**딥러닝 알고리즘을 이용한**

**악성코드 탐지 시스템 구축**

**목 차**

[**1.** **프로젝트 서론** 7](#_Toc463421703)

[**1.1** **프로젝트 소개** 7](#_Toc463421704)

[**1.2** **프로젝트 목적** 8](#_Toc463421705)

[**1.3** **프로젝트 팀 구성** 8](#_Toc463421706)

[**1.4** **프로젝트 동기** 8](#_Toc463421707)

[**1.5** **프로젝트 수행 일정** 9](#_Toc463421708)

[**1.6** **프로젝트 개발 아이디어** 9](#_Toc463421709)

[**1.7** **분석 및 개발 도구** 10](#_Toc463421710)

[**2.** **딥러닝 학습을 위한 악성코드 정적 분석** 10](#_Toc463421711)

[**2.1** **악성코드** 11](#_Toc463421712)

[**2.2** **악성코드 정적 분석 지표** 11](#_Toc463421713)

[**2.2.1** **PE 포맷 분석** 11](#_Toc463421714)

[**2.2.1.1** **Dos Header** 12](#_Toc463421715)

[**2.2.1.2** **Characteristics** 12](#_Toc463421716)

[**2.2.1.3** **Compile Time** 13](#_Toc463421717)

[**2.2.1.4** **Import Library** 14](#_Toc463421718)

[**2.2.2** **IP, URL String** 15](#_Toc463421719)

[**2.2.3** **Packing** 16](#_Toc463421720)

[**2.2.4** **XOR** 19](#_Toc463421721)

[**2.2.5** **VirusTotal / White List** 20](#_Toc463421722)

[**2.3** **정적 분석 스크립트 개발** 22](#_Toc463421723)

[**2.3.1** **정적 분석 스크립트 및 결과 값** 22](#_Toc463421724)

[**2.3.1.1** **Dos Header / Characteristics / Compile Time** 22](#_Toc463421725)

[**2.3.1.2** **Raw Data Size / Section Entropy** 23](#_Toc463421726)

[**2.3.1.3** **Section Name** 24](#_Toc463421727)

[**2.3.1.4** **Import Library** 26](#_Toc463421728)

[**2.3.1.5** **IP, URL String** 29](#_Toc463421729)

[**2.3.1.6** **Packing** 31](#_Toc463421730)

[**2.3.1.7** **XOR** 32](#_Toc463421731)

[**2.3.1.8** **VirusTotal** 33](#_Toc463421732)

[**2.3.2** **딥러닝 학습 데이터 추출 순서** 35](#_Toc463421733)

[**3.** **Tensorflow를 이용한 딥러닝 시스템 구축** 36](#_Toc463421734)

[**3.1** **Neural Network** 36](#_Toc463421735)

[**3.1.1** **Neural Network 개념** 36](#_Toc463421736)

[**3.1.2** **Neural Network 구성도** 36](#_Toc463421737)

[**3.2** **Tensorflow** 37](#_Toc463421738)

[**3.2.1** **Tensorflow 개념** 37](#_Toc463421739)

[**3.2.2** **Tensorflow 개발 구축 환경** 37](#_Toc463421740)

[**3.2.3** **Tensorflow 개발 구축 과정** 38](#_Toc463421741)

[**3.3** **정적 분석 데이터를 이용한 알고리즘 구성** 39](#_Toc463421742)

[**3.3.1** **알고리즘의 Layer 및 Node 개수 조절** 39](#_Toc463421743)

[**3.3.2** **Cost 값 최적화를 위한 Learning rate 값 조절** 41](#_Toc463421744)

[**3.3.3** **Overfitting 방지를 위한 Dropout 값 조절** 43](#_Toc463421745)

[**3.4** **정적 분석 데이터 학습 및 저장** 46](#_Toc463421746)

[**3.4.1** **정적 분석 데이터 학습시키기** 46](#_Toc463421747)

[**3.4.2** **학습된 Weight 값 저장 및 불러오기** 46](#_Toc463421748)

[**3.4.3** **학습 데이터 추가 기능** 47](#_Toc463421749)

[**3.5** **데이터 학습 후 White list 작성하기** 47](#_Toc463421750)

[**3.6** **딥러닝 구성도** 48](#_Toc463421751)

[**4.** **웹 기반 학습 및 분석 환경 구축** 49](#_Toc463421752)

[**4.1** **Apache2 + php5 환경 구축** 49](#_Toc463421753)

[**4.1.1** **Apache2 설치** 49](#_Toc463421754)

[**4.1.2** **php5 설치** 50](#_Toc463421755)

[**4.2** **웹 사이트 디자인 및 구축** 50](#_Toc463421756)

[**4.2.1** **웹 사이트 로고 디자인** 50](#_Toc463421757)

[**4.2.2** **웹 사이트 구성 제작** 51](#_Toc463421758)

[**4.2.3** **메인 페이지 구축** 52](#_Toc463421759)

[**4.2.4** **파일 업로드 페이지 구축** 54](#_Toc463421760)

[**5.** **악성코드 탐지 테스트** 57](#_Toc463421761)

[**5.1** **알고리즘 환경에 따른 테스트 결과** 57](#_Toc463421762)

[**5.2** **최종 테스트 결과 및 분석** 61](#_Toc463421763)

[**6.** **결론** 63](#_Toc463421764)

[**6.1** **성과** 63](#_Toc463421765)

[**6.2** **문제점 및 개선 방안** 63](#_Toc463421766)

[**6.2.1** **정적 분석** 63](#_Toc463421767)

[**6.2.2** **API 함수 기준** 64](#_Toc463421768)

[**6.2.3** **Deep Learning 환경 구축** 64](#_Toc463421769)

[**7.** **참고자료** 65](#_Toc463421770)

[**8.** **부록** 66](#_Toc463421771)

**그 림 목 차**

[[그림 1- 1] 프로젝트 팀 구성 8](#_Toc463421823)

[[그림 1- 2] 딥러닝 기술을 응용한 산업분야 분포도 9](#_Toc463421824)

[[그림 1- 3] 프로젝트 수행 일정 9](#_Toc463421825)

[[그림 2- 1] 파일 암호형 랜섬웨어 11](#_Toc463421842)

[[그림 2- 2] 악성코드 정적 분석 지표 11](#_Toc463421843)

[[그림 2- 3] Notepad Dos Header 12](#_Toc463421844)

[[그림 2- 4] Upack Notepad Dos Header 12](#_Toc463421845)

[[그림 2- 5] EXE 확장자 DLL의 Injection 13](#_Toc463421846)

[[그림 2- 6] NT\_HEADERS 구조체 / FILE\_HEADER 구조체 13](#_Toc463421847)

[[그림 2- 7] FILE\_HEADER Characteristics 13](#_Toc463421848)

[[그림 2- 8] Notepad의 Modification Time, Compile Time 14](#_Toc463421849)

[[그림 2- 9] FILE\_HEADER Compile Time 14](#_Toc463421850)

[[그림 2-10] IMAGE\_IMPORT\_DESCRIPTOR 구조체 15](#_Toc463421851)

[[그림 2-11] 악성코드에 주로 사용되는 함수 15](#_Toc463421852)

[[그림 2-12] 악성코드내 URL String이 포함된 샘플 16](#_Toc463421853)

[[그림 2-13] UPX 패커 실행 압축 16](#_Toc463421854)

[[그림 2-14] 일반적인 Section Name 17](#_Toc463421855)

[[그림 2-15] UPX\_Notepad [위] / Notepad [아래] 18](#_Toc463421856)

[[그림 2-16] 정보의 양에 따른 Entropy 18](#_Toc463421857)

[[그림 2-17] UPX Notepad Entropy 19](#_Toc463421858)

[[그림 2-18] XOR 예시 19](#_Toc463421859)

[[그림 2-19] Test 원본 (위) / Test XOR (아래) 20](#_Toc463421860)

[[그림 2-20] Viper XOR 탐지 20](#_Toc463421861)

[[그림 2-21] VirusTotal 검사 결과 21](#_Toc463421862)

[[그림 2-22] 신뢰할 수 있는 사이트 21](#_Toc463421863)

[[그림 2-23] Dos Header 변조 / DLL 여부 / Compile Time 변경 확인 함수 22](#_Toc463421864)

[[그림 2-24] Dos Header 변조 / DLL 여부 / Compile Time 변경 확인 결과 1 23](#_Toc463421865)

[[그림 2-25] Dos Header 변조 / DLL 여부 / Compile Time 변경 확인 결과 2 23](#_Toc463421866)

[[그림 2-26] Raw Data Size 값 0인지 여부 / Entropy 값 확인 함수 23](#_Toc463421867)

[[그림 2-27] Raw Data Size 값 0인지 여부 / Entropy 값 확인 결과 24](#_Toc463421868)

[[그림 2-28] 파일 섹션 이름을 저장하는 함수 24](#_Toc463421869)

[[그림 2-29] 파일 섹션 이름을 저장 결과 25](#_Toc463421870)

[[그림 2-30] Section Name 리스트 호출 함수 / Section Name 비교 함수 25](#_Toc463421871)

[[그림 2-31] DLL, API 추출 함수 26](#_Toc463421872)

[[그림 2-32] DLL, API 저장 함수 26](#_Toc463421873)

[[그림 2-33] DLL, API 저장 결과 27](#_Toc463421874)

[[그림 2-34] DLL, API 리스트 호출 함수 27](#_Toc463421875)

[[그림 2-35] DLL, API 리스트 비교 함수 28](#_Toc463421876)

[[그림 2-36] Hex Data String 변환 함수 29](#_Toc463421877)

[[그림 2-37] 출력 가능 문자 30](#_Toc463421878)

[[그림 2-38] IP 하드코딩 / URL 하드코딩 여부 확인 함수 30](#_Toc463421879)

[[그림 2-39] URL 하드코딩 출력 결과 31](#_Toc463421880)

[[그림 2-40] 패킹여부 확인 함수 31](#_Toc463421881)

[[그림 2-41] 패킹 확인 결과 31](#_Toc463421882)

[[그림 2-42] XOR 여부 확인 함수 32](#_Toc463421883)

[[그림 2-43] XOR 확인 결과 32](#_Toc463421884)

[[그림 2-44] Hash값 확인 함수 33](#_Toc463421885)

[[그림 2-45] VirusTotal 질의 결과 함수 33](#_Toc463421886)

[[그림 2-46] Hash 값 확인 / VirusTotal 질의 결과 출력 1 34](#_Toc463421887)

[[그림 2-47] Hash 값 확인 / VirusTotal 질의 결과 출력 2 34](#_Toc463421888)

[[그림 2-48] 학습 데이터 추출 순서도 35](#_Toc463421889)

[[그림 2-49] 학습 데이터 순서 분류 기준 35](#_Toc463421890)

[[그림 2-50] 학습 데이터 결과 값 35](#_Toc463421891)

[[그림 3- 1] 일반적인 Neural Network 알고리즘 36](#_Toc463421892)

[[그림 3- 2] Neural Network 알고리즘 내부 구성도 37](#_Toc463421893)

[[그림 3- 3] Tensorflow 개발 구축 환경 37](#_Toc463421894)

[[그림 3- 4] Tensorflow 개발 구축 과정 1 38](#_Toc463421895)

[[그림 3- 5] Tensorflow 개발 구축 과정 2 38](#_Toc463421896)

[[그림 3- 6] Tensorflow 개발 구축 과정 3 38](#_Toc463421897)

[[그림 3- 7] Tensorflow 개발 구축 과정 4 38](#_Toc463421898)

[[그림 3- 8] 정적 분석 데이터를 이용한 알고리즘 구성도 39](#_Toc463421899)

[[그림 3- 9] V data의 Layer 및 Node 개수 40](#_Toc463421900)

[[그림 3-10] X data의 Layer 및 Node 개수 40](#_Toc463421901)

[[그림 3-11] Overshooting 41](#_Toc463421902)

[[그림 3-12] Cost값 낮추기 42](#_Toc463421903)

[[그림 3-13] 학습빈도에 따른 Cost 값 변화 42](#_Toc463421904)

[[그림 3-14] 학습빈도에 따른 Accuracy값 변화 42](#_Toc463421905)

[[그림 3-15] Dropout 을 사용한 알고리즘 변화 43](#_Toc463421906)

[[그림 3-16] Dropout 여부에 따른 Hypothesis 그래프 변화 43](#_Toc463421907)

[[그림 3-17] Dropout 설정 44](#_Toc463421908)

[[그림 3-18] Dropout 없는 경우 데이터 학습율 44](#_Toc463421909)

[[그림 3-19] Dropout 없는 경우 데이터 테스트 결과 44](#_Toc463421910)

[[그림 3-20] Dropout 있는 경우 데이터 학습율 45](#_Toc463421911)

[[그림 3-21] Dropout 있는 경우 데이터 테스트 결과 45](#_Toc463421912)

[[그림 3-22] For문을 이용한 데이터 학습 46](#_Toc463421913)

[[그림 3-23] 학습된 Weight 값 저장 46](#_Toc463421914)

[[그림 3-24] 학습된 Weight 값 불러오기 46](#_Toc463421915)

[[그림 3-25] 데이터 추가 기능 47](#_Toc463421916)

[[그림 3-26] White List 작성 47](#_Toc463421917)

[[그림 3-27] 딥러닝 전체 구성도 48](#_Toc463421918)

[[그림 4- 1] Apache2 설치 49](#_Toc463421932)

[[그림 4- 2] Php5 설치 50](#_Toc463421933)

[[그림 4- 3] 웹 사이트 로고 디자인 50](#_Toc463421934)

[[그림 4- 4] 웹 사이트 구성도 51](#_Toc463421935)

[[그림 4- 5] 메인 페이지 구축 52](#_Toc463421936)

[[그림 4- 6] Logo.php 52](#_Toc463421937)

[[그림 4- 7] 메인 사이트 초기 화면 53](#_Toc463421938)

[[그림 4- 8] Browse 버튼 클릭 시 화면 53](#_Toc463421939)

[[그림 4- 9] 파일이 Upload 된 화면 53](#_Toc463421940)

[[그림 4-10] 파일 업로드 페이지 1 54](#_Toc463421941)

[[그림 4-11] 파일 업로드 페이지 2 55](#_Toc463421942)

[[그림 4-12] 파일 업로드 페이지 3 56](#_Toc463421943)

[[그림 4-13] Veritas 웹페이지 Upload 출력 결과 56](#_Toc463421944)

[[그림 5- 1] 시뮬레이션 조건 1 58](#_Toc463421982)

[[그림 5- 2] 시뮬레이션 조건 2 58](#_Toc463421983)

[[그림 5- 3] 시뮬레이션 조건 3 58](#_Toc463421984)

[[그림 5- 4] 시뮬레이션 조건 4 59](#_Toc463421985)

[[그림 5- 5] 시뮬레이션 조건 5 59](#_Toc463421986)

[[그림 5- 6] 시뮬레이션 조건 6 59](#_Toc463421987)

[[그림 5- 7] 시뮬레이션 조건 7 60](#_Toc463421988)

[[그림 5- 8] 시뮬레이션 조건 8 60](#_Toc463421989)

[[그림 5- 9] 최종 테스트 결과 61](#_Toc463421990)

[[그림 5-10] 조건 8의 학습 데이터 및 테스트 데이터 Cost 61](#_Toc463421991)

[[그림 5-11] 조건 8의 학습율 및 탐지율 62](#_Toc463421992)

1. **프로젝트 서론**
   1. **프로젝트 소개**

이전의 악성코드 탐지 기술은 악성코드의 특징을 비교하는 방식을 이용하였다. 그래서 최근 악성코드들은 이러한 특징을 변경하는 등 탐지를 우회하기 위해 다양한 기술을 사용하고 있다. 이러한 악성코드의 변화는 분석과 탐지를 어렵게 하고 있다.

최근 보안업계에 거론되고 있는 기술 중 하나는 딥러닝이다. 딥러닝은 Neural Network를 활용한 기계학습으로 학습 데이터를 통하여 특징 값을 추출하고 변화를 스스로 판단하여 새로운 데이터가 무엇인지 판단한다. 이러한 딥러닝의 특징은 변화하는 데이터의 분석에 적합할 것이다.

따라서 본 프로젝트는 딥러닝을 이용한 악성코드 탐지 기술을 연구함으로써, 시시각각 변화하는 악성코드를 탐지하는 딥러닝을 개발하고 그를 통해 악성코드에 대한 피해를 줄이고자 한다.

* 1. **프로젝트 목적**

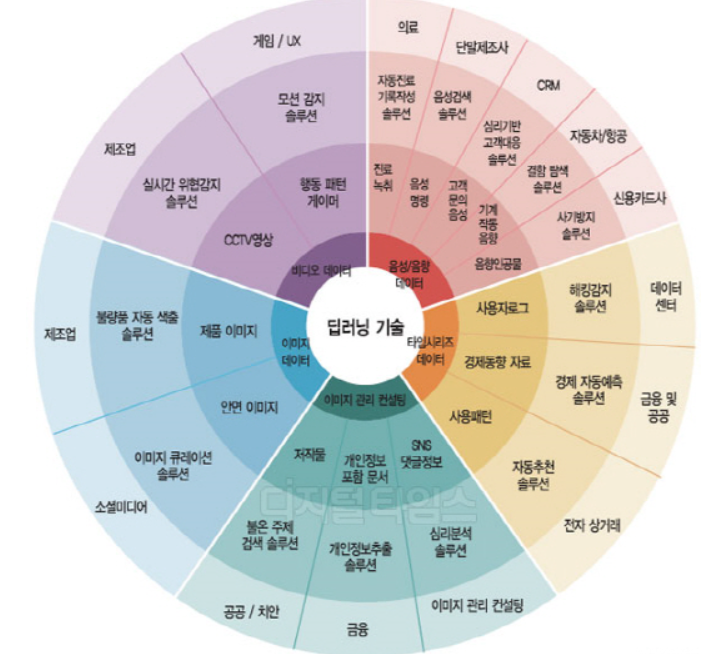
본 프로젝트는 딥러닝 기술을 활용하여 악성 코드를 탐지하는 시스템 구축을 목적으로 한다. 딥러닝 기술이 적용된 악성 코드를 판별하는 시스템을 구축하여 의심스러운 파일을 사전에 파악해서 피해 규모를 줄이고자 한다. 또한 악성 코드를 정적 분석하는데 있어 오픈소스를 활용함으로써 정적 분석 기법의 이해와 개발 능력을 키울 수 있으며, 딥러닝 기술 또한 이해와 개발 능력을 키우는데 목적을 둔다.

* 1. **프로젝트 팀 구성**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 성명 | 장해훈 | 정유성 | 김경덕 | 박다문 | 이하빈 |
| 직책 | PM | 팀원 | 팀원 | 팀원 | 팀원 |
| 담당업무 | Static Analysis | Static Analysis | Deep Learning | Deep Learning | API/White list  PHP |
| 업무수행도구 | Python 2.7 peframe  Viper | Python 2.7 peframe  Viper | Python 2.7  Ubuntu14.04  Tensorflow | Python 2.7  Ubuntu14.04  Tensorflow | Python 2.7  Ubuntu14.04  Apache2 |

[그림 1-1] 프로젝트 팀 구성

* 1. **프로젝트 동기**



[그림 1-2] 딥러닝 기술을 응용한 산업분야 분포도

현재 딥러닝 기술이 미래산업의 기대주로 관심이 많아짐에 따라 [그림 1–1]과 같이 다양한 산업분야에서 이를 응용한 기술들을 선보이고 있다. 이에 APT 보안 솔루션도 딥러닝 기술과 융합이 가능할지 의문이 들었고 개발이 가능하다면 어떤 방식의 기술로 융합이 될지 이를 분석하고, 탐구하고 싶은 열의에 본 프로젝트를 시작하게 되었다.

* 1. **프로젝트 수행 일정**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 일정  내용 | 수행기간 (2016.08.01 ~ 2016.09.25) | | | | | | | |
| 1주차 | 2주차 | 3주차 | 4주차 | 5주차 | 6주차 | 7주차 | 8주차 | |
| 자료조사 및 분석 |  |  |  |  |  |  |  |  | |
| 정적 분석 기법 숙지 |  |  |  |  |  |  |  |  | |
| 딥러닝 기법 숙지 |  |  |  |  |  |  |  |  | |
| 정적 분석 스크립트 개발 |  |  |  |  |  |  |  |  | |
| 딥러닝 학습 및 테스트 |  |  |  |  |  |  |  |  | |
| 소스코드 수정 및 보완 |  |  |  |  |  |  |  |  | |
| 보고서 작성 |  |  |  |  |  |  |  |  | |

[그림 1-3] 프로젝트 수행 일정

* 1. **프로젝트 개발 아이디어**

본 프로젝트는 정적 분석을 이용한 악성코드 탐지 딥러닝 시스템이다.

