악성 네트워크 탐지를 위한 AI 모델 및 XAI 활용

2025. 04. 07

KAIST CSRC

CYBER SECURITY RESERCH CENTER

목차

✓데이터 전처리 & 모델링 기초

시간	주제	내용 개요
10:00 ~ 10:50	네트워크 보안과 악성 트래픽 개요	- 악성 네트워크 탐지 개요 및 필요성 - 기존 탐지 방식(포트 기반, 패턴 매칭, 머신러닝 등)의 한계 - 최신 AI 기반 탐지 기술 소개
11:00 ~ 11:50	네트워크 트래픽 데이터 이해	- 패킷, 플로우, 세션의 개념 - 정형 데이터 vs 비정형 데이터 (페이로드 분석의 중요성) CICFlowmeter - 데이터셋 구성 및 샘플 예제 탐색
12:00 ~ 13:00	점심시간	
13:00 ~ 13:50	네트워크 트래픽 데이터 이해(2)	CICFlowmeter - 데이터셋 구성 및 샘플 예제 탐색
14:00 ~ 14:50	자연어 처리(NLP) 모델 개요	- NLP를 이용한 악성 탐지 개념 이해 - 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식
15:00 ~ 15:50	모델링 실습: BERT 기반 악성 탐지 모델 구축	- BERT를 활용한 악성 페이로드 탐지 실습 - 모델 학습 및 성능 평가 (Precision, Recall 등)
16:00 ~ 16:50	모델 성능 개선 전략	- 데이터 균형 조정 (SMOTE 등 활용) - 미세 조정(Fine-tuning) 기법 - 하이브리드 모델 적용 방안 (MLP + BERT 등)

네트워크 보안과 악성 트래픽 개요

1. 악성 네트워크 탐지 개요 및 필요성

✓악성 네트워크 트래픽은 단순한 시스템 침해를 넘어 국가 안보, 공공 서비스, 개인정보에 이르기까지 광범위한 피해 유발

✓C2 통신, 정보 유출, RAT 제어 등은 모두 패킷 레벨에서 관찰 가능

✓탐지가 어려운 이유: 암호화, 난독화, 표준 프로토콜 위장

사례	설명	출처
폴란드 총리 소속 정당 사이버 공격	폴란드 총리 도날드 투스크의 시민연단당(PO)이 외부 세력의 사이버 공격을 받아 IT 시스템이 침해됨	link
우크라이나 국영 철도사 사이버 공격	우크라이나 국영 철도사인 우크잘리즈니챠(Ukrzaliznytsia)가 대규모 사이버 공격으로 온라인 서비스가 중단됨.	link
이탈리아 외무부 및 공항 사이버 공격	친러시아 해커 그룹이 이탈리아 외무부와 밀라노 주요 공항 등 약 10개의 공식 웹사이트를 DDoS 공격으로 마비시킴.	link
카도카와 및 니코니코 사이버 공격	러시아 연계 해커 그룹 'BlackSuit'이 일본의 카도카와와 니코니코에 랜섬웨어 공격을 감행하여 25만여 명의 사용자 데이터 유출.	link

2. 실제 사례 분석(폴란드 총리 소속 정당 사이버 공격)

✓공격 방식

- 폴란드 총리 도날드 투스크의 시민연단당(PO) 소속 시스템이 외부 세력의 사이버 공격을 받아 침해됨 → 정당 내부 시스템 및 IT 인프라 접근 시도→ 선거 개입 및 정치 정보 탈취 가능성

✓탐지 가능성

- 비인가 외부 IP의 내부 시스템 접근
- 불규칙적인 로그인 시도, 이상 세션 지속
- 내부 정보 외부 전송 시 FTP/SMTP 등 비정상 트래픽 확인

✓탐지 포인트

- SMTP, HTTP POST, FTP 등에서 내부 문서 유출 시도 식별
- entropy 기반으로 압축 또는 암호화된 페이로드 식별 가능
- 의심 C2 도메인 혹은 커맨드 문자열 탐지

2. 실제 사례 분석(우크라이나 국영 철도사 사이버 공격)

✓공격 방식

- 철도청 웹/서비스 대상 **DDoS** 트래픽 유입 → **DNS, HTTP** 서비스에 대한 비정상 연결 폭주

✓탐지 가능성

- 네트워크 레벨에서 동시 접속 수 / 응답 실패율 / 특정 포트 트래픽 급증 감지 가능
- 평소 대비 급격한 트래픽 변화 → **AI** 기반 모델로 학습 가능

✓탐지 포인트

- **Flow** 기반 탐지 (접속 건수/세션 시간/패킷 수 이상 탐지)
- **HTTP/DNS** 연속 요청 확인
- 단순 반복 **GET**이나 요청 **flood** 패턴 식별

2. 실제 사례 분석(이탈리아 외무부 및 공항 사이버 공격)

✓공격 방식

- 친러 해커 그룹이 이탈리아 외무부 및 공항 웹사이트에 **DDoS** 공격→ 서비스 불능 상태 **(**웹 다운, 응답 지연 등**)**

✓탐지 가능성

- 단기간 집중 접속
- 비정상 **SNI, TLS** 핸드셰이크 과다
- **HTTP** 혹은 **TLS** 내 연결 과다 감지

✓탐지 포인트

- **TLS/HTTP** 탐지 및 **Payload**
- 특정 **User-Agent, SNI, IP** 기반 의심 트래픽 분석

2. 실제 사례 분석(카도카와 및 니코니코 사이버 공격)

✓공격 방식

- **BlackSuit** 랜섬웨어 감염 → 내부 시스템 암호화 및 정보 탈취

✓탐지 가능성

- 공격 초기에 **C2** 연결, **EXE** 전달, 암호화 전 이력 식별 가능
- 이메일/**FTP**를 통한 악성 페이로드 전달

✓탐지 포인트

- **Payload**기반 **PE header, powershell, cmd, base64** 등 문자열 탐지
- 암호화 분석을 통한 의심
- **SMTP/HTTP** 트래픽 기반 → 랜섬노트 전달 흔적 확인 가능

3. 기존 탐지 방식

✓**“과거부터 현재까지 가장 널리 사용되는 네트워크 보안 기법”**

✓**특정 패턴, 룰, 연결 형태를 기반으로 악성 행위를 탐지**

- 매칭: 공격 노스를 매칭 및 마이크로 저장
- 공격 형태에 따라 시그니처 상유 정의
- 현재 포트, 메시지 형식에 해당

탐지 방식	설명	대표 도구
시그니처 기반 탐지	저의된 메시지와 룰을 비교하여 탐지	Snort, Suricata
플로우 기반 탐지	데이터 접속 형태를 분석해 이상 행위 검출	NetFlow, Zeek
정적/동적 분석	파일 요소 유효성, 신호 표현 분석	YARA, Cuckoo Sandbox

3. 기존 탐지 방식

도구	유형	설명
Snort	시그니처 기반 IDS	룰 기반 탐지, 텍스트 기반 정책 정의
Suricata	시그니처 + 멀티스레드 IDS/IPS	고속 처리 + 네트워크 프로토콜 분석 지원
Zeek(Bro)	플로우 기반 이상탐지	로그 생성, 세션 기반 행동 분석
NetFlow/IPFIX	통계 기반 흐름 분석	연결 수, 패킷 크기, 시간 기반 이상 탐지
YARA	정적 분석 룰 엔진	악성코드 내부 구조 탐지
Cuckoo Sandbox	동적 분석 플랫폼	악성코드 실행 후 행동 패턴 기록 및 분석

✓**주요 탐지 도구 적용 예시**

- **Snort:** 기업 방화벽 연계 탐지 정책 구축
- **Zeek:** 기관 단위의 장기 접속, 세션 기반 분석
- **NetFlow: VPN/**내부망에서 이상 접속 감시
- **Cuckoo:** 신규 의심 실행파일 격리 후 분석

3. 기존 탐지 방식

✓시그니처 기반 탐지는 이미 알려진 공격 패턴을 룰로 정의하여 탐지는 방식

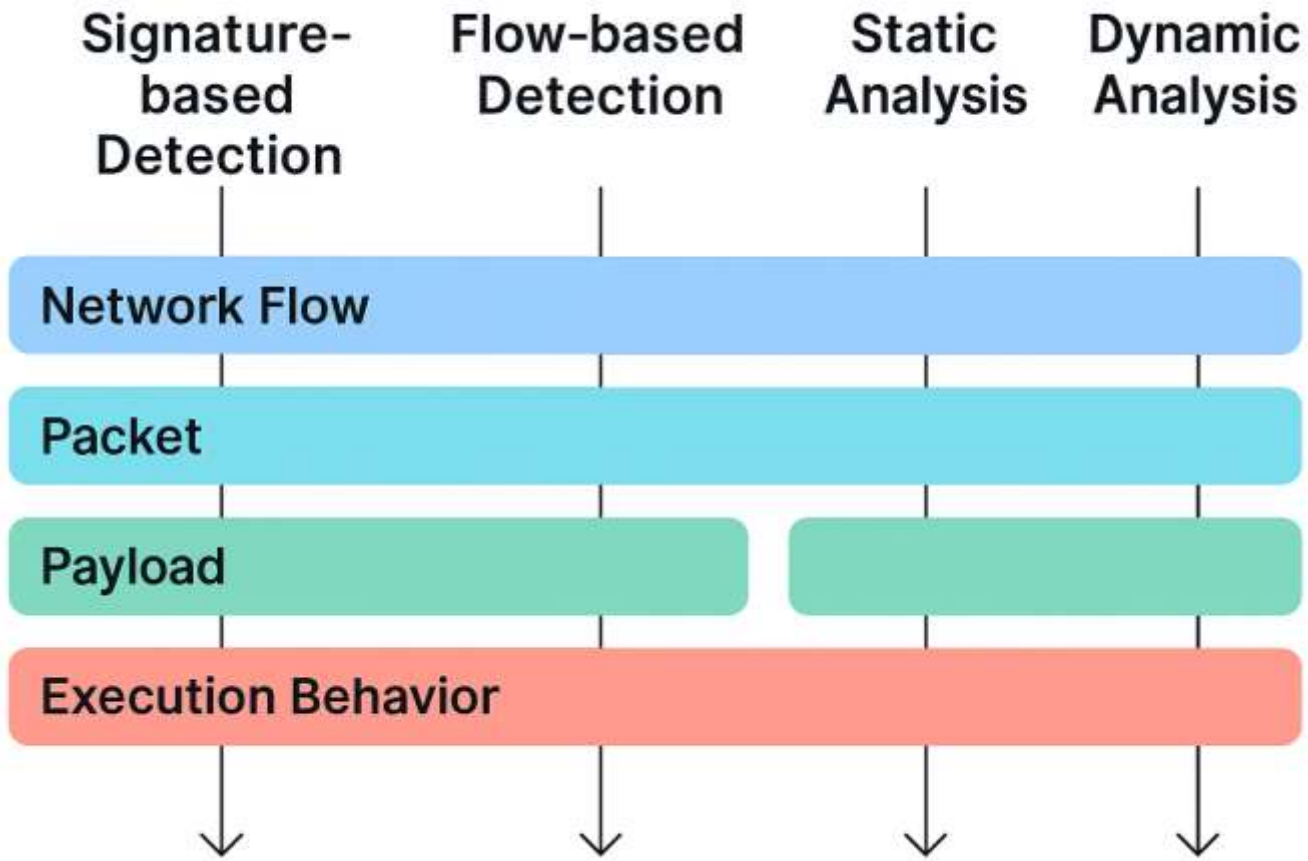
- 탐지 정확도는 높음
- 그러나 새로운 공격 탐지에 대응 불가
- 간단한 문자열 우회에도 취약

