석사학위논문

저전력 임베디드 프로세서를 위한 인공신경망 기반의 암호화 랜섬웨어 탐지

2022년

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

김 현 지

석 사 학 위 논 문
지도교수 서화정

저전력 임베디드 프로세서를 위한 인공신경망 기반의 암호화 랜섬웨어 탐지

Neural Network-based Cryptography Ransomware Detection for Low-End Embedded Processors

2021년 12월 일

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

김 현 지

석 사 학 위 논 문
지도교수 서화정

저전력 임베디드 프로세서를 위한 인공신경망 기반의 암호화 랜섬웨어 탐지

Neural Network-based Cryptography Ransomware Detection for Low-End Embedded Processors

위 논문을 공학 석사학위 논문으로 제출함

2021년 12월 일

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

김 현 지

김현지의 공학 석사학위 논문을 인준함

2021년 12월 일

심사위원장 최 원석 (인)

심 사 위 원 이 웅희 (인)

심 사 위 원 서 화정 (인)

국 문 초 록

저전력 임베디드 프로세서를 위한 인공신경망 기반의 암호화 랜섬웨어 탐지

한 성 대 학 교 대 학 원
I T 융 합 공 학 과
I T 융 합 공 학 전 공
김 현 지

암호화 랜섬웨어에는 피해자의 파일을 암호화하는 프로세스가 있으며, 암호화 랜섬웨어는 피해자에게 암호화된 파일을 복호화 하기 위한 키에 대한 금전을 요구한다. 본 논문에서는 사물 인터넷(IoT) 플랫폼을 위한 블록 암호 알고리즘을 탐지하여 암호화 랜섬웨어를 방지하는 새로운 접근 방식 두 가지를 제시한다. AVR 패키지의 일반 소프트웨어와 경량 블록 암호 라이브러리(FELICS)에서 C 언어로 작성된 블록 암호 구현은 구성된 신경망을 통해 학습 및 평가되었다. 암호 알고리즘 및 정상 소프트웨어 샘플을 컴파일 하여 얻은 바이너리 파일의 opcode에서 시퀀스 및 빈도수 특성을 추출한다. 후기 융합 방식을 통해 하나의 소스 데이터에서 위와 같은 두 개의 특징을 추출하고 각 네트워크를 통해 학습한 후 융합한다. 제안 방법을 통해 암호화 랜섬웨어 바이러스 또는 정상 소프트웨어를 성공적으로 분류할 수 있다. 또한 블록 암호를 사용하여 파일을 암호화하는 바이너리 코드를 학습된 네트워크에 입력하면 암호화 랜섬웨어를 탐지해낼 수 있다. 또 다른 제안 기법은 시퀀스와 빈

도수 특성을 추출하지 않고, opcode와 opcode sequence를 단어와 문장으로 생각하여 단어 임베딩을 수행한 뒤, 트랜스포머의 인코더 구조 기반의 신경망에 입력한다. 해당 기법을 통해 비슷한 수준의 분류 성능을 유지하면서 파일 크기를 0,5배로 감소시켰다. 두 제안 기법에 대한 탐지 성공률은 정밀도와 재현율의 조화 평균인 F-측정값으로 평가되었다. 각각 암호화 랜섬웨어 탐지 성공률 97%, 98%를 달성했을 뿐만 아니라, 정상 펌웨어와 경량 암호 알고리즘별 분류와 SPN(Substitution-Permutation-Network) 구조, ARX(Addition-Rotation-eXclusive-or 구조) 및 정상 펌웨어에 대한 분류 또한 가능하다.

【주요어】 딥러닝, 암호, 랜섬웨어, 사물인터넷

목 차

제 1 장 서 론 ····· 1

제 1 절 랜섬웨어 탐지 동향

제 2 장 관련 연구 ····· 4

제 1 절 IoT 환경에서의 랜섬웨어 ····· 4
제 2 절 암호화 함수 호출 기반의 랜섬웨어 탐지 기법 동향 ····· 4
제 3 절 Fair Evaluation of Lightweight Cryptographic Systems (FELICS) ····· 5

제 3 장 제안 기법 ····· 7

제 1 절 컨볼루션 퓨전 신경망 기반의 암호화 랜섬웨어 탐지 ····· 7
1) 데이터 생성 및 전처리 ····· 8
2) 신경망 구조 ····· 10
3) 학습 및 탐지 ····· 11
4) Pruning을 통한 네트워크 경량화 ····· 14
제 2 절 트랜스포머의 인코더 기반의 암호화 랜섬웨어 탐지 ····· 15
1) 데이터 생성 및 전처리 ····· 16
2) 신경망 구조 및 학습 ····· 17
3) 암호화 랜섬웨어 탐지 ····· 21

제 4 장 실험 및 평가 ····· 22

제 1 절 실험 환경 ····· 22
제 2 절 데이터 ····· 22
제 3 절 컨볼루션 퓨전 신경망 기반의 암호화 랜섬웨어 탐지 ····· 23
1) Opcode 기반 vs Instruction 기반에 대한 성능 평가 ····· 24
2) Sequence-Only vs Frequency and Opcode Sequence에

대한 성능 평가 ····· 29
3) 경량화 된 네트워크에 대한 성능 평가 ····· 30
제 4 절 트랜스포머의 인코더 기반의 암호화 랜섬웨어 탐지 ····· 31

제 5 절 제안 기법 간의 비교 ····· 34
제 6 절 다른 기법과의 비교 ····· 35

제 5 장 결론 및 향후 연구 방안 ····· 37

참 고 문 헌 ····· 39

ABSTRACT ····· 42

표 목 차

[표 2-1] 암호화 함수 호출 기반의 랜섬웨어 탐지 기법 비교 ····· 5
[표 3-1] 컨볼루션 퓨전 네트워크의 하이퍼파라미터 ····· 13
[표 3-2] 트랜스포머 기반 네트워크의 하이퍼파라미터 ····· 20
[표 4-1] 데이터셋 세부사항 ····· 23
[표 4-2] Opcode 기반 학습에 대한 F-measure ····· 25
[표 4-3] Instruction(operand+opcode)기반 학습에 대한 F-measure ····· 25
[표 4-4] 각 구조에 따른 명령어 빈도수 비교 ····· 29
[표 4-5] Sequence-only 모델에 대한 F-measure ····· 30
[표 4-6] Pruning 기법을 적용한 네트워크에 대한 F-measure ····· 31
[표 4-7] Pruning 적용 전과 후의 파일 크기 및 F-measure ····· 31
[표 4-8] 임베딩 차원이 128일 때, Conv1D 레이어의 유닛 수 및 kernel size에 따른 validation F-measure와 파라미터 수 ···· 33
[표 4-9] 임베딩 차원이 8일 때, Conv1D 레이어의 유닛 수 및 kernel size에 따른 validation F-measure와 파라미터 수 ·· 33
[표 4-10] 트랜스포머 인코더 기반의 네트워크에 대한 F-measure ·· 34
[표 4-11] 두 제안기법에 대한 파일 크기 및 탐지 단계의 F-measure ····· 35
[표 4-12] 다른 기법과의 비교 ····· 36

그 림 목 차

[그림 3-1] 컨볼루션 퓨전 신경망 기반의 랜섬웨어 탐지 기법의 구조도 8
[그림 3-2] 각 함수에서 opcode sequence 추출 ····· 10
[그림 3-3] 데이터 전처리 ····· 10
[그림 3-4] 컨볼루션 퓨전 네트워크 구조도 ····· 11
[그림 3-5] Conv1D의 동작 과정 ····· 13
[그림 3-6] 암호화 랜섬웨어 탐지 과정 ····· 14
[그림 3-7] 트랜스포머의 인코더 기반의 암호화 랜섬웨어 탐지 시스템 구조도 ····· 15
[그림 3-8] 입력 데이터 전처리를 위한 opcode 임베딩 과정 ····· 17
[그림 3-9] 트랜스포머 인코더 기반의 신경망 구성도 ····· 19
[그림 3-10] 트랜스포머 인코더 기반의 암호화 랜섬웨어 탐지 단계의 구성도 ····· 21
[그림 4-1] 각 알고리즘의 핵심 함수에 대한 opcode sequence에 대한 시각화 ····· 28
[그림 4-2] 각 알고리즘에 사용된 opcode의 빈도수에 대한 시각화 ··· 28

제 1 장 서론

제 1 절 랜섬웨어와 랜섬웨어 탐지 기법 동향

2017년 다수의 Microsoft Windows 사용자들이 파일 공유 프로토콜에 존재하는 취약점으로 인해 Wannacry 랜섬웨어 바이러스에 감염되었다[1]. 200,000대 이상의 컴퓨터가 피해를 입었으며, 이는 랜섬웨어 역사상 가장 큰 공격이었다.

랜섬웨어는 크게 두 가지 종류가 있으며, 파일을 암호화하는 암호화 랜섬웨어와 파일을 잠그는 locker 랜섬웨어로 나뉜다. locker 유형은 대상 시스템을 잠그므로 사용자가 더 이상 접근할 수 없다[2]. 그러나 데이터가 다른 장치에 복사될 수 있으며, 완전히 암호화되지 않았기 때문에 데이터를 복구할 수 있다. 한편, 암호화 랜섬웨어는 피해자의 파일을 암호화하며, 암호화하기 위해 사용되는 암호화 알고리즘은 수학적 가정에 따라 안전하도록 설계되었기 때문에 유효한 비밀 키 없이는 복구할 수 없다. 피해자는 파일을 복구하기 위해 해커에게 비용을 지불해야 하고, 얻어낸 비밀 키를 사용하여 원본 파일을 복구할 수 있다. 대부분의 사용자가 디지털 기기를 사용하고 있기 때문에 랜섬웨어는 큰 위협이 될 수 있다. 따라서, 랜섬웨어 공격을 방지하기 위해 탐지 및 피해 복구에 많은 노력을 기울이고 있는 상황이다.

랜섬웨어 탐지를 위한 데이터 트래픽 분석과 함수호출 분석을 통한 방법이 있으며, 성능 향상을 위해 머신 러닝이 활발히 연구되고 있다. 또한, 암호화 랜섬웨어의 특성상 많은 연구들이 암호화 기능에 초점을 맞추고 있다.

참고문헌 [3]에서는 랜섬웨어 네트워크의 행위와 패킷 선택을 분석하여 랜섬웨어 바이러스를 식별한다. 참고문헌 [4]에서는 랜섬웨어 바이러스 탐지를 위해 API의 우위를 분석하여 랜섬웨어를 특성화하고 구별하였다. 참고문헌 [5]에서는 다단계 빅데이터 마이닝을 위한 프레임워크를 활용한다. 해당 연구에서 랜섬웨어는 지도 머신 러닝을 통해 함수 호출, 동적 연결 라이브러리(DLL), 기계어 코드 수준 등 다양한 수준에서 분석되었다.

참고문헌 [6]에서는 동적 바이너리 분석을 통해 수집된 데이터를 암호화 코드를 특성화하기 위해 활용하였다. 참고문헌 [7]에서는 바이너리 코드에서 블록 암호 알고리즘과 매개변수를 자동으로 식별하는 접근 방식을 제시하였으며, 이는 정적 기반 분석 방법이다. 참고문헌 [8]는 랜섬웨어가 공개키 알고리즘을 수행하여 파일을 암호화하기 때문에 비대칭키 암호화(AKC) 알고리즘을 대상으로 하였으며, 공개키 암호화 알고리즘에 의해 수행되는 암호화 프로세스를 감지하기 위해 정수 곱셈 명령을 모니터링 하였다.

그러나 블록 암호구조는 이와 같은 관련 연구 분야에서 거의 주목받지 못했으며, 사물인터넷(IoT) 환경에서 랜섬웨어를 방어하기 위한 연구도 비교적 많지 않다.

