

네트워크 침입탐지시스템

기본 구조와 원리

○ 문서 정보

- 작성자: 양봉열 (xeraph@nchovy.com)
- 작성일: 2009년 9월 12일
- 라이선스: [크리에이티브 커먼즈 저작자표시-비영리-변경금지 2.0 대한민국](#)

○ 목차

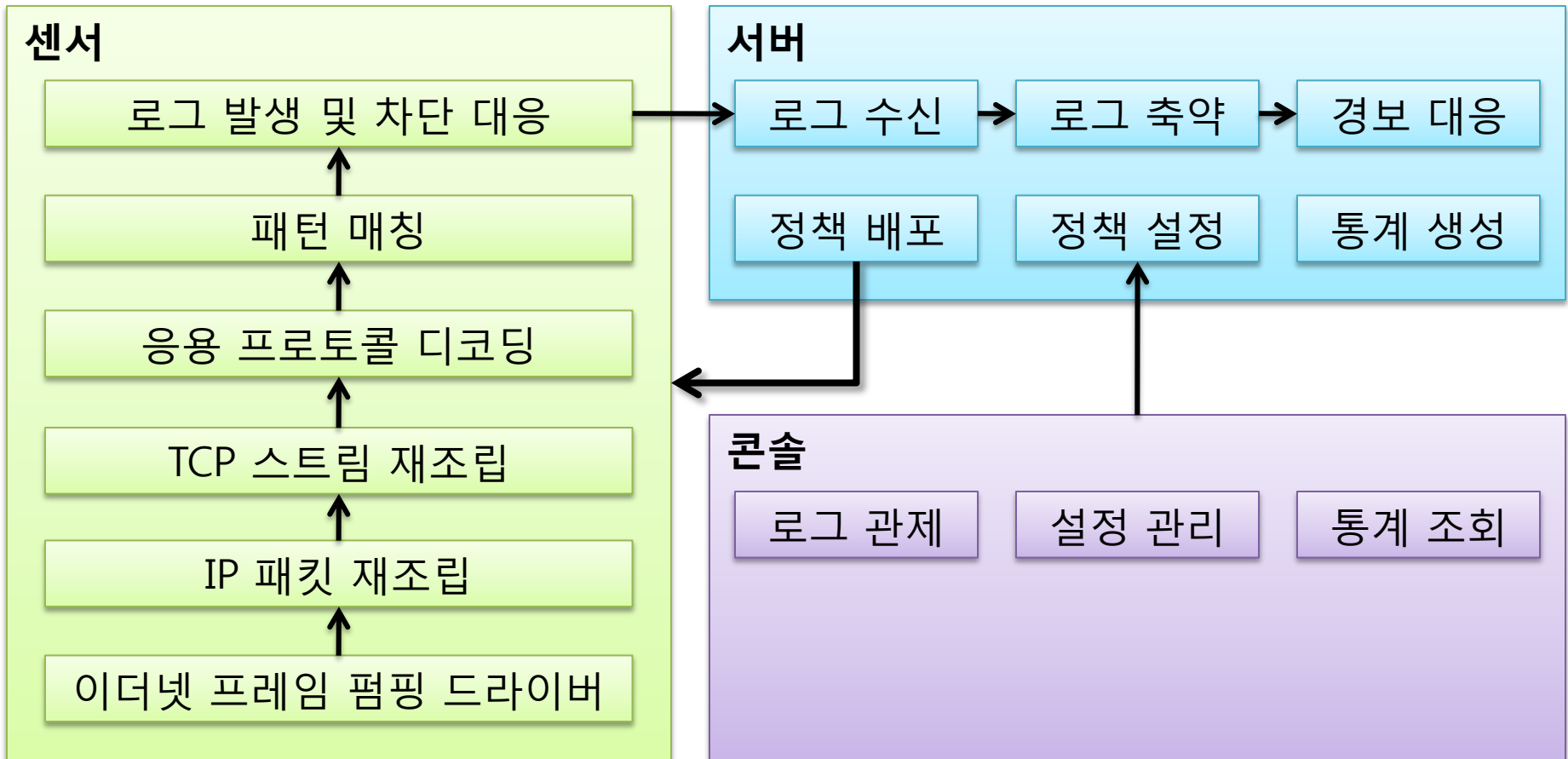
- IDS의 정의와 역사
- NIDS 아키텍처
- NIDS 배치
- 이더넷 트래픽 펌핑
- IP 단편화와 재조립
- TCP 재조립
- 응용 프로토콜 디코딩
- 패턴 매칭
- 룰 및 탐지 회피
- 네트워크 이상 탐지
- 패킷 덤프 및 로깅

