Journal of KIIT. Vol. 18, No. 2, pp. 25-32, Feb. 28, 2020, pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2020.18.2.25

파일 접근 행태의 LSTM 학습을 활용한 악성 코드 탐지 기법

이 윤 석*

Malicious Code Detection Method Using LSTM Learning on the File Access Behavior

Yunseok Rhee*

본 연구는 2019학년도 한국외국어대학교 교내학술연구비 지원을 받아 수행됨

요 약

본 논문에서는 딥러닝을 통해 다양한 악성 코드의 파일 접근 행태를 학습하고 이를 실시간으로 탐지하는 방법을 제안하였다. 기존의 정적 코드 분석과 달리, 본 연구에서는 독립된 가상 머신 상에서 실제 악성 코드를 동작시키고, 이 코드가 파일 시스템에 접근하는 시스템 콜 패턴과 경향을 수집하여 이를 학습에 활용하였다. 특히, 주요 파일 보호를 위해 FUSE 기반의 사용자 공간 파일 시스템을 이용함으로써, 커널의 수정없이 파일 및 디렉토리에 대한 접근 정보를 수집할 수 있도록 설계하였다. 비교적 긴 입력 패턴의 순차성을 효과적으로 학습시키기 위해, 개선된 RNN 모델인 LSTM 네트워크를 사용하였다. 데이터 셋으로는, 기존 연구에서 활용된 악성 코드 샘플에서 5종을 선정하고, 4,500개의 악성 및 양성 코드 패턴을 생성하였다. 분류 성능은 92.2%의 accuracy를 나타내 기존 기계학습보다 3.8 ~ 44.7%의 향상을 보였다.

Abstract

In this paper, we propose a malicious code detection method based on deep learning of files access behaviors. Unlike the static code analysis, this method runs actual malicious codes on an isolated virtual machine and collects file system call patterns and behavior. In particular, for this study, we employ the FUSE-based user space file system, and it enables collecting all the file access data without modifying the kernel. In order to effectively train long input patterns, the LSTM network, an improved RNN model, is adopted. As a data set, five kinds of malicious code samples used in the previous study are selected and about 4,500 malicious and positive code patterns are generated. The classification performance is 92.2%, which is 3.8 ~ 44.7% better than the existing machine learning.

Keywords
deep learning, file access behavior, malicious code, FUSE, LSTM

* 한국외국어대학교 컴퓨터전자시스템공학부 교수
- ORCID: https://orcid.org/0000-0002-0824-6852

• Received: Jan. 03, 2020, Revised: Feb. 13, 2020, Accepted: Feb. 16, 2020
• Corresponding Author: Yunseok Lee
Dept. of Computer & Electronic Systems Engineering, Hankuk University of Foreign Studies, Oedae-ro 81, Mohyeon, Yongin 17035, Korea
Tel.: +82-31-330-4259, Email: rheeys@hufs.ac.kr

Ⅰ. 서 론

악성 코드(Malicious code 또는 malware)는 개인용 단말 기기, 서버, 클라우드, 네트워크 설비 등에 이르기까지 매우 다양한 정보 기기에서 광범위하게 발견되고 있으며, 바이러스, 웜, 트로이 목마, 스파이웨어, 봇, 루트킷(Rootkits), 랜섬웨어 등의 다양한 형태로 전파되고 있다. 특히, 사물 인터넷(IoT)의 발달과 함께 악성 코드의 피해도 점차 증가하는 추세인데, Cybersecurity Ventures사의 자료에 따르면, 악성 코드 공격에 따른 피해액이 2021년에는 세계적으로 수 조 달러에 이를 것으로 예상되고 있다[1].

기존의 Antivirus 소프트웨어들은 대개 이들 악성 코드를 분석하여, 각 코드를 식별할 수 있는 특징적인 바이트 시퀀스, 즉 시그니쳐(Signature)를 활용하고 있다. 그러나 이와 같은 방법은 특징 분석에 앞선 초기 공격이나 실시간 공격에 대응이 어렵고, 악성 코드 생성 툴킷 [2]을 이용해 손쉬운 변형 코드 생성 등이 가능해, 기존의 수작업에 의존하는 분석 대응방법은 현재와 같이 급증하는 악성 코드에 효과적으로 대처하기 어렵다.

