Snort 기반
IDS 탐지 환경 구축 및
AgentTesla 악성코드 탐지 프로젝트

작성자 : 김윤성

보고서 작성일 : 26.06.25

목차

1. 프로젝트 개요 및 핵심 성과

1.1 프로젝트 성과

본 프로젝트는 환경 구축부터 분석·탐지·검증·문서화까지 전 과정을 직접 수행하였으며, 핵심 산출물과 성과는 다음과 같다.

- **안전한 분석 환경 및 탐지 파이프라인 구축:** VMware 기반의 격리된 Windows 11 환경을 구축하고, Snort 2와 BASE 대시보드(Snort 탐지 결과를 웹으로 보여주는 도구)를 연동하여 실무 관제와 유사한 IDS(침입 탐지 시스템) 탐지 환경을 완성하였다.
- **기준 기반의 악성코드 수집·선별:** 주어진 샘플을 분석하는 데 그치지 않고 'C2(공격자가 감염 호스트를 원격 제어하는 서버) 통신'과 '평문 통신'이라는 실무적 기준을 세웠다. 이후 MalwareBazaar에서 조건에 부합하는 검체(CEaN.exe)를 직접 찾아 반복 검증으로 선별하였다.
- **악성코드 분석 및 IOC 도출:** 평문 통신 기반의 AgentTesla 패밀리의 CEaN.exe 검체를 대상으로 분석을 수행하여, 이 검체가 브라우저·메일 등의 자격증명을 수집해 평문 SMTP로 외부 유출하는 것을 확인하고 파일·레지스트리·네트워크 침해지표를 확보하였다.
- **Snort 탐지 룰 작성·검증:** 도출한 IOC를 바탕으로 탐지 룰을 작성하고, 실제 악성코드를 실행해 정찰·정보 유출 등 악성 행위가 BASE 대시보드에 탐지·출력되는 전 과정을 검증하였다.
- **트러블슈팅을 통한 문제 해결:** 환경 구축 중의 버전 호환성 문제, 난독화(코드를 읽기 어렵게 변형하는 기법)로 인한 정적 분석의 한계, 체크섬 오프로드(패킷 검증을 네트워크 카드 하드웨어에 넘기는 기능)로 인한 탐지 누락 등을 마주하였고, 원인을 단계적으로 좁혀 직접 해결하였다.
- **분석 보고서 도출:** 전체 분석 과정과 IOC, 탐지 룰을 실무 형식으로 문서화한 「악성코드 분석 보고서 - CEaN.exe (AgentTesla)」를 최종 산출하였다.

1.2 프로젝트 개요 및 요약

단순히 이론 학습에 머물지 않고, 실무와 유사한 보안관제·악성코드 분석 환경을 직접 부딪히며 경험하고자 본 프로젝트를 기획하였다. 처음 접하는 보안 실무 영역이라 모르는 것이 많았으나, 완벽히 이해한 뒤 넘어가기보다 우선 결과물을 만들고 부족한 부분을 채워 나가는 방식을 택하였다. 이를 통해 특정 기술의 습득을 넘어 '모르는 문제를 스스로 찾아 해결하는 능력'과 '어떤 일이든 해결할 수 있는 기본기'를 기르는 것을 최우선 목표로 삼았다.

본 프로젝트는 다음 세 가지 구체적 목표를 두었다.

- 첫째, 그동안 학습한 네트워크 기초 지식을 실제 트래픽·패킷 분석에 적용하는 것.
- 둘째, 실제 악성코드 검체를 직접 분석하여 동작 원리를 이해하는 것.
- 셋째, 도출한 지표로 탐지 룰을 직접 작성·검증하고 보고서까지 작성하며 보안관제 실무의 전체 흐름을 경험하는 것.

이를 위해 **탐지 환경 구성 → 악성코드 샘플 선정 → 악성코드 분석 → 분석 보고서 작성 → 탐지 룰 작성 → 탐지 검증**의 파이프라인 순서로 전 과정을 수행하였다. 격리된 가상 환경(VMware) 내에서 인포스틸러(정보 탈취형 악성코드) 유형인 AgentTesla 검체를 수집·분석하였고, 확보한 침해지표(IOC, 공격을 식별하는 흔적)를 바탕으로 Snort 탐지 룰을 적용해 BASE 대시보드에 탐지 알림이 출력되는 전 과정을 완수하였다. 다만 이번에는 기초 분석에 중점을 두었고, 코드 레벨 심화 분석은 범위에서 제외하였다.

1.3 주요 트러블슈팅 요약

표 1. 주요 트러블슈팅 요약

단계	문제 상황 · 원인	해결	상세
환경 구성	분리한 2-VM 구조에서 다른 VM의 악성 통신이 탐지되지 않음 — 가상환경은 미러링(트래픽 복사)을 지원하지 않아 목적지 외 패킷을 볼 수 없음	탐지 VM 내부에서 악성코드를 직접 실행하는 단일 VM 구조로 재설계	3.3
환경 구성	Snort·ADOdb 최신 버전이 BASE 연동에서 동작하지 않음 — output database 기능 제거, PHP 5.2 비호환 등 버전 간 호환성 문제	기능이 남아 있고 호환되는 구버전으로 다운그레이드	3.3
샘플 선정	평문으로 통신하고 C2가 살아 있는 샘플을 찾기 어려움 — 등록 샘플 상당수가 C2 종료 상태이거나 암호화 통신을 사용	자동 필터링 대신 다수 샘플을 직접 내려받아 통신을 반복 확인	4.3
악성코드 분석	언패킹으로 C2·자격증명 등 핵심 정보가 평문으로 나오지 않음 — de4dot 식별 실패, mal_unpack은 부분 덤프만 성공	동적 분석으로 검체를 실행해 실제 악성 행위의 결정적 증거를 확보	5.2
탐지 검증	룰을 작성했으나 여러 번 시도해도 탐지되지 않음 — Snort가 체크섬 미완성 패킷을 비정상으로 판단해 폐기(체크섬 오프로드)	원인을 트래픽→인터페이스→룰 문법→패킷 처리 순으로 좁혀 -k none 적용	6.3