○ 정의

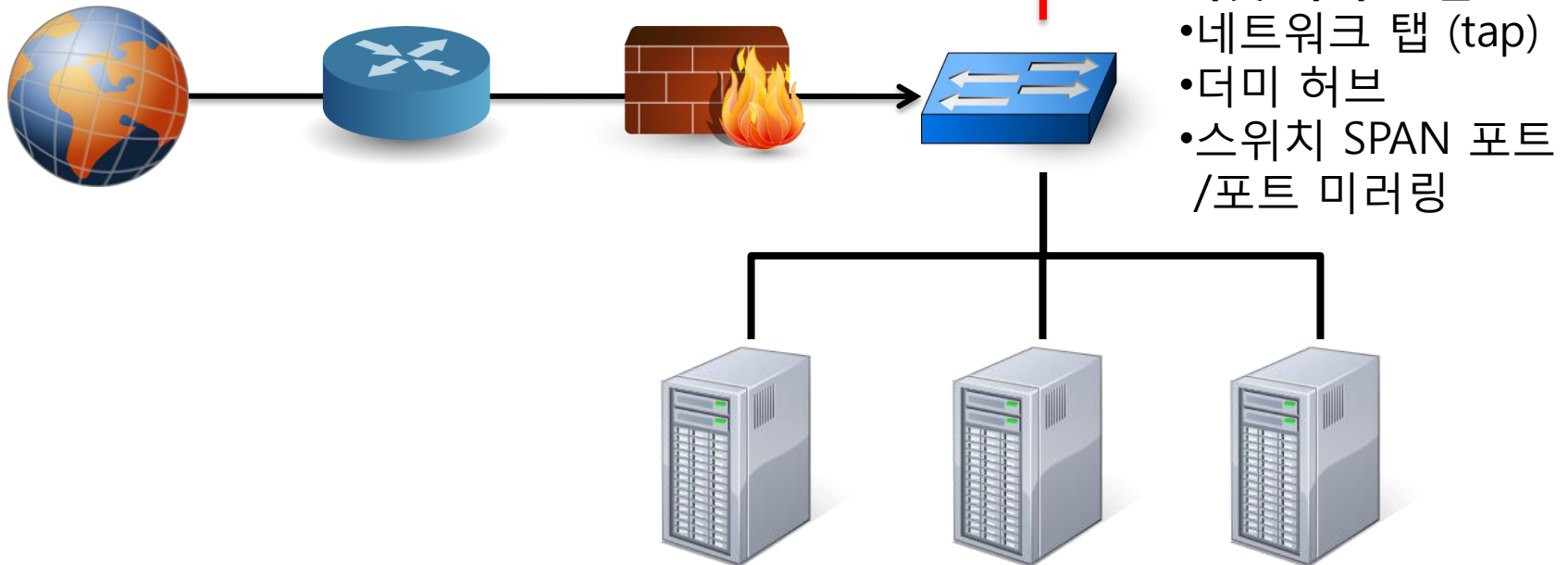
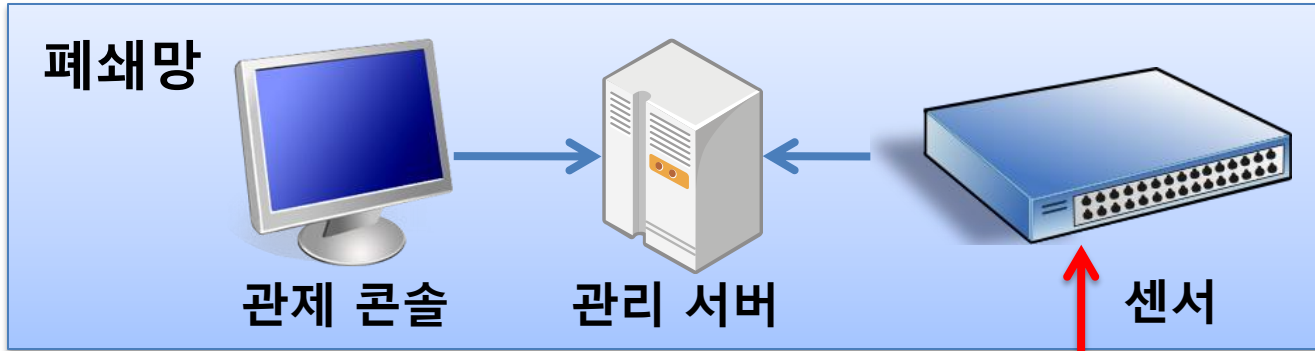
- 컴퓨터 시스템을 대상으로 인가되지 않은 데이터 접근, 조작, 서비스 방해 등의 침입 시도를 찾아내도록 설계된 소프트웨어나 하드웨어
- 배포 위치에 따른 구분
 - 네트워크 기반 침입탐지시스템
 - 호스트 기반 침입탐지시스템

○ 역사

- 1980년, James P. Anderson이 감사 로그를 이용한 오용 탐지 기법 제시
- 1990년대 초부터 상업적 침입탐지시스템의 개발이 시작됨
- 1997년부터 침입탐지 시장이 본격적으로 성장함 (ISS, Cisco 등)
- 2000년 시큐어소프트 수호신 IDS, 인젠 네오와처, 윈스텍넷 스나이퍼 IDS 출시
- 2003년 윈스텍넷 스나이퍼 IPS 출시 등 IPS 및 UTM으로 발전하면서 IDS 시장 쇠퇴
- <http://www.securityfocus.com/infocus/1514>



NIDS 배치

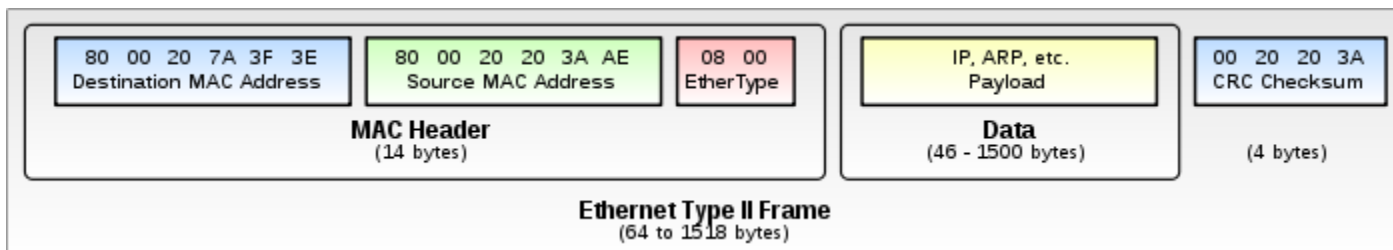


○ 정의

- LAN에서 사용되는 프레임 기반 컴퓨터 네트워킹 기술의 집합
- IEEE 802.3로 표준화 됨: 물리 계층과 데이터 링크 계층의 MAC 부분을 정의

○ 프레임 유형

- Ethernet II (DIX frame): 16비트 길이의 EtherType으로 프로토콜 번호 지정
- IEEE 802.3: EtherType을 길이로 대체하고 802.2 LLC 헤더가 뒤에 추가됨
 - LLC 헤더 뒤에 SNAP 헤더가 붙는 경우도 있음
 - 프레임 유형 구분은 관례에 따라 EtherType 값의 범위를 이용함
 - 64~1522의 경우 802.3 Ethernet
 - 1536(0x0600) 이상인 경우 Ethernet II의 프로토콜 식별자로 해석



○ 정의

– 이더넷 네트워크 인터페이스 장치로부터 이더넷 프레임들을 센서 입력으로 올리는 작업

○ 구현 방식

– 커널에서 제공하는 네트워킹 인터페이스를 이용하는 방식

- 기존에 벤더에서 제공하는 드라이버를 그대로 이용할 수 있으므로 높은 호환성 획득
- 커널에서 불필요한 작업이 추가로 수행되므로 효율성 저하
- 리눅스의 경우 libpcap 및 libipq가 널리 사용됨
- 윈도우의 경우 NDIS 프로토콜 혹은 인터미디어트 드라이버를 구현해야 함

