

수업 내용 정리 보고서

작성일: 2026년 6월 30일

챕터 구성

- Snort의 작동 원리와 탐지 규칙 구조 분석
- 로그 출력 옵션에 따른 탐지 데이터 비교
- 운영 명령어 최적화 및 네트워크 미러링 실습

Snort의 작동 원리와 탐지 규칙 구조 분석

네트워크 침입 탐지 시스템(NIDS)인 Snort는 네트워크 트래픽을 실시간으로 분석하여 악성 활동을 탐지하는 보안 솔루션입니다. Snort는 단순한 탐지 도구를 넘어, 설정에 따라 침입 방지 시스템(IPS)으로도 동작할 수 있으며, 호스트 기반(HIDS)으로도 활용될 수 있는 유연성을 가집니다.

1. Snort의 패킷 처리 아키텍처

Snort가 네트워크 패킷을 수집하고 분석하는 과정은 정해진 데이터 흐름을 따릅니다. 패킷은 가장 먼저 **NIC(Network Interface Card)**를 통해 들어오며, 이후 **DAQ(Data Acquisition library)** 계층을 거치게 됩니다. DAQ는 Snort가 다양한 환경(예: Live Packet, PCAP 파일, 특정 OS의 패킷 드라이버 등)에서 패킷을 효율적으로 획득할 수 있도록 추상화된 인터페이스를 제공합니다.

DAQ를 통해 수집된 패킷은 다음과 같은 단계를 거쳐 처리됩니다:

- **Decoder:** 수집된 로우(Raw) 데이터를 Snort가 이해할 수 있는 형태의 패킷 구조로 해독합니다.
- **Preprocessor:** TCP 세그먼트 재조립, IP 단편화(Fragmentation) 처리 등 탐지 엔진이 분석하기 좋게 데이터를 정규화합니다.
- **Detection Engine:** 사전에 정의된 탐지 규칙(Rule)과 패킷의 내용을 비교하여 일치 여부를 판단합니다.

실제 Snort를 빌드하거나 설치할 때, 어떤 DAQ 모듈이 활성화되었는지는 시스템의 패킷 수집 성능과 호환성에 직결됩니다. 예를 들어, Linux 환경에서는 AFPacket이나 NFQ 모듈이 중요하게 사용됩니다.

Snort 빌드 과정에서 확인된 AFPacket, PCAP 등 주요 DAQ 모듈의 생성 여부

위 화면을 통해 AFPacket, BPF, Dump, PCAP 등의 모듈은 성공적으로 빌드(yes)되었으나, Divert와 netmap 모듈은 빌드되지 않았음(no)을 확인할 수 있습니다. 이는 시스템 환경에 따라 특정 가속 모듈의 지원 여부가 달라질 수 있음을 의미합니다.

2. Snort 탐지 규칙(Rule)의 4계층 구조

Snort의 핵심은 "어떤 패킷을 탐지할 것인가"를 정의하는 규칙(Rule)에 있습니다. 모든 Snort 규칙은 체계적인 4단계 계층 구조로 이루어져 있으며, 이를 정확히 이해해야 정교한 탐지 정책을 수립할 수 있습니다.

규칙의 구조는 다음과 같이 구분됩니다:

1. **1단: Action (동작):** 패킷이 규칙에 일치할 때 수행할 동작을 지정합니다. (예: **alert** - 로그 기록, **drop** - 패킷 차단 등)
2. **2단: Header (헤더):** 탐지할 트래픽의 기본 정보를 정의합니다. 프로토콜(TCP, UDP, ICMP 등), 출발지/목적지 IP 주소, 출발지/목적지 포트 번호가 포함됩니다.
3. **3단: Options (옵션):** 가장 상세한 분석 단계로, 패킷 내부의 특정 데이터(content)나 메시지(msg) 등을 검사합니다.
4. **4단: Metadata (메타데이터):** 규칙의 고유 식별 번호인 sid(Signature ID)와 규칙의 수정 버전인 rev(Revision)를 포함합니다.

Snort 룰의 4단 구성 요소인 Action, Header, Options, Metadata 구조

예를 들어, `alert icmp any any → any any (msg:"[Text] ICMP Detected"; content:"msg"; sid:1000001; rev:1;)`라는 규칙은 "모든 IP와 포트 사이의 ICMP 트래픽 중 특정 메시지가 포함된 경우 alert를 발생시키고, 이 규칙의 ID는 1000001이다"라는 의미를 담고 있습니다.