✓플로우 기반 탐지는 트래픽의 정보(패킷 수, 시간, 포트 등)를 기반으로 탐지하는 방식

- 성능에 부담이 적고 암호화된 트래픽도 탐지 가능성이 있음
- 그러나 실제 내용(content)에 대한 분석은 불가능해 오탐률이 있음

3. 기존 탐지 방식

- ✓ **Signature-based Detection**→ 주로 정형화된 패턴 매칭 기반으로, 네트워크 흐름(Network Flow), 패킷, 페이로드 단위까지 탐지 가능→ 빠르지만 새로운 위협이나 변조된 공격에 취약
- ✓ **Flow-based Detection**→ 통계 기반 트래픽 흐름 분석 (예: NetFlow), 패킷 및 네트워크 흐름 중심→ 암호화된 트래픽도 분석 가능하나, 콘텐츠 기반 분석은 어려움
- ✓ **Static Analysis**→ 페이로드 수준까지 들어가 정적인 코드/데이터 구조 분석→ 난독화, 압축 등에 우회될 수 있음
- ✓ **Dynamic Analysis**→ 코드 실행 결과를 기반으로 행위(Execution Behavior) 수준까지 확인 가능→ 리소스가 많이 소모되지만 신종 공격 탐지에 유리



3. 네트워크 기반 악성 탐지 방법 비교

✓**기존 탐지 방식 우회 방법**

공격기법	설명	탐지 실패 원인
문자열 변경	악성 코드 내 문자열을 유사 문자나 난독화 방식으로 변경	시그니처 기반 탐지 회피
HTTPS 암호화	HTTP대신 HTTPS로 C2 통신 수행	콘텐츠 암호화로 패킷 내용 탐지 불가
C2 서버 변조	정적 IP가 아닌, 주기적으로 바뀌는 도메인 사용	DNS 응답 기반으로 통신 시 탐지 어려움
암호/암호화된 페이로드	Zip, base64, XOR등으로 악성 페이로드 감춤	정적 분석 및 시그니처 탐지 무력

```
# 탐지용 룰
content: "powershell.exe -enc"

# 우회 예시
powershe11.exe -enc ABCDE==
```

4. 최신 AI 기반 탐지 기술

✓ **왜 AI 기반 탐지인가?**

- 기존 시그니처 기반 탐지의 우회 취약성을 극복
- 대규모 네트워크 트래픽에서 비정상 행위 패턴 학습
- 패킷/플로우/페이로드 기반의 정밀 분석 가능

AI model	특징	활용점
Depp Packet Inspection + AI	패킷 내부를 분석하여 비정상 탐지	암호화 전 트래픽 탐지 가능
Flow-based Anomaly Detection(ML)	트래픽 메타데이터 기반 이상 탐지	DdoS, 봇넷
Autoencoder/GAN 기반	정상 트래픽 학습 -> 이상탐지	Zero-day탐지
NLP 기반 Payload 분석	텍스트처럼 패킷/페이로드 분석	C2 명령, 스크립트 탐지

1. 개요

✓학습 목표 (Learning Objectives)

- 네트워크에서 수집되는 트래픽 데이터의 기본 구성에 대한 이해
- 패킷, 플로우, 세션의 개념을 구분
- 정형 vs 비정형 트래픽 데이터를 구분하고, 각각의 활용 방향
- 이후 탐지 모델에 활용될 수 있는 데이터셋 구성 방식에 대한 이해 기반 마련

✓왜 네트워크 트래픽 데이터를 이해해야 하는가?

- 악성 네트워크 탐지 모델의 핵심은 데이터 기반의 이상 탐지
- “정확한 탐지를 위해선 어떤 데이터를 보고 어떤 특징을 뽑을지” 이해가 중요
- 단순 패킷 수치만으로는 고도화된 공격 탐지가 어렵고, 비정형 데이터(페이로드, 명령어 등) 분석이 요구됨

1. 개요

✓패킷

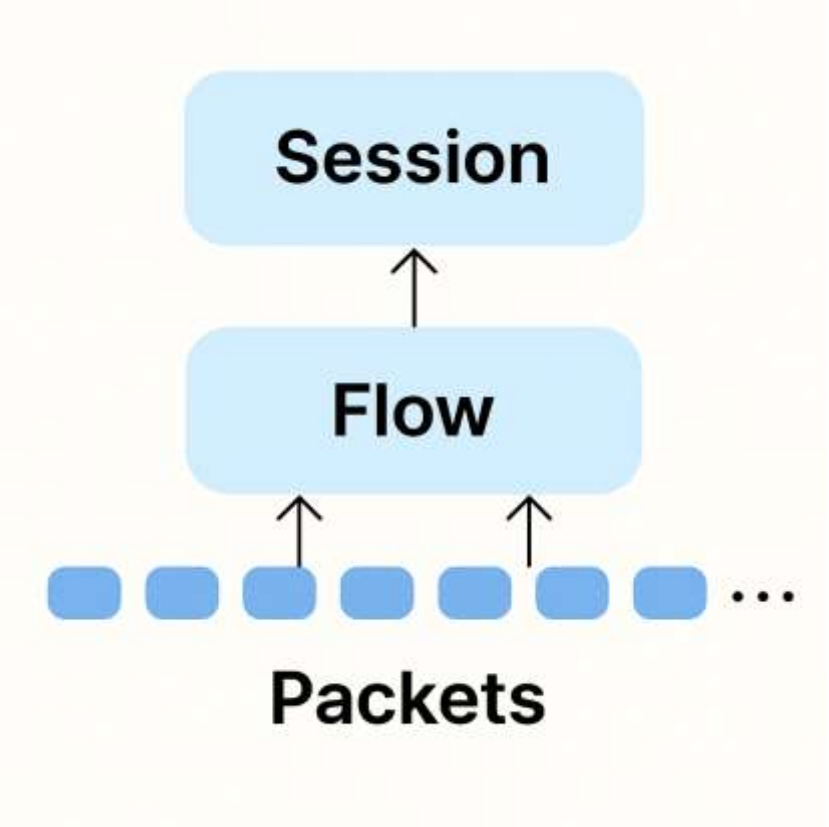
- 네트워크를 통해 전달되는 가장 작은 데이터 단위
- 헤더 + 페이로드(payload)로 구성
- 각 패킷은 송신지/수신지 IP, 포트, 프로토콜 정보를 포함

✓플로우

- 일정 시간 동안 동일한 세션에서 발생한 연속된 패킷들의 집합
- 일반적으로 6-튜플 (srcIP, dstIP, srcPort, dstPort, srcMac, destMac)으로 정의
- 통계 기반 이상 탐지(예: 평균 패킷 크기, 지속시간 등)에 자주 사용

✓세션

- 하나의 논리적 통신 단위(예: 사용자 로그인 과정 전체, 파일 다운로드 전체)
- TCP/UDP의 연결 기반 흐름을 포괄
- 하나의 세션 안에 다수의 플로우가 포함될 수 있음



2. 정형 데이터 vs 비정형 데이터

✓정형 데이터란?

- 테이블 형태로 정리된 구조화된 데이터
- 예: 플로우 로그 (src_ip, dst_ip, src_port, packet_count 등)
- 쉽게 머신러닝 입력값으로 사용 가능
- 대부분 메타 정보 기반 탐지에 활용

✓비정형 데이터란?

- 구조화되지 않은 원시 데이터
- 예: 네트워크 Payload, 이메일 내용, HTTP 헤더, 인코딩된 파일 등
- 공격자가 숨기거나 암호화하여 탐지를 어렵게 함