– 네트워크 인터페이스 장치를 직접 제어하는 방식

- 특정 NIC용 디바이스 드라이버를 자체 제작하거나 커스터마이징
- 장치의 성능을 최대한으로 뽑아낼 수 있으며 오버헤드 역시 최소화 가능
- 윈도우의 경우 NDIS 미니포트 드라이버를 구현해야 함
- 하드웨어 일체형 보안 장비의 경우 드라이버 및 운영체제까지 자체 개발함

○ 패킷 손실 (Packet Loss)

- 네트워크 인터페이스 장치는 수신한 프레임들을 DMA로 링 버퍼에 저장함
- 링 버퍼가 가득 차면 이후 장치에서 수신하는 프레임들은 그냥 버려짐
- 프레임이 손실되면 해당 세션은 디코딩 실패로 인해 제대로 탐지할 수 없게 됨
- 같은 대역폭에서도 프레임 길이에 따라 처리량이 달라짐
 - 1Gbps의 경우,
 - 64바이트: 195만 3125 fps
 - 128바이트: 97만 6562 fps
 - 256바이트: 48만 8281 fps
 - 512바이트: 24만 4140 fps
 - 길이가 짧을수록 오버헤드가 증가하므로 손실 가능성이 높아짐

○ IP 헤더

4 Bits	8 Bits	16 Bits	24 Bits
Version	IHL	Type of Service	Total Length
Identification		Flags	Fragment Offset
Time to Live	Protocol	Header Checksum	
Source IP Address			
Destination IP Address			
IP Options			Padding
Data			

- **Version:** IP 버전 (이하 버전 4 기준)
- **IHL:** IP 헤더 길이 (4바이트 단위)
- **Type of Service:** QoS (대부분 0)
- **Total Length:** 전체 길이 (헤더 포함)
- **Identification:** 패킷 식별자
단편화된 패킷들은 식별자가 동일함
- **Flags:**
More Fragment (001)
Don't Fragment (010)
- **Fragment Offset:** 단편의 상대적 위치
- **Time to Live:** 1 hop 지날 때마다 차감
- **Protocol:** 페이로드의 프로토콜 번호
[IANA 프로토콜 번호 할당 문서](#) 참조
(ICMP 1, IGMP 2, TCP 6, UDP 17)
- **Header Checksum:** 헤더 무결성 확인
- **Source IP Address:** 출발지 주소
- **Destination IP Address:** 목적지 주소
- **IP Options:** 선택적 기능
[IANA 옵션 번호 할당 문서](#) 참조

○ IP 헤더 체크섬 계산 ([RFC791](#))

- 헤더를 2바이트 단위로 끊어서 1의 보수 합을 구한 다음 1의 보수를 취함.
 - 1의 보수 합은 캐리 발생 시 LSB에 더함

○ 효율적인 체크섬 계산 ([RFC1071](#))

- $[A, B] + [C, D] + \dots + [Y, Z]$
 - '+'는 1의 보수 합을 의미
- 교환, 결합 가능
 - 계산을 임의로 묶거나 순서를 바꿔서 수행해도 결과는 바뀌지 않음
 - $([A, B] + [C, D] + \dots + [J, 0]) + ([0, K] + \dots + [Y, Z])$
- 바이트 순서 독립성
 - $[B, A] + [D, C] + \dots + [Z, Y]$ 로 나온 결과도 바이트 순서만 바뀔 뿐 동일
- 병렬 연산
 - $[A, B] + [C, D] + [E, F] + [G, H]$ 을 $[A, B, C, D] + [E, F, G, H]$ 후 절반으로 접어서 다시 '+' 해도 동일하게 계산된다.

○ 체크섬 계산 코드

```
while( count > 1 ) {  
    /* This is the inner loop */  
    sum += * (unsigned short) addr++;  
    count -= 2;  
}
```

```
/* Add left-over byte, if any */  
if( count > 0 )  
    sum += * (unsigned char *) addr;
```

```
/* Fold 32-bit sum to 16 bits */  
while (sum >> 16)  
    sum = (sum & 0xffff) + (sum >> 16);
```

```
checksum = ~sum;
```

캐리 발생 시 다시 더해야 함

○ 체크섬 계산 연습

```
Internet Protocol, Src: 72.14.203.100 (72.14.203.100), Dst: 192.168.0.7 (192.168.0.7)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 52
  Identification: 0x7dc0 (32192)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 47
  Protocol: TCP (0x06)
  Header checksum: 0x39e2 [correct]
  Source: 72.14.203.100 (72.14.203.100)
  Destination: 192.168.0.7 (192.168.0.7)
0000  00 16 ea b3 a8 0e 00 08 9f 4e d6 b4 08 00 45 00  .....N...E.
0010  00 34 7d c0 00 00 2f 06 39 e2 48 0e cb 64 c0 a8  .4}.../.9.H..d..
0020  00 07 00 50 0f 92 44 16 5f 8d dc 4b 40 a6 80 12  ...P..D. _..K@...
```

○ IP 단편의 재조립 과정

– 경우의 수

- 빈 공간에 완전히 딱 맞아떨어지는 경우
- 앞이나 뒤의 일부분만 채우는 경우
- 중간에 끼어들어서 남은 공간을 둘로 쪼개는 경우

– RFC815

- 패킷이 도착하면 hole descriptor list 조회
- 빈 공간이 메워지는 경우 해당 hole descriptor를 list에서 제거
- hole descriptor list가 완전히 비워지는 경우 재조립 처리 가능

– 용어 정의

- hole.first: 빈 공간의 첫번째 바이트 번호
- hole.last: 빈 공간의 마지막 바이트 번호
- hole descriptor: hole.first와 hole.last의 쌍
- hole descriptor list: hole descriptor의 목록

○ IP 단편의 재조립 과정

– RFC815 알고리즘

1. hole descriptor list에서 hole descriptor 선택, 더 이상 없으면 8번 단계로
2. fragment.first가 hole.last보다 크면 1번 단계로
3. fragment.last가 hole.first보다 작으면 1번 단계로
4. hole descriptor list에서 현재 항목 삭제
5. fragment.first가 hole.first보다 크면,
[hole.first, fragment.first - 1] hole descriptor를 새로 생성
6. fragment.last가 hole.last보다 작고 More Fragment가 참이면,
[fragment.last + 1, hole.last] hole descriptor를 새로 생성
7. 1번 단계로
8. 재조립 완료