본 논문에서는 저전력 임베디드 프로세서에서 블록 암호의 암호화 과정을 분류하여 랜섬웨어 바이러스를 예방하는 두 가지의 새로운 접근 방식을 제안한다. 대상 임베디드 프로세서 상에서의 암호 알고리즘 소스코드를 분석하여 암호화 랜섬웨어와 정상적인 소프트웨어로 분류하도록 한다. 이를 위해 인공신경망을 사용하며, FELICS(Fair Evaluation of Lightweight Cryptographic Systems)의 블록 암호 알고리즘과 Alf와 Vegard의 RISC(AVR) 프로세서 패키지의 정상적인 프로그램의 바이너리 코드를 학습하고 평가한다.

먼저 각 블록암호와 정상 펌웨어의 바이너리 파일을 두 가지 방식으로 전처리한다. 하나는 opcode의 흐름을 나타내는 시계열 데이터이고 다른 하나는 opcode의 빈도수이다. 즉, 하나의 소스 데이터에서 opcode의 순서와 빈도수라는 두 가지 특징을 추출한 후, 각각의 신경망에 입력하며, 후위에서 두 네트워크를 융합하여 암호화 프로세스의 특징을 학습한다.

또 다른 제안 방법은 트랜스포머의 인코더 구조를 기반으로 하는 방식이다. 바이너리 파일에서 추출한 opcode들을 문장을 구성하는 단어에 대응시킨다. 즉, opcode sequence는 단어들이 나열된 하나의 문장이 되며, 해당 opcode들을 단어 임베딩하여 벡터 형태로 표현한 후 신경망에 입력하여 각 opcode간의 연관성, 순서, 패턴 등의 정보를 학습한다.

이와 같이 두 제안 기법에 대한 학습을 진행한 후, 학습된 모델에 파일 암호화 과정 또는 양성 펌웨어의 바이너리 파일을 입력하여 분류한다. 제안방법

은 암호화 과정을 성공적으로 분류하고 랜섬웨어 바이러스를 탐지하였다.

본 논문의 2장에서는 관련 연구에 대해 설명하고, 3장 및 4장에서 두 가지의 제안 기법에 대한 설명과 실험 및 평가를 진행한다. 마지막으로 5장에서 결론을 맺는다.

제 2 장 관련 연구

제 1 절 IoT 환경에서의 랜섬웨어

IoT의 급속한 발전으로 인해 랜섬웨어 바이러스 방지 및 보안 강화가 IoT 기반 서비스의 기본 구성 요소로 자리를 잡고 있다[9]. 안전한 IoT 환경을 위해 다음과 같이 많은 연구들이 진행되고 있다. 참고문헌 [10]에서는 머신러닝 기반의 랜섬웨어 탐지 방법을 제시하고 있다. 일부 프로세스의 전력 소비 패턴을 모니터링한 후 양성 애플리케이션에서 악성 랜섬웨어를 탐지한다. 참고문헌[11]에서는 opcode 시퀀스를 이용하여 악성코드를 탐지하는 딥러닝 기반의 방법을 제시하고 있으며, opcode는 벡터 공간으로 변환된 후 학습된다. 참고문헌 [12]에서는 행위 기반 방식을 제안한다. TCP/IP(Transmission Control Protocol/Internet Protocol) 헤더를 추출하고 C&C(Command and Control) 서버 블랙리스트에 올려 랜섬웨어 공격을 탐지한다. 참고 문헌 [13]에서 명령 시퀀스는 이미지로 변환된 후, 차원 축소 및 통계적 방법을 사용하여 다중 클래스를 분리합니다. 그러나 이러한 접근 방식들은 고급 IoT 플랫폼에 중점을 두었으며 원거리에서 데이터를 수집하기 위한 IoT 서비스에는 저전력 마이크로컨트롤러가 사용된다. 따라서 IoT 기반 서비스의 보안성 향상을 위해서 저전력 마이크로컨트롤러에 대한 랜섬웨어 탐지 메커니즘이 필요하다. 본 연구에서는 자원 제약적인 저사양 IoT 환경에서 암호화 랜섬웨어를 탐지하는 새로운 메커니즘을 제시한다.

제 2 절 암호화 함수 호출 기반의 랜섬웨어 탐지 기법 동향

암호화 랜섬웨어는 피해자의 파일을 암호화하기 위해 암호화 기능을 사용하므로 이를 탐지하기 위해서는 암호화 기능을 탐지해야 한다. [표 2-1]은 암호화 함수 호출 기반의 랜섬웨어 탐지 방법을 비교한 결과이다. 참고문헌[6]에서는 암호화 함수의 특징을 통해 대칭키와 공개키 암호를 검출하였다. 이

접근 방식은 대상 아키텍처를 기반으로 하는 발견적 방법을 사용하였다. 참고문헌 [7]에서는 바이너리 파일에서 추출한 데이터 흐름 그래프를 활용하였고, 암호화 함수 호출은 하위 그래프 동형을 사용하여 식별된다. 참고문헌 [8]에서는 RSA(Rivest-Shamir-Adleman) 알고리즘에서 많이 호출되는 곱셈 명령어를 모니터링하여 공개키 암호 알고리즘을 감지하도록 하였다. 최근, [14]에서는 딥러닝 알고리즘을 통해 악성코드 탐지를 개선할 수 있음을 보여주었다. 그러나 이 방법은 암호화 랜섬웨어를 대상으로 하지 않으며 고급 데스크톱이 대상 플랫폼이다. 본 논문에서는 마이크로컨트롤러를 대상으로 하는 암호화 랜섬웨어 탐지 메커니즘을 제시한다.

Category	[6]	[7]	[8]	This Work
Cryptography	SKC and AKC	SKC	AKC	SKC
Approach	Dynamic	Static	Dynamic	Static
Algorithm	Heuristics	Data graph flow	System monitor	Deep learning
Architecture	Desktop	Desktop	Desktop	Embedded Processor

[표 2-1] 암호화 함수 호출 기반의 랜섬웨어 탐지 기법 비교 (SKC와 AKC는 각각 Symmetric Key Cryptography(대칭키 암호알고리즘)와 Asymmetric Key Cryptography(비대칭키 암호 알고리즘)를 의미)

제 3 절 Fair Evaluation of Lightweight Cryptographic Systems (FELICS)

2015년에 룩셈부르크 대학교(University of Luxembourg)에서 암호화 벤치마킹 프레임워크(FELICS)를 발표하였으며[15], 전 세계의 암호화 엔지니어들

은 FELICS에 임베디드 프로세서에 대한 다수의 블록 암호 구현을 제출하였다.

본 논문에서는 FELICS의 블록 암호 구현물을 데이터로 사용하였으며, 저전력 IoT 환경에서 널리 사용되는 8-bit AVR인 ATmega128을 대상 마이크로컨트롤러로 하였다. 마이크로컨트롤러는 수정된 Harvard 아키텍처 기반 8-bit 단일 칩이며 8-bit 단위의 레지스터와 명령어를 사용한다.

또한, 본 논문에서 대상으로 하는 블록 암호는 다음과 같은 두 가지 범주로 분류할 수 있다. Addition, Rotation, Bitwise eXclusive-or(ARX) 및 Substitution-Permutation-Network(SPN)이 있으며, 두 아키텍처는 수행하는 연산과 전체 구조가 서로 다르며 특징 지을 수 있다. 본 논문에서는 두 아키텍처의 특징을 기반으로 바이너리 코드를 분류하였다.

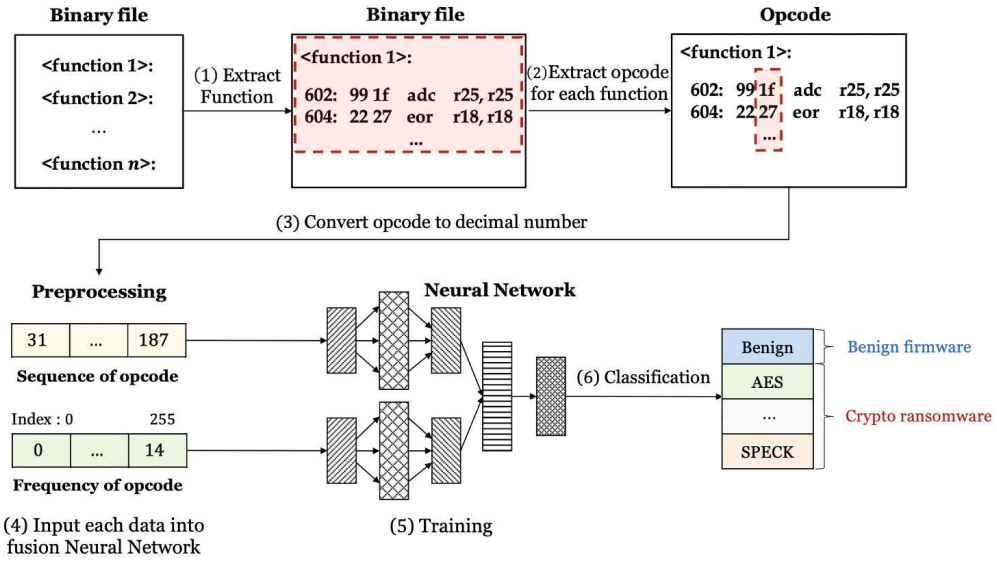
제 3 장 제안 기법

본 논문에서는 저사양 IoT 기기를 위한 인공신경망 기반의 암호화 랜섬웨어 탐지 방법을 제안한다. 일반적으로 저사양 IoT 기기는 센서 데이터 등을 수집하기 위해 리프노드로 사용된다. 기지국 (펌웨어 서버)에서 해당 기기들을 관리하며, 더 나은 서비스를 위해 정기적으로 IoT 기기의 펌웨어를 업데이트 한다. 이 때, 암호화 랜섬웨어가 IoT 기기들에 삽입될 위험이 있으며, 기지국 또는 해당 기기들이 암호화 랜섬웨어 바이러스를 감지해야한다. 제안 기법은 펌웨어가 디바이스에 배포되기 전에 인공 신경망을 활용하여 암호화 프로세스의 실행 여부에 따라 암호화 랜섬웨어와 정상 펌웨어를 분류하며, 이에 따라 저사양 IoT 기기에서 암호화 랜섬웨어를 탐지할 수 있도록 한다. 컨볼루션 퓨전 신경망 및 트랜스포머의 인코더 구조를 기반으로 하는 두 가지 방법을 제안하며, 바이너리 코드로부터 추출한 데이터를 활용한다. 랜섬웨어는 피해자의 파일에 대해 암호화를 수행하므로, 암호화 과정이 랜섬웨어의 특징이 될 수 있다. 제안 기법은 암호화 프로세스의 실행 여부 판단을 위해 경량 블록 암호 알고리즘을 대상으로 한다. 경량 블록 암호 알고리즘의 소스코드를 컴파일하면 소스코드의 바이너리 파일을 얻을 수 있으며, 해당 바이너리 파일에는 암호화, 복호화, 키스케줄 등의 여러 함수에 대한 명령어들이 존재한다. 서로 다른 암호 알고리즘은 다른 연산을 수행하므로 각 알고리즘에서 사용하는 핵심 연산이 특징이 될 수 있다. 따라서 각 알고리즘에서 암호화를 수행하는 핵심 함수들의 명령어들을 추출하고, 이를 암호화 프로세스를 탐지하기 위한 특징으로 사용한다. 또한, Tensorflow Lite 모델을 통해 저사양 기기에서 신경망 추론이 가능하도록 한다.

