짧은 시간에 대량의 파일을 암호화해야 하므로 블록암호와 같은 대칭키 알고리즘을 사용한다. 각 블록암호는 고유한 특성을 가지고 있고, 프로그래밍 언어로 구현하는 방법에 따라서 그 고유한 특성이 더욱 구체화된다. 이렇게 구체화된 특성은 프로그램 바이너리에 흔적으로 남게 된다. 이 흔적을 미리 알 수 있다면, 프로그램을 메모리에 실행하지 않더라도 그 흔적을 찾음으로써 해당 블록암호가 포함되어 있다는 것을 판단할 수 있다. 블록암호가 포함되어 있다는 것은 잠재적으로 암호화 기능을 수행할 것이라고 볼 수 있기 때문에 기존 탐지방법을 보완할 수 있고, 결국 탐지율을 높일 수 있을 것이다. 그런데 같은 AES라도 구현하는 방법에 따라서 그 흔적이 상이하게 나타난다. 이번 장에서는 다양한 구현 방법에 따라 나타나는 흔적을 살펴본다. 대부분의 랜섬웨어는 AES를 사용하고 있으나, 향후 다른 블록암호를 사용할 수 있다. 짧은 시간에 더욱 많은 파일들을 암호화하는 것이 랜섬웨어의 목적이라면, AES보다는 HIGHT나 LEA와 같은 경량 블록암호를 채택하는 것이 더 적합할 수 있다. 이러한 이유로 AES 뿐만 아니라 구조가 전혀 다르고 암호화 속도가 더욱 빠른 LEA, HIHGT의 구현 특성에 따른 흔적도 살펴본다.

3.1 AES의 구현 특성

AES(Advanced Encryption Standard)는 국제 표준 블록암호로서 정보보호시스템 뿐만 아니라 대부분의 랜섬웨어에서 사용되고 있다. AES의 구현 방법은 일반적인 구현형태와 최적화된 구현형태로 나눌 수 있다. 각각의 구현형태를 이해하면, 구현으로부터 생산된 특정 데이터를 알 수 있다. 어떤 실행 파일을 메모리에 실행하지 않더라도 바이너리에서 특정 데이터를 찾는 것을 통해서, AES가 구현된 형태로 포함되어 있는 지 알 수 있다.

AES는 16바이트의 평문블록을 스테이트(state)라는 논리적 단위로 변환한 후 라운드함수의 입력으로 반복 적용한다. 라운드함수의 반복횟수는 비밀키의 크기에 따라 10~14회 적용된다. 라운드 함수는 4가지 요소로 구성되어 있으며, 진행 순서대로 SubBytes, ShiftRows, MixColumns, AddRoundKey와 같다. SubBytes는 스테이트의 각 원소에 대해서 기약 다항식 $m(x)=x^8+x^4+x^3+x+1$을 가지는 $GF(2^8)$에서 곱셈에 대한 역원 x^{-1}을 구한다. 그리고 x^{-1}에 대해서 아핀(affine) 연산을 한다. 아핀변환의 곱셈은 상수 정방행렬과 수행되고, 덧셈은 상수 열벡터와 수행된다[6].

비밀키의 크기에 따라 라운드 횟수가 10~14회이고, 스테이트의 원소는 16개 이므로, 위에서 설명한 SubBytes의 총 연산횟수는 160~224회가 된다. 이것은 암호화 처리속도 측면에서 매우 비효율적이다. 스테이트의 각 원소는 8비트이므로, 256개의 값을 도출할 수 있다. 이렇게 계산된 값을 테이블의 형태로 저장하며, 이것을 S-box라 한다. 8비트 입력 x에 대해서 위의 SubBytes의 연산을 수행한 결과인 S-box는 다음과 같다.

$$\{f(0), f(1), ..., f(255)\} = \{ox63, ox7c, ..., ox16\}$$

위의 SubBytes의 연산은 S-box를 참조하는 것으로 대체할 수 있다. 구현된 거의 모든 AES 소스코드는 현실적인 처리속도를 위해서 S-box를 가지고 있다. 따라서 AES가 구현된 바이너리는 S-box의 흔적을 가지고 있다. 그 흔적을 찾는 것으로, 바이너리에 AES가 포함되어 있다는 사실을 알 수 있다.