○ IP 단편화 공격

– Ping of Death

- IP 패킷은 헤더 포함하여 최대 길이 65535로 제한됨 (RFC791)
- Fragment Offset 필드는 13비트이고 8바이트 단위이므로 65528까지 표현 가능
- 마지막 Fragment가 최대 길이를 넘어가는 경우 재조립할 때 크래시 발생 가능
- ping -l 65510 호스트주소

– Teardrop

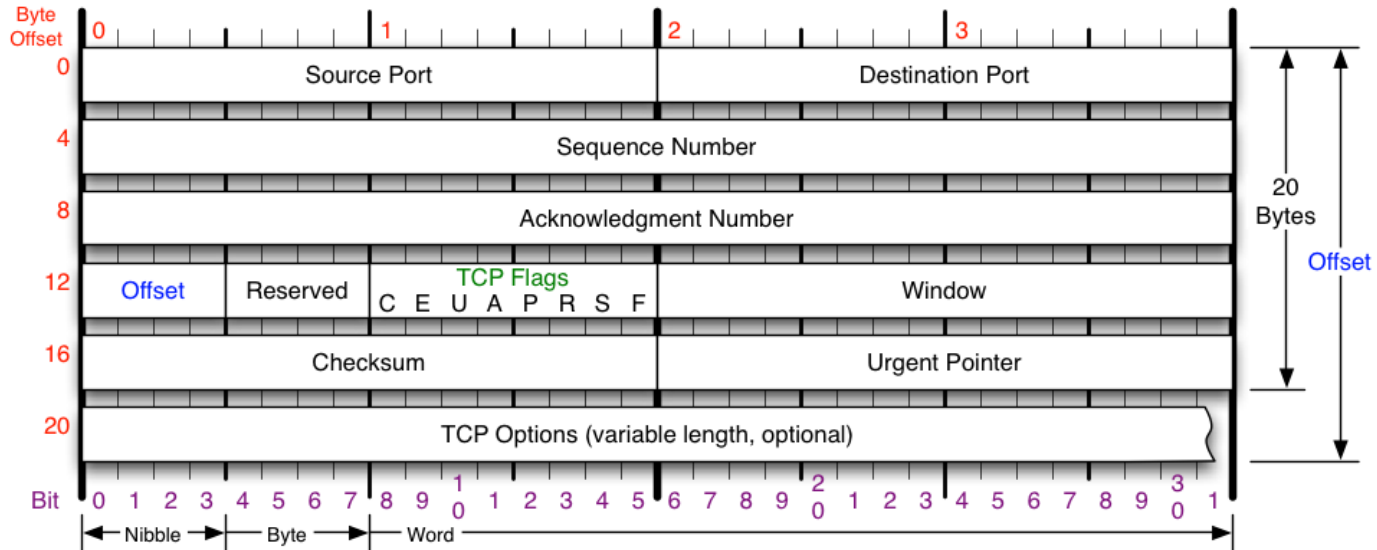
- 데이터 영역이 겹치는 서로 모순되는 Fragment를 전송
 - MF = 1 Fragment Offset = 1400
 - MF = 0 Fragment Offset = 1300