✓정형 데이터는 “누가, 어디로, 얼마나”를 확인 할 수 있으며 비정형 데이터는 “무엇이, 어떻게”를 확인할 수 있음

- 기존 탐지(정형 데이터)는 패턴을 놓칠 수 있음

2. 정형 데이터 vs 비정형 데이터

src_ip	dst_ip	src_port	dst_port	src_mac	dst_mac	protocol	timestamp	flow_durat	flow_byts_	flow_pkts_	fwd_pkts_s	bwd_pkts_	tot_fwd_pk	tot_bwd_p	totlen_fwd	totlen_bw	fwd_pkt_le	fwd_pkt_le	fwd_pkt_le	fwd_pkt_le	bwd_pkt_l	bwd_pkt_l	bwd_pkt_l
192.168.12	173.247.25	49346	80	52:54:00:f6	52:54:00:9b	6	1.37E+09	84361	52844.32	130.392	59.26909	71.12291	5	6	1238	1830	526	54	247.6	227.3549	648	54	305
192.168.12	93.171.172	49357	80	52:54:00:f6	52:54:00:9b	6	1.37E+09	974179	3149.319	11.29156	5.132527	6.159032	5	6	1238	1830	526	54	247.6	227.3549	648	54	305
192.168.12	199.195.24	49359	8000	52:54:00:f6	52:54:00:9b	6	1.37E+09	1676656	4146.945	13.12136	5.964253	7.157103	10	12	2519	4434	612	54	251.9	238.8608	1434	54	369.5
192.168.12	199.195.24	49360	8000	52:54:00:f6	52:54:00:9b	6	1.37E+09	1297022	3211.202	12.33595	5.396979	6.938973	7	9	2251	1914	524	54	321.5714	228.3023	496	54	212.6667
192.168.12	199.195.24	49361	8000	52:54:00:f6	52:54:00:9b	6	1.37E+09	1312480	2663.66	12.19066	6.095331	6.095331	8	8	1855	1641	526	54	231.875	223.5008	380	54	205.125
192.168.12	199.195.24	49362	8000	52:54:00:f6	52:54:00:9b	6	1.37E+09	1999723	3429.975	11.00152	5.000693	6.000831	10	12	3773	3086	804	54	377.3	275.0567	1036	54	257.1667
192.168.12	199.195.24	49363	8000	52:54:00:f6	52:54:00:9b	6	1.37E+09	1546237	3172.864	12.2879	5.173851	7.114045	8	11	2759	2147	520	54	344.875	222.2484	498	54	195.1818
192.168.12	199.195.24	49364	8000	52:54:00:f6	52:54:00:9b	6	1.37E+09	1370182	34982.94	45.24946	17.51592	27.73354	24	38	2730	45203	525	54	113.75	155.0952	1434	54	1189.553
192.168.12	199.195.24	49367	8000	52:54:00:f6	52:54:00:9b	6	1.37E+09	647751	40830.5	57.12071	24.70085	32.41987	16	21	1238	25210	368	54	77.375	75.23868	1434	54	1200.476
10.8.18.10	10.8.18.1	53399	53	00:08:02:1	20:e5:2a:b	17	1.47E+09	1035860	370.7065	3.861526	1.930763	1.930763	2	2	176	208	88	88	88	0	104	104	104
10.8.18.10	10.8.18.1	64480	53	00:08:02:1	20:e5:2a:b	17	1.47E+09	26433	11122.46	75.663	37.8315	37.8315	1	1	75	219	75	75	75	0	219	219	219
10.8.18.10	74.117.178	49171	443	00:08:02:1	20:e5:2a:b	6	1.47E+09	1092822	5991.827	13.72593	6.405435	7.320497	7	8	773	5775	236	60	110.4286	77.8751	1514	60	721.875
10.8.18.10	74.117.178	49172	443	00:08:02:1	20:e5:2a:b	6	1.47E+09	1501439	5119.089	11.9885	5.328222	6.660277	8	10	1216	6470	443	60	152	131.9233	1514	60	647
10.8.18.10	10.8.18.1	50312	53	00:08:02:1	20:e5:2a:b	17	1.47E+09	35584	4777.428	56.20504	28.10252	28.10252	1	1	77	93	77	77	77	0	93	93	93
10.8.18.10	204.155.14	49174	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	369405	503.5124	8.121168	5.414112	2.707056	2	1	126	60	66	60	63	3	60	60	60
10.8.18.10	204.155.14	49173	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	1765868	766.7617	3.964056	2.265175	1.698881	4	3	662	692	476	60	165.5	179.284	572	60	230.6667
10.8.18.10	10.8.18.1	57601	53	00:08:02:1	20:e5:2a:b	17	1.47E+09	441742	484.4457	4.52753	2.263765	2.263765	1	1	80	134	80	80	80	0	134	134	134
10.8.18.10	185.93.0.17	49176	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	408902	454.8767	7.336721	4.891148	2.445574	2	1	126	60	66	60	63	3	60	60	60
10.8.18.10	185.93.0.17	49175	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	845020	3424.771	11.83404	5.91702	5.91702	5	5	962	1932	716	60	192.4	261.8103	1514	60	386.4
10.8.18.10	65.181.125	49170	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	6790691	181.1303	1.325344	0.736302	0.589042	5	4	596	634	350	60	119.2	115.4234	454	60	158.5
10.8.18.10	65.181.125	49169	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	21738130	14.07665	0.230011	0.138006	0.092004	3	2	186	120	66	60	62	2.828427	60	60	60
10.8.18.10	10.8.18.1	51232	53	00:08:02:1	20:e5:2a:b	17	1.47E+09	1467883	237.0761	2.725013	1.362506	1.362506	2	2	158	190	79	79	79	0	95	95	95
10.8.18.10	65.181.125	49177	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	16085754	107976.5	106.8026	30.39957	76.40301	489	1229	29613	1707271	131	60	60.55828	5.453388	1514	60	1389.155
10.8.18.10	65.181.113	49178	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	22802413	387.4151	2.19275	0.8771	1.31565	20	30	3148	5686	307	60	157.4	84.95434	407	60	189.5333
10.8.18.10	10.8.18.1	137	137	00:08:02:1	20:e5:2a:b	17	1.47E+09	12103887	81.79191	0.743563	0.743563	0	9	0	990	0	110	110	110	0	0	0	0
10.8.18.10	65.181.113	49182	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	14715134	191.7074	0.815487	0.407743	0.407743	6	6	1243	1578	937	60	207.1667	326.3987	669	60	263
10.8.18.10	185.93.0.17	49176	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	404097	1041.829	9.898613	4.949307	4.949307	2	2	120	301	60	60	60	0	241	60	150.5
10.8.18.10	204.155.14	49173	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	404137	593.858	9.897634	4.948817	4.948817	2	2	120	120	60	60	60	0	60	60	60
10.8.18.10	204.155.14	49174	80	00:08:02:1	20:e5:2a:b	6	1.47E+09	404000	594.0594	9.90099	4.950495	4.950495	2	2	120	120	60	60	60	0	60	60	60

2. 정형 데이터 vs 비정형 데이터

✓**Detected Protocol: 자동 분석된 프로토콜 (예: HTTP, FTP 등)**

✓**Entropy: 페이로드의 엔트로피(난이도) 수치→ 암호화/압축 여부, 우회 가능성 판단 지표**

✓**Decoded Payload: ASCII 또는 UTF 기반으로 사람이 읽을 수 있는 문자열로 변환된 내용→ 실제 악성 코드, 명령, 경로 등이 포함 될 수 있음**

3. 데이터 전처리 with CicFlowmeter

✓정형 데이터 변환 흐름

- 패킷 캡처 (PCAP)Wireshark, tcpdump 등으로 수집
- 패킷 기반 데이터를 Flow 단위로 그룹화
- 정형 데이터 생성 (CSV)
- 각 Flow에 대해 80여 개의 특징 추출(예: Flow Duration, Packet Length Mean, Fwd Header Length Total 등)

3. 데이터 전처리 with CicFlowmeter

Feature	내용	Feature	내용	Feature	내용
Destination Port	목적지 포트	Bwd Packet Length Std	반대방향 패킷 길이의 표준 편차	Bwd IAT Mean	반대방향 플로우의 인터-도착 시간 평균
Flow Duration	플로우의 지속 시간 (밀리초)	Flow Bytes/s	플로우의 바이트 전송 속도 (초당 바이트)	Bwd IAT Std	반대방향 플로우의 인터-도착 시간 표준 편차
Total Fwd Packets	전방향(forward)으로 전송된 총 패킷 수	Flow Packets/s	플로우의 패킷 전송 속도 (초당 패킷)	Bwd IAT Max	반대방향 플로우의 인터-도착 시간 최대값
Total Backward Packets	반대방향(backward)으로 전송된 총 패킷 수	Flow IAT Mean	플로우 간 인터-도착 시간(Inter-Arrival Time)의 평균 (밀리초)	Bwd IAT Min	반대방향 플로우의 인터-도착 시간 최소값
Total Length of Fwd Packets	전방향으로 전송된 패킷들의 총 길이 (바이트)	Flow IAT Std	플로우 간 인터-도착 시간의 표준 편차	Fwd PSH Flags	전방향 패킷에 설정된 PSH 플래그 수
Total Length of Bwd Packets	반대방향으로 전송된 패킷들의 총 길이 (바이트)	Flow IAT Max	플로우 간 인터-도착 시간의 최대값	Bwd PSH Flags	반대방향 패킷에 설정된 PSH 플래그 수
Fwd Packet Length Max	전방향 패킷의 최대 길이 (바이트)	Flow IAT Min	플로우 간 인터-도착 시간의 최소값	Fwd URG Flags	전방향 패킷에 설정된 URG 플래그 수
Fwd Packet Length Min	전방향 패킷의 최소 길이 (바이트)	Fwd IAT Total	플로우 간 인터-도착 시간(Inter-Arrival Time)의 평균 (밀리초)	Bwd URG Flags	반대방향 패킷에 설정된 URG 플래그 수
Fwd Packet Length Mean	전방향 패킷의 평균 길이 (바이트)	Fwd IAT Mean	전방향 플로우의 인터-도착 시간 평균	Fwd Header Length	전방향 패킷의 헤더 길이 총합.
Fwd Packet Length Std	전방향 패킷 길이의 표준 편차	Fwd IAT Std	전방향 플로우의 인터-도착 시간 표준 편차	Bwd Header Length	반대방향 패킷의 헤더 길이 총합
Bwd Packet Length Max	반대방향 패킷의 최대 길이 (바이트)	Fwd IAT Max	전방향 플로우의 인터-도착 시간 최대값	Fwd Packets/s	초당 전방향(forward)으로 전송된 패킷 수
Bwd Packet Length Min	반대방향 패킷의 최소 길이 (바이트)	Fwd IAT Min	전방향 플로우의 인터-도착 시간 최소값	Bwd Packets/s	초당 반대방향(backward)으로 전송된 패킷 수
Bwd Packet Length Mean	반대방향 패킷의 평균 길이 (바이트)	Bwd IAT Total	반대방향 플로우의 인터-도착 총 시간	Min Packet Length	플로우 내에서 전송된 패킷들의 최소 길이(바이트)