제 1 절 컨볼루션 퓨전 신경망 기반의 암호화 랜섬웨어 탐지

본 절에서는 컨볼루션 퓨전 신경망 기반의 암호화 랜섬웨어 탐지 기법을 제안한다. [그림 3-1]은 전체적인 시스템 구조도이다. 앞서 언급하였듯이 대상이 되는 경량 블록 암호 알고리즘의 소스코드를 컴파일한 후 얻은 바이너

리 파일로부터 핵심 함수를 추출한다. 이후, 해당 핵심 함수들의 opcode를 순서대로 추출한다. 8-bit AVR을 타겟보드로 하였으므로 opcode는 8-bit로 표현 가능한 0~255 사이의 값을 가진다. 또한, 문자열 형태이므로 16진수 opcode로 바꾼 후 10진수로 변환되도록 인코딩한다. 변환된 데이터는 opcode 순서를 나타내는 배열과 opcode의 빈도수를 나타내는 배열로 표현되고, 해당 배열들은 서로 다른 신경망에 입력되고, 정상 펌웨어와 암호화 랜섬웨어 또는 각각의 암호화 알고리즘으로 분류된다.



[그림 3-1] 컨볼루션 퓨전 신경망 기반의 랜섬웨어 탐지 기법의 구조도

1) 데이터 생성 및 전처리

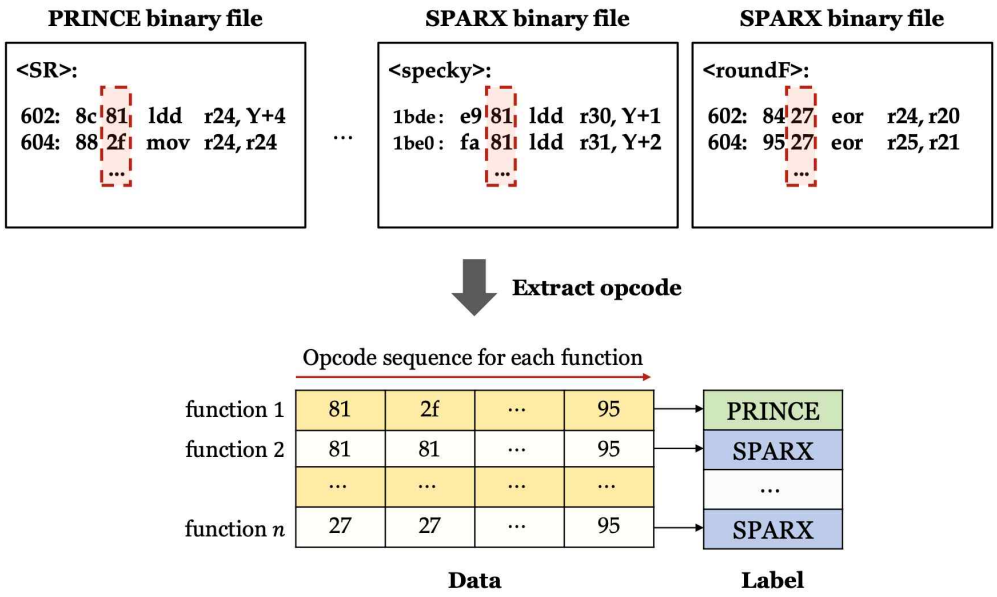
경량 블록 암호 알고리즘의 소스코드를 컴파일하여 바이너리 파일을 얻을 수 있다. 암호화 프로세스는 랜섬웨어의 특징이므로 바이너리 파일에서 암호화를 수행하는 함수를 추출하고, 함수별로 opcode를 추출한다. 또한, 각 함수에 대한 opcode들은 해당 함수의 특징이 되므로 [그림 3-2]와 같이 각 opcode들로부터 생성되는 sequence 및 빈도수 데이터는 해당 opcode들이 속하는 함수를 포함하는 암호 알고리즘의 이름으로 라벨링되도록 한다.

[그림 3-3]은 신경망 학습을 위한 전처리 과정을 나타낸다. 추출한 문자열 형태의 opcode를 16진수로 바꾼 후 10진수로 변환한다. 해당 값은 8-bit

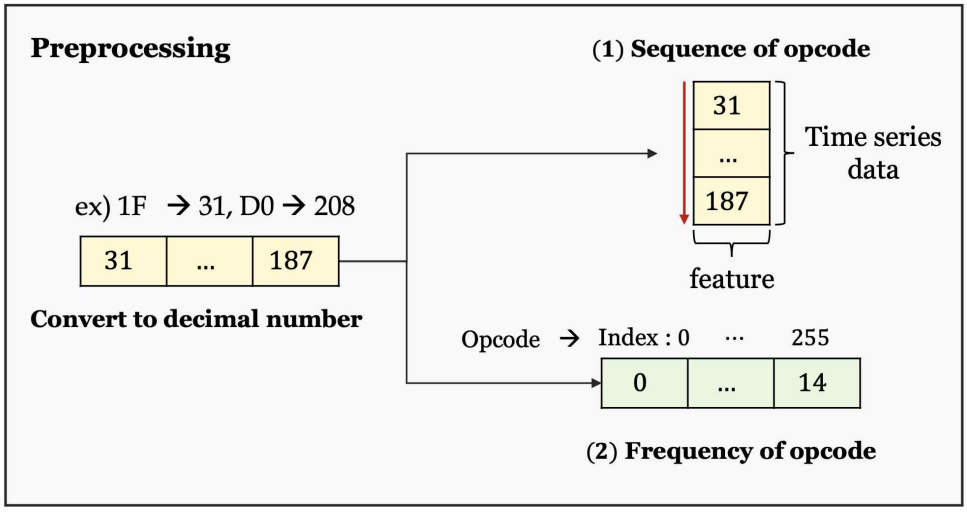
AVR을 타겟으로 하였으므로 0~255로 표현된다. 그러면 0~255 범위의 10진수 형태의 opcode가 생성된다. 각 알고리즘의 핵심 함수로부터 추출된 opcode는 순서대로 수행되므로 시계열 데이터이고, 따라서 각 명령어가 하나의 시계열 데이터의 특징이 되도록 구성한다.

이와 같은 과정을 통해 구성된 각 opcode sequence 배열은 길이가 다르므로 신경망에 입력하기 위해서는 길이를 통일시켜주어야 한다. 모든 sequence 배열에 대해 최소 길이가 14이고, 최대 길이가 13,859 (모든 연산에 대해 함수를 사용하지 않은 경우)이다. 대부분의 소스코드는 함수를 사용하여 작성되었으며, 전체 데이터에서도 2개에 불과하다. 백분위수를 계산한 결과, 길이가 878인 경우가 전체 데이터의 90%를 차지한다. 따라서 opcode sequence 배열의 최대길이를 1000으로 설정하였다. 길이가 1000보다 작은 경우 나머지 부분은 0으로 채워지고, 긴 경우는 1000까지 잘리도록 한다.

또한, 앞서 설명하였듯이 opcode sequence를 학습하는 신경망과 빈도수를 학습하는 신경망을 각각 훈련하므로, opcode의 빈도수에 대한 데이터도 생성해야한다. 따라서 10진수 형태의 opcode sequence 배열에 존재하는 opcode들을 빈도수 배열의 인덱스로 사용하여 해당 opcode들이 몇 번 사용되는지 측정한 값을 입력한다.



[그림 3-2] 각 함수에서 opcode sequence 추출

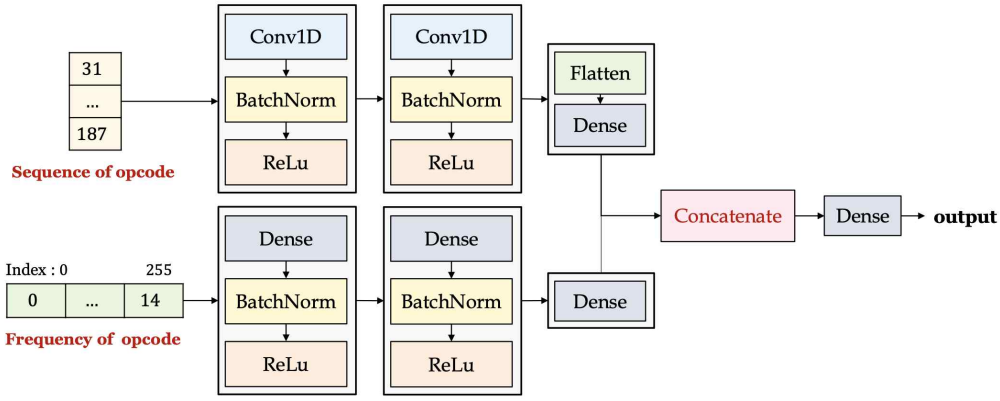


[그림 3-3] 데이터 전처리

2) 신경망 구조

해당 기법에서는 [그림 3-4]와 같이 두 가지 유형의 데이터를 학습하기 위해 두 개의 신경망을 구성한다. 두 신경망은 각 데이터에 대한 특징들을 독립

적으로 학습한 후, 마지막 특징맵에서 하나의 네트워크로 합쳐지며 출력층을 거쳐 분류한다. 즉, 하나의 소스 데이터에서 두 가지 특징을 추출한 후 학습하는 구조이다. 해당 구조는 각 신경망 모델에 대해 오류가 전파되는 후기융합이 적용된 구조이며, 두 모델의 결과를 결합하여 판단하므로 신경망의 성능 향상을 위한 효과적인 방식이다. 또한, 두 입력 데이터에 대한 분류 결과 중 어느 하나가 오분류되어도 두 모델이 서로 독립적이므로 최종 분류 결과가 잘못되는 문제를 방지할 수 있으며, 각 입력에 대한 책임 및 타당성을 고려할 수 있다.



[그림 3-4] 컨볼루션 퓨전 네트워크 구조도

3) 학습 및 탐지

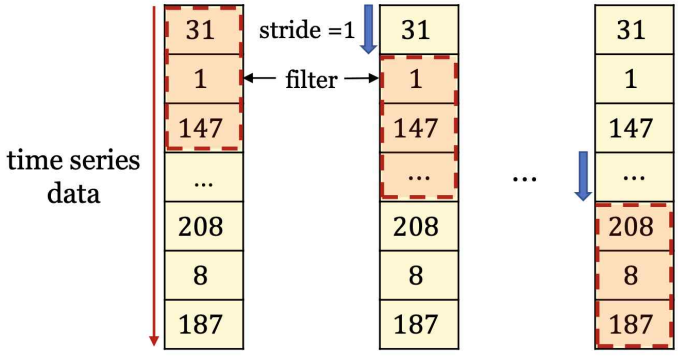
학습 단계에서는 앞서 전처리된 데이터를 신경망에 각각 입력한다. 즉, 각 알고리즘의 핵심 함수에 해당하는 opcode들의 순서와 빈도수를 학습한다. 각 데이터는 각 신경망 모델의 모든 레이어를 거친 후, 11개의 암호화 알고리즘과 1개의 정상 펌웨어로 분류되며, 암호화 알고리즘이 탐지된 데이터는 암호화 랜섬웨어로 판단한다.