2. 프로젝트 환경 구성 및 사용 도구

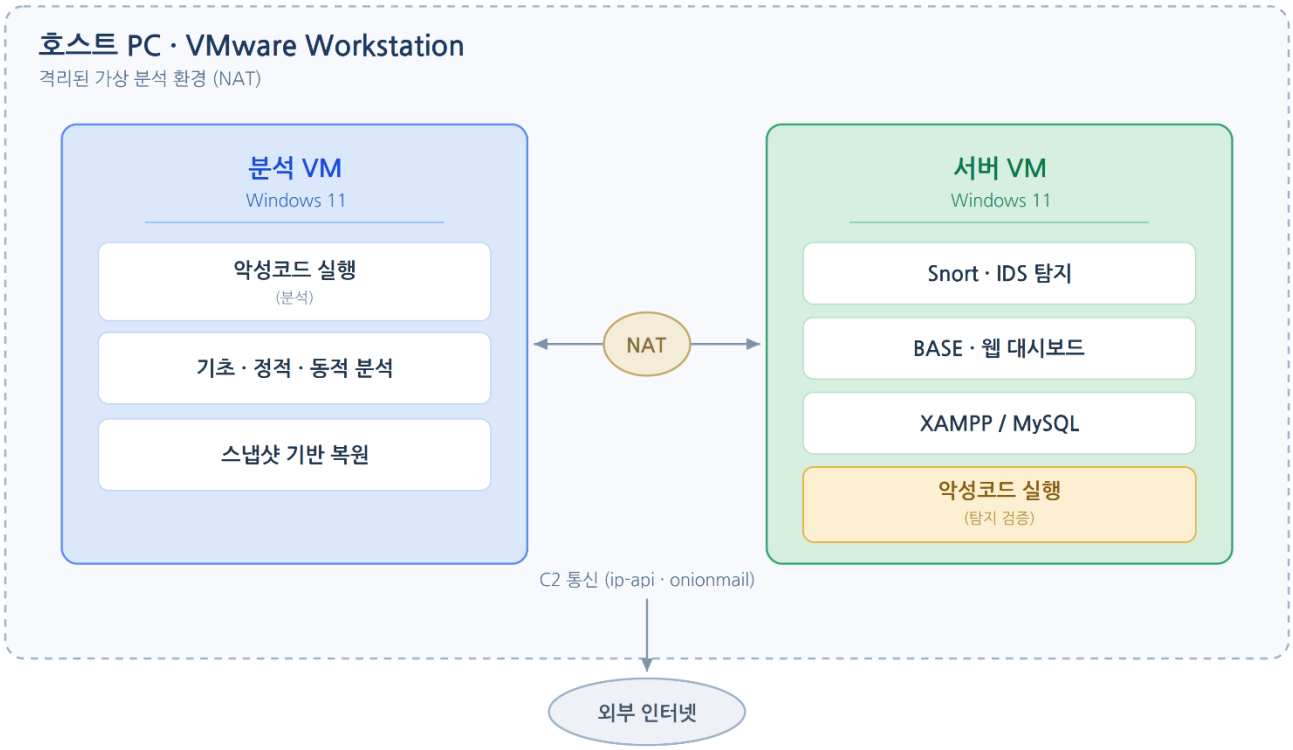
2.1 전체 아키텍처

본 프로젝트는 호스트 PC의 VMware를 사용하여 분석 VM과 서버 VM 두 대로 구성하였다. 분석 VM에서는 악성코드의 기초·정적·동적 분석을 수행하고, 서버 VM에는 Snort·BASE·XAMPP/MySQL로 탐지 환경을 구축하였다. 두 VM의 역할은 분리하되, 탐지 검증 단계에서는 서버 VM 내부에서 악성코드를 직접 실행해 그 통신을 같은 VM의 Snort가 탐지하는 방식을 택하였다.

실제 보안관제 환경에서는 사무실의 여러 대 PC가 스위치를 거쳐 통신하며, 이때 스위치가 들어오

는 모든 트래픽을 복사해 탐지 장비(IDS)로 전달한다. IDS는 복사된 트래픽을 분석하고, 그 결과를 SIEM(여러 보안 장비의 로그를 모아 분석하는 통합 관제 시스템) 같은 대시보드로 종합해 보여준다. 본 프로젝트도 이 구조를 본떠, 처음에는 악성코드를 실행하는 클라이언트 VM과 탐지를 담당하는 서버 VM을 분리한 2대 구조를 의도하였다.

그러나 가상 환경에는 트래픽을 미러링(복사)하는 기능이 없어, 직접 통신하는 대상 외에는 패킷을 볼 수 없다. 이 제약으로 인해 분리 구조 대신, 탐지 환경 VM 내부에서 악성코드를 직접 실행해 그 통신을 같은 VM의 Snort가 캡처하는 단일 VM 구조로 재설계하였다. 단일 VM에서는 미러링 없이 패킷을 그대로 캡처할 수 있기 때문이다. (대응 과정은 3.3 트러블슈팅 참고)



[그림 1. 분석 VM과 서버 VM으로 구성한 전체 아키텍처]

2.2 프로젝트에 사용된 툴 간단 설명

표 2. 프로젝트 사용 도구 목록 및 용도

분류	도구	주요 용도
가상화	VMware	격리된 가상머신(VM)을 구성해 악성코드를 안전하게 실행·분석
샘플 수집	MalwareBazaar	악성코드 공개 저장소(샘플 검색·다운로드)
자동 분석	VirusTotal	다수 백신 엔진으로 파일을 일괄 검사(탐지명·패밀리·행위 태그 확인)
기초 분석	ExeinfoPE, DIE (Detect It Easy)	실행파일의 컴파일러·패커 식별, 패킹(압축·암호화) 여부 확인. 두 도구로 교차 검증
언패킹	mal_unpack, de4dot	패킹된 파일을 풀어 원본을 추출(de4dot은 .NET 전용)

정적 분석	PEview	PE(윈도우 실행파일) 구조 확인(섹션 테이블·Import API)
정적 분석	BinText	파일 내 문자열 추출(URL·경로·명령어 등 단서 확보)
동적 분석	Process Explorer	프로세스 부모-자식 트리와 생성·종료 관계 확인
동적 분석	Autoruns	자동 실행 등록 항목(Run 키 등)으로 지속성 확인
동적 분석	Process Monitor (ProcMon)	파일·레지스트리·프로세스·네트워크 활동 실시간 기록
네트워크	CurrPorts	열린 포트·연결 확인(악성코드 통신 IP·포트 파악)
네트워크	Wireshark	패킷 분석기(통신 내용·핸드셰이크·pcap 상세 분석)
탐지	Snort	오픈소스 침입 탐지 시스템(IDS), 작성한 규칙으로 악성 트래픽 탐지
탐지	BASE	Snort 탐지 결과를 웹으로 보여주는 대시보드(PHP 기반)
탐지	XAMPP	Apache·MySQL·PHP를 묶은 패키지, BASE 구동 환경 제공
탐지	MySQL	관계형 데이터베이스, Snort 탐지 알림 저장

3. Snort + BASE 환경구성

3.1 환경구성

탐지 환경은 Snort가 탐지한 경보를 MySQL에 저장하고, BASE가 이를 읽어 웹 대시보드로 보여주는 구조이다. 다음 순서로 구축하였다.

① **Snort 설치 및 동작 확인** - Snort 2.9.2.3을 설치하고 snort.conf를 Windows 환경에 맞게 수정하였다. IDS 모드로 ICMP(ping) 탐지가 되는지까지 확인해, 탐지 엔진 자체의 정상 동작을 먼저 검증하였다.