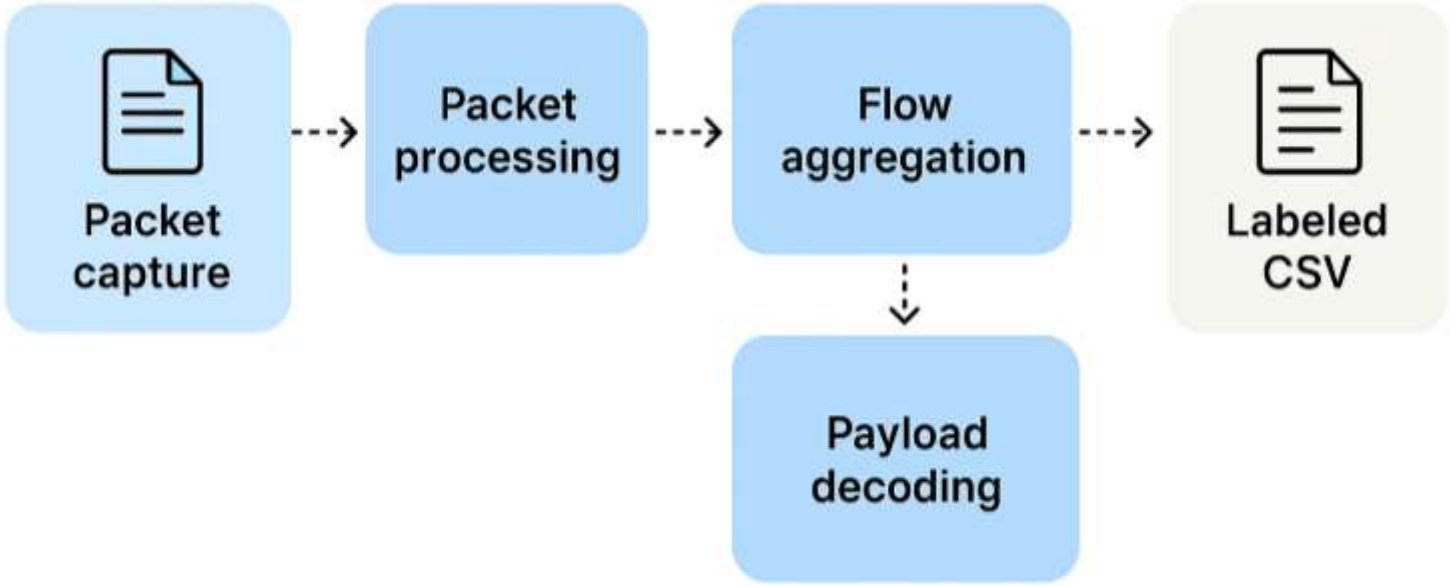
3. 데이터 전처리 with CicFlowmeter

Feature	내용	Feature	내용	Feature	내용
Max Packet Length	플로우 내에서 전송된 패킷들의 최대 길이(바이트)	Average Packet Size	플로우 내 패킷들의 평균 크기(바이트).	Init_Win_bytes_forward	전방향 초기 윈도우 크기(바이트).
Packet Length Mean	플로우 내에서 전송된 패킷들의 평균 길이(바이트)	Avg Fwd Segment Size	전방향 패킷의 평균 세그먼트 크기.	Init_Win_bytes_backward	반대방향 초기 윈도우 크기(바이트).
Packet Length Std	플로우 내에서 전송된 패킷 길이의 표준 편차	Avg Bwd Segment Size	반대방향 패킷의 평균 세그먼트 크기.	act_data_pkt_fwd	전방향 활성 데이터 패킷 수.
Packet Length Variance	플로우 내에서 전송된 패킷 길이의 분산	Fwd Avg Bytes/Bulk	전방향 평균 바이트/벌크.	min_seg_size_forward	전방향 최소 세그먼트 크기.
FIN Flag Count	전방향 패킷에 설정된 FIN 플래그 수	Fwd Avg Packets/Bulk	전방향 평균 패킷/벌크.	Active Mean	활성화된 플로우의 평균 시간.
SYN Flag Count	플로우 내에서 SYN 플래그가 설정된 패킷의 수.	Fwd Avg Bulk Rate	전방향 평균 벌크 전송 속도.	Active Std	활성화된 플로우 시간의 표준 편차.
RST Flag Count	플로우 내에서 RST 플래그가 설정된 패킷의 수.	Bwd Avg Bytes/Bulk	반대방향 평균 바이트/벌크.	Active Max	활성화된 플로우 시간의 최대값.
PSH Flag Count	플로우 내에서 PSH 플래그가 설정된 패킷의 수.	Bwd Avg Packets/Bulk	반대방향 평균 패킷/벌크.	Active Min	활성화된 플로우 시간의 최소값.
ACK Flag Count	플로우 내에서 ACK 플래그가 설정된 패킷의 수.	Bwd Avg Bulk Rate	반대방향 평균 벌크 전송 속도.	Idle Mean	유휴 플로우의 평균 시간.
URG Flag Count	플로우 내에서 URG 플래그가 설정된 패킷의 수.	Subflow Fwd Packets	서브플로우 내 전방향 패킷 수.	Idle Std	유휴 플로우 시간의 표준 편차.
CWE Flag Count	플로우 내에서 CWE 플래그가 설정된 패킷의 수.	Subflow Fwd Bytes	서브플로우 내 전방향 바이트 수.	Idle Max	유휴 플로우 시간의 최대값.
ECE Flag Count	플로우 내에서 ECE 플래그가 설정된 패킷의 수.	Subflow Bwd Packets	서브플로우 내 반대방향 패킷 수.	Idle Min	유휴 플로우 시간의 최소값.
Down/Up Ratio	다운로드 트래픽과 업로드 트래픽의 비율.	Subflow Bwd Bytes	서브플로우 내 반대방향 바이트 수.		

3. 데이터 전처리 with CicFlowmeter

✓목적 및 활용

- ML/AI 학습을 위한 정형 데이터 확보
- 공격 유형별 통계적 특성 비교 가능
- 라벨링된 학습용 CSV 구축



4. 데이터셋 구성 및 샘플 예제 탐색

✓패킷 수집

- Malware-trafics의 악성 데이터와 실시간 pcap Capture를 통해 benign .pcap 파일 수집

✓Flow 세션 구성

- on_packet_received() 함수에서 6튜플 (src/dst IP, port, MAC) 정보를 기준으로 플로우 키를 생성
- 일정 시간 이상 간격이 벌어지거나 FIN 패킷이 등장하면 새 세션으로 분리

✓페이로드 flow 단위 결합

- 각 플로우 내 패킷의 페이로드를 누적하여 하나의 플로우 페이로드로 구성
- decode_payload() 함수를 통해 HTTP, SMTP, FTP, DNS 등 다양한 프로토콜에 맞춰 비정형 데이터를 디코딩
- 이후 base64 디코딩, entropy 계산, ASCII 문자열 추출 등 XAI 기반 탐지 포인트 생성

✓특징 추출 및 CSV 저장

- garbage_collect() 함수에서 일정 시간 이상 유지된 플로우의 데이터를 정리
- 디코딩된 payload, 트래픽 통계 정보, 포트 정보 등을 정리하여 .csv 형식으로 저장 → AI 학습에 활용 가능.

5. Payload 디코딩 실습

✓실습 목표

- 네트워크 트래픽 내 비정형 데이터(페이로드)를 디코딩하여 의미 있는 정보 추출
- **decode_payload** 함수를 활용한 자동 프로토콜 감지 및 디코딩 로직 이해
- 다양한 프로토콜 **(HTTP, DNS, TLS, SMTP** 등)에 대해 사람이 읽을 수 있는 데이터 확인

✓핵심 함수

- **decode_payload:** 자동 프로토콜 감지
  - 헤더 정보, 시그니처, 엔트로피 기반 프로토콜 추정
  - 예: **DNS, HTTP, TLS, SMB, IRC, Bot, RAT** 등
- 프로토콜별 디코딩
  - 각 프로토콜에 맞는 구조 분석 후 **ASCII** 문자열로 추출
  - 커스텀 악성 패턴**(Base64, C2** 등)도 식별 가능
- 엔트로피 분석
  - **> 6.7**일 경우 암호화 또는 난독화 가능성
  - **<= 6.7**이면 구조적 또는 일반 텍스트 가능성

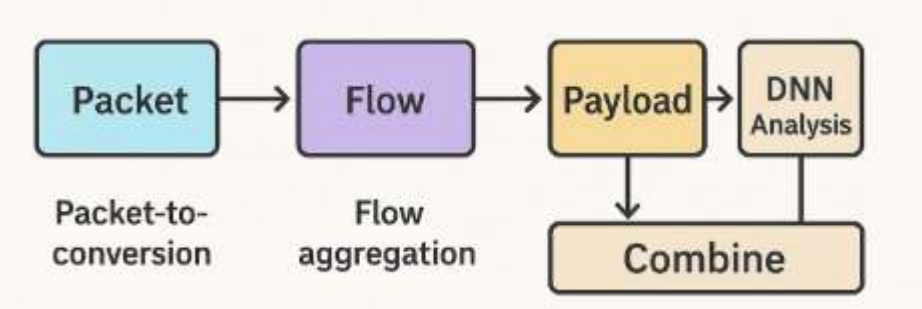
5. Payload 디코딩 실습

✓주요 구성 요소

- detect_protocol(payload) : 포트 없이도 프로토콜 감지
- calculate_entropy(payload) : 랜덤성 기반 암호화 여부 판단
- bts(payload) : ASCII 변환
- decode_XX(payload) : 프로토콜별 디코더 호출최종 결과: 사람이 읽기 쉬운 분석 리포트 생성

✓동작 방식

- known 포트가 있으면 decode_XX 호출
- Unknown 포트일 경우 페이로드 안에서 시그니처 기반 프로토콜 추론
  - DNS, TLS, SMTP, FTP, SMB, mDNS, MQTT, BitTorrent 등 50개 이상 패턴 보유Entropy 기반 암호화 여부도 판단 가능



5. Payload 디코딩 실습

프로토콜	악용사례	프로토콜 구조(요약)
HTTP/HTTPS	웹 기반 명령 전달 및 악성 파일 유포	GET/POST 헤더, 본문
DNS	C2 서버 주소 조회, 데이터 은닉	Transaction ID, Flags, QDCount, ANCount
FTP	백도어 설치, 정보 유출	파일 목록 출력(권한, 이름 등)
SMTP	스팸메일, 피싱 공격	From, To, Subject, SMTP 코드
SMTP Response	메일 전송 성공/실패 코드 분석	220, 250, 550 등 상태코드
Email Body	피싱 메시지 본문	텍스트 기반 이메일 본문
DHCP	Fake DHCP 서버 통한 MITM	XID, Flags, HType
NTP	NTP 반사 DDoS	LI_VN_MODE, Stratum, Poll, Precision
LLMNR	LLMNR Spoofing을 통한 자격증명 탈취	Transaction ID, Flags, Questions
SMB	EternalBlue, 파일 공유 통한 악성코드 확산	헤더: ySMB
SNMP	장비 정보 수집, 디도스	OID 및 관리 정보
NBNS	NBNS Spoofing 통한 인증정보 탈취	Transaction ID, Name, Query Type
TLS Handshake	초기 핸드셰이크 분석으로 인증서 정보 탐색	ContentType, Version, HandshakeType
mDNS	서비스 검색 정보 탈취	Queried Domain, Type, Class
DNS-SRV	서비스 위치정보 조회	Queried Service, Port, Target
LDAP	AD 탐색 또는 인증 우회	OID, DN, CN
DCE/RPC	Windows 서비스 바인딩, 악용 시 원격 코드 실행	PacketType, UUID, CallID
Custom Encrypted	Base64, Zlib 압축된 악성 페이로드	암호화 여부 분석
Kerberos	티켓 위조 공격, 인증 우회	Timestamp, Domain
TLS/PKI	인증서 검증 우회, OCSP/CRL 조작	OCSP, CRL URL
TLS Application	암호화된 명령 및 악성파일 전송	TLS 버전, RecordType
TLS Session	SNI 확인 통한 도메인 식별	Handshake + Application 포함
XMRig/Monero	불법 채굴 봇 탐지	jsonrpc, algo, job_id
Ethereum DevP2P	노드 정보 수집 및 전파	hello, discovery 메시지
SIP	VoIP 서버 해킹, 인증 우회	INVITE, BYE, ACK
IRC	봇넷 C2 통신	JOIN, PRIVMSG, NICK
BitTorrent	DHT 기반 데이터 은닉	Handshake, DHT Query
WHOIS	도메인 등록 정보 수집	Registrar, Domain info
Custom C2	특정 악성 봇 통신 탐지	gPa, Kn 등 사용자 정의 패턴
IMAP Bot	IMAP 기반 C2 통신 탐지	eastex.net, IMAP4S
RevengeRAT	윈도우 RAT 백도어	UTF-16 Base64 기반 명령
Infostealer	시스템 정보 탈취	OS, IP, BotID 등 필드
MQTT	IoT 공격, 인증 우회	Connect Flags, KeepAlive
VNC	원격 제어 도구 통한 해킹	Security Types, RFB Version
WS-Discovery	기기 스캔 및 서비스 검색 악용	SOAP Action, Message ID
IKE	IPSec 키 교환, VPN 해킹 시도	SPI, Exchange Type
CoAP	IoT 환경 공격에 사용	Message ID, URI-Path
Rsync	원격 동기화 기능 악용	Label_IP_Port 구조
Custom DVKT	장비 전용 C2 명령 프로토콜	Signature, MAC
Domain Tunnel	도메인 이름 기반 C2 통신	도메인 문자열 포함 여부
MGCP	VoIP 게이트웨이 해킹	Transaction ID, 명령어
TDS	MSSQL 비연결형 인증 시도	CONNECTIONLESS_TDS 마커
Printer(BJNP)	네트워크 프린터 취약점 이용	BJNP, CANON, MFNP