○ 테스트

– fragrouter

- 모든 패킷을 지정된 크기로 분할하여 전송
- <http://monkey.org/~dugsong/fragroute/>

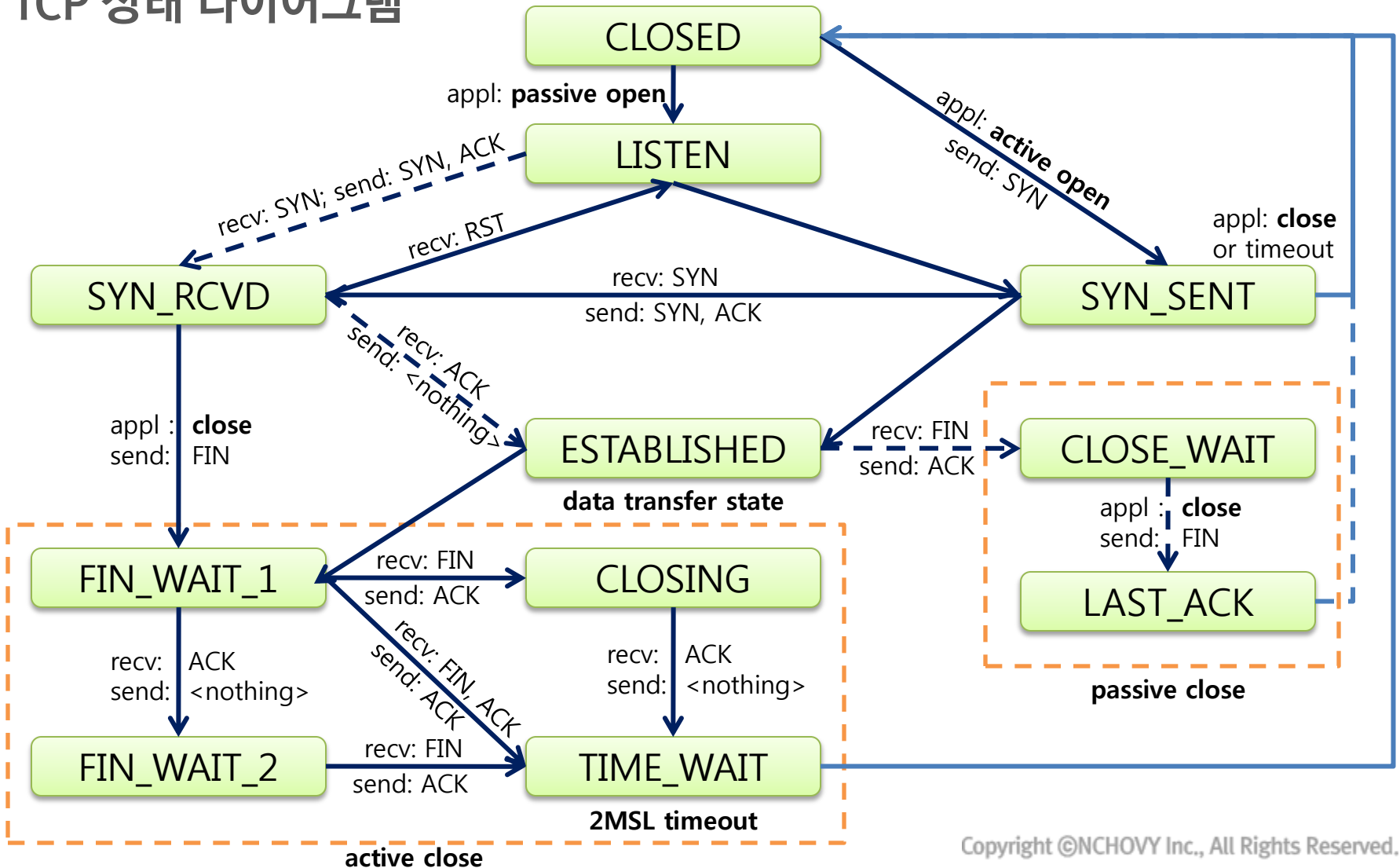
○ TCP 헤더



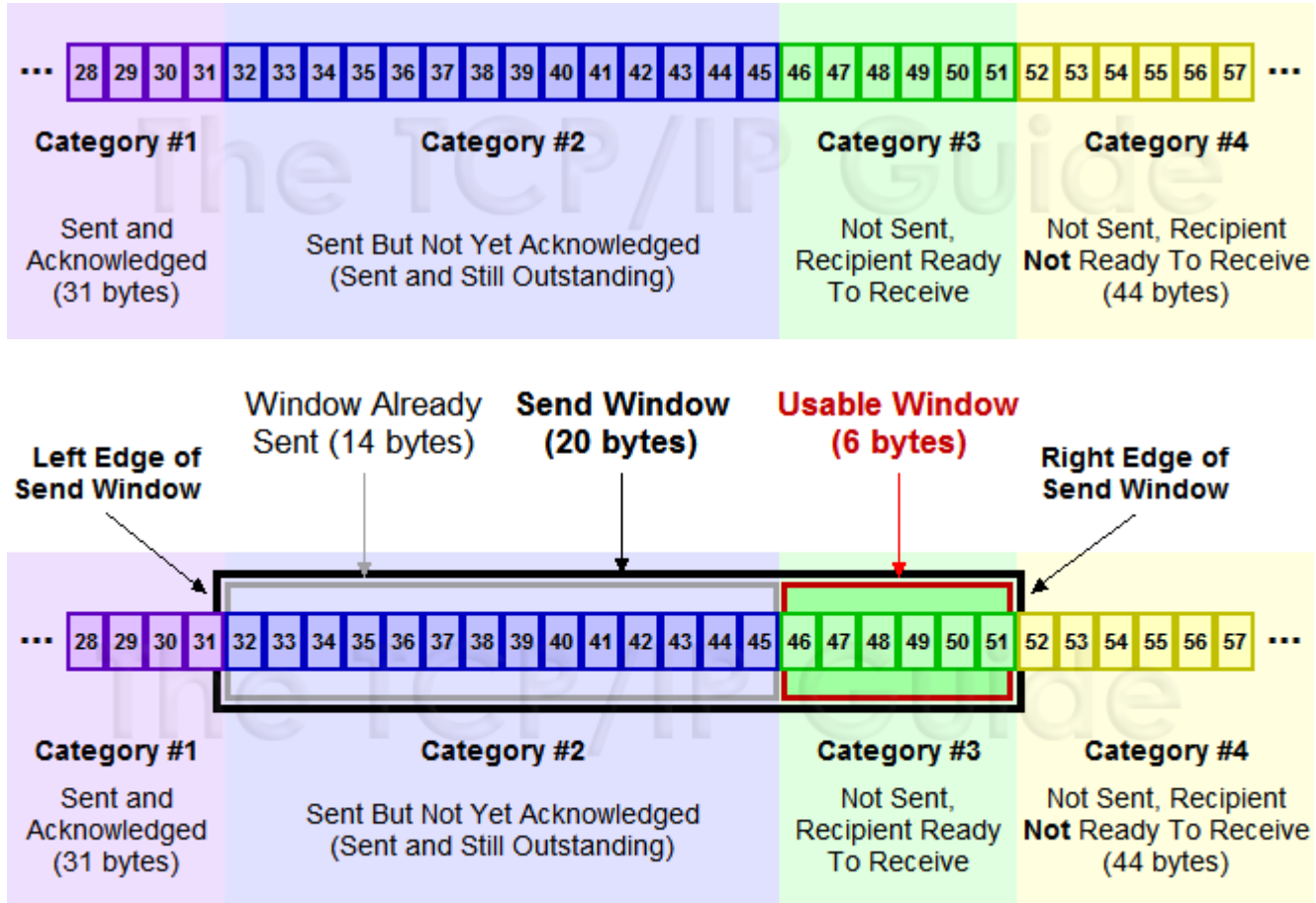
TCP Flags	Congestion Notification	TCP Options	Offset																											
C E U A P R S F	ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.	0 End of Options List 1 No Operation (NOP, Pad) 2 Maximum segment size 3 Window Scale 4 Selective ACK ok 8 Timestamp	Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.																											
<p>Congestion Window</p> <p>C 0x80 Reduced (CWR)</p> <p>E 0x40 ECN Echo (ECE)</p> <p>U 0x20 Urgent</p> <p>A 0x10 Ack</p> <p>P 0x08 Push</p> <p>R 0x04 Reset</p> <p>S 0x02 Syn</p> <p>F 0x01 Fin</p>	<table border="1"> <thead> <tr> <th>Packet State</th> <th>DSB</th> <th>ECN bits</th> </tr> </thead> <tbody> <tr> <td>Syn</td> <td>00</td> <td>11</td> </tr> <tr> <td>Syn-Ack</td> <td>00</td> <td>01</td> </tr> <tr> <td>Ack</td> <td>01</td> <td>00</td> </tr> <tr> <td>No Congestion</td> <td>01</td> <td>00</td> </tr> <tr> <td>No Congestion</td> <td>10</td> <td>00</td> </tr> <tr> <td>Congestion</td> <td>11</td> <td>00</td> </tr> <tr> <td>Receiver Response</td> <td>11</td> <td>01</td> </tr> <tr> <td>Sender Response</td> <td>11</td> <td>11</td> </tr> </tbody> </table>	Packet State	DSB	ECN bits	Syn	00	11	Syn-Ack	00	01	Ack	01	00	No Congestion	01	00	No Congestion	10	00	Congestion	11	00	Receiver Response	11	01	Sender Response	11	11	<p>Checksum</p> <p>Checksum of entire TCP segment and pseudo header (parts of IP header)</p>	<p>RFC 793</p> <p>Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.</p>
Packet State	DSB	ECN bits																												
Syn	00	11																												
Syn-Ack	00	01																												
Ack	01	00																												
No Congestion	01	00																												
No Congestion	10	00																												
Congestion	11	00																												
Receiver Response	11	01																												
Sender Response	11	11																												