한편, 최근 딥러닝 기술은 악성 코드 분석과 탐지 연구에도 매우 적극적으로 활용되고 있다. 딥러닝을 통한 학습의 첫 단계는 악성 코드를 분석하여 특징 정보를 추출하고 이를 축약 표현하는 과정이다. 특히, 코드 바이너리에 대한 정적(Static) 분석 중심이던 기존 기술과 달리, 악성 코드가 실행 시에 호출하는 API 또는 시스템 콜(System call)들의 패턴을 관찰하여, 그 특징을 추출하는 동적(Dynamic) 분석이 활발히 연구되고 있다[3]-[5]. 또한, 코드 영역에 대한 어셈블리 정보를 바이트 수준 n-gram으로 표현해 다양한 악성 징후를 탐지하기도 하고[6], 특정 opcode 또는 시스템 콜들이 유사 악성 코드 군에서 나타나는 빈도 등을 악성 코드 분류에 활용하기도 한다[7]. 또한, 매우 큰 특징 벡터의 크기를 효과적으로 줄이고 모델링을 단순화하기 위해, 데이터 출현 빈도나 상호 연관성을 기반으로 특징을 추출하고, 이를 auto-encoder를 사용하여 특정 정보를 축약하기도 한다[8][9].

다음 단계는, 추출된 코드의 특징 정보(벡터)에 대해 악성 코드 여부를 판정하는 분류 모델(Classification model)을 설계하는 과정이 필요하다. 이 과정에서는 베이즈 네트워크, decision tree, KNN (K-Nearest Neighbor), 다중 퍼셉트론, HMM(Hidden Markov Model), SVM(Support Vector Machine) 등의 기계 학습방법[10][11]은 물론, CNN(Convolution Neural Network), RNN(Recurrent Neural Network) 등의 DNN (Deep Neural Network) 기반 딥러닝 기법이 활용되고 있다[12].

본 논문에서는 리눅스 시스템에서 실행되는 악성 코드의 파일 접근 행태를 딥러닝 네트워크를 통해 학습시키고, 실제 악성 코드가 실행될 때 이를 실시간으로 탐지하는 방법을 소개한다. 일반적으로 이와 같은 응용에는 RNN이 사용되는데, RNN은 인접한 은닉 계층들이 순환적으로 연결되게 함으로써, 순차적 특징을 정보의 학습에 유용하다. 그러나 RNN에서 일련의 순차 정보를 지닌 입력 벡터의 길이가 매우 긴 경우, 멀리 떨어진 정보 간 연결성이 약해지는 문제가 발생하므로, 이를 보완한 RNN 모델인 LSTM(Long Short-Term Memory) 네트워크를 사용하였다.

또한, 악성 코드의 파일 접근 특징을 실행 환경에서 관찰하기 위해 가상 머신을 사용한 독립된 시스템 환경을 구축하고, 운영체제 커널의 수정없이 파일 접근 행태를 탐지하기 위해 사용자 공간 파일 시스템인 FUSE(File system in USErspace)를 사용하여, 어플리케이션들의 모든 파일 접근 시스템 콜을 사용자 수준에서 수집할 수 있도록 하였다.

본 논문은 2장에서 관련 연구와 요소 기술을 간략히 기술하고, 3장에서 제안 시스템을 소개한다. 그리고 4장에서 실험 결과와 성능을 비교하고, 5장에서 결론을 맺는다.

Ⅱ. 관련 연구

2.1 LSTM 네트워크

LSTM은 상당히 긴 입력 시퀀스를 효과적으로 학습하기 위해 제안되었으며, 내부적으로 셀(Cell)이라는 메모리 유닛을 통해 비교적 긴 시간 동안의 정보를 효과적으로 학습시킨다[13].

