

인공지능 기반 침입 탐지 시스템

팀 명 :	도구조
지도 교수 :	양환석 교수님
팀 장 :	김명섭
팀 원 :	박재희
	송우영
	천호범
	박채환

2022. 11
중부대학교 정보보호학과

목 차

1. 서론

1.1 연구 배경	4
1.2 연구 필요성	4

2. 관련 연구

2.1 Python	4
2.2 XGBoost	4
2.3 SelectKBest	5
2.4 Random Forest	5
2.5 DNN	5
2.6 LSTM	5
2.7 NIDS	6

3. 본론

3.1 시스템 구성	7
3.2 데이터 셋	7
3.2.1 NSL-KDD	7
3.2.1 CSE-CIC-IDS-2018	8
3.3 특징 추출	8
3.3.1 DoS	9
3.3.2 Probe	9
3.3.2 DDoS	10
3.3.2 Brute Force	12

3.4 침입 탐지 프로그램	13
3.4.1 프로그램 UI	13
3.4.2 데이터 변환과 패킷 탐지	14
4. 결론	
4.1 결론	15
4.2 기대 효과.....	15
5. 별첨	
5.1 소스 코드.....	15
5.2 발표 자료.....	16

1. 서론

1.1 연구 배경

현재 머신러닝을 적용한 자동화된 침입탐지기술이 각광받고 있다. 많은 양의 데이터에서 복잡한 패턴을 찾을 수 있는 딥러닝 기법의 효율성 때문이다. 수없이 많은 네트워크 패킷들을 분석하여 정상 행위의 패턴을 학습하는 기술을 비정상 행위를 탐지하는 것을 목표로 하는 침입탐지 시스템에 사용한 결과 좋은 결과를 보이고 있다. 이에 따라 우리는 기계학습으로 NSL-KDD와 CSE- CIC-IDS-2018의 데이터셋을 사용해 비정상 탐지에 활용할 수 있도록 정확도를 높이고 실제 네트워크 환경에서 동작하는 침입탐지 시스템을 구현하려고 한다.

1.2 연구 필요성

최근 정보통신 기술이 발달하며 급격하게 정보량이 늘어나고, 그에 따라 해킹을 시도하는 사례가 많이 늘고 있다. 이에 따라 여러 종류의 공격 행위를 탐지하는 침입탐지 시스템이 필요하게 되었다. 효과적으로 대응하기 위해 기존보다 빠르고 저렴한 네트워크 침입탐지 시스템의 개발과 기술 국산화가 필수적이다.

2. 관련 연구

2.1 Python

Python은 웹 애플리케이션, 소프트웨어 개발, 데이터 과학, 기계학습(ML)에 널리 사용되는 프로그램 언어이다. 무료로 다운로드 가능하며 Windows, macOS, Linux 및 Unix와 같은 다양한 컴퓨터 운영 체제에서 호환 가능하다. 다른 언어에 비해 더 적은 코드 줄을 사용하고, 거의 모든 작업에 재사용 가능한 코드가 포함된 대규모 표준 라이브러리가 있다. 다른 프로그래밍 언어보다 인간 언어에 가까운 고급언어이다. 인터넷에서 유용한 리소스가 많아 배우기 쉽기 때문에 개발 속도를 증가시킨다. 모든 것을 객체로 간주하지만, 구조적 및 함수형 프로그래밍 등의 다른 프로그래밍 유형도 지원하여 모든 유형의 시스템과 원활하게 통합된다. Python은 해석된 언어로 코드를 한 줄씩 직접 실행한다. 오류를 빠르게 찾을 수 있다. 문제가 발생하면 커뮤니티에서 빠른 지원을 받을 수 있다.

2.2 XGBoost

XGBoost는 Extreme Gradient Boosting의 약자이다. Boosting은 앙상블 기법 중 하나로 약한 예측 모형들의 학습 에러에 가중치를 두고, 순차적으로 다음 학습 모델에 반영하여 강한 예측모형을 만드는 것이다. 이러한 Boosting 기법을 이용하여 구현한 알고리즘은 Gradient Boost가 대표적인데 이 알고리즘을 병렬 학습이 지원되도록 구현한 라이브러리가 XGBoost이다. 강력한 앙상블 모델인 GBM의 단점인 과적합(Overfitting), 속도 문제를 보완하기 위해 생겼다. GBM과 마찬가지로 가중치 업데이트를 경사하강법을 사용한다. 성능과 자원 효율이 좋고, Regression, Classification 문제를 모두 지원하여 인기 있게 사용

되는 알고리즘이다.

2.3 SelectKBest

특징 선택 방법 중 Filter Method라는 전처리단에서 주로 사용할만 하며 통계기법등을 사용하여 상관관계가 높은 변수나, 성능이 높은 변수를 추출하는 방법이 있다. Filter Method 중에서도 단일 변수 선택 방식으로 카이스퀘어 검정 통계값(chi2), 분산분석 F검정 통계값(f_classif), 상호정보량(mutual_info_classif) 각각의 독립변수를 하나만 사용한 예측모형의 성능을 이용하여 가장 분류성능 혹은 상관관계가 높은 변수만 선택하는 방법이다. 이러한 방법들을 사용할 수 있도록 해주는 모듈 중 하나가 SelectKBest이다. SelectKBest는 Scikit-learn의 모듈로 target변수와 그외 변수 사이의 상관관계를 계산하여 가장 상관관계가 높은 변수 k개를 선정할 수 있도록 해주는 모듈이다.

2.4 Random Forest

Random Forest는 최적의 기준 변수를 랜덤하게 선택하는 머신러닝 기법이다. 앙상블 학습의 대표적인 모델로 Bagging을 사용하여 랜덤으로 일부의 feature만 선택하여 Decision tree를 만들고, 해당 과정을 반복하여 여러 개의 Decision tree를 형성합니다. 여러 개의 Decision tree에서 나온 예측값을 토대로 가장 많이 나온 값을 최종 예측값으로 선정하여 특정 Decision에서 overfitting이 발생하더라도 여러 개의 결과를 합쳐 최종 결과를 도출하기 때문에 overfitting을 방지할 수 있다.

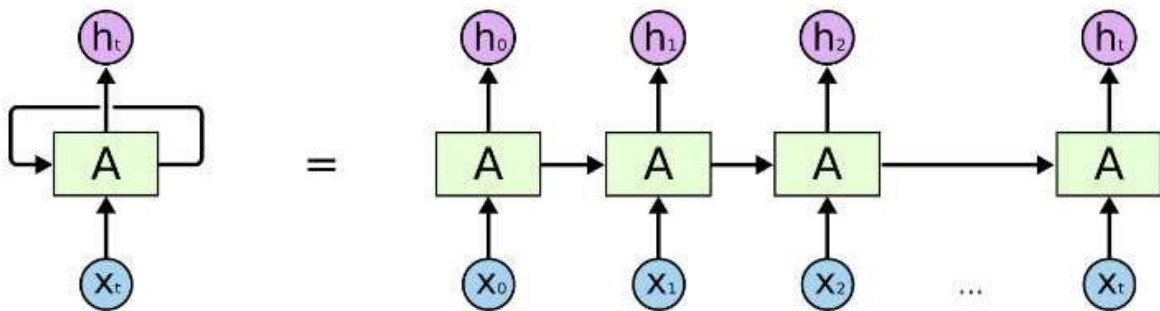
2.5 DNN

DNN은 모델 내 은닉층을 많이 늘려서 학습의 결과를 향상시키는 방법으로 은닉층을 2개 이상 지닌 학습방법을 뜻한다. 각 노드는 이전 층에서 주어지는 입력값 Weight를 곱하고 곱한 값들은 전부 더해지고 그 합은 활성화 함수의 입력으로 들어가게 된다. 활성화 함수의 결과가 노드의 출력에 해당하며, 이 출력값이 궁극적으로 분류나 회귀 분석에 쓰이게 된다. Bias를 더한 뒤 활성화 함수를 거쳐 출력값을 계산하며 이 과정을 입력층에서 출력층으로 순차적으로 진행하여 최종적으로 DNN의 출력값을 계산한다. 다층의 은닉층을 포함하여 다양한 비선형적 관계를 학습할 수 있지만, 학습을 위한 많은 연산량과 과하게 학습하여 실제 데이터에 대해 오차가 증가하는 과적합, 기울기 값의 소실 문제 등이 발생할 수 있다. 드롭아웃, ReLU(Rectified Linear Unit), 배치 정규화 등의 법이 적용되면서 딥러닝의 핵심 모델로 활용되고 있다.