3. 탐지 로그의 종류 및 상세 분석

Snort는 탐지된 이벤트를 저장할 때 사용자의 목적에 따라 다양한 로그 형식을 제공합니다. 실무에서는 탐지된 이벤트의 요약 정보만 빠르게 확인하거나, 사고 분석(Forensics)을 위해 패킷의 모든 세부 정보를 확인하는 두 가지 방식이 주로 사용됩니다.

3.1 alert_fast 옵션 (요약 로그)

`alert_fast`는 탐지된 이벤트의 핵심 정보(타임스탬프, SID, 메시지, 출발지/목적지 IP)만을 한 줄로 요약하여 출력합니다. 대량의 트래픽을 모니터링할 때 시스템 부하를 줄이고 빠르게 상황을 파악하기에 적합합니다.

`alert_fast` 옵션 적용 시 출력되는 ICMP 탐지 이벤트 요약 로그

3.2 alert_full 옵션 (상세 로그)

`alert_full`은 요약 정보 외에도 IP 헤더의 세부 필드(TTL, TOS, Identification, Length, DgmLen 등)와 프로토콜의 상세 상태(Type, Code, Sequence Number 등)를 모두 포함합니다. 이는 특정 공격 패턴의 정밀 분석이 필요할 때 필수적입니다.

`alert_full` 옵션 적용 시 출력되는 ICMP 패킷의 상세 필드 정보

상세 로그를 통해 분석가들은 단순히 "ICMP가 탐지되었다"를 넘어, TTL 값의 변화를 통해 OS 핑거프린팅 시도를 감지하거나, 특정 ID 값을 통해 패킷 번호 여부를 판단할 수 있습니다.

로그 파일 확인 실습: 사용자는 `/var/log/snort` 디렉토리 내에서 생성된 로그 파일을 직접 확인할 수 있습니다.

실제 로그 파일의 내용을 `cat` 명령어로 확인하면, `alert_fast.txt`에는 요약된 정보가, `alert_full.txt`에는 위에서 살펴본 상세한 패킷 레이어 정보가 기록되어 있음을 알 수 있습니다.

4. 실무 운영 효율화: 명령어 단축과 설정 적용

Snort 3 버전부터는 설정 파일로 .lua 파일을 사용하며, 명령어가 다소 복잡해질 수 있습니다. 실무 환경에서는 긴 명령어를 매번 입력하는 번거로움을 줄이기 위해 셸(Shell)의 함수(Function)나 별칭(Alias)을 활용합니다.

실무에서는 `snort -c /usr/local/snort/etc/snort/snort.lua -A alert_full`과 같은 긴 명령어를 입력하는 대신, 이를 함수로 만들어 `snortc`라는 짧은 명령어로 실행할 수 있도록 설정합니다. 이때 주의할 점은 새로운 함수를 정의한 후에는 반드시 `source /etc/profile.d/snort-alias.sh`를 실행한 후에는 Snort++ 3.12.2.0이 정상적으로 로드되어 동작하는 것을 확인할 수 있습니다.

snortc 함수를 정의하여 Snort 실행 명령어를 간소화한 결과

위 실행 과정을 보면, 처음에 `snortc` 명령을 바로 사용했을 때는 "command not found" 오류가 발생하지만, `source /etc/profile.d/snort-alias.sh`를 실행한 후에는 Snort++ 3.12.2.0이 정상적으로 로드되어 동작하는 것을 확인할 수 있습니다.

5. 패킷 캡처를 위한 네트워크 구성 (SPAN)

Snort가 네트워크 트래픽을 제대로 감시하기 위해서는 스위치에서 흐르는 트래픽을 Snort가 연결된 인터페이스로 복제해주는 과정이 필요합니다. 이를 **SPAN(Switched Port Analyzer)** 또는 **Port Mirroring**이라고 합니다.

- **SPAN:** 동일한 스위치 내의 특정 포트 트래픽을 다른 포트로 복제합니다.
- **RSPAN (Remote SPAN):** 서로 다른 스위치 간에 트래픽을 전달하여 원격지에서 모니터링할 수 있게 합니다.
- **ERSPAN (Encapsulated Remote SPAN):** 트래픽을 IP/GRE로 캡슐화하여 레이어 3 네트워크를 통해 전달합니다.