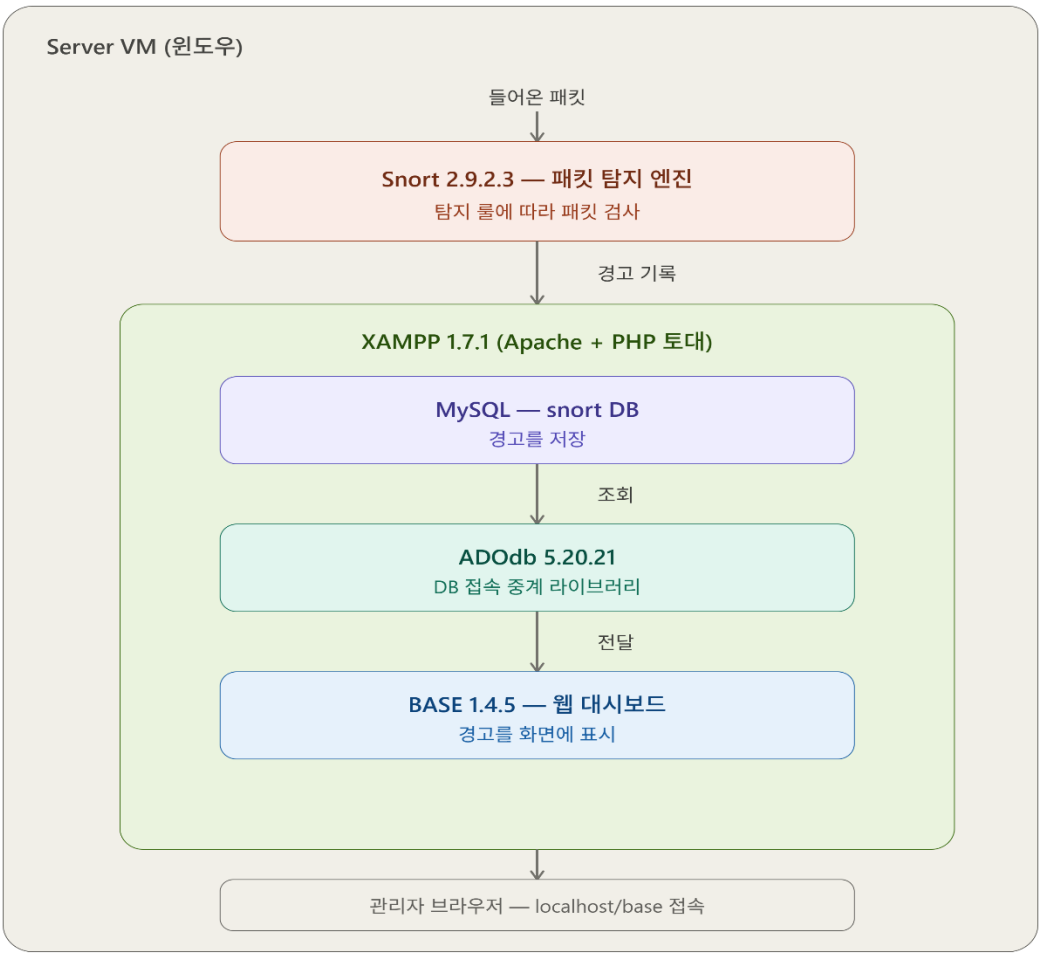
② **XAMPP 설치** - BASE가 동작할 토대인 Apache·PHP·MySQL을 한 번에 설치하는 XAMPP 1.7.1을 설치하였다. BASE는 PHP 웹 애플리케이션이며 경보를 MySQL에서 읽으므로, 이 세 가지가 먼저 준비되어야 한다.

③ **snort DB·스키마 생성** - MySQL에 경보를 저장할 snort 데이터베이스를 만들고, 경보가 담길 테이블 구조(스키마)를 적용하였다. 경보를 어디에 어떤 형태로 쌓을지 그릇을 먼저 만든 것이다.

④ **snort.conf에 DB 출력 설정 추가** - Snort가 경보를 ③의 MySQL DB로 직접 보내도록 output database 설정을 추가하였다. 이로써 탐지 측(Snort)과 저장 측(MySQL)이 연결된다.

⑤ **ADOdb 배치** - PHP가 MySQL에 접근하도록 중계하는 라이브러리 ADOdb 5.20.21을 배치하였다. BASE가 DB를 읽으려면 이 다리가 필요하다.

⑥ **BASE 설치·연결** - BASE 1.4.5를 설치하고 ③의 snort DB를 바라보도록 설정한 뒤, localhost/base로 접속해 대시보드 출력을 확인하였다.



[그림 2. Snort·BASE·MySQL을 연동한 서버 VM 내부 구성]

3.2 왜 Snort 2와 BASE인가

Snort 2 선택 이유

오픈소스 IDS/IPS인 Snort는 Snort 3가 최신 버전이다. Snort 3는 2021년 재설계된 버전으로 멀티스레드, JSON 출력, 플러그인 구조를 갖춰 확장성이 높지만, 아직 Windows를 공식 지원하지 않는다. 반면 Snort 2는 1998년에 나온 단일 스레드 구버전이라 성능·확장성이 떨어지고 룰 문법에도 민감하지만, Windows를 공식 지원하고 안정적으로 동작한다. 이번 프로젝트는 점유율이 높은 Windows 환경에서 탐지 환경을 구성하는 것이 목적이므로, 불편을 감수하고 Snort 2를 선택하였다. Snort 3 환경은 추후 별도로 다룰 계획이다.

표 3. Snort 2와 Snort 3 특징 비교

항목	Snort 2	Snort 3
출시·세대	1998년 출시된 구버전	2021년에 재설계된 현재 표준
개발 언어	C	C++ (구조 재설계)
스레드 처리	단일 스레드 (CPU 1개만 사용)	멀티스레드 (CPU 전체 활용)

설정 파일	snort.conf (텍스트 기반)	snort.lua (Lua 스크립트 기반)
설정 변경 반영	서비스 재시작 필요	재시작 없이 즉시 적용
로그 출력	텍스트 (외부 도구로 변환 필요)	JSON 네이티브 출력 (ELK·SIEM 즉시 연동)
확장 구조	기능이 본체에 고정	플러그인 구조 (자유롭게 확장)
룰 문법	공백·줄바꿈에 민감	유연한 문법 (sticky buffer 등)
Windows 지원	공식 지원, 안정적 동작	공식 미지원 (Linux 최적화, Windows 빌드는 비공식)
선택 결과	이번 프로젝트 채택	Phase 2-C에서 별도 진행 예정

BASE 선택 이유

Snort 로그를 SIEM(여러 보안 장비의 로그를 모아 분석하는 통합 관제 시스템) 환경처럼 보기 위한 대시보드로 Splunk, ELK, BASE를 검토하였다. Splunk는 구독형 상용 SIEM 제품으로, 이미 완성된 플랫폼이라 구독 후 사용하는 형태에 가깝다. 직접 환경을 구성하며 학습하려는 이번 목적과는 맞지 않아 제외하였다. ELK는 Snort 3와는 궁합이 좋으나 Snort 2에서는 연동 난이도가 올라가고 불안정해질 수 있어 제외하였으며, 추후 Linux 환경에서 Snort 3와 함께 다룰 계획이다. BASE는 오래된 도구지만, 화려한 기능보다 가볍게 탐지 흐름을 이해하기에 적합하다. 무엇보다 Windows 환경에서 구성이 가능하다는 점이 가장 큰 선택 이유였다.

표 4. 탐지 대시보드(SIEM) 도구 비교

항목	**Splunk**	**ELK**	**BASE**
유형	상용 SIEM 제품	오픈소스 로그 분석 스택	Snort 전용 PHP 대시보드
완성도	완성된 플랫폼 (기능 과다)	직접 구성형 파이프라인	경량 단일 대시보드
Snort 2 궁합	-	낮음 (텍스트 로그 → 수동 grok 파싱 필요)	높음 (output database 직접 연동)
Windows 구성	-	어려움	가능
학습 적합성	직접 구성 학습엔 부적합 (이미 완성된 제품)	현대 SIEM 구조 학습에 적합	흐름 이해·입문에 적합
제외·선택 사유	직접 구성 목적과 불일치, 기능 과함	Snort 2 연동 난이도 (추후 Linux+Snort 3에서 진행)	Windows 구성 가능 + 입문에 가벼움
결과	제외	제외	이번 프로젝트 채택

3.3 트러블슈팅 - 환경 제약과 버전 호환성 해결

① 탐지 구조 변경 - 가상환경의 미러링 제약

본 프로젝트는 실제 보안관제와 유사하게, 악성코드를 실행하는 클라이언트 VM과 탐지를 담당하는 서버 VM을 분리한 2대 구조를 의도하였다. 실무 관제에서는 스위치가 들어오는 모든 트래픽을 복사

해 탐지 장비(IDS)로 전달하고, IDS가 이를 분석하는 구조이기 때문이다.

그러나 가상 환경은 패킷을 목적지 주소에 해당하는 대상에게만 전달한다. 따라서 직접 통신하는 대상 외의 트래픽은 볼 수 없어, 다른 VM에서 발생한 악성 통신을 탐지 VM이 받아낼 수 없었다. 모든 트래픽을 받아들이는 무차별 모드(promiscuous mode, 자신에게 향하지 않은 패킷까지 모두 수신하는 모드)가 존재하지만, VMware에서는 보안 정책상 제한되어 있어 실무처럼 트래픽을 미러링(복사)해 탐지 장비로 보내는 구조를 만들 수 없었다.