2.6 LSTM

LSTM은 딥러닝 분야에서 사용되는 RNN 아키텍처로 기존의 RNN은 단기 메모리만을 가지고 recurrently 학습했다면, LSTM은 단기 메모리와 장기 메모리를 나눠 학습 후, 두 메모리를 병합해 이벤트 확률을 예측한다. LSTM은 본격적인 연산 전에 장기/단기 정보를 담은 메모리를 분류하고 이 메모리와 이벤트를 기반으로 각각 Long term memory, Short term memory에 적합한 내용을 따로따로 학습시킨다. 또한, 모든 RNN이 모듈을 반복시

키는 체인과 같은 형태를 하고 있는 것 처럼 LSTM도 똑같이 체인과 같은 구조를 가지고 있지만, 각 반복 모듈은 다른 구조를 갖고 있다. 단순한 neural network layer 한 층 대신에, 4개의 layer가 특별한 방식으로 서로 정보를 주고 받도록 되어 있다. neural network는 이전에 일어난 사건을 바탕으로 나중에 일어나는 사건을 생각하지 못한다. Recurrent neural network (RNN)는 이 문제를 해결하고자 하는 모델이다. RNN은 스스로를 반복하면서 이전 단계에서 얻은 정보가 지속되도록 한다. RNN은 input x 를 받아서 h 를 내보낸다. A를 둘러싼 반복은 다음 단계에서의 network가 이전 단계의 정보를 받는다는 것을 보여준다.



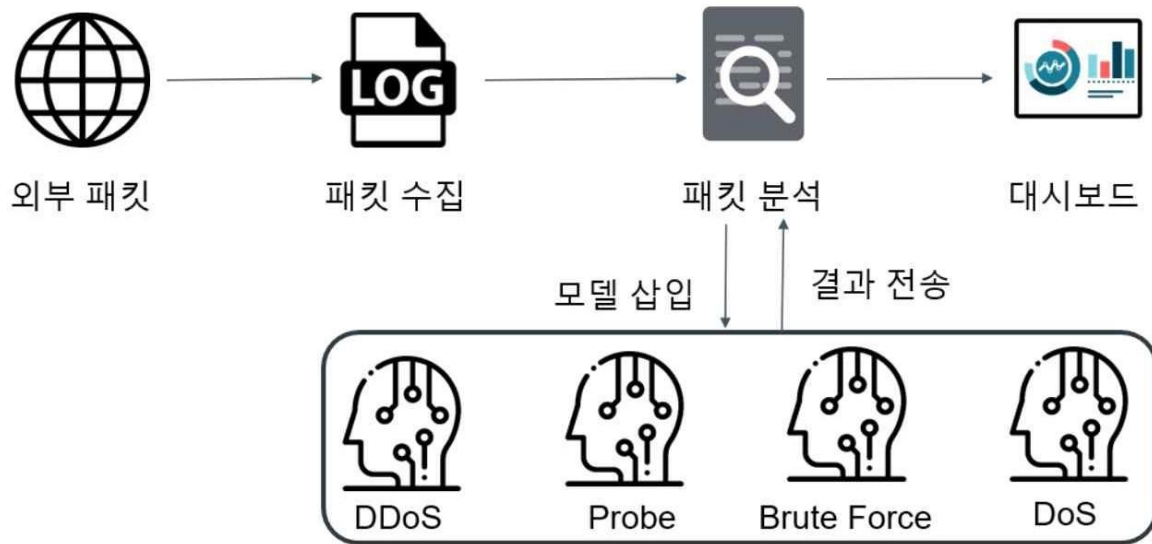
[그림 1 LSTM]

2.7 NIDS

NIDS(network intrusion detection system)는 네트워크 침입 탐지 시스템으로 IDS의 한 종류로 네트워크 트래픽을 감시하여 서비스 거부 공격, 포트 스캔, 컴퓨터를 크랙하려는 시도 등과 같은 악의적인 동작들을 탐지하는 시스템이다. NIDS는 네트워크 세그먼트당 하나의 감지기만 설치하면 되므로 설치에 용이하고 트래픽을 몇몇 위치에만 설치하므로 초기 구축 비용이 저렴하다. 운영체제에 독립적이므로 구현 및 관리가 쉽고 캡처된 트래픽에 대해 침입자가 제거하기가 어려우며 네트워크에서 개별 실행되어 개별 서버의 성능 저하가 없다는 장점이 있다. 하지만 암호화된 패킷을 분석할 수 없고 고속 네트워크 환경에서는 패킷 손실률이 많아 탐지율이 떨어지며 호스트 상에서 수행되는 세부 행위에 대해 탐지할 수 없다는 단점도 가지고 있다

3. 본론

3.1 시스템 구성



[그림 2 시스템 구성]

3.2 데이터 셋

3.2.1 NSL-KDD

NSL-KDD는 2009년 Tavallae가 KDD CUP 99 의 본질적인 문제점을 비판하며 제안한 데이터 셋으로 KDD CUP 99를 보완하여 41개의 특징과 공격유형을 나타내는 라벨, 총 42개의 라벨로 이루어져있다. DoS, Probe, R2L, U2R 4가지 공격과 normal로 구분된다

3.2.2 CSE-CIC-IDS-2018

Attack Class	Attack Type
DoS	Apach2, smurf, Neptune, Back, Teardtop, Pod, Land, Mailbomb, Processtable, UDPstorm
Probe	Satan, Saint, Ipsweep, Portsweep, Nmap, Mscan
R2L	WareClient, Guess_Password, Imap, WareMaster, Ftp_Write, Named, MultiHop, Phf, Spy, Sendmail, SnmpGetAttack, SnmpGuess, Worm, Xsnoop, Xlock
U2R	Buffer_Overflow, Httpunnel, Rootkit, LoadModule, Perl, Xterm, Ps, SQLattack

[표 1 NSL-KDD]

데이터는 총 148517개로 normal 77054개, DoS 53385개, Probe 14077개, R2L 3882개, U2R 119개로 이루어져 있으며 두 가지 유형의 훈련 및 시험 데이터 셋을 제공한다. KDDTrain+는 공격 유형 레이블을 포함한 전체 훈련 데이터 셋이고 KDDTrain-는 KDDTrain+의 20%를 포함하는 훈련 데이터 셋이다. KDDTest+은 공격유형 레이블을 포함한 전체 시험 데이터 셋이고 KDDTest-21는 KDDTest+에서 난이도 수준 21을 제거한 시험 데이터 셋이다. 난이도 수준 21은 공격 검출이 가장 쉬운 레코드를 의미한다. NSL-KDD는 네트워크 비정상 검출 알고리즘 혹은 IDS/IPS 성능을 평가할 때 전 세계적으로 가장 많이 사용되는 것으로 알려져 있다.