악성코드 탐지를 위한 정적 분석 기법에는 PE 포맷 분석, IP / URL 하드코딩 여부 확인, Packing 여부 확인, XOR 인코딩 여부 확인, VirusTotal 결과 비교 등이 있으며 이를 이용하여 파일의 악성행위 여부를 판별한다.

딥러닝은 수많은 데이터의 특징 값을 추출하고 그를 통해 학습하는 기능을 가지고 있다. 그래서 우리는 정적 분석 데이터를 이용한 딥러닝 보안 솔루션을 제공하고자 한다.

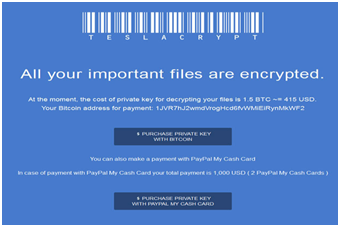
* 1. **분석 및 개발 도구**

악성코드 탐지를 위한 정적 분석에 있어 Python 2.7 버전과 peframe 오픈소스를 참조하여 PE 포맷 분석, IP / URL 하드코딩 여부, Packing 여부 등을 확인하는데 활용하였고, Viper 툴을 참조해 파일의 XOR 인코딩 여부를 확인하는 스크립트를 구성하였다. 그리고 딥러닝 개발을 위해 Ubuntu14.04 LTS 환경에서 Python 2.7 버전의 Tensorflow 소프트웨어 패키지를 사용하여 알고리즘을 구성하였다.

1. **딥러닝 학습을 위한 악성코드 정적 분석**

* 1. **악성코드**

**악성코드란 컴퓨터 바이러스, 웜, 스파이웨어, 트로이 목마 등 사용자, 컴퓨터, 네트워크에 해를 끼치는 모든 소프트웨어이다.**

****

[그림 2-1] 파일 암호형 랜섬웨어

**최근에는** 랜섬웨어**라는 악성코드가 유행하며 네트워크를 통하여 전파되는 등의 복합적인 악성코드들이 나타나고 있다. [그림 2-1]의 파일 암호형** 랜섬웨어**가 한 예이다.**

* 1. **악성코드 정적 분석 지표**

**악성코드 분석 시 먼저 정적 분석부터 진행한다. 정적 분석은 실제 프로그램을 실행시키지 않고, 프로그램의 기능을 파악하기 위해 코드나 프로그램의 구조를 분석하는 과정이다. 프로그램을 실행하지 않기에 정확한 목적을 알 수 없다. 그렇기 때문에 정적 분석을 통한 악성코드 판별은 가능한 많은 정보를 수집하여야 하고 다양한 기법을 사용해야 한다.**

|  |  |  |
| --- | --- | --- |
| **지표** | **용도** | **비고** |
| **Dos Header** | **Dos Header 영역 수정 여부 확인** | **Dos Header 영역에 악성행위 삽입** |
| **Characteristics** | **확장자를 수정한 DLL탐지** | **DLL의 확장자를 변경하여 탐지 우회** |
| **Compile Time** | **Compile Time 임의 수정 여부 확인** | **악성코드 분석과 사고대응을 어렵게함** |
| **Import Library** | **Import된 함수 추출** | **악성코드에서 주로 사용되는 함수 비교** |
| **IP/URL 하드코딩** | **IP/URL 하드코딩 탐지** | **해커의 서버 IP/URL 하드코딩** |
| **Packing** | **코드를 감추는 Packing 여부 확인** | **실행압축으로 탐지 우회** |
| **XOR** | **XOR 사용 여부 확인** | **XOR 인코딩을 통한 난독화** |

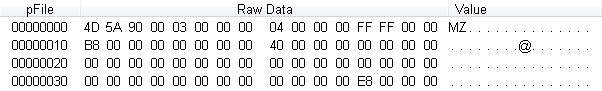
[그림 2-2] 악성코드 정적 분석 지표

* + 1. **PE 포맷 분석**

**PE 파일 포맷은 Windows에서 사용되는 exe, dll, sys 등의 실행코드를 관리할 수 있는 필수정보를 담은 데이터 구조이다. PE 파일은 파일 유형, 사용 함수, 파일 정보를 알 수 있는 매우 유용한 포맷이다. PE 파일 포맷은 매우 다양한 정보를 담고 있으나, 해당 프로젝트에서는 아래와 같은 정보를 이용한다.**

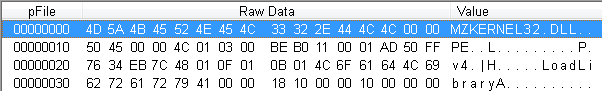
* + - 1. **Dos Header**

**Dos Header는 하위 호환성을 위하여 만든 것으로 총 64 Bytes의 구조체이다.**

****

[그림 2-3] Notepad Dos Header

**[그림 2-3]은 감염되지 않은 Notepad의 Dos Header이다. Dos Header는 별도의 수정을 하지 않는 한 변동되는 경우는 거의 없다.**

****

[그림 2-4] Upack Notepad Dos Header

**[그림 2-4]는 Upack packer를 이용한 Notepad이다. KERNEL32.DLL과 LoadLibraryA가 추가된 것을 확인 할 수 있다. 이와 같이 악성코드에서는 원하는 정보 또는 5A4D(MZ)대신 XOR 복호화 키를 설정하거나 악성코드 다운로드 명령들을 숨겨놓기도 한다. 이러한 Dos Header의 변경은 악성코드로 의심할 수 있다.**

* + - 1. **Characteristics**

**악성코드의 침투 기법 중 DLL Injection이 있다. 이는 프로그램에 해커가 원하는 DLL을 이식하는 방법이다. DLL은 확장자가 DLL이 아니어도 실행이 가능하다.**

C:\Users\wjddb\Desktop\project\dll_to_exe.png

**[그림 2-5]** **EXE 확장자 DLL의 Injection**

**[그림 2-5]와 같이 확장자가 DLL이 아닌 경우에도 DLL Injection이 성공하는 것을 확인할 수 있었다. 그러므로 확장자가 DLL이 아니지만 본래 DLL인 파일의 경우 악성코드로 의심할 수 있다. DLL의 여부는 NT\_HEADER 영역에서 확인이 가능하다.**

|  |  |
| --- | --- |
| **tyedef strct \_IMAGE\_NT\_HEADERS {**  **DWORD Signature;**  **IMAGE\_FILE\_HEADER FileHeader;**  **IMAGE\_OPTIONAL\_HEADER32 OptionalHeader;**  **} IMAGE\_NT\_HEADERS32, \*PIMAGE\_NT\_HEADERS32** | **typedef struct \_IMAGE\_FILE\_HEADER {**  **WORD Machine;**  **WORD NumberOfSections;**  **DWORD TimeDateStamp;**  **DWORD PointerToSymbolTable;**  **DWORD NumberOfSymbols;**  **WORD SizeOfOptionalHeader;**  **WORD Characteristics;**  **} IMAGE\_FILE\_HEADER, \*PIMAGE\_FILE\_HEADER** |

[그림 2-6] ****NT\_HEADERS 구조체 / FILE\_HEADER 구조체****

**확장자가 아닌 PE를 통한 DLL 확인방법은 [그림 2-6] NT\_HEADERS 구조체의 FILE\_HEADER 구조체 안의 Characteristics의 정보를 통하여 파일의 형식을 알 수 있다. DLL의 경우 0x2000 bit의 플래그가 세팅된다.**

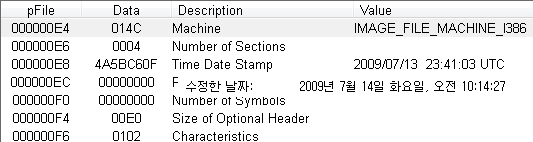
**nt_header_file_header**

**[그림 2-7] FILE\_HEADER Characteristics**

**[그림 2–7]의 마지막 2Bytes 0x210E(리틀 엔디언)에서 0x2000 플래그가 세팅되어있어 이를 통하여 DLL임을 확인 할 수 있다.**

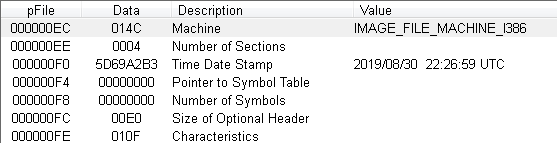
* + - 1. **Compile Time**

**Compile Time의 조작은 악성코드 분석과 사고대응을 어렵게 하는 등 악성임을 감추기 위해 조작하는 경우가 있다. [그림 2-8] (FILE\_HEADER 구조체) TimeDateStamp는 ‘1970. 01. 01 00:00:00’ 기준으로 지난 초를 말한다. 즉, TimeDateStamp가 0x3일 경우 Compile Time은 ‘1970. 01. 01 00:00:03’ 이다.**

****

**[그림 2-8] Notepad의 Modification Time, Compile Time**

**Notepad와 경우 수정 시간(Modification Time)과 컴파일 시간(Compile Time)은 근소한 차이를 보인다. Notepad 이외에도 각종 Program Files에 설치 되어있는 exe 파일의 경우에도 같은 결과를 보여주었다. 결론적으로 정상파일의 경우 수정 시간과 컴파일 시간의 차이는 크지 않다는 것이다.**

****

**[그림 2-9] FILE\_HEADER Compile Time**

**[그림 2–9]는 컴파일 시간을 조작한 악성코드이다. 현재 ‘2016.09.23’와 많이 떨어진 미래의 시간이 컴파일 시간으로 지정되어있는 것을 확인 할 수 있다. 이와 같이 컴파일 시간을 조작하는 경우 또한 악성코드로 의심할 수 있다.**

* + - 1. **Import Library**

**Import Library는 파일 내의 Import 되어있는 DLL과 함수를 확인하여 악성코드에서 주로 사용되는 DLL과 API를 파악하는 기법이다.**

|  |
| --- |
| typedef struct \_IMAGE\_IMPORT\_DESCRIPTOR {  union {  DWORD Characteristics;  DWORD OriginalFirstThunk ;  };  DWORD TimeDateStamp;  DWORD ForwarderChain;  DWORD Name;  DWORD FirstThunk;  } IMAGE\_IMPORT\_DESCRIPTOR |

[그림 2-10] ****IMAGE\_IMPORT\_DESCRIPTOR 구조체****

**파일에 Import된 DLL과 함수는 [그림 2-10] IMAGE\_IMPORT\_DESCRIPTOR 구조체에 존재한다. DLL 이름은 DWORD Name에서, 함수 리스트는 DWORD** OriginalFirstThunk에서 확인할 수 있다.

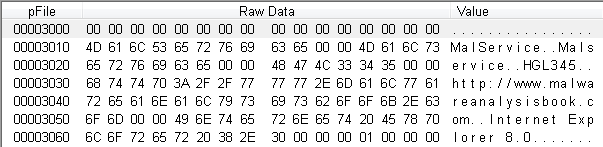
|  |  |  |
| --- | --- | --- |
| **함 수 명** | **기 능** | **악 성 코 드** |
| **CreateFileMapping** | **파일을 메모리로 로드** | **자신이 원하는 파일을 메모리로 로드** |
| **CreateRemoteThread** | **원격 프로세스에서 스레드를 시작** | **다른 프로세스에 코드 주입** |
| **CreateService** | **부팅시 시작할 수 있는 서비스를 생성** | **영구적 설치, 은폐를 목적** |
| **InternetOpenUrl** | **특정 URL 연결을 오픈** | **지정된 URL로 연결, 악성코드 다운** |
| **MapViewOfFile** | **매핑된 베이스 주소 포인터 반환** | **파일 내용 수정을 목적** |
| **OpenProcess** | **다른 프로세스 핸들을 오픈** | **다른 프로세스에 코드 주입** |
| **ReadProcessMemory** | **원격 프로세스 메모리를 읽음** | **다른 프로세스의 메모리 읽기** |
| **SetFileTime** | **파일의 생성, 접근, 수정 시간 변경** | **시간 변경으로 행위 숨김** |
| **VirtualProtectEx** | **메모리 지역 보호를 변경** | **읽기 전용 섹션 -> 실행 가능 섹션** |
| **WriteProcessMemory** | **원격 프로세스 데이터 작성** | **프로세스 인젝션** |

[그림 2-11] 악성코드에 ****주로 사용되는 함수****

**[그림 2-11]의 표는 악성코드에 주로 사용되는 함수의 일부분이다. 이외에 다양한 함수가 사용되며 해커가 직접 생성한 함수 또한 실제로 사용된다. 이렇듯 악성코드에 주로 사용되는 함수들이 포함된 파일의 경우 악성코드 혹은 파일 분석용 유틸리티 일 것이다. 악성코드와 사용하는 함수가 같은 경우 또한 있을 것이며 이를 모두 악성코드로 의심한다면 정확한 분류를 할 수 없을 것이다. 이를 위한 설명은 ‘2.2.5 VirusTotal / White List’에서 설명하도록 한다.**

* + 1. **IP, URL String**

**악성코드의 실행 방식은 다양하며 그 중 IP, URL 하드코딩은 파일 실행 시 해당 IP 또는 URL에 접속하여 악성코드를 다운 받는 등 해커가 지정한 행위를 할 것이다.**

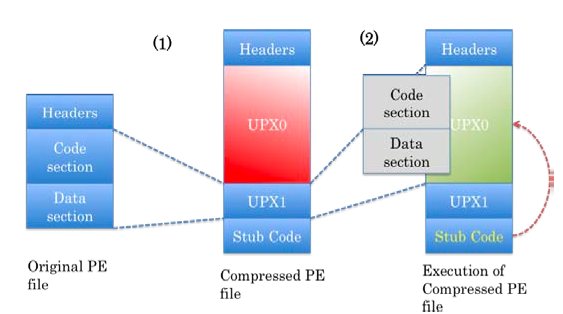
****

[그림 2-12] ****악성코드내 URL String이 포함된 샘플****

**[그림 2-12]는 악성코드 샘플의 URL 주소 하드코딩 예제이다. 이와 같이 IP 또는 URL 주소를 파일 내에 하드코딩하여 해커의 서버에 접속, 악성코드 다운받거나 원격 접속하는 등의 행위를 한다. 최근 악성코드들은 하드코딩을 하지 않고 내부 로직을 통하여 생성하거나 XOR 등을 통하여 암호화 하는 등의 방식을 사용한다. 그렇기 때문에 IP, URL 주소 하드코딩이 된 파일의 경우 악성코드로 의심할 수 있다.**

* + 1. **Packing**

**패킹(Packing)이란 실행 압축을 말한다. 실행 압축과 일반 압축의 두드러진 차이점은 PE 파일의 실행 가능 여부이며 일반 PE파일을 실행 압축파일로 만들어 주는 유틸리티를 패커라고 부른다. 패커의 특징으로는 파일크기 축소로 네트워크를 통한 전송이 유용, 내부 코드와 리소스 감추기가 있다. 많은 악성코드들은 이러한 패커의 특징을 이용하여 자신의 목적을 숨기는 용도로 사용하고 있다.**

****

**[그림 2-13] UPX 패커 실행 압축**

**UPX 패커는 잘 알려진 패커 중 하나이다. [그림 2-13]은 UPX 패커의 실행 압축과 압축 파일 실행 시 이루어지는 과정이다. 기존의 섹션이 사라지고 UPX1에 압축되며 실행 시 UPX1의 데이터들은 UPX0에 해제된다. 이렇듯 패커는 실행하기 전 자신의 코드와 리소스를 숨기는 기능을 한다. 그래서 패킹여부 파악은 악성코드 판별에 많은 도움을 줄 것이다.**

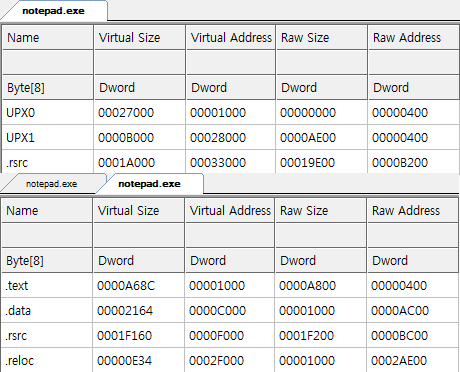
**패킹여부 파악에는 세 가지의 방식을 통하여 판단할 것이다.**

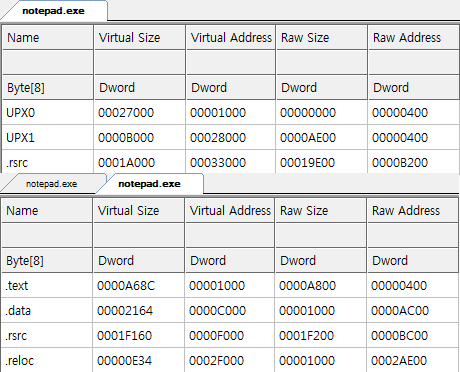
* **Section Name 비교**
* **Raw Size**
* **Section Entropy**

|  |  |
| --- | --- |
| **섹션명** | **용도** |
| **.txt** | **프로그램 실행코드를 담고 있다.** |
| **.data** | **전역 변수와 전역변수 할당을 위한 영역** |
| **.rdata** | **문자열과 상수 등을 담고 있는 영역** |
| **.bss** | **초기화 되지 않은 전역 변수를 담고 있는 영역** |
| **.edata** | **Export 할 API들의 정보를 담고 있는 영역** |
| **.idata** | **Import 할 API들의 정도를 담고 있는 영역** |
| **.rsrc** | **리소스 관련 데이터들을 담고 있는 영역** |

[그림 2-14] ****일반적인 Section Name****

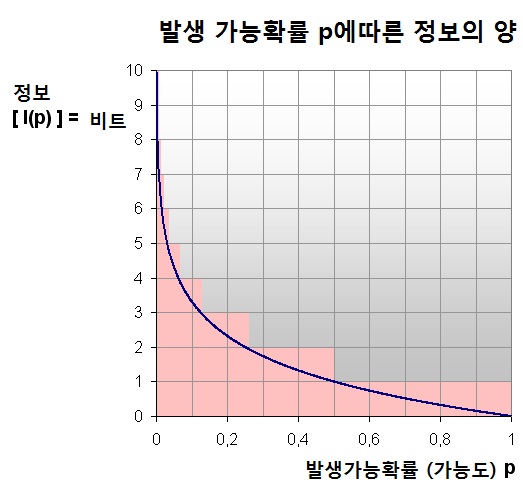
**[그림 2–14] 표의 섹션 이름들은 별도의 패킹이 되지 않은 일반적인 파일에서 볼 수 있는 이름들이다. 하지만 패킹이 이루어진 실행 압축 파일의 경우, 위와는 다른 섹션 이름들을 가지게 된다.**

****

****

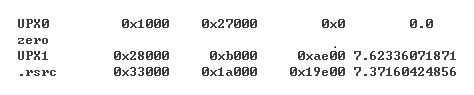
**[그림 2-15]** **UPX\_Notepad [위] / Notepad [아래]**

**[그림 2-15]는 UPX 패커를 통하여 패킹을 한 Notepad와 일반적인 Notepad이다. 패킹을 수행한 파일의 경우 [그림 2-14] 표 목록에는 없는 UPX0, UPX1을 찾아볼 수 있다. 또한 Raw Size가 0인 것을 확인 할 수 있다.**

****

[그림 2-16] ****정보의 양에 따른 Entropy****

[그림 2–16]과 같이 **Entropy**란 어떤 일에 대한 결과에 해당하는 각각의 경우의 수가 나타날 확률이 비슷해 결과를 예측하기 어려울수록 복잡함의 정도가 높아져 **Entropy** 값이 증가한다. 반대로 어느 한 경우가 나타날 확률이 낮아진다.