✓**현재 Decoding 가능한 프로토콜**

5. Payload 디코딩 실습

Offset	필드 이름	크기	설명
0-1	Transaction ID	2	클라이언트가 보낸 요청과 응답을 매칭하기 위한 ID
2-3	Flags	2	요청/응답 여부, 재귀 가능 여부, 오류 코드 등 포함
4-5	QDCOUNT	2	질의(Query)섹션에 포함된 질의 개수
6-7	ANCOUNT	2	응답(Answer) 섹션에 포함된 리소스 레코드 개수

✓ **DNS 패킷 및 디코딩**

- 플레그
  - **QR (0: Query, 1: Response)**
  - **Opcode (표준 질의, 역질의 등)**
  - **AA (권한 있는 응답 여부)**
  - **TC (트렁크 여부)**
  - **RD (재귀 요청 허용 여부)**
  - **RA (재귀 가능 여부)**
  - **RCODE (응답 코드: 0 성공, 3 도메인 없음 등)**

✓ **실행 python preprocess_pcap.py --input_file DNS_practice.pcap**

5. Payload 디코딩 실습

✓FTP 디코딩

- FTP의 LIST 응답은 디렉토리 구조 등을 출력
- 파일 유형, 크기, 날짜, 이름을 추출 및 분석
- 읽을 수 없는 응답이나 비표준 포맷도 "Unparsed"로 표시해 유연하게 처리

✓실행 python preprocess_pcap.py --input_file FTP_practice.pcap

```
Detected Protocol: FTP LIST Response
Directory  |   4096 bytes | Apr 05 12:34 | folder
File       |   1024 bytes | Apr 04 09:21 | file.txt
Link       |     13 bytes | Apr 03 10:12 | symlink -> /some/target
End of FTP LIST Response
Readable ASCII Data: drwxr-xr-x 2 user group 4096 Apr 05 12:34 folder ...
```

5. Payload 디코딩 실습

✓SMTP 디코딩 및 SMTP 응답 패킷 디코딩

- SMTP 이메일 메시지를 분석하여 송신자/수신자, 제목, 피싱 링크, 응답 코드 등을 추출
- 이메일 본문 내용 디코딩

✓디코딩 방식 및 순서

- Response code 추출: 3자리 숫자(예: 220, 250, 550)로 시작하는 SMTP 응답 메시지들을 추출
- From, To, Subject를 이메일 헤더에서 정규표현식으로 추출
- http:// 또는 https://로 시작하는 URL 매핑
- 본문 디코딩

✓실행 python preprocess_pcap.py --input_file SMTP_practice.pcap

5. Payload 디코딩 실습

✓**DHCP 디코딩**

- Op 코드, htype 네트워크 타입, hlen 하드웨어 주소 길이, xid 트랜잭션 식별자, secs 요청 이후 경과 시간, flags 브로드캐스트 여부 등 options IP 요청/할당 정보 포함

✓**개요공격자가 가짜 DHCP 서버를 네트워크에 삽입하여 클라이언트에게 악의적인 설정(IP, Gateway, DNS 등) 가능**

✓**공격 효과모든 트래픽을 공격자의 게이트웨이로 전달 → MITM(중간자 공격) 가능**

✓**악성 DNS로 유도하여 피싱 사이트 접속 유도**

✓**실행 python preprocess_pcap.py --input_file dhcp_practice.pcap**

5. Payload 디코딩 실습

✓NTP 디코딩

- Li_vn_mode: Leap Indicator, Version, Mode 정보를 담은 바이트 필드
- Stratum: 시간 계층(0=unsynchronized, 1=primary ref like GPS)
- Poll: 시간 동기화 주기
- Precision: 클럭 정밀도 지표

✓NTP를 활용한 대규모 DDoS 공격이 대표

- 공격자는 spoofed IP(타겟 IP)를 출발지로 설정한 NTP 요청을 다수의 NTP 서버에 보냄
- 응답은 수십 배 증폭된 채 타겟에게 도착 (bandwidth 폭탄)

✓실행 python preprocess_pcap.py --input_file NTP_practice.pcap

5. Payload 디코딩 실습

✓LLMNR 프로토콜

- LLMNR은 로컬 네트워크 상에서 호스트 이름을 IP 주소로 변환하기 위한 프로토콜
- DNS가 설정되지 않은 환경에서도 이름 해석이 가능하도록 설계

✓악용 사례

- 악의적인 사용자가 네트워크에서 스푸핑 공격(예: 인증정보 탈취)
- 네트워크에 연결된 사용자의 이름 질의 요청을 가로채고, 자신의 IP를 응답으로 보내 인증 정보를 탈취하는 LLMNR/NBT-NS Poisoning 공격

✓구성

- transaction_id: LLMNR 질의 요청의 고유 식별자. 요청-응답 쌍을 구분
- flags: 요청/응답 여부, 오류 코드 등의 플래그 정보
- qdcount: 질의 수

✓실행 python preprocess_pcap.py --input_file llmnr_practice.pcap

5. Payload 디코딩 실습

✓SMB(Server Message Block) 프로토콜

- SMB는 네트워크를 통해 파일, 프린터, 직렬 포트 등 다양한 리소스를 공유할 수 있게 해주는 Windows 기반 프로토콜

✓악용 사례

- EternalBlue (CVE-2017-0144): SMBv1 취약점을 이용한 원격 코드 실행 공격. WannaCry 랜섬웨어의 주요 전파 수단.
- SMB 리레이 공격 (NTLM Relay): 공격자가 SMB 인증 요청을 가로채서 중간자 공격을 수행할 수 있음.
- 패스워드 해시 탈취: 인증 시 전달되는 해시 값을 탈취하여 재사용 공격 가능.

✓구성

- SMB hex Data

✓실행 python preprocess_pcap.py --input_file SMB_practice.pcap

5. Payload 디코딩 실습

✓SNMP (Simple Network Management Protocol) 프로토콜

- SNMP는 네트워크 장비(라우터, 스위치, 프린터 등)의 상태 모니터링 및 제어를 위해 사용되는 프로토콜

✓악용 사례

- 정보 수집 (Reconnaissance)
  - SNMP 커뮤니티 스트링이 public일 경우 인증 없이 장비 정보를 획득 가능→ 공격자는 장비 모델, 인터페이스, 라우팅 테이블 등의 정보를 활용해 후속 공격 수행
- DDoS 증폭 공격 (SNMP Reflection Attack)
  - 응답 크기가 요청보다 훨씬 크기 때문에 공격자가 스푸핑을 통해 증폭 가능→ 수십 Gbps 규모로도 확장 가능

✓구성

- SNMP hex Data

✓실행 python preprocess_pcap.py --input_file SNMP_practice.pcap

5. Payload 디코딩 실습

✓NetBIOS Name Service (NBNS)

- NBNS는 주로 윈도우 환경에서 사용하는 프로토콜로, IP 주소를 알기 위해 컴퓨터 이름(예: DESKTOP-1234)을 질의하는 데 사용
- DNS처럼 동작하지만, UDP 포트 137을 사용하고 로컬 네트워크 내에서 주로 작동
- NBNS는 SMB 트래픽 이전에 사용되며, 이름 → IP 주소 매핑을 요청하거나 응답하는 패킷을 전송함.

✓악용 사례

- NBNS Spoofing: 악성 호스트가 허위 응답을 먼저 보내 사용자의 요청을 가로채기 (Man-in-the-Middle)
- Responder 도구: 대표적인 툴로, 네트워크 내에서 NBNS 요청을 가로채고 SMB 인증을 탈취
- SMB Relay 공격 연계: 사용자의 인증 해시(NTLM)를 탈취하여 SMB 서버로 중계해 권한 상승 가능
- 정보 수집: 패킷만 봐도 내부 장치 이름, 도메인 구조 등을 노출시킬 수 있음구성

✓실행 python preprocess_pcap.py --input_file nbns_practice.pcap

5. Payload 디코딩 실습

✓TLS(Transport Layer Security)

- HTTPS, IMAPS, SMTPS 등에 사용되는 암호화된 통신 프로토콜
- 서버와 클라이언트가 핸드셰이크 → 인증서 → 키 교환 → 암호화된 데이터 전송 순으로 보안 세션을 설정함

✓악용 사례

- TLS 트래픽 내부에 C2 명령이나 악성 페이로드를 암호화하여 은폐할 수 있음
- TLS 핸드셰이크를 흉내 낸 악성 트래픽을 통해 우회
- SNI 필드를 통해 C2 도메인을 은폐 가능

✓패킷 구성

- Content type: TLS 종류
- Version(major, minor): TLS 버전

✓실행 python preprocess_pcap.py --input_file TLS_practice.pcap

5. Payload 디코딩 실습

✓mDNS(Multicast DNS)

- 로컬 네트워크 상에서 DNS 서버 없이 기기 이름을 IP로 매핑하기 위한 프로토콜
- Bonjour, Chromecast, 스마트 TV, 프린터 등이 네트워크 상에서 자동으로 검색되도록 할 때 사용