그런데 AES의 최적화 구현기술에 따라서 S-box와는 전혀 다른 값을 가진 테이블이 나타날 수 있다. 위에서 설명한 S-box보다 더욱 더 빠르게 라운드 함수를 처리하도록 한 새로운 테이블의 결과이다. 이 결과는 앞에서 설명한 일반적인 구현 형태에서의 S-box와는 완전히 다른 형태이다.

최적화 구현에서는 SubBytes에 해당되는 S-box 뿐만 아니라 MixColumns까지 미리 계산된

결과를 테이블의 형태로 저장한다. ShiftRows는 MixColumns 연산 후에 스테이트의 각 열에 대해서 적절한 이동(shift)연산을 통해 계산해도 무방하다. MixColumns는 (1)과 같이 스테이트의 각 열과 상수 정방행렬 X를 $GF(2^8)$에서 곱셈을 수행한다.

$$\begin{bmatrix} S_{1,i} \\ S_{2,i} \\ S_{3,i} \\ S_{4,i} \end{bmatrix} X = \begin{bmatrix} S_{1,i} \\ S_{2,i} \\ S_{3,i} \\ S_{4,i} \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} = S_{1,i} \begin{bmatrix} 2 \\ 1 \\ 1 \\ 3 \end{bmatrix} + S_{2,i} \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix} + S_{3,i} \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} + S_{4,i} \begin{bmatrix} 1 \\ 1 \\ 3 \\ 2 \end{bmatrix} \quad (1)$$

(1)의 우변처럼 스테이트의 각 성분, 즉, S-box의 각 원소는 상수 정방행렬 X의 각 열과 $GF(2^8)$에서 스칼라 곱을 한 것으로 나타낼 수 있다. S-box의 첫 번째 원소에 대해서 (1)의 우변의 첫 번째 항처럼 계산된 결과는 다음과 같다.

$$0x63 \times 2 \| 0x63 \times 1 \| 0x63 \times 1 \| 0x63 \times 3 = 0xc66363a5$$

마찬가지로 S-box의 마지막 원소에 대해서 (1)의 우변의 첫 번째 항처럼 계산된 결과는 다음과 같다.

$$0x16 \times 2 \| 0x16 \times 1 \| 0x16 \times 1 \| 0x16 \times 3 = 0x2c16163a$$

(1)의 식의 우변의 항은 4개 이므로, 구현에 따라 각 항에 대한 유일한 테이블이 각각 존재할 수 도 있고, 첫 번째 항에서 아랫방향으로 순환 이동한 결과를 사용할 수도 있다. 다르게 표현하면 첫 번째 항에 대한 결과는 반드시 저장하고 있어야 한다. 최적화 구현 형태에서는 상수 정방행렬과 곱한 결과를 찾음으로써 AES가 포함되어 있다는 사실을 알 수 있다.

3.2 LEA의 구현 특성

LEA(Lightweight Encryption Algorithm)는 S-box가 없는 128비트 경량 블록암호이다. 비밀키의 크기에 따라 라운드 함수의 반복 횟수는 24~32회 이다. 라운드 함수는 ARX(Addtion, Rotation, XOR) 연산으로 구성되어 있다. LEA의 라운드 함수는 S-box를 배제한 ARX연산으로 구성되어 있어 데이터를 고속으로 암호화한다[7]. 랜섬웨어는 짧은 시간에 대량의 파일을 암호화해야 하기 때문에 AES 보다 LEA와 같은 경량 블록암호가 적합할 수 있다.