[그림 2-17] UPX Notepad Entropy

**[그림 2–17]은 오픈 소스 peframe을 이용하여 출력한 Entropy의 값이다. UPX0의 Raw Size는 0이기 때문에 Entropy의 값은 0, [그림 2–13]을 참조하였을 때 UPX1에는 Section들의 정보가 들어가있기 때문에 Entropy 값 7 이상이 나오게 된다. 이렇듯 0에 근접하거나 7 이상인 Entropy의 값을 나타낸다.**

**즉, Section Name, Raw Size, Entropy 값을 종합하여 패킹여부를 판단 할 수 있으며 악성코드로 의심 할 수 있다.**

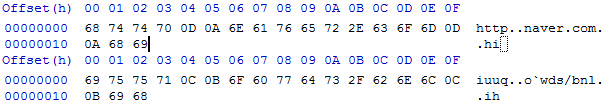
* + 1. **XOR**

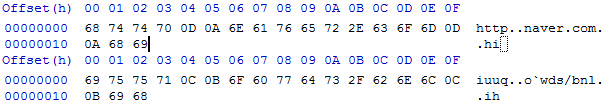
**악성코드는 악성코드 호출함수, 하드코딩 된 IP, URL 등 자신의 악성 행위를 감추기 위하여 XOR를 이용하는 경우가 적지 않다.**

|  |  |  |
| --- | --- | --- |
| **XOR** | **0** | **1** |
| **0** | **0** | **1** |
| **1** | **1** | **0** |

[그림 2-18] ****XOR 예시****

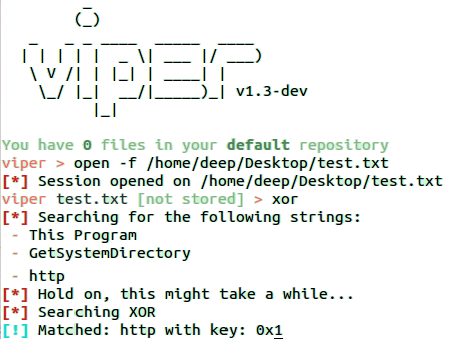
**[그림 2–18]의 XOR는 같을 경우 0, 다를 경우 1을 반환하는 논리연산자이다. 암호화에 사용할 경우 XOR에 사용한 키를 알지 못하면 본래의 값으로 복원하는 것이 불가능하다. 복호화 시 모든 키 값을 대입하여야 하기에 오랜 시간이 소요되어 단시간에 악성코드 여부를 판별해야 하는 경우 불편함이 있다.**

****

****

[그림 2-19] ****Test 원본 (위) / Test XOR (아래)****

**[그림 2-19] 원본에서 매우 간단한 단어들을 XOR 1로 암호화 하였다. 결과와 같이 출력된 문자열을 통하여 키 값을 알 수 없으며 본래의 데이터 또한 알아볼 수 없다.**

****

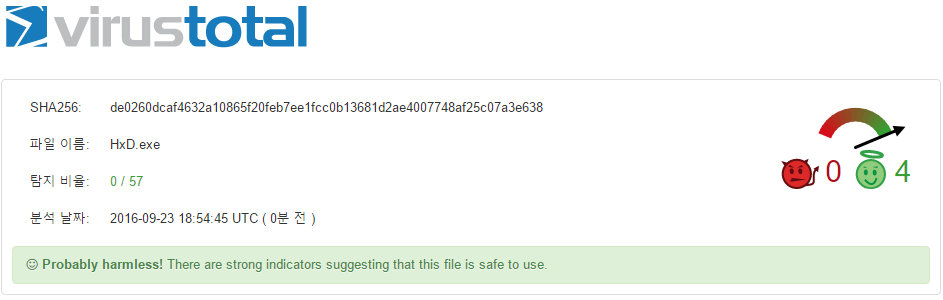
[그림 2-20] ****Viper XOR 탐지****

**파일 정적 분석 프로그램인 Viper는 XOR 탐지를 지원한다. [그림 2-20]은 [그림 2-19]의 XOR 예시를 탐지한 결과이다. XOR 키로 사용한 1을 탐지하였으며 매칭된 문자열 URL 또한 출력하였다.**

**Viper는 모든 파일에 대한 XOR의 유무를 탐지하는 것이 아니라 지정된 복호화시 지정된 문자열이 탐지될 경우에만 XOR를 탐지하는 제한적 XOR 방식이다. 하지만 URL 하드코딩, 악성코드에 주로 사용되는 함수 등과 결합하면 XOR를 탐지하는 좋은 지표가 될 것이다.**

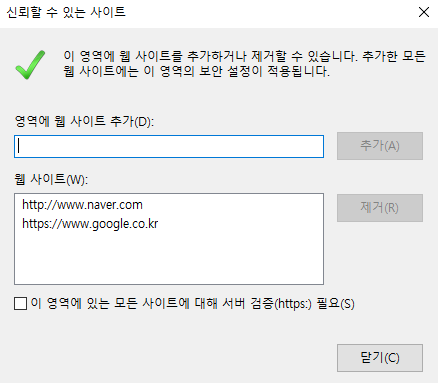
* + 1. **VirusTotal / White List**

**VirusTotal은 전 세계의 백신엔진의 검사를 거쳐 악성코드의 유무를 가려내는 신뢰성있는 악성코드 판별 사이트이다. 이러한 신뢰성 때문에 많은 개발자, 해커들이 자신의 프로그램을 검증하기 위하여 사용하기도 한다.**

****

**[그림 2-21] VirusTotal 검사 결과**

**[그림 2-21]은 파일 분석 유틸리티 HxD를 검사한 결과이다. HxD는 HDD를 손쉽게 접근하여 내부를 들여볼 수 있으며, 파일의 정보를 수정하거나 하드코딩 하는 등 다양한 악성 행위를 가능하게 할 수 있는 기능을 가지고 있다. 이렇듯 악성 프로그램과도 같이 보이는 유틸리티들이 악성프로그램이 아닐 수도 있다. 그렇기 때문에 이러한 문제를 해결하기 위한 방식 중 하나가 ‘White List’이다.**

****

[그림 2-22] ****신뢰할 수 있는 사이트****

**White List는 우리의 실생활에도 자주 사용되는 것을 볼 수 있다. 가장 대표적인 예시는 [그림2-22]와 같은 신뢰할 수 있는 사이트 이다. 웹 서핑 중 ‘신뢰할 수 있는 사이트를 등록하시오’ 라는 메시지를 자주 볼 수 있을 것이다. 즉, White List는 믿을 수 있는 것들을 모아둔 리스트이다. 이를 이용하여 악성코드와 비슷한 행위의 정상파일에 대한 오진율을 낮출 수 있을 것이다.**

White List에 대한 자세한 구현 방법은 ‘3.6 Deep Learning 구성도’에서 설명하도록 한다.

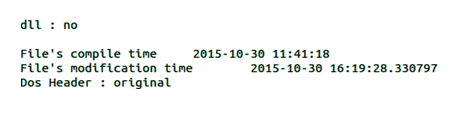
* 1. **정적 분석 스크립트 개발**

* + 1. **정적 분석 스크립트 및 결과 값**
       1. **Dos Header / Characteristics / Compile Time**

|  |
| --- |
| def PE\_INFO(file\_name, pe):  dll = pe.FILE\_HEADER.Characteristics  image\_dll = 0b10000000000000  com\_time = datetime.datetime.fromtimestamp(pe.FILE\_HEADER.TimeDateStamp)  modi\_time = datetime.datetime.fromtimestamp(os.path.getmtime(file\_name))  Dos\_cparhdr = pe.get\_word\_from\_offset(0x08)  cpu\_a = pe.FILE\_HEADER.Machine  if (int(dll)&image\_dll):  is\_dll = 1  else:  is\_dll = 0  if hex(Dos\_cparhdr) !='0x4':  e\_cparhdr = 1  else:  e\_cparhdr = 0  c\_time = str(com\_time).split("-")  m\_time = str(modi\_time).split("-")  num = int(c\_time[0]) - int(m\_time[0])  if abs(num) > 1:  is\_time =1  else:  is\_time=0  return is\_dll, e\_cparhdr, is\_time |

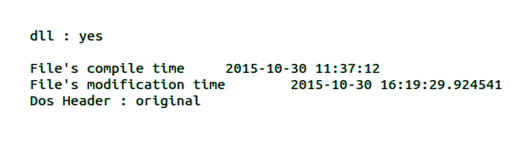
[그림 2-23] Dos Header 변조 / DLL 여부 / Compile Time 변경 확인 함수

[그림 2–23]의 PE\_INFO() 함수는 DOS\_HEADER, NT\_HEADERS의 구조체 정보를 매개변수로 받고 DOS\_HEADER 수정 여부, DLL 여부, 컴파일 시간 수정 여부를 판단한다. 첫 번째 DOS\_HEADER, DOS\_HEADER는 총 64Bytes의 구조체로 8Bytes에 위치한 e\_cparhdr의 값을 비교하여 같으면 0, 다르면 1을 반환한다. 두 번째, DLL 여부는 NT\_HEADERS의 FILE\_HEADER Characteristics와 DLL의 플래그 0x2000을 AND 연산 하여 결과가 1이면 1을 0이면 0을 반환한다. 세 번째 Compile Time 수정 여부, NT\_HEADERS의 FILE\_HEADER TimeStamp의 Compile Time과 Modification Time을 비교하여 1년 이상의 많은 차이가 날 경우 1을 아닐 경우 0을 반환한다. 반환된 데이터들은 딥러닝 악성코드 판별자료로 사용한다.



[그림 2-24] Dos Header 변조 / DLL 여부 / Compile Time 변경 확인 결과 1

[그림 2–24]를 보면 확장자가 DLL인지 확인 결과 No가 출력 되었고 파일의 생성 시간과 컴파일 시간 차이가 없는 것을 확인할 수 있고 Dos Header의 변화가 없는 것을 확인 할 수 있다.



[그림 2-25] Dos Header 변조 / DLL 여부 / Compile Time 변경 확인 결과 2

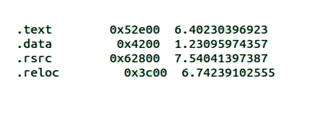
[그림 2–25]를 보면 EXE 확장자로 변경한 DLL파일 또한 DLL임을 검출할 수 있고 파일의 생성 시간과 컴파일 시간 차이가 없는 것을 확인할 수 있고 Dos Header의 변화가 없는 것을 확인 할 수 있다.

* + - 1. **Raw Data Size / Section Entropy**

|  |
| --- |
| def SECTIONS(pe):  raw\_size = 0 entropy = 0  print '\n'  for section in pe.sections:  if section.SizeOfRawData == 0:  raw\_size = 1  if (section.get\_entropy() > 0 and section.get\_entropy() < 1) or section.get\_entropy() > 7:  entropy = 1  return raw\_size, entropy |

[그림 2-26] Raw Data Size 값 0인지 여부 / Entropy 값 확인 함수

[그림 2–26]의 SECTIONS() 함수는 파일의 PE 구조 중 FILE HEADER의 구조체 정보를 매개변수로 받고 for문을 통해 해당 파일의 PE 구조 중 각 섹션의 크기가 0이면 1을, 아니면 0을 반환한다. 또한 각 섹션의 Entropy 값이 0 초과 1 미만의 값을 가지거나 7을 초과하면 1을 반환하고, 아니면 0을 반환한다. 반환된 데이터들은 딥러닝 악성코드 판별자료로 사용된다.



[그림 2-27] Raw Data Size 값 0인지 여부 / Entropy 값 확인 결과

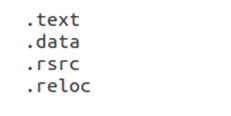
[그림 2–27]을 보면 각 섹션마다 이름과 해당 섹션의 RawDataSize와 Entropy를 구하는 것을 확인할 수 있다.

* + - 1. **Section Name**

|  |
| --- |
| def sname\_list(txt\_file):  L=[]  i=0  j=0  V=[]  for root, dirs, files in os.walk(txt\_file):  for file in files:  L.append(file)  for c in L:  s2 =sys.argv[1]+"/"  s2 = s2+L[i]  try:  pe\_fun = pefile.PE(s2)  i = i+1  for section in pe\_fun.sections:  name = section.Name.strip("\x00")  if name in V:  j=0  else:  V.append(name)  except:  print 'not found Dos Header'  f = open(sys.argv[3],'w')  z=0  for q in V:  f.write(V[z]+"\n")  z=z+1  f.close() |

[그림 2-28] 파일 섹션 이름을 저장하는 함수

[그림 2–28]의 sname\_list() 함수는 딥 러닝 교육 데이터 중 SECTION\_NAME의 기준이 되는 리스트를 생성하는 함수이다. 그리고 위 함수는 검사에 기준이 될 파일들이 잇는 디렉토리 경로를 매개변수로 받는다. 디렉토리 안에 있는 파일들의 파일명을 L\_리스트에 저장시키고 저장 된 파일명을 한 개씩 가져와 중복 없이 섹션의 이름을 V\_리스트에 저장 한 뒤 메모장에 저장시킨다.



[그림 2-29] 파일 섹션 이름을 저장 결과

[그림 2–29]는 CALC.EXE 파일을 sname\_list() 함수로 메모장에 저장한 결과이다.

|  |
| --- |
| def R\_snlist(txt\_file):  f = open(txt\_file)  i=0  j=0  for line in f:  line = line.strip('\n')  line = line.strip('\r')  L\_sn.append(line)  f.close()  def sn\_ox(pe\_fun):  f = open(sys.argv[3],'a')  L\_c\_sn=[]  for c in pe\_fun.sections:  name = c.Name.strip("\x00")  L\_c\_sn.append(name)  i = 0  for b in L\_sn:  if b in L\_c\_sn:  f.write("1\t")  else:  f.write("0\t")  f.close() |

[그림 2-30] Section Name 리스트 호출 함수 / Section Name 비교 함수

[그림 2–30]의 R\_snlist() 함수는 [그림 2-29]을 통해 저장한 메모장의 경로를 매개변수로 사용하고 메모장 정보를 한 줄씩 읽어 L\_SN\_배열에 저장시키고 sn\_ox() 함수는 파일의 PE 구조 중 FILE HEADER의 구조체 정보를 매개변수로 받아 파일의 섹션정보를 L\_C\_SN\_배열에 저장하고 딥러닝 교육 데이터 중 SECTION\_NAME의 기준이 되는 리스트안에 동일한 색션정보가 존재하면 1을 아니면 0을 반환한다. 반환 된 데이터들은 딥러닝 악성코드 판별자료로 사용한다.

|  |
| --- |
| def function(functions ,pe\_fun):  e = 0  try:  for entry in pe\_fun.DIRECTORY\_ENTRY\_IMPORT:  for imp in entry.imports:  if entry.dll in functions :  if imp.name in functions[entry.dll]:  e = 0  else :  functions[entry.dll].append(imp.name)  else:  functions.setdefault(entry.dll,[]).append(imp.name)  except:  print 'no import table' |

[그림 2-31] DLL, API 추출 함수

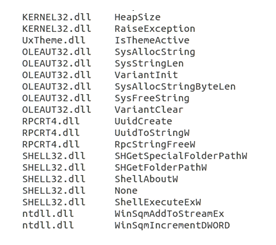
[그림 2-31]의 function() 함수는 DLL, API를 저장하기 위한 DICTIONARY 변수와 파일의 PE 구조 중 IMPORT\_DESCRIPTOR 구조체 정보를 매개변수로 받고 for문을 통해 파일의 IMPORT된 DLL과 API 함수를 중복되지 않게 DICTIONARY 변수에 저장시킨다.

* + - 1. **Import Library**

|  |
| --- |
| def save\_func(file\_name, functions):  f = file(file\_name, 'w')  for dlls in functions:  i = 0  for apis in functions[dlls]:  f.write(dlls)  f.write('\t')  f.write(str(functions[dlls][i]))  f.write('\n')  i+=1;  f.close |

[그림 2-32] DLL, API 저장 함수

[그림 2–32]의 save\_func() 함수는 저장할 딥러닝 교육 데이터 중 DLL, API 함수의 기준이 값들을 저장 할 파일명과 function() 함수를 통해 얻은 DLL 이름과 API 함수가 담긴 DICTIONARY 변수를 매개변수로 받으며 [TAB]과 [ENTER]를 사용하여 각 DLL 이름과 해당 DLL에 속하는 API에 대한 함수명을 메모장에 저장시킨다.



[그림 2-33] DLL, API 저장 결과

[그림 2–33]는 CALC.EXE 파일을 save\_func()함수로 메모장에 저장한 결과를 보여준다.

|  |
| --- |
| def R\_list(txt\_file):  f = open(txt\_file)  i=0  for line in f:  line = line.strip('\n')  dll, api = line.split('\t')  api = api.strip('\r')  if dll in L\_dll:  i=0  else: L\_dll.append(dll) L\_api.append(api)  f.close() |

[그림 2-34] DLL, API 리스트 호출 함수

[그림 2–34]의 R\_list() 함수는 save\_func()를 통해 저장된 파일의 파일명을 매개변수로 받고 한 줄씩 읽어 L\_DLL\_리스트와 L\_API\_리스트에 중복되지 않게 딥러닝 교육 데이터 중 DLL 이름, API 함수명의 기준이 되는 리스트를 생성하는 함수이다.

|  |
| --- |
| def W\_ox(pe\_fun):  f = open(sys.argv[3],'a')  a = 0  L\_c\_dll=[]  L\_c\_api=[]  MAL\_API = ['OpenProcess',  'VirtualAllocEx',  ......  'MapViewOfFile',  'Module32First',  'NtQueryDirectoryFile'  ] try:  m = pe\_fun.DIRECTORY\_ENTRY\_IMPORT for c\_entry in pe\_fun.DIRECTORY\_ENTRY\_IMPORT:  if c\_entry.dll in L\_c\_dll:  a=0  else:  L\_c\_dll.append(c\_entry.dll)  for c\_imp in c\_entry.imports:  if c\_imp.name in L\_c\_api:  a=0  else:  L\_c\_api.append(c\_imp.name)  for c\_mal in mal\_api:  if c\_mal in L\_c\_api:  f.write("1\t")  else:  f.write("0\t")  for entry in L\_dll:  if entry in L\_c\_dll:  f.write("1\t")  else:  f.write("0\t")  for imp in L\_api:  if imp in L\_c\_api:  f.write("1\t")  else:  f.write("0\t")  except:  for c\_mal in MAL\_API:  f.write("0\t")  for entry in L\_dll:  f.write("0\t")  for imp in L\_api:  f.write("0\t")  f.close() |

[그림 2-35] DLL, API 리스트 비교 함수

[그림 2–35]의 W\_ox() 함수는 파일의 PE 구조 중 IMPORT\_DESCRIPTOR 구조체 정보를 매개변수로 받는다. L\_C\_DLL 리스트와 L\_C\_API리스트에 파일의 DLL 이름정보와 API 함수명 정보를 저장한다. R\_list() 함수를 통해 얻은 리스트를 통해 L\_C\_DLL 리스트와 L\_C\_API 리스트를 비교하여 같은 DLL 이름이나 같은 API 함수명 있는 경우에는 1, 아니면 0을 반환한다. 그리고 악성코드들이 주로 사용하는 주요 함수들이 저장된 MAL\_API 리스트와 L\_C\_API 리스트를 비교하여 동일한 API 함수가 있는 경우 1, 아니면 0을 반환한다. 반환된 데이터들은 딥러닝 악성코드 판별자료로 사용한다.

* + - 1. **IP, URL String**

|  |
| --- |
| printable = set(string.printable)  def process(stream):  found\_str = ""  while True:  data = stream.read(1024\*4)  if not data:  break  for char in data:  if char in printable:  found\_str += char  elif len(found\_str) >= 4:  #if len bigger than 4 return found\_str  yield found\_str  found\_str = ""  else:  found\_str = "" |

[그림 2-36] Hex Data String 변환 함수

[그림 2–36]의 printable = set(string.printable)를 설정하는 이유는 파이썬에서 제공한 라이브러리에서 출력 가능한 문자인지를 알려주기 위함이다. [그림 2–37]에서 출력 가능한 문자를 확인할 수 있다.

|  |
| --- |
| digits = '0123456789'  whitespace = ' \t\n\r\v\f'  lowercase = 'abcdefghijklmnopqrstuvwxyz'  uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  octdigits = '01234567'  punctuation = """!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~""" |

[그림 2-37] 출력 가능 문자

[그림 2–36]의 process() 함수는 ip\_search() 함수로부터 넘겨받은 파일 스트림을 매개변수로 받아 read() 함수를 사용해 분석 대상 파일의 내용을 4096Byte씩 읽고 DATA에 저장시킨다. DATA를 하나씩 추출해서 해당 값이 출력 가능한 문자인지 판단하고 가능한 문자이면 문자를 found\_str에 이어 붙여 문자열을 만들고 불가능한 문자이면 found\_str에 저장된 문자열을 다음 [그림 2–38]의 ip\_search() 함수로 반환해 준다. 단, 문자열의 길이가 4이하인 경우는 그냥 ip\_search() 함수로 반환하지 않는다.

|  |
| --- |
| def ip\_search(file\_name):  is\_ip = 0  is\_dns = 0  PEtoStr = open(file\_name, 'rb')  for found\_str in process(PEtoStr):  #search ip address  m = re.search('(\d{1,3}\.){3}\d{1,3}', str(found\_str))  if m:  if m.group(0) != '127.0.0.1' and m.group(0) !='6.0.0.0':  is\_ip = 1  #search http or https address  n = re.search('h\D{3,4}\:\/\/\D{0,20}', str(found\_str))  if n:  is\_dns = 1  PEtoStr.close()  return is\_ip, is\_dns. |

[그림 2-38] IP 하드코딩 / URL 하드코딩 여부 확인 함수

[그림 2–38]의 ip\_search() 함수는 문자열 정보로 추출하고 싶은 파일의 이름을 매개변수로 받고 매개변수로 받은 파일명을 이진 파일 읽기 모드로 접근한다. ip\_search() 함수의 for문으로 [그림 2–36] process() 함수를 통해 반환 받은 값을 IP형식 검출 정규식으로 처리한 결과가 127.0.0.1과 6.0.0.0을 제외한 IP형식(x.x.x.x)이 존재할 경우 is\_ip 변수를 1 아니면 0을 반환하고 URL 형식 검출 정규식으로 URL이 존재할 경우 is\_dns 변수를 1아니면 0을 반환한다. 반환된 데이터들은 딥러닝 악성코드 판별자료로 사용한다.