3.2.2 CSE-CIC-IDS-2018

CSE-CIC-IDS-2018은 캐나다 사이버보안 연구소에서 제공하는 데이터셋으로 80개의 특징을 가지고 있으며 Brute-force, Heartbleed, Botnet, DoS, DDoS, 웹 공격 및 내부 네트워크 침투의 7가지의 공격 시나리오가 포함되어있다. 공격 인프라에는 50대의 머신이 포함되어 있으며 피해 조직에는 5개의 부서가 있으며 420대의 머신과 30대의 서버가 있다. 데이터 세트는 CICFlowMeter-V3 를 사용하여 캡처하였고, 캡처된 트래픽에서 80개의 기능과 함께 각 시스템의 네트워크 트래픽 및 시스템 로그가 포함되어있다.

attack	Tools	Duration	Attacker	Victim
Bruteforce attack	FTP – Patator, SSH – Patator	One day	Kali linux	Ubuntu 16.4 (Web Server)
DoS attack	Hulk, GoldenEye, Slowloris, Slowhttptest	One day	Kali linux	Ubuntu 16.4 (Apache)
DoS attack	Heartleech	One day	Kali linux	Ubuntu 12.04 (Open SSL)
Web attack	Damn Vulnerable Web App (DVWA), In-house selenium framework (XSS and Brute-force)	Two days	Kali linux	Ubuntu 16.4 (Web Server)
Infiltration attack	First level: Dropbox download in a windows machine, Second Level: Nmap and portscan	Two days	Kali linux	Windows Vista and Macintosh
Botnet attack	Ares (developed by Python): remote shell, file upload/download, capturing screenshots and key logging	One day	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)
DDoS+PortScan	Low Orbit Ion Canon (LOIC) for UDP, TCP, or HTTP requests	Two days	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)

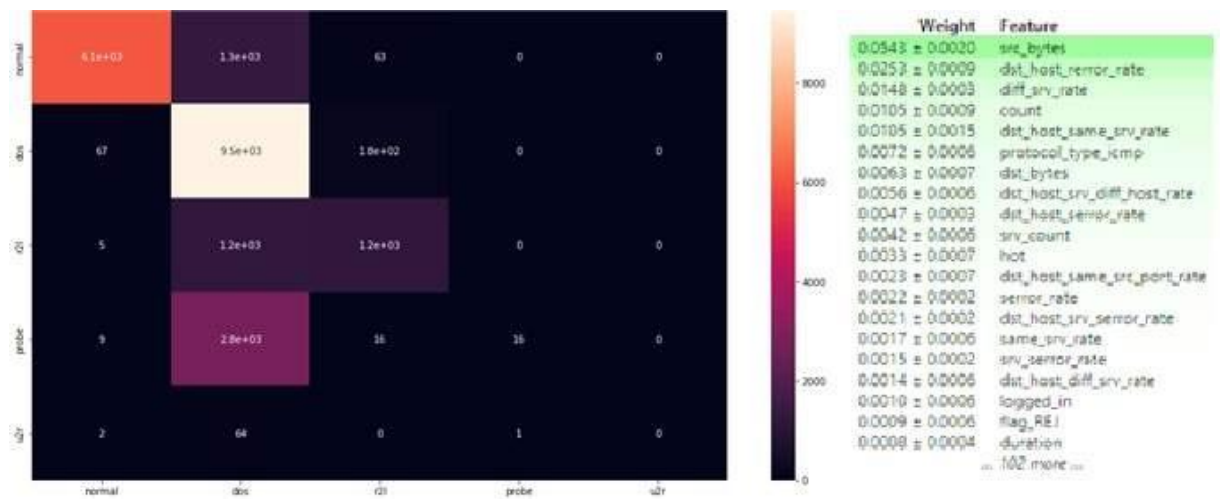
[그림 3 CSE-CIC-IDS-2018]

3.3 특징 추출

머신 러닝의 성능은 어떤 데이터를 입력하는지가 굉장히 의존적이며 가장 이상적인 입력 데이터는 부족하지도 과하지도 않은 정확한 정보만 포함될 때이다. 그렇기에 각각의 공격에 대해 어떤 feature가 유용한지 아닌지 확인하는 과정을 거쳐 추출하여 계산시간을 절약하고 잡음에 의한 과적합이 발생하지 않도록 했다.

3.3.1 DoS

Class 라벨을 제외한 41가지 특징 중 DoS 공격에 대한 연관 특징, K-means, RFE, Eli5 알고리즘을 사용하여 특징 중요도를 추출하였다. 상위 20가지의 특징들에 대해 체크를 한 후 2가지 이상 중복된 22가지의 특징값을 선정하였다.



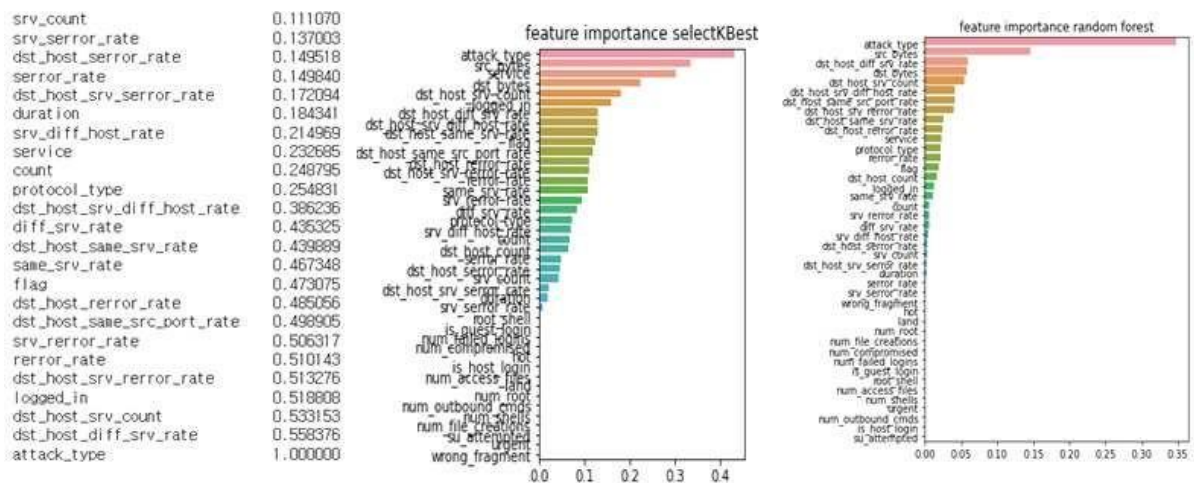
[그림 4 DoS K-means, Eli5]

index	특징	연관특성	k-means	rfe	eli5	최종
1	duration					duration
3	service					service
4	flag					flag
5	src_bytes					src_bytes
6	dst_bytes					dst_bytes
7	land					land
8	wrong_fragment					wrong_fragment
10	hot					hot
23	count					count
24	srv_count					srv_count
25	error_rate					error_rate
26	srv_error_rate					srv_error_rate
27	error_rate					error_rate
29	same_srv_rate					same_srv_rate
30	diff_srv_rate					diff_srv_rate
32	dst_host_count					dst_host_count
34	dst_host_same_srv_rate					dst_host_same_srv_rate
35	dst_host_diff_srv_rate					dst_host_diff_srv_rate
36	dst_host_same_src_port_rate					dst_host_same_src_port_rate

[그림 5 DoS 특징 추출]

3.3.1 Probe

NSL-KDD는 41개의 특징을 가지며 4가지의 공격 유형으로 이루어져 있다. 그중 Probe 공격에 관련있는 특징들을 추출하기 위해 $\text{abs}(\text{corr}[\text{'attack_type'}])$ 을 이용해 상관관계수가 0.1 이상인 특징들을 확인하였다. 또한 유용한 특징들은 선택하기 위해 분산분석 F검정을 이용한 selectKBest와 RandomForest를 사용하여 19개의 특징을 선정하였다.

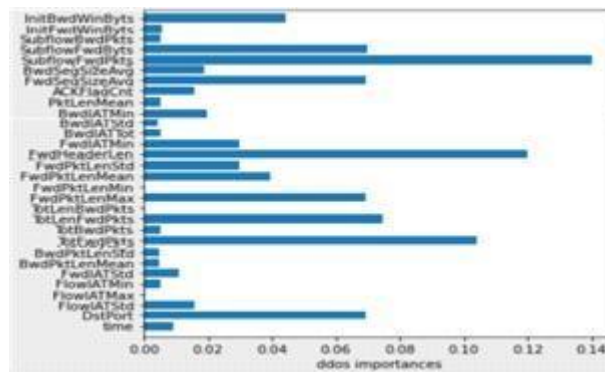


DDOS를 탐지하기 위한 모델을 만들기 위해서는 데이터 전처리 작업이 필요하다. 모든 데이터 셋의 변수를 모두 측정하고 다룰 수는 없다. 정확도를 올리기 위해서 두 변수간의 관계를 -1~1 값으로 표현했을 때 -1에 가까울 수록 음의 상관관계, 1에 가까울 수록 양의 상관관계를 가지는 `df.corr()[&공격유형&].sort_values(ascending=False)`을 이용하여 특징을 확인하였다.

```
df.corr()["DDOS"].sort_values(ascending=False)
```

[그림 7 DDoS 상관관계]

데이터셋에서 유용한 특성을 선택하는 또 다른 방법으로 앙상블 기법인 랜덤 포레스트를 사용하였다. 랜덤 포레스트를 사용하면 앙상블에 참여한 모든 결정 트리에서 계산한 평균적인 불순도 감소로 특성 중요도를 측정할 수 있다. 데이터셋이 선형적으로 구분 가능한지 여부를 가정할 필요가 없다. 사이킷런의 RandomForestClassifier 모델을 훈련한 후 feature_importances_ 속성에서 확인할 수 있다.