LEA는 S-box가 없기 때문에 앞에서 설명한 AES를 식별하는 방법을 사용할 수 없다. 대신 라운드 키를 생성하는 과정에서 사용되는 상수 $\delta[i]$ $(0 \le i \le 7)$를 찾는 것으로 LEA를 식별할 수 있다.

δ[i]는 $\sqrt{766965}$에서 유도된 값이고, 76, 69, 65 는 LEA 각 문자의 아스키 코드 값에 해당된다[8]. 라운드 키 생성 과정에서 사용되는 8개의 δ는 Table 1과 같다. δ는 구현방법과 관련 없이 반드시 사용하므로, 이 값을 찾음으로써 LEA를 식별할 수 있다.

그런데 LEA의 최적화 구현에서는 위의 δ외에 추가적인 값이 나타날 수 있다. 비밀 키의 크기에 따른 24~32개의 라운드 키의 생성에서 δ의 왼쪽 순환이동한 결과를 사용한다. 따라서 정해진 순환이동 비트양에 따라 δ를 미리 순환 이동해서 테이블의 형태로 저장 및 참조한다면, 라운드 키를 고속으로 생성할 수 있다. Table 2.처럼 각 δ는 왼쪽으로 1비트씩 총 k번 만큼 순환 이동한다.

최적화 구현에서는 각 δ에 대한 순환이동 결과를 테이블의 형태로 저장하고 있으므로, 이 값을 찾음으로써, LEA알고리즘을 식별할 수 있다.

Table 1. Constants used in key schedule

each δ	Value	each δ	Value
δ[0]	c3efe9db	δ[4]	715ea49e
δ[1]	44626b02	δ[5]	c785da0a
δ[2]	79e27c8a	δ[6]	e04ef22a
δ[3]	78df30ec	δ[7]	e5c40957

Table 2. Circular shift operation

each δ	k times shift	each δ	k times shift
δ[0]	29	δ[4]	32
δ[1]	30	δ[5]	32
δ[2]	32	δ[6]	32
δ[3]	32	δ[7]	32

3.3 HIGHT의 구현 특성

HIGHT(HIGh security and light weigHT)는 법 2^8에서의 덧셈과 뺄셈, XOR, 순환이동의 연산으로 구성된 Feistel 구조의 변형으로서, 128비트 비밀 키를 가지는 64비트 블록암호이다[9]. LEA와 마찬가지로 S-box가 없는 경량 블록암호이다. 랜섬웨어는 짧은 시간 내에 대량의 파일을 암호화해야하기 때문에 AES 보다 HIGHT와 같은 경량 블록암호가 적합할 수 있다.

HIGHT 역시 LEA처럼 S-box가 없기 때문에 앞에서 설명한 AES를 식별하는 방법을 사용할 수 없다. 또한 LEA처럼 그 자체를 식별할 수 있는 충분한 양의 상수 값도 없다. 유일한 상수는 라운드 키 생성과정에서 사용되는 LFSR(Linear Feedback Shift Register)의 초기 7비트 내부 상태 값 0x5a이고, 이것으로 HIGHT를 식별하기에는 정보가 충분하지 않다.

라운드 키 생성과정에서 사용되는 LFSR의 연결 다항식은 x^7+x^3+1이고, 2^7-1의 주기를 갖는다. 총 127번의 LFSR을 실시하며, 라운드 키 생성과정에서 그 내부 상태 값 $\delta_i(0 \le i \le 127)$를 사용한다. LFSR의 동작은 Fig 1.과 같다. 라운드 키를 생성하는데 127번의 LFSR 연산을 실시하는 것은 비효율적이다. 127번의 LFSR연산은 평문, 암호문 그리고 비밀 키에 의존하지 않는 독립적인 연산이다. 이것은 값을 미리 계산해서 테이블의 형태로 저장할 수 있다는 것을 의미한다. 미리 계산된 δ_i의 값은 다음과 같다.

$$\{\delta_0, \delta_1, \delta_2, ..., \delta_{126}, \delta_{127}\}$$
$$= \{0x5a, 0x6d, 0x36, ..., 0x35, 0x5a\}$$

주기가 127이기 때문에 $\delta_0 = \delta_{127}$이다. 미리 계산된 LFSR의 내부 상태들을 참조하면 라운드 키를 고속으로 생성할 수 있다. 이렇게 미리 계산된 LFSR의 테이블 값을 찾는 것으로 HIGHT를 식별할 수 있다.