이때 Snort가 설치된 인터페이스는 **Promiscuous Mode(무차별 모드)**로 동작해야 합니다. 일반적인 NIC는 목적지 MAC 주소가 자기 것이 아니면 패킷을 버리지만, 무차별 모드에서는 자신에게 오지 않는 유니캐스트 패킷이라도 모두 수집하여 분석할 수 있습니다. 반면, `tcpdump` 등을 사용할 때 `-p` 옵션을 주면 이 모드가 해제되어 패킷을 드롭하게 되므로 주의가 필요합니다.

실무 및 학습 체크리스트

- **DAQ 확인:** Snort 설치 후 `AFPacket`이나 `PCAP` 같은 핵심 모듈이 정상적으로 빌드되었는지 확인했는가?
- **Rule 검증:** 규칙 작성 시 `sid`를 중복되지 않게 부여했는가? `Action`과 `Header`의 형식이 문법에 맞는가?
- **로그 선택:** 실시간 모니터링 중인가(`fast`)? 아니면 정밀 분석 중인가(`full`)? 상황에 맞는 로그 옵션을 선택했는가?
- **환경 설정:** 셸 함수나 Alias를 적용한 후 `source` 명령어로 현재 세션에 반영했는가?
- **네트워크 구성:** 스위치에서 SPAN 설정이 완료되었으며, Snort 인터페이스가 **Promiscuous Mode**로 동작하고 있는가?

로그 출력 옵션에 따른 탐지 데이터 비교

Snort는 네트워크 트래픽을 실시간으로 분석하여 설정된 룰(Rule)에 부합하는 위협을 탐지합니다. 이때 탐지된 이벤트는 사용자가 지정한 **로그 출력 옵션**에 따라 서로 다른 형식으로 저장됩니다. 실무에서는 시스템의 리소스 상태와 분석 목적(단순 모니터링 vs 상세 포렌식)에 따라 적절한 옵션을 선택하는 것이 매우 중요합니다.

1. Snort 탐지 및 로그 생성의 기초

로그가 생성되기 위해서는 먼저 Snort가 네트워크 패킷을 수집할 수 있는 환경이 갖춰져야 합니다. Snort는 **DAQ(Data Acquisition library)**를 통해 NIC(Network Interface Card)로부터 패킷을 가져옵니다. 빌드 과정에서 `AFPacket`, `BPF`, `PCAP` 등 다양한 DAQ 모듈이 정상적으로 생성되었는지 확인하는 것이 실습의 첫 단계입니다.

패킷 수집이 완료되면 Snort는 정의된 룰에 따라 트래픽을 검사합니다. Snort 룰은 크게 4단계의 계층 구조로 이루어져 있으며, 이 구조를 이해해야 어떤 조건에서 로그가 발생하는지 파악할 수 있습니다.

- **1단계: Action** (패킷 탐지 시 수행할 동작, 예: `alert`, `drop`)
- **2단계: Header** (출발지/목적지 IP, 포트, 프로토콜 정보)
- **3단계: Options** (패킷 내부의 특정 내용 검색, 예: `msg`, `content`)
- **4단계: Metadata** (룰의 고유 ID(`sid`) 및 버전(`rev`) 정보)

Snort 설치 및 빌드 과정에서 DAQ 모듈이 정상적으로 구성되었는지 확인한 결과는 다음과 같습니다.

Build `AFPacket` DAQ module.. : yes 및 `Divert`, `netmap` 모듈의 no 표시

```
code_coverage_enabled: no
code_coverage_cppflags:
code_coverage_cflags:
code_coverage_ldflags:

Build AFPPacket DAQ module.. : yes
Build BPF DAQ module..... : yes
Build Divert DAQ module.... : no
Build Dump DAQ module..... : yes
Build FST DAQ module..... : yes
Build netmap DAQ module.... : no
Build NFQ DAQ module..... : yes
Build PCAP DAQ module..... : yes
Build Savefile DAQ module.. : yes
Build Trace DAQ module..... : yes
Build GWLB DAQ module..... : yes
```

Snort 빌드 시 DAQ 모듈(AFPacket, BPF, PCAP 등)의 생성 및 구성 결과

를 구조를 시각화하여 분석하면 다음과 같은 계층을 가집니다.

1단 Action부터 4단 Metadata까지의 단계별 색상 구분 및 설명

3. 구조: Snort Rule 4단 구조 분석

[Snort 룰 읽는 예시]

```

alert icmp any any → any any ( msg:"[Text] ICMP Detected"; content:"msg"; sid:1000001;
                                rev:1; )
  
```

1단: Action	2단: Header	3단: Options	4단: Metadata
메시지 합치기 동작 (alert, drop 등)	주소 정보 (src Port, dst Port 등)	내용물 검색 (msg, content)	고유 번호 및 버전 (sid, rev)

Snort 룰의 4단계 계층 구조: Action, Header, Options, Metadata

2. 로그 출력 옵션 비교: alert_fast vs alert_full

탐지된 이벤트의 상세 수준을 결정하는 두 가지 대표적인 옵션인 `alert_fast`와 `alert_full`을 비교하면 다음과 같습니다.