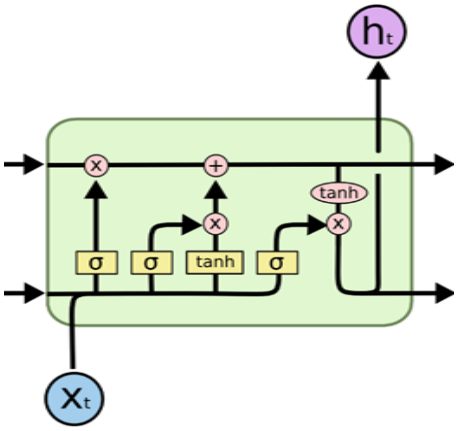
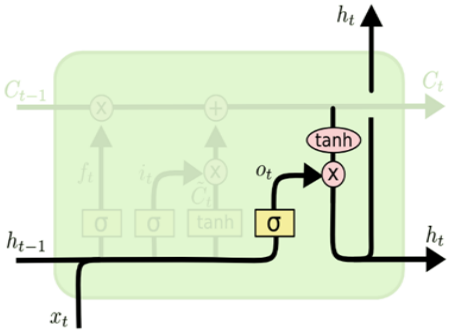


그림 1. LSTM의 기본 연결 형태
Fig. 1. Basic connection structure of LSTM



$$o_t = \sigma(W_o \lfloor h_{t-1}, x_t \rfloor + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

그림 2. LSTM의 출력 계층 계산 과정
Fig. 2. Output layer computation flow of LSTM

그림 1에 보이는 바와 같이, LSTM은 한 개의 셀 상태를 업데이트하기 위해 4개의 인접 계층과 상호작용한다. 또한, 각 계층은 3개의 입력, 즉 입력값 x_t, 출력값 h_{t-1}, 이전 사이클에서의 셀 상태 c_{t-1}를 받는다.

LSTM의 첫 단계는 이전 사이클의 정보 중 얼마를 셀에 저장할 것인지 결정하는 것이다. 이것은 fortet gate (f_t)라는 시그모이드 함수(σ)에 의해 결정되는데, [0, 1] 사이의 값을 출력하는 시그모이드 함수값과 c_{t-1}의 곱으로 다음과 같이 계산된다.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (1)$$

다음 단계는 어떤 새로운 정보를 셀에 저장할지를 결정하기 위해, 먼저 input gate (i_t)를 사용하여 어떤 값을 업데이트할지를 결정하고, 그 후 tanh 계층에서 해당 셀에 전달될 후보값($\widetilde{C}_t$)들의 벡터를 다음과 같이 생성한다.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2)$$
$$\widetilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

그리고 이전 상태 c_{t-1}는 forget gate f_t와 곱해진 후, 그 결과는 다시 $i_t * \widetilde{C}_t$에 더해져 셀에 저장될 새 후보값을 얻게 된다.

마지막으로, 그림 2에 보이는 것처럼 현재 사이클의 출력값 h_t가 결정되는데, 이 값은 2단계를 거쳐 필터링된다.

먼저 시그모이드 함수를 사용하여 그 셀 상태의 어떤 부분을 출력할지를 결정하고, 이 값을 tanh 함수를 통과한 셀 상태값과 곱하는데, 이 과정을 통해 앞서 결정된 일부분만을 출력하도록 할 수 있다. 이와 같은 네트워크 특성은, 언어 모델 또는 일련의 순차 데이터에서 선행 정보가 주어졌을 때, 그 뒤따라 나타날 정보들의 종류나 형태를 효과적으로 예측하게 한다. 결과적으로, LSTM의 내부 셀은 입력 데이터의 장기간 정보를 저장하는 기능을 지원하고, 이와 함께 입력 데이터의 관찰 구간을 계속 조절함으로써 입력 시퀀스의 히스토리를 유지하게 한다.

악성 코드 탐지의 경우, 어떤 프로세스에서 특정 시스템 콜이나 API가 발생하는 빈도 뿐만아니라, 이들이 나타나는 패턴이나 행태(경향)가 더 의미있다. 따라서, 이와 같은 함수 호출 패턴이 정상적인 범주의 어플리케이션에 해당되는지 확인하기 위해서 비교적 장기적인 함수 호출(또는 파일 접근) 패턴을 관찰해야 하는데, LSTM 네트워크는 이 목적에 적합하다.