[그림 2-39] URL 하드코딩 출력 결과

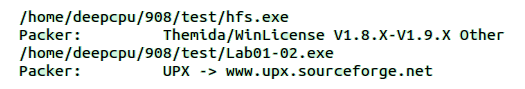
[그림 2–39]를 보면 URL이 하드코딩 된 것을 확인 할 수 있다.

* + - 1. **Packing**

|  |
| --- |
| def peid(pe):  packer = 0  signatures = peutils.SignatureDatabase(pathname + os.sep +  'modules' + os.sep + 'userdb.txt')  matches = signatures.match\_all(pe , ep\_only = True)  if matches:  packer = 1  print'Packer : ',matches[0][0]  return packer |

[그림 2-40] 패킹여부 확인 함수

[그림 2–40]의 peid() 함수는 패킹 여부를 검사하고, 패킹되어 있는 경우 peframe에서 제공해주는 시그니처 DB를 이용하여 해당 패커의 이름을 출력해주는 함수이며 시그니처 DB는 userdb.txt라는 이름으로 저장되어 있고 SignatureDatabase() 함수를 통해 패커의 시그니처 정보를 받아오고 파일의 시그니처와 비교하여 매칭이 되면 1을 아니면 0을 반환한다. 반환된 데이터들은 딥러닝 악성코드 판별자료로 사용한다.



**[그림 2-41] 패킹 확인 결과**

[그림 2–41]은 peid() 함수를 이용해서 UPX로 패킹된 파일(Lab01-02.exe)의 결과 값과 Themida로 패킹된 파일(hfs.exe)의 결과 값이다. 2개 파일의 시그니처를 통해 패킹 여부를 확인할 수 있다.

* + - 1. **XOR**

|  |
| --- |
| def xor(file\_name):  terms = ['This Program',  'GetSystemDirectory',  'CreateFile',  'GetProcAddress',  'LoadLibrary',  'http', ..........  'bind',  'connect',]  is\_xor = 0  C=[]  count = 0  f = open(file\_name,'rb')  fa = f.read()  for key in range(1, 256):  if(count==1):  break  fb = Bytearray(fa)  for i in range(len(fb)):  fb[i]^=key  for term in terms:  if term in fb and count==0:  count = 1  is\_xor = 1  if is\_xor == 1:  print'Xor Check : O'  else:  print'Xor Check : X'  return is\_xor |

[그림 2-42] XOR 여부 확인 함수

[그림 2–42]의 xor() 함수는 1Byte 키를 이용한 xor 여부를 확인하려는 파일의 이름을 매개변수로 받고 매개변수로 받은 파일명을 이진 파일 읽기 모드로 접근하고 데이터를 1Byte씩 읽어 변수 fa에 저장시키고 1부터 255까지 xor를 수행하고 수행한 값이 악성행위에 자주 사용되는 함수 및 문자들을 저장한 리스트 term에 포함 되어있으면 1 포함되지 않으면 0을 반환하고 반환 된 데이터들은 딥러닝 악성코드 판별자료로 사용한다.

C:\Users\JHH\Desktop\이미지\간격 + 이미지\2-42.png

[그림 2-43] XOR 확인 결과

[그림 2–43]을 보면 파일의 크기(Byte)만큼 1Byte씩 xor 디코딩을 하여 리스트에서 일치하는 문자열과 키 값이 출력되는 것을 확인할 수 있다.

* + - 1. **VirusTotal**

|  |
| --- |
| def hash(exename , bloacksize):  sha\_256 = hashlib.sha256()  try:  f= open(exename , "rb")  except IOError as e:  print("file open error",e)  return  while True:  buf = f.read(bloacksize)  if not buf:  break  sha\_256.update(buf)  return sha\_256.hexdigest() |

[그림 2-44] Hash값 확인 함수

[그림 2–44]의 hash() 함수는 해쉬 값을 얻고 싶은 파일명과 변수 blocksize를 매개변수로 받고 hashlib에서 제공하는 sha256() 함수를 이용해 HASH 객체를 생성하고 분석 대상 파일을 이진 파일 읽기 모드로 접근한 뒤 파일을 blocksize 만큼씩 Byte를 읽어 보기 쉬운 16진수로 변환해준다.

|  |
| --- |
| def VT\_Request(key, hash):  #initialize VirusTotal api key  params = {'apikey': key, 'resource': hash}  url = requests.get('https://www.VirusTotal.com/vtapi/v2/file/report', params=params)  json\_response = url.json()  response = int(json\_response.get('response\_code'))  if response == 0:  z=0  elif response == 1:  positives = int(json\_response.get('positives'))  if positives < 10:  positives = 0  else:  positives = 1  return positives |

[그림 2-45] VirusTotal 질의 결과 함수

[그림 2–45]의 VT\_Request() 함수는 VirusTotal의 오픈소스를 사용하기 위한 사용자 KEY와 파일의 해쉬 값을 매개변수로 받는다. VirusTotal의 URL 정보를 가지고 해쉬 값에 대한 내용을 분석한 후 원하는 값을 가져온다. VirusTotal에서 가져온 값이 악성코드라고 판단하면 1을, 아니면 0을 반환시킨다. 반환된 데이터들은 딥러닝 악성코드 판별자료로 사용한다.



[그림 2-46] Hash 값 확인 / VirusTotal 질의 결과 출력 1

[그림 2 – 46]에서 hash() 함수를 통해 파일의 해쉬 값을 얻고 VT\_Request()를 통해 해쉬 값이 악성코드인지 아닌지 결과를 받아 화면에 출력한다.

C:\Users\JHH\Desktop\이미지\간격 + 이미지\2-46.png

[그림 2-47] Hash 값 확인 / VirusTotal 질의 결과 출력 2

[그림 2-47]을 보면 파일명이 같아도 다른 해쉬 값을 얻는 것을 확인 할 수 있다.

* + 1. **딥러닝 학습 데이터 추출 순서**

|  |  |  |
| --- | --- | --- |
| 정적 분석  1~9 | Suspect API  10~31 | DLL / API /SESSION NAME LIST  32~18066 |

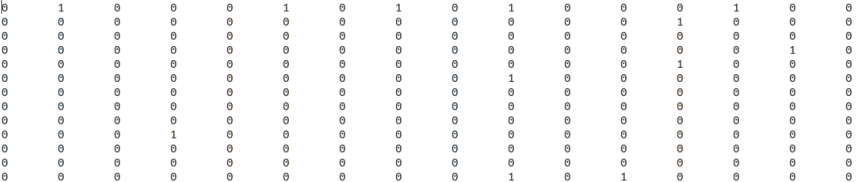
[그림 2-48] 학습 데이터 추출 순서도

[그림 2-48]은 정적 분석으로 통해 추출한 18,066개의 특징값을 순서대로 표현한 것이다.

|  |  |  |  |
| --- | --- | --- | --- |
| **순서** | **기준** | **0** | **1** |
| 1 | Raw Size | Raw Size != 0 | Raw Size = 0 |
| 2 | Section entropy | entropy = 1~7 | 0<entropy<1 & entropy>7 |
| 3 | Characteristics | 0x2000 flag ( O ) | 0x2000 flag ( X ) |
| 4 | Dos Header | Original | Modification |
| 5 | Compile Time | Original | Modification |
| 6 | IP String | X | 0 |
| 7 | URL String | X | 0 |
| 8 | Packer | X | 0 |
| 9 | XOR | X | 0 |
| 10~31 | Suspect API | X | 0 |
| 32~18066 | DLL / API / SESSION NAME LIST | | |

[그림 2-49] 학습 데이터 순서 분류 기준

[그림 2-49]는 딥러닝 학습을 위한 악성코드의 특징 값을 나타낸 도표이다. 1~9번의 데이터는 악성코드 판별에 기초가 되는 특징 값이다. 그리고 10~31번의 데이터는 주로 악성코드에서 사용되는 API 함수이다. 그 외의 데이터는 악성코드 및 일반 파일에서 추출한 DLL, API, SESSION NAME에 대한 특징 값이다.



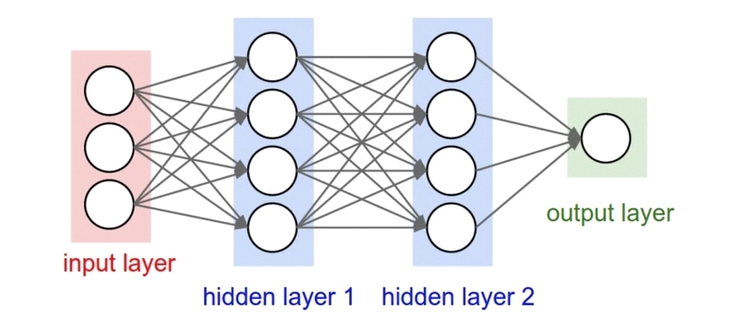
[그림 2-50] 학습 데이터 결과 값

[그림 2-50]은 [그림 2-49]의 기준으로 추출한 학습 데이터의 결과 값이다.

1. **Tensorflow를 이용한 딥러닝 시스템 구축**
   1. **Neural Network**
      1. **Neural Network 개념**

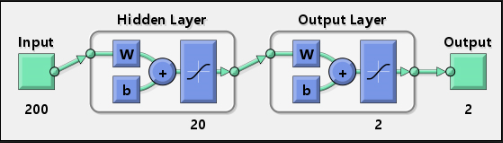
뉴럴 네트워크(Neural Network)는 뉴로라고도 불리는 것으로, 인간의 뇌 기능을 적극적으로 모방하려는 생각에 기초를 두고 있다. 즉, 무언가를 보고, 그것이 무엇인가를 인식하여, 필요에 따라 행동을 취한다는, 인간에게는 아주 간단하고 당연한 사고방식을 컴퓨터에 학습시키려는 것이다. 한마디로 말하면, 제어대상과 관련된 다양한 요인을 설정하고, 이들의 결합과 결합의 무게(Weight)를 생각하는 방법이다. 계산을 하는 과정에 학습기능을 부가함으로써 최적의 제어가 가능하다.

* + 1. **Neural Network 구성도**



**[그림 3-1] 일반적인 Neural Network 알고리즘**

일반적인 Neural Network의 알고리즘은 [그림 3-1]과 같이 표현할 수 있다.



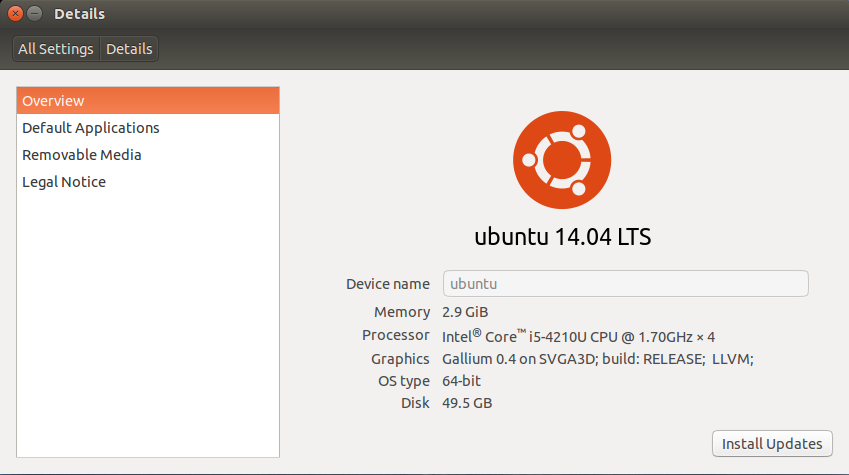
[그림 3-2] Neural Network 알고리즘 내부 구성도

[그림 3-2]는 Neural Network를 보다 이해하기 쉽게 직렬로 표현한 알고리즘이다. Neural Network는 Input data와 Weight(가중치)를 이용해서 Hypothesis(가상선)로 나타낸다. 딥러닝에서는 Back Propagation(역전달 학습)이 제대로 진행될 수 있도록 앞쪽 Layer에서는 Relu 함수로 나타내고 마지막 Layer에서 Sigmoid 함수 형태로 나타낸다. Relu와 Sigmoid 함수는 활성화 함수라고 하는데 이것은 특정 상황에 대해서 활성화를 할지 안할지를 결정하는 함수를 나타내는 말이다. Hypothesis가 만들어지면 이것에 대한 Cost(오차 정도)를 함수로 나타낸다. 그리고 이 Cost를 Optimizer함수를 이용해서 낮춰가는 것을 학습한다고 한다.

* 1. **Tensorflow** 
     1. **Tensorflow 개념**

Tensorflow는 구글(Google)사의 검색, 음성 인식, 번역 등의 구글 앱에 사용되는 기계 학습용 엔진으로, 2015년에 공개된 오픈소스 소프트웨어이다. Tensorflow는 C++ 언어로 작성되었고, 파이선(Python) 응용 프로그래밍 인터페이스(API)를 제공한다. Tensorflow는 빠르고 유연하여 한 대의 스마트 폰에서도 운영될 수 있고, 데이터센터의 수천 대 컴퓨터에서도 동작될 수 있다.

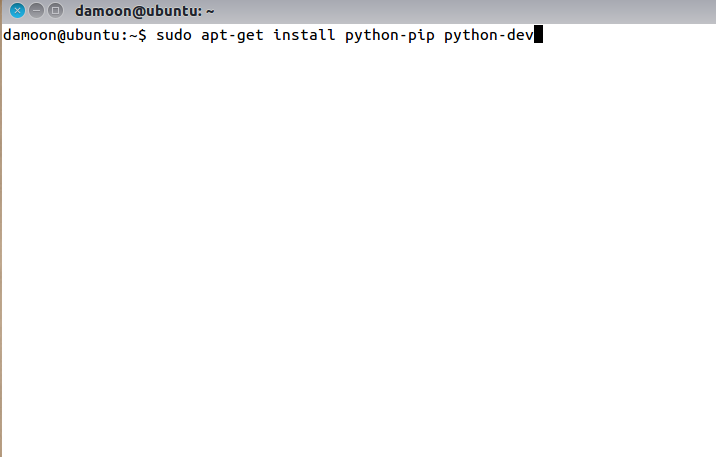
* + 1. **Tensorflow 개발 구축 환경**

****

[그림 3-3] Tensorflow 개발 구축 환경

[그림 3-3]과 같이 가상 머신(Virtual Machine)의 Ubuntu 14.04 LTS에서 Tensorflow를 구축하였다.

* + 1. **Tensorflow 개발 구축 과정**



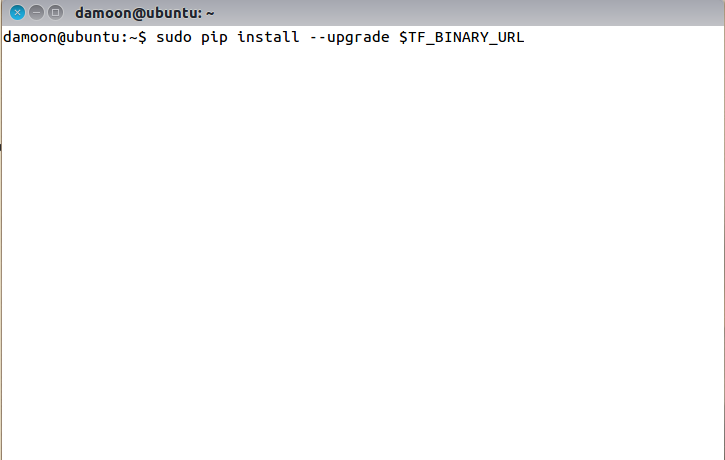
[그림 3-4] Tensorflow 개발 구축 과정 1

Tensorflow는 Python에서 작성된 소프트웨어 패키지이므로 먼저 pip (패키지 관리시스템)를 설치해야한다.



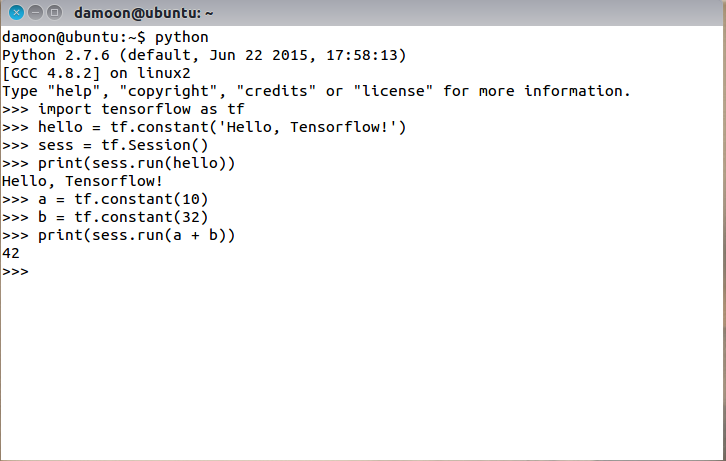
[그림 3-5] Tensorflow 개발 구축 과정 2

Tensorflow 소프트웨어 설치 경로 URL을 다음과 같이 공개된 URL로 설정한다.



[그림 3-6] Tensorflow 개발 구축 과정 3

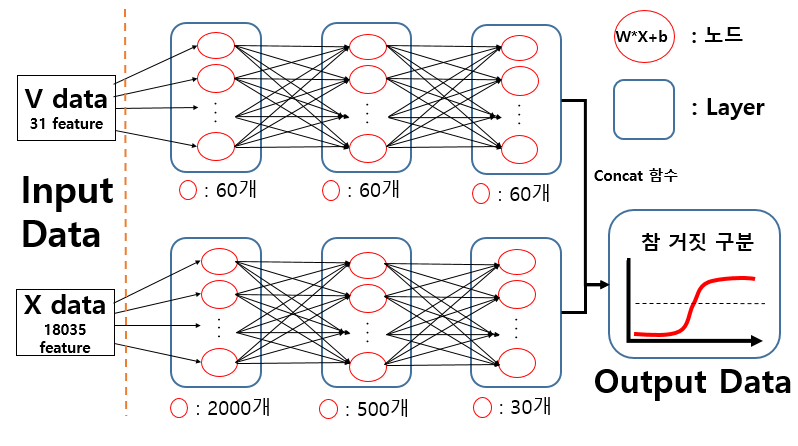
Tensorflow 소프트웨어를 설치한다.



[그림 3-7] Tensorflow 개발 구축 과정 4

올바르게 설치 되었으면 다음과 같은 간단한 연산 수식을 Tensorflow 함수를 사용해 정상적인 값이 출력되는지 확인한다. 정상적인 값이 출력이 되면 Tensorflow를 성공적으로 설치한 것이고, Python 코드에서 Tensorflow 함수를 사용할 수 있다.

* 1. **정적 분석 데이터를 이용한 알고리즘 구성**



[그림 3-8] 정적 분석 데이터를 이용한 알고리즘 구성도

단순한 Neural Network는 [그림 3-2] 알고리즘 구성도와 같이 한 방향의 Layer로 진행이 되지만, 정적 분석 데이터를 이용한 알고리즘 구성도 [그림 3-8]의 경우는 Layer를 병렬로 구성하였다. 정적 분석으로 악성행위를 판별하는 데 있어 좀 더 중요하다고 생각하는 31개의 Feature data를 V data로 설정하여 첫 번째 Layer에 입력하였고 나머지 18035개의 Feature data를 X data로 설정하여 두 번째 Layer에 입력하여 병렬 알고리즘으로 구성하였다.

Neural Network에서 가장 효과적으로 정적 분석 데이터를 학습시키기 위해서는 다음과 같은 설정값 조절이 필요하다.