이 제약으로 인해, 탐지 환경이 구성된 서버 VM 내부에서 악성코드를 직접 실행하는 단일 VM 구조로 재설계하였다. 단일 VM에서는 악성코드가 발생시킨 패킷을 미러링 없이 같은 VM의 Snort가 그대로 캡처할 수 있기 때문이다.

이 과정에서 프로젝트를 처음 계획한 의도대로 진행하지 못하는 경우가 있고, 실제 환경과 가상 환경 사이에 차이가 존재할 수 있음을 경험하였다. 중요한 것은 환경의 제약을 파악하고, 목적에 부합하도록 구조를 조정해 나가는 것임을 알게 되었다.

② 버전 호환성 - '최신'이 아니라 '연동되는' 버전 찾기

Snort + BASE 환경을 구성하면서, 처음에는 최신 버전일수록 더 나은 구성이라 판단해 Snort 3 비공식 빌드를 시도하였다. 그러나 Windows 설치 과정이 까다로워, 자료를 안정적으로 구할 수 있는 Snort 2로 방향을 바꾸었다. 이후 BASE 연동 과정에서 두 차례 다운그레이드를 거쳤다.

첫 번째는 Snort 본체였다. 처음 설치한 2.9.20에서는 BASE 연동에 필요한 output database(Snort가 탐지 결과를 MySQL로 직접 보내는 설정) 기능이 동작하지 않았다. 이 기능이 2.9.3.0부터 제거된 것을 확인하고, 기능이 남아 있는 마지막 버전인 2.9.2.3으로 내려 진행하였다.

두 번째는 ADOdb(PHP가 데이터베이스에 접속하도록 중계하는 라이브러리)였다. 최신 라인인 5.22.x를 받았으나, 이 버전은 PHP 7 이상에서만 동작해 XAMPP 1.7.1에 포함된 PHP 5.2와 호환되지 않았다. PHP 5를 지원하는 마지막 라인인 5.20.21로 내려 해결하였다.

두 다운그레이드 이후, Client에서 Server로 ping을 보내 BASE 대시보드에 alert가 표시되는 것을 확인하였다. 이 과정을 통해, 보안상 최신 버전 유지가 권장되지만 최신이 항상 더 나은 선택은 아니며, 실무에서도 도구 간 호환성 때문에 구버전을 유지해야 하는 경우가 있음을 알게 되었다.

(참고 블로그: https://itcase.tistory.com/entry/19-Snort-환경-구성)

③ Npcap 충돌 - 네트워크 카드 미인식

Snort 실행 전 snort -W(사용 가능한 네트워크 카드 목록을 출력하는 명령)로 인터페이스를 확인했으나 목록이 비어 있었다. 원인은 C:₩Snort₩bin에 남은 옛 캡처 라이브러리(Packet.dll·wpcap.dll)가 현재 사용하는 Npcap(Windows용 패킷 캡처 드라이버)과 충돌한 것이었다. 두 파일을 삭제한 뒤 카드가 정상 인식되고 ping 통신이 복구되는 것을 확인하였다.

4. 악성코드 샘플 선정

4.1 선정 기준

본 프로젝트는 실제 보안관제와 유사한 네트워크 패킷 분석 경험을 목표로 한다. 따라서 샘플 선정의 핵심 기준은 두 가지였다.

첫째, **네트워크 행위로 악성 행위를 수행하는 악성코드일 것.** 스파이웨어, 백도어, 인포스틸러(정보 탈취형 악성코드), RAT(원격 제어 악성코드) 계열이 여기 해당한다. C2(Command & Control, 공격자가 감염 호스트를 원격 제어하는 서버) 통신이 발생해야 탐지 룰을 작성할 대상이 생기기 때문이다.

둘째, **통신이 암호화되지 않을 것.** payload(패킷의 실제 데이터) 기반 Snort 시그니처를 작성하려면 패킷 내용을 읽을 수 있어야 한다. 암호화되면 IP·포트·통신 패턴 같은 메타데이터 기반 룰만 가능해 탐지 정밀도가 떨어진다.

이 두 기준으로 RedLine, AgentTesla, RAT 패밀리 등을 검색해 샘플을 수집하였다. 이후 CurrPorts와 Wireshark로 C2 통신 여부와 평문 통신 여부를 직접 확인하였고, 그 결과 AgentTesla 패밀리의 CEaN.exe를 선정하였다.

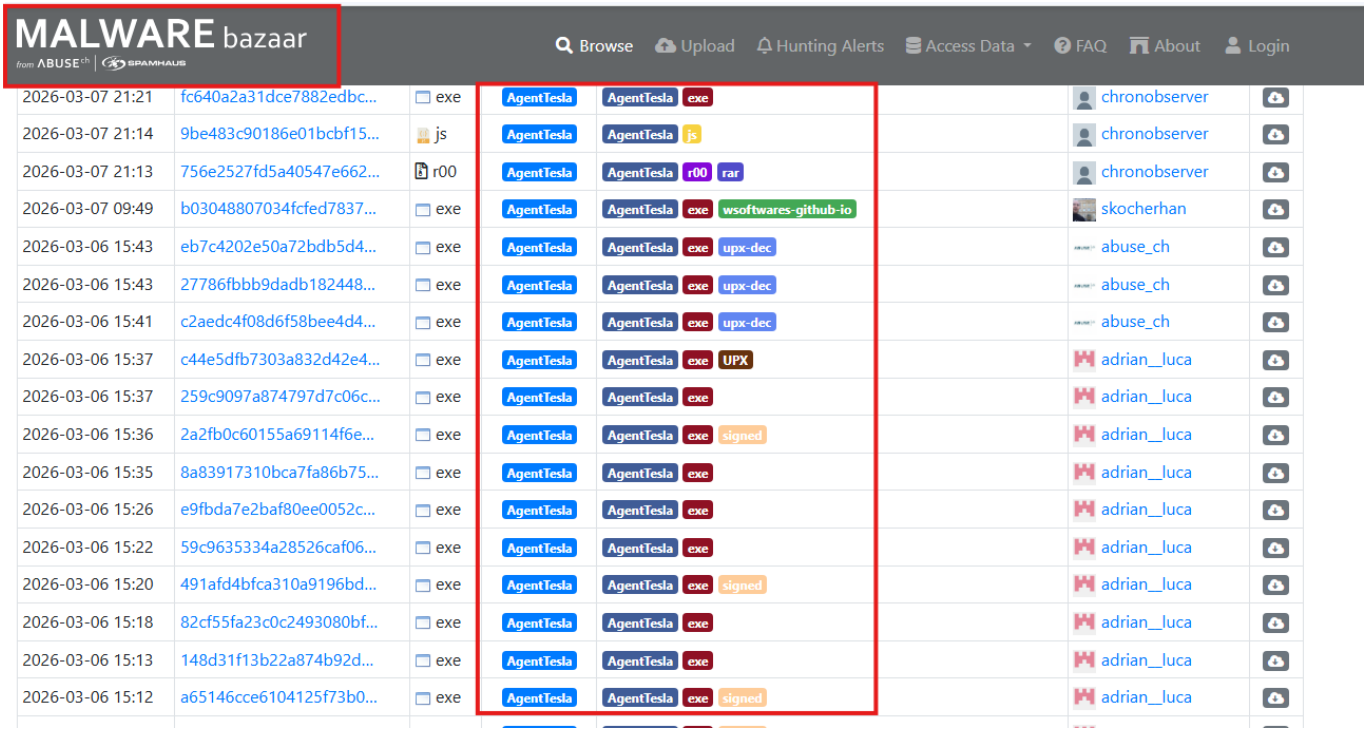
4.2 샘플 선정 과정

악성코드 샘플을 받을 수 있는 다양한 사이트가 있지만 대표적으로 MalwareBazaar, VirusShare, VirusTotal, Hybrid Analysis 등이 있다. 이 중 MalwareBazaar를 선정하였으며, 이유는 접근성·신뢰성·수집 편의성을 모두 충족하기 때문이다.