시계열 데이터인 opcode sequence는 [그림 3-4]와 같이 컨볼루션 1-dimension(Conv1D)을 주로 사용하여 학습된다. 순환신경망 (Recurrent Neural Network, RNN)이 시계열 데이터 학습에 적합하지만, 제안 기법은

저사양 IoT 기기를 타겟으로 하므로 파라미터가 적으며 필터를 통해 데이터의 순서 정보를 고려할 수 있는 컨볼루션 레이어를 사용한다. [그림 3-5]는 해당 레이어의 동작 과정을 나타낸다. 해당 신경망의 각 행에는 opcode 데이터가 순서대로 저장되어 있으며, 필터가 해당 데이터를 1칸씩(stride) 이동하며 컨볼루션 연산을 수행한다. 해당 필터는 입력 데이터의 맨 앞부터 맨 마지막으로 이동하므로 sequence 데이터를 학습할 수 있다. 이 때, 필터의 범위 내에서 데이터의 특징을 추출할 수 있으며, 필터가 클수록 더 넓은 범위의 sequence를 고려할 수 있다. 필터의 크기 및 개수와 stride 등의 하이퍼파라미터들은 실험을 통해 최적화한다. 또한, 빈도수 데이터는 전연결층으로 구성된 신경망을 거쳐 학습된다. 이는 순서 정보를 학습할 필요가 없으므로 전연결층을 사용하였으며, 해당 네트워크의 하이퍼파라미터 또한 실험을 통해 최적화한다. 제안 기법은 12개의 클래스를 가진 다중 클래스 분류이다. 따라서 출력 계층에 대한 'Softmax' 활성화 함수와 'Categorical crossentropy' 손실함수를 사용한다. 데이터가 1byte opcode이기 때문에 데이터 범위는 0에서 255까지이며, 최대 길이 값은 앞서 언급하였듯이 1000으로 설정됩니다. [표 3-1]은 컨볼루션 퓨전 네트워크의 하이퍼파라미터이며, 실험을 통해 최적화 한 결과이다.

Hyperparameters	Descriptions
The number of labels	12
Range of data	0~255 integer (1-byte opcode)
Sequence max length	1,000
Conv1D filters	256, 64
Conv1D filter size	16, 4
Conv1D stride	1
Batch size	8
Epochs	50
Loss	Categorical crossentropy
Optimizer	Adam (lr = 0.0002)
Activation	ReLu (hidden), Softmax (output)

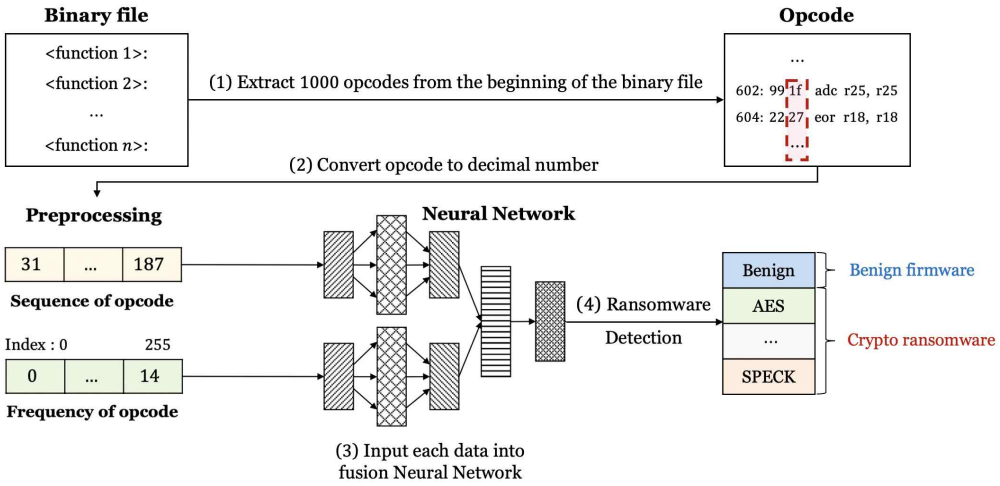
[표 3-1] 컨볼루션 퓨전 네트워크의 하이퍼파라미터



[그림 3-5] Conv1D의 동작 과정

[그림 3-6]은 암호화 랜섬웨어 탐지 과정의 구조도이다. 탐지 단계에서는 위의 과정을 거쳐 학습된 모델을 사용하여 추론한다. 암호화 랜섬웨어 탐지에

사용되는 데이터는 학습 데이터셋으로 사용한 암호 알고리즘들을 사용하여 파일을 암호화하는 코드를 작성한 후, 동일하게 컴파일하여 얻은 바이너리 파일이다. 학습 과정과 달리, 바이너리 파일에서 각 함수를 추출하지 않고 전체 바이너리 파일에 대해 맨 처음부터 opcode를 1,000개씩 추출한다. 학습 데이터와 입력 크기가 같아야 하므로 이러한 과정을 거치며, 전처리 단계는 학습 과정과 동일하다. 학습된 신경망에 전처리 된 데이터를 입력할 경우, 해당 데이터는 정상 펌웨어, AES(Advanced Encryption Standard), PRESENT 등 12개의 클래스 중 하나로 분류된다. 분류된 클래스가 암호화 알고리즘인 경우 해당 데이터는 암호화 과정을 수행하는 것으로 판단하며, 암호화 랜섬웨어로 결정된다.



[그림 3-6] 암호화 랜섬웨어 탐지 과정

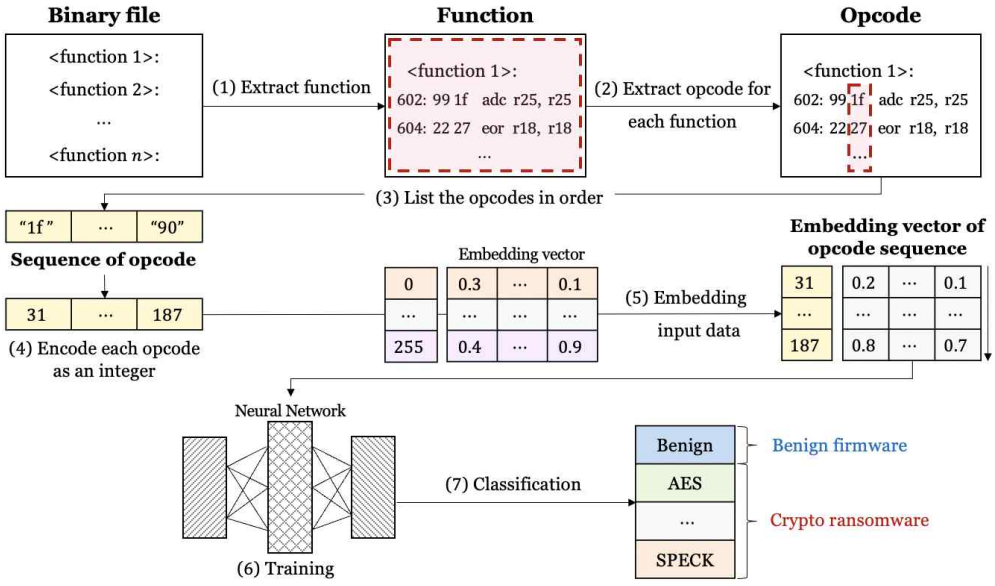
4) Pruning을 통한 네트워크 경량화

Prining은 기존 네트워크의 파라미터를 제거하여 네트워크의 복잡도를 줄임으로써 효율성을 높이면서 정확성을 유지하기 위한 기법이다. 신경망 내부 연산을 수행하기 위한 계산 리소스 및 용량을 줄일 수 있다. 제안 시스템은 저사양 IoT 기기에서 암호화 랜섬웨어를 탐지하는 것이 목적이므로 성능 저하가 발생하지 않는 범위 내에서 Pruning 기법을 적용한다. 또한, Pruning이 적용된 후, 해당 모델은 Edge 장치에서 추론이 가능하도록 tflite로 변환된다.

즉, Pruning 된 tflite 모델을 IoT 기기에 배포하면 IoT 환경에서 암호화 랜섬웨어 탐지가 가능하다.

제 2 절 트랜스포머의 인코더 기반의 암호화 랜섬웨어 탐지

본 절에서는 트랜스포머의 인코더 기반의 암호화 랜섬웨어 탐지 기법을 제안한다. [그림 3-7]은 전체적인 시스템 구조도이다. 3장 1절에서 설명하였듯이 각 암호화 알고리즘의 바이너리 파일에서 핵심 함수를 추출하고, 해당 함수에서 opcode를 추출한다. 1절의 컨볼루션 퓨전 네트워크 기반의 제안 기법은 해당 opcode를 10진수로 변환하였으나 본 제안 기법에서는 문자열 형태의 opcode를 문장의 단어로써 사용한다. 즉, 10진수로 변환하지 않으며 각 opcode를 벡터로 임베딩한 후, 해당 벡터를 입력데이터로 사용한다. 1절의 제안 기법과 달리 빈도수 특징을 갖는 데이터와 신경망 없이 단일 네트워크로 구성하여 학습 및 추론을 수행하고, 정상 펌웨어와 암호 알고리즘으로 분류되며 암호 알고리즘은 암호화 랜섬웨어로 판단한다.