[그림 8 DDoS Random Forest 특징 중요도]

eli5라이브러리를 이용하여 특성 값을 순열로 만든 후 모델의 예측 오차 증가량을 계산하여 특성값의 중요도를 측정하는 순열 특성 중요도를 사용하였다. 중요도가 마이너스인 특성을 제외해도 모델 학습 속도는 좋아지지만 성능의 거의 영향이 없다.

Weight	Feature
0.0023 ± 0.0001	Dst Port
0.0016 ± 0.0001	Subflow Fwd Byts
0.0013 ± 0.0001	Fwd Header Len
0.0013 ± 0.0001	Tot Fwd Pkts
0.0012 ± 0.0001	Pkt Len Mean
0.0012 ± 0.0001	Subflow Bwd Pkts
0.0012 ± 0.0001	Tot Bwd Pkts
0.0011 ± 0.0002	Fwd Pkt Len Max
0.0009 ± 0.0001	Subflow Fwd Pkts
0.0009 ± 0.0001	Fwd Pkt Len Mean
0.0008 ± 0.0001	Fwd Pkt Len Std
0.0008 ± 0.0001	Fwd Seg Size Avg
0.0008 ± 0.0001	Init Bwd Win Byts
0.0008 ± 0.0001	TotLen Fwd Pkts
0.0008 ± 0.0001	Bwd IAT Min
0.0008 ± 0.0001	Pkt Len Var
0.0008 ± 0.0001	Pkt Size Avg
0.0007 ± 0.0001	Pkt Len Max
0.0007 ± 0.0001	PSH Flag Cnt
0.0007 ± 0.0001	Bwd Header Len
0.0007 ± 0.0001	Bwd Pkts/s
0.0005 ± 0.0001	Flow IAT Mean
0.0005 ± 0.0001	Fwd IAT Std
0.0001 ± 0.0000	Flow IAT Std

[그림 9 DDoS Eli5 특징 중요도]

위 세가지를 참조하여, 25가지의 특징값을 선정하였다.

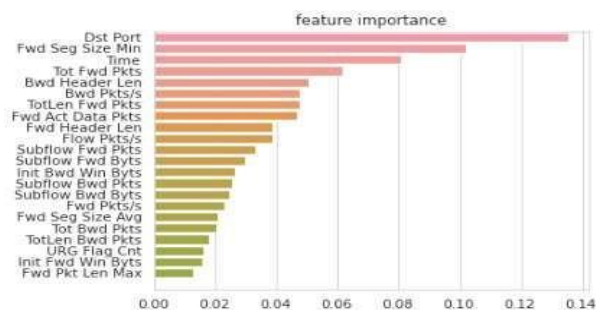
3.3.1 Brute Force

Brute Force 공격은 특징 추출을 진행하기 전 Random Forest, gradient Boosting, XGBoosting,으로 특성 간 상관관계를 분석하였다. CSE-CIC-IDS-2018 데이터 셋에 대해 적합한 출력값을 가지는 Random Forest 특징 추출을 진행하였다.

Subflow Bwd Byts	2.109619e-01	Fwd Seg Size Min	0.246614	Fwd Seg Size Min	0.247514
Dst Port	4.979543e-02	Dst Port	0.246244	Dst Port	0.246606
Time	3.755921e-02	Init Fwd Win Byts	0.091865	Flow Duration	0.095358
Init Bwd Win Byts	2.544292e-02	TotLen Bwd Pkts	0.000000	Fwd Pkt Len Max	0.000000
Bwd IAT Max	6.094368e-04	Flow IAT Std	0.000000	Bwd Pkts/s	0.000000
Fwd Seg Size Min	4.919330e-04	Fwd Pkt Len Min	0.000000	Flow IAT Mean	0.000000
ACK Flag Cnt	3.044140e-05	Protocol	0.000000	Protocol	0.000000
Flow Pkts/s	1.278539e-05	Flow Duration	0.000000	Tot Fwd Pkts	0.000000
Pkt Len Var	1.035008e-05	Tot Fwd Pkts	0.000000	Tot Bwd Pkts	0.000000
Fwd Pkt Len Std	6.088280e-06	Tot Bwd Pkts	0.000000	TotLen Fwd Pkts	0.000000
Bwd Pkt Len Std	6.088280e-06	TotLen Fwd Pkts	0.000000	TotLen Bwd Pkts	0.000000
Bwd Pkts/s	4.870624e-06	Fwd Pkt Len Mean	0.000000	Fwd Pkt Len Mean	0.000000
Fwd IAT Max	2.435312e-06	Fwd Pkt Len Max	0.000000	Fwd Pkt Len Min	0.000000
Flow IAT Mean	2.435312e-06	Flow IAT Min	0.000000	Flow IAT Min	0.000000
Fwd Pkts/s	1.217656e-06	Bwd Pkts/s	0.000000	Fwd Pkt Len Std	0.000000
Pkt Len Mean	6.088280e-07	Fwd Pkt Len Std	0.000000	Bwd Pkt Len Max	0.000000
Fwd Header Len	6.088280e-07	Bwd Pkt Len Max	0.000000	Bwd Pkt Len Min	0.000000
Bwd IAT Std	0.000000e+00	Bwd Pkt Len Min	0.000000	Bwd Pkt Len Mean	0.000000

[그림 10 Random Forest, Gradient Boosting, XGBoosting 상관관계]

Random Forest의 지니 불순도를 이용해서 각 변수의 중요도를 계산했고 feature_importances_를 이용하여 계산된 중요도를 시각화하여 21가지의 특징을 선정하였다.

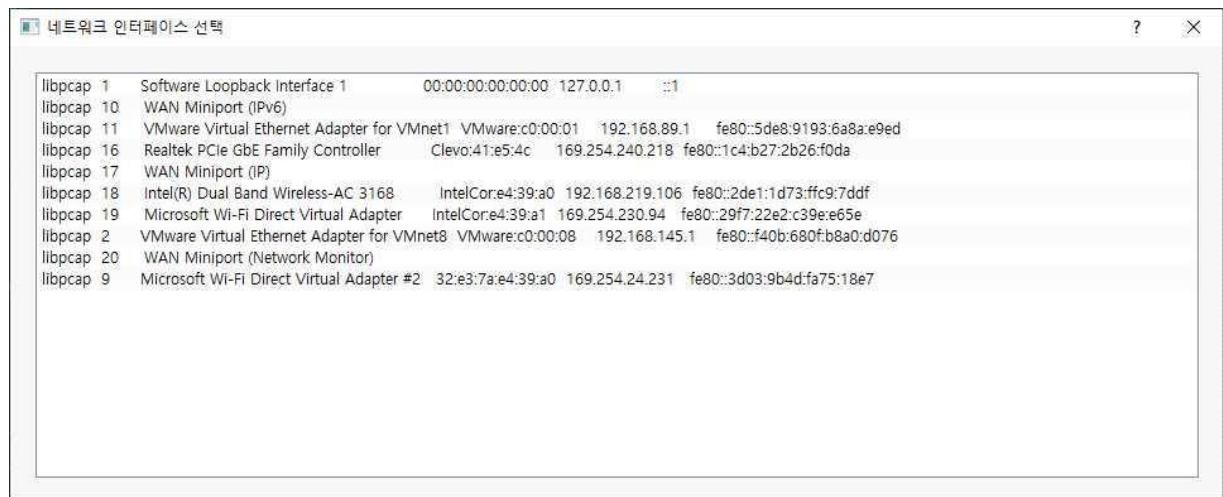


[그림 11 Random Forest 특징 시각화]