구분	alert_fast (요약형)	alert_full (상세형)
주요 특징	최소한의 정보만 기록하여 성능 최적화	패킷의 헤더 필드까지 상세히 기록
포함 정보	타임스탬프, SID, 메시지, 프로토콜, IP 주소	alert_fast 정보 + TTL, TOS, ID, 길이, Type, Code 등
권장 용도	대량의 트래픽 모니터링, 실시간 경보	침해 사고 분석, 정밀 포렌식, 규칙 검증

실제로 ICMP(Ping) 탐지 룰을 적용했을 때, alert_fast 옵션은 이벤트의 발생 여부와 기본적인 통신 흐름을 빠르게 보여줍니다.

타임스탬프와 IP 주소, 탐지 메시지가 간략히 나열된 로그

[illegible]

Snort의 ICMP 탐지 이벤트 요약 로그 출력

만면, `alert_full` 옵션은 동일한 이벤트를 기록하되, 해당 패킷이 어떤 속성을 가지고 있었는지(예: TTL 값, IP 길이 등)를 매우 구체적으로 보여주어 공격의 패턴을 파악하는 데 결정적인 근거를 제공합니다.

TTL, TOS, ID, IpLen 등 상세 필드가 포함된 로그

```
[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.611499 172.16.16.97 -> 172.16.1.2
ICMP TTL:64 TOS:0x0 ID:21976 IpLen:20 DgLen:36
Type:0 Code:0 ID:0 Seq:0 ECHO REPLY

[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.612623 172.16.1.2 -> 172.16.16.97
ICMP TTL:128 TOS:0x0 ID:1 IpLen:20 DgLen:36
Type:8 Code:0 ID:0 Seq:0 ECHO

[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.612630 172.16.16.97 -> 172.16.1.2
ICMP TTL:64 TOS:0x0 ID:21977 IpLen:20 DgLen:36
Type:0 Code:0 ID:0 Seq:0 ECHO REPLY

[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.613655 172.16.1.2 -> 172.16.16.97
ICMP TTL:128 TOS:0x0 ID:1 IpLen:20 DgLen:36
Type:8 Code:0 ID:0 Seq:0 ECHO

[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.614703 172.16.1.2 -> 172.16.16.97
ICMP TTL:128 TOS:0x0 ID:21978 IpLen:20 DgLen:36
Type:0 Code:0 ID:0 Seq:0 ECHO REPLY

[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.614712 172.16.16.97 -> 172.16.1.2
ICMP TTL:64 TOS:0x0 ID:21979 IpLen:20 DgLen:36
Type:0 Code:0 ID:0 Seq:0 ECHO REPLY

[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.615749 172.16.1.2 -> 172.16.16.97
ICMP TTL:128 TOS:0x0 ID:1 IpLen:20 DgLen:36
Type:8 Code:0 ID:0 Seq:0 ECHO

[*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:14:39.615758 172.16.16.97 -> 172.16.1.2
ICMP TTL:64 TOS:0x0 ID:21980 IpLen:20 DgLen:36
Type:0 Code:0 ID:0 Seq:0 ECHO REPLY
```

ICMP 탐지 시 출력되는 상세 패킷 필드 정보

3. 실습을 통한 로그 데이터 검증

실습에서는 Snort를 실행하여 생성된 로그 파일이 지정된 경로에 올바르게 저장되었는지 확인하고, 각 파일의 내용을 직접 검증하였습니다. 로그 파일은 일반적으로 `/var/log/snort/` 디렉토리 내에 생성됩니다.

먼저, 로그 저장 디렉토리로 이동하여 생성된 파일 목록을 확인합니다.