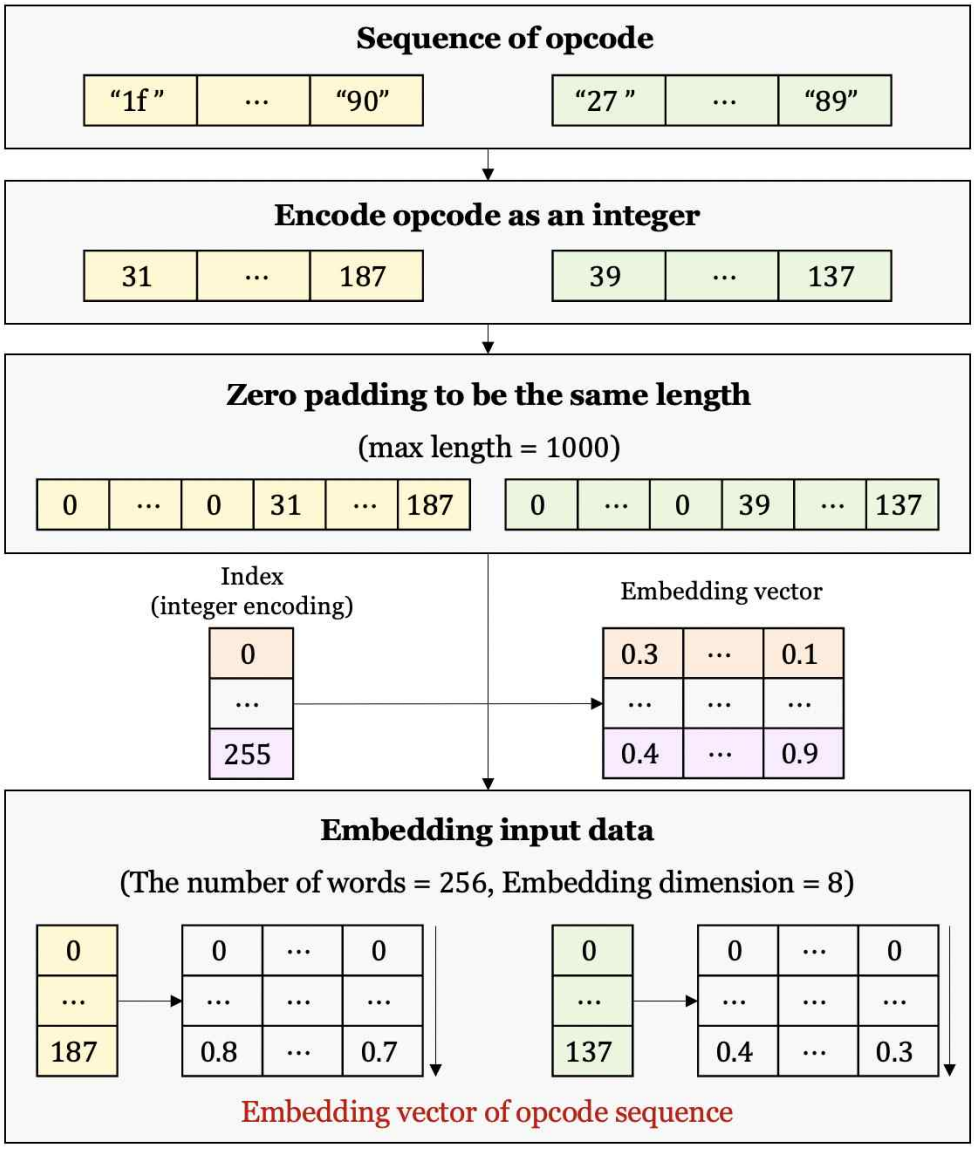


[그림 3-7] 트랜스포머의 인코더 기반의 암호화 랜섬웨어 탐지 시스템 구조도

1) 데이터 생성 및 전처리

1절의 데이터 생성 과정에서 설명한 것과 같이 8-bit AVR을 타겟으로 하여 컴파일하여 바이너리 파일을 얻는다. 바이너리 파일에서 각 알고리즘의 핵심함수를 추출한 후, 해당 함수 내의 opcode를 추출한다. 추출된 opcode들은 순서대로 나열되며 시계열 데이터가 된다. 본 제안 기법에서는 이러한 시계열 데이터의 학습을 위해 트랜스포머 네트워크를 사용한다. 따라서 각 opcode를 단어로 보고 opcode sequence를 문장으로 생각한다. 이러한 데이터를 네트워크에 입력하기 위해서는 단어 임베딩을 통한 전처리 과정이 필요하다. 하나의 단어인 opcode를 d차원 벡터로 나타내는 과정이다. 총 256개의 opcode가 존재하므로 이를 one-hot 인코딩 할 경우, 하나의 opcode가 256차원의 벡터로 표현된다. 임베딩 차원은 실험을 통해 8차원으로 설정하였으며, 단어의 개수는 opcode의 개수인 256으로 설정하였다. 임베딩 차원을 256차원에서 8차원으로 감소시켰으므로 메모리 사용량을 줄일 수 있으며, 이는 저사양 IoT 기기에서의 추론 시 이점이 된다. 또한, 임베딩 과정 이전에 1절과 동일하게 제로 패딩(zero padding)을 수행하여 배열의 길이를 1,000으로 맞추어 주어야 한다.

이와 같은 전처리 과정은 다양한 임베딩 기법을 통해 수행될 수 있으며, 본 연구에서는 embedding layer를 통해 진행하였다. 따라서 해당 레이어를 트랜스포머의 인코더 구조의 앞쪽에 붙임으로써 신경망의 입력 레이어로 사용하여, 학습 과정에서 역전파 시의 기울기를 가지고 임베딩 값을 조정해나간다.



[그림 3-8] 입력 데이터 전처리를 위한 opcode 임베딩 과정

2) 신경망 구조 및 학습

[그림 3-9]는 제안 기법의 신경망 구성도이다. 트랜스포머 모델 중 인코더만을 사용하며, 인코더 뒷부분에는 다중 클래스 분류를 위한 분류기를 덧붙인다. 트랜스포머 모델을 사용함으로써 시계열 데이터 학습의 순차적인 처리과정을 없애고, 입력 데이터를 한 번에 처리하도록 하므로 병렬처리가 가능하

고, 효율성이 향상된다.

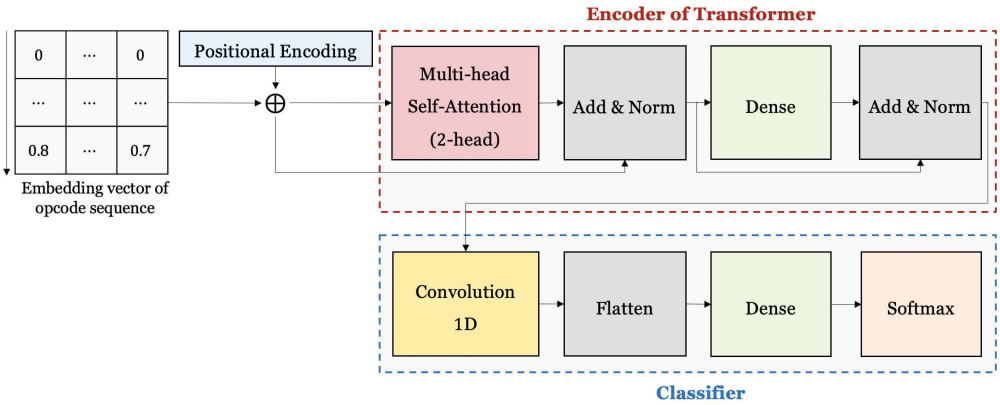
먼저, 전처리 과정에서 임베딩된 벡터를 네트워크에 입력한다. 이 때, 시계열 데이터이므로 순서 정보를 고려하기 위해 'Positional Encoding'이 수행된다. 해당 과정은 토큰 간 거리, 긴 문장 표현, 위치 값 예측 등이 가능한 sine(sin), cosine(cos) 함수를 통해 -1~1 사이의 값으로 인코딩한다. 전체 차원이 d차원일 경우, 각 opcode의 위치 정보 값에 대해 짝수는 sin, 홀수는 cos을 적용하여 위치 정보를 담는 행렬을 생성한다. 이후, 입력된 임베딩된 opcode를 나타내는 행렬과 요소별 덧셈을 수행한다. 이를 통해 입력된 opcode들의 순서 정보를 학습하도록 한다.

위치정보가 더해진 값은 트랜스포머 모델의 인코더에 입력된 후, 분류기에 입력된다. 즉, 제안 기법에서는 트랜스포머 모델의 인코더와 디코더 중 인코더만을 사용한다. 이를 통해 입력된 opcode sequence의 모든 요소들의 상관관계를 분석하며, 문맥에 대한 전반적인 정보와 특징을 추출한다. 인코더에 입력되면 그림에서 볼 수 있듯이 'Multi-head self-attention'를 거친다. 해당 부분은 여러 개의 attention을 수행하는 부분이며, attention 기법은 Query를 기준으로 각 Key들이 얼마나 영향을 미치는지 계산한 후, 해당 결과를 반영하여 학습 시에 필요한 가중치를 조정한다. 이 때 각 opcode들이 Query 또는 Key가 되므로 opcode 간의 연관성과 다양한 특징을 학습할 수 있다. 또한, 각 head마다 서로 다른 Value, Key, Query로 나뉘므로 서로 다른 결과를 얻는다. 즉, 여러 개의 head를 사용함에 따라 각 opcode에 대한 다양한 관계와 특징을 학습할 수 있으며, head의 개수는 실험을 통해 2개로 설정하였다. 또한, feed forward를 위한 'Dense' 레이어의 유닛은 4로 설정하였다. 이외에도 정규화를 위한 레이어의 하이퍼파라미터 등이 있다.

1절의 제안 기법과 비교하면, Convolution 1-dimension의 경우는 필터 범위 외의 데이터는 고려하기 어렵지만 본 제안 기법의 경우 전체 opcode sequence를 한 번에 입력하고, 인코더를 통해 전체 문맥과 요소 간의 연관성을 학습하기 때문에 더 효율적이며 적합한 방법으로 판단된다.

위와 같은 트랜스포머의 인코더 부분을 거친 후, 분류를 위한 네트워크에

입력된다. 두 부분은 개별 네트워크가 아니며, 인코더의 출력을 'Convolution 1D' 레이어에 입력한다. 해당 레이어는 필터를 통해 가중치를 공유하므로 파라미터가 적으며, 1절에서 설명한 것과 같이 시계열 데이터를 고려할 수 있다. 'Flatten' 레이어를 통해 1차원 벡터로 평탄화 해준 후, 출력층에 입력되어 12개의 클래스로 분류하도록 한다. [표 3-2]는 제안 기법의 하이퍼파라미터이며, 해당 값들은 실험을 통해 설정되었다.



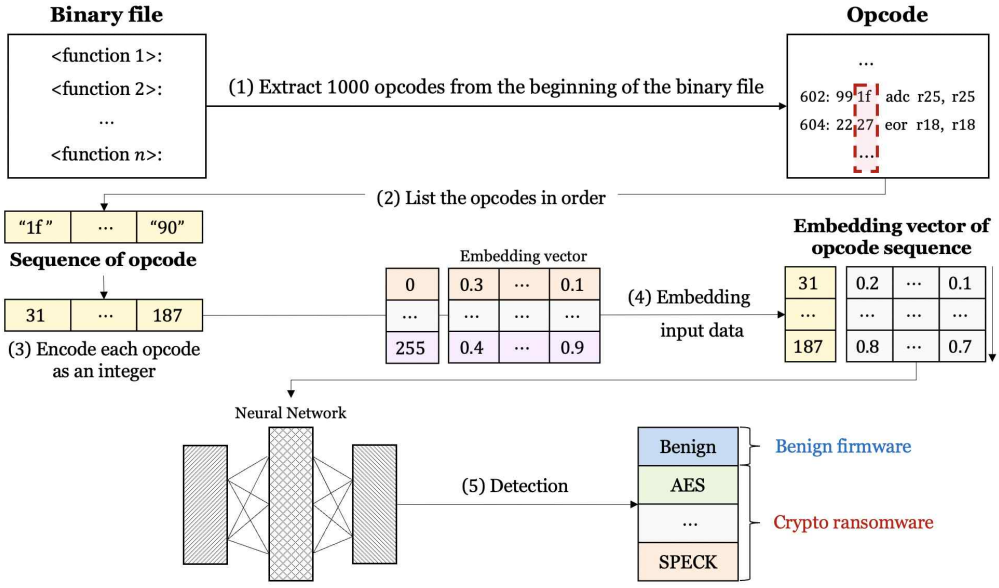
[그림 3-9] 트랜스포머 인코더 기반의 신경망 구성도

Hyperparameters	Descriptions
The number of labels	12
Sequence max length	1,000
The number of words	256 (0~255 opcodes)
Embedding dimension	8
The number of heads	2
The number of encoder	1
Epsilon of normalization	1e-6
Feed Forward unit	4
Conv1D filters	16
Conv1D kernel size	8
Conv1D stride	1
Batch size	32
Epochs	70
Loss	Sparse categorical crossentropy
Optimizer	Adam (lr = 0.0002)
Activation	ReLu (hidden), Softmax (output)

[표 3-2] 트랜스포머 기반 네트워크의 하이퍼파라미터

3) 암호화 랜섬웨어 탐지

[그림 3-10]은 저사양 IoT 기기에서의 암호화 랜섬웨어 탐지 과정이다. 1절에서 언급하였듯이 학습된 네트워크를 IoT 기기에서의 추론이 가능한 TFLite 모델로 변환한 후 배포한다. 주어진 암호 알고리즘을 통해 파일을 암호화하는 코드를 작성하고, 컴파일하여 얻은 바이너리 파일의 맨 처음부터 opcode를 1,000개씩 추출한 후, 제안 기법의 전처리 과정을 거친다. 이후, 학습된 신경망에 입력하면, 정상 펌웨어와 각 암호 알고리즘으로 분류가 가능하다.



[그림 3-10] 트랜스포머 인코더 기반의 암호화 랜섬웨어 탐지 단계의 구성도

제 4 장 실험 및 평가

제 1 절 실험 환경

본 실험에서는 클라우드 기반 서비스인 Google Golaboratory를 사용하였으며, Intel Xeon CPU(13GB RAM), Nvidia GPU(12GB RAM) 및 Ubuntu 18.04.5 LTS를 지원한다. 프로그래밍 환경으로는 Python 3.7.11 및 Tensorflow 2.6.0을 사용하였다. 또한, 타겟 보드로는 ATmega128를 대상으로 하였으며, GCC-AVR 컴파일러를 사용한다.