3.4 패킷 탐지 프로그램

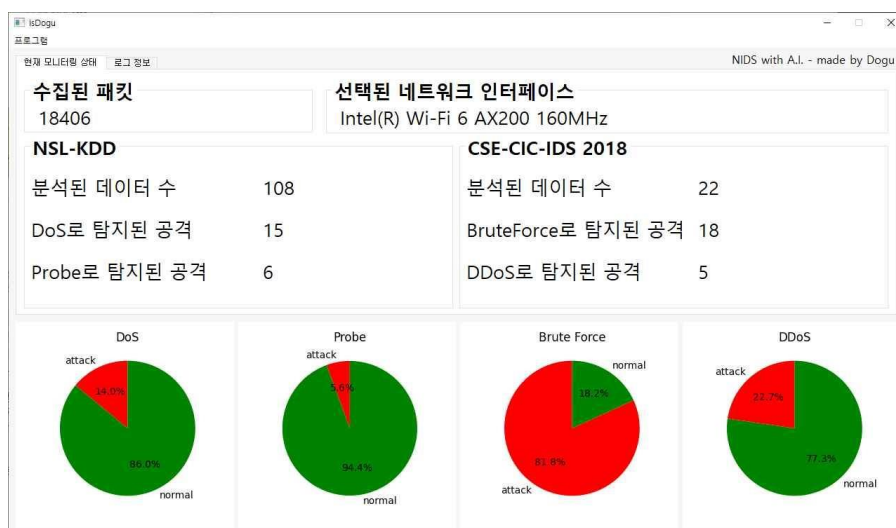
3.4.1 프로그램 UI 구성

해당 프로그램의 UI는 네트워크 인터페이스 선택, 데이터 수치 확인, 로그 정보 확인, 그래프 파트로 구성되었다. 네트워크 인터페이스 선택 UI에서는 현재 디바이스에 활성화되어 있는 네트워크 인터페이스들을 나열하고 사용자가 어느 네트워크에서 패킷을 받을지 선택할 수 있다.



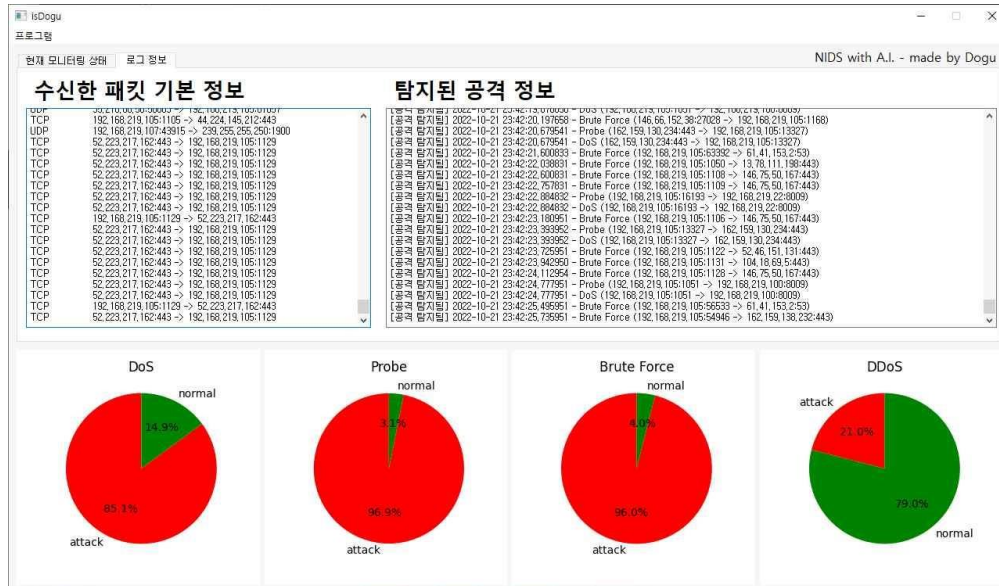
[그림 12 네트워크 선택 UI]

데이터 수치 확인 파트에서는 선택한 네트워크로 수신된 패킷의 수, 공격 탐지 모델에 쓰일 NSL-KDD와 CSE-CIC-IDS 2018 형식의 특징 값으로 변환된 데이터의 수와 각각의 공격으로 탐지된 횟수를 확인할 수 있다.



[그림 13 모니터링 UI]

로그 정보 확인 탭에서는 수신된 패킷의 정보, 공격이 탐지되었을 때 해당 패킷의 기본적인 정보가 기록되고 사용자가 확인할 수 있다. 그래프를 사용하여 각 공격 탐지 모델에서 나온 공격, 정상 데이터 비율을 시각적으로 확인할 수 있다.



[그림 14 로그 정보 탭 UI]

3.4.2 데이터 변환과 패킷 탐지

패킷을 NSL-KDD 데이터 형식으로 변환하는 코드는 C++로만 작성되어 있기 때문에, 파이썬으로 작성된 해당 프로그램에 포함시키기 위해서 DLL의 형태로 제작되었다. 반면에 CSE-CIC-IDS 2018의 데이터 형식으로 변환하는 코드는 JAVA, Python으로만 작성되어 있으며, 여기에선 Python으로 작성된 코드를 사용한다.

파이썬에서는 Scapy를 이용해서 패킷을 받고 C++에서는 Npcap을 사용하여 패킷을 받는데, 하나의 프로그램에서 두 개의 패킷 캡처 라이브러리를 사용하는 이유는 NSL-KDD와 CSE-CIC-IDS-2018로 변환하는 각각의 코드들이 요구하는 패킷의 형태가 달라 호환이 되지 않기 때문이다. 그래서 어느 하나를 특정해 사용하려면 복잡한 변환 과정을 추가로 거쳐야 하는데, 이렇게 되면 초기 구상 단계에서 생각한 것보다 구현 난이도가 너무 높아지게 되어 여기서는 프로그램 성능을 조금 손해 보더라도 Npcap과 Scapy를 같이 사용하기로 하였다.

그렇게 NSL-KDD의 형식으로 변환된 데이터들은 DoS, Probe 탐지 모델로 입력되고, CSE-CIC-IDS 2018의 형식으로 변환된 데이터들은 Brute Force, DDoS 탐지 모델로 입력되어 해당 데이터가 공격인지 정상인지 판단하게 된다.

4. 결론

4.1 결론

본 프로젝트는 네트워크 침입에 대해 인공지능을 사용하여 공격에 대해 빠르게 탐지할 수 있는지에 대해 살펴보았다. 총 4가지 공격 방법을 선정하였고, 선정방법으로는 데이터셋에서 데이터의 양이 가장 많으며 잘 알려진 공격위주로 선정하였다. 4가지 공격별로 필요한 특징을 추출한 후 학습을 진행하였고 모델별, 학습횟수에 따라서 성능이 차이나는 것을 확인할 수 있었다. 실제 패킷 분석기와 모델을 연결하여 공격을 시도했을때 선정한 공격들의 기준이 비슷하여 중복탐지가 되는 경우도 발생하지만 각 공격에 대해 높은 탐지율을 보여주고 있다.

4.2 기대효과

네트워크를 이용한 공격은 기술의 발전과 함께 다양해지고 정교한 형태로 발전하고 있다. AI기반의 침입 탐지 시스템은 들어오는 패킷을 가지고 공격 여부를 판단한다. 따라서 위 기술은 아직 알려지지 않은 네트워크 공격들을 잡아낼 수 있다고 생각한다. 단 공격을 쉽게 탐지할 수 있는 만큼 공격이 아닌 패킷을 잡아내는 위험성이 존재하므로 잡아낸 패킷들의 정보를 사람이 한 번 더 확인을 해봐야 한다고 생각한다. 앞으로 많은 시도와 기술의 발전으로 인하여 이러한 부분은 차츰 더 나은 결과를 도출해 낼 수 있을 것이고 기술이 발전함에 따라 보안에서 더 안전한 환경을 만들어주며 특히 관제에서 전에는 인공지능을 사용함으로써 사람 대신 들어오는 모든 패킷을 검사해주고 이로 인해 시간 절약, 정확성 등 더 안전하고 쾌적한 환경을 만들어 줄 수 있을 거라 생각된다.

