Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.28, NO.5, Oct. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.5.1197

명령 실행 모니터링과 딥 러닝을 이용한 파워셸 기반 악성코드 탐지 방법

이 승 현,† 문 종 섭‡
고려대학교 정보보호대학원

PowerShell-based Malware Detection Method Using Command Execution Monitoring and Deep Learning

Seung-Hyeon Lee,† Jong-Sub Moon‡
Graduate School of Information Security, Korea University

요 약

파워셸은 닷넷 프레임워크를 기반에 둔, 커맨드 라인 셸이자 스크립트 언어로, 그 자체가 가진 다양한 기능 외에도 윈도우 운영체제 기본 탑재, 코드 은닉 및 지속의 수월함, 다양한 모의 침투 프레임워크 등 공격 도구로서 여러 이점을 가지고 있다. 이에 따라 파워셸을 이용하는 악성코드가 급증하고 있으나 기존의 악성코드 탐지 기법으로 대응하기에는 한계가 존재한다. 이에 본 논문에서는 파워셸에서 실행되는 명령들을 관찰할 수 있는 개선된 모니터링 기법과, Convolutional Neural Network(CNN)을 이용해 명령에서 특징을 추출하고 실행 순서에 따라 Recurrent Neural Network(RNN)에 전달하여 악성 여부를 판단하는 딥 러닝 기반의 분류 모델을 제안한다. 악성코드 공유 사이트에서 수집한 파워셸 기반 악성코드 1,916개와 난독화 탐지 연구에서 공개한 정상 스크립트 38,148개를 이용하여 제안한 모델을 5-fold 교차 검증으로 테스트한 결과, 약 97%의 True Positive Rate(TPR)와 1%의 False Positive Rate(FPR)로 모델이 악성코드를 효과적으로 탐지함을 보인다.

ABSTRACT

PowerShell is command line shell and scripting language, built on the .NET framework, and it has several advantages as an attack tool, including built-in support for Windows, easy code concealment and persistence, and various pen-test frameworks. Accordingly, malwares using PowerShell are increasing rapidly, however, there is a limit to cope with the conventional malware detection technique. In this paper, we propose an improved monitoring method to observe commands executed in the PowerShell and a deep learning based malware classification model that extract features from commands using Convolutional Neural Network(CNN) and send them to Recurrent Neural Network(RNN) according to the order of execution. As a result of testing the proposed model with 5-fold cross validation using 1,916 PowerShell-based malwares collected at malware sharing site and 38,148 benign scripts disclosed by an obfuscation detection study, it shows that the model effectively detects malwares with about 97% True Positive Rate(TPR) and 1% False Positive Rate(FPR).

Keywords: PowerShell, malware, execution monitoring, deep learning

Received(08. 16. 2018), Modified(09. 21. 2018), Accepted(09. 21. 2018)

† 주저자, sniper2801@korea.ac.kr
‡ 교신저자, jsmoon@korea.ac.kr(Corresponding author)

I. 서 론

마이크로소프트에서 개발한 파워셸은 닷넷 프레임워크(.NET Framework)를 기반에 둔, 작업 기반의 커맨드 라인 셸이자 스크립트 언어이다[20]. 파워셸은 기본적으로 'cmdlet(command-let)'이라 불리는 수백 가지의 명령어를 탑재하고 있는데 이들은 파이프라인을 통해 입력과 출력으로 닷넷 프레임워크 객체를 주고받을 수 있어 기존 명령 프롬프트에서 할 수 있던 것보다 훨씬 더 다양한 작업을 수행할 수 있다. 이뿐만 아니라 리플렉션(reflection)을 통해 닷넷 프레임워크 라이브러리나 윈도우 네이티브 라이브러리의 함수에 접근하는 것도 가능하다[11].

악성코드 제작자는 이러한 파워셸의 강력한 기능을 이용해 실행 파일 없이 스크립트만으로 대부분의 목적을 달성할 수 있고, 파워셸이 윈도우 7 이상의 운영체제에 기본적으로 설치가 되어있기 때문에[21] 윈도우 운영체제를 사용하는 수많은 사용자를 대상으로 공격하는 것이 가능하다. 또한, 스크립트는 문자열이기 때문에 레지스트리와 같은 곳에 기록하고 참조하거나 시작 프로그램, 작업 스케줄러를 이용해 명령 줄로 호출하는 등 코드를 은닉하고 지속성을 유지하기 위한 다양한 기법을 사용할 수 있고, 'PowerShell Empire', 'PS〉Attack' 등 파워셸을 이용한 모의침투 프레임워크는 파워셸을 악성 행위에 사용하는 것을 수월하게 한다.

위와 같은 파워셸의 공격 도구로서의 이점은 2014년 이후 'poweliks'나 'kovter'와 같은 악성코드에서 활용되기 시작하였으며, 2017년에는 파워셸을 이용한 악성코드가 2016년에 비해 432%가, 4분기 동안에만 267%나 증가하였다[15]. 이에 따라 파워셸을 이용한 악성코드를 탐지하기 위한 기법이 필요한 상황이지만, 기존의 악성코드 탐지 방법으로 대응하기에는 한계가 존재한다.

파워셸은 정상적인 시스템 프로그램이기에 원천적으로 실행을 차단할 수 없으며, 차단하더라도 다른 프로세스에서 파워셸 엔진을 불러와 명령을 실행하는 것이 가능하다. 따라서 파워셸 프로그램이 아닌 파워셸에서 실행되는 악성 명령을 차단해야 하며, 이를 위해 윈도우 10의 파워셸 버전 5에서는 Anti-Malware Scan Interface(AMSI), 모듈 로깅, 스크립트 블록 로깅과 같은 모니터링 기법을 제공하나 기능이 제한적이다. 게다가 유연한 파워셸 문법과 리플렉션, 와일드카드와 같은 기능들은 스크립트에 대한 다양한 방식의 난독화를 가능하게 하여[2] 기존 안티바이러스의 시그니쳐 기반 탐지를 더욱 어렵게 한다.