`cd snort` 및 `cat alert_fast.txt` 명령어를 통한 로그 출력

```
[root@localhost ~]# cd log
[root@localhost ~]# ls
icmp      dnf.log.2      maillog-20260624  secure-20260624
macanda   dnf.rpm.log    mariadb          snort
audit     dnf.log        messages-20260520  spooler
btcp      hawkkey.log    messages-20260520  spooler-20260520
btcp-20260501  hawkkey.log-20260520  messages-20260527  spooler-20260527
dnf       hawkkey.log-20260527  messages-20260609  spooler-20260609
cron      hawkkey.log-20260609  messages-20260624  spooler-20260624
cron-20260520  hawkkey.log-20260624  http              host
cron-20260527  http              private           tallylog
cron-20260609  http              testlog          xferlog
cron-20260624  http              xferlog          xferlog
dnf.log      dnf.log-20260520  dnf.log-20260520  dnf.log-20260602
dnf.log-1    dnf.log-20260527  dnf.log-20260527  dnf.log-20260609
dnf.log-2    dnf.log-20260609  dnf.log-20260609  dnf.log-20260624
[root@localhost ~]# cd snort
[root@localhost ~]# ls
alert_fast.txt
[root@localhost ~]# cat alert_fast.txt
[06/30-11:20:22.073991][*] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.074025 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.16.97 -> 172.16.1.2
06/30-11:20:22.074068 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.0.1 -> 172.16.16.97
06/30-11:20:22.074138 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.07457 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.16.97 -> 172.16.1.2
06/30-11:20:22.074954 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.074972 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.16.97 -> 172.16.1.2
06/30-11:20:22.078379 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.078397 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.16.97 -> 172.16.1.2
06/30-11:20:22.078460 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.16.97 -> 172.16.1.2
06/30-11:20:22.078478 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.080849 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.080866 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.16.97 -> 172.16.1.2
06/30-11:20:22.081806 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.081844 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.16.97 -> 172.16.1.2
06/30-11:20:22.082028 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
**] [Priority: 0] [ICMP] 172.16.1.2 -> 172.16.16.97
06/30-11:20:22.082226 [**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [
```

이 절의 실습 결과를 확인하는 화면

이후 `cat` 명령어를 사용하여 각 로그의 실제 데이터 형식을 비교하였습니다. `alert_full.txt` 파일에서는 단순한 통신 기록을 넘어 IP 헤더의 상세 필드 값들을 확인할 수 있습니다.

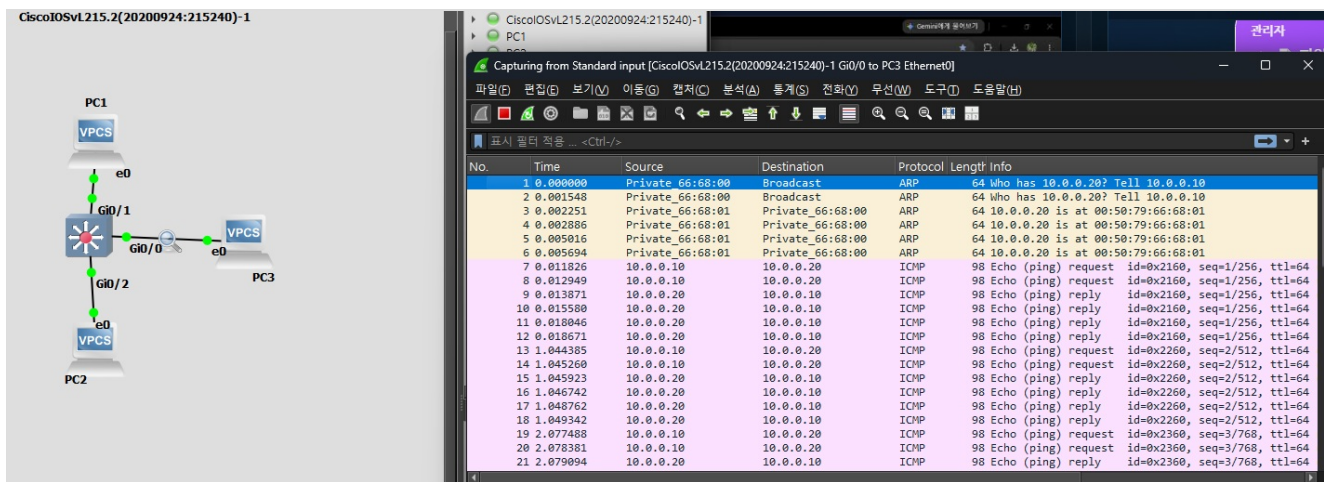
`cat alert_full.txt` 명령어로 확인된 TTL, TOS, ID 등의 상세 필드

```
[root@localhost snort]# ls
alert_fast.txt alert_full.txt
[root@localhost snort]# cat alert_full.txt
[**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:23:24.897796 172.16.1.2 -> 172.16.16.97
ICMP TTL:128 TOS:0x0 ID:1 IpLen:20 DgmLen:36
Type:8 Code:0 ID:0 Seq:0 ECHO
[**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:23:24.897826 172.16.16.97 -> 172.16.1.2
ICMP TTL:64 TOS:0x0 ID:3043 IpLen:20 DgmLen:36
Type:0 Code:0 ID:0 Seq:0 ECHO REPLY
[**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:23:24.899564 172.16.1.2 -> 172.16.16.97
ICMP TTL:128 TOS:0x0 ID:1 IpLen:20 DgmLen:36
Type:8 Code:0 ID:0 Seq:0 ECHO
[**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:23:24.899582 172.16.16.97 -> 172.16.1.2
ICMP TTL:64 TOS:0x0 ID:3044 IpLen:20 DgmLen:36
Type:0 Code:0 ID:0 Seq:0 ECHO REPLY
[**] [1:1000001:1] "[Text Mode] ICMP Secret Message Detected" [**]
[Priority: 0]
06/30-11:23:24.900873 172.16.1.2 -> 172.16.16.97
ICMP TTL:128 TOS:0x0 ID:1 IpLen:20 DgmLen:36
Type:8 Code:0 ID:0 Seq:0 ECHO
```