5. 별첨

5.1 소스코드

https://github.com/myeongseop2/NIDS_project.git

5.2 발표 PPT

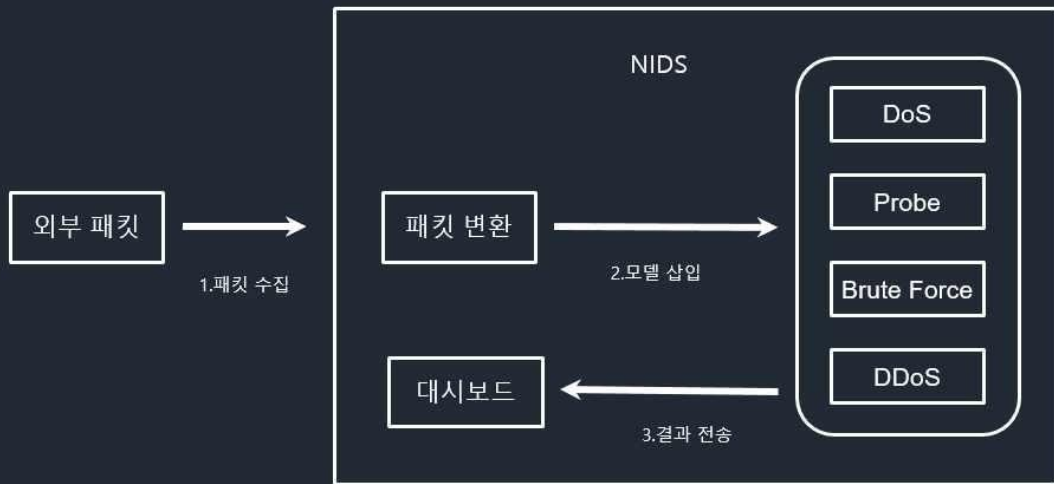
인공지능 기반 침입탐지시스템

김명섭
박재희
송우영
천호범
박채환

목차

1. 시스템 구상도
2. 개발환경 및 개발내용
3. 개발 시스템 운영
4. 결론 및 기대효과

구상도



개발 환경 및 개발 내용



OS

개발언어

개발환경

개발 환경 및 개발 내용

NSL-KDD	Number of Records:					
	Total	Normal	DoS	Probe	U2R	R2L
KDDTrain+20%	25192	13449 (53%)	9234 (37%)	2289 (9.16%)	11 (0.04%)	209 (0.8%)
KDDTrain+	125973	67343 (53%)	45927 (37%)	11656 (9.16%)	52 (0.04%)	995 (0.8%)
KDDTest+	22544	9711 (43%)	6082 (27%)	3551 (16%)	19 (0.08%)	1161 (5.1%)

Tools	Duration	Attacker	Victim
Bruteforce attack FTP - Patator SSH - Patator	One day	Kali linux	Ubuntu 16.4 (Web Server)
DoS attack Hulk, GoldenEye, Slowloris, Slowhttptest	One day	Kali linux	Ubuntu 16.4 (Apache)
DoS attack Heartbleech	One day	Kali linux	Ubuntu 12.04 (Open SSL)
Web attack Damn Vulnerable Web App (DVWA), In-house selenium framework (XSS and Brute-force)	Two days	Kali linux	Ubuntu 16.4 (Web Server)
Infiltration attack First level Dropbox download in a windows machine, Second Level: Nmap and portscan	Two days	Kali linux	Windows Vista and Macintosh
Botnet attack Aves (developed by Python) remote shell, file upload/download, capturing screenshots and key logging	One day	Kali linux	Windows Vista, 7, 8, 1, 10 (32-bit) and 10 (64-bit)
DDoS+PortScan Low Orbit Ion Canon (LOIC) for UDP, TCP, or HTTP requests	Two days	Kali linux	Windows Vista

CEC-CIC-IDS

개발 환경 및 개발 내용

dst Port	Protocol	Flow Duration	tot Fwd Pkts	tot Bwd Pkts	totLow Fwd Pkts	totLow Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Avg	...	Active Std	Active Max	Active Min	Idle Max	Idle Std	Idle Min	Date Time Index
0	0	0	112947719	0	0	0	0	0	0.00000	...	0.0	0	0	0	0	0	14022018_01100
1	0	0	113641496	0	0	0	0	0	0.00000	...	0.0	0	0	0	0	0	14022018_01100
2	0	0	113638023	0	0	0	0	0	0.00000	...	0.0	0	0	0	0	0	14022018_01100
3	22	0	6453066	15	10	1238	2275	744	0.00000	...	0.0	0	0	0	0	0	14022018_01100
4	22	0	3804008	14	11	1340	2238	744	0.00000	...	0.0	0	0	0	0	0	14022018_01100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
383608	22	0	16	1	1	0	0	0	0.00000	...	0.0	0	0	0	0	0	14022018_01100
383608	22	0	0	1	1	0	0	0	0.00000	...	0.0	0	0	0	0	0	14022018_01100
383608	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
duration	protocol_type	service	flag	src_bytes	dst_bytes	...	...	...	...	...	...	...	...	...	...	...	...
0	0	tcp	data	SF	491	0	0	0	0	0	...	...	...	...	...	...	...
1	0	udp	other	SF	144	0	0	0	0	0	...	...	...	...	...	...	...
2	0	tcp	http	SF	232	6153	0	0	0	0	...	...	...	...	...	...	...
3	0	tcp	http	SF	199	420	0	0	0	0	...	...	...	...	...	...	...
4	0	tcp	http	SF	287	2251	0	0	0	0	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12126	0	tcp	pop3	RSTO	0	36	0	0	0	0	...	...	...	...	...	...	...
12127	0	tcp	pop3	RSTO	0	36	0	0	0	0	...	...	...	...	...	...	...
12128	5	tcp	smtp	RSTO	0	44	0	0	0	0	...	...	...	...	...	...	...
12129	2	tcp	telnet	SF	24	109	0	0	0	0	...	...	...	...	...	...	...
12130	0	tcp	sunrpc	REJ	0	0	0	0	0	0	...	...	...	...	...	...	...