이에 본 논문에서는 파워셸에서 실행되는 다양한 종류의 명령의 실행을 더 자세하고 명확하게 관찰할 수 있는 개선된 모니터링 기법과, 모니터링된 일련의 명령들로부터 일반적인 특징을 추출하여 악성 여부를 판단할 수 있는 딥 러닝 기반의 분류 모델을 제안함으로써, 기존 파워셸 실행 모니터링 기법과 악성코드 탐지 기법의 한계를 극복하고, 파워셸을 이용하여 악성 행위를 수행하는 악성코드를 효과적으로 탐지하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 파워셸 기반 악성코드와 이에 대응하기 위한 실행 모니터링 및 탐지 기법, 그리고 본 논문에서 이용하는 딥 러닝 모델에 관련된 연구를 소개하고, 3장에서 본 논문에서 제안하는 모델의 구성을 기술한다. 4장에서 제안한 모델의 성능을 평가하기 위한 실험을 보이고, 5장에서 결론과 향후 연구 방향을 제시한다.

II. 관련 연구

2.1 파워셸 기반 악성코드

파워셸 기반의 악성코드는 주로 Symantec, Palo Alto Network 등 보안 업체들에 의해 분석되었다[6, 10, 16, 23].

파워셸은 특정 목표를 노리는 공격부터 일반적인 악성코드까지 광범위한 영역에서 사용되며, 주로 다운로더 등 악성 행위를 개시하는 단계나 침입한 호스트의 네트워크 내 다른 컴퓨터에 명령을 실행하는 부차적 감염 단계 등에 활용된다. 그리고 스크립트는 이메일, 오피스 문서 매크로, 익스플로잇 등 다양한 경로를 통해 실행될 수 있다[16]. 발견되는 악성 스크립트의 종류는 다운로더가 가장 많으며 이외 셸코드 인젝터, 원격 스크립트 실행기, 백도어 생성기 등이 존재한다[23].

2.2 파워셸 실행 모니터링

AMSI는 파워셸, JScript, VBScript 등의 실행을 모니터링할 수 있는 인터페이스다[19]. 이를 통해 파워셸 엔진에서 실행되는 스크립트를 실시간으로 확인할 수 있어, 난독화나 인코딩이 어느 정도 해

제된 상태의 스크립트 내용을 확인하는 것이 가능하다[13].

스크립트 블록 로깅은 파워셸 스크립트 처리의 기본 단위인 스크립트 블록(script block)을 기록하는 기능으로, AMSI와 유사하게 실행되는 스크립트 내용을 확인할 수 있다[22].

모듈 로깅은 파이프라인을 통해 실행되는 특정 모듈의 명령에 대해 기록하는 기능으로, 파워셸에서 실행되는 대부분의 cmdlet을 모니터링할 수 있다.

그러나 AMSI나 스크립트 블록 로깅이 제공하는 역난독화 효과는 제한적이며, 모듈 로깅은 COM 객체나 닷넷 프레임워크 객체의 메소드 호출을 기록하지 못한다는 한계가 있다. 그리고 리플렉션을 이용하여 비활성화하거나, 작업 경로에 가짜 AMSI DLL 파일을 위치시키는 등 AMSI를 우회할 수 있는 여러 가지 기법들이 알려져 있으며[8], 두 가지 로깅 기능 또한 리플렉션을 이용하여 비활성화하는 등의 우회 기법이 존재한다.

이에 Rousseau[11]는 닷넷 프레임워크 기반 프로그램의 함수를 후킹(hooking)하는 기법을 이용하여 기존에 윈도우에서 제공하는 모니터링 기법에 존재하는 한계를 극복할 수 있음을 시사하였고, Tanda[13]는 Rousseau가 제시한 기법 중에서 메모리에 로드된 기계어 코드를 수정하는 방법을 이용하여 AMSI를 통해 전송되는 스크립트 내용과, 문자열을 스크립트로 해석하여 실행하는 명령인 'Invoke-Expression' cmdlet에 입력된 문자열을 모니터링하는 것이 가능함을 보였다.

2.3 파워셸 기반 악성코드 탐지

Hendler 등[5]은 처음으로 파워셸 기반의 악성코드를 탐지하는 방법을 연구하였다. 이 연구에서는 파워셸이 실행되는 명령 줄 문자열을 입력으로 사용하였으며, 문자 자체를 특징으로 이용하는 CNN, RNN 모델과 자연어 처리 기법에서 사용하는 특징인 3-gram, Bag Of Words를 이용한 로지스틱 회귀 모델을 분류기로 제시하였다. 제시한 모델들을 학습시킨 결과 모든 모델이 준수한 성능을 제공하지만, CNN 모델의 일반화 성능이 다른 모델들보다 더 우수한 것을 확인하였다.

FireEye[18]에서도 기계 학습과 자연어 처리 기법을 이용한 탐지 모델을 제시하였다. 이 모델은 파워셸이 실행되는 명령 줄을 토큰들로 분리하고 의미적으로 같은 토큰으로 확장한 후 토큰들을 벡터화시킨다. 이 벡터를 입력으로 하여 심층 신경망, 커널 Support Vector Machine(SVM) 등의 알고리즘을 이용해서 이진 분류한다.

2.4 딥 러닝 모델

2.4.1 Convolutional Neural Network

CNN은 이미지나 텍스트 등의 입력을 필터를 통해 합성하여 지역적인 특징을 추출하는 인공 신경망으로, 크게 컨볼루션 망과 완전연결 망으로 이루어져 있다. CNN의 일반적 구조는 Fig. 1.과 같다[1].