VirusShare는 신뢰도가 높으나 운영자 승인을 거쳐야 가입이 가능해 접근성이 낮다.
VirusTotal·Hybrid Analysis는 자동화 분석 플랫폼(악성코드를 격리 환경에서 자동 실행·관측하는 sandbox)으로 샘플 수집도 가능하나, VirusTotal은 유료 계정에서만 다운로드가 허용되고 Hybrid Analysis는 회원가입이 필요해 본 단계에서는 제외하였다. 반면 MalwareBazaar는 가입 없이 다운로드가 가능하고 태그 기반 검색을 지원해, 원하는 조건의 샘플을 효율적으로 좁힐 수 있었다.

표 5. 악성코드 샘플 수집 사이트 비교

사이트	접근성	신뢰성	수집 편의성
MalwareBazaar	높음 (가입 불필요)	높음 (업로더·태그 검증)	높음 (태그 검색 지원)
VirusShare	낮음 (운영자 승인 필요)	높음	보통
VirusTotal	낮음 (유료 계정 필요)	높음	높음
Hybrid Analysis	보통 (회원가입 필요)	높음	보통



2026-03-07 21:21	fc640a2a31dce7882edbc...	exe	AgentTesla	AgentTesla exe	chronobserver	
2026-03-07 21:14	9be483c90186e01bcbf15...	js	AgentTesla	AgentTesla js	chronobserver	
2026-03-07 21:13	756e2527fd5a40547e662...	r00	AgentTesla	AgentTesla r00 rar	chronobserver	
2026-03-07 09:49	b03048807034fcfed7837...	exe	AgentTesla	AgentTesla exe wsoftwares-github-io	skocherhan	
2026-03-06 15:43	eb7c4202e50a72bdb5d4...	exe	AgentTesla	AgentTesla exe upx-dec	abuse_ch	
2026-03-06 15:43	27786fbbb9dadb182448...	exe	AgentTesla	AgentTesla exe upx-dec	abuse_ch	
2026-03-06 15:41	c2aedc4f08d6f58bee4d4...	exe	AgentTesla	AgentTesla exe upx-dec	abuse_ch	
2026-03-06 15:37	c44e5dfb7303a832d42e4...	exe	AgentTesla	AgentTesla exe UPX	adrian_luca	
2026-03-06 15:37	259c9097a874797d7c06c...	exe	AgentTesla	AgentTesla exe	adrian_luca	
2026-03-06 15:36	2a2fb0c60155a69114f6e...	exe	AgentTesla	AgentTesla exe signed	adrian_luca	
2026-03-06 15:35	8a83917310bca7fa86b75...	exe	AgentTesla	AgentTesla exe	adrian_luca	
2026-03-06 15:26	e9fbda7e2baf80ee0052c...	exe	AgentTesla	AgentTesla exe	adrian_luca	
2026-03-06 15:22	59c9635334a28526caf06...	exe	AgentTesla	AgentTesla exe	adrian_luca	
2026-03-06 15:20	491afd4bfca310a9196bd...	exe	AgentTesla	AgentTesla exe signed	adrian_luca	
2026-03-06 15:18	82cf55fa23c0c2493080bf...	exe	AgentTesla	AgentTesla exe	adrian_luca	
2026-03-06 15:13	148d31f13b22a874b92d...	exe	AgentTesla	AgentTesla exe	adrian_luca	
2026-03-06 15:12	a65146cce6104125f73b0...	exe	AgentTesla	AgentTesla exe signed	adrian_luca	

[그림 3. MalwareBazaar에서 태그 기반으로 샘플을 탐색하는 화면]

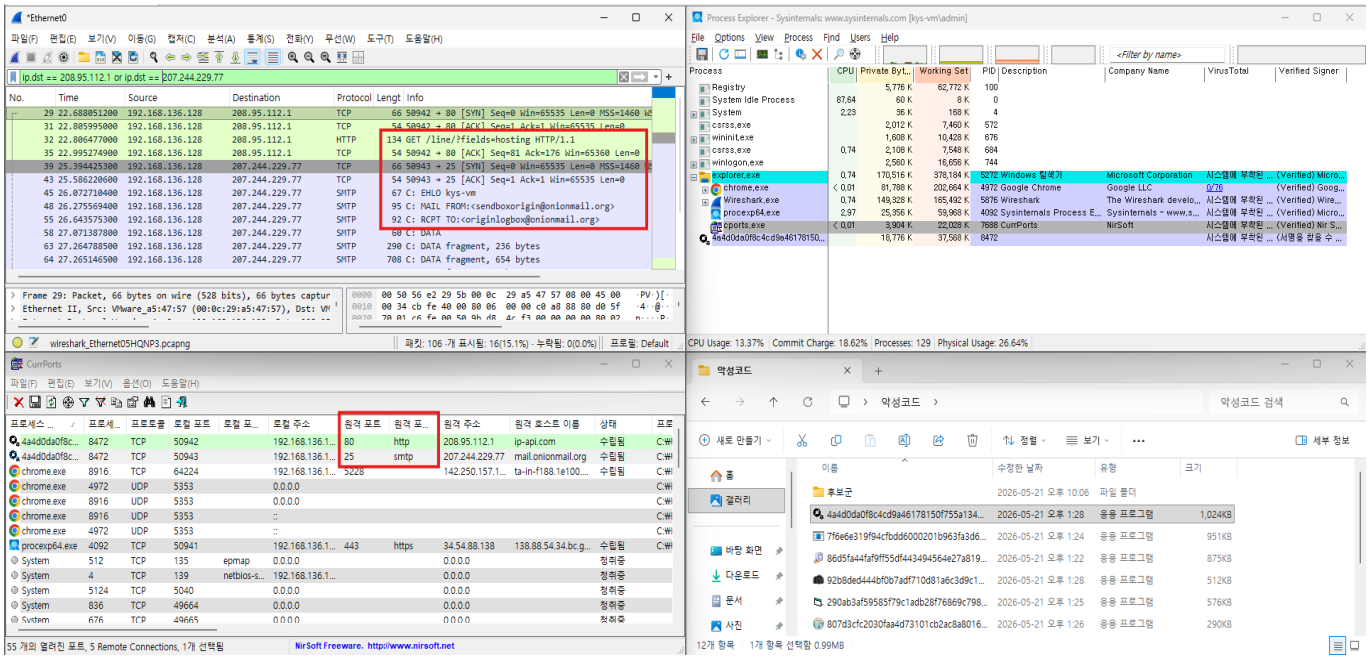
4.3 트러블슈팅 - 조건에 맞는 샘플을 반복 검증으로 확보

C2 서버가 살아 있고 평문으로 통신하는 샘플을 찾는 것은 생각보다 어려움이 있어다. 원인은 두 가지였다.

첫째, 상당수 샘플은 C2 서버 연결조차 확인되지 않았다. 샘플 사이트에 등록되는 악성코드는 이미 악성 행위가 탐지된 것들이라, 추가 피해 차단이나 추적 회피를 위해 C2 서버가 종료된 경우가 많은 것으로 추정된다. 둘째, 행위를 은닉하기 위해 평문보다 암호화 통신을 사용하는 샘플이 더 많았다. 평문 통신은 payload(패킷의 실제 데이터) 기반 Snort 시그니처 작성의 전제 조건이므로, 암호화 샘플은 본 프로젝트 목적에 부적합하다.