CEC-CIC-IDS

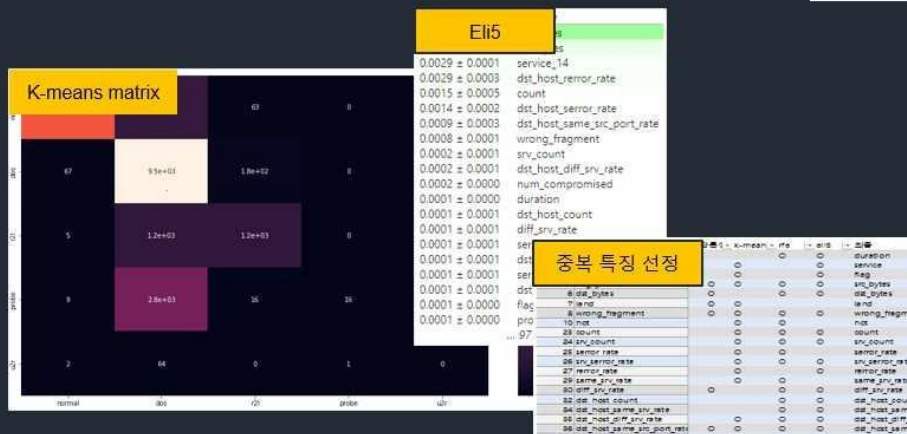
DoS Probe

DDoS BruteForce

NSL-KDD

개발 환경 및 개발 내용

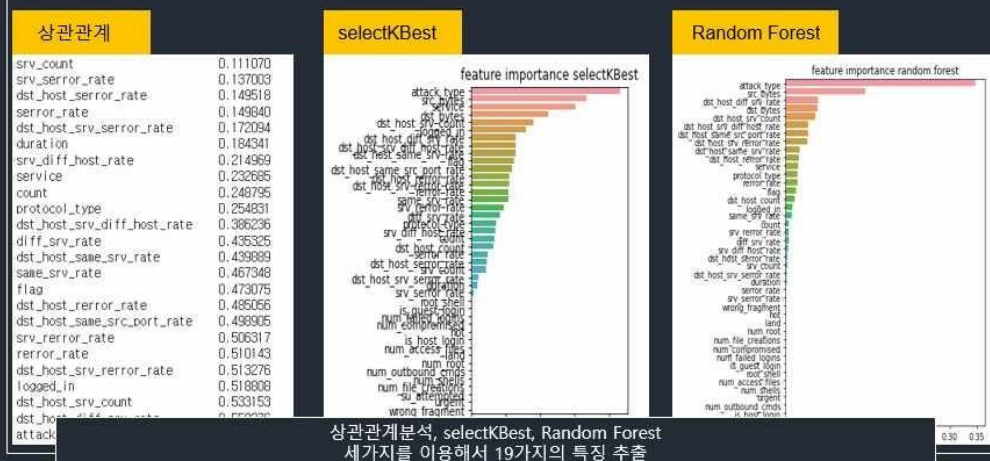
DoS



DoS 연관 특징, K-means, RFE, E15를 사용하여
2가지 이상 중복되는 22가지의 특징 추출

개발 환경 및 개발 내용

Probe



상관관계분석, selectKBest, Random Forest
세가지를 이용해서 19가지의 특징 추출

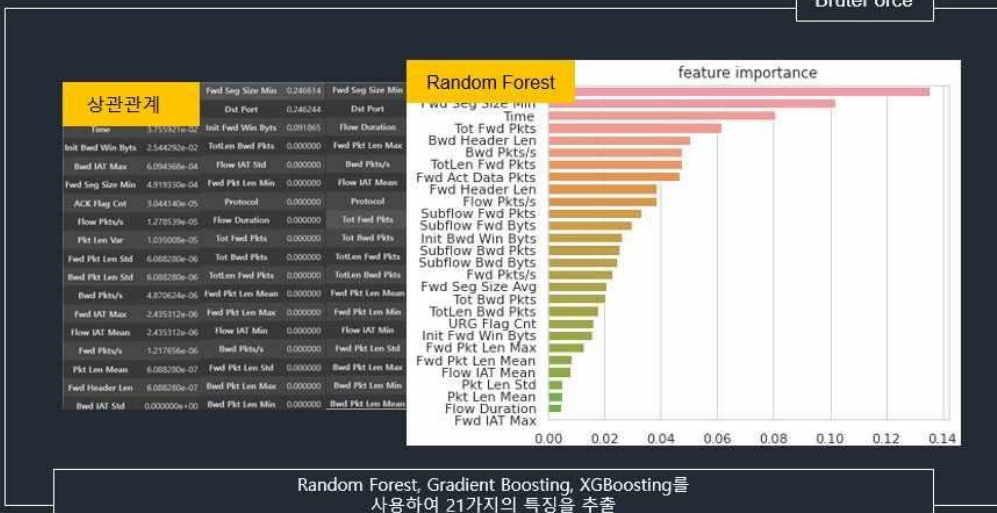
개발 환경 및 개발 내용

DDoS



개발 환경 및 개발 내용

BruteForce



개발 환경 및 개발 내용

DoS

```
def create_model():
    K.clear_session()
    model = Sequential()
    model.add(LSTM(4, input_shape=(28, 1)))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
model = create_model()
model.summary()
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 4)	96
dense (Dense)	(None, 2)	10
Total params: 106		
Trainable params: 106		
Non-trainable params: 0		

```
Epoch 745: saving model to ./LSTM_745.h5
197/825 [====>.....] - ETA: 10s - loss: 0.0156 - accuracy: 0.9951
Epoch 745: saving model to ./LSTM_745.h5
296/825 [=====>.....] - ETA: 9s - loss: 0.0149 - accuracy: 0.9953
Epoch 745: saving model to ./LSTM_745.h5
396/825 [=====>.....] - ETA: 7s - loss: 0.0153 - accuracy: 0.9953
Epoch 745: saving model to ./LSTM_745.h5
496/825 [=====>.....] - ETA: 5s - loss: 0.0151 - accuracy: 0.9954
Epoch 745: saving model to ./LSTM_745.h5
597/825 [=====>.....] - ETA: 4s - loss: 0.0152 - accuracy: 0.9953
Epoch 745: saving model to ./LSTM_745.h5
699/825 [=====>.....] - ETA: 2s - loss: 0.0158 - accuracy: 0.9951
Epoch 745: saving model to ./LSTM_745.h5
798/825 [=====>.....] - ETA: 0s - loss: 0.0159 - accuracy: 0.9951
```

30000번 학습을 실행했을 때,
loss값은 0.0149,
accuracy: 0.9953로 가장 성능이 좋았다.

개발 환경 및 개발 내용

Probe

```
# DNN
```

```
dnn = Sequential()
dnn.add(Dense(64, activation='relu', input_shape=(19,)))
dnn.add(Dropout(0.1))
dnn.add(Dense(32, activation='relu'))
dnn.add(Dropout(0.1))
dnn.add(Dense(16, activation='relu'))
dnn.add(Dropout(0.1))
dnn.add(Dense(8, activation='relu'))
dnn.add(Dropout(0.1))
dnn.add(Dense(4, activation='relu'))
dnn.add(Dropout(0.1))
dnn.add(Dense(1, activation='sigmoid'))
dnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	1280
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 8)	136
dropout_3 (Dropout)	(None, 8)	0
dense_4 (Dense)	(None, 4)	36
dense_5 (Dense)	(None, 1)	5
Total params: 4,065		
Trainable params: 4,065		
Non-trainable params: 0		

```
320/100 [====>.....] - 2s 560/step - loss: 0.0832 - accuracy: 0.9963 - val_loss: 0.2250 - val_accuracy: 0.9543
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0840 - accuracy: 0.9965 - val_loss: 0.2080 - val_accuracy: 0.9541
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0839 - accuracy: 0.9969 - val_loss: 0.2543 - val_accuracy: 0.9546
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0870 - accuracy: 0.9980 - val_loss: 0.2219 - val_accuracy: 0.9543
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0865 - accuracy: 0.9961 - val_loss: 0.2093 - val_accuracy: 0.9539
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0869 - accuracy: 0.9985 - val_loss: 0.2093 - val_accuracy: 0.9536
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0840 - accuracy: 0.9938 - val_loss: 0.2024 - val_accuracy: 0.9538
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0866 - accuracy: 0.9992 - val_loss: 0.2040 - val_accuracy: 0.9536
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0869 - accuracy: 0.9991 - val_loss: 0.1779 - val_accuracy: 0.9536
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0866 - accuracy: 0.9964 - val_loss: 0.1817 - val_accuracy: 0.9536
Epoch 560/10000
320/100 [====>.....] - 2s 560/step - loss: 0.0867 - accuracy: 0.9984 - val_loss: 0.1935 - val_accuracy: 0.9533
```

160000번 학습을 실행했을 때,
loss값은 0.0832,
accuracy: 0.9741로 가장 성능이 좋았다.