컨볼루션 망은 여러 개의 컨볼루션 층과 풀링 층을 조합하여 구성한다.

컨볼루션 층은 이전 입력의 모든 채널의 값을 하나로 합성하는 여러 개의 필터가 존재하는 층이다. 필터 하나는 '가로 X 세로 X 이전 층의 채널 개수' 크기의 가중치 행렬이며, 필터가 위치한 곳의 입력값과 필터의 가중치를 곱한 값을 모두 더하고, 이 값을 Rectified Linear Unit(ReLU)[9]과 같은 활성화 함수에 넣어 하나의 합성 값을 생성한다.

필터를 입력값 행렬에서 가로와 세로로 일정한 간격(stride)으로 움직여가면서 합성 값을 추출하기 때문에 이 합성 값들은 '새로운 가로 X 새로운 세로' 크기의 행렬을 형성하는데 이를 '특징 맵(feature map)'이라고 한다. 이때 새로운 크기는 입력 행렬의 크기, 필터의 이동 간격, 필터의 크기, 패딩 여부 등에 좌우된다. 필터마다 행렬(채널)을 출력하므로 전체적으로 한 층에서 '새로운 가로 X 새로운 세로 X 필터 개수' 크기의 행렬을 출력한다.

풀링 층은 이전 계층의 채널마다 일정한 '가로 X 세로' 영역에서 평균값, 최댓값 등을 추출하여 '새로운 가로 X 새로운 세로 X 이전 층의 채널 개수' 크기의 행렬을 출력한다. 이는 '가로 X 세로' 영역에서 중요한 하나의 값만 남도록 만드는 것이기 때문에 이

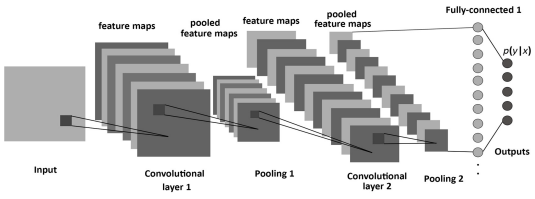

Fig. 1. Structure of CNN

전 층 특징 맵의 차원을 축소하는 역할을 하며, 과적합(overfitting)의 가능성을 낮추어 줄 수 있다.

여러 번의 컨볼루션 층과 풀링 층을 거쳐 최종적으로 만들어진 특징 맵은 완전연결 망의 입력으로 들어가게 되고, 완전연결 망에서는 분류와 같은 작업을 수행하게 된다.

2.4.2 Recurrent Neural Network

RNN은 순차 데이터 처리에 특화된 인공 신경망이다. 이 모델에서는 입력을 각각의 시간 단계(timestep)마다 받아들이고, 각 시간 단계의 은닉층에서는 현재 시간 단계의 입력과 함께 이전 시간 단계의 은닉층의 활성화 값을 모두 이용하여 현재의 활성화 값을 생성한다. 이는 현재 시점에서의 판단에 과거 시점의 판단을 고려함을 의미한다.

Long-Short Term Memory(LSTM)[4]는 RNN에서 사용될 수 있는 은닉층 활성화 유닛(셀)의 형태 중 하나로, 기존의 RNN을 학습시킬 때 시간 단계의 간격이 커질수록 오류에 의한 기울기 값이 잘 전달되지 않는 'vanishing gradient' 현상을 해결하기 위해 이용한다.

LSTM 셀을 사용하는 RNN의 구조는 Fig. 2.와 같다. $t \in (1,2,...,T-1,T)$는 시간 단계를 의미하며, x_t, y_t, h_t는 각각 t 시간 단계에서의 입력값, 출력층의 활성화 값, 은닉층의 활성화 값을, U, V, W는 각각 입력층과 은닉층을 연결하는 가중치, 은닉층과 출력층을 연결하는 가중치, 은닉층을 서로 연결하는 가중치를 의미한다. c_t는 t 시간 단계에서의 상태(state) 값으로, LSTM 셀에서 은닉층의 활성화 값의 연산에 사용하는 내부적인 값이며 일종의 장기 기억과 같은 역할을 한다.

현재 시간 단계에서의 상태 값과 활성화 값은 여러 게이트에 의해 결정된다. 게이트에는 입력 게이트, 출력 게이트, 그리고 망각 게이트가 있다. 입력 게이트는 입력을 얼마나 상태 값에 반영할지, 출력 게이트는 상태 값을 어느 정도까지 활성화 값으로 내보낼지, 망각 게이트는 과거의 상태 값을 얼마나 남길 것인지를 정한다. 각 게이트 값은 현재 시간 단계의 입력값과 이전 시간 단계의 은닉층 활성화 값에 의해 결정된다.

LSTM 셀을 사용하는 RNN에서 t 시간 단계에서의 출력값 y_t를 구하는 과정은 수식 (1)과 같다.

$$
\begin{aligned}
&a_t = \tanh(U^a x_t + W^a h_{t-1} + b^a) \\
&\forall G \in (i,f,o), G_t = \sigma(U^G x_t + W^G h_{t-1} + b^G) \\
&c_t = c_{t-1} \times f_t + a_t \times i_t \\
&h_t = \tanh(c_t) \times o_t \\
&y_t = z(Vh_t + c)
\end{aligned}
\tag{1}
$$

여기서, σ는 시그모이드(sigmoid) 함수이고, a_t, i_t, f_t, o_t는 각각 t 시간 단계에서의 은닉층으로 들어오는 입력값, 입력 게이트 값, 망각 게이트 값, 출력 게이트 값이다. 그리고 z, c는 출력층에서의 활성화 함수와 편향 값을 의미한다. 각 게이트를 계산하는 데 사용된 가중치 U, W 그리고 편향 값(bias) b를 구별하기 위해 위첨자 a, i, f, o를 붙였다.