전송 제어 프로토콜 (TCP)

TCP 상태 다이어그램



○ 슬라이딩 윈도우



○ 주요 대상 프로토콜

- HTTP, FTP, TFTP, DNS, WINS, NETBIOS, LDAP, SNMP, SMTP, IMAP, POP3
- TDS, TNS, SSH, TELNET, RLOGIN, RSH, RWHO, FINGER, IRC, MSNP, ICQ ...
- 응용 프로토콜 분석에 기반한 룰 문법 제공을 통하여 정교한 룰 제작 가능
 - 예1) HTTP Cookie 헤더를 통한 SQL 인젝션 시도 탐지
 - 예2) DNS 요청 패킷 타입이 NULL인 경우
 - 예3) SMB NetPathCanonicalize 함수 호출 매개변수에 \..\.. 패턴이 있는 경우

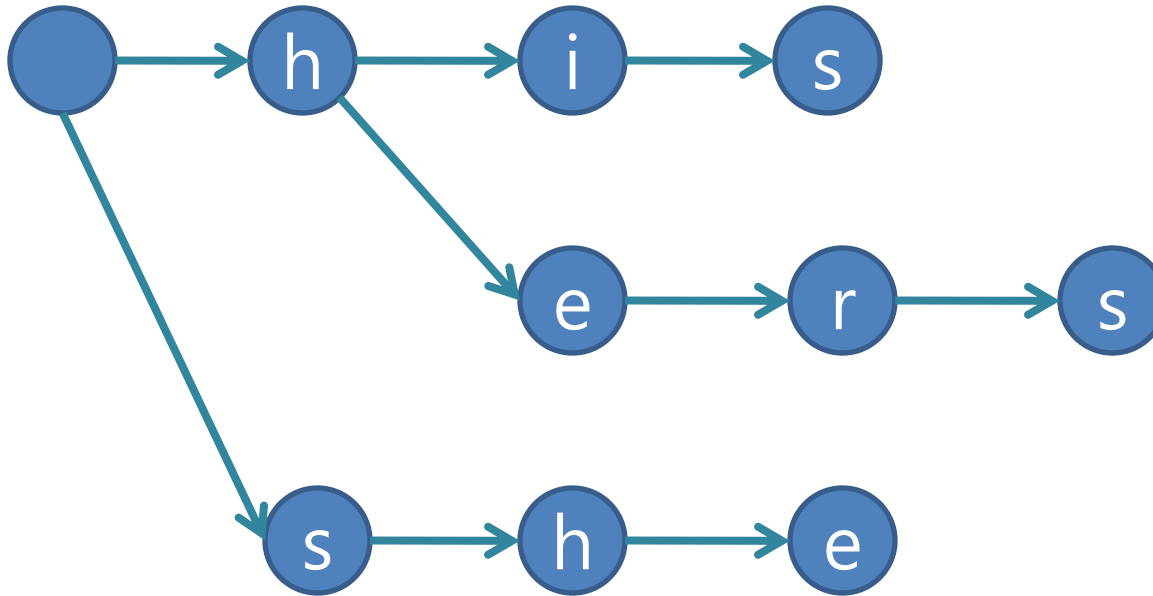
○ 파일 분석

- 네트워크 프로토콜 뿐 아니라 전송되는 첨부 파일 등 검사 필요 (DLP 등)
- 오피스 문서를 이용한 취약점 공격 증가
- 취약점 예) [CVE-2008-4841](#), [CVE-2008-4837](#), [CVE-2008-4026](#)

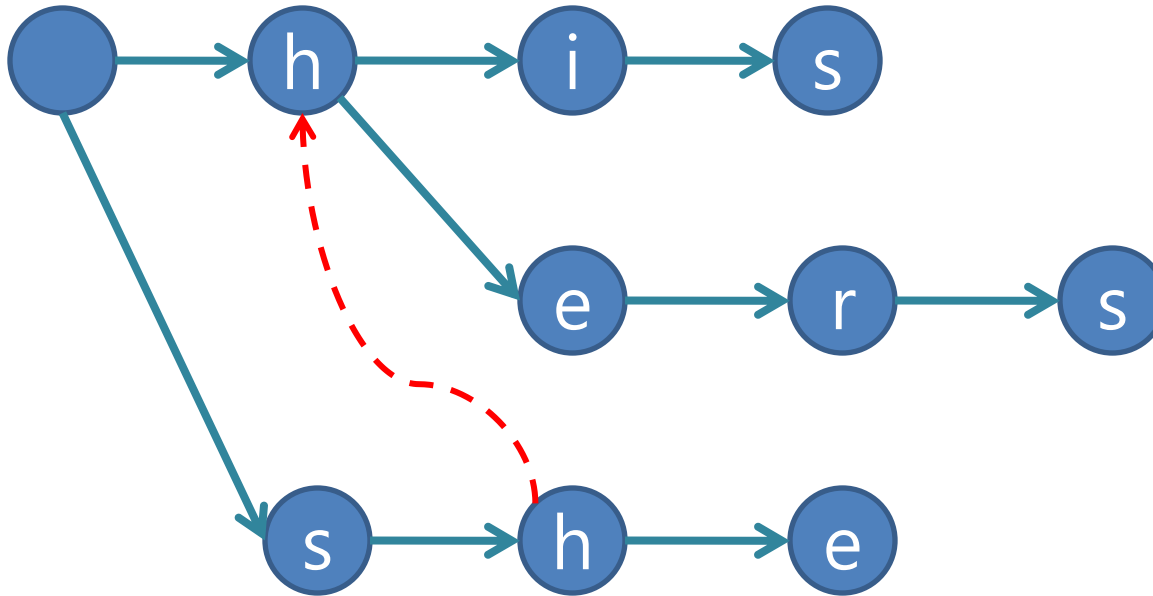
○ 비정상 행위 탐지

- 표준 규약에 위배되거나 공격일 가능성이 높은 메시지를 탐지
 - 비정상적으로 길이가 긴 경우 버퍼 오버플로우 공격 의심
 - 통상적으로 사용되지 않는 값이거나 범위를 벗어나는 값
 - 헤더 정보와 페이로드의 불일치나 무결성 오류 등