✓악용 사례

- Lateral Movement: 공격자가 로컬 네트워크 내 장치 이름을 탐색하여 측면 이동
- 정보 수집 (Reconnaissance): mDNS 응답을 통해 기기 이름, OS 정보 노출 가능
- mDNS Spoofing: 가짜 응답을 보내서 다른 장치가 공격자 IP로 트래픽을 보내게 유도

✓실행 python preprocess_pcap.py --input_file mDNS_practice.pcap

5. Payload 디코딩 실습

✓DNS-SRV(Service Resource Record)

- 특정 서비스를 도메인 기반으로 탐색할 수 있게 도와주는 레코드 타입
- 예: _ldap._tcp.example.com → LDAP 서비스를 제공하는 서버를 반환함

✓악용 사례

- DNS-SRV Spoofing: 공격자가 가짜 _ldap._tcp 응답을 보내 자신의 서버로 인증 트래픽 유도
- MITM: LDAP, Kerberos 요청을 공격자가 중간에서 가로채고 조작
- Reconnaissance: Active Directory 등 내부 네트워크 구조 탐색에 악용

✓주요 구성

- Queried Service, Answer Name: 통신 서비스 이름
- 질의 정보

✓실행 python preprocess_pcap.py --input_file DNSSRV_practice.pcap

5. Payload 디코딩 실습

✓LDAP (Lightweight Directory Access Protocol)

- 디렉토리 서비스를 조회하거나 수정하기 위한 응용 계층 프로토콜
- 조직 내 사용자, 그룹, 권한 등의 정보를 계층적 구조 (트리) 로 관리하는 데 사용되며
- Active Directory, OpenLDAP, Novell eDirectory 등에 사용

✓악용 사례

- LDAP Injection: 사용자 입력을 LDAP 쿼리에 삽입해 인증 우회 또는 정보 노출 유도
- 정보 수집: 내부 AD 구조나 사용자 정보를 수집하는 데 사용 (Reconnaissance 단계)
- 권한 상승: 잘못 구성된 LDAP 정책을 악용해 관리자 권한 획득

✓주요 구성

- OID (Object Identifier): LDAP에서 객체 유형, 속성 등을 고유하게 식별
- DN (Distinguished Name): 사용자/그룹의 전체 경로를 나타내는 식별자 (ex: DC=example,DC=com)
- CN (Common Name): 객체의 실제 이름 (ex: CN=John Doe)

✓실행 python preprocess_pcap.py --input_file LDAP_practice.pcap

5. Payload 디코딩 실습

✓DCE/RPC (Distributed Computing Environment / Remote Procedure Call)

- DCE/RPC는 원격 시스템에서 함수를 호출할 수 있게 해주는 마이크로소프트의 프로토콜
- 보통 Windows 시스템의 포트 135를 통해 통신하며, Active Directory, 파일 공유, 프린터, WMI 등에서 사용
- SMB, HTTP, NetBIOS 등의 다양한 전송 계층 위에서 동작

✓악용 사례

- EternalBlue (MS17-010): SMB + RPC 조합으로 수행
- Remote Code Execution (RCE): RPC 인터페이스를 통해 악성 함수를 원격 호출
- 권한 상승 및 인증 우회: 취약한 바인딩 및 인증 구조를 악용
- Reconnaissance 공격: Active Directory 관련 RPC 호출로 사용자/호스트 정보 수집

5. Payload 디코딩 실습

✓**DCE/RPC (Distributed Computing Environment / Remote Procedure Call)(2)**

✓**주요 구성**

- **Version / Minor:** DCE/RPC 프로토콜 버전
- **Packet Type:** 요청/응답/바인드/오류 등 패킷 유형
- **Flags:** 전송 관련 옵션
- **Data Representation:** 바이트 순서, 인코딩 정보
- **Fragment Length:** 전체 패킷 길이
- **Auth Length:** 인증 관련 데이터 길이
- **Call ID:** RPC 호출 구분용 ID
- **Service UUID:** 호출 대상 인터페이스의 고유 ID

✓**실행 python preprocess_pcap.py --input_file DCERPC_practice.pcap**

5. Payload 디코딩 실습

✓Kerberos

- 네트워크 상에서 안전한 인증을 위해 설계된 비밀 키 기반 인증 프로토콜
- 주로 **Windows** 환경에서 **Active Directory(AD)** 인증 시스템의 기반이 됨
- **ASN.1/BER** 인코딩 형식을 사용하며 **Kerberos**와 함께 쓰이는 경우가 많음

✓악용 사례

- **Golden Ticket Attack:** 공격자가 도메인 키를 탈취한 후 유효한 **Kerberos** 티켓을 위조해 인증 우회
- **Silver Ticket Attack:** 특정 서비스에만 유효한 티켓을 위조해 권한 상승
- **LDAP Reconnaissance:** 공격자가 **LDAP**를 통해 **AD** 구조 및 사용자 정보를 수집
- **Pass-the-Ticket:** 유출된 티켓을 재사용하여 인증 우회.

✓실행 python preprocess_pcap.py --input_file kerberos_practice.pcap

5. Payload 디코딩 실습

✓XMRig & Monero 채굴 통신

- 암호화폐를 채굴하기 위한 오픈소스
- 일반적으로 마이너는 **Stratum** 프로토콜 또는 **JSON-RPC over TCP**를 사용하여 채굴풀과 통신

✓악용 사례

- **XMRig**을 설치하여 리소스를 탈취주요
- 공격 형태
  - 웹 기반 **Cryptojacking: JS**로 채굴 실행
  - 악성 파일/**XMRig Dropper:** 시스템에 설치되어 지속 실행
  - 봇넷 채굴: 수천 대의 감염 장비에서 병렬 채굴 수행

5. Payload 디코딩 실습

✓XMRig & Monero 채굴 통신(2)

✓구성

- Jsonrpc: JSON-RPC 버전 ("2.0")
- Method: login, submit, job, result 등
- Params: algo, agent, login, blob, target 등
- job_id: 채굴 작업 ID
- Blob: 해시할 데이터 블록 (Hex)
- Target: 난이도 목표값
- Agent: XMRig 버전 및 마이너 정보

✓실행 python preprocess_pcap.py --input_file XMrig_practice.pcap

5. Payload 디코딩 실습

✓Ethereum DevP2P 채굴 통신

- 이더리움 클라이언트들이 서로 통신하기 위해 사용하는 핵심 P2P 네트워크 프로토콜
- 노드 검색(Discovery), 상태 동기화, 블록/트랜잭션 전파
- DevP2P는 보통 UDP 포트 30301(Discovery)과 TCP 포트 30303(데이터 동기화)에서 사용

✓악용 사례

- 악성 DevP2P 트래픽 유입
- 암호화폐 탈취 도구와의 연계
- 정보 수집

✓실행 python preprocess_pcap.py --input_file p2p_practice.pcap

5. Payload 디코딩 실습

✓SIP (Session Initiation Protocol)

- IP 네트워크 상에서 음성 통화(VoIP), 영상 통화, 메시징, 존재 확인 등 멀티미디어 세션을 생성·수정·종료하는 데 사용되는 신호 제어 프로토콜
- 일반적으로 UDP 또는 TCP 포트 5060에서 동작하며, HTTP와 유사한 텍스트 기반 구조를 가집니다.

✓악용 사례

- VoIP 스팸 (SPIT): 자동화된 SIP 메시지를 대량 발송하여 VoIP 시스템에 전화를 걸거나 메시지를 보내는 공격
- SIP Invite Flood: 수많은 INVITE 요청을 전송하여 SIP 서버를 과부하시킴 (DoS)
- Session Hijacking: 중간에서 Call-ID, CSeq 등을 탈취해 세션을 탈취하는 공격
- SIP Credential Brute Force: 인증이 필요한 REGISTER 메시지를 반복적으로 시도하여 계정 탈취 시도.

✓실행 python preprocess_pcap.py --input_file SIP_practice.pcap

5. Payload 디코딩 실습

✓ **IRC (Internet Relay Chat)**

- 인터넷을 통한 텍스트 기반의 채팅 프로토콜,
- 일반적으로 TCP 포트 6667을 사용하며, Nick / JOIN / PRIVMSG / QUIT 등의 명령어로 대화를 구성합니다.
- 사용자 간 실시간 메시지 전송, 채널 기반 대화, 공개 및 비공개 채팅 기능 제공

✓ **악용 사례**

- IRC는 구조가 단순하고 설정이 자유로워, 많은 악성코드가 C2 통신으로 이용
- 감염된 PC는 IRC 채널에 자동으로 접속하여 봇마스터의 명령을 수신하고 실행

✓ **프로토콜 구성**

- NICK: 사용자 닉네임 설정
- JOIN: 채널 접속
- PRIVMSG: 특정 사용자 또는 채널로 메시지 전송
- QUIT: 채팅 종료 및 퇴장 메시지

✓ **실행 python preprocess_pcap.py --input_file IRC_practice.pcap**

5. Payload 디코딩 실습

✓MQTT (Message Queuing Telemetry Transport)

- 경량 메시징 프로토콜로, IoT(사물인터넷) 장치 간 통신에 사용
- Publisher-Subscriber 구조로 작동하며, 메시지는 Broker를 통해 전송

✓악용 사례

- IoT 봇넷 제어: Mirai 같은 봇넷이 MQTT를 통해 원격 명령을 수신하거나 데이터를 업로드
- 정보 유출: 센서 데이터, 장치 인증 정보 등이 평문으로 MQTT를 통해 송수신될 수 있어 유출 위험 존재.

✓프로토콜 구성

- Protocol Level: MQTT 버전 (예: 4 for 3.1.1)
- Connect Flags: 사용자 인증, 클린 세션 등

✓실행 python preprocess_pcap.py --input_file mqtt_practice.pcap

5. Payload 디코딩 실습

✓VNC (Virtual Network Computing)

- 원격 데스크톱 제어 프로토콜
- RFB(Remote Frame Buffer) 프로토콜을 사용하여 GUI 화면을 원격으로 제어
- 일반적으로 5900번 포트를 사용하며, 클라이언트와 서버 간에 키보드/마우스/화면을 공유

✓악용 사례

- 원격 제어 백도어: 공격자가 피해자의 시스템에 VNC 서버를 심어 원격 조종
- Brute-force 공격을 통해 VNC 인증 우회 시도
- 암호 미설정 또는 None 인증을 사용하는 경우, 무방비 상태의 원격 시스템 노출 가능
- 내부망 침해: 악성코드가 VNC를 통해 내부 시스템 탐색 및 확장 공격 가능.