III. 제안 방법

3.1 시스템 구성

본 논문에서 제안하는 시스템은 아래 Fig. 3.에서 볼 수 있듯이 크게 '명령 실행 모니터'와 '분류기'로 구성되어 있다.

명령 실행 모니터에서는 파워셸 엔진에서 실행되는 명령과 인자들을 모니터링한다. 모니터링하는 명령의 종류는 다음과 같다.

- cmdlet 호출
- 일반 프로그램 호출
- COM(Component Object Model) 객체 메소드 호출
- 닷넷 프레임워크 객체 메소드 호출

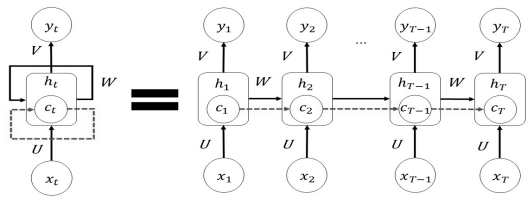

Fig. 2. Structure of RNN with LSTM Cell

Fig. 3. Structure of The Proposed Model

분류기에서는 모니터링한 일련의 명령과 인자들에서 CNN과 RNN을 이용해 특징을 추출하고 악성 여부를 판단한다.

기존 Hendler 등[5]과 FireEye[18]의 연구에서는 실행할 스크립트가 담긴 명령 줄의 문자열로부터 추출하는 정적인 특징을 탐지에 이용한다. 그러나 스크립트 문자열은 다양한 방식의 난독화를 통해 얼마든지 형태를 변경할 수 있으며, 탐지 모델에 학습되는 특징이 악성코드에 포함된 명령이나 인자들보다는 난독화 패턴이나 특정 변수 이름 등에 치우칠 수 있다. 그리고 개별적으로 보았을 때는 정상적으로 보이는 여러 명령들이 연달아 실행됨으로써 목표한 악성 행위를 달성하는 것도 가능할 것이다.

즉 단일한 스크립트나 명령 줄을 정적으로 관찰하는 것만으로는 파워셸에서 실제로 어떤 작업이 수행될지 파악하는 데에 한계가 존재한다는 것이다. 이에 본 논문에서는 명령 실행 모니터를 통해 파워셸 엔진에서 실제로 어떤 명령이 실행되는지를 스크립트의 다양한 형태 변형에 강건하게 동적으로 모니터링하고, CNN과 RNN을 이용하여 일련의 명령과 인자 그리고 이들의 순서에서 악성 행위의 고유한 패턴을 찾아 이를 탐지하고자 한다.

3.2 명령 실행 모니터

명령 실행 모니터는 파워셸 엔진 어셈블리인 'System.Management.Automation'의 여러 함수에 후킹하여 감시함으로써 cmdlet뿐만 아니라 일반 프로그램, 닷넷 프레임워크 객체 메소드 그리고 COM 객체 메소드가 호출되었을 때 명령 이름과 인자들을 추출하고, 이를 실행 순서대로 분류기에 전달한다. Table 1.에 윈도우 10의 파워셸 버전 5에서 명령 실행을 감시할 수 있는 지점들을 정리하였다.

명령의 경우 전달할 때 모듈 혹은 클래스 이름 일부와 결합하여 표현한다. 예를 들어 cmdlet의 경우 'Microsoft.PowerShell.Utility' 모듈에 속해 있는 'Start-Sleep'이 호출될 때, 모듈의 이름을 온점으로 구분하였을 때의 마지막 단어와 명령 이름을 결합하여 'Utility.Start-Sleep'으로 표현하며, 메소드의 경우 'System.Net.WebClient' 클래스 객체에 속해 있는 메소드인 'DownloadString'이 호출된 경우 'WebClient.DownloadString'으로 표현한다. 이를 통해 명령 이름과 함께 모듈이나 클래스 이름의 중요 부분을 간결한 형태로 포함할 수 있다.

Table 1. Inspection Points of PowerShell Command Invocation (System.Management.Automation.dll)

Invocation Type	Inspection Point Method (Class)
Cmdlet / Native Executable	DoExecute (CommandProcessorBase)
.NET Framework Object Method	InvokeMethod (PSInvokeMemberBinder)
COM Object Method	BindInvokeMember (IDispatchMetaObject)

'Write-Host' 등 콘솔 출력 관련 명령이나 'Import-Module' 등 모듈 관련 명령과 같이 실행되어도 시스템에 영향을 미치지 않으며 오히려 분류 모델에서 노이즈로 작용할 수 있는 명령들은 전달하지 않으며, 동일한 (명령 이름, 인자들) 쌍으로 이루어진 호출은 한 번만 전달한다.

인자는 문자열 타입만 전달한다. 다만 이들 중 3글자 이하로 이루어진 문자열들은 랜덤한 문자열이거나 일반 프로그램을 실행할 때 사용되는 기본적 옵션인 경우가 많아 무시한다.

3.3 분류기

분류기의 전체적인 구조는 Fig. 4.와 같다.

분류기에서는 모니터링을 통해 확인된 일련의 명령과 인자들로부터 CNN을 이용해 특징을 추출하고 이 특징들을 순서에 맞게 RNN에 전달하여 파워셸을 이용한 악성 행위가 발생하였는지를 판단한다.