제 2 절 데이터

3장에서 언급하였듯이 암호화 랜섬웨어가 실행될 때, 정상 펌웨어와 달리 암호화 프로세스를 수행하기 때문에 암호화 연산을 랜섬웨어의 특징으로 설정하였다. [표 4-1]은 본 실험에서 사용한 데이터셋이다. 대칭키 암호 알고리즘인 substitution-permutation-network(SPN) 및 addition-rotation-exclusive-or(ARX) 구조에 해당하는 경량 블록 암호 알고리즘과 IoT 환경에서의 일반 펌웨어로 구성된다. SPN과 ARX에 속하는 암호 알고리즘들은 FELICS 구현 중 C 언어로 작성된 암호화 모듈이며, 일반 펌웨어로는 RFID(Radio-Frequency Identification), WiFi, xBee, Bluetooth 등이 사용되었다. 괄호 안의 값은 데이터 개수이며, 탐지 단계에서는 테스트 세트에 속하는 경량 블록 암호 및 일반 펌웨어의 바이너리 파일을 맨 앞부터 1,000개씩 잘라 사용한다.

Architecture	Examples
SPN (88)	RECTANGLE(5), PRIDE(25), PRINCE(33), PRESENT(10), and AES(15)
ARX (191)	SPECK(37), RC5(12), LEA(9), SIMON(19), HIGHT(58), and SPARX(56)
Benign (66)	XBee, GPS, WiFi, Bluetooth, and RFID

[표 4-1] 데이터셋 세부사항

제 3 절 컨볼루션 퓨전 신경망 기반의 암호화 랜섬웨어 탐지

컨볼루션 퓨전 신경망 기반의 암호화 랜섬웨어 탐지에 대한 실험은 3가지 방법으로 수행된다.

첫 번째 실험은 opcode를 추출하여 수행하며, 동일한 과정을 opcode 뿐만 아니라 operand까지 포함된 전체 instruction을 추출한 후 성능을 측정한다. 두 번째 실험은 opcode 기반의 학습을 진행하지만, opcode sequence만 고려한 네트워크와 opcode 빈도수와 sequence를 모두 고려한 네트워크에 대한 실험 및 평가를 진행한다. 마지막으로는 빈도수 및 opcode 순서 정보를 모두 고려하는 opcode 기반의 네트워크를 경량화한 후 성능 평가를 진행한다.

본 실험에서는 정밀도와 재현율의 조화 평균인 F-measure를 사용하여 성능을 측정한다. 또한, 모든 클래스에 대해 데이터의 수가 동일하지 않은 불균형 데이터셋을 사용하므로 각 실험에 대해 micro F-measure (각 클래스를 동일한 가중치로 고려하는 것이 아니라, 각 클래스의 데이터 개수를 고려)를 사용하여 평가하며, 해당 결과는 10회 실험의 평균값이다. Validation F-measure, Test F-measure는 각각 Validationset과 Testset을 통해 실험한 결과이며, Detection F-measure는 Test set에 대해 학습 시의 전처리 과정이 아닌 탐지 단계의 전처리 과정을 적용한 후 실험한 결과이다.

또한, 각 평가는 Each algorithm vs. Benign firmware, SPN vs. ARX vs.

Benign firmware 그리고 Cryptoransomware vs. Benign firmware로 나뉜다. 각 실험은 개별 암호화 알고리즘 및 정상 펌웨어를 분류하는 실험, SPN 구조의 암호화 알고리즘과 ARX 구조의 암호화 알고리즘 및 정상 펌웨어를 분류하는 실험, 그리고 암호화 알고리즘을 암호화 랜섬웨어로 판단하여 정상 펌웨어와 분류하는 경우에 대한 평가이다.

1) Opcode 기반 vs Instruction 기반에 대한 성능 평가

해당 실험은 3장의 제안 기법인 opcode 기반의 학습과 instruction(operand + opcode) 기반의 학습의 성능을 비교한다. Instruction 기반의 경우 전처리 방법은 동일하지만, operand가 해당 암호화 연산의 특징값으로 포함된다. 동일 연산을 수행할 경우, 동일한 opcode를 갖는다. 그러나, operand는 동일한 연산을 수행하여도 다를 가능성이 있으며, 실제 바이너리 파일을 확인한 결과 operand가 다른 경우가 다수 존재하였다. [표 4-2]와 [표 4-3]은 opcode 기반 및 instruction 기반 학습에 대한 F-measure이다. Opcode 기반의 방법이 instruction 기반의 방법보다 전반적으로 높은 성능을 보였다. 전체 암호 알고리즘을 암호화 랜섬웨어로 판단하고, 정상 펌웨어와 분류하는 경우는 동일한 성능을 달성하였으나, 각 알고리즘과 정상 펌웨어를 분류(12개의 클래스)하는 경우에 대해서는 validation, test, detection에서 각각 4%, 10%, 10% 더 높은 성능을 달성한 것으로 보아 opcode만을 반영하는 것이 더 정확한 분류가 가능한 것으로 생각된다. 이러한 이유로 instruction 기반이 아닌 opcode만을 추출하는 방법을 제안한 것이다.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-measure
Each algorithm vs. Benign firmware	0.93	0.85	0.80
SPN vs. ARX vs. Benign firmware	0.99	0.98	0.97
Cryptoransomware vs. Benign firmware	0.99	0.98	0.97

[표 4-2] Opcode 기반 학습에 대한 F-measure

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-measure
Each algorithm vs. Benign firmware	0.89	0.75	0.70
SPN vs. ARX vs. Benign firmware	0.95	0.93	0.95
Cryptoransomware vs. Benign firmware	0.99	0.98	0.97

[표 4-3] Instruction (operand+opcode) 기반 학습에 대한 F-measure

각 알고리즘과 정상 펌웨어를 분류하는 경우, Opcode 기반 방법에서는 SPECK과 LEA 또는 SPECK과 SPARX 간의 오분류가 발생하였으며, instruction 기반 방법에서는 동일한 실험에 대해 SIMON과 SPECK, SPARX와 SPECK, LEA와 SPECK에서 오분류가 발생하였다.

SPN vs. ARX vs. Benign firmware 실험의 경우, validationset과 testset에

대해 SPECK과 정상 펌웨어를 오분류하였으며, detection 단계에서는 정상 펌웨어가 SPARX로 오분류되었다. 즉, opcode 기반의 경우 SPN구조와 ARX구조에 해당하는 알고리즘을 오분류 하는 경우는 없었다. 그러나, instruction 기반의 방법은 validationset 및 testset에서 RECTANGLE, PRIDE, HIGHT 및 정상 펌웨어에 대한 오분류가 발생하였다. 또한 detection 과정에서 RECTANGLE, SPARX, SPECK 및 일반 펌웨어 간에 오분류가 발생하였으며, SPN 및 ARX 구조가 잘못 분류되는 경우도 존재하였다. 즉, 두 방법이 암호화 랜섬웨어 탐지에 대해서는 동일한 성능을 보이지만, opcode 기반의 방법이 SPN 및 ARX 구조에 대한 오분류가 없었으며, 다른 두 실험에 대해 더 좋은 성능을 달성하였다. 또한, 해당 실험을 통해 블록 암호 알고리즘의 구조에 따른 연산 특징도 존재하며, 함수마다 사용되는 명령어 패턴이 다르다는 것을 알 수 있다.

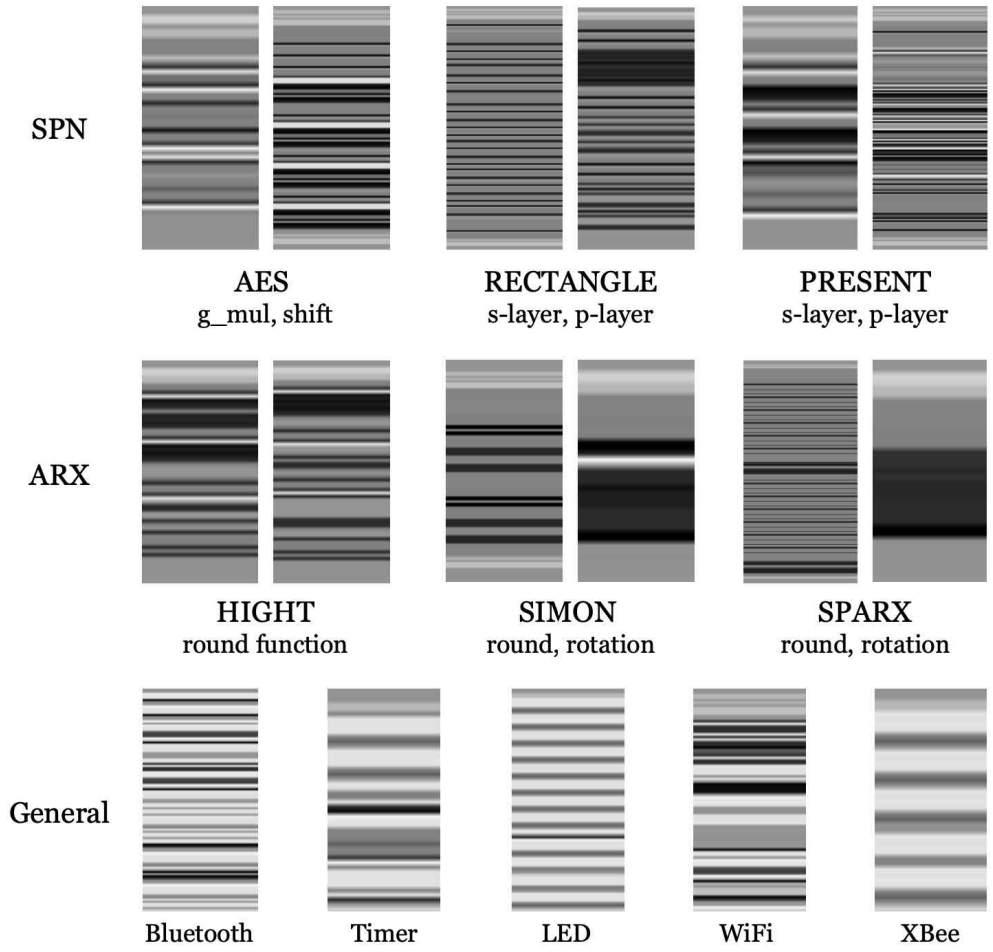
[그림 4-1]은 각 암호 알고리즘에 속하는 핵심 함수와 해당 함수의 opcode sequence를 시각화한 결과이다. 실제로 같은 암호 알고리즘에 속하는 같은 함수의 경우, 같은 연산을 수행하므로 모두 거의 유사한 패턴을 가졌다. 그림에서 볼 수 있듯이 서로 다른 함수는 다른 패턴을 가지며, 하나의 함수 내에서도 유사한 명령어 패턴이 반복된다. 또한, ARX 구조를 갖는 SIMON과 SPARX의 rotation 함수는 거의 비슷한 패턴을 보인다. 정상 펌웨어의 경우는 암호화 알고리즘의 opcode sequence와 비교하였을 때 확연히 다름을 볼 수 있다. [그림 4-2]는 암호 알고리즘의 구조인 SPN과 ARX, 그리고 정상 펌웨어에서 사용되는 opcode들의 빈도수를 시각화한 결과이다. 즉, 0~255사이의 opcode들을 인덱스로 사용하고 각 인덱스마다 해당 opcode가 등장한 횟수를 나타낸 것이다. 따라서 해당 이미지는 256열로 구성되며, 중앙 부분은 128이다. 또한, 빈도수를 이미지로 나타내기 위해 0~255사이의 값으로 정규화 하였다. gray scale 이미지의 경우 0은 검은색을 의미하고, 255는 흰색을 의미한다. 따라서 흰색에 가까울수록 많이 사용된 opcode가 된다. [그림 4-2]와 [표 4-4]를 비교한 결과는 다음과 같다.