또한 라운드 함수 내부에서 F_0과 F_1이라는 순환이동 함수를 각각 2번씩 사용하는데, 이 함수의 입출력은 각각 8비트이다. F_0은 입력 값에 대해서 각각 1, 2, 7비트 좌측 순환 이동한 결과를 XOR한다. F_1은 입력 값에 대해서 각각 3, 4, 6비트 좌측

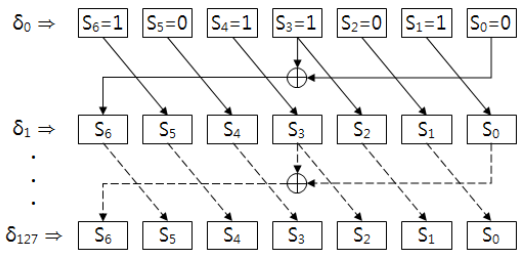


Fig. 1. LFSR operation

순환 이동한 결과를 XOR한다. 라운드 함수 반복횟수가 32회이고, 하나의 평문 블록을 암호화하는데, F_0과 F_1은 각각 64회 실시된다. 이것은 처리속도 측면에서 매우 비효율적이다. 최적화 구현에서는 F_0과 F_1의 모든 입력 값에 대해서 미리 계산 후 그 결과를 테이블에 저장한다. 입출력 값이 8비트 이므로, 8비트 요소 256개를 가진 2개의 테이블이 다음과 같이 존재하게 된다.

$$\{F_0(0), F_0(1), ..., F_0(254), F_0(255)\}$$
$$= \{0x00, 0x86, ..., 0x79, 0xff\}$$

$$\{F_1(0), F_1(1), ..., F_1(254), F_1(255)\}$$
$$= \{0x00, 0x58, ..., 0xa7, 0xff\}$$

이렇게 미리 계산된 F_0과 F_1의 테이블의 값을 찾는 것으로 HIGHT를 식별할 수 있다.

IV. 실 험

이번 장에서는 바이너리의 내용에서 블록암호의 포함 여부를 확인할 수 있는 지, 랜섬웨어를 대상으로 실험했다. 또한 정상적인 암호모듈도 블록암호를 사용하기 때문에 실험대상에 포함했다. 랜섬웨어의 경우 WannaCry, Mamba, Rex를 대상으로 실험했다. 암호모듈의 경우, 국내에서 인터넷뱅킹과 같은 여러 금융서비스를 이용할 때 설치되는 보안 관련 프로그램 중에서, 이 논문에서 제안하는 방법으로 LEA와 HIGHT가 포함된 암호모듈 A와 B를 찾을 수 있었다.

4.1 랜섬웨어 WannaCry

WannaCry는 2017년 5월 전 세계적으로 많은 피해를 발생시킨 랜섬웨어로서 파일을 불법적으로 암

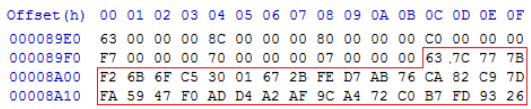
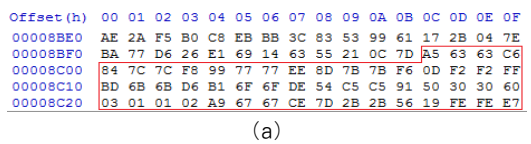


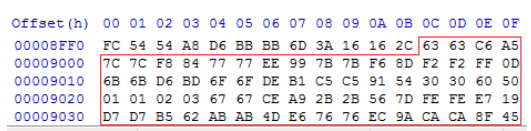
Fig. 2. AES's S-box in WannaCry

호화하는 것뿐만 아니라, Windows의 SMB (Server Message Block) 프로토콜 취약점을 악용해 네트워크로 전파되는 것이 특징이다[10].