alert_full.txt 옵션 적용 시 출력되는 상세 패킷 정보

이러한 로그 데이터는 네트워크 토폴로지 상에서 실제로 발생하는 패킷 캡처 결과와 일치해야 합니다. 구성된 네트워크 환경에서 인터페이스를 통해 캡처된 ICMP 패킷 흐름은 다음과 같습니다.

VPCS-스위치 연결 구조 및 ARP/ICMP 프로토콜 패킷 캡처 화면



네트워크 토폴로지 구성 및 인터페이스를 통한 패킷 캡처 결과

4. 효율적인 운영을 위한 관리 팁

실무에서는 Snort 실행 시 매번 긴 설정 파일 경로와 옵션을 입력하는 것이 번거롭고 실수를 유발할 수 있습니다. 이를 방지하기 위해 Alias(별칭) 기능을 활용하여 명령어를 단축하는 방법을 사용합니다.

예를 들어, /etc/profile.d/에 쉘 스크립트를 작성하여 snortc라는 짧은 명령어로 설정 파일(.lua)을 로드하고 특정 인터페이스에서 동작하도록 설정할 수 있습니다. 설정 후에는 다음과 같이 간단한 명령으로 Snort를 즉시 실행할 수 있습니다.

export SNORT_LUA 설정 후 source 명령을 통한 설정 파일 로드 성공

```
[root@localhost snort]# cat > /etc/profile.d/snort-alias.sh << 'EOF'
export SNORT_LUA="/usr/local/snort/etc/snort/snort.lua"

snortc() {
    snort -c "$SNORT_LUA" "$@"
}
EOF
[root@localhost snort]# snortc -i ens160 -A alert_full
-bash: snortc: 명령어를 찾을 수 없음
[root@localhost snort]# source /etc/profile.d/snort-alias.sh
[root@localhost snort]# snortc -i ens160 -A alert_full
o")~ Snort++ 3.12.2.0
-----
Loading /usr/local/snort/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
alert_full
alert_fast
ips
classifications
references
file_inspect
ftp_data
port_scan
output
active
alerts
daq
decode
host_cache
host_tracker
hosts
```

snortc 명령어를 생성하여 Snort 실행 명령을 간소화한 설정 및 결과

5. 학습 요약 및 점검 사항

이번 장에서는 Snort의 탐지 메커니즘과 로그 옵션에 따른 데이터 차이를 학습했습니다. 실무 및 시험 관점에서의 핵심 포인트는 다음과 같습니다.

- **로그 선택 기준:** 트래픽이 매우 많은 환경에서 성능 저하를 막으려면 `alert_fast`를, 정밀한 분석이 필요할 때는 `alert_full`을 사용하십시오.
- **데이터 완전성 확인:** `alert_full` 로그에서 TTL, TOS, Type/Code 값이 비정상적이라면, 이는 단순한 통신이 아닌 패킷 조작(Crafted Packet)을 통한 공격 가능성을 시사합니다.
- **점검 순서:**
 1. DAQ 모듈이 정상적으로 빌드되었는가?
 2. 작성한 룰의 4계층 구조(Action~Metadata)가 문법에 맞는가?
 3. 로그 파일이 `/var/log/snort/` 경로에 적절한 이름으로 생성되었는가?
 4. 출력된 로그의 상세 정보가 실제 네트워크 패킷의 특성을 반영하고 있는가?

