

「 주제 」

네트워크 보안 강화를 위한 정밀한 패킷 필터링 시스템

보안 프로젝트 - 함준형

「 순서 」

목차

1

패킷 필터링을 해야하는 이유

2

DPI, XDP, DPDK, SDN

3

다양한 공격으로부터 서버를 보호하는 방법

4

마무리



「 패킷 필터링을 해야하는 이유 」

패킷 필터링 왜 해야하는 걸까

보안 강화

악성 코드나 해킹 시도와 같은 외부의 위협으로부터 네트워크를 보호를 해야합니다.

트래픽 관리

필요 없는 패킷을 필터링하여 네트워크 트래픽을 줄이고 네트워크 성능을 향상 시킬 수 있습니다.

데이터 무결성

특정 유형의 패킷이 들어오거나 나가는 것을 방지하여 데이터 무결성을 유지하는데 도움이 됩니다.

「 DPI, XDP, DPDK, SDN 」

패킷을 필터링하기 위해 사용되는 기술들

DPI 패킷의 페이로드까지 검사하여 네트워크 보안을 적용하는 기술

XDP 커널 내부에서 고성능 네트워크 처리를 가능하게 하는 프레임워크

DPDK 고속 패킷 처리를 가능하게 하는 라이브러리와 드라이버 모음

SDN 네트워크 관리를 소프트웨어를 통해 제어하는 접근법

「 다양한 공격으로부터 서버를 보호하는 방법 」

서버를 보호하는 다양한 방법

해외망 차단

IP 차단

포트 차단

패킷 차단

「 계획 」

테스트 환경 구축

1. "고가용성 네트워크 인프라 구축을 위한 BGP 프로토콜 활용 연구" 를 통한 인프라 구축
2. DPI, XDP, DPDK, SDN 등 다양한 패킷 필터링 예제 참고 후 패킷 필터링 소프트웨어 개발
3. Web Stresser를 통한 DDoS 공격 테스트
4. Grafana(k8s Network Monitoring) 를 통한 모니터링

「마무리」

일정순서

1

11.24 아카마이 PoC & 미팅

2

11.27 U+, 호스트웨이 미팅

3

11.29 소프트웨어 적용 및 테스트

4

11.30 모니터링 및 보고서 작성

「실습」

허니팟 구축

XDP 기반 UDP Flooding 소스코드 작성

XDP(eXpress Data Path)는 커널에서 네트워크 패킷을 최적화하여 처리하는 프레임 워크이다.

네트워크 드라이브 수준에서 패킷을 처리하여, 커널 스택을 통과하는 오버헤드를 최소화하고 패킷 처리 성능을 향상시킨다.

패킷 분류: XDP는 BPF(Berkeley Packet Filter) 기반의 사용자 정의 프로그램을 사용하여 패킷을 분류함.

패킷 처리: XDP는 분류된 패킷을 XDP_PASS, XDP_DROP, XDP_TX, XDP_REDIRECT와 같은 방법으로 처리함.

XDP_PASS: 정상 패킷을 커널 스택으로 전달

XDP_DROP: 악성 패킷 드랍

XDP_TX: 패킷을 같은 네트워크 인터페이스로 재전송(패킷 미머링)

XDP_REDIRECT: 패킷을 다른 네트워크 인터페이스로 전송(패킷 라우팅)

```
#include <arpa/inet.h> // 추가된 헤더
#include <linux/bpf.h>
#include <linux/if_ether.h>
#include <linux/ip.h>
#include <linux/udp.h> // 추가된 헤더
#include <bpf/bpf_helpers.h>

SEC("xdp")
int block_udp(struct xdp_md *ctx) {
    void *data_end = (void *)(long)ctx->data_end;
    void *data = (void *)(long)ctx->data;

    struct ethhdr *eth = data;
    struct iphdr *ip;
    struct udphdr *udp;

    if ((void *)(eth + 1) > data_end) // 수정된 부분
        return XDP_PASS;

    if (eth->h_proto != htons(ETH_P_IP))
        return XDP_PASS;

    ip = (struct iphdr *)(eth + 1);
    if ((void *)(ip + 1) > data_end) // 수정된 부분
        return XDP_PASS;

    if (ip->protocol != IPPROTO_UDP)
        return XDP_PASS;

    udp = (struct udphdr *)(ip + 1);
    if ((void *)(udp + 1) > data_end) // 수정된 부분
        return XDP_PASS;

    if (udp->dest != htons(80))
        return XDP_PASS;

    return XDP_DROP;
}

char _license[] SEC("license") = "GPL";
```


「실습」

장비 구축

악성 패킷 차단 장비 개발

DPI(Deep Packet Inspection)는 패킷의 페이로드까지 검사함.
WAF나 UTM 장비에서 많이 사용되는 기술임.