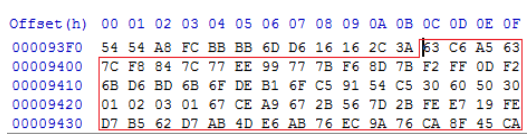
WannaCry에서는 AES가 포함되어 있음을 확인할 수 있었다. Fig 2.처럼 일반적인 형태의 S-box가 존재했다. 뿐만 아니라 MixColumns에서 사용되는 상수 정방행렬 X의 각 열을 스칼라 곱한 테이블 4개를 추가적으로 발견할 수 있었다. Fig 3.의 (a)는 S-box의 각 원소와 X의 첫 번째 열 {2, 1, 1, 3}을 $GF(2^8)$에서 스칼라 곱한 테이블의 일부이다. S-box의 첫 번째 원소 0x63와 열 {2, 1, 1, 3}을 곱한 결과는 0xc66363a5이다. 이 값이 리틀 엔디안(little-endian) 시스템에서는 (a)처럼 순서대로 0xa5, 0x63, 0x63, 0xc6와 같이 나타난다. (b), (c), (d)는 각각 두 번째 열 {3, 2, 1, 1}, 세 번째 열 {1, 3, 2, 1}, 마지막 열 {1, 1, 3, 2}을 스칼라 곱한 테이블의 일부이다. 이처럼 WannaCry에서는 S-box뿐만 아니라 MixColumns에서 사용되는 상수 정방행렬 X의 각 열을 곱한 테이블도 발견할 수 있었다. 따라서 WannaCry는 최적화된 AES 알고리즘을 포함하고 있다는 것을 보여준다.



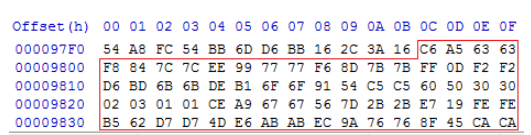
(a)



(b)



(c)



(d)

Fig. 3. AES's Multiplication by (a) Column {2, 1, 1, 3}, (b) Column {3, 2, 1, 1}, (c) Column {1, 3, 2, 1}, (d) Column {1, 1, 3, 2} in WannaCry

4.2 랜섬웨어 Rex

Rex는 2016년 8월에 발견됐으며, 리눅스 환경에서 동작한다. Rex는 Windows 외에 다른 운영체제도 랜섬웨어로부터 자유롭지 않다는 것을 보여준다[11]. Rex에서도 WannaCry와 동일하게 AES의 S-box, 상수 정방행렬 X의 각 열을 곱한 4개의 테이블을 발견할 수 있었다. 따라서 Rex는 최적화된 AES 알고리즘을 포함하고 있다.

4.3 랜섬웨어 Mamba

Mamba는 2016년 11월 미국 샌프란시스코의 열차시스템을 감염시킨 랜섬웨어이다. 사회기반시설이 랜섬웨어로부터 공격을 받을 수 있다는 것을 보여준 대표적인 사례이다[12].

Mamba에서도 AES의 흔적을 찾을 수 있었다. 그러나 Mamba의 테이블 형태는 WannaCry, Rex의 것과 다르다는 것을 발견할 수 있었다. WannaCry와 Rex에서 S-box의 성분은 각각 열 {2, 1, 1, 3}, 열 {3, 2, 1, 1}, 열 {1, 3, 2, 1}, 열 {1, 1, 3, 2}과 스칼라 곱한 결과였다. 그러나 Mamba에서는 각 열의 성분이 대칭 이동한 형태였다. 즉, S-box의 각 성분이 각각 열 {3, 1, 1, 2}, 열 {1, 1, 2, 3}, 열 {1, 2, 3, 1}, 열 {2, 3, 1, 1}과 곱한 형태이다. Fig 4.는 대칭 이동된 첫 번째 열 {3, 1, 1, 2}와 스칼라 곱한 테이블의 모습의 일부이다. 평문을 스테이트로 변환할 때는 평문의 각 바이트를 스테이트의 열 순서대로 배치해야 한다. 예를 들어, 평문블록의 첫 4바이트 B_1, B_2, B_3, B_4는