2.2 악성 코드의 동적 특성 분석

악성 코드의 정적 분석과 달리, 동적 분석은 실제 악성 코드가 실행될 때 나타나는 파일, 메모리, 네트워크, I/O 장치 등에 대한 접근 행태를 관찰하는 것이다. 따라서, 별도의 차단된 실행 환경이 확보되어야 한다. 기존에는 샌드박스(Sandbox)를 이용한 동적 분석이 있었지만, 최근의 악성 코드들이 샌

드박스 등의 존재나 실행 환경을 알아채고 이를 회피하는 기능을 갖추고 있어, 본 연구에서는 가상 머신을 활용하였다. 생성된 가상 머신에는 리눅스 운영체제와 FUSE 파일 시스템을 설치하여 악성 코드 실행 환경을 구축하였다. 악성 코드가 FUSE를 통해 특정 파일 시스템(또는 디렉토리, 파일 등)에 접근하는 모든 접근 행태를 관찰하고 이를 기록할 수 있으며, 특히 동적 분석은 악성 코드 자체가 변형을 일으켜 정적 분석이 어려운 경우에도 이를 탐지할 수 있는 장점이 있다.

악성 코드의 동적 분석 및 탐지를 위해 통계적 모델링에 기반한 전통적인 기계 학습 기법이 많이 활용되었다. 이벤트 패턴 데이터의 순차적 발생과 상태 간 전이확률을 학습시킨 은닉 마코프 모델(Hidden Markov model)은 악성 코드의 주요 패턴에 대한 탐지율은 높으나, 학습에 필요한 데이터의 양이 매우 많아야 하고, 악성코드의 패턴 교란에는 대처하기 어렵다[14]. 또한, 커널 상에서 악성 코드의 자원 사용 통계량과 특징 정보를 분석하여, 각 클래스를 분류하는 최적화된 초평면(Hyperplane)을 찾아내는 SVM을 활용한 연구도 있지만, 학습에 필요한 특징 정보에 종류에 따라 학습 효과와 탐지 능력이 큰 편차를 보이는 문제를 안고 있다[15]. 이와 같은 행태 분석과는 달리, 통계적 모델에 정적 코드 분석 데이터를 통계적 모델에 사용하기도 하였다[16]. 그러나 이와 같은 연구들의 문제는 기계 학습 기법의 특성 상, 매우 제한적인 조건들을 전제한다는 것이다. 예를 들어, 은닉 마코프 모델의 경우에는 memoryless 성질을 가정하거나, 특정 커널 내에서 프로세스(또는 코드) 간에 유사도를 결정하는 척도를 미리 정의함으로 변형된 악성 코드에 대한 대응이 어렵다는 문제를 갖는다.

III. 시스템 구성 및 학습 방법

3.1 사용자 공간 파일시스템 (FUSE)

FUSE는 일반 사용자들이 커널 수정없이 특정 목적을 위한 파일 시스템을 구현할 수 있도록 지원하는 사용자 수준의 파일 시스템 프레임워크이다[17]. 원래 Unix 계열에서만 지원되었지만, 현재는 MS Windows를 포함하는 대부분의 운영체제에서 사용 가능하다.

새로운 파일시스템을 구현하기 위해, 사용자는 libfuse 라이브러리에 링크될 핸들러 프로그램을 작성해야 한다. 이 프로그램은 운영체제 커널 파일시스템(리눅스의 경우 VFS)으로 요청된 파일 시스템 콜(open/read/write/close/stat/lseek 등)을 FUSE를 통해 전달받아 이를 대신 처리하고 응답하도록 설계되어 있다.

그림 3의 예를 들어 설명하면, 사용자가 ls -l /tmp/fuse 명령을 통해 커널에 파일 /tmp/fuse에 대한 접근 요청을 하면, VFS는 FUSE의 libfuse에 등록된 사용자 핸들러(여기서는 ./hello)를 동작시킨다. 이 때, 이 핸들러에게 VFS에 요청된 파일명(즉, /tmp/fuse)이 전달되고, 핸들러는 이를 처리한 결과를 다시 FUSE를 통해 커널에 전달하는 과정을 거쳐 사용자 파일 시스템의 특수 기능을 수행하도록 한다. 따라서 사용자 어플리케이션이 FUSE 파일 시스템에 접근하며 호출하는 모든 시스템 콜은 이미 등록된 핸들러에 의해 캡쳐되고, 핸들러는 이 프로세스와 관련된 모든 로그를 저장할 수 있다.