효율을 높이기 위해, sandbox(악성코드를 격리 환경에서 자동 실행·관측하는 자동화 분석 플랫폼) 기반 분석 사이트에서 조건에 맞지 않는 샘플을 사전에 걸러내는 방법도 고려하였다. 그러나 분석 사이트 가입에도 시간이 걸릴뿐더러, 자동화 분석 결과만으로는 C2 서버가 실시간으로 살아 있는지 확인하는 데 한계가 있다고 판단하였다. 결국 이 상황에서는 다수의 샘플을 우선 내려받아 네트워크 통신을 직접 확인하는 단순 반복 검증이 더 빠른 방법이라고 보았다.

이 과정을 통해, 모든 문제에 정교한 요령이 항상 더 효율적인 것은 아니며, 조건 충족률이 낮은 모집단에서는 직접적인 반복 검증이 더 합리적일 수 있음을 체감하였다.



[그림 4. CurrPorts·Wireshark로 C2 통신과 평문 여부를 확인하는 과정]

5. 악성코드 분석

5.1 악성코드 분석 요약

분석 대상 CEaN.exe(AgentTesla 계열, VirusTotal 54/69 탐지)는 트로이목마·인포스틸러(정보 탈취형 악성코드) 유형으로, VMware 기반 격리 환경에서 분석하였다.

CEaN.exe는 .NET 계열 악성코드로, 표준 패커가 아닌 .NET 내부 압축·난독화(코드를 읽기 어렵게 변형하는 기법)로 실제 페이로드(악성 기능 본체)를 숨긴 형태였다. 완전한 언패킹(압축·암호화된 페이로드를 메모리에서 복원하는 과정)은 이루어지지 않았으나, mal_unpack(메모리 덤프 기반 언패킹 도구)으로 일부 문자열을 확보하였다. 여기서 호텔 예약·관리 서비스 또는 AI 플랫폼으로 위장한 흔적과 한국어·중국어 리소스가 확인되어, 동아시아 사용자를 노린 정황이 관찰되었다.

실행 시에는 외부 프로세스 없이 자기 자신을 자식 프로세스로 재실행해 본체를 동작시킨다(.NET 런타임이 메모리에서 페이로드를 전개하는 구조로, 디스크 기반 백신 스캔을 회피하는 효과가 있다). 이후 자신을 `%APPDATA%\Roaming\JVKPZv.exe`로 복사해 `S·H`(시스템·숨김) 속성으로 은폐하고, .ps1 런처를 드롭한 뒤 `HKCU…\Run` 키에 PowerShell 실행 항목을 등록해 로그인 시마다 본체가 자동 재실행되는 지속성을 확보한다(정상 도구를 악용하는 LotL 기법).

이어 브라우저(Chrome·Edge 등)·VPN·FTP·메일 등의 자격증명을 광범위하게 탐색해 저장된 비밀번호와 Windows 자격증명까지 수집하고, `ip-api[.]com`으로 분석 환경 여부를 정찰한 뒤, 수집 정보를 `onionmail[.]org`로 평문 SMTP를 통해 유출한다. 메일 전송 완료 응답까지 확인해 실제 유출을 검증하였다.

분석 결과를 바탕으로 파일·레지스트리·네트워크 침해지표(IOC)를 정리하고, 이를 근거로 Snort 탐지

룰을 작성·검증하였다(6장). 코드 및 메모리 단위의 심화 분석은 본 프로젝트 범위에서 제외하였다. 상세 분석 내용은 별도 첨부한 「 악성코드 분석 보고서 [CEaN.exe (AgentTesla)] 」에 기술되어 있다.

5.2 트러블슈팅 - 언패킹 한계 대응과 ProcMon 활용

① 언패킹 실패 - 정적·동적 분석의 상보성

악성코드 분석에서 자주 부딪히는 어려움은 패킹된 검체의 언패킹이다. UPX 같은 표준 패커라면 전용 도구로 쉽게 풀 수 있지만, 공격자는 분석을 방해하기 위해 암호화나 난독화를 적용한다.

이번 분석에서는 de4dot과 mal_unpack으로 언패킹을 시도하였다. de4dot은 "Unknown Obfuscator"로 난독화 종류 자체를 식별하지 못해 실패하였고, mal_unpack으로는 메모리에서 모듈을 덤프하는 데 성공해 위장 정체와 암호화 사용 정황 등은 확보하였다. 다만 C2 주소나 자격증명 같은 핵심 정보는 평문으로 나오지 않았다.

처음에는 정적 분석을 위해 언패킹을 완전히 끝내 문자열을 전부 확보해야 한다고 생각하였다. 그러나 분석을 거듭하면서, 이 검체처럼 정적 분석만으로는 핵심 정보가 드러나지 않는 경우도 있음을 알게 되었다. 정적 분석으로 확보한 문자열은 분석에 추가 정보를 주고 방향을 잡아준다는 분명한 장점이 있지만, 그 자체가 모든 것을 밝혀주는 필수 조건은 아니었다. 실제 악성 행위의 결정적 증거는 동적 분석으로 검체를 실행해 확보할 수 있었고, 결국 정적·동적 분석은 어느 한쪽이 더 중요하다기보다 서로를 보완하는 관계임을 이해하게 되었다.

② ProcMon 활용 - 필터 순서 고정 방법론

ProcMon(Process Monitor)은 파일·레지스트리·프로세스·네트워크 활동을 한꺼번에 기록하는 도구로, 그만큼 로그량이 많다. 따라서 사용 목적을 명확히 하고 필터 순서를 정해두는 것이 관건이다.

ProcMon은 처음부터 새로 관찰을 시작하는 도구라기보다, 앞선 분석에서 발견한 행위를 더 자세히 파헤치고 그 증거를 시간순으로 잡아내는 용도가 크다. 이에 분석에 불필요한 노이즈를 먼저 걸러내고, 앞선 분석에서 확인한 행위에 필터 순서를 고정해두는 방식으로 접근하였다. 이렇게 하면 방대한 로그 속에서도 필요한 내용을 수월하게 확인할 수 있다.