XDP(eXpress Data Path)는 네트워크 드라이브 수준에서 패킷을
처리하는 기술임.

BPF(Berkeley Packet Filter)기반의 사용자 정의 프로그램을 사
용해 특정 패턴의 패킷을 처리할 수 있음.

DPDK(Data Plane Development Kit)는 네트워크 패킷을 빠르게
처리하기 위한 라이브러리와 드라이버 모음임.

XDP와 DPDK를 이용한 IPS 장비나 고성능 DDoS 보안 장비를 만
들 수 있음.

가비아 계열사 엑스게이트, 아카마이 도움을 받아 실 서비스 환경에
도입할 수 있을 수준으로 개발알 예정



「실습」

장비 구성도



악성 패킷 필터 방법

Suricata, Zeek(Bro)를 통해 DPI 기능을 이용.

XDP는 BPF 언어로 작성 후 clang 컴파일러를 통해 컴파일.

```
bpftool prog load xpd.o /sys/fs/bpf/xdp
```

```
ip link set dev enp1s0 xdp obj /sys/fs/bpf/xdp
```

를 통해 모든 패킷을 1번 이더넷에서 받아 XDP를 통해 처리를 하고
정상 패킷을 2번 이더넷으로 REDIRECTION.

DPI, DPDK도 한 이더넷에서 모든 패킷을 받아 세밀하게 패킷을 처
리를 하고 정상 패킷만을 하단 서버에 배포를 하도록 설정.

「실습」

Suricata Rules 설정

무차별 대입 공격(Brute Force Attack) 방지

TCP 22번 포트로 SYN Flag 기준 60초동안 동일한 IP로부터 5개 이상의 패킷이 감지되면 차단하는 Rule.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Possible SSH Brute Force Attack"; flags: S+;  
threshold: type both, track by_src, count 5, seconds 60; sid:10001; rev:1;)
```

「실습」

Zeek Rules 설정

SQL Injection 방지

OR, AND, SELECT, UNION 키워드가 포함되면 SQL Injection 공격이라고 판단하고 차단하는 Rule.

키워드 방식으로 필터링을 하게 되면 오탐이 많아지기에 추후에는 Rule 수정이 필요함.

```
module SQL_Injection;

export {
  redef enum Notice::Type += {
    SQL_Injection_Attempt
  };
}

event http_request(c: connection, method: string, version: string, request_uri: string) {
  if ( /[\\s\\S]*[';\"\\s](OR|AND|SELECT|UNION)[\\s\\S]*[';\"\\s]/ in request_uri ) {
    NOTICE([$note=SQL_Injection_Attempt,
      $msg=fmt("SQL Injection attempt detected from %s", c$id$orig_h),
      $conn=c,
      $identifier=fmt("%s", c$id$orig_h)]);
  }
}
```

「실습」

DPDK 설정

SYN Flooding 공격 완화

IP 위변조(Spoofing), TCP 세그먼트 재조립, 동시성 처리 등 추후 소스코드를 보완할 점이 많음.

```
#define MAX_PACKET_SIZE 4096
#define MAX_SYN_RATE 10000

int main()
{
    int s = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
    char packet[MAX_PACKET_SIZE];
    struct iphdr *iph = (struct iphdr *) packet;
    struct tcphdr *tcph = (struct tcphdr *) (packet + iph->ihl*4);

    memset(packet, 0, MAX_PACKET_SIZE);
    int syn_rate = 0;
    char *src_ip = "0.0.0.0";

    while(1)
    {
        size_t bytes = recvfrom(s, packet, sizeof(packet), 0, NULL, NULL);
        if (bytes > 0)
        {
            inet_ntop(AF_INET, &(iph->saddr), src_ip, INET_ADDRSTRLEN);
            if (iph->protocol == 6 && tcph->syn == 1 && tcph->ack == 0) // Check if it's TCP SYN
            {
                syn_rate++;
                if (syn_rate >= MAX_SYN_RATE)
                {
                    printf("TCP SYN flooding detected from IP: %s", src_ip);
                    syn_rate = 0;
                    if (syn_rate >= MAX_SYN_RATE)
                    {
                        printf("TCP SYN flooding detected from IP: %s", src_ip);
                        syn_rate = 0;