중요 파일 데이터의 보호를 위해 파일 접근 과정을 효과적으로 관찰해야 하는데, 파일 시스템의 모든 파일에 대해 파일 접근을 모니터링하는 것은 파일 시스템의 성능을 전체적으로 저하시키는 문제가 있다.

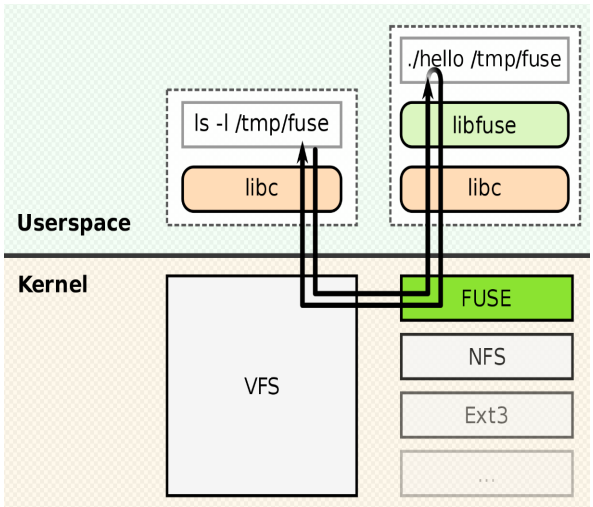


그림 3. 커널과 FUSE 간 파일 시스템 콜 동작
Fig. 3. File system call operation between Kernel and FUSE

따라서 본 연구에서는 중요 파일에 대한 악성 코드의 접근을 탐지하기 위해, FUSE를 사용하여 해당 파일 시스템을 분리 구현하고, 여기에 파일 시스템 콜 행태의 관찰 기능과 악성 코드 판단 기능을 부가하였다. 또한, 앞서 말한대로, FUSE는 사용자 수준의 파일 시스템이므로 본 연구의 결과물을 다양한 컴퓨터시스템에 쉽게 이식할 수 있다는 장점을 갖는다.

본 연구에서는 VMWare Workstation Player 가상 머신 위에 Ubuntu 18.04 리눅스를 설치하고, FUSE 파일 시스템을 구성하였다. FUSE 파일 시스템에서 제공되는 핸들러는 리눅스 VFS에서 처리하는 대부분의 파일 시스템 콜을 관찰하도록 작성되었다. 일반 파일, 디렉토리, 심볼릭 링크를 다루는 54개의 함수를 지원하며 전체 리스트는 표 1과 같다.

표 1. FUSE 지원 핸들러 함수 리스트
Table 1. List of Handler Functions supported by FUSE

```
mount()unmount() sysfs()stafs() fstatfs()
ustat() chroot()chdir() fchdir() getcwd()
mkdir() rmdir() getdents() readdir()
link() unlink() rename() readlink()
symlink() chown() fchown() lchown()
chmod() fchmod() utim() stat() fstat()
lstat() access() open() close() creat()
umask() dup() dup2() fcnt() select()
poll() truncate() ftruncate() lseek()
read() write() readv() writev()
pread() pwrite() nmap() munmap()
fdatasync() fsync() sync() msync() flock()
```

3.2 시스템 콜 패턴 추출

가상 머신 내에서 악성 코드가 실행되면, 모든 파일 접근 시스템 콜이 FUSE에 의해 수집되고, FUSE는 해당 프로세스 이름, PID, 파일 접근 시각, 파일 경로명, 시스템 콜, 전달 인자 등의 로그를 저장한다. 각 어플리케이션 실행 중에 각 5분 단위로 이벤트 시퀀스를 추출하였으며, 본 연구에서 사용된 데이터 셋은 Maltrieve[18], Virus Share[19]에서 제공되는 악성 코드 샘플 중 5종을 사용하였고, 양성(Benign) 코드로는 주로 파일 접근을 활발히 하는 오픈오피스, 크롬 웹라우저, VLC 미디어 플레이어를 사용하였다.