6. 탐지 검증

6.1 작성한 Snort 룰

악성코드 분석 결과로 정리한 IOC를 바탕으로, Snort 2 문법으로 탐지 룰을 작성하였다. 단일 지표에 의존하지 않도록 정찰·DNS 질의·SMTP 유출 등 다양한 악성행위를 기반으로 구성하였다.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"[AgentTesla] ip-api reconnaissance
request"; flow:established,to_server; content:"/line/?fields=hosting"; http_uri; nocase;
sid:1000007; rev:1;)
```

대표적으로 sid:1000007 룰은 CEaN.exe가 `ip-api[.]com`과 통신해 분석 환경 여부를 확인하던 정찰 패킷을 기반으로 작성하였다. 내부망($HOME_NET)의 모든 포트에서 외부($EXTERNAL_NET) 80번 포트로 나가는 TCP 통신을 대상으로 하되, `flow:established,to_server` 옵션으로 연결이 수립된 뒤

클라이언트→서버 방향으로 가는 패킷만 검사해 불필요한 오탐을 줄였다. 핵심 탐지 조건은 `content:"/line/?fields=hosting"`으로, `http_uri` 옵션을 통해 이 문자열을 HTTP 요청의 URI 영역에 한정해 검사한다. `nocase`는 대소문자를 구분하지 않아 우회를 방지한다.

표 6. 작성한 Snort 탐지 룰 목록

sid	탐지 대상	탐지 지점
1000001	C2(유출 서버) IP 통신	onionmail 유출 IP 3종(5[.]189[.]162[.]105 / 207[.]244[.]229[.]77 / 173[.]249[.]33[.]206)
1000004	외부 SMTP 아웃바운드	25번 포트로 나가는 SMTP 통신
1000005	악성 도메인 DNS 질의	onionmail
1000006	악성 도메인 DNS 질의	ip-api
1000007	분석 환경 정찰	HTTP URI 패턴 /line/?fields=hosting
1000008	SMTP 발신 주소	MAIL FROM + sendboxorigin@onionmail[.]org
1000009	SMTP 수신 주소	RCPT TO + originlogbox@onionmail[.]org
1000010	유출 메일 제목 패턴	Subject: PW_
1000011	유출 본문(자격증명·시스템정보)	Username: + Computer Name: + OSFullName:

*더 자세한 탐지룰은 별도 첨부한 「 악성코드 분석 보고서 [CEaN.exe (AgentTesla)] 」부록 2에 기술되어 있다.

6.2 검증 과정

탐지 검증은 다음 순서로 진행하였다.

① **BASE 대시보드 준비** - XAMPP를 실행해 Apache와 MySQL을 동작시키고, 웹 브라우저에서 localhost/base에 접속해 BASE 대시보드가 정상적으로 열리는 것을 확인하였다.

② **Snort IDS 모드 실행** - Snort 설치 폴더로 이동해 `snort -W`로 캡처할 네트워크 인터페이스를 확인한 뒤, `snort -c C:\Snort\etc\snort.conf -i 4 -k none` 명령으로 Snort를 IDS 모드로 실행하였다.

③ **테스트룰로 환경 점검** - 탐지 누락 시 룰 문제인지 환경 문제인지 구분하기 위해, 사전에 로컬룰에 저장한 파이프라인 점검 룰(`alert icmp any any -> any any (msg:"ICMP test ping detected"; sid:1000099; rev:1;)`)로 환경의 정상 동작을 먼저 점검하였다. 새 터미널에서 `ping 8.8.8.8`을 실행한 뒤, Snort 터미널의 실시간 로그와 BASE 전달을 확인하였다.

④ **악성코드 실행** - 룰 검증을 위해 악성코드를 실행하였다.

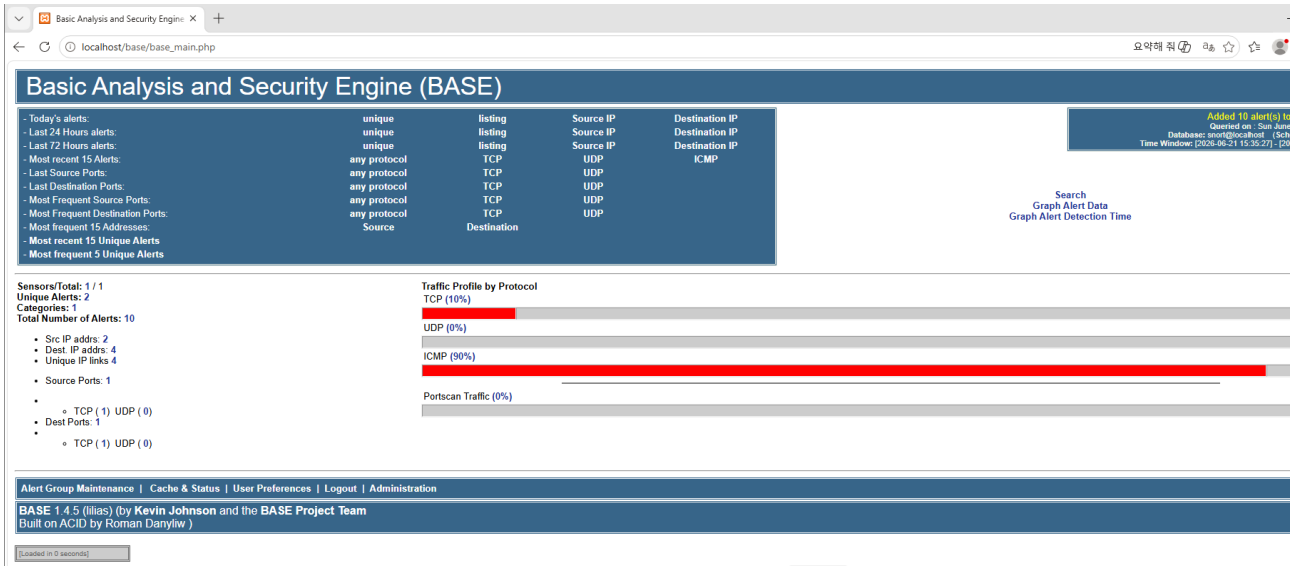
⑤ **탐지 결과 확인** - 악성코드 실행 후 Snort 터미널에 실시간 로그가 출력되고 BASE 대시보드에도 정상적으로 전달되는 것을 확인하였다. ip-api 정찰(sid:1000007), 악성 도메인 DNS 질의(sid:1000005·1000006), SMTP 발신·수신 주소(sid:1000008·1000009), 유출 본문(sid:1000011) 등이 탐지되어, 정찰부터 유출까지 악성 행위의 각 단계가 작성한 룰에 포착됨을 검증하였다.

```
C:\Snort\bin>snort -W

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.2.3-ODBC-MySQL-WIN32 GRE (Build 205)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-team
           Copyright (C) 1998-2012 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.3

Index   Physical Address        IP Address      Device Name     Description
-----   ----------------        ----------      -----------     -----------
    1   00:00:00:00:00:00       disabled        \Device\NPF_{14E03716-210B-4B77-A752-412DEFD6B574}   W
AN Miniport (Network Monitor)
    2   00:00:00:00:00:00       disabled        \Device\NPF_{091DF9BC-91D7-4E9E-B20E-5CD83EE80777}   W
AN Miniport (IPv6)
    3   00:00:00:00:00:00       disabled        \Device\NPF_{2DFDFEC2-0B89-4E65-830C-B5EFB7312C3C}   W
AN Miniport (IP)
    4   00:0C:29:9C:24:4D       192.168.136.145 \Device\NPF_{64E99978-6FB8-4903-A221-010FA68F5F74}   I
ntel(R) 82574L Gigabit Network Connection
    5   00:00:00:00:00:00       disabled        \Device\NPF_Loopback    Adapter for loopback traffic c
apture

C:\Snort\bin>snort -i 4 -c C:\Snort\etc\snort.conf -k none
```

[그림 5. Snort IDS 모드 실행하는 화면]