운영 명령어 최적화 및 네트워크 미러링 실습

Snort와 같은 네트워크 침입 탐지 시스템(NIDS)을 실무에서 운영할 때는 매우 긴 설정 파일 경로와 다양한 옵션을 매번 입력해야 하는 번거로움이 있습니다. 효율적인 운영을 위해서는 명령어 단축(Aliasing)을 통해 오타를 줄이고 운영 속도를 높이는 과정이 필수적입니다. 또한, 탐지된 위협을 정확히 분석하기 위해서는 네트워크 트래픽을 적절히 복제하여 Snort로 전달하는 미러링 기술에 대한 이해가 선행되어야 합니다.

1. Snort 운영 환경 및 룰 구조 이해

효율적인 명령어 최적화에 앞서, Snort가 정상적으로 동작하기 위한 구성 요소와 탐지 규칙(Rule)의 기본 구조를 점검해야 합니다. Snort는 데이터를 수집하는 DAQ(Data Acquisition) 모듈이 핵심이며, 빌드 과정에서 필요한 모듈(AFPacket, BPF, PCAP 등)이 정상적으로 생성되었는지 확인해야 합니다.

DAQ 모듈 설치 시 AFPacket이나 PCAP과 같은 주요 모듈이 'yes'로 표시되어야 정상적인 패킷 수집이 가능합니다. 만약 특정 모듈이 'no'로 나타난다면 해당 라이브러리 누락을 의심해야 합니다.

확인된 DAQ 모듈들은 Snort가 네트워크 인터페이스로부터 패킷을 읽어들이는 하위 계층 역할을 수행합니다.

Snort가 트래픽을 탐지하는 기준은 룰(Rule)에 정의되어 있습니다. Snort 룰은 크게 **Action(동작)**, **Header(주소 정보)**, **Options(내용 검색)**, **Metadata(고유 번호 및 버전)**의 4단계 구조로 이루어집니다. 이 구조를 정확히 알아야 탐지된 로그가 어떤 규칙에 의해 발생했는지 판단할 수 있습니다.

각 단계는 룰의 성격(예: ICMP 탐지, 특정 문자열 포함 여부 등)을 결정하는 핵심 요소입니다.

2. 운영 명령어 최적화: Shell Alias 활용

Snort 3 버전부터는 기존의 `snort.conf` 대신 LUA 기반의 `snort.lua` 설정 파일을 주로 사용합니다. 실행 시마다 `snort -c /usr/local/snort/etc/snort/snort.lua -A alert_full -i ens160`와 같이 긴 명령어를 입력하는 것은 비효율적이며 실수할 가능성이 높습니다. 이를 해결하기 위해 셸 함수(Shell Function)를 활용한 명령어 단축 실습을 진행합니다.

운영 환경에서는 `/etc/profile.d/` 디렉토리에 별도의 스크립트를 생성하여, 모든 사용자가 로그인할 때 자동으로 단축 명령어가 로드되도록 설정합니다. 이번 실습에서는 `snortc()`라는 함수를 정의하여 설정 파일 경로와 인터페이스 등을 간소화하였습니다.

먼저 `/etc/profile.d/snort-alias.sh` 파일을 생성하고, `SNORT_LUA` 환경 변수와 함께 실행 시 인자(`$@`)를 전달받는 함수를 작성합니다. 설정 후에는 `source` 명령어를 통해 변경 사항을 즉시 반영해야 합니다.

명령어 실행 결과, `snortc` 입력만으로 설정 파일 로딩과 함께 Snort++ 엔진이 정상적으로 구동됨을 확인할 수 있습니다. 만약 `command not found` 오류가 발생한다면 `source` 명령을 통해 스크립트를 적용했는지 반드시 확인해야 합니다.

3. 탐지 결과 로그 분석 및 저장 옵션

Snort는 탐지된 이벤트를 저장할 때 로그의 상세도에 따라 두 가지 주요 옵션을 제공합니다. `alert_fast`는 이벤트 발생 시간, SID, 메시지, IP 주소 등 핵심 요약 정보만을 빠르게 출력하며, `alert_full`은 패킷의 프로토콜 세부 필드까지 모두 포함하여 심층 분석을 가능하게 합니다.

`alert_fast` 옵션을 적용하면 시스템 부하를 최소화하면서 대량의 이벤트를 요약된 형태로 확인할 수 있습니다. 이는 주로 실시간 모니터링 및 전체적인 이벤트 흐름 파악에 사용됩니다.