* + 1. **알고리즘의 Layer 및 Node 개수 조절**

|  |
| --- |
| nw1 = tf.get\_variable("N\_w1", shape=[31 ,60],  initializer=tf.contrib.Layers.xavier\_initializer())  nw2 = tf.get\_variable("N\_w2", shape=[60 , 60],  initializer=tf.contrib.Layers.xavier\_initializer())  nw3 = tf.get\_variable("N\_w3", shape=[60, 60],  initializer=tf.contrib.Layers.xavier\_initializer())  nb1 = tf.Variable(tf.zeros([60]), name="N\_Bias1")  nb2 = tf.Variable(tf.zeros([60]), name="N\_Bias2")  nb3 = tf.Variable(tf.zeros([60]), name="N\_Bias3")  \_nL1 = tf.nn.relu(tf.matmul(V, nw1) + nb1)  nL1 = tf.nn.dropout(\_nL1, drop\_late)  \_nL2 = tf.nn.relu(tf.matmul(nL1, nw2) + nb2)  nL2 = tf.nn.dropout(\_nL2, drop\_late)  \_nL3 = tf.nn.relu(tf.matmul(nL2, nw3) + nb3)  nL3 = tf.nn.dropout(\_nL3, drop\_late) |

[그림 3-9] V data의 Layer 및 Node 개수

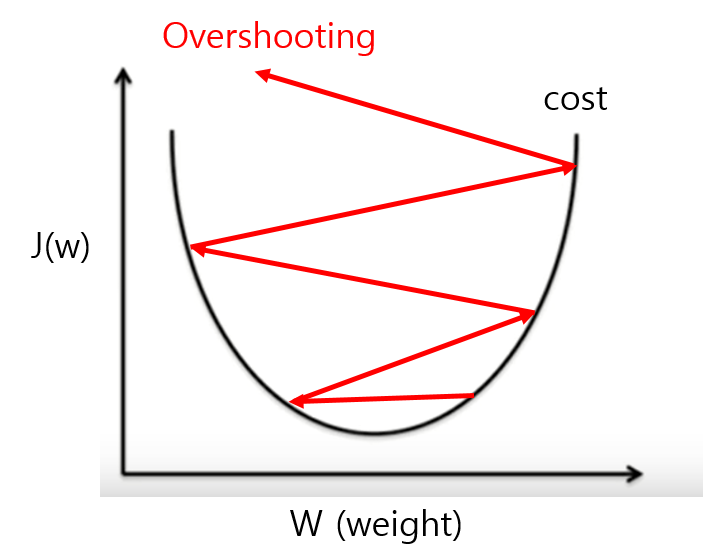
정적 분석으로 악성행위를 판별하는 기준점인 Feature data는 총 18,066개이다. 18,066개 중 가장 중요하다고 생각되는 31개의 V data는 [그림 3-9]의 소스코드와 같이 첫 번째 Layer에서 31개의 Node 수를 60개로 증가시켰고, 두 번째와 세 번째 Layer에서는 60개의 Node 수를 그대로 유지하였다.

|  |
| --- |
| w1 = tf.get\_variable("w1", shape=[18035, 2000],  initializer=tf.contrib.Layers.xavier\_initializer())  w2 = tf.get\_variable("w2", shape=[2000, 500],  initializer=tf.contrib.Layers.xavier\_initializer())  w3 = tf.get\_variable("w3", shape=[500, 30],  initializer=tf.contrib.Layers.xavier\_initializer())  w4 = tf.get\_variable("w4", shape=[90, 1],  initializer=tf.contrib.Layers.xavier\_initializer())    b1 = tf.Variable(tf.zeros([2000]), name="Bias1")  b2 = tf.Variable(tf.zeros([500]), name="Bias2")  b3 = tf.Variable(tf.zeros([30]), name="Bias3")  b4 = tf.Variable(tf.zeros([1]), name="Bias4")  \_L2 = tf.nn.relu(tf.matmul(X, w1) + b1)  L2 = tf.nn.dropout(\_L2, drop\_late)  \_L3 = tf.nn.relu(tf.matmul(L2, w2) + b2)  L3 = tf.nn.dropout(\_L3, drop\_late)  \_L4 = tf.nn.relu(tf.matmul(L3, w3) + b3)  L4 = tf.nn.dropout(\_L4, drop\_late)  LL = tf.concat(1,[L4,nL3])  hypothesis = tf.Sigmoid(tf.matmul(LL, w4) + b4) |

[그림 3-10] X data의 Layer 및 Node 개수

[그림 3-10]을 보면, 첫 번째 Layer에서는 18035개의 X data를 2,000개의 Node로 추출하였고, 두 번째 Layer에서는 2,000개의 Node 수를 500개로, 세 번째 Layer에서는 500개의 Node 수를 30개로 추출했다. 세 번째 Layer에서 추출 된 Node 30개와 V data 마지막 Layer의 Node 60개를 concat() 함수로 합쳐 90개의 Node를 네 번째 Layer에서 1개의 Output data로 추출했다. Concat() 함수로 V data와 X data를 합치는 Layer에서 V data의 Node 수를 X data보다 2배로 많게 설정하여 V data의 가중치(Weight)를 높게 부여하였다.

* + 1. **Cost 값 최적화를 위한 Learning rate 값 조절**



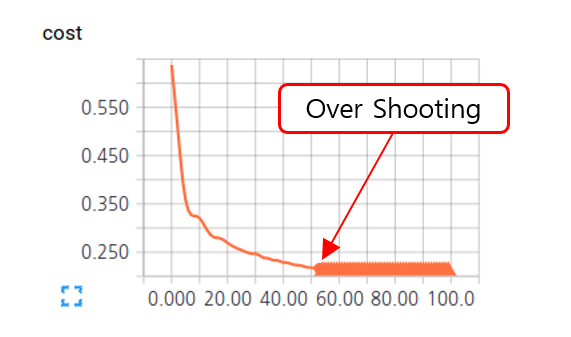
[그림 3-11] Overshooting

데이터를 학습시키는 과정에서 Overshooting이 발생할 수 있다. Overshooting이란 Cost값을낮추는 과정에서 [그림 3-11]과 같이 값이 그래프 밖으로 튕겨나가면서 결과 값이 NAN (Not A Number)이 나타나는 현상을 말한다. Learning rate 값이 클 경우에는 Overshooting이 발생할 확률이 높아지기 때문에 이를 방지하기 위해서는 Learning rate 값 조절이 필요하다. 그리고 Cost값을 최소한으로 줄여 최적화시켜 기존 데이터를 학습시키는 과정에서 생기는 오차율을 0에 가깝게 만들어 학습율의 정확도를 높여줘야 한다.

|  |
| --- |
| Cost = -tf.reduce\_mean(Y \* tf.log(hypothesis) + (1-Y) \* tf.log(1-hypothesis))  a = tf.Variable(0.01) #learning rate  Optimizer = tf.train.GradientDescentOptimizer(a)  train = Optimizer.minimize(Cost) |

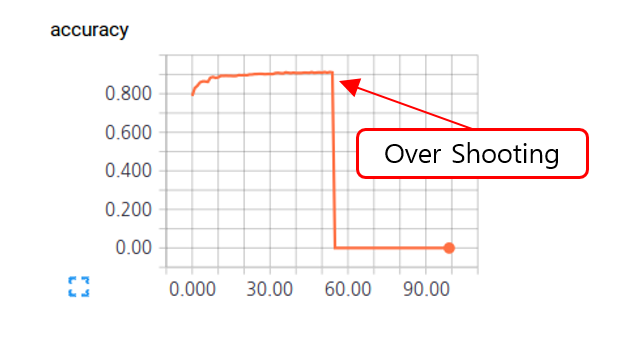
[그림 3-12] Cost값 낮추기

[그림 3-12]를 보면 Cost값을 최소한으로 최적화해주는 GradientDescentOptimizer() 함수에서 Learning rate 값을 설정할 수 있다.



[그림 3-13] 학습빈도에 따른 Cost 값 변화

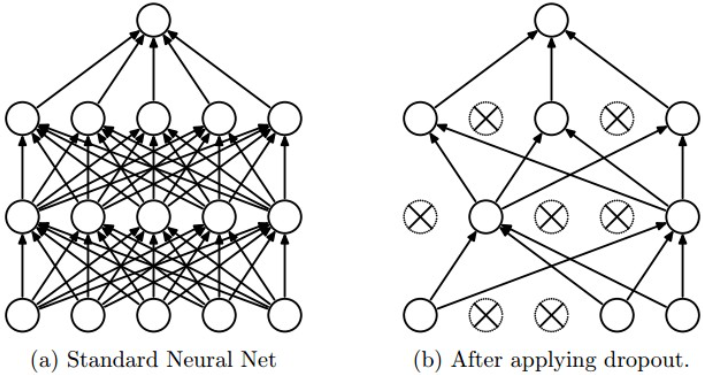
[그림 3-13]의 그래프처럼 학습빈도가 많아질수록 Cost값이 떨어지는 것을 볼 수 있다.



[그림 3-14] 학습빈도에 따른 Accuracy값 변화

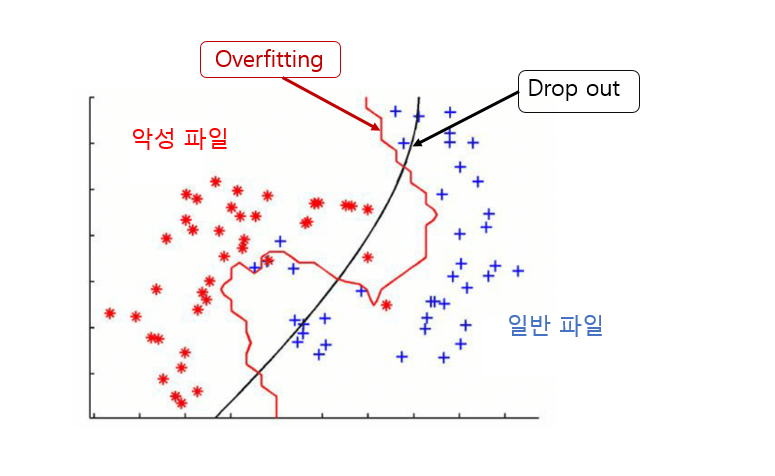
[그림 3-14]의 그래프처럼 학습시키는 정확도는 점차적으로 올라가는 것을 확인할 수 있다. 그리고 일정 정확도를 유지하다가 Overshooting이 발생하여 값이 0으로 떨어지는 것을 확인할 수 있다.

* + 1. **Overfitting 방지를 위한 Dropout 값 조절**



[그림 3-15] Dropout 을 사용한 알고리즘 변화

같은 데이터를 가지고 학습을 반복하면, 가지고 있는 데이터에 대한 학습율이 높아지면서 Learning data에만 맞춰지게 되어서 학습율은 높아지지만 학습하지 않은 데이터를 넣는 경우에는 탐지율이 떨어지게 되는데 이런 현상을 Overfitting이라고 한다. Layer 수와 Node 수가 많은 알고리즘에서는 Overfitting이 발생하기 쉽다. 이러한 경우에는 Dropout 값 설정을 통해서 Overfitting의 발생하는 것을 방지할 수 있는데 [그림 3-15]의 알고리즘과 같이 Dropout을 설정하면 몇 개의 Node 수를 생략해서 Output data를 추출하는 모습을 볼 수 있다.



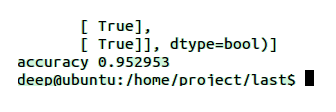
[그림 3-16] Dropout 여부에 따른 Hypothesis 그래프 변화

[그림 3-16]의 그래프는 같은 학습 환경에서 Dropout을 적용한 경우(검은 곡선)와 적용하지 않은 경우(빨간 곡선)에 대한 Hypothesis 모습이다. Dropout을 통해서 Hypothesis를 유연하게 만들어 주는 것을 확인 할 수 있고 이는 Test data의 탐지율을 높이는 결과를 가져올 수 있다. 결과적으로 Dropout은 학습율은 조금 낮추지만 Test data의 탐지율을 높여주는 기능을 수행한다.

|  |
| --- |
| drop\_rate = 0.7  \_L4 = tf.nn.relu(tf.matmul(L3, w3) + b3)  L4 = tf.nn.dropout(\_L4, drop\_rate) |

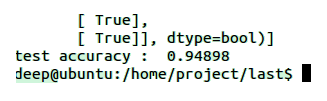
**[그림 3-17] Dropout 설정**

[그림 3-17]은 Dropout을 설정하여 각각의 Layer에 미리 설정해둔 Dropout rate를 부여해주는 코드이다.



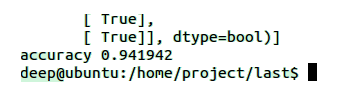
[그림 3-18] Dropout 없는 경우 데이터 학습율

[그림 3-18]은 Dropout이 없을 때 150번을 학습한 결과이다. 학습율이 약 95%로 나타난 것을 확인할 수 있다.

****

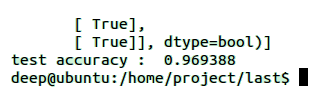
**[그림 3-19] Dropout 없는 경우 데이터 테스트 결과**

[그림 3-19]는 그에 대한 Test 파일의 탐지율이다. 그 결과 약 94%의 탐지율이 나온 것을 볼 수 있다.



[그림 3-20] Dropout 있는 경우 데이터 학습율

[그림 3-20]은 Dropout rate를 0.6으로 준 상태에서 150번을 학습한 결과이다. 약 94%의 학습율이 나온 것을 볼 수 있다.



[그림 3-21] Dropout 있는 경우 데이터 테스트 결과

[그림 3-21]은 150번 학습 후의 Test 파일의 탐지율이다. 약 96%의 탐지율이 나온 것을 볼 수 있다.

이와 같은 결과 값들을 통해 Dropout을 설정한 경우가 하지 않는 경우보다 학습율이 조금 떨어지지만 Test 결과가 더 높게 나오는 것을 알 수 있다.

* 1. **정적 분석 데이터 학습 및 저장**
     1. **정적 분석 데이터 학습시키기**

|  |
| --- |
| for step in xrange(start,start+50):  sess.run(train, feed\_dict={X: x\_data, Y: y\_data, V: v\_data, drop\_late: d\_rate}) |

[그림 3-22] For문을 이용한 데이터 학습

정적 분석 데이터를 효과적으로 학습시켜 이상적인 결과 값을 찾으려면 [그림 3–22]와 같이 for문을 이용한 반복 학습이 필요하다.

* + 1. **학습된 Weight 값 저장 및 불러오기**

|  |
| --- |
| global\_step.assign(step).eval()  saver.save(sess, ckpt\_dir + "/save.ckpt", global\_step=global\_step) |

[그림 3-23] 학습된 Weight 값 저장

[그럼 3-23]은 학습이 모두 끝난 후에 Weight 값을 “save.ckpt” 파일 형식으로 저장하는 코드이다.

|  |
| --- |
| ckpt\_dir = “./save path”  saver = tf.train.Saver()  ckpt = tf.train.get\_checkpoint\_state(ckpt\_dir)  if ckpt and ckpt.model\_checkpoint\_path:  saver.restore(sess, ckpt.model\_checkpoint\_path) |

[그림 3-24] 학습된 Weight 값 불러오기

[그림 3–24]는 파일의 저장경로를 지정하고 경로에 저장파일이 있으면 저장파일을 불러와서 실행시키는 코드이다.

* + 1. **학습 데이터 추가 기능**

|  |
| --- |
| f2 = open(TestFileName)  data = f2.read()  f3 = open(filename, 'a')  f3.write(data)    f2.close()  f3.close() |

[그림 3-25] 데이터 추가 기능

[그림 3-25]는 학습용 데이터를 추가하는 기능이다. 정적 분석된 데이터는 .txt 파일로 저장되기 때문에 write() 함수를 이용해 원하는 데이터를 기존 학습용 데이터에 추가한다.

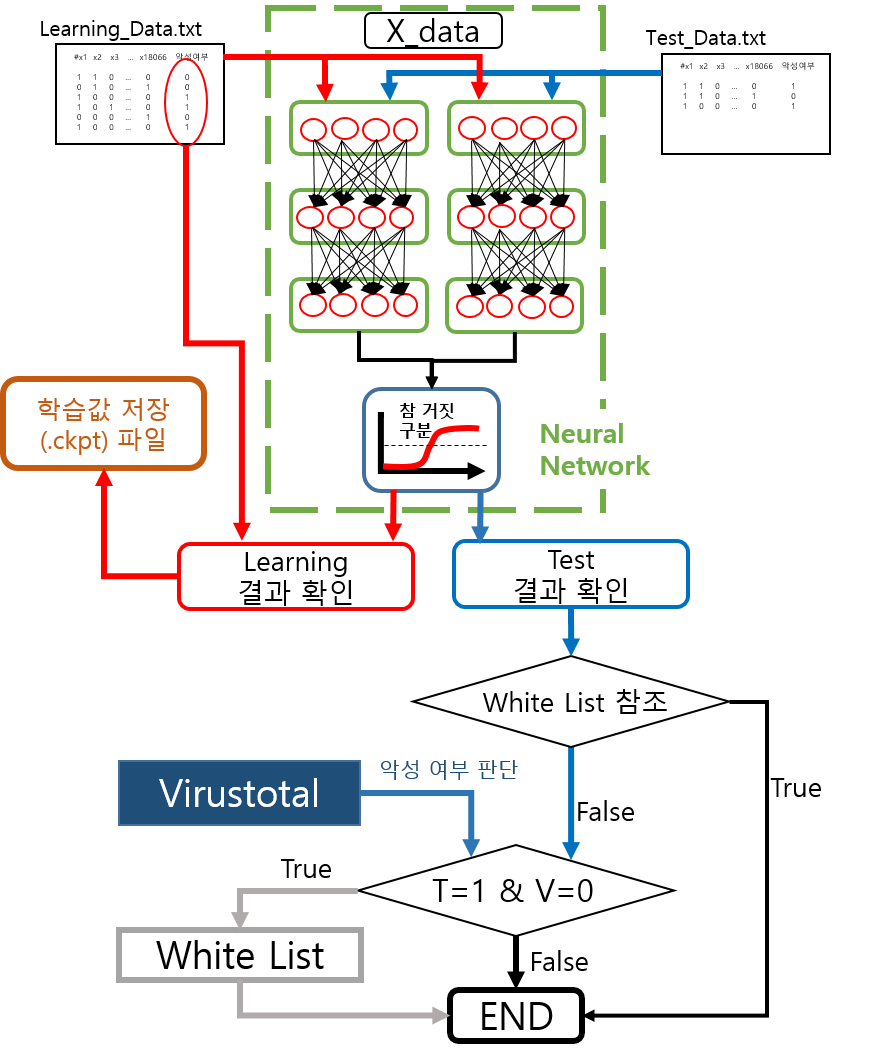
* 1. **데이터 학습 후 White list 작성하기**

|  |
| --- |
| TestList = []  TestList.append(sess.run(tf.floor(hypothesis+0.5),  feed\_dict={V: t\_v\_data, X: t\_x\_data, Y: t\_y\_data, drop\_late:1.0}))  i=0  for l in L:  hashlist.append(int(VT\_Request(api , hash\_check[i])))  if int(TestList[0][i]) == 1 and hashlist[i] == 0:  if hash\_check[i] in white\_list:  i=i  else:  white = open('/home/project/0909/white\_list.txt' ,'a')  white.write(hash\_check[i]+'\t'+L[i]+'\n')  white.close()  i = i+1 |

[그림 3-26] White List 작성

[그림 3-26]은 White List를 생성하는 코드이다. Test List는 악성코드 판별 결과 값을 저장하는 튜플이다. 또한 VirusTotal결과 값인 VT\_Request와 Test List의 결과값을 비교하여 일반파일에 대한 오탐시 White List에 파일의 이름과 Hash값을 저장하고 차후 탐지시 White List를 참조하여 오탐율을 낮춘다.

* 1. **딥러닝 구성도**



[그림 3-27] 딥러닝 전체 구성도

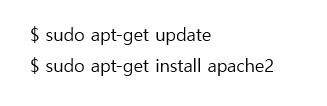
[그림 3-27]은 딥러닝의 학습과정을 순서대로 나타낸 구성도이다. 정적 분석으로 추출한 데이터(Learning Data.txt)를 Neural Network 알고리즘을 통해서 학습시키고, 학습된 값은 (.ckpt)파일로 저장한다. 저장된 학습 값을 통하여 테스트 파일을 검사할 때 사용한다.

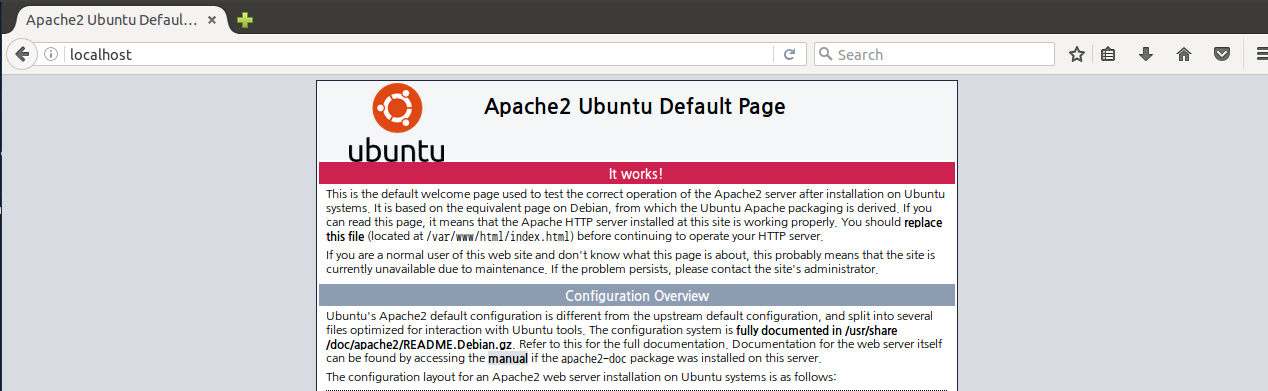
이 때 VirusTotal을 참조하여 VirusTotal(V)이 0(일반파일), Test(T) 결과가 1(악성파일)이 나온 경우에는 White List에 추가한다. White List는 추후에 다른 파일을 테스트할 때 참조 값으로 사용한다.