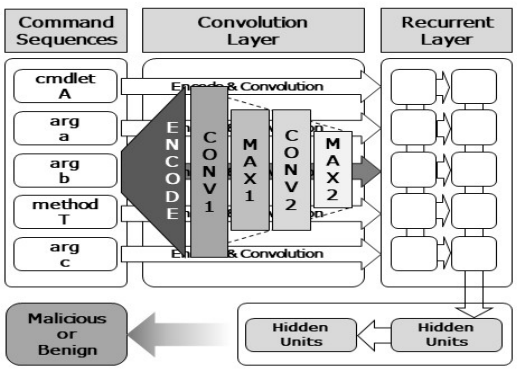

Fig. 4. Overall Structure of Classifier

3.3.1 CNN 입력

모니터링을 통해 확인한 명령과 인자들을 분류기에 입력하기 위해 Hendler 등[5]이 파워셸 명령줄을 표현할 때 사용한 것과 유사한 방식으로 문자들을 인코딩하여 행렬로 표현한다. 알파벳 소문자와 공백 그리고 다음의 문자들은 행렬에서 각각 하나의 열에 해당하며, 해당하는 문자의 열의 비트를 1로 설정한다.

```
- ‘ ! % & ( ) * , . / : ; ? @ [
\ ] _ ` { | } + 〈 = 〉 # $ ^ ~ ″
```

그리고 '0'~'9'까지의 숫자는 하나의 열에 대응되며, 알파벳 대문자의 경우 대응되는 알파벳 소문자와 함께 대문자를 나타내는 열의 비트를 1로 설정한다. 그리고 앞서 설명한 문자 중 어느 것에도 속하지 않는 문자들은 따로 하나의 열에 해당한다. 이에 따라 행렬의 열의 수는 62가 된다.

각각의 명령 또는 인자는 최대 64글자까지만 사용하며 이를 넘는 부분은 잘라낸다. 명령의 경우 명령 실행 모니터에서 언급한 표현 방식에 의해 64글자를 넘는 경우가 드물며, 인자의 경우 악성코드에서 모니터링된 전체 인자 중 길이가 약 5%만이 64글자를 넘는데 이러한 경우도 Base64 문자열이거나 'Invoke-Expression' 등 명령을 통해 실행되어 이후 단계의 모니터링에서 내용이 확인될 스크립트 문자열 등 일정 부분이 잘려도 큰 영향이 없는 것들이 대부분이기 때문이다.

문자열이 최대 64글자이므로 행렬의 행의 수는 64이고, 64글자보다 짧은 문자열의 경우 남은 부분을 널(null)로 패딩한다. 따라서 하나의 명령 또는 인자는 '64 X 62' 크기의 행렬로 표현된다.

3.3.2 CNN

컨볼루션 망은 컨볼루션 층(conv1) - 최댓값 풀링 층(max1) - 컨볼루션 층(conv2) - 최댓값 풀링 층(max2)으로 구성하였다. 컨볼루션 층의 활성화 함수는 ReLU를 사용하였으며, 모든 층에서 패딩(필터 일부가 입력 행렬을 벗어날 때 그 영역을 0으로 가정하고 계산)을 사용하였다. 컨볼루션 망의 구성은 Table 2.와 같다. 본 논문에서 사용한 입력은 2차원이 아닌 1차원(문자열) 형태이므로, 커널과 출력 행렬도 3차원이 아니라 2차원 형태이다.

Table 2. Structure of Convolution Layers

Layer	Kernel	Stride	Filter	Output
input	-	-	-	64 X 62
conv1	3 X 62	1	128	64 X 128
max1	2	2	-	32 X 128
conv2	4 X 128	1	64	32 X 64
max2	2	2	-	16 X 64

각각의 명령 혹은 인자의 문자열은 컨볼루션 망을 거쳐 '16 X 64' 크기의 행렬이 된다. 이를 펼쳐서 1,024차원의 벡터로 만들고 여기에 해당 문자열이 어떠한 명령 종류 혹은 인자인지 구분하는 4차원을 추가하여 1,028차원의 특징 벡터를 만든다.

3.3.3 RNN

CNN에서 생성한 하나의 명령 또는 인자의 특징 벡터를 하나의 시간 단계로 보고 차례대로 RNN 모델에 전달한다. 시간 단계를 배열할 때는 '명령 1 - 명령 1 인자 1 - … - 명령 1 마지막 인자 - 명령 2 - 명령 2 인자 1 - …' 과 같은 방식으로 배열한다. 분류기에 입력하는 명령의 개수는 최대 64개, 각 명령 당 인자의 개수는 최대 4개까지로 제한한다. 이는 모니터링 결과 99% 이상의 악성코드, 약 95%의 정상 스크립트에서 64개 이하의 명령이 실행되었으며, 여러 개의 옵션 인자가 전달되는 일부 일반 프로그램의 호출을 제외하고는 대부분 명령의 평균 인자 수가 4 이하인 점을 고려한 것이다.

RNN은 두 개의 은닉층을 가지며 각 은닉층에는 256개의 LSTM 셀이 사용된다.

최종적으로 악성 여부를 판단하기 위해, 마지막 시간 단계의 LSTM 셀 활성화 값들이 다층 퍼셉트론에 전달된다. 다층 퍼셉트론은 두 개의 은닉층을 가지며 각 은닉층은 512개의 노드를 가지고 있고 활성화 함수는 ReLU를 사용한다. 다층 퍼셉트론의 출력층에서 로지스틱 회귀를 수행하여 값이 0.5 이상이면 악성, 0.5 미만이면 정상으로 판단한다.

IV. 실험 및 평가

4.1 실험 데이터

실험을 위한 데이터는 파워셸 기반 악성코드의 경

우 'virusshare.com', 'malwares.com' 등 악성코드 공유 사이트에서 수집한 1,916개를 이용하였다. 여기에는 73개의 'kovter', 56개의 'poweliks', 53개의 'dridex' 변종, 13개의 'rozena' 등의 악성코드 종류가 포함되어 있으며, 이외에도 악성 행위를 수행할 응용 프로그램 혹은 파워셸 스크립트를 다운로드하여 실행하는 다운로더, 기계어 코드 혹은 닷넷 어셈블리 코드를 메모리에 로드하여 실행하는 인젝터, 공격자에게 시스템에 대한 접근 권한을 제공하는 백도어 등 일반적인 악성코드들이 포함되어 있다.