✓실행 python preprocess_pcap.py --input_file vnc_practice.pcap

자연어 처리(NLP) 모델 개요

1. 자연어 처리 모델 개요와 악성 탐지 응용

✓목표

- 자연어 처리(NLP)의 기본 개념을 이해하고 보안 데이터에서의 활용 가능성 확인
- 주요 NLP 모델(BERT)의 구조를 이해하고 학습 방식을 파악
- 전이 학습(Transfer Learning)의 개념과 보안 응용에 대한 이해 확대

✓진행

- NLP 기술의 활용 분야 개요
- 악성 탐지에서의 텍스트 기반 패턴 이해
- BERT 구조 및 특징 설명
- 전이 학습을 통한 모델 커스터마이징 개념 소개
- 보안 로그/네트워크 패킷 등에서의 NLP 활용 가능성 소개

자연어 처리(NLP) 모델 개요

1. 자연어 처리 모델 개요와 악성 탐지 응용

✓자연어 처리(NLP)란?

- 자연어 처리(NLP)는 인간의 언어를 이해하고 처리하기 위한 인공지능(AI) 기술의 한 분야
- 텍스트, 음성 등 비정형 데이터를 구조화된 정보로 변환
- 문장 분류, 감정 분석, 질의응답, 요약, 번역 등 다양한 작업 수행 가능

✓보안 분야에서의 활용 사례

- 악성 이메일 탐지 (피싱 메일, 스팸 필터링)
- 악성 URL 및 도메인 탐지
- 로그 분석을 통한 이상 행위 탐지
- 네트워크 페이로드 내 명령/키워드 추출

✓기존 시그니처 기반 탐지보다 일반화된 패턴 학습 가능성 존재

자연어 처리(NLP) 모델 개요

1. 자연어 처리 모델 개요와 악성 탐지 응용

✓자연어 처리(NLP)란?

- 자연어 처리(NLP)는 인간의 언어를 이해하고 처리하기 위한 인공지능(AI) 기술의 한 분야
- 텍스트, 음성 등 비정형 데이터를 구조화된 정보로 변환
- 문장 분류, 감정 분석, 질의응답, 요약, 번역 등 다양한 작업 수행 가능

✓보안 분야에서의 활용 사례

- 악성 이메일 탐지 (피싱 메일, 스팸 필터링)
- 악성 URL 및 도메인 탐지
- 로그 분석을 통한 이상 행위 탐지
- 네트워크 페이로드 내 명령/키워드 추출

✓기존 시그니처 기반 탐지보다 일반화된 패턴 학습 가능성 존재

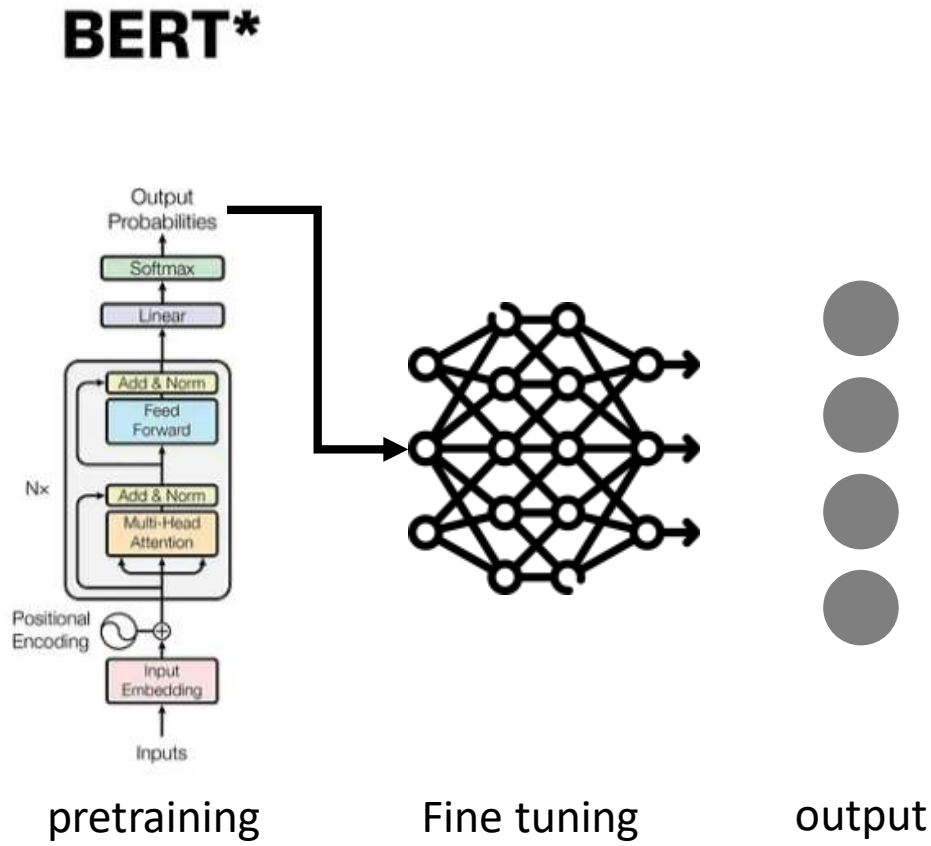
자연어 처리(NLP) 모델 개요

2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓BERT 구조 개요

- BERT는 Transformer 기반의 사전 학습 언어 모델로서, 양방향 인코더 구조
- 입력 문장을 토큰화한 후, 각 토큰은 다음 세 가지 임베딩의 합으로 표현
  - Token Embedding
  - Segment Embedding
  - Position Embedding
- 모델 흐름
  - [CLS] + 문장 A + [SEP] + 문장 B + [SEP] 형태로 입력 구성
  - 각 토큰은 Transformer 인코더 층을 거쳐 문맥을 반영한 임베딩으로 변환
  - 최종 [CLS] 토큰 임베딩은 분류 등에 사용됨

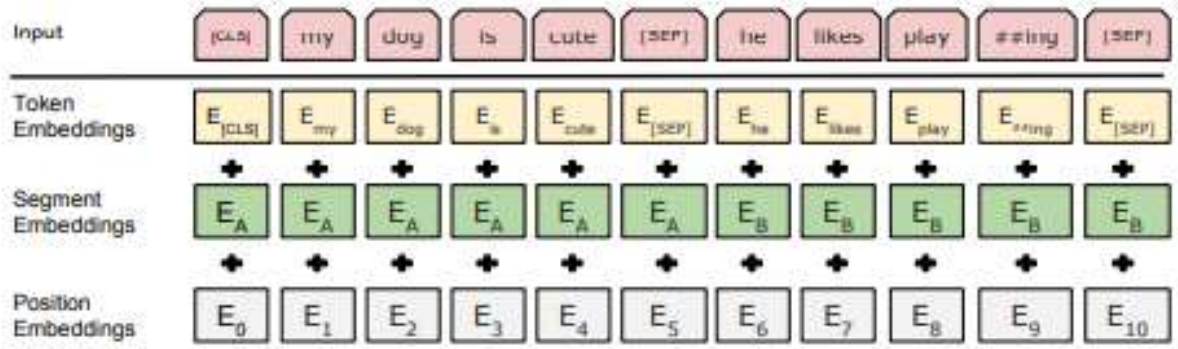
✓BERT는 범용 언어 모델로 네트워크 보안을 위해 fine-tuning을 추가하여 특화된 탐지 모델 생성



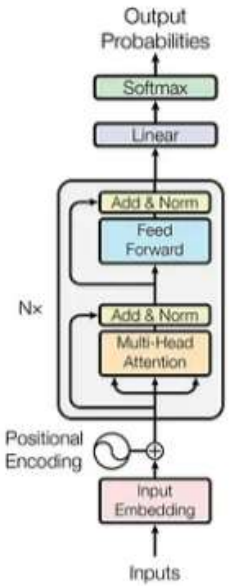
2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓Pre-training(사전 학습)

- **Input Embedding:**단어 + 위치 정보를 포함한 벡터로 변환→ **[CLS], [SEP]** 같은 특수 토큰 포함
  - **WordPiece Embedding:** 단어를 서브워드 단위로 분할하여 표현 **(ex. un + ##able)**
  - **Position Embedding:** 순서 정보를 인코딩, **Transformer**는 순서를 인식하지 못하기 때문에 필수
  - **Segment Embedding:** (선택적)문장 **A**, 문장 **B** 구분용 정보문장쌍 입력 시 사용 **(NSP** 태스크 등)
- **Multi-Head Attention:**문장의 각 단어가 다른 모든 단어와의 관계를 동시에 파악→ 문맥 기반 이해의 핵심
- **Feed Forward + Add & Norm:**각 단어의 표현을 정제 및 강화



BERT*



pretraining

2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓**Tokenizer**

- 자연어 → 모델이 이해할 수 있는 형태 (숫자 시퀀스) 로 바꾸는 도구
- BERT는 문장을 토큰 단위로 분리 후 ID로 인코딩함

✓**WordPiece Tokenizer**

- 사용자주 나오는 단어는 그대로, 생소한 단어는 부분 단위로 쪼갬
- 예: ['un', '##believ', '##able'] → unbelievable

✓**실습: python AI/tokenizer_practice.py**

2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓ Embedding의 필요성

- 원래 단어는 문자열(string)로 되어 있으므로, 모델이 수치 계산을 할 수 있도록 숫자 벡터로 변환해야 함
- 단어의 의미적 유사성을 벡터 공간에 반영할 수 있음 (예: king - man + woman ≈ queen)
- BERT에서는 WordPiece Embedding + Position Embedding을 더해서 문맥과 위치 모두 고려

✓ Embedding에 있어 왜 단어를 쪼개는 이유는?

- 희귀어/오타/신조어 처리에 유연한 처리르 위함

✓ Attention 이란?

- 하나의 단어가 문장 내 다른 단어들과 얼마나 관련이 있는지를 가중치로 표현
- 예: 문장 "The cat sat on the mat."에서 "sat"는 "cat"과 가장 관련 깊음 → attention score ↑
- 입력 전체를 참고하므로 긴 문장에서의 문맥 유지가 뛰어남

✓ Self-Attention이란?

- 입력 시퀀스 내부의 각 단어가 자기 자신을 포함한 모든 단어를 동시에 바라보는 구조
- 각 단어가 문맥을 스스로 이해하는 메커니즘
- Transformer에서 인코더와 디코더 모두에 사용

2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓Multi-Head Attention과 Self-Attention의 차이는?

항목	Self-Attention	Multi-Head Attention
설명	단일한 방식으로 모든 단어 간 관계를 분석	여러 개의 self-attention을 병렬로 수행
목적	문맥 고려	서로 다른 관점/특징을 추출해 조합
결과	하나의 attention value	다양한 관점의 분석된 attention 값들의 함

- **Self-Attention:** “이 단어는 다른 단어들과 얼마나 관련 있어?”
- **Multi-Head:** “이 단어를 여러 방식으로 (위치 기준, 의미 기준 등) 동시에 바라보자”

✓Transformer란?