3.3 LSTM 네트워크 학습

양성 또는 악성코드를 각각을 30분간 수행시켜 수집된 파일 접근 로그에 나타나는 각 시스템 콜들에 대해, 그림 4에 보이는 바와 같이 데이터 전처리 과정을 거쳐 대응된 콜 번호(정수)로 인코딩하고, 이들을 각 50개의 길이로 분할하여 입력 벡터를 구성한다. 각 입력 벡터에는 또한 양성 또는 악성의 레이블을 붙이고, 이와 같이 구성된 입력 데이터의 일부는 학습용으로, 나머지는 테스트용으로 사용한다. 현재 LSTM 입력 계층의 벡터 크기는 50개로 제한했지만, 더 긴 패턴의 연결 관계를 파악해야 하는 경우, RNN의 특성 상 이를 쉽게 확장할 수 있다.

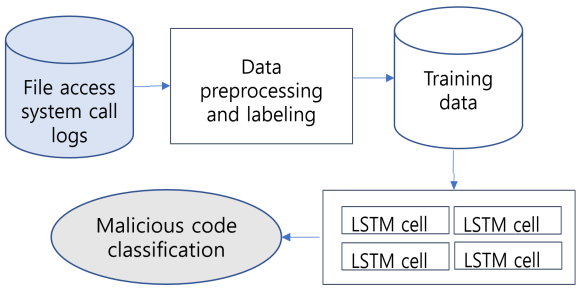


그림 4. LSTM 네트워크의 데이터 처리 과정
Fig. 4. Data processing flow for LSTM network

IV. 실험 및 결과

4.1 실험 방법

본 연구에서는 TensorFlow를 사용하여 50개의 LSTM 셀을 가진 3계층 LSTM 네트워크를 기본 네트워크로 구성하였고, 악성 및 양성의 이진 분류 문제이므로 시그모이드 활성 함수와 Adam optimizer를 사용하였다. 3계층은 입력, 학습, 출력 계층의 기본 구조이며, 50개의 LSTM 셀은 시스템 콜의 순차성을 학습시키는 기본 윈도우 크기로 설정하였다. 이 값은 어플리케이션의 실행 지역성과 관련이 있으며, 예비 실험에서 이 값이 충분히 큰 것으로 확인하였다.

실험에 사용된 입력 데이터는 악성 패턴 3,500개, 양성 패턴 1,000개로 구성되었으며, 각각의 2/3는 학습용, 나머지는 테스트(검증)용으로 사용되었다.

또한, 학습 시 미니 배치의 크기는 64였고, 500 epoch를 반복하였다. 실험에 사용된 연산 장치는 32GB 메모리를 장착한 NVidia Quadro P5000이었으며, 학습 시간은 6시간이 소요되었다.

성능 척도는 다음과 같이 accuracy(ACC), precision(PR), recall(RC)를 측정하였다.

$$accuracy\,(\%) = \frac{TP+TN}{TP+TN+FP+FN} \times 100 \quad (3)$$

$$precision = \frac{TP}{TP+FP}$$

$$recall = \frac{TP}{TP+FN}$$

TP, TN, FP, FN은 다음과 같이 정의된다.

- TP(true positive): 악성 코드를 악성 코드라고 판정
- FP(false positive): 양성 코드를 악성 코드라고 판정
- TN(true negative): 양성 코드를 양성 코드라고 판정
- FN(false negative): 악성 코드를 양성 코드라고 판정

4.2 실험 결과

본 논문에서 제안하는 LSTM 네트워크와 선행 연구에서 사용한 은닉 마코프 모델(HMM), Support Vector Machine(SVM) 등 2개의 기계 학습 모델과 성능을 비교하였다. HMM은 53개의 입력 심볼과 50개의 관찰 상태를 지원하고 Baum-Welch 알고리즘을 사용하여 확률을 갱신하고, SVM은 50개의 시스템 콜 이벤트를 입력 값(벡터)으로 받으며 양성/악성 입력에 대해 지도 학습(supervised learning)을 수행한다.