                        char cmd[100];
                        snprintf(cmd, sizeof(cmd), "iptables -A INPUT -s %s -j DROP", src_ip);
                        system(cmd);
                    }
                }
            }
        }
    }

    return 0;
}
```

「실습」

공격 테스트

UDP Flooding

간단한 UDP Flooding 차단 Rule 설정



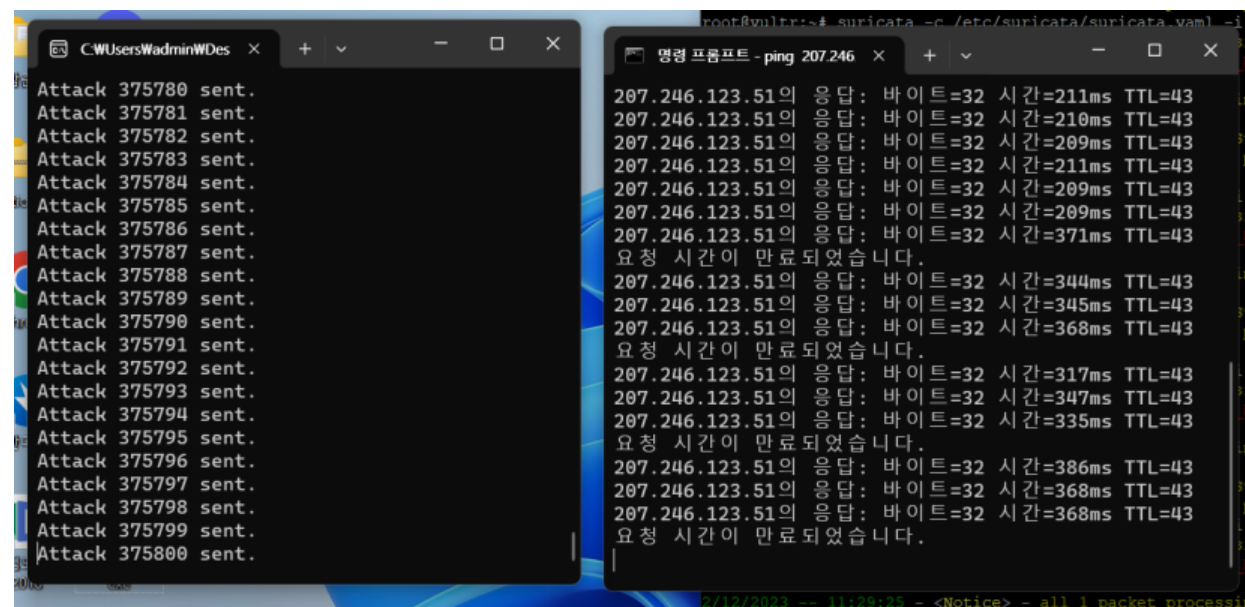
```
alert udp any any -> $HOME_NET any (msg:"Potential UDP Flood"; threshold: type both, track by_src, count 10000, seconds 1; sid:1000001; rev:1;)
```

「실습」

공격 테스트

UDP Flooding

서버에 UDP Flooding 공격 진행



```
root@vultr:~# suricata -c /etc/suricata/suricata.yaml -i e
C:\Users\Wadmin\WDes x + - - - x
Attack 375780 sent.
Attack 375781 sent.
Attack 375782 sent.
Attack 375783 sent.
Attack 375784 sent.
Attack 375785 sent.
Attack 375786 sent.
Attack 375787 sent.
Attack 375788 sent.
Attack 375789 sent.
Attack 375790 sent.
Attack 375791 sent.
Attack 375792 sent.
Attack 375793 sent.
Attack 375794 sent.
Attack 375795 sent.
Attack 375796 sent.
Attack 375797 sent.
Attack 375798 sent.
Attack 375799 sent.
Attack 375800 sent.