SPN 구조의 암호화 알고리즘은 S-box 동작 부분에서 일반적으로 메모리 접근 연산이 자주 수행된다. 실제로 LD, ST 및 MOVW와 같은 메모리 접근

연산이 가장 많이 사용되었으며, 해당 연산은 십진수로 변환할 경우 각각 129, 131 및 1에 해당한다. 따라서 이미지의 가운데 부분에 해당하는 명령어가 자주 사용되었으며, 그림에서 볼 수 있듯이 흰색에 가깝게 표현되었다. 1에 해당하는 이미지의 가장 왼쪽 부분 또한 흰색에 가까움을 확인할 수 있다. XOR, ADIW 등과 같이 ARX와 유사한 연산을 사용하였으나, LD, ST 부분이 가장 선명한 흰색이며 다른 명령어의 경우 상대적으로 회색으로 표현되었다.

ARX(Addition-Rotation-Xor) 구조의 암호 알고리즘은 ADIW, XOR, SBCI, ADC와 같은 산술 및 논리 연산을 자주 수행한다. 세 번째로 많이 사용되는 ADIW의 opcode를 십진수로 변환하면 150에 해당한다. XOR과 ADC는 각각 30과 40이고 중앙으로부터 왼쪽 부분에 위치한다. 이러한 산술 연산들은 SPN 구조에 비해 뚜렷한 흰색으로 나타남을 확인할 수 있다. 이처럼 동일한 구조의 암호화 알고리즘은 유사한 명령 패턴을 갖는 경향이 있다.

이에 비해 일반 펌웨어에는 RJMP, BRNE, OUT과 같은 암호화 코드에서 거의 볼 수 없는 많은 분기 명령과 I/O 레지스터에 접근하는 명령어를 사용한다. 또한, NOP와 같은 연산은 암호화 알고리즘에서 사용되지 않는다. 수집된 일반 펌웨어는 동일한 알고리즘이 아니므로 동작 과정이 서로 다르다. 따라서 암호화 알고리즘과 같은 공통 패턴을 갖기 어려우며, 특정한 명령어를 반복적으로 여러 번 수행하지 않으므로 암호 알고리즘에 비해 뚜렷한 특징을 갖지 않는다.



[그림 4-1] 각 알고리즘의 핵심 함수에 대한 opcode sequence에 대한 시각화

[그림 4-2] 각 알고리즘에 사용된 opcode의 빈도수에 대한 시각화

Architecture	Sorted by Frequency							
	1	2	3	4	5	6	7	8
SPN	LD	ST	MOVW	XOR	ADIW	ANDI	MOV	STD
ARX	LD	MOVW	ADIW	STD	XOR	SBCI	ADC	LDD
Benign	LD	MOVW	RJMP	BRNE	BREQ	NOP	OUT	SBCI

[표 4-4] 각 구조에 따른 명령어 빈도수 비교

2) Sequence-Only vs Frequency and Opcode Sequence에 대한 성능 평가

본 실험에서는 opcode sequence만을 반영한 네트워크와 빈도수 및 opcode를 모두 고려한 네트워크를 평가하였다. 두 가지 특성을 모두 반영하는 제안 기법의 경우, 후기 융합(late fusion)기법을 적용하였다. 앞서 언급했듯이 후기 융합 모델은 입력 중 하나가 잘못되는 문제에 대처할 수 있으며 독립적인 오류 전파를 갖는다. 또한, 두 네트워크의 특징맵을 융합함으로써 각 입력에 대해 학습된 강점(기능)에 중점을 둘 수 있다. 따라서 일반적으로 하나의 네트워크를 사용하는 경우보다 더 나은 성능을 달성할 수 있다.

[표 4-5]는 opcode sequence만을 반영한 sequence-only 모델로 실험한 결과이다. [표 4-2]와 비교하면 암호화 랜섬웨어와 일반 펌웨어를 분류하는 경우, test과 detection에 대해 분류 및 탐지 성공률이 각각 5%, 4% 감소하였다. validation 및 test 경우 SPN과 ARX 사이의 오분류는 발생하지 않았으며, detection에서 두 개의 알고리즘 구조 및 정상 펌웨어 간의 분류 성공률이 10% 감소하였다. 또한, 정상 펌웨어와 각 알고리즘별 분류의 경우, 성능이 크게 저하되었다. 해당 모델은 암호화 랜섬웨어 탐지에 대해, 빈도수를 함께 고려한 경우에 비해서 성공률이 전반적으로 감소하는 경향이 있다.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-measure
Each algorithm vs. Benign firmware	0.86	0.76	0.67
SPN vs. ARX vs. Benign firmware	0.99	0.93	0.87
Cryptoransomware vs. Benign firmware	0.99	0.93	0.93

[표 4-5] Sequence-only 모델에 대한 F-measure

3) 경량화 된 네트워크에 대한 성능 평가

본 실험에서는 학습된 네트워크의 파일 크기를 줄이기 위해 pruning 기법을 적용하였다. [표 4-6]은 pruning 기법이 적용된 네트워크의 F-measure를 나타낸다. 각 암호 알고리즘 및 정상 펌웨어 분류에 대한 성능은 test 및 detection에서 0.03 증가하였고, 구조별 분류에 대한 성능은 test와 detection에서 0.03 감소하였다. 또한, 암호화 랜섬웨어 및 일반 펌웨어 분류의 경우, test에서는 0.03 감소하였고 detection에서는 동일한 성능을 달성하였다. 즉, 암호화 랜섬웨어를 탐지하는 경우에 대해서는 기존의 네트워크와 동일한 성능을 유지하였다. 이와 같이 성능 저하를 최소화해야 하므로, 경량화 후의 파일 크기는 기존 파일에 비해 7%만 감소하였다. [표 4-7]은 pruning 적용 전과 후의 파일 크기와 F-measure이다. 해당 표의 F-measure는 'Cryptoransomware vs. Benign firmware'에 대한 결과이다.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-measure
Each algorithm vs. Benign firmware	0.93	0.88	0.83
SPN vs. ARX vs. Benign firmware	0.99	0.95	0.94
Cryptoransomware vs. Benign firmware	0.99	0.95	0.97

[표 4-6] Pruning 기법을 적용한 네트워크에 대한 F-measure

Model		File Size	F-measure of Detection
Convolution fusion NN	TFLite	1.74MB	0.97
	Pruned TFLite	1.63MB	0.97

[표 4-7] Pruning 적용 전과 후의 파일 크기 및 F-measure

제 4 절 트랜스포머의 인코더 기반의 암호화 랜섬웨어 탐지

본 실험은 트랜스포머의 인코더 및 분류기로 구성된 네트워크를 활용하여 암호화 알고리즘과 정상 펌웨어를 분류하고 암호화 랜섬웨어를 탐지하는 것을 목표로 하였다. 3장 2절에 언급된 하이퍼파라미터들 중 모델의 복잡도를 감소시켜서 경량화 하기 위해 임베딩 차원 및 분류기의 레이어 수 및 뉴런 수에 따른 실험을 진행하였으며 3절과 같이 F-measure를 통해 성능을 평가

하였다. 이외에도 Multi-head self attention의 head 수, feed forward에 해당하는 dense 레이어의 유닛 수 등의 요소가 있으며 해당 요소들도 하이퍼파라미터 튜닝을 통해 설정하였다. 또한, IoT 기기에 배포하기 위한 tflite 모델의 용량을 측정한 후, 3절의 [표 4-7]과 비교하였다.

저사양 디바이스와 같이 리소스가 제한된 환경에서는 성능이 다소 떨어지더라도 크기가 작고 추론이 빠른 모델을 사용한다. 최근에는 모델 성능을 최대한 유지하면서 모델의 크기를 줄이려는 연구들이 다수 진행되었다. 따라서, 성능과 모델 복잡도를 고려하여 효율적인 네트워크를 찾기 위해 해당 실험을 진행하였으며, 임베딩 차원, 분류기의 레이어 뉴런 수 및 커널 크기에 따른 성능을 비교한다. 위의 3가지 요소를 제외한 나머지 하이퍼 파라미터는 최적화를 통해 설정하였으며, 최종적으로는 3장의 [표 3-2]의 하이퍼파라미터를 사용하였다. 또한, 3절의 실험 결과를 통해 11개의 암호화 알고리즘과 정상 펌웨어를 분류하는 것에 대한 F-measure가 가장 낮고 암호화 랜섬웨어 탐지 성능이 가장 높으므로 본 실험은 'Each algorithm vs. Benign firmware'에 대한 validation F-measure를 측정한다. 실험을 거쳐 결정된 값들을 통해 다른 두 실험을 진행한 후 그 결과를 분석하였다. [표 4-8]과 [표 4-9]는 임베딩 차원, 컨볼루션 레이어의 유닛 수 및 커널 크기에 따른 F-measure와 파라미터 수를 나타낸다. 모든 조합에서 0.94이상의 성능을 달성하였으며, 최대 0.03의 차이를 보였다. 각 하이퍼파라미터 조합에 대해 큰 차이가 나지는 않았으며, 가장 높은 성능을 달성한 경우는 kernel size가 8, Conv1D 레이어의 유닛 수가 16, embedding dimension이 8이다. embedding dimension을 256에서 8으로 줄이고 필터의 수를 최소화함으로써 파라미터 수를 확연히 감소시켰으며, 이때의 파라미터 수는 202,432이다. 이는 다른 경우들에 비해 적은 파라미터수를 가지면서도 가장 높은 성능을 보이므로, 저사양 IoT 기기를 타겟으로 하기에 적합한 결과이다.

Embedding dimension = 128		The number of units of Conv1D		
		64	32	16
kernel size	8	0.95 (1,122,640)	0.95 (708,528)	0.96 (501,472)
	16	0.94 (1,182,032)	0.95 (738,224)	0.95 (516,320)

[표 4-8] 임베딩 차원 (Embedding dimension)이 128일 때, Conv1D 레이어의 유닛 수 및 kernel size에 따른 validation F-measure와 파라미터 수 (F-measure (파라미터 수)로 표기)

Embedding dimension = 8		The number of units of Conv1D		
		64	32	16
kernel size	8	0.96 (777,520)	0.95 (394,128)	0.97 (202,432)
	16	0.95 (775,472)	0.94 (393,104)	0.95 (201,920)

[표 4-9] 임베딩 차원 (Embedding dimension)이 8일 때, Conv1D 레이어의 유닛 수 및 kernel size에 따른 validation F-measure와 파라미터 수 (F-measure (파라미터 수)로 표기)

[표 4-10]은 위와 같이 설계된 신경망을 통해 Each algorithm vs. Benign firmware, SPN vs. ARX vs. Benign firmware, Cryptoransomware vs. Benign firmware에 대해 실험을 수행한 결과이다. 분류 결과 및 탐지 결과를 분석한 결과, 3절에 설명한 것과 같이 ARX와 SPN 구조를 오분류하는 경우는 없었으며 SPECK과 SPARX, LEA와 SPECK을 오분류 하는 경우가 존재하였다. 즉, SPN과 ARX는 구조상 서로 사용하는 명령어 패턴이 다르며 그 특징이 뚜렷함을 알 수 있다. 또한, 암호화 랜섬웨어 탐지의 경우는 SPECK과 정상 펌웨어에 대한 오분류가 발생하였다.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-measure
Each algorithm vs. Benign firmware	0.97	0.95	0.85
SPN vs. ARX vs. Benign firmware	1.00	0.98	0.98
Cryptoransomware vs. Benign firmware	1.00	0.98	0.98

[표 4-10] 트랜스포머 인코더 기반의 네트워크에 대한 F-measure

제 5 절 제안 기법 간의 비교

[표 4-11]은 두 제안기법에 대한 파일 크기 및 F-measure를 나타낸다. Opcode를 단어에 대응시키고 하나의 함수에 등장하는 opcode들을 단어들으로 이루어진 하나의 문장으로 대응시켜 트랜스포머의 인코더 구조를 적용한 제안 방법은 3절에 언급한 첫 번째 제안 기법과 달리 빈도수 정보를 넣지 않았으나 하나의 sequence를 한 번에 입력하여 각 opcode들의 연관성과 문맥 등을 학습함으로써 0.01 더 높은 F-measure를 달성하였다. 또한, 파일 크기는 pruning을 적용한 컨볼루션 퓨전 네트워크가 트랜스포머 인코더 기반의 네트워크의 1.96배이다. 성능차이는 크게 나지 않지만 파일 크기를 비교하였을 때 트랜스포머 인코더 구조 기반의 제안 기법이 자원 제약적인 환경에서의 암호화 랜섬웨어 탐지에 있어 더 효과적임을 알 수 있다.