1. **웹 기반 학습 및 분석 환경 구축**
   1. **Apache2 + php5 환경 구축**

Windows등의 다른 OS와는 달리 Ubuntu에서 Apache2와 php5의 설치는 간단하다. 몇 가지 명령어를 통해 설치를 진행한다.

* + 1. **Apache2 설치**

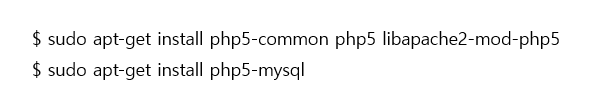


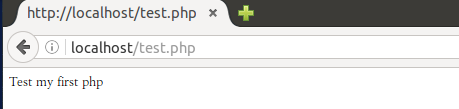


[그림 4-1] Apache2 설치

[그림 4–1] 위의 명령어를 사용하면 [그림 4-1] 아래와 같이 Apache2 의 페이지가 정상적으로 출력되는 것을 확인할 수 있다.

* + 1. **php5 설치**

****



[그림 4-2] Php5 설치

[그림 4–2] 위의 명령어를 사용하면 [그림 4-2] 아래와 같이 php5 의 페이지가 정상적으로 출력되는 것을 확인할 수 있다.

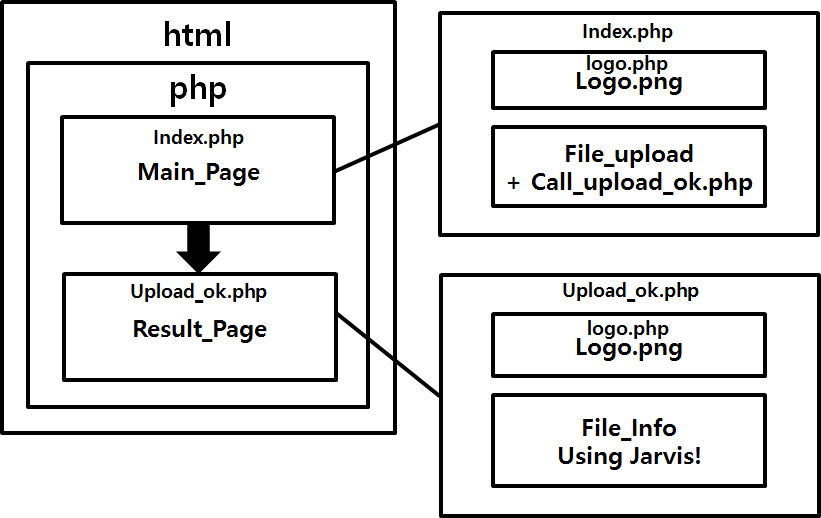
* 1. **웹 사이트 디자인 및 구축**
     1. **웹 사이트 로고 디자인**



[그림 4-3] 웹 사이트 로고 디자인

웹 페이지명은 ‘진리’ 라는 뜻을 가진 ‘Veritas’이다. 구체적으로는 진리, 성실, 진상, 참이라는 뜻으로, 확인하고자 하는 파일의 악성코드 진상을 파악해준다는 의미에서 나온 이름이다.

* + 1. **웹 사이트 구성 제작**



[그림 4-4] 웹 사이트 구성도

다음은 [그림 4–4]와 같이 웹사이트 구성도를 제작한 단계이다.

Main으로 사용한 툴은 Tool HTML(HyperText Marking Language)에 PHP(Personal Hypertext Preprocessor)의 모듈을 넣은 것이다. PHP를 고른 이유는 C언어 문법과 가장 유사하기 때문이다. HTML의 역할은 Web site의 기본 틀을 구성하고 PHP로 내용을 출력할 수 있도록 Table을 만들어준다. 그리고 PHP의 역할은 C언어와 유사하다는 말대로 함수를 구현하거나 다양한 프로그래밍 기법을 사용할 수 있다. 그래서 다른 PHP를 불러오거나, Deep Learning System에서 세부내용을 출력하는 것 등의 기능을 구현할 수 있다.

구체적인 Page별 기능을 소개하자면, Main\_page는 index.php이고 Upload\_ok.php를 호출한다. 이는 파일을 Upload하고 나서 Uploaded File을 처리하기 위함이다. Upload\_ok\_page는 Uploaded File을 Deep Learning System으로 정적 분석하여 악성코드 인지 아닌지 판별 후 출력한다. 이 부분이 메인 기능으로써 Table을 통해 가독성을 높였다.

* + 1. **메인 페이지 구축**

|  |
| --- |
| /\* index.php \*/  <html>  <body>  <div id="wrap">  <div id="header" align=center >  <?php include "./lib/logo.php"; ?>  </div>  <div id="content" align=center >  <head><meta http-equiv="Content-Type" content="text/html; charset=utf-8"></head>  <form enctype='multipart/form-data' action='upload\_ok.php' method='post'>  <input type='file' name='myfile'>  <button>SEND</button>  </form>  </div>  </body>  </html> |

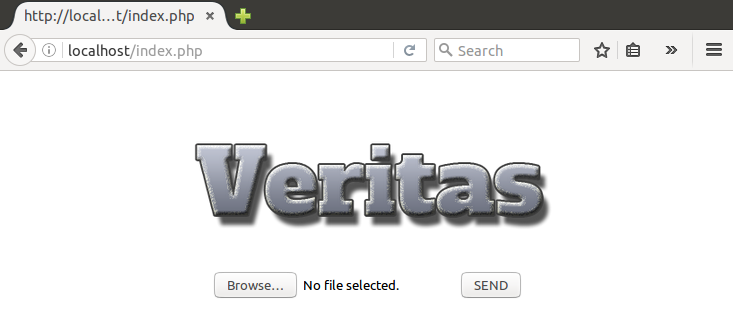
[그림 4-5] 메인 페이지 구축

[그림 4–5]를 보면 Index.php의 메인 페이지 기본 구성은 [로고]–[Browse]–[Send] 세 단계로 구성되어있다. [로고] logo.php로 화면 상단 부분에 출력하고 클릭 시 새로고침이 되는 기능을 수행한다. 그리고 [Browse] 버튼을 눌러 컴퓨터 내의 파일을 페이지에 upload한다. 또한 [SEND] 버튼을 누르면 upload\_ok.php와 연결되어 웹페이지에 upload한 파일을 정적 분석 및 Deep Learning한 결과를 출력한다.

|  |
| --- |
| /\* Logo.php \*/  <div id="logo" >  <a href="./index.php">  <img src="./logo.png" border="0"/>  </a>  </div> |

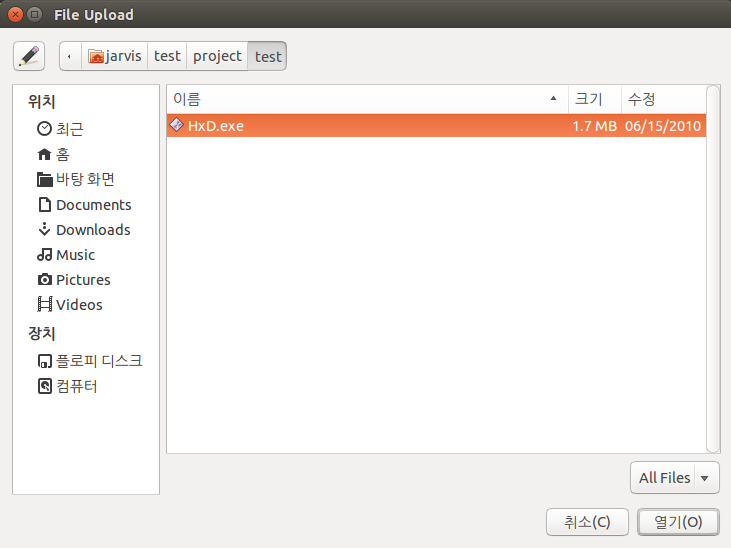
[그림 4-6] Logo.php

[그림 4–6]은 logo.png를 불러 출력하고 클릭 시 index.php로 새로고침을 할 수 있는 기능을 보여준다.

****

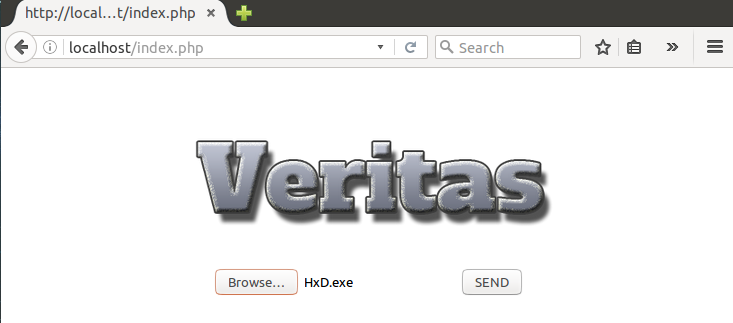
[그림 4-7] 메인 사이트 초기 화면

[그림 4–7]의 메인 사이트 초기 화면 상태에서 Browse를 클릭할 수 있다.

****

[그림 4-8] Browse 버튼 클릭 시 화면

[그림 4–8]의 화면과 같이 출력된 화면에서 악성코드 여부를 확인하고 싶은 파일을 선택할 수 있다.

****

[그림 4-9] 파일이 Upload 된 화면

다음 [그림 4–9]와 같이 Upload 된 파일명을 확인할 수 있다.

* + 1. **파일 업로드 페이지 구축**

|  |
| --- |
| /\* upload\_ok.php\*/  <head><meta http-equiv="Content-Type" content="text/html; charset=utf-8"></head>  <body>  <!-- 웹페이지 logo 출력 -->  <div id="wrap">  <div id="header"align=center >  <?php include "./lib/logo.php"; ?>  </div> |

[그림 4-10] 파일 업로드 페이지 1

[그림 4–10]을 보면 upload\_ok.php의 메인 페이지 기본 구성은 [로고]–[Table1]–[Table2] 순서대로 구성되어있다. [로고] logo.php는 화면 상단 부분에 출력하고 클릭 시 새로고침이 되는 기능을 수행한다.

|  |
| --- |
| <!--테이블 1 생성 -->  <table width='600' align='center'>  <tr bgcolor='#ccccc' align='center'><td> 파일명</td><td>확장자</td><td>파일형식</td><td>파일크기</td>  <?php    // 설정  $uploads\_dir = '/test';  $allowed\_ext = array('exe','dll','EXE','DLL');    // 변수 정리  $error = $\_FILES['myfile']['error'];  $name = $\_FILES['myfile']['name'];  $ext = array\_pop(explode('.', $name));    // 오류 확인  if( $error != UPLOAD\_ERR\_OK ) {  switch( $error ) {  case UPLOAD\_ERR\_INI\_SIZE:  case UPLOAD\_ERR\_FORM\_SIZE:  echo "파일이 너무 큽니다. ($error)";  break;  case UPLOAD\_ERR\_NO\_FILE:  echo "파일이 첨부되지 않았습니다. ($error)";  break;  default:  echo "파일이 제대로 업로드되지 않았습니다. ($error)";  }  exit;  }  // 확장자 확인  if( !in\_array($ext, $allowed\_ext) ) {  echo "허용되지 않는 확장자입니다. [.exe/.dll 파일]만 올려주시기 바랍니다.";  exit;  }    // 파일 이동  move\_uploaded\_file( $\_FILES['myfile']['tmp\_name'], "$uploads\_dir/$name");  // 파일 정보 출력 1  echo ("<tr><td width='120' align='center'>$name</td><td align='center'>$ext</td><td width='200' align='center'>{$\_FILES['myfile']['type']}</td><td width='100' align='center'>{$\_FILES['myfile']['size']}</td></tr>");  system("python /home/project/0909/920.py /test -c /result/909.txt 2 1 > /result/abc.txt");  system("rm /test/\*");  ?> |

[그림 4-11] 파일 업로드 페이지 2

[그림 4-11]을 보면 index.php에서 Upload한 파일을 정적 분석(Static Analyization) 및 딥러닝(Deep Learining)하여 파일의 분석 내용을 웹페이지로 출력해 사용자에게 제공한다. [Table1]은 파일의 기본적인 정보를 알려준다. [파일명/확장자/파일형식/파일크기]를 출력한다.

|  |
| --- |
| <!--테이블 2 생성 -->  <table width='600' align='center'>  <tr bgcolor='#ccccc' align='center'><td>파일 상세 정보</td></tr>  <tr><td>  <?php  $tdcount = 2; $numtd =2;  print "<table width='780' align='center'>";  $f = fopen("/result/abc.txt", "r");  while (!feof($f)) {  $arrM = explode(" ",fgets($f));  $row = current( $arrM );  if ($tdcount == 1) //$tdcount=1이면  print "<tr align='center'>"; print "<td align='center'>$arrM[0]</td> <td align='center'> $arrM[1]</td><td align='center'>$arrM[2]</td><td align='center'>$arrM[3]</td>";    if ($tdcount == $numtd) {  print "</tr>";  $tdcount = 1;  }  }  if ($tdcount!= 1) {  while ($tdcount <= $numtd) {  print "<td>&nbsp;</td>"; $tdcount++;  } print "</tr>";  }  print "</table>";  ?>  </td></tr></table>  </body> |

[그림 4-12] 파일 업로드 페이지 3

[그림 4-12]의 [Table2]는 해당 파일의 Section마다 VirtualAddress, Virtual Size, SizeOfRawData의 값들을 출력한다. 그 아래는 좀 더 구체적으로 정적 분석에 대한 결과 값을 보여준다.

① Dll check : 해당 파일이 Dll인지 아닌지 확인 (O/X)

② Dos Header Check : Dos Header가 수정이 되었는지 안되었는지를 확인 (Original/Modified)

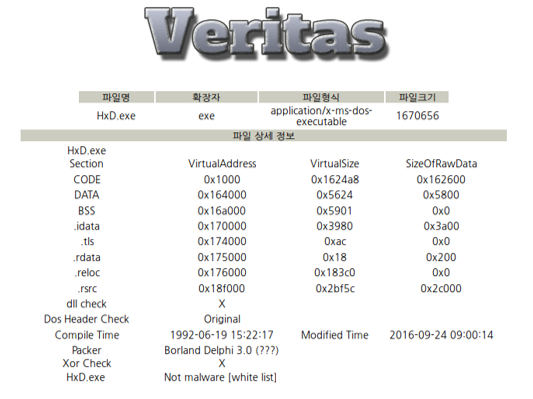
③ Compile Time : 처음 컴파일 된 시간

④ Modified Time : 수정 시간

⑤ Packer : Packing 할 때 무엇이 쓰였는지를 출력

⑥ XOR Check : XOR 가 된 파일인지 아닌지를 검사

⑦ Result : 파일의 악성코드 여부 최종 판결 (Not Malware / Malware)



[그림 4-13] Veritas 웹페이지 Upload 출력 결과

[그림 4–13]을 보면 악성코드 여부를 확인하고 싶은 파일을 Veritas 웹페이지에 Upload 하면 어떠한 결과 값이 출력되었는지 각각의 세부 정보를 확인 할 수 있다.

1. **악성코드 탐지 테스트** 
   1. **알고리즘 환경에 따른 테스트 결과**

본 테스트는 가장 효과적인 악성코드 탐지 알고리즘을 판별하기 위해 다양한 조건을 가지고 진행하였다. 조건의 기준점으로는 Layer의 구성 방법, Node의 개수, Dropout의 여부가 있으며 총 8 가지 조건으로 테스트하여 각각의 학습율을 확인하였다. 또한 학습하지 않은 300여개의 악성코드 Testdata와 100여개의 일반파일 Testdata를 약 100개씩 나누어 총 4번의 테스트를 통해서 평균 탐지율을 측정했다.

테스트 환경 = Feature : 18,066 / Number of Data : 2,000

1. 조건 1: 기본 Neural Network 알고리즘 구성, Node시작= 5000, Dropout = 0.6
2. 조건 2: 기본 Neural Network 알고리즘 구성, Node시작= 1500, Dropout = 0.6
3. 조건 3: 기본 Neural Network 알고리즘 구성, Node시작= 5000, Dropout = X
4. 조건 4: 기본 Neural Network 알고리즘 구성, Node시작= 1500, Dropout = X
5. 조건 5: 병렬 Neural 알고리즘 구성, Node시작= 5000, Dropout = 0.6
6. 조건 6: 병렬 Neural 알고리즘 구성, Node시작= 1500, Dropout = 0.6
7. 조건 7: 병렬 Neural 알고리즘 구성, Node시작= 5000, Dropout = X
8. 조건 8: 병렬 Neural 알고리즘 구성, Node시작= 1500, Dropout = X

[그림 5-1]부터 [그림 5–8]까지는 악성코드 탐지를 가장 효과적으로 하기 위해 딥러닝 알고리즘을 시뮬레이션한 결과이다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 1. 기본 레이어 구성, Learning rate = 0.05, Dropout = 0.6** | | | | |
| X data Layer Node = 5000 – 1000 – 100 – 1  V data Layer Node = 없음 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 91.99 | 91 | 96 | 96.93 | 87.65 |
| 93.44 | 92 | 97 | 96.93 | 87.65 |
| **94.69** | **93** | **97** | **96.93** | **90.12** |
| 94.84 | 92 | 97 | 97.95 | 90.12 |
| Overshooting | - | - | - | - |

[그림 5-1] 시뮬레이션 조건 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 2. 기본 레이어 구성, Learning rate = 0.05, Dropout = 0.6** | | | | |
| X data Layer Node = 1500 – 500 – 30 – 1  V data Layer Node = 없음 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 90.54 | 90 | 97 | 94.89 | 87.65 |
| 92.14 | 91 | 97 | 96.93 | 88.88 |
| 93.04 | 91 | 97 | 96.93 | 87.65 |
| 94.04 | 91 | 97 | 96.93 | 87.65 |
| **94.39** | **92** | **97** | **96.93** | **90.12** |

[그림 5-2] 시뮬레이션 조건 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 3. 기본 레이어 구성, Learning rate = 0.05, Dropout = X** | | | | |
| X data Layer Node = 5000 – 1000 – 100 – 1  V data Layer Node = 없음 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 93.34 | 92 | 98 | 97.95 | 86.41 |
| **94.94** | **94** | **98** | **97.95** | **87.65** |
| 95.24 | 93 | 98 | 95.91 | 91.35 |
| 96.34 | 95 | 95 | 94.89 | 91.35 |
| 97.09 | 94 | 94 | 94.89 | 91.35 |

[그림 5-3] 시뮬레이션 조건 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 4. 기본 레이어 구성, Learning rate = 0.05, Dropout = X** | | | | |
| X data Layer Node =1500 – 500 – 30 – 1  V data Layer Node = 없음 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 93.34 | 92 | 97 | 96.93 | 86.41 |
| 94.64 | 93 | 98 | 97.95 | 86.42 |
| **95.04** | **93** | **98** | **97.95** | **90.12** |
| 95.59 | 94 | 98 | 94.89 | 91.35 |
| 97.04 | 95 | 94 | 93.87 | 91.34 |

[그림 5-4] 시뮬레이션 조건 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 5. 병렬 레이어 구성, Learning rate = 0.05, Dropout = 0.6** | | | | |
| X data Layer Node = 5000 – 1000 – 30 – 1  V data Layer Node = 없음 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 91.19 | 92 | 97 | 95.91 | 86.41 |
| 92.84 | 91 | 97 | 96.93 | 87.65 |
| 93.94 | 92 | 98 | 96.93 | 87.65 |
| **94.69** | **93** | **97** | **97.95** | **90.12** |
| 95.29 | 93 | 96 | 94.89 | 91.35 |

[그림 5-5] 시뮬레이션 조건 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 6. 병렬 레이어 구성, Learning rate = 0.05, Dropout = 0.6** | | | | |
| X data Layer Node = 1500 – 500 – 30 – 1  V data Layer Node = 60 – 60-60 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 91.57 | 91 | 96 | 96.93 | 87.65 |
| 92.96 | 92 | 97 | 96.93 | 88.88 |
| 94.12 | 92 | 97 | 97.95 | 88.88 |
| **94.98** | **93** | **97** | **97.95** | **91.35** |
| 95.36 | 93 | 97 | 95.91 | 91.35 |