표 2에 보이는 바와 같이, LSTM이 기존 두 기계 학습모델에 비해 우수한 결과를 보였다. 특히, accuracy는 HMM에 비해서는 44.7%, SVM에 비해서는 3.8%의 성능 향상을 보였다. 이 결과에서, LSTM은 그 특성 상, 악성코드가 일으키는 상당히 긴 시스템 콜 패턴을 학습하는 능력이 우수함을 확인할 수 있었다. 이에 비해 HMM은 관찰 패턴의 길이가 길어질수록 시스템 콜(이벤트) 간의 연결성에 관한 탐지력이 낮아져, 본 연구에서와 같은 동적 실행 패턴 기반의 학습에 매우 취약함을 확인할 수 있었다. 한편, SVM은 순차성을 갖는 이벤트 패턴 데이터의 학습에 상당히 효과적임을 보였지만, 관찰 패턴의 길이가 길어지면서 LSTM 네트워크보다 성능이 떨어지는 결과를 보였다.

표 2. 모델별 성능 비교
Table 2. Comparison with 3 models

	ACC(%)	PR	RC
LSTM	92.2	0.956	0.904
HMM	47.5	0.716	0.476
SVM	87.4	0.842	0.874

다음은 LSTM 네트워크를 구성은 계층의 수가 성능에 미치는 영향을 살펴보기 위해, 표 3에 보이는 바와 같이 3개의 계층을 갖는 기본 네트워크(3L-LSTM)에 1개씩의 계층을 각각 추가한 4L-LSTM과 5L-LSTM을 구성하여 실험하였다. 4L-LSTM의 경우, 기본 네트워크보다 더 우수한 결과를 보였지만, 5L-LSTM에서는 오히려 성능이 저하되었는데, 이것은 계층의 증가가 오히려 overfitting으로 인한 오분류가 나타난 결과로 분석된다. 따라서 시스템 콜 패턴과 같이 비교적 표현이 단순한 구조에서는 네트워크 계층을 적정하게 유지하는 것이 바람직한 것으로 판단된다.

표 3. LSTM 네트워크의 계층 수에 따른 성능 비교
Table 3. Comparison with multi-layer LSTM networks

	ACC(%)	PR	RC
3L-LSTM	92.2	0.956	0.904
4L-LSTM	94.4	0.988	0.927
5L-LSTM	89.7	0.972	0.874

V. 결 론

본 논문에서는 딥러닝을 통해 다양한 악성 코드의 파일 접근 행태를 학습하고 이를 실시간으로 탐지하는 방법을 제안하였다. 기존의 정적 코드 분석과 달리, 본 연구에서는 독립된 가상 머신 상에서 실제 악성 코드를 동작시키고, 이 코드가 파일 시스템에 접근하는 시스템 콜 패턴과 경향을 수집하여 이를 학습에 활용하였다. 특히, 주요 파일 보호를 위해 FUSE 기반의 사용자 공간 파일 시스템을 이

용함으로써, 커널의 수정없이 파일 및 디렉토리에 대한 접근 정보를 수집할 수 있도록 설계하였다.

비교적 긴 입력 패턴의 순차성을 효과적으로 학습시키기 위해, 개선된 RNN 모델인 LSTM 네트워크를 사용하였다. 데이터 셋으로는, 기존 연구에서 활용된 Virus Share, Maltrieve 등의 악성 코드 샘플에서 5종을 선정하고, 이로 부터 3,500개의 악성 코드 패턴을 생성하여 실험에 사용하였다.

기존의 전통적 기계학습 모델인 은닉 마코프 모델과 SVM과 분류 성능을 비교하였는데, 92.2%의 accuracy를 나타내 기존 기계학습보다 3.8 ~ 44.7%의 향상을 보였다. 다만, 비교적 단순한 시스템 콜 패턴 학습에는 적정한 수의 LSTM 계층이 필요하고, 이를 초과할 경우 오히려 overfitting으로 인한 성능 저하가 나타남을 확인할 수 있었다.

본 연구 결과는 리눅스 뿐만아니라, FUSE를 지원하는 다양한 운영체제에서 활용될 수 있을 뿐만 아니라, 최근 심각한 피해 규모가 점점 증가하는 랜섬웨어 등의 다양한 악성 코드 탐지에도 적용할 수 있을 것으로 기대된다. 또한, 향후에는 파일 시스템 콜 뿐만아니라 메모리 접근, I/O 접근 등의 시스템 자원 사용 행태를 관찰하여, 악성 코드를 효과적으로 조기에 탐지하는 방법을 연구하고자 한다.