윈도우 파워셸의 기본 실행 정책에서는 스크립트 파일을 바로 실행할 수 없어 악성코드들은 주로 응용 프로그램, 바로 가기, 오피스 문서 등의 형식을 가지고 있으며, 이 파일들은 명령 줄이나 문서 매크로 등을 이용하여 파워셸을 실행한다. 어떠한 방식이든 결국 파워셸 엔진을 거쳐 명령이 실행되므로, 스크립트 파일과 다름없이 명령 실행 모니터를 통해 악성코드가 호출하는 일련의 명령들을 관찰할 수 있다.

정상 스크립트의 경우 Bohannon 등[3]의 파워셸 스크립트 난독화 탐지 연구에서 공개한 샘플 데이터[17]를 이용하였다. 이 샘플 데이터는 'Github', 'PoshCode', 'PowerShell Gallery', 'Microsoft Technet ScriptCenter', 'Github Gist'에서 수집된 파워셸 스크립트 파일과 모듈 파일로 이루어져 있으며, 이 중 옵션을 지정하지 않아도 실행이 가능한 파워셸 스크립트 파일 38,148개를 실험에 사용하였다.

분류기의 일반화 능력을 확인하기 위해 학습 및 검증 셋과 테스트 셋을 분리하였다. 악성코드의 경우 학습 및 검증 셋은 공유 사이트에 최초로 파일이 업로드된 날짜가 2018년 2월 28일 이전까지인 1,566개의 파일, 테스트 셋은 2018년 3월 1일부터 9월 7일까지인 350개의 파일로 구성하였으며, 정상 스크립트의 경우 테스트 셋으로 9,541개의 파일을 무작위로 선택하고 나머지 28,607개의 파일을 학습 및 검증 셋으로 사용하였다.

4.2 실험 환경 구성

모니터링을 위해 사용한 가상 머신의 환경과 분류기 학습 및 테스트를 위해 사용한 시스템의 환경을 Table 3.에 정리하였다.

명령 실행 모니터는 Tanda의 연구에서 공개한 소스 코드[24]에서 이용한 기법을 활용하였다. 닷넷 프레임워크에서 어셈블리 형태로 작성된 관리 코드(managed code)가 실행될 때 격리된 환경인 'AppDomain'이 생성되는데, 이 생성 과정에서 사용되는 기본 'AppDomainManager' 클래스 대신 자신이 구현한 커스텀 클래스를 불러와 생성 과정에 참여시키는 것이 가능하다. 커스텀 클래스를 이용해 현재 AppDomain에 파워셸 엔진 어셈블리가 로드되어 있는지 확인하고, 어셈블리에 포함된 특정한 함수(명령 실행 감시 지점)의 기계어 코드가 있는 메모리 주소를 찾아 함수를 후킹 할 수 있다.

Table 3. Experiment Environment Specification

Virtual Machine For Execution Monitoring	
OS	Windows 10 Pro N (Version 1709, Build 16299.125)
Powershell	5.1.16299.98
.NET Framework	4.7.1
Classifier Training & Testing System	
CPU	AMD Ryzen 7 1800X
RAM	64GB
GPU	NVIDIA GeForce GTX 1080 Ti

분류기는 파이썬 3.5.2와 'TensorFlow' 라이브러리(GPU용 1.8.0 버전)를 이용하여 구현하였다.

4.3 실험 결과

4.3.1 명령 실행 모니터

윈도우 10과 파워셸 버전 5가 설치된 가상 머신에서 각 샘플을 실행하고 일정 시간 동안 파워셸 엔진에서 실행되는 명령을 모니터링한다. 정상 스크립트의 경우 1분 동안 모니터링하며, 악성코드의 경우 파워셸이 주로 응용 프로그램이나 오피스 문서 매크로 등에 의해 이차적으로 실행되고 'Start-Sleep'과 같은 대기 명령어들이 종종 사용되므로 실행에 지연이 발생하는 점을 고려하여 3분 동안 모니터링한다.

Fig. 5.는 기계어 코드를 메모리에 불러와 실행하는 악성코드 샘플 중 하나의 스크립트 내용이다.

기존에 윈도우에서 제공하는 방법으로는 스크립트 문자열을 AMSI 혹은 스크립트 블록 로깅을 통해서 확인하거나, 타입 객체를 가져오는 'Add-Type'과 일정 시간 대기하는 'Start-Sleep'의 호출을 모듈 로깅을 통해서 확인할 수 있을 뿐이지만, 본 논문에

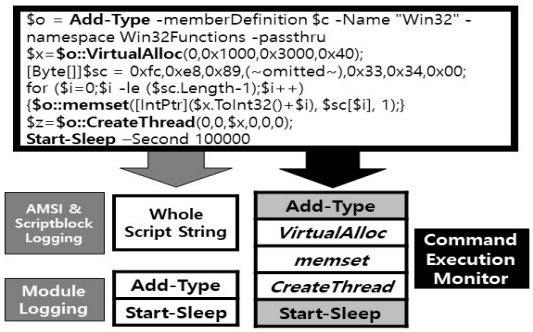

Fig. 5. An Example of Command Execution Monitoring

서 제안한 명령 실행 모니터를 통해서는 윈도우 네이티브 라이브러리 타입 객체를 통해 참조된 함수인 'VirtualAlloc', 'memset', 'CreateThread'를 호출하는 것까지 관찰할 수 있어 스크립트의 행위를 더 명확하게 파악하는 것이 가능하였다.