[그림 5-6] 시뮬레이션 조건 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 7. 병렬 레이어 구성, Learning rate = 0.05, Dropout = X** | | | | |
| X data Layer Node = 5000 – 1000 – 30 – 1  V data Layer Node = 60 – 60-60 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 91.69 | 91 | 97 | 96.93 | 87.65 |
| 92.99 | 92 | 97 | 97.95 | 87.65 |
| 94.24 | 93 | 97 | 97.95 | 87.65 |
| 94.89 | 93 | 96 | 98.97 | 90.12 |
| **95.04** | **94** | **97** | **98.97** | **90.12** |

[그림 5-7] 시뮬레이션 조건 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **조건 8. 병렬 레이어 구성, Learning rate = 0.05, Dropout = X** | | | | |
| X data Layer Node = 1500 – 500 – 30 – 1  V data Layer Node = 60 – 60-60 | | | | |
| 학습율 | TestData 1 | TestData 2 | TestData 3 | TestData 4 |
| 93.99 | 92 | 98 | 96.93 | 87.66 |
| 94.94 | 94 | 97 | 97.95 | 90.12 |
| **95.24** | **95** | **98** | **97.95** | **90.12** |
| 95.39 | 94 | 97 | 94.89 | 90.12 |
| 95.44 | 94 | 97 | 94.89 | 91.35 |

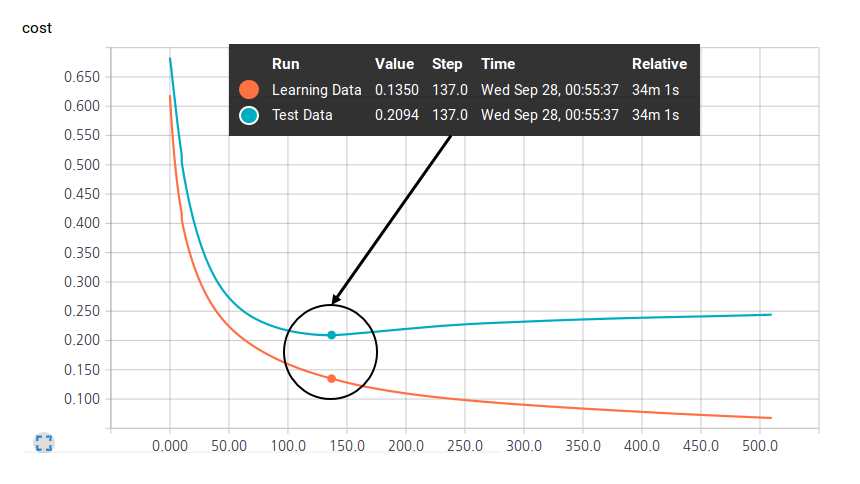
[그림 5-8] 시뮬레이션 조건 8

* 1. **최종 테스트 결과 및 분석**

|  |  |  |
| --- | --- | --- |
| **조건** | **최적 학습율** | **최적 테스트 평균** |
| 조건 1 | 94.69 | 94.26 |
| 조건 2 | 94.39 | 94.01 |
| 조건 3 | 94.94 | 94.4 |
| 조건 4 | 95.04 | 94.76 |
| 조건 5 | 94.69 | 94.51 |
| 조건 6 | 94.98 | 94.82 |
| 조건 7 | 95.04 | 95.02 |
| **조건 8** | **95.24** | **95.26** |

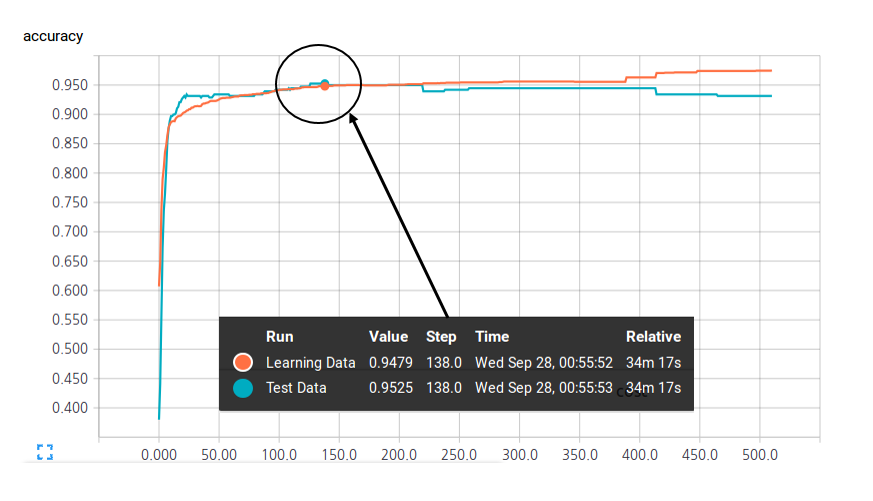
[그림 5-9] 최종 테스트 결과

학습하지 않은 악성코드 300 여개와 일반파일 100 여개를 조건에 따라 테스트한 결과 평균적으로 95%의 정확도가 나타나는 것을 확인할 수 있었다. 그 중에서 Layer를 병렬로 주고 Dropout을 사용하지 않은 ‘조건 8’ 경우의 테스트 결과가 95.26%로 가장 높게 나타나는 것이 확인되었다.



[그림 5-10] 조건 8의 학습 데이터 및 테스트 데이터 Cost

가장 높은 결과값을 가진 ‘조건 8’의 Cost변화를 나타낸 그래프이다. 130회 이상의 학습부터 Test Data의 Cost가 점차 올라가는 것을 볼 수 있다. 이 구간부터 Overfitting이 발생하였고 Test Data의 그래프에서 가장 낮은 Cost값을 가지는 부분이 최적화된 학습환경이라는 것을 알 수 있다.



[그림 5-11] 조건 8의 학습율 및 탐지율

[그림 5-11]은 ‘조건 8’의 최고 탐지율 위치를 나타낸 그래프이다. 표시된 지점을 기점으로 Test Data의 탐지율이 Overfitting으로 인해 조금씩 감소하는 것을 확인할 수 있다.

결과를 살펴보면 위의 그래프처럼 Overfitting이 발생하는 것을 줄이기 위해 Dropout을 사용하지만 ‘조건 7’의 경우를 살펴보면 Dropout을 사용한 경우에 오히려 탐지율이 더 낮게 나오는 것이 확인되었다. Learning Data의 수가 적고 학습 횟수가 많지 않기 때문에 오히려 학습에 방해가 된 것으로 판단된다.

또한 테스트 환경에서는 병렬 Layer로 Layer수가 4개, Node수가 1500개 일때 가장높은 학습율을 얻었고 Node수가 증가하여도 학습율에는 큰 변화가 없었다.

1. **결론**
   1. **성과**

본 프로젝트에서는 기계학습을 이용하여 악성파일 탐지 방법을 제시하였다. 이 방법은 정적 분석을 이용한 API 함수와 PE Header 등을 활용하여 특징들을 추출할 수 있으며, 약 2000개의 학습 데이터를 이용하여 학습 환경을 구성하였다.

부족한 학습 데이터의 오탐, 미탐을 보완하기 위해 Virustotal의 결과 값도 함께 참조한다. 학습 결과 중 오탐이 발생 하는 것은 해당 파일의 Hash 값을 추출해 White List를 생성하여 추가하였다. 해당 파일을 다시 Test할 경우에는 작성된 List를 참조하도록 구현함으로써 오탐율을 줄였다.

학습하지 않은 악성코드 300 여개와 일반파일 100 여개를 테스트한 결과 95.26%의 결과값을 얻었으며 ‘6.2.1 정적 분석’을 고려하였을 때 이는 만족할 만한 결과이다. 또한 알고리즘 구성시 병렬 Layer를 구성하며 Learning Data의 개수에 따라 Dropout을 사용하는 것으로 높은 결과값을 얻을 수 있다.

또한 정적 분석만을 활용하기 때문에 동적 검사에 의해 파일이 실행되는 위험을 제거할 수 있었고 덕분에 안전한 파일 검사 환경 구현이 가능했다. 이러한 특징 값은 기본적인 보안 솔루션의 발전에 도움이 될 것이다.

본 프로젝트를 통해서 기계학습이라는 새로운 분야를 접할 수 있었고 기본적인 학습 프로그램의 구현 방법과 여러 기능들을 배울 수 있었다. 딥러닝을 구현하기에는 데이터의 수가 부족했던 것이 아쉽지만 앞으로 보완해 나갈 수 있다고 판단된다.

* 1. **문제점 및 개선 방안**
     1. **정적 분석**

**정적 분석은 일반적으로 사용되는 기술이다. 실제 프로그램을 실행시키지 않고, 프로그램의 기능을 파악하기 위해 코드나 프로그램의 구조를 분석하는 과정이다. 하지만 프로그램을 실행하지 않기에 정확한 목적을 알 수 없어서 악성코드 판별에 한계가 있다. 예를 들어, 해쉬 값을 이용한 VirusTotal과 같이 많은 백신들의 엔진을 통해서 결과를 얻는다고 해도 악성코드가 해쉬 값을 변조한다면 신뢰할 수 없다. 또한 쉘코드를 입힌 putty 같은 악성파일은 더더욱 판별하기 힘들다. 그렇기 때문에 정적 분석으로는 신종 악성코드를 잡는데에는 한계가 있다.**

* + 1. **API 함수 기준**

**우리는 의심가는 API가 포함되어 있으면 악성코드로 의심하는 가정하에 탐지를 시도하였다. 예를 들어, UAC를 우회하기 위해서 자주 사용되는 함수로써** 최대한 높은 권한 수준으로 응용 프로그램이 실행되는 highestAvailable() 함수와응용 프로그램을 시작한 프로세스와 동일한 권한으로 응용 프로그램이 실행되는 **anInvoker() 함수 등이 있다. 이런 함수들은 일반 파일에서 거의 포함되지 않았기 때문에 악성 파일들이 주로 사용하는 함수를 리스트화 하여 악성코드를 분석하는데 사용하여 탐지하였다. 그렇다 보니 일반 파일이어도 의심되는 API 함수가 많으면 악성코드로 판단하는 단점이 생겼다.**

**대표적으로 HxD가 있다. HxD는 파일 분석 유틸리티이며 HDD를 손쉽게 접근하여 내부를 들여볼 수 있으며 파일의 정보를 수정하거나 하드코딩 하는 등의 다양한 악성 행위를 가능하게 할 수 있는 윈도우 API를 가지고 있다. 그렇기 때문에 일반 파일을 악성 파일로 판별하는 문제가 있다.**

* + 1. **Deep Learning 환경 구축**

현재 딥러닝의 Learning Data 의 수는 약 2,000개 인데 이는 학습을 하기에는 많은 수는 아니다. 그렇기 때문에 추후에 Learning Data가 많아지게 되면 학습율 및 탐지율이 낮아지게 될 가능성이 있다. 좀 더 신뢰할 수 있는 학습율을 구하기 위해서는 데이터 수의 증가와 Layer 및 Node 수 조절을 통한 학습 환경 최적화와 Relu와 Otimizer 함수의 변화를 통한 개선방법이 있다.

1. **참고자료**

**서적**

1. 이승원 / [리버싱 핵심원리] / 인사이트 / (2013. 10. 07)
2. 정재준 / [자료구조 알고리즘 머신러닝 상세분석] / 커널연구회 / (2016.05.06)
3. 마이클 시코스키, 앤드류 호닉 / [실전 악성코드와 멀웨어 분석] / 에이콘출판 / (2013.10.29)
4. 황재호 / [PHP 프로그래밍 입문: 기초부터 다지는 웹 사이트 제작 실습] / 한빛아카데미 / (2013.05.30)
5. 조정원, 최우석, 이도경, 정지훈 / [파이썬 오픈소스 도구를 활용한 악성코드 분석] / 에이콘출판 / (2015.09.30)

**Site**

1. Google, peframe 오픈소스 [https://code.google.com/p/peframe]
2. Github, Viper 오픈소스 [https://github.com/viper-framework/viper]
3. Google, VirusTotal [https://www.virustotal.com/]
4. Youtube, TensorFlow 강의 [http://hunkim.github.io/ml/]
5. Google, TensorFlow 오픈소스 [https://www.tensorflow.org/]
6. Facebook, TensorFlow KR 페이지 [https://www.facebook.com/groups/TensorFlowKR]
7. **부록**

**아래 내용은 악성코드에서 주로 사용되는 API 함수의 세부정보이다.**

**■ OpenProcess()**

|  |  |  |
| --- | --- | --- |
| 설명 | 프로세스 아이디(PID)에 대한 프로세스 핸들을 구한다 | |
| Syntax | HANDLE WINAPI OpenProcess(  \_In\_ DWORD dwDesiredAccess,  \_In\_ BOOL bInheritHandle,  \_In\_ DWORD dwProcessId  ); | |
| 매개변수 | DWORD dwDesiredAccess | 프로세스의 접근 권한 |
| BOOL blnheritHandle | 핸들 상속 여부 |
| DWORD dwProcessId | 핸들을 얻고자 하는 프로세스의 PID |
| 반환 | 성공 했을 경우 특정 프로세스의 핸들을, 실패하면 NULL을 반환한다 | |
| 헤더 | WinBase.h (include Windows.h) | |

**■ VirtualAllocEx()**

|  |  |  |
| --- | --- | --- |
| 설명 | 준비 또는 지정된 프로세스의 가상 주소 공간의 메모리 영역을 불러온다. | |
| Syntax | LPVOID WINAPI VirtualAllocEx(  \_In\_ HANDLE hProcess,  \_In\_opt\_ LPVOID lpAddress,  \_In\_ SIZE\_T dwSize,  \_In\_ DWORD flAllocationType,  \_In\_ DWORD flProtect  ); | |
| 매개변수 | hProcess | 프로세스에 대한 핸들 |
| lpAddress | 당신이 할당할 페이지의 영역에 대한 원하는 출발 주소를 지정 포인터 |
| dwSize | 바이트 단위로 할당 된 메모리 영역의 크기 |
| flAllocationType | 메모리 할당 타입 |
| flProtect | 페이지의 영역에 대한 메모리 보호가 할당된다. |
| 반환 | 성공 : 할당 된 페이지 영역의 베이스 Address  실패 : 0(NULL) | |
| 헤더 | WinBase.h | |

**■ WriteProcessMemory()**

|  |  |  |
| --- | --- | --- |
| 설명 | 지정된 프로세스의 메모리 영역에 데이터를 씁니다. | |
| Syntax | BOOL WINAPI WriteProcessMemory(  \_In\_ HANDLE hProcess,  \_In\_ LPVOID lpBaseAddress,  \_In\_ LPCVOID lpBuffer,  \_In\_ SIZE\_T nSize,  \_Out\_ SIZE\_T \*lpNumberOfBytesWritten  ); | |
| 매개변수 | hProcess | 프로세스 메모리에 핸들이 변경된다. |
| lpBaseAddress | 데이터가 기록된 지정된 프로세스의 기본 주소에 대한 포인터. |
| lpBuffer | 데이터를 포함하는 버퍼에 대한 포인터가 지정된 프로세스의 주소 공간에 기록된다. |
| nSize | 바이트 수를 지정된 프로세스에 기록된다. |
| lpNumberOfBytesRead | 지정된 프로세스에 전송 된 바이트 수를받는 변수의 포인터. |
| 반환 | 성공 : 0이 아닌 값  실패 : 0(NULL) | |
| 헤더 | WinBase.h | |

**■ ReadProcessMemory()**

|  |  |  |
| --- | --- | --- |
| 설명 | 실행되고 있는 프로세스 내의 메모리에 데이터를 읽어온다. 완벽한 메모리 영역을 제어하지 않으면, 연산 오류를 유발한다 | |
| Syntax | BOOL WINAPI ReadProcessMemory(  \_In\_ HANDLE hProcess,  \_In\_ LPCVOID lpBaseAddress,  \_Out\_ LPVOID lpBuffer,  \_In\_ SIZE\_T nSize,  \_Out\_ SIZE\_T \*lpNumberOfBytesRead  ); | |
| 매개변수 | HANDLE hProcess | 메모리를 읽기 원하는 프로세스의 핸들 입력 |
| LPCVOID lpBaseAddress | 쓰거나 읽을 메모리 주소의 포인터 |
| LPVOID lpBuffer | 읽은 값을 저장할 버퍼 |
| SIZE\_T nSize | 읽어올 바이트 수. DWORD 형이면 4를 넣고 아니면 sizeof 함수를 사용해도 상관없다 |
| SIZE\_T \*lpNumberOfBytesRead | 완료 후, 전송된 실제 데이터의 크기를 바이트 단위로 얻기 위한 변수의 주소를 지정 (보통 필요 없기 때문에 0&으로 지정) |
| 반환 | 호출 성공 : NULL이 아닌 값  호출 실패 : NULL | |
| 헤더 | WinBase.h (include Windows.h) | |

**■ CreateRemoteThread()**

|  |  |  |
| --- | --- | --- |
| 설명 | 다른 프로세스의 가상 어드레스 공간에서 실행 스레드를 생성한다.  사용 CreateRemoteThreadEx의 다른 프로세스의 가상 어드레스 공간에서 실행 스레드를 생성하는 기능을 선택적으로 확장 속성을 지정한다. | |
| Syntax | HANDLE WINAPI CreateRemoteThread(  \_In\_  HANDLE                 hProcess,  \_In\_  LPSECURITY\_ATTRIBUTES  lpThreadAttributes,  \_In\_  SIZE\_T                 dwStackSize,  \_In\_  LPTHREAD\_START\_ROUTINE lpStartAddress,  \_In\_  LPVOID                 lpParameter,  \_In\_  DWORD                  dwCreationFlags,  \_Out\_ LPDWORD                lpThreadId  ); | |
| 매개변수 | hProcess | 생성된 스레드인 프로세스에 핸들 |
| lpThreadAttributes | SECURITY\_ATTRIBUTES 구조체로 가는 포인터 |
| dwStackSize, | 바이트 스택의 초기 크기. 시스템은 가까운 페이지에 이 값을 반올림한다. 이 인자가 0이면, 새로운 스레드는 실행 기본 크기를 사용한다. |
| lpStartAddress | 응용 프로그램 정의 함수에 대한 포인터 |
| lpParameter | 변수에 대한 포인터는 스레드 함수에 전달 |
| dwCreationFlags | 스레드의 생성을 제어하는 ​​플래그 |
| lpThreadId | 스레드 식별자를받는 변수의 포인터.  이 매개 변수 인 경우 **NULL** , 스레드 식별자는 반환되지 않는다. |
| 반환 | 성공 : 새로운 스레드에 대한 핸들이다.  실패 : 실패하면 반환 값은NULL, 확장 된 오류 정보를 얻으려면, GetLastError를 호출. | |
| 헤더 | WinBase.h | |

**■ bind()**

|  |  |  |
| --- | --- | --- |
| 설명 | 소켓의 로컬 주소를 연결하는 함수 | |
| Syntax | int bind(  \_In\_ SOCKET                s,  \_In\_ const struct sockaddr \*name,  \_In\_ int                   namelen  ); | |
| 매개변수 | s | 언바운드(unbound) 소켓을 식별하는 설명자 |
| name | 바운드 소켓에 할당하는 로컬주소의 소켓구조체를 가르키는 포인터 |
| namelen | 이름 인자가 가르키는 값의 길이 |
| 반환 | 성공 : 0 반환  실패 : SOCKET\_ERROR를 반환하고, 특정 에러 코드를 호출하여 검색 할 수 있다. | |
| 헤더 | Winsock2.h | |