```
Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
Commencing packet processing (pid=6188)
06/21-15:35:27.606100  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 192.168.136.2
06/21-15:35:58.450705  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 8.8.8.8
06/21-15:35:58.492477  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.136.145
06/21-15:35:59.461849  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 8.8.8.8
06/21-15:35:59.493612  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.136.145
06/21-15:36:00.475800  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 8.8.8.8
06/21-15:36:00.509253  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.136.145
06/21-15:36:01.481212  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 8.8.8.8
06/21-15:36:01.516064  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.136.145
06/21-15:36:46.771573  [**] [1:1000007:1] [AgentTesla] ip-api reconnaissance request [**] [Priority: 0] {TCP} 192.168.136.145:64810 -> 208.95.112.1:80
06/21-15:36:50.903557  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:50.924772  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 192.168.136.2
06/21-15:36:51.133974  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:51.872787  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.108360  [**] [1:1000008:1] [AgentTesla] SMTP sender address [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.108360  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.362799  [**] [1:1000009:1] [AgentTesla] SMTP recipient address [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.362799  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.638422  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.888103  [**] [1:1000010:1] [AgentTesla] SMTP mail subject pattern [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.888103  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.888797  [**] [1:1000011:1] [AgentTesla] DATA body exfiltration [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.888797  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.889162  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:52.889514  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:36:53.732003  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:38:30.933257  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:38:31.168943  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:38:31.171170  [**] [1:1000004:1] [AgentTesla] outbound SMTP(25) traffic [**] [Priority: 0] {TCP} 192.168.136.145:63786 -> 152.53.162.126:25
06/21-15:39:13.831353  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 192.168.136.2
06/21-15:39:14.645413  [**] [1:9999999:1] TEST - ICMP detected pipeline check [**] [Priority: 0] {ICMP} 192.168.136.145 -> 192.168.136.2
```

[그림 6. 악성 행위가 탐지되어 터미널에 출력된 실시간 로그]



[그림 7. 작성한 룰에 의해 BASE에 탐지·출력된 결과]

6.3 트러블슈팅 - 체크섬 오프로드로 인한 탐지 누락 해결

분석 결과와 IOC로 작성한 룰을 local.rules에 저장하고 검증했으나, 여러 번 시도해도 탐지되지 않았다. 원인을 단계적으로 배제하며 좁혔다.

① **트래픽 부재 가능성 배제** - 악성코드의 네트워크 활동은 언제든 중단될 수 있어, 탐지 실패가 트래픽 부재 때문일 가능성을 먼저 확인하였다. Wireshark로 캡처한 결과 악성코드는 여전히 통신 중이었다. 따라서 원인은 악성코드가 아닌 탐지 룰 또는 Snort 환경으로 좁혀졌다.

② **인터페이스 번호 오지정 정정** - `snort -W`로 재확인한 결과, 최초 실행 시 인터페이스 번호를 잘못 지정했음을 발견하였다. 번호를 정정해 재실행했으나 탐지는 여전히 되지 않았다.

③ **룰 파일 문법 검증** - 룰 파일 자체의 문법 문제인지, 혹은 Snort 엔진(인터페이스 캡처 → 룰 매칭 → 알람 발생)이 애초에 정상 동작하는지를 함께 확인하기 위해 테스트룰 (alert icmp any any -> any any (msg:"ICMP test ping detected"; sid:1000099; rev:1;)) 을 추가하고 문법 검사를 거쳐 이상이 없음을 확인하였다. ping 테스트 결과 테스트룰은 정상 동작했다. 이는 Snort 엔진 전체 파이프라인이 정상이며 룰 문법에도 문제가 없음을 의미한다. 그러나 악성코드 관련 탐지는 여전히 이루어지지 않았다. 이로써 룰 문법과 Snort 동작에는 문제가 없음이 확인되었으며, 원인은 사용자 설정이 아닌 Snort의 패킷 검증 단계로 좁혀졌다.

④ **체크섬 오프로드 확인 및 해결** - 원인은 체크섬 오프로드였다. 체크섬 오프로드란 패킷의 오류 검사값(체크섬)을 소프트웨어가 아닌 네트워크 카드(하드웨어)에서 채우도록 넘기는 기능이다. 현재 환경에서는 악성코드가 패킷을 만들 때 체크섬을 비워두고, 마지막에 랜카드가 체크섬을 채워 외부로 내보내는 구조였다. 이때 Snort는 랜카드에 도달하기 전의 패킷을 캡처하는데, 체크섬이 비어 있어 이를 비정상 패킷으로 판단하고 룰과 매칭하지 않은 채 버렸다. 그 결과 탐지가 이루어지지 않은 것이다. Snort 실행 시 체크섬 검증을 비활성화하는 `-k none` 옵션을 적용하자, 체크섬이 채워지지 않은 패킷도 검사 대상에 포함되어 작성한 룰이 정상 탐지되는 것을 확인하였다.

이번 과정에서, 재부팅이나 드라이버 변경 시 네트워크 인터페이스 번호가 바뀔 수 있어 Snort 실행

전 인터페이스를 확인하는 습관이 필요함을 알았다. 또한 문제 발생 시 원인 구간을 가르는 테스트 룰이 유용하다고 판단해, 앞서 사용한 테스트룰을 삭제하지 않고 검증 절차의 사전 단계로 상시 편입하였다.

7. 회고 및 발전 계획

7.1 회고

이 프로젝트는 분석·탐지 환경 구성부터 샘플 선정, 악성코드 분석, 탐지 룰 작성·검증, 보고서 작성까지 보안관제의 한 흐름을 직접 수행한 경험이었다.

진행 과정에서 대부분이 처음이라 모르는 것이 많았으나, 모른다는 사실에 위축되기보다 하나씩 찾아가며 진행하였다. 처음에는 모든 것을 완벽히 이해한 뒤 다음으로 넘어가려 했지만, 막힐 때마다 멈춰 서기보다 우선 결과물을 만들고 부족한 부분을 다시 채우는 방식이 이해와 효율 모두에서 효과적임을 확인하였다. 완벽한 준비보다 일단 부딪쳐 결과물을 만들어보고 개선하는 것이 더 많은 것을 배우게 한다는 점이 이번 프로젝트의 가장 큰 깨달음이었다.

결국 이번 프로젝트는 특정 기술 하나를 익혔다는 것 이상으로, 모르는 문제를 스스로 찾아 해결하는 능력, 처음 접하는 분야에도 도전하는 자세, 진행 과정을 빠짐없이 기록·정리하는 습관 등 어떤 일이든 해결할 수 있는 기본기를 기르는 계기가 되었다. 특히 분석부터 탐지까지의 전 과정을 실무를 가정해 문서로 정리하면서, 보고서 작성이 큰 비중을 차지하는 보안 업무의 특성을 함께 경험하였다.

7.2 발전 계획

이번 프로젝트는 점유율이 높은 Windows 환경에서 Snort 2와 BASE로 안정적인 탐지 환경을 우선 완성하는 데 중점을 두었다. 이를 기반으로 다음 방향의 확장을 계획하고 있다.

가장 큰 방향은 **환경의 현대화**다. Snort 3는 더 간편한 설정과 다양한 기능을 갖춘 표준이지만 Windows를 공식 지원하지 않으므로, Linux 환경에서 Snort 3를 구성하면 이 한계를 해소하면서 최신 구성을 경험할 수 있다. 실무에서도 사무실 PC는 주로 Windows를 쓰지만 관제 장비나 서버는 가볍고 라이선스 제약이 적은 Linux를 많이 사용하므로, 관제 서버를 Linux로 구성하는 것은 실무 환경에 더 가깝다. 이와 함께 시각화·분석 도구도 현재의 BASE에서 실무 표준에 가까운 ELK Stack(Elasticsearch·Logstash·Kibana, 로그 수집·분석·시각화 스택)으로 옮기면 SIEM 환경에 더 가까운 경험을 쌓을 수 있다. 나아가 이 구성에서는 탐지에 그치는 IDS를 넘어 차단까지 수행하는 IPS(침입방지 시스템)도 다뤄볼 수 있다.

분석 단계에서도 다음과 같은 개선점을 확인하였다.

① **샘플 파일명 사전 정리** - 샘플의 초기 파일명은 보통 해시값이라 분석·문서화 과정에서 다루기 번거롭다. VirusTotal로 원본 파일명을 확인해 알아보기 쉬운 이름으로 미리 바꿔두면 이후 작업이 수월하다.

② **언패킹 덤프 범위 확대** - mal_unpack은 종료 트리거 기본값(/trigger A)이 첫 페이로드 탐지 시점

에 덤프 후 종료되도록 설정되어 있어, 다단계로 전개되는 후속 페이로드를 놓칠 수 있다. 다음 분석에서는 /trigger T와 충분히 긴 /timeout을 함께 적용해 더 많은 단계의 덤프를 확보할 수 있을지 시도해볼 계획이다.