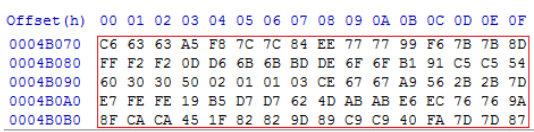


Fig. 4. AES's Multiplication by Column {3, 1, 1, 2} in Mamba

스테이트의 첫 번째 열에 {B_1, B_2, B_3, B_4}와 같이 배치가 된다. 그러나 Mamba에서는 스테이트의 첫 번째 열에 {B_4, B_3, B_2, B_1}처럼 배치가 된다. 이 순서에 맞게 MixColumns에서 사용되는 상수 정방행렬 X의 각 열의 성분도 거꾸로 배치되어야 한다. 이것은 리틀엔디안 시스템의 특성을 고려해서 평문과 스테이트간의 변환 과정을 더욱 효율적으로 구현한 것이다. 따라서 Mamba의 AES는 WannaCry와 Rex의 AES보다 더욱 진보된 형태라 할 수 있다.

4.4 정상 암호모듈 A

이 논문에서 제안하는 방법을 사용한 결과, 인터넷뱅킹과 같은 금융서비스 이용 시 설치되는 보안관련 프로그램 중에서 LEA가 구현된 암호모듈 A를 찾을 수 있었다. Fig 5.처럼 $\delta[0]$의 값에 대한 순환이동 결과를 테이블로 가지고 있었다. 뿐만 아니라 나머지 7개의 δ에 대한 순환이동 결과의 테이블도 각각 찾을 수 있었다. 이 암호모듈에서는 LEA 뿐만 아니라 AES도 포함되어 있었다.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0003C180	00	01	02	03	DB	E9	EF	C3	B7	D3	DF	87	6F	A7	BF	0F
0003C190	DE	4E	7F	1F	BC	9D	FE	3E	78	3B	FD	7D	F0	76	FA	FB
0003C1A0	E1	ED	F4	F7	C3	DB	E9	EF	87	B7	D3	DF	0F	6F	A7	BF
0003C1B0	1F	DE	4E	7F	3E	BC	9D	FE	7D	78	3B	FD	FB	F0	76	FA
0003C1C0	F7	E1	ED	F4	EF	C3	DB	E9	DF	87	B7	D3	BF	0F	6F	A7

Fig. 5. LEA's the result of shift operation

4.5 정상 암호모듈 B

또 다른 보안관련 프로그램의 파일들로부터 HIGHT가 구현된 암호모듈 B를 찾을 수 있었다. HIGHT에서 사용되는 LFSR의 모든 내부 상태 값이 Fig 6.처럼 존재했다. LFSR의 모든 내부 상태 값이 존재한다는 것은 라운드키를 고속으로 생성할 수 있다는 것을 의미한다. 그리고 보다 빠르게 암호화하는데 도움을 주는 F_0의 모든 결과 값, F_1의 모든 결과 값을 각각 Fig 7., Fig 8.처럼 찾을 수 있었다.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000014E0	18	A2	40	00	00	00	00	00	5A	6D	36	1B	0D	06	03	41
000014F0	60	30	18	4C	66	33	59	2C	56	2B	15	4A	65	72	39	1C
00001500	4E	67	73	79	3C	5E	6F	37	5B	2D	16	0B	05	42	21	50
00001510	28	54	2A	55	6A	75	7A	7D	3E	5F	2F	17	4B	25	52	29