4.3.2 분류기 학습

분류기의 학습과 성능 측정은 5-fold 교차 검증 방식을 통해 이루어졌다. 학습은 확률적 경사 하강법(stochastic gradient descent)과 Adam 최적화 알고리즘[7]을 사용하였으며, 분류 비용 함수는 교차 엔트로피 함수를 사용하였다. 미니 배치 크기는 100이고 각 교차 검증마다 학습 셋을 10번 반복해서 학습하였다.

분류기가 지나치게 학습 셋의 특성에 적응하여 일반화 능력이 감소하는 과적합 현상을 막기 위하여 여러 기법을 사용하였다. 학습 과정에서 사용되는 손실 함수에 분류 비용 외에 분류기 가중치 값들의 L2 노름(norm)을 정규화 비용으로 포함해 가중치의 절댓값들이 지나치게 커지는 것을 막으며, 분류기의 다층 퍼셉트론에는 은닉층의 노드별로 0.5의 확률로 노드를 사용하지 않는 드롭아웃(dropout) 기법[12]을 적용하였다.

그리고 학습에 사용되는 악성코드와 정상 스크립트의 비율이 약 1 대 18로 정상 스크립트가 훨씬 많아, 분류기가 정상 스크립트를 정확히 판단하는 것에만 치우쳐 학습되는 것을 방지하기 위해 Tran 등[14]이 제시한 기법을 활용하였다. 이 연구에서는 도메인 생성 알고리즘 이용 봇넷을 탐지하기 위해 LSTM 모델을 이용하는데, 알고리즘에 의해 생성된 도메인의 집단별 샘플 수가 최소 25개에서 최대 42,166개에 이르는 등 불균형적이라는 문제가 있었다. 이로 인해 분류 모델이 비대칭적으로 학습되는 것을 막기 위해 i번째 집단의 분류에서 발생한 교차 엔트로피 비용에 $(n_i)^{-\gamma}$을 곱해주었다. 여기서 n_i는 i번째 집단의 샘플 수이며 $\gamma \in [0,1]$은 비례의 정도를 조절하는 값으로, 1에 가까워질수록 집단 간의 비율이 동일한 것과 같이 보정하는 효과가 강해진다. 이 기법을 사용함으로써 사용하지 않았을 때에 비해 모든 집단의 TPR의 평균을 최대 약 10%p까지 향상하는 것이 가능하였다.

본 논문에서는 교차 엔트로피로 분류 비용을 구할 때 정상 스크립트 집단의 분류에서 발생한 비용은 그대로 사용하고, 샘플 수가 적은 악성코드 집단의 분류에서 발생한 비용에는 수식 (2)를 곱하여 악성코드 집단 샘플의 영향력을 증폭한다.

$$\mu = (\frac{n_{norm}}{n_{mal}})^{\gamma}, \gamma > 0 \quad (2)$$

n_{norm}은 정상 스크립트 샘플 수, n_{mal}은 악성코드 샘플 수이며, γ는 다양한 값으로 실험한 결과 1.2일 때 가장 우수한 성능을 얻을 수 있었다.

4.3.3 분류기 성능 평가

분류기의 성능을 측정하기 위해 본 논문에서는 TPR, FPR, 그리고 Area Under Curve(AUC)를 사용하였으며, 5번의 교차 검증에서 구한 값들의 평균을 계산하여 분류기의 평균 성능을 측정한다.

TPR은 'recall'이라고도 불리며, 전체 악성코드 샘플 중에서 얼마나 많은 샘플을 악성으로 분류하였는지에 대한 비율이다. FPR은 전체 정상 스크립트 샘플 중에서 얼마나 많은 샘플을 악성으로 잘못 분류하였는지에 대한 비율이다. 그리고 Receiver Operating Characteristic(ROC)은 분류기의 분류 기준에 따라 달라지는 (FPR, TPR) 쌍들을 표현한 선이며, 이 선 아래 영역의 크기를 표현한 것이 AUC이다. TPR과 AUC는 클수록, FPR은 작을수록 분류 성능이 좋다.

Table 4.에서는 인자의 사용 여부와 입력받는 최대 명령의 수에 따른 성능의 변화를 표시하였다. 표에서 굵게 표시한 부분이 3장에서 제시한 기본 설정이다. 기본 설정에서 테스트 셋 기준 약 97%의

Table 4. Classifier Test on Various Parameters

ARG		X	O	O	O	O
LEN		**64**	16	32	**64**	128
AUC	V	0.9924	0.9963	0.9970	**0.9973**	0.9955
	T	0.9871	0.9919	0.9898	**0.9923**	0.9816
TPR	V	0.9821	0.9834	0.9827	**0.9866**	0.9834
	T	0.9726	0.9589	0.9634	**0.9737**	0.9691
FPR	V	0.0200	0.0102	0.0096	**0.0099**	0.0096
	T	0.0213	0.0096	0.0096	**0.0103**	0.0099

V : Validation Set / T : Test Set
ARG : Use of Arguments
LEN : Maximum Command Sequence Length

TPR과 1%의 FPR이라는 높은 분류 성능을 확인할 수 있다. 그리고 인자를 사용하지 않으면 FPR이 약 두 배가 증가하는 것을 확인할 수 있으며, 입력받는 최대 명령의 수가 커질수록 테스트 셋의 TPR이 향상되지만 필요 이상으로 커지면 성능이 소폭 저하되는 것을 확인할 수 있다.

Hendler 등[5]이 제시한 모델의 경우 FPR이 1%일 때 단일 분류기 중에서 가장 성능이 높은 4-레이어 CNN이 89%, 4-레이어 CNN과 3-gram 로지스틱 회귀 모델의 앙상블 모델이 92%의 TPR을 가진다. Hendler 등은 명령 줄 문자열을 입력으로 사용하였으나, 본 논문에서는 이러한 문자열이나 스크립트를 실행하여 추출한 일련의 명령 실행 순서들을 입력으로 사용하기에 직접적인 비교는 어렵지만, 파워셸 기반 악성코드를 탐지한다는 목적의 측면에서 본 논문에서 제시한 모델이 더 우수한 성능을 보인다고 할 수 있다.