References

[1] Steve Morgan, et al., "Hackerpocalypse: A Cybercrime Revelation", Cybersecurity Ventures, Aug. 2016.

[2] Hamad Binsalleeh and T. Ormerod, et al., "On the analysis of the Zeus botnet crimeware toolkit", Proceedings of International Conference on Privacy Security and Trust, Ottawa, ON, Canada, pp. 31-38, Aug. 2010.

[3] Yanfang Ye and Tao Li, et al., "A survey on malware detection using data mining techniques", ACM Computing Surveys, Vol. 50, No. 3, Article 41, pp. 41:1~41:40, Jun. 2017.

[4] Wei Zhong and Feng Gu, "A multi-level deep learning system for malware detection", Expert Systems with Applications, Vol. 133, pp. 151-162, Nov. 2019.

[5] Fei Xiao, Zhaowen Lin, Yi Sun, and Yan Ma, "Malware Detection Based on Deep Learning of Behavior Graphs", Mathematical Problems in Engineering, Vol. 2019, pp. 1-10, Feb. 2019. https://doi.org/10.1155/2019/8195395.

[6] Richard Zak, Edward Raff, and Charles Nicholas, "What can n-grams learn for malware detection?", Proceedings of 12th International Conference on Malicious and Unwanted Software, IEEE, Fajardo, Puerto Rico, pp. 109-118, Oct. 2017.

[7] Yanfang Ye, Tao Li, Yong Chen, and, and Qingshan Jiang, "Automatic malware categorization using cluster ensemble", Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, Washington DC USA, pp. 95-104, Jul. 2010.

[8] Robert Moskovitch and Clint Feher, et al., "Unknown malcode detection using opcode representation", Intelligence and Security Informatics, Springer, pp. 204-215, 2008.

[9] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakulaet, "Autoencoder-based feature learning for cyber security applications", 2017 International Joint Conference on Neural Networks, Anchorage, AK, USA, pp. 3854-3861, May 2017.

[10] Ivan Firdausi, Charles lim, Alva Erwin, and Anto Satriyo Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection", In Proceedings of Second International Conference on Advances in Computing, Control and Telecommunication Technologies, Jakarta, Indonesia, pp. 201-203, Dec. 2010.

[11] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection", IET Information Sciences, Vol. 231, pp. 203-216,

Aug. 2013.
[12] William Hardy and Lingwei Chen, et al., "A deep learning framework for intelligent malware detection", In Proceedings of the International Conference on Data Mining, Las Vegas, Nevada, USA, pp. 61-67, Jul. 2016.
[13] Sepp Hocheriter and Jurgen Schmidhuber, "Long short-term memory", Neural Computation, Vol. 9, No. 8, pp. 1735-1780, Aug. 1997.
[14] S. Attaluri, S. McGhee, and M. Stamp, "Profile Hidden Markov Models and Metamorphic Virus Detection", Journal in computer virology, Vol. 5, No. 2, pp. 151-169, May 2009.
[15] J. Pfoh, C. Schneider, and C. Eckert, "Leveraging String Kernels for Malware Detection", Network and System Security, pp. 206-209, 2013.
[16] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data Mining Methods for Detection of New Malicious Executables", Proceedings 2001 IEEE Symposium on Security and Privacy, Oakland, CA, USA, pp. 38-49, 2001.
[17] Linux Fuse Interface, https://github.com/libfuse/libfuse, [accessed: Dec. 3, 2019]
[18] K. Maxwell. Maltrieve. https://github.com/krmaxwell/maltrieve, [accessed: Nov. 12, 2019]
[19] J. M. Roberts. Virus Share. https://virusshare.com, [accessed: Nov. 5, 2019]

저자소개

이 윤 석 (Yunseok Rhee)

1988년 2월 : 서울대학교 계산통계학과 (이학사)
1999년 2월 : 한국과학기술원 전산학과 (공학박사)
1999년 3월 ~ 현재 : 한국외국어대학교 컴퓨터전자시스템공학부 교수
관심분야 : 분산시스템, 운영체제, 임베디드 시스템, 인터넷 서비스