명령 프롬프트 - ping 207.246 x + - - - x
207.246.123.51의 응답: 바이트=32 시간=211ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=210ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=209ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=211ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=209ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=209ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=371ms TTL=43
요청 시간이 만료되었습니다.
207.246.123.51의 응답: 바이트=32 시간=344ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=345ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=368ms TTL=43
요청 시간이 만료되었습니다.
207.246.123.51의 응답: 바이트=32 시간=317ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=347ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=335ms TTL=43
요청 시간이 만료되었습니다.
207.246.123.51의 응답: 바이트=32 시간=386ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=368ms TTL=43
207.246.123.51의 응답: 바이트=32 시간=368ms TTL=43
요청 시간이 만료되었습니다.
2/12/2023 -- 11:26:25 - <Notice> - all i packet processing
```

「실습」

공격 테스트

UDP Flooding

UDP Flooding 공격 모니터링

No.	Time	Source	Destination	Protocol	Length	Info
1992...	89.195669	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.195694	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.195776	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.195811	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.195882	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.195910	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.195965	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.195987	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.196040	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.196063	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.196115	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.196137	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.196194	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.196217	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.196285	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18
1992...	89.196314	192.168.0.17	207.246.123.51	UDP	42	50658 → 22 Len=0
1992...	89.196376	192.168.0.17	207.246.123.51	UDP	60	50658 → 22 Len=18

Time to Live: 128

Protocol: UDP (17)

Header Checksum: 0x0000 [validation disabled]

[Header checksum status: Unverified]

Source Address: 192.168.0.17

Destination Address: 207.246.123.51

▼ User Datagram Protocol, Src Port: 50658, Dst Port: 22

Source Port: 50658

Destination Port: 22

Length: 26

Checksum: 0x0c0f [unverified]

[Checksum Status: Unverified]

[Stream index: 0]

> [Timestamps]

UDP payload (18 bytes)

▼ Data (18 bytes)

Data: 334245216f64793653414d50424521646969

[Length: 18]

0000 70 5d cc ab ee 10 dc 41 a9 f4 5a 7e 08 00 45 00 p].....A ..Z...E.
0010 00 2e 3e 30 00 00 80 11 00 00 c0 a8 00 11 cf f6 .>0.....
0020 7b 33 c5 e2 00 16 00 1a 0c 0f 33 42 45 21 6f 64 {3.....3BE!oc
0030 79 36 53 41 4d 50 42 45 21 64 69 69 y6SAMPBE !dii

wireshark_Wi-Fi0WUBF2.pcapng || 패킷 수: 1992748 · 표시됨: 1992748(100.0%) · 누락됨: 0(0.0%) || 프로파일: Default

「실습」

공격 테스트

UDP Flooding

TCP + ICMP + UDP 모든 패킷들 중 약 5.5% 패킷이 Drop.
간단한 Rule 설정이므로 대부분 악성 패킷이 서버에 유입되는 것으로 확인.

세밀한 Rule 설정을 통해 악성 패킷을 구분하고 Drop 해야함.

```
root@vultr:~# suricata -c /etc/suricata/suricata.yaml -i enp1s0
2/12/2023 -- 11:23:48 - <Notice> - This is Suricata version 6.0.10 RELEASE running in SYSTEM mode
2/12/2023 -- 11:23:48 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match the pattern
/etc/suricata/rules/suricata.rules
2/12/2023 -- 11:23:48 - <Notice> - all 1 packet processing threads, 4 management threads initialized,
engine started.
^C2/12/2023 -- 11:24:19 - <Notice> - Signal Received. Stopping engine.
2/12/2023 -- 11:24:21 - <Notice> - Stats for 'enp1s0': pkts: 322904, drop: 17840 (5.52%), invalid
chksum: 0
```

「 성능 강화 」

성능 강화

소스코드 수정을 통한 성능 개선

「 성능 강화 」

Testbed Instruction Report

XDP를 활용한 패킷 드롭 실험환경 구축 가이드 및 테스트베드

A Comprehensive Guide and Testbed for Packet Drop Experimentation Using XDP

제 출 자: 함준형, 한승연

2023. 12. 04.

소프트웨어학부 정보보안학 전공

배 재 대 학 교

테스트베드 구축 및 인스트럭션 보고서 작성

21학번 2학년 한승연 학생의 피드백을 프로젝트 적용

- gcc, clang과 같은 컴파일러를 통해 c 코드를 어셈블리 코드로 변환하고 소스코드를 수정하여 성능을 개선할 수 있을듯

「 테스트 」

공격 테스트

UDP Flooding 공격

메인 PC에서 메인 PC의 VM으로 UDP Flooding 공격

공격 PC에서 UDP Flooding 공격 진행

6. 대상 IP 입력

```
C:\Users\WY\VMAppData\WLoca x + v
-----OpsIDied's SAMP Server Crasher-----
Enter the host IP.
```

7. Port 번호 입력