Fig. 6. HIGHT's internal states of LFSR

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00001560	74	3A	5D	2E	57	6B	35	5A	00	86	0D	8B	1A	9C	17	91
00001570	34	B2	39	BF	2E	A8	23	A5	68	EE	65	E3	72	F4	7F	F9
00001580	5C	DA	51	D7	46	C0	4B	CD	D0	56	DD	5B	CA	4C	C7	41
00001590	E4	62	E9	6F	FE	78	F3	75	B8	3E	B5	33	A2	24	AF	29

Fig. 7. HIGHT's F0

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00001660	6E	E8	63	E5	74	F2	79	FF	00	58	B0	E8	61	39	D1	89
00001670	C2	9A	72	2A	A3	FB	13	4B	85	DD	35	6D	E4	BC	54	0C
00001680	47	1F	F7	AF	26	7E	96	CE	0B	53	BB	E3	6A	32	DA	82
00001690	C9	91	79	21	A8	F0	18	40	8E	D6	3E	66	EF	B7	5F	07

Fig. 8. HIGHT's F1

AES, LEA, HIGHT를 식별하는 방법에 대해서 알아보았지만 같은 방법으로 다양한 암호알고리즘을 식별할 수 있을 것이다. 실험대상은 바이너리가 원문 형태였기 때문에 알고리즘을 식별할 수 있었다. 만약 바이너리의 코드가 암호화되어 있다면, 알고리즘을 식별할 수 없게 된다.

V. 한계점 및 향후 연구

이 논문의 목적은 기존 탐지방법들이 암호화여부를 판단할 수 없음으로써 부정오류(false negative)를 발생할 수 있는 상황에서 '바이너리가 암호알고리즘을 탑재하고 있음'과 같은 추가적인 단서를 제공하는 것이다.

실험에서 살펴본 것처럼 제안하는 방법은 랜섬웨어 뿐만 아니라 정상적인 암호모듈에 구현된 암호알고리즘을 식별하는데 적용할 수 있다. 이것은 긍정오류(false positive)가 발생된 다는 것을 의미한다. 따라서 제안하는 방법만으로는 랜섬웨어를 탐지할 수 없고, 기존 탐지방법들과 결합이 필요하다.

랜섬웨어의 가장 치명적인 특징은 파일을 암호화하는 것인데, 2장에서 살펴본 것처럼 기존의 탐지방법들은 특정 상황에서 암호화 여부를 판단하는 것이 모호할 수 있다.

제안하는 방법이 기존 탐지방법들과 결합한다면, 다음과 같은 조건에서 부정오류 확률을 줄일 수 있다.

- 어떤 프로세스가 저장장치를 순회한다.
- 짧은 시간동안 다양한 확장자를 가진 다수의 파일에 쓰기 행위를 한다.
- 그러나 파일 쓰기 전/후의 엔트로피 변화율이 미미해서 암호화 행위인지 결정할 수 없다.

- 또한 파일의 특정 포맷을 유지한 채 일부분만 암호화해서 정상적인 수정행위로 간주된다. 혹은 텍스트 파일과 같이 포맷이 없는 파일의 일부분만 암호화한다.
- 그 프로세스에는 특정 암호알고리즘이 구현된 상태로 포함되어 있다는 사실을 알 수 있다.

만약 위와 같은 조건을 모두 만족하는 프로세스가 긍정오류로 간주된다면, 그 정상 프로세스는 매우 특별하다. 그러한 특별한 프로세스는 화이트리스트 기반으로 관리되어야 할 것으로 생각된다.

이 논문의 실험에서는 랜섬웨어의 코드영역이 원문형태 였기 때문에 암호알고리즘을 식별할 수 있었다. 만약 코드영역이 암호화되어 있다면, 알고리즘을 식별하는 데 어려움이 발생한다. CPU에 의해서 코드가 실행되기 위해서는 암호화된 코드가 반드시 복호화되어야 한다. 따라서 코드영역에서 변화(복호화)가 발생 되었다고 인지할 수 있는 근거 그리고 변화 시점에 대한 추가적인 연구가 필요하다.