Model		File Size	F-measure of Detection
Convolution fusion NN	TFLite	1.74MB	0.97
	Pruned TFLite	1.63MB	0.97
Transformer encoder NN	TFLite	0.83MB	0.98

[표 4-11] 두 제안기법에 대한 파일 크기 및 탐지 단계의 F-measure

제 6 절 다른 기법과의 비교

[표 4-12]은 다른 기법과 제안 기법을 비교한 표이다. 다른 연구의 경우 OpenSSL 및 Cryptopp 라이브러리와 같은 암호화 알고리즘 구현을 사용하였다. 제안 기법은 8-bit AVR 환경이므로 FELICS에서 C로 구현된 소스코드를 사용하였다.

암호화 랜섬웨어는 주로 AES와 RSA 알고리즘을 사용한다. 따라서 이 두 가지 알고리즘을 탐지할 수 있어야 한다. Grobert et al.[6]에는 chains, mnemonic-const 및 verifier의 세 가지 접근 방식이 있다. 그 중 검증자 식별 방식은 대칭 암호화 과정의 존재 여부와 매개변수를 확인하는 방식으로 가장 성능이 좋으며, 검증자 방법은 AES에서 0.946의 탐지 성공률을 달성했다. 그러나 MD5(Message-Digest) 알고리즘과 RSA는 탐지할 수 없었고, chains 방식을 사용하면 AES와 RSA를 모두 탐지할 수 있다. Caballero et al.[16]에서는 암호화 과정의 존재를 식별할 수 있는 또 다른 요소인 키 스케줄링 과정을 고려한다. 해당 방법은 간단한 메커니즘으로 AES, 데이터 암호화 표준(DES), RC4, MD5 및 RSA를 감지할 수 있다. 본 논문에서 제안하는 방법은 더 다양한 암호화 알고리즘을 다룰 수 있다. 저전력 임베디드 프로세서 환경을 위한 경량 블록 암호 탐지가 가능하였다. 따라서 IoT 환경에 대한

암호화 랜섬웨어를 탐지하기 위해 사용할 수 있다.

	Grobert et al.[6]	Caballero et al.[16]	This Work
Algorithm	Heuristic	Field semantics inference	Deep learning
Implementation	OpenSSL, Cryptopp, Beecrypt	OpenSSL, Cryptopp, Beecrypt	FELICS
Description	AES, DES, RC4, and RSA	AES, DES, RC4, MD5, and RSA	[표 4-1]과 동일

[표 4-12] 다른 기법과의 비교

제 5 장 결론 및 향후 연구 방안

본 논문에서는 임베디드 프로세서에 대한 블록 암호 모듈과 정상 펌웨어를 분류하여 잠재적인 암호화 랜섬웨어를 식별하는 두 가지의 새로운 방법을 제안한다. 첫 번째 제안 기법은 바이너리 파일의 opcode를 10진수로 변환한 후, 변환된 데이터를 함수의 특징을 나타내는 2가지 데이터로 전처리한다. 하나는 opcode의 순서이고 다른 하나는 opcode의 빈도수이며 각 특징을 학습하기 위한 두 개의 신경망을 구성하여 각각 학습한다. 이후, 하나의 네트워크로 융합되므로 두 가지 특성을 모두 고려할 수 있다. 해당 네트워크는 시계열 데이터 학습에 사용할 수 있고 학습할 가중치가 적은 컨볼루션 1D 레이어로 구성되었다. 또한, 각 알고리즘이 수행하는 연산을 분류하기 위해 함수 단위로 구분하여 각 알고리즘의 특징으로 사용하였다. 그리고 블록암호 분류는 블록암호의 고유한 특성을 활용하여 2가지 구조를 갖는다. SPN과 ARX에서 주로 사용하는 opcode의 종류와 빈도수가 다르며, 실험을 통해 빈도수가 분류 성능에 영향을 미치는 특성임을 확인하였다. 이러한 접근 방식은 분류 및 탐지 성능을 크게 향상시켰다. 또한 모델의 용량을 줄이기 위해 가지치기(pruning)를 수행하였다. 그 결과, 파일 크기는 7% 감소했으며 F-measure는 기존 모델과 동일하게 0.97을 달성하였다.

다음으로는, 비슷한 수준의 성능을 달성하면서 리소스가 제한된 환경을 고려하여 모델의 크기를 감소시키는 것을 중점으로 한 기법을 제안하였다. 바이너리 파일로부터 추출한 opcode를 문장을 구성하는 단어에 대응시킨 후, 단어 임베딩을 통해 벡터로 변환한다. 변환된 데이터를 트랜스포머의 인코더 구조 기반의 신경망에 입력한 후 학습한다. 해당 구조를 사용하면 입력 데이터 전체를 순차적으로 처리하지 않고 동시에 처리할 수 있다. 또한, 시퀀스를 고려할 수 있으며 각 opcode 간의 연관성, 특징 등을 추출하고 학습하여 성공적으로 분류해낼 수 있다. 또한, 이러한 제안 기법을 통해 빈도수 특성을 위한 신경망을 따로 구성하지 않을 수 있고, 실험을 통해 임베딩 차원을 감소시켜 네트워크의 크기를 최소화하였다. 해당 기법은 컨볼루션 퓨전 네트워크 기반의 기법에 비해 0.509배의 파일 크기를 가지며, F-measure는 0.01 더 높은

0.98을 달성하였다.

향후 작업으로 또 다른 암호화 프로세스의 특징인 키 스케줄 과정이나, 공개키 암호화 및 해시 함수를 포함하여 다른 암호화 모듈에 대한 분류 방법을 탐색할 것이다. 또한, 다양한 단어 임베딩 기법을 활용하여 더 효율적인 신경망을 구성할 계획이다.

참 고 문 헌

1. 국외문헌

Mohurle, S.; Patil, M. A brief study of Wannacry threat: Ransomware attack 2017. Int. J. Adv. Res. Comput. Sci. 2017, 8.

Kharaz, A.; Arshad, S.; Mulliner, C.; Robertson, W.; Kirda, E. UNVEIL: A large-scale, automated approach to detecting ransomware.

Tseng, A.; Chen, Y.; Kao, Y.; Lin, T. Deep Learning for Ransomware Detection; IEICE Technical Report; IEICE: Tokyo, Japan, 2016; Volume 116, pp. 87-92.

Vinayakumar, R.; Soman, K.; Velan, K.S.; Ganorkar, S. Evaluating shallow and deep networks for ransomware detection and classification. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13-16 September 2017; pp. 259-265.

Poudyal, S.; Dasgupta, D.; Akhtar, Z.; Gupta, K. A multi-level ransomware detection framework using natural language processing and machine learning. In Proceedings of the 14th International Conference on Malicious and Unwanted Software MALCON, Nantucket Island, MA, USA, 1-4 October 2019.

Gröbert, F.; Willems, C.; Holz, T. Automated identification of cryptographic primitives in binary programs. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Menlo Park, CA, USA, 20-21 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 41-60.

Lestringant, P.; Guihéry, F.; Fouque, P.A. Automated identification of cryptographic primitives in binary code with data flow graph

isomorphism. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, Singapore, 14-17 April 2015; pp. 203-214.

Kiraz, M.S.; Genç, Z.A.; Öztürk, E. Detecting Large Integer Arithmetic for Defense Against Crypto Ransomware. 2017. Available online: https://eprint.iacr.org/2017/558/20170608:200345 (accessed on 7 March 2021).

Yaqoob, I.; Ahmed, E.; ur Rehman, M.H.; Ahmed, A.I.A.; Al-garadi, M.A.; Imran, M.; Guizani, M. The rise of ransomware and emerging security challenges in the Internet of Things. *Comput. Netw.* 2017, *129*, 444-458.

Azmoodeh, A.; Dehghantanha, A.; Conti, M.; Choo, K.K.R. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. *J. Ambient Intell. Humaniz. Comput.* 2018, *9*, 1141-1152.

Azmoodeh, A.; Dehghantanha, A.; Choo, K.K.R. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE Trans. Sustain. Comput.* 2018, *4*, 88-95.

Zahra, A.; Shah, M.A. IoT based ransomware growth rate evaluation and detection using command and control blacklist- ing. In Proceedings of the 2017 23rd International Conference on Automation and Computing (ICAC), Huddersfield, UK, 7-8 September 2017; pp. 1-6.

[Karimi, A.; Moattar, M.H. Android ransomware detection using reduced opcode sequence and image similarity. In Proceedings of the 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 26-27 October 2017; pp. 229-234.

Kumar, R.; Xiaosong, Z.; Khan, R.U.; Ahad, I.; Kumar, J. Malicious code detection based on image processing using deep learning. In Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, Chengdu, China, 21-23 March 2018; pp. 81-85.

Dinu, D.; Biryukov, A.; Großschädl, J.; Khovratovich, D.; Le Corre, Y.; Perrin, L. FELICS—Fair evaluation of lightweight crypto- graphic systems. In Proceedings of the NIST Workshop on Lightweight Cryptography, Gaithersburg, MD, USA, 20-21 July 2015; Volume 128.

Caballero, J.; Poosankam, P.; Kreibich, C.; Song, D. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-Engineering. In Proceedings of the 16th ACM Conference on Computer and Communications Security CCS '09, Chicago IL, USA, 9-13 November 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 621-634.

ABSTRACT

Neural Network-based Cryptography Ransomware Detection for Low-End Embedded Processors

Kim, Hyun-Ji

Major in IT Convergence Engineering

Dept. of IT Convergence Engineering

The Graduate School

Hansung University

Crypto-ransomware has a process to encrypt the victim's files, and crypto-ransomware requests the victim for money for a key to decrypt the encrypted file. In this paper, we present two new approaches to prevent crypto-ransomware by detecting block cipher algorithms for Internet of Things (IoT) platforms. The generic software of the AVR package and the lightweight block cipher library (FELICS) written in C language was trained through the neural network, and then we evaluated the result. Sequence and frequency characteristics are extracted from opcodes of binary files obtained by compiling cryptographic algorithms and benign software samples. Through the late fusion method, the above two features are extracted from one source data, learned through each network, and then fused. The proposed method can successfully classify crypto-ransomware viruses or general software. Crypto-ransomware can

also be detected by entering binary code that uses block ciphers to encrypt files into a trained network. Another proposed method does not extract sequence and frequency characteristics, but considers opcodes and opcode sequences as words and sentences, performs word embedding, and then inputs them to the neural network based on the encoder structure of the transformer model. Through this approach, the file size was reduced by 0.5 times while maintaining a similar level of classification performance compared to the first proposed method. The detection success rate for the two proposed methods was evaluated with the F-measured value, which is the harmonic mean of precision and recall. In addition to achieving 97% and 98% crypto-ransomware detection success rates, respectively, classification by normal firmware and lightweight encryption algorithm, Substitution-Permutation-Network (SPN) structure, Addition-Rotation-eXclusive-or structure (ARX) and normal firmware classification is also possible.

【KEYWORD】 deep learning, cryptography, ransomware, internet of things