개발 환경 및 개발 내용

DDOS

```
model = Sequential()

model.add(LSTM(128, input_shape=(1, 21), return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(128))
model.add(Dropout(0.1))

model.add(Dense(32, activation='tanh'))
model.add(Dropout(0.2))

model.add(Dense(1, act

opt = tf.keras.optimiz

model.compile(
    loss='binary_crossentropy',
    optimizer=opt,
    metrics=['accuracy']
)
```

Layer (type)	Output Shape	Param #
cu_dnnlstm (CuDNNLSTM)	(None, 1, 128)	77312
dropout (Dropout)	(None, 1, 128)	0
cu_dnnlstm_1 (CuDNNLSTM)	(None, 128)	133096
dropout_1 (Dropout)	(None, 128)	0
dense (Dense)	(None, 32)	4128
dropout_2 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33
Total params: 213,569		
Trainable params: 213,569		
Non-trainable params: 0		

```
Epoch 1/100 ..... - 1s 8ms/step - loss: 0.6574 - accuracy: 0.6883 - val_loss: 0.6196 - val_accuracy: 0.7000
Epoch 2/100 ..... - 0s 4ms/step - loss: 0.5800 - accuracy: 0.7105 - val_loss: 0.5294 - val_accuracy: 0.7269
Epoch 3/100 ..... - 0s 4ms/step - loss: 0.5118 - accuracy: 0.7342 - val_loss: 0.4602 - val_accuracy: 0.7381
Epoch 4/100 ..... - 0s 4ms/step - loss: 0.4595 - accuracy: 0.7841 - val_loss: 0.4023 - val_accuracy: 0.8050
Epoch 5/100 ..... - 0s 4ms/step - loss: 0.4115 - accuracy: 0.8005 - val_loss: 0.3568 - val_accuracy: 0.8181
Epoch 6/100 ..... - 0s 4ms/step - loss: 0.3810 - accuracy: 0.8128 - val_loss: 0.3225 - val_accuracy: 0.8531
Epoch 7/100 ..... - 0s 4ms/step - loss: 0.3541 - accuracy: 0.8341 - val_loss: 0.2993 - val_accuracy: 0.8838
Epoch 8/100 ..... - 0s 4ms/step - loss: 0.3542 - accuracy: 0.8542 - val_loss: 0.2791 - val_accuracy: 0.8969
Epoch 9/100 ..... - 0s 4ms/step - loss: 0.3378 - accuracy: 0.8907 - val_loss: 0.2543 - val_accuracy: 0.9062
```

88000번 학습을 실행했을 때,
loss값은 0.5674,
accuracy: 0.9399로 가장 성능이 좋았다.

개발 환경 및 개발 내용

Brute Force

```
def a_list():
2   model = Sequential()
3   model.add(CuDNNGRU(64, input_shape=(1, 21), return_sequences=True))
4   model.add(Dropout(0.2))
5   model.add(CuDNNGRU(128, return_sequences=False))
6   model.add(Dropout(0.2))
7   model.add(Dense(1))
8   model.add(Activation('sigmoid'))
9
10  #
11  # accuracy]]

Layer (type)      Output Shape      Param #
-----
cu_dnngru (CuDNNGRU) (None, 1, 64)      16704
dropout (Dropout)   (None, 1, 64)      0
cu_dnngru_1 (CuDNNGRU) (None, 128)        74496
dropout_1 (Dropout) (None, 128)        0
dense (Dense)       (None, 1)          129
activation (Activation) (None, 1)          0

Total params: 91,329
Trainable params: 91,329
Non-trainable params: 0
```

```
.....-30000.....
9/9 ..... - 0s 3ms/step - loss: 3.9485 - accuracy: 0.9300
.....-30000.....
9/9 ..... - 0s 3ms/step - loss: 3.7696 - accuracy: 0.9300
.....-27000.....
9/9 ..... - 0s 3ms/step - loss: 3.9036 - accuracy: 0.8963
.....-20000.....
9/9 ..... - 0s 3ms/step - loss: 5.2772 - accuracy: 0.0698
.....-20000.....
9/9 ..... - 0s 3ms/step - loss: 4.3436 - accuracy: 0.8876
.....-30000.....
9/9 ..... - 0s 3ms/step - loss: 4.8827 - accuracy: 0.9300
.....-91000.....
9/9 ..... - 0s 3ms/step - loss: 4.3514 - accuracy: 0.9070
.....-30000.....
9/9 ..... - 0s 4ms/step - loss: 3.1072 - accuracy: 0.9300
.....-30000.....
9/9 ..... - 0s 3ms/step - loss: 9.6917 - accuracy: 0.1473
.....-34000.....
9/9 ..... - 0s 4ms/step - loss: 14.5943 - accuracy: 0.0584
.....-35000.....
9/9 ..... - 0s 4ms/step - loss: 0.8029 - accuracy: 0.000e+00
.....-30000.....
9/9 ..... - 0s 3ms/step - loss: 7.0244 - accuracy: 0.0009
```

320000번 학습을 실행했을 때,
loss값은 3.1072,
accuracy: 0.9380로 가장 성능이 좋았다.

개발 시스템 운영

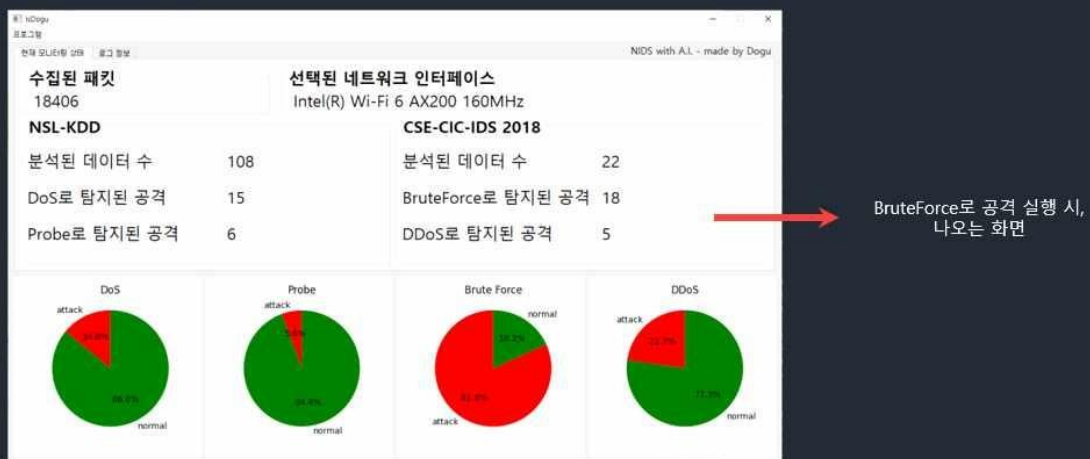
첫 실행화면

네트워크 선택 후 프로그램 시작



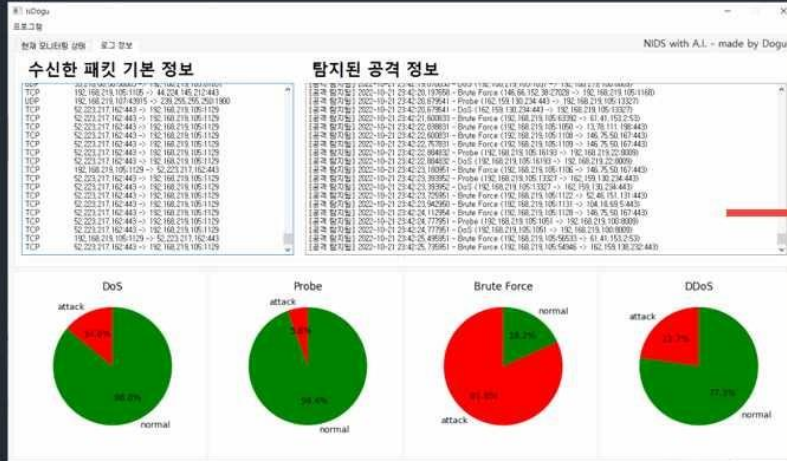
개발 시스템 운영

패킷 받고 공격이 탐지됐을때 화면



개발 시스템 운영

로그



결론 및 기대효과

- 4가지의 공격방법을 선정하여 네트워크 침입에 대해 인공지능을 사용하여 공격에 대해 빠르게 탐지할 수 있는지를 살펴보았다
- 모델별, 학습횟수에 따라 성능이 차이나는 것을 확인하였다
- 실제 패킷과 모델을 연결 하여 공격을 시도했을 때 각 공격에 대해 높은 탐지율을 보여주었다
- 지속 적인 연구를 통해 아직 알려지지 않은 네트워크 공격들과 새롭게 등장하는 공격들을 잡아 낼 수 있을 것이다.

감사합니다.