VI. 결 론

암호알고리즘을 구현하게 되면 알고리즘의 특징이 바이너리에 흔적으로 남게 된다. 같은 알고리즘이라도 구현하는 방법에 따라서 흔적이 상이하게 나타나게 되는데, 알고리즘의 설계를 준수해야 하므로 그 방법이 한정되어 있다. 앞서 실험에서 살펴본 것처럼 바이너리의 내용에서 특정 정보를 찾음으로써, 어떤 블록암호 알고리즘이 바이너리에 포함되어 있는지 알 수 있다.

랜섬웨어는 공통적으로 블록암호 알고리즘을 사용해서 파일을 암호화한다. 파일의 암호화 여부가 모호할 경우 랜섬웨어를 탐지 못할 수 있다. 그러한 특정 상황에서 블록암호가 구현되어 있다는 사실을 미리 알 수 있다면, 그것이 랜섬웨어일 것이라는 조건이 추가된다.

그러나 바이너리의 실행코드가 암호화되어 있을 경우에는 알고리즘을 식별할 수 없으므로, 향후에는 메모리에서 복호화 되는 시점과 그 영역에서 알고리즘의 흔적을 찾는 연구가 추가적으로 필요하다.

References

[1] "Trends of ransomware in 2016 and its outlook for next year," Korea Internet & Security Agency, Jan. 2017.

[2] "Trends report on cyber threat in the third quarter 2017," Korea Internet & Security Agency, Oct. 2017.

[3] Amin Kharaz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda, "Unveil: a large-scale, automated approach to detecting ransomware," Proceedings of the 25th USENIX Security Symposium, pp. 757-772, Aug. 2016.

[4] Brian M. Bowen, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo, "Baiting inside attackers using decoy documents," ADA500672, Department of Computer Science, Columbia University, Sep. 2008.

[5] Jae-Yeol Kim, "A study on the recovery of ransomware infected file through real-time file behavior analysis," Master's Thesis, Korea University, May. 2017.

[6] "Advanced encryption standard (AES)," Federal Information Processing Standards Publication 197, Nov. 2001.

[7] D. Hong, D. Kim, and D. Kwon, " 128-Bit block cipher LEA," TTAK.KO-12.0223, Dec. 2013.

[8] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee, "LEA: a 128-bit block cipher for fast encryption on common processors," Proceedings of the WISA 2013, pp. 3-27, Aug. 2013.

[9] S. Lee, Y. Yeom, H. Park, and H. Kim, "64-Bit block cipher HIGHT," TTAK.KO-12.0040/R1, Dec. 2008.

[10] "Wannacry report," https://www.pandasecurity.com/mediacenter/src/uploads/2017/05/WannaCry_Report-en.pdf

[11] "Linux security: a closer look at the late

st linux threats," https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/linux-security-a-closer-look-at-the-latest-linux-threats

[12] "Mamba ransomware allows riders free entry to San Francisco Muni," https://usa.kaspersky.com/blog/mamba-hddcryptor-ransomware/10519/

〈저 자 소 개〉

윤 세 원 (Se-won Yoon) 정회원
2011년 9월~현재: 숭실대학교 컴퓨터학과 박사과정
〈관심분야〉 정보보호, 네트워크 보안, 암호학

전 문 석 (Moon-Seog Jun) 종신회원
1981년 2월: 숭실대학교 전산과 졸업
1986년 2월: University of Maryland Computer Science 석사
1989년 2월: University of Maryland Computer Science 박사
1986년 9월~1989년 12월: University of Mary 강사
1989년 3월~7월: Morgan State University 조교수
1989년 9월~1991년 2월: NMSU, PSL 연구소 책임연구원
1991년 3월~현재: 숭실대학교 정교수
〈관심분야〉 정보보호, 네트워크 보안, 전자여권, 암호학