**■ CreateProcess()**

|  |  |  |  |
| --- | --- | --- | --- |
| 설명 | 프로세스를 생성하는 함수 | | |
| Syntax | BOOL WINAPI CreateProcessW(  \_In\_opt\_ LPCWSTR lpApplicationName,  \_Inout\_opt\_ LPWSTR lpCommandLine,  \_In\_opt\_ LPSECURITY\_ATTRIBUTES lpProcessAttributes,  \_In\_opt\_ LPSECURITY\_ATTRIBUTES lpThreadAttributes,  \_In\_ BOOL bInheritHandles,  \_In\_ DWORD dwCreationFlags,  \_In\_opt\_ LPVOID lpEnvironment,  \_In\_opt\_ LPCWSTR lpCurrentDirectory,  \_In\_ LPSTARTUPINFOW lpStartupInfo,  \_Out\_ LPPROCESS\_INFORMATION lpProcessInformation  ); | | |
| 매개변수 | LPCWSTR lpApplicationName | 실행파일의 경로. | |
| LPWSTR lpCommandLine | 커맨드 문자열에 대한 포인터로 프로세스 실행 시 넣어줄 추가 명령 | |
| LPSECURITY\_ATTRIBUTES lpProcessAttributes | 프로세스 생성시의 보안 속성 설정 포인터.  대부분 NULL 설정을 줌 | |
| LPSECURITY\_ATTRIBUTES lpThreadAttributes | 프로세스의 스레드 생성시의 보안 속성 설정 포인터.  대부분 NULL 설정을 줌 | |
| BOOL bInheritHandles | 핸들 상속 여부 플래그.  True : 부모 프로세스의 핸들 테이블을 상속  False : 부모 프로세스의 핸들 테이블을 상속하지 않음 | |
| DWORD dwCreationFlags | 프로세스 생성시의 추가 플래그 | |
| CREATE\_SUSPENDED | 생성될 프로세스의 주 스레드를 SUSPENDED 상태로 생성.  Resume스레드 함수를 실행시키기 전까지 실행이 중지됨 |
| LPVOID lpEnvironment | 환경 블록에 대한 포인터. | |
| LPCWSTR lpCurrentDirectory | 생성되는 프로세스가 인식할 현재 디렉토리 경로 | |
| LPSTARTUPINFOW lpStartupInfo | 프로세스의 시작 정보로 STARTUPINFO 구조체 포인터 | |
| LPPROCESS\_INFORMATION lpProcessInformation | 프로세스의 정보 담긴 PROCESS\_INFORMATION 구조체 포인터 | |
| 반환 | 성공 여부에 따라 True(성공), False(실패) 반환 | | |
| 헤더 | WinBase.h | | |

**■ HttpSendRequest()**

|  |  |  |
| --- | --- | --- |
| 설명 | 발신자가 일반적으로 전달되는 것 이상으로 추가적인 데이터를 전송하도록 허용하는 HTTP 서버에 지정된 요청을 보낼 때 사용하는 함수이다. | |
| Syntax | BOOL HttpSendRequest(  \_In\_ HINTERNET hRequest,  \_In\_ LPCTSTR   lpszHeaders,  \_In\_ DWORD     dwHeadersLength,  \_In\_ LPVOID    lpOptional,  \_In\_ DWORD     dwOptionalLength  ); | |
| 매개변수 | hRequest | 핸들은 호출에 의해 반환 [**HttpOpenRequest의**](https://msdn.microsoft.com/en-us/library/windows/desktop/aa384233(v=vs.85).aspx) 기능 |
| lpszHeaders | 이 포인터 **는 null**추가 헤더를 포함로 끝나는 문자열은 요청에 추가한다. |
| dwHeadersLength | 추가 헤더의 크기, **TCHARs**. |
| lpOptional, | 선택적 데이터를 포함하는 버퍼에 대한 포인터는 요청 헤더 직후에 전송된다. 이 매개 변수는 일반적으로 POST와 PUT 작업에 사용된다. |
| dwOptionalLength | 바이트 옵션 데이터의 크기.  보낼 할 임의의 데이터가 없는 경우,이 인자는 제로 일 수 있다. |
| 반환 | 성공 : TRUE  실패 : FALSE. 오류 정보 : GetLastError 호출 | |
| 헤더 | Wininet.h | |

**■ StartService()**

|  |  |  |
| --- | --- | --- |
| 설명 | 서비스를 시작하는 함수 | |
| Syntax | BOOL WINAPI StartService(  \_In\_     SC\_HANDLE hService,  \_In\_     DWORD     dwNumServiceArgs,  \_In\_opt\_ LPCTSTR   \*lpServiceArgVectors  ); | |
| 매개변수 | hService | 서비스에 대한 핸들 |
| dwNumServiceArgs | lpServiceArgVectors의 배열 내의 문자열의 수 |
| lpServiceArgVectors | 인자로써 서비스하는 ServiceMain 함수로 전달하는 널 종료 문자열. |
| 반환 | 성공 : 0이 아닌 값 반환  실패 : 0 반환 | |
| 헤더 | Winsvc.h | |

**■ GetKeyState()**

|  |  |  |
| --- | --- | --- |
| 설명 | 지정된 가상 키의 상태를 가져온다. | |
| Syntax | SHORT WINAPI GetKeyState(  \_In\_ int nVirtKey  ); | |
| 매개변수 | nVirtKey | 가상 키. 원하는 가상 키가 문자 또는 숫자 (A에서 Z, a에서 z를 통해, 또는 0 ~ 9)이면, nVirtKey은 해당 문자의 ASCII 값으로 설정해야한다. |
| 반환 | 반환 값은, 특정 가상 키의 상태를 지정  상위 비트가 1이면, 키는 아래이고 그렇지 않으면 업이다.  하위 비트가 1인 경우, 키는 토글된다. | |
| 헤더 | Winuser.h | |

**■ InternetReadFile()**

|  |  |  |
| --- | --- | --- |
| 설명 | InternetOpenUrl, FtpOpenFile 또는 HttpOpenRequest에 의해 열린 핸들에서 데이터를 읽어오는 기능의 함수이다. | |
| Syntax | BOOL InternetReadFile(  \_In\_  HINTERNET hFile,  \_Out\_ LPVOID    lpBuffer,  \_In\_  DWORD     dwNumberOfBytesToRead,  \_Out\_ LPDWORD   lpdwNumberOfBytesRead  ); | |
| 매개변수 | hFile | 이전 호출에서 반환 된 핸들 |
| lpBuffer | 데이터를 수신 버퍼에 대한 포인터. |
| dwNumberOfBytesToRead | 바이트 수를 읽을 수 있습니다. |
| lpdwNumberOfBytesRead | 바이트 수를받는 변수의 포인터 |
| 반환 | 성공 : TRUE  실패 : FALSE (GetLastError 참조) | |
| 헤더 | Wininet.h | |

**■ InternetConnect()**

|  |  |  |
| --- | --- | --- |
| 설명 | 특정 사이트에 대한 파일 전송 프로토콜 (FTP) 또는 HTTP 세션을 연다. | |
| Syntax | HINTERNET InternetConnect(  \_In\_ HINTERNET     hInternet,  \_In\_ LPCTSTR       lpszServerName,  \_In\_ INTERNET\_PORT nServerPort,  \_In\_ LPCTSTR       lpszUsername,  \_In\_ LPCTSTR       lpszPassword,  \_In\_ DWORD         dwService,  \_In\_ DWORD         dwFlags,  \_In\_ DWORD\_PTR     dwContext  ); | |
| 매개변수 | hInternet | InternetOpen 함수에 대한 이전 호출에 의해 반환 된 핸들 |
| lpszServerName | 널 인터넷 서버의 호스트 이름을 지정합니다로 끝나는 문자열 포인터이다. |
| nServerPort | 서버의 전송 제어 프로토콜 / 인터넷 프로토콜 (TCP / IP) 포트. 이러한 플래그를 사용할 경우에만 포트를 설정합니다. 서비스는 값으로 설정된다. |
| lpszUsername | 로그온 할 수 있는 사용자의 이름을 지정하는 null로 끝나는 문자열 포인터이다. |
| lpszPassword | 로그인 할 때 사용하는 암호를 포함하는 null로 끝나는 문자열 포인터이다. |
| dwService | 서비스의 종류에 액세스 할 수 있다.  INTERNET\_SERVICE\_FTP  INTERNET\_SERVICE\_GOPHER  INTERNET\_SERVICE\_HTTP |
| dwFlags | 사용하는 서비스에 특정 옵션을 선택 |
| dwContext | 콜백에서 리턴 핸들 애플리케이션 콘텍스트를 식별하는 데 사용되는 응용 프로그램에서 정의 된 값을 포함하는 변수의 포인터 |
| 반환 | 성공 : 세션에 대한 유효한 핸들 리턴.  실패 : NULL | |
| 헤더 | Wininet.h | |

**■ connect()**

|  |  |  |
| --- | --- | --- |
| 설명 | 특정의 소켓에 대한 연결을 확립하는 함수 | |
| Syntax | int connect(  \_In\_ SOCKET                s,  \_In\_ const struct sockaddr \*name,  \_In\_ int                   namelen  ); | |
| 매개변수 | s | 연결되지 않은 소켓을 식별하는 기술자 |
| name | 연결이 반드시 되어있는 SOCKADDR 구조체를 가리키는 포인터 |
| namelen | 이름 인자가 가르키는 값의 길이 |
| 반환 | 찬성 : 0을 반환  실패 : SOCKET\_ERROR를 반환 | |
| 헤더 | Winsock2.h | |

**■ FindFirstFile()**

|  |  |  |
| --- | --- | --- |
| 설명 | 특정 디렉토리 아래에 있는 모든 파일과 서브 디렉토리 혹은 특정 조건의 파일과 서브 디렉토리를 얻는다 | |
| Syntax | HANDLE WINAPI FindFirstFile(  \_In\_ LPCTSTR lpFileName,  \_Out\_ LPWIN32\_FIND\_DATA lpFindFileData  ) | |
| 매개변수 | LPCTSTR lpFileName | 특정한 경로나 디렉토리 또는 (\*) 또는 (?)를  포함한 wildcard character 파일명을 지정할 수 있다 |
| LPWIN32\_FIND\_DATA  lpFindFileData | WIN32\_FIND\_DATA 구조체에 찾은 파일 또는 서브 디렉토리의  정보를 담는다 |
| 반환 | 호출 성공 : search handle 반환  호출 실패 : INVALID\_HANDLE\_VALUE 반환 | |
| 헤더 | FileAPI.h (include Windows.h) | |

**■ ControlService()**

|  |  |  |
| --- | --- | --- |
| 설명 | 서비스에 제어 코드를 보내는 함수.  서비스를 중지 할 때 추가 정보를 지정하려면 ControlServiceEx의 기능을 사용한다. | |
| Syntax | BOOL WINAPI ControlService(  \_In\_  SC\_HANDLE        hService,  \_In\_  DWORD            dwControl,  \_Out\_ LPSERVICE\_STATUS lpServiceStatus  ); | |
| 매개변수 | hService | 서비스에 대한 핸들.  이 핸들에 의해 반환되는 함수는  OpenService 또는 CreateService이다. |
| dwControl | 이 인자의 자리에는 다음 인자들 중 하나가 차지한다.  SERVICE\_CONTROL\_CONTINUE  SERVICE\_CONTROL\_INTERROGATE  SERVICE\_CONTROL\_NETBINDADD  SERVICE\_CONTROL\_NETBINDDISABLE  SERVICE\_CONTROL\_NETBINDENABLE  SERVICE\_CONTROL\_NETBINDREMOVE  SERVICE\_CONTROL\_PARAMCHANGE  SERVICE\_CONTROL\_PAUSE  SERVICE\_CONTROL\_STOP |
| lpServiceStatus | 가장 최근 서비스 상태의 정보를 받는 SERVICE\_STATUS 구조체를 가리키는 포인터이다. |
| 반환 | 성공 : 0이 아닌 값을 반환  실패 : 0을 반환 | |
| 헤더 | Winsvc.h | |

**■ GetForegroundWindow()**

|  |  |
| --- | --- |
| 설명 | 앞 창 (사용자가 현재 작동하고 있는 창)에 대한 핸들을 가져오는 함수다. 시스템은 다른 스레드보다 앞 창 윈도우를 생성 스레드에 약간 더 높은 우선 순위를 할당한다. |
| Syntax | HWND WINAPI GetForegroundWindow(void); |
| 반환 | 반환 값은 앞 창 윈도우에 대한 핸들이다. 앞 창이 될 수 NULL 같은 창이 활성화를 잃는 경우와 같은 특정 상황이다. |
| 헤더 | Winuser.h |

**■ InternetOpenUrl()**

|  |  |  |
| --- | --- | --- |
| 설명 | 전체 FTP 또는 HTTP URL에 의해 지정된 리소스를 연다 | |
| Syntax | HINTERNET InternetOpenUrl(  \_In\_ HINTERNET hInternet,  \_In\_ LPCTSTR   lpszUrl,  \_In\_ LPCTSTR   lpszHeaders,  \_In\_ DWORD     dwHeadersLength,  \_In\_ DWORD     dwFlags,  \_In\_ DWORD\_PTR dwContext  ); | |
| 매개변수 | hInternet | 현재 인터넷 세션 핸들이다. 핸들은 반드시 이전 InternetOpen 호출에 의해 반환돼야 한다. |
| lpszUrl | 읽기를 시작하기 위한 URL을 구체화하는 null 종료 문자 변수를 가리키는 포인터이다. |
| lpszHeaders | HTTP 서버로 보내지는 헤더를 구분하는 null 종료 문자열을 가리키는 포인터이다. |
| dwHeadersLength | 추가되는 헤더의 크기 |
| dwFlags | 이 인자는 다음 중 하나이다.  INTERNET\_FLAG\_EXISTING\_CONNECT  INTERNET\_FLAG\_HYPERLINK  INTERNET\_FLAG\_IGNORE\_CERT\_CN\_INVALID  INTERNET\_FLAG\_IGNORE\_CERT\_DATE\_INVALID  INTERNET\_FLAG\_IGNORE\_REDIRECT\_TO\_HTTP  INTERNET\_FLAG\_IGNORE\_REDIRECT\_TO\_HTTPS  INTERNET\_FLAG\_KEEP\_CONNECTION  INTERNET\_FLAG\_NEED\_FILE  INTERNET\_FLAG\_NO\_AUTH  INTERNET\_FLAG\_NO\_AUTO\_REDIRECT  INTERNET\_FLAG\_NO\_CACHE\_WRITE  INTERNET\_FLAG\_NO\_COOKIES  INTERNET\_FLAG\_NO\_UI  INTERNET\_FLAG\_PASSIVE  INTERNET\_FLAG\_PRAGMA\_NOCACHE  INTERNET\_FLAG\_RAW\_DATA  INTERNET\_FLAG\_RELOAD  INTERNET\_FLAG\_RESYNCHRONIZE  INTERNET\_FLAG\_SECURE |
| dwContext | 어떤 콜백 함수에 반환 된 핸들과 함께 전달되는 응용 프로그램 정의 값을 지정하는 변수의 포인터. |
| 반환 | 성공 : URL에 유효한 핸들 반환  실패 : NULL 반환 | |
| 헤더 | Wininet.h | |

**■ Module32First()**

|  |  |  |
| --- | --- | --- |
| 설명 | 스냅샷의 첫 번째 엔트리에 있는 정보를 가져 온다 | |
| Syntax | BOOL WINAPI Module32First(  \_In\_ HANDLE hSnapshot,  \_Inout\_ LPMODULEENTRY32 lpme  ) | |
| 매개변수 | HANDLE hSnapshot | CreateToolHelp32Snapshot로 얻은 HANDLE 값 |
| LPMODULEENTRY32 lpme | MOUDLEENTRY32Entry 구조체의 대한 포인터 |
| 반환 | 모듈 목록의 첫 번째 항목이 존재하면 TRUE, 그렇지 않으면 FALSE | |
| 헤더 | TlHelp32.h | |

**■ GetModuleFilename()**

|  |  |  |
| --- | --- | --- |
| 설명 | 지정된 모듈을 포함하는 파일에 대한 완전한 경로를 검색하는 함수이다.  모듈은 현재 프로세스에 의해 로드 되어 있어야 한다.  다른 프로세스에 의해 로드 된 모듈 파일을 찾으려면 GetModuleFileNameEx의 기능을 사용해야 한다. | |
| Syntax | DWORD WINAPI GetModuleFileName(  \_In\_opt\_ HMODULE hModule,  \_Out\_    LPTSTR  lpFilename,  \_In\_     DWORD   nSize  ); | |
| 매개변수 | hModule | 요청된 경로의 로드되는 모듈을 가리키는 포인터이다. |
| lpFilename | 모듈의 완전한 경로를받는 버퍼에 대한 포인터. |
| nSize | lpFilename버퍼의 크기 |
| 반환 | 성공 : 문자, 널 종료 문자를 포함하지 않은 버퍼에 복사되는 문자열의 길이  실패 : 0 반환 | |
| 헤더 | WINBASE.H | |

**■ MapViewOfFile()**

|  |  |  |  |
| --- | --- | --- | --- |
| 설명 | 파일의 데이터에 접근하기 위한 영역을 프로세스 주소 공간 내에 확보한다 | | |
| Syntax | LPVOID WINAPI MapViewOfFile(  \_In\_ HANDLE hFileMappingObject,  \_In\_ DWORD dwDesiredAccess,  \_In\_ DWORD dwFileOffsetHigh,  \_In\_ DWORD dwFileOffsetLow,  \_In\_ SIZE\_T dwNumberOfBytesToMap  ) | | |
| 매개변수 | HANDLE  hFileMappingObject | CreateFileMapping으로 얻은 핸들을 넘겨준다 | |
| DWORD dwDesiredAccess | FILE\_MAP\_READ | CreateFileMapping에서 PAGE\_READONLY로  설정한 경우 |
| FILE\_MAP\_WRITE | CreateFileMapping에서 PAGE\_READ로  설정한 경우 |
| FILE\_MAP\_ALL\_ACCESS | FILE\_MAP\_READ | FILE\_MAP\_WIRTE |  FILE\_MAP\_COPY와 같다 |
| FILE\_MAP\_COPY | CreateFileMapping에서 PAGE\_WRITECOPY로  설정, 데이터를 쓰게 되면 새로운 페이지가 생성 된다 |
| FILE\_MAP\_EXECUTE | 데이터를 코드로 수행할 수 있다 |
| DWORD  dwFileOffsetHigh | File Offset의 최대 크기 | |
| DWORD dwFileOffsetLow | File offset의 최소 크기 | |
| SIZE\_T  dwNumberOfBytesToMap | 매핑 파일의 바이트 수를 뷰에 매핑한다  0ㅇ면, 매핑은 지정된 파일 매핑의 오프셋 단부터 연장된다 | |
| 반환 | 호출 성공 : 매핑 된 뷰의 시작 Address  호출 실패 : NULL | | |
| 헤더 | WinBase.h (include Windows.h) | | |

**■ CreateService()**

|  |  |  |
| --- | --- | --- |
| 설명 | 서비스 객체를 작성해, 지정된 서비스 제어 관리자 데이터베이스에 추가합니다. | |
| Syntax | SC\_HANDLE WINAPI CreateService(  \_In\_      SC\_HANDLE hSCManager,  \_In\_      LPCTSTR   lpServiceName,  \_In\_opt\_  LPCTSTR   lpDisplayName,  \_In\_      DWORD     dwDesiredAccess,  \_In\_      DWORD     dwServiceType,  \_In\_      DWORD     dwStartType,  \_In\_      DWORD     dwErrorControl,  \_In\_opt\_  LPCTSTR   lpBinaryPathName,  \_In\_opt\_  LPCTSTR   lpLoadOrderGroup,  \_Out\_opt\_ LPDWORD   lpdwTagId,  \_In\_opt\_  LPCTSTR   lpDependencies,  \_In\_opt\_  LPCTSTR   lpServiceStartName,  \_In\_opt\_  LPCTSTR   lpPassword  ); | |
| 매개변수 | hSCManage | 서비스 제어 관리자 데이터베이스에 대한 핸들 |
| lpServiceName | 서비스의 이름을 설치한다. 최대 문자열 길이는 256 자이다. |
| lpDisplayName | 서비스를 식별하는 사용자 인터페이스 프로그램에 의해 사용된다 |
| dwDesiredAccess | 서비스에 액세스 할 수 있다. 요청 된 액세스를 허용하기 전에, 시스템은 호출 프로세스의 액세스 토큰을 확인한다 |
| dwServiceType | 서비스 유형이다. |
| dwStartType | 서비스의 옵션이다. |
| dwErrorControl | 서비스를 시작하는 데 실패하면 오류의 심각도 및 조치를 취하는 매개변수. |
| lpBinaryPathName | 서비스 이진 파일의 정규화된 경로이다. 경로에 공백이 포함 된 경우가 올바르게 해석되도록 이를 인용해야한다. |
| lpLoadOrderGroup | 이 서비스가 속한 로드 순서 그룹의 이름이다.. 서비스가 그룹에 속하지 않은 경우, NULL 또는 빈 문자열을 지정한다. |
| lpdwTagId | 지정된 그룹에 고유 한 태그 값을 받는 변수 포인터 lpLoadOrderGroup의 매개변수이다. 기존 태그를 변경하지 않을 경우 NULL을 지정한다. |
| lpDependencies | 서비스 또는 시스템이 서비스 전에 시작해야 하는 시스템인 오더링 그룹을 로드 하거나 구분 된 이름의 이중 null로 끝나는 배열에 대한 포인터이다. |
| lpServiceStartName | 서비스가 실행해야하는 계정의 이름 |
| lpPassword | lpServiceStartName의 매개 변수로부터의 계정이름의 암호 |
| 반환 | 성공 : 상기 서비스에 대한 핸들 반환  실패 : NULL 반환 | |
| 헤더 | Winsvc.h | |