```
-----OpsIDied's SAMP Server Crasher-----
Enter the host IP.
192.168.111.134
Enter the host port.
```

8. 공격횟수 입력(0은 무제한)

```
C:\Users\WY\VMAppData\WLoca x + v
-----OpsIDied's SAMP Server Crasher-----
Enter the host IP.
192.168.111.134
Enter the host port.
88
Enter the number of attacks to send.
```

9. 공격 진행

```
C:\Users\WY\VMAppData\WLoca
Attack 020100 sent.
Attack 020101 sent.
Attack 020102 sent.
Attack 020103 sent.
Attack 020104 sent.
Attack 020105 sent.
Attack 020106 sent.
Attack 020107 sent.
Attack 020108 sent.
Attack 020109 sent.
Attack 020110 sent.
Attack 020111 sent.
Attack 020112 sent.
Attack 020113 sent.
Attack 020114 sent.
```

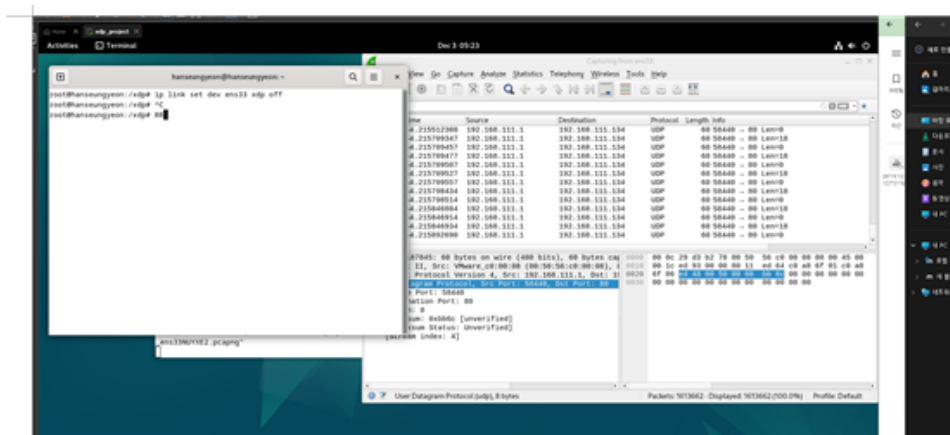
「 테스트 」

공격 모니터링

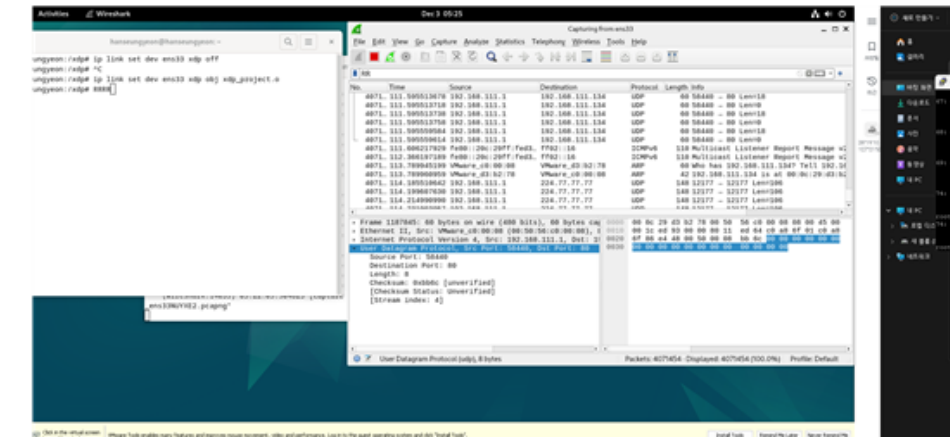
WireShark를 통한 패킷 모니터링

XDP를 비활성화하면 UDP 80 포트로 악성 패킷이 유입되는 것을 확인할 수 있음

XDP를 활성화하면 패킷이 유입되는 것이 없는것을 확인할 수 있음



XDP를 비활성화 했을때 WireShark에서 UDP 80포트로 패킷이 유입되는 것을 확인



XDP를 활성화 했을 때 WireShark에서 패킷이 차단되는 것을 확인할 수 있음

「 보안 프로젝트 - 함준형 」

감사합니다

네트워크 보안 강화를 위한 정밀한 패킷 필터링 시스템

Web. <https://clast.kr>

E-Mail. master@clast.kr