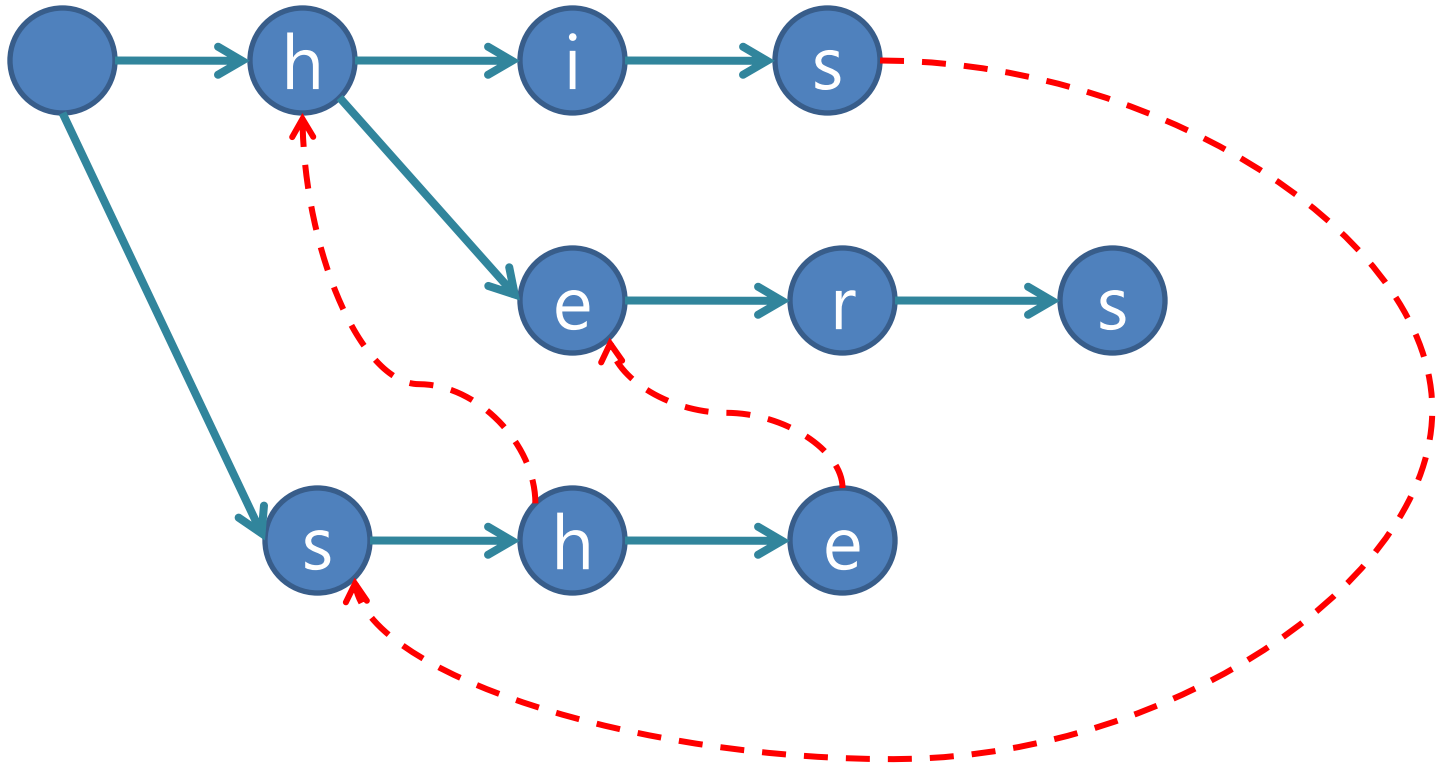
○ Aho-Corasick: Trie 빌드



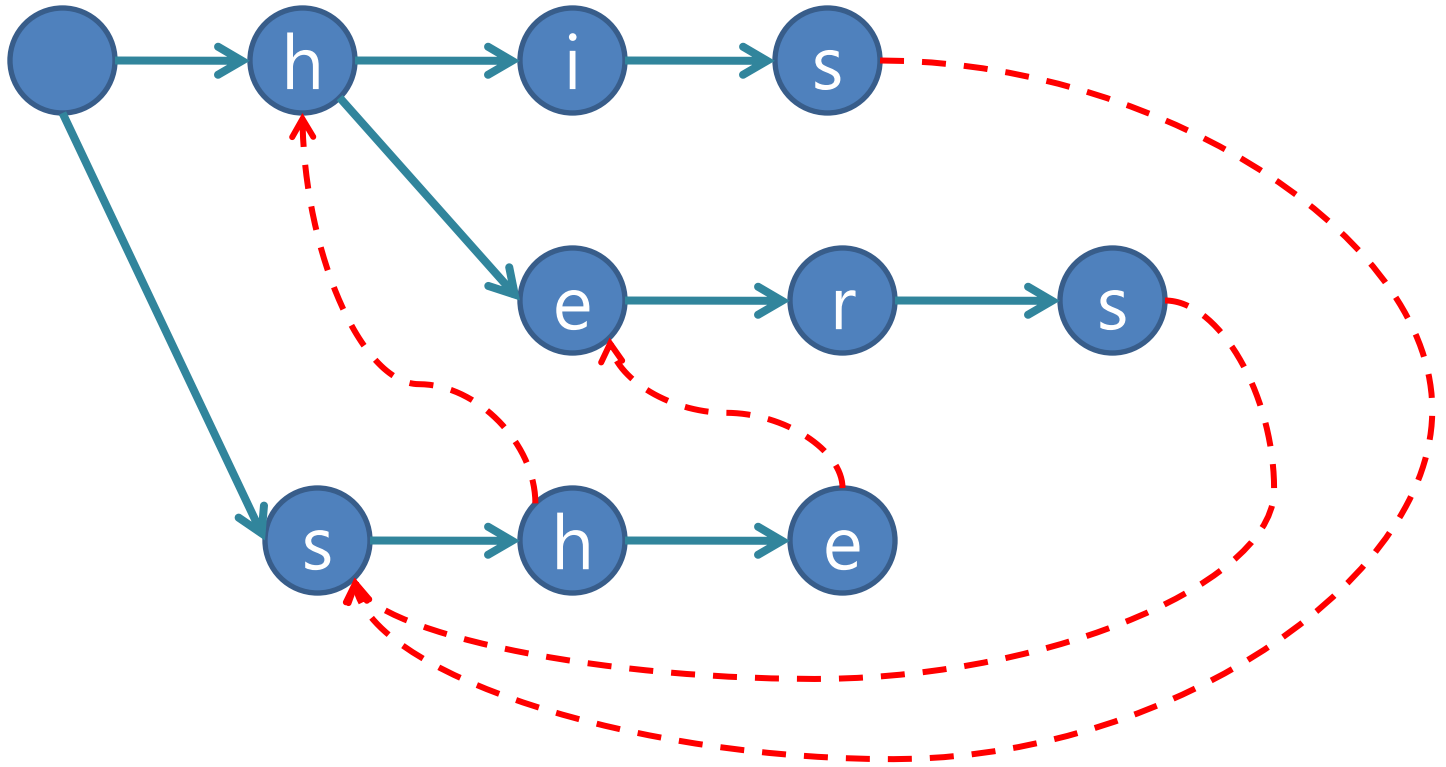
○ Aho-Corasick: Failure Link 연결 1



○ Aho-Corasick: Failure Link 연결 2



○ Aho-Corasick: Failure Link 연결 3



○ PCAP 파일 포맷

– 사실상 패킷 덤프 표준으로 사용되고 있는 파일 포맷

- tcpdump, tcpdump, wireshark 등

– Global Header

- magic number: 바이트오더 판별 용도 (0xa1b2c3d4)
- version_major, version_minor: 대부분 2.4
- thiszone: GMT와 지역 시간대 차이를 초 단위로 표시
- snaplen: 캡처 시 최대 패킷 길이 제한
- network: 데이터링크 계층 타입 (예를 들면 이더넷 1, 토큰링 6)

– Packet Header

- ts_sec, ts_usec
- incl_len: 캡처된 패킷 데이터 길이
- orig_len: 원래 패킷 데이터 길이 (snaplen으로 인해 잘릴 수 있음)



○ Syslog ([RFC5424](#), [RFC5426](#))

– UDP 514번 포트 사용

– 전송 크기

- 수신자는 2048 바이트 이상의 메시지를 받을 수 있어야 함 (최소 480 바이트)
- 송신자는 IP단편화를 피할 수 있도록 가급적 최소 MTU 값 이하로 보내야 함
 - 패킷 손실 발생 시 로그를 재전송할 방법이 없음

– 헤더

- RFC와 달리 실제 환경에서는 헤더에 PRI만 들어가는 경우가 대부분
- PRI는 <> 안에 최대 3자리 숫자로 표시되고 Facility와 Severity를 표현함
 - PRI 값 = Facility x 8 + Severity
 - Facility 값은 local0 ~ local7 (16~23)을 주로 사용자 정의하여 사용
 - Severity는 8단계 표시 (Emergency 0 부터 Debug 7까지)

– 본문

- 정해진 양식은 없으며 가급적 UTF-8 인코딩 사용 (이 때 BOM 마크 필수)

– 스노트 Syslog 예시

- <129>snort[24858]: [122:17:0] (portscan) UDP Portscan[Priority: 3]: {PROTO:255} 220.45.142.139 -> 10.10.0.10

○ SNMP Trap

– 용도

- 에이전트에서 이벤트 발생 시 매니저에게 트랩 메시지를 전송하여 통보

– 전송 및 포맷

- UDP 162번 포트 사용
- ASN.1로 정의되며 주로 BER이나 DER로 데이터 인코딩
 - ASN.1은 데이터 구성을 표현하는 문법이고

– 트랩 타입

- 일반 유형(Generic)과 사용자 정의(Specific)로 나누어짐