요약 로그에서는 어떤 IP 간에 통신이 발생했는지, 어떤 규칙(SID)에 걸렸는지 빠르게 식별할 수 있습니다.

반면, 침해 사고가 의심되어 정밀 분석이 필요한 경우에는 `alert_full` 옵션을 사용합니다. 이 모드에서는 ICMP 패킷의 TTL(Time To Live), TOS(Type of Service), ID, IP Length, 데이터그램 길이(DgmLen) 등 프로토콜 계층의 상세 필드 값이 모두 기록됩니다.

상세 로그를 통해 공격자가 사용한 패킷의 특이값(예: 비정상적인 TTL이나 ID 값)을 찾아내어 공격 유형을 정교하게 분류할 수 있습니다.

실제 운영 중에는 로그 파일이 지정된 경로(예: `/var/log/snort/`)에 생성되었는지 확인해야 합니다. `ls` 명령어를 통해 `alert_fast.txt`와 `alert_full.txt` 파일이 정상적으로 생성되었는지 검증합니다.

로그 파일의 존재 여부는 Snort가 정상적으로 탐지 이벤트를 디스크에 기록하고 있음을 의미합니다.

생성된 `alert_full.txt` 파일을 `cat` 명령어로 확인하면, 앞서 살펴본 상세 패킷 정보들이 누적되어 저장되어 있음을 알 수 있습니다.

이 파일은 추후 포렌식 분석 시 중요한 증거 자료로 활용됩니다.

4. 네트워크 미러링(SPAN)을 통한 패킷 수집

Snort와 같은 센서가 네트워크 트래픽을 탐지하려면, 스위치를 통과하는 트래픽의 복사본을 센서로 전달해주어야 합니다. 이를 위해 스위치에서 제공하는 미러링 기술을 사용합니다. 주요 방식은 다음과 같습니다.

- **SPAN (Switched Port Analyzer):** 동일한 스위치 내의 특정 포트 트래픽을 다른 포트로 복제합니다.
- **RSPAN (Remote SPAN):** 트래픽을 다른 스위치에 있는 목적지 포트로 전달하기 위해 VLAN을 이용합니다.
- **ERSPAN (Encapsulated Remote SPAN):** IP/GRE 캡슐화를 통해 레이어 3 네트워크를 넘어 원격지까지 트래픽을 전달합니다.

Cisco 스위치 환경에서는 `monitor session` 명령어를 통해 미러링을 설정합니다. 예를 들어, **Gi0/1** 포트의 트래픽(rx/tx 모두)을 **Gi0/0** 포트로 복제하도록 설정하면, **Gi0/0**에 연결된 Snort 센서가 전체 트래픽을 관찰할 수 있습니다.

이때 주의할 점은 Snort나 tcpdump와 같은 캡처 도구가 **Promiscuous Mode(무차별 모드)**로 동작해야 한다는 것입니다. 기본적으로 NIC는 자신의 MAC 주소로 향하는 유니캐스트 트래픽만 수신하지만, 무차별 모드가 활성화되어야 미러링된 다른 대상의 트래픽을 버리지 않고 수집할 수 있습니다. 만약 `tcpdump -p` 옵션을 사용하여 이 모드를 비활성화하면 미러링된 트래픽을 놓치게 됩니다.

실제 구성된 토폴로지에서 스위치 포트를 통해 미러링된 트래픽을 Wireshark로 분석하면, ARP 요청 및 ICMP Echo Request/Reply와 같은 다양한 프로토콜이 정상적으로 캡처되는 것을 확인할 수 있습니다.

토폴로지상의 PC 간 통신이 스위치 미러링 포트를 통해 센서로 정확히 전달되고 있음을 검증할 수 있습니다.

실무 및 시험 대비 점검 리스트

- **명령어 최적화:** source 명령어를 통해 `/etc/profile.d`에 등록한 스크립트를 현재 세션에 즉시 반영했는가?
- **로그 분석:** 단순 모니터링 시에는 `alert_fast`를, 상세 침해 분석 시에는 `alert_full`을 선택하여 운영 효율성을 고려했는가?
- **패킷 수집:** 미러링 설정 후 캡처 도구가 **Promiscuous Mode**로 동작하고 있는가? (tcpdump 사용 시 `-p` 옵션 주의)
- **환경 점검:** Snort 실행 전 DAQ 모듈이 운영 환경(AFPacket, PCAP 등)에 맞게 `yes` 상태로 빌드되었는가?