V. 결 론

본 논문에서는 파워셸을 이용한 악성코드를 탐지하기 위해 파워셸의 명령 실행을 모니터링하는 개선된 기법과 모니터링된 명령으로부터 특징을 추출하여 악성 여부를 판단하는 딥 러닝 기반의 분류 모델을 제안하였다.

명령 실행 모니터는 cmdlet뿐만 아니라 일반 프로그램, 닷넷 프레임워크 객체 메소드 그리고 COM 객체 메소드의 호출에 대하여 명령과 인자들을 확인할 수 있게 해줌으로써 기존에 윈도우에서 제공하는 기능보다 더 많은 가시성을 제공해준다. CNN과 RNN이 결합된 딥 러닝 기반의 분류 모델은 명령과 인자들의 문자열 그리고 이들의 순서로부터 특징을 추출하여 악성 여부를 판단한다.

파워셸 기반의 악성코드와 정상 파워셸 스크립트를 대상으로 본 논문에서 제안한 모델의 성능을 실험한 결과 검증 셋과 테스트 셋 모두에 대하여 높은 분류 성능을 발휘함을 확인할 수 있었다.

본 논문과 관련하여 향후 명령 실행 모니터를 통해 실시간으로 파워셸 기반 악성코드를 탐지할 수 있는 효율적인 탐지 모델이나 명령의 실행 순서와 같은 동적 특징과 스크립트 문자열의 특성과 같은 정적 특징을 함께 사용하는 탐지 모델에 관한 연구가 가능할 것이다.

References

[1] S. Albelwi and A. Mahmood, "A framework for designing the architectures of deep convolutional neural networks," Entropy, vol. 19, no. 6, pp. 242-263, May 2017

[2] D. Bohannon, "Invoke-obfuscation: powershell obfusk8tion techniques & how to (try to) d"e`tec`t 'th'+'em," DerbyCon, Sep. 2016

[3] D. Bohannon and L. Holmes, "Revoke-obfuscation: powershell obfuscation detection and evasion using science," Blackhat USA, July 2017

[4] F.A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with LSTM," 9th International Conference on Artificial Neural Networks, pp. 850-855, Sep. 1999

[5] D. Hendler, S. Kels, and A. Rubin, "Detecting malicious powershell commands using deep neural networks," Proceedings of the 2018 on Asia Conference on Computer and Communications Security. ACM, pp. 187-197, June 2018

[6] R. Kazanciyan and M. Hastings, "Investigating powershell attacks," BlackHat USA, Aug. 2014

[7] D.P. Kingma and J.L. Ba, "Adam: a method for stochastic optimization," arXiv preprint arXiv:1412.6980v9, Jan.

2017
〔8〕 N. Mittal, "AMSI: how windows 10 plans to stop script-based attacks and how well it does it," Blackhat USA, Aug. 2016
〔9〕 V. Nair and G.E. Hinton, "Rectified linear units improve restricted boltzmann machines," Proceedings of the 27th international conference on machine learning, pp. 807-814, June 2010
〔10〕 S.M. Pontiroli and F.R. Martinez, "The tao of .NET and powershell malware analysis," Virus Bulletin Conference, Sep. 2015
〔11〕 A. Rousseau, "Hijacking .NET to defend powershell," arXiv preprint arXiv:1709.07508, Sep. 2017
〔12〕 N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929-1958, June 2014
〔13〕 S. Tanda, "Powershell inside out: applied .NET hacking for enhanced visibility," Code Blue, Nov. 2017
〔14〕 D. Tran, H. Mac, V. Tong, H.A. Tran, and L.G. Nguyen, "A LSTM based framework for handling multiclass imbalance in DGA botnet detection," Neurocomputing, vol. 275, pp. 2401-2413, Jan. 2018
〔15〕 McAfee, "McAfee labs threats report march 2018," McAfee, Mar. 2018
〔16〕 Symantec, "Increased use of powershell in attacks," Symantec, 2016
〔17〕 https://aka.ms/PowerShellCorpus
〔18〕 FireEye, "Malicious powershell detection via machine learning," https://www.fireeye.com/blog/threat-research/2018/07/malicious-powershell-detection-via-machine-learning.html
〔19〕 Microsoft, "Antimalware scan interface," https://docs.microsoft.com/en-us/windows/desktop/AMSI/antimalware-scan-interface-portal
〔20〕 Microsoft, "PowerShell," https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting
〔21〕 Microsoft, "Installing windows powershell," https://docs.microsoft.com/en-us/powershell/scripting/setup/installing-windows-powershell
〔22〕 Microsoft, "Script tracing and logging," https://docs.microsoft.com/en-us/powershell/wmf/5.0/audit_script
〔23〕 Palo Alto Networks, "Pulling back the curtains on encodedcommand powershell attacks," https://researchcenter.paloaltonetworks.com/2017/03/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/
〔24〕 GitHub, "DotNetHooking," https://github.com/tandasat/DotNetHooking

〈저자소개〉

이 승 현 (Seung-Hyeon Lee) 학생회원
2017년 2월: 경찰대학 법학과 학사
2017년 3월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 정보보호, 악성코드, 소프트웨어 역공학, 디지털 포렌식

문 종 섭 (Jong-Sub Moon) 종신회원
1981년 2월: 서울대학교 계산통계학과 학사
1983년 2월: 서울대학교 계산통계학과 석사
1991년 2월: Illinois Institute of Technology 전산학과 박사
1993년 3월~현재: 고려대학교 전자 및 정보공학부 교수
2001년 2월~현재: 고려대학교 정보보호대학원 겸임교수
〈관심분야〉 정보보호, 운영체제, 침입탐지