 - Cold Start(0), Warm Start(1), Link Down(2), Link Up(3), Authentication Failure(4), EGP Neighbor Loss(5), Enterprise Specific(6)
 - Generic Trap 타입이 6인 경우 Specific Trap 타입을 이용함

○ SNMP Trap v1

```
PDU ::=
  CHOICE {
    get-request
      GetRequest-PDU,

    get-next-request
      GetNextRequest-PDU,

    get-response
      GetResponse-PDU,

    set-request
      SetRequest-PDU,

    trap
      Trap-PDU
  }
```

```
IMPLICIT SEQUENCE {
  enterprise OBJECT IDENTIFIER,
  agent-addr NetworkAddress,
  generic-trap
    INTEGER {
      coldStart(0),
      warmStart(1),
      linkDown(2),
      linkUp(3),
      authenticationFailure(4),
      egpNeighborLoss(5),
      enterpriseSpecific(6)
    },
  specific-trap INTEGER,
  time-stamp TimeTicks,
  variable-bindings VarBindList
}
```

○ SNMP Trap v1 패킷 예제

```

30 29 02 01 00 04 06 6E
63 68 6F 76 79 A4 1C 06
07 2B 06 01 04 01 96 26
40 04 0A 00 01 7B 02 01
06 02 01 00 43 03 13 65
3C 30 00
    
```

BER Type-Length-Value 구조

30 29: 시퀀스 타입, 길이 41 (0x29)

02 01 00: INT 타입, 길이 1, 값 1

04 06 6E 63 68 6F 76 79:

OctetString 타입, 길이 6, 문자열 NCHOVY

A4 1C: Trap-PDU 타입, 길이 28 (0x1C)

06 07 2B 06 01 04 01 96 26

- ① OID 타입, 길이 7, 첫 2개 숫자는 $X * 40 + Y$ 로 인코딩
- ② 큰 숫자 표현 시 MSB를 MORE DATA 신호 비트로 사용
- ③ 96, 26 => 10010110, 00100110 => 101100100110
- ④ 즉, .1.3.6.1.4.1.2854
- ⑤ 길이에서도 큰 숫자 표현 시 동일한 기법 사용

40 04 0A 00 01 7B: NetworkAddress 타입

43 03 13 65 3C: TimeTicks 타입 (1/100초 단위)

30 00: 시퀀스 타입 (Variable Bindings 부분)