- **Google**이 **2017**년 발표한 논문 《**Attention is All You Need**》에서 제안
- **RNN, LSTM** 없이 순차 정보 처리 가능한 신경망 구조
- 문장의 순서를 유지하면서 **병렬 연산**이 가능하여 긴 문자에서 전체 문맥을 한 번에 반영 가능**(RNN** 보다 효율적**)**

2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓Tokenizer는 사전에 정의된 단어만 사용 가능한가?

- No
- BERT의 Tokenizer는 WordPiece 방식으로 훈련된 vocabulary(단어 사전)를 사용
- 사전에 없는 단어가 등장하면 부분 단위로 쪼개서 처리

✓subword 방식의 장점?

- 기존 단어 단위(token) 방식은 사전에 없는 단어(OOV: out-of-vocabulary)를 처리하지 못함
- subword 방식은 모르는 단어를 부분 단위로 쪼개서 처리할 수 있어 희귀 단어 처리 가능

✓한글 처리 방식은?

- BERT에는 한국어 전용 사전(vocab)을 사용한 KoBERT, KoElectra, klue/bert-base 등이 있음
- Tokenizer 한글 형태소, 자모 분리 등을 고려한 방식으로 구성

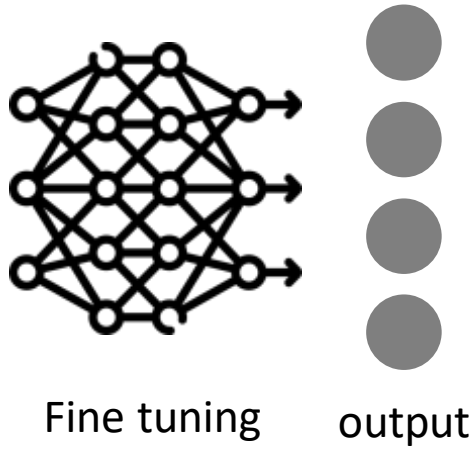
2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓Fine Tuning(미세 조정)

- 사전학습된 BERT는 **일반 언어 패턴**을 이해하고 있음
- 실제 업무에 맞는 특정 태스크(예: 악성 탐지, 분류, QA 등)에 맞게 추가 학습하는 과정

✓특징

- 기존 BERT 파라미터는 대부분 유지하고, 출력층만 새로 학습
- 적은 데이터와 시간으로도 높은 성능 도달
- NLP 모델을 보안 분야에 빠르게 적용 가능



2. 주요 NLP 모델 (BERT)의 기본 구조 및 학습 방식

✓BERT의 사전 학습 방식(실제 악성 트래픽에 어떻게 적용)

- Masked Language Modeling (MLM)
  - 설명: 문장 내 특정 토큰을 [MASK] 처리하고 이를 예측보안 로그 적용
  - 예: 악성 트래픽 내에서 의심 단어 예측

```
Input: "USER anonymous\nPASS [MASK]\nCWD /tmp\n"
BERT Prediction: [MASK] → "1234", "ftp", "guest"
```

- Next Sentence Prediction (NSP)
  - 기존 BERT 파라미터는 대부분 유지하고, 출력층만 새로 학습
  - **설명:** 두 문장이 실제로 연속되는지 학습
  - **보안 예시 (decode_smtp + decode_http):**
    - 문장 A: "SMTP From: spam@malicious.com“
    - 문장 B: "HTTP GET /malware.exe“
    - NSP 결과: 이어지는 문장 → 악성 이메일 → 외부 파일 다운로드

3. BERT를 활용한 악성 페이로드 탐지 실습

✓실행: python train.py --train_model_type NLP --MAXLEN 100

✓MAXLEN은 -1(전체)를 사용하면 성능이 좋아지나 모든 텍스트를 읽기 때문에 오래걸림

✓Arguments 설명

Argument	Default	Type	Description
--gpu	0	Str	Gpu 번호 할당
--data_base_dir	../model_data/	Str	데이터가 저장된 기본 디렉토리 경로
--tabular_data_name	Tabular	Str	정형(tabular) 데이터의 이름
--bert_file	nlp_data.txt	Str	Bert 입력용 텍스트 데이터 파일명
--train_model_type	['NLP']	List	학습할 모델 선택, 다중 모델 가능(xgboost, bert, NLP 등)
--model_output_dir	../out_model	Str	학습된 모델이 저장될 디렉토리
--bert_check_point	None	Str	추가 학습을 위해 미리 학습한 모델의 체크포인트 경로
--model_name	quadminers_edu	Str	출력 모델 파일 이름
--random_seed	42	Int	랜덤 시드 값(재현 가능성을 위함)
--MAXLEN	-1	Int	BERT 입력 토큰 최대 길이, -1이면 모든 Text

모델링 실습: BERT 기반 악성 탐지 모델 구축

3. BERT를 활용한 악성 페이로드 탐지 실습

✓실행: python evalution.py --eval_model_type NLP --MAXLEN 100

✓MAXLEN은 -1(전체)를 사용하면 성능이 좋아지나 모든 텍스트를 읽기 때문에 오래걸림

모델 성능 개선 전략

1. 데이터 불균형 처리 (Data Imbalance Handling)

✓문제점

- 악성 트래픽이 전체에서 소수 → 모델이 정상 트래픽에 과도하게 편향될 수 있음
- 악성(positive) vs 정상(negative) 데이터의 수가 불균형한 경우
  - 예: 정상 95%, 악성 5%
  - 이럴 경우 모델은 정상 데이터만 예측해도 정확도는 높게 나오지만 실제로는 악성 탐지를 거의 못함.

✓해결 방법

방법	설명	라이브러리
Oversampling(과샘플링)	소수 클래스의 데이터를 복제하거나 합성	Imblearn.over_sampling.SMOTE
Undersampling(과소샘플링)	다수 클래스 데이터를 줄임	Imblearn.under_sampling.RandomUnserSampler
Class Weight 조정	학습 시 손실함수에 클래스별 가중치 부여	Sklearn, torch 등
Ensemble + Sampling	균형 조정된 데이터 기반으로 앙상블	BalancedBaggingClassifier

모델 성능 개선 전략

2. BERT 모델의 성능 향상

✓**Fine-Tuning 전략**

✓**BERT Fine-Tuning**

- **BERT는 사전학습된 언어모델로, downstream task(분류, QA 등)에 맞춰 파인튜닝 함**
- **보통 CLS 토큰의 출력을 linear layer에 통과시켜 사용**

기법	설명
Full Fine-Tuning	BERT 전체 레이어를 학습 (많은 메모리, 높은 성능 가능)
Feature Extraction	BERT는 freeze, 출력만 사용해 다른 분류기 학습
Partial Fine-Tuning	일부 레이어만 학습 (보통 마지막 N개 layer만)
Adapter Tuning	사이에 작은 네트워크 삽입 → 파라미터 효율성 높음

```
for name, param in bert_model.named_parameters():
    if 'encoder.layer.10' in name or 'encoder.layer.11' in name:
        param.requires_grad = True  # 마지막 두 개만 학습
    else:
        param.requires_grad = False
```

모델 성능 개선 전략

2. BERT 모델의 성능 향상

✓**Fine-Tuning 전략**

✓**MLP / Linear 모델 Fine-Tuning**

- **BERT의 출력을 받아 간단한 Fully Connected Layer를 학습에 튜닝 기법 적용**
- **이 경우 weight 초기화, layer normalization, dropout 등 기법을 통해 정규화 하며 튜닝**

기법	설명
Learning Rate Tuning	BERT와 다른 learning rate를 부여 (e.g. head: 1e-3, BERT: 1e-5)
Weight Decay	과적합 방지
Dropout / BatchNorm	안정적인 학습을 위한 정규화
Early Stopping	검증 손실 기준으로 학습 조기 종료

```
optimizer = AdamW([
    {'params': bert_model.parameters(), 'lr': 5e-5},
    {'params': classifier_head.parameters(), 'lr': 1e-3}
], weight_decay=0.01)
```

모델 성능 개선 전략

2. BERT 모델의 성능 향상

✓**Fine-Tuning 전략**

✓**Grid Search**

- 가능한 하이퍼파라미터 조합을 모두 탐색해 최적의 조합을 찾는 방식
- 직관적이고 단순한 방법이나 연산량이 많음(병렬 처리 가능)

```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier

param_grid = {
    'hidden_layer_sizes': [(64,), (128,), (64, 64)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam'],
    'alpha': [0.0001, 0.001],
    'learning_rate_init': [0.001, 0.01]
}

clf = GridSearchCV(MLPClassifier(max_iter=200), param_grid, cv=3)
clf.fit(X_train, y_train)
print("Best Params:", clf.best_params_)
```

모델 성능 개선 전략

2. BERT 모델의 성능 향상

✓**Fine-Tuning 전략**

✓**Grid Search**

- 가능한 하이퍼파라미터 조합을 모두 탐색해 최적의 조합을 찾는 방식
- 직관적이고 단순한 방법이나 연산량이 많음(병렬 처리 가능)

```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier

param_grid = {
    'hidden_layer_sizes': [(64,), (128,), (64, 64)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam'],
    'alpha': [0.0001, 0.001],
    'learning_rate_init': [0.001, 0.01]
}

clf = GridSearchCV(MLPClassifier(max_iter=200), param_grid, cv=3)
clf.fit(X_train, y_train)
print("Best Params:", clf.best_params_)
```

모델 성능 개선 전략

2. BERT 모델의 성능 향상

✓"Fine-Tuning 전략: BERT vs MLP"

항목	BERT Fine-Tuning	MLP Fine-Tuning
학습 대상	Transformer Layer	Fully Connected Layer
적용 위치	자연어 처리 할 때	BERT 출력
주요 전략	레이어 freeze/부분 학습, Adapters	Dropout, LR tuning, EarlyStopping
학습량	많음	적음
활용 예	분류, QA, NER 등	최종 예측, 앙상블

모델 성능 개선 전략

3. Hybrid 모델

✓Hybrid 모델이란?

- 서로 다른 유형의 데이터를 처리하는 모델을 결합하여 시너지 효과를 내는 방식
- 예: 정형 데이터(Meta info, Flow features) + 비정형 데이터(Payload text)

✓장점서로 다른 특성 보완

- 예: Port는 benign, Payload는 악성일 경우 상호보완적 판단 가능

✓모델 성능 향상

- 다양한 관점에서 특성 학습 → 일반화 능력 ↑

✓Explainable AI (XAI) 적용 용이

- 각각의 입력 특성에 대해 기여도 분석 가능