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

1. 최상위 디렉토리(/) 폴더에 XDP 폴더 생성
2. XDP 소스코드 작성
3. XDP를 eBPF 프로그램으로 확인하는 방법
4. 공격 PC에서 UDP Flooding 공격 진행
5. 테스트 영상

최상위 디렉토리(/) 폴더에 XDP 폴더 생성

명령어:
```
root@hanseungyeon:/# touch xdp
```

결과:
```
root@hanseungyeon:/# cd xdp
root@hanseungyeon:/xdp#
```

XDP 소스코드 작성

```c
#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("prog")
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";
```

eBPF를 이용한 패킷 중 목적지 포트가 80인 UDP 패킷을 차단하는 소스코드

1. SEC(“prog”)- 이 프로그램은 XDP를 통해 패킷 처리를 수행한다.
2. int block_udp(struct xdp_md *ctx) - 이 함수는 각 패킷에 대해 호출되며, 패킷의 데이터를 가리키는 포인터를 매개변수로 받는다. => 패킷 처리 함수
3. Ethernet 헤더 검사: if ((void *)(eth + 1) > data_end) return XDP_PASS; - 이 if문은 Ethernet 헤더가 완전한지 확인합니다. Ethernet 헤더 이후에 데이터가 없다면, 즉, eth + 1이 data_end보다 크다면, XDP_PASS를 반환하고, 패킷을 그대로 통과시킨다.
4. IP 패킷 확인: if (eth->h_proto != htons(ETH_P_IP)) return XDP_PASS; - 이 if문은 패킷이 IP 패킷인지 확인합니다. Ethernet 헤더의 프로토콜 필드(h_proto)가 IP를 의미하는 ETH_P_IP와 같지 않다면, XDP_PASS를 반환하고, 패킷을 그대로 통과시킨다.
5. IP 헤더 검사: if ((void *)(ip + 1) > data_end) return XDP_PASS; - 이 if문은 IP 헤더가 완전한지 확인합니다. IP 헤더 이후에 데이터가 없다면, 즉, ip + 1이 data_end보다 크다면, XDP_PASS를 반환하고, 패킷을 그대로 통과시킨다.
6. UDP 패킷 확인: if (ip->protocol != IPPROTO_UDP) return XDP_PASS; - 이 if문은 패킷이 UDP 패킷인지 확인합니다. IP 헤더의 프로토콜 필드(protocol)가 UDP를 의미하는 IPPROTO_UDP와 같지 않다면, XDP_PASS를 반환하고, 패킷을 그대로 통과시킨다.
7. UDP 헤더 검사: if ((void *)(udp + 1) > data_end) return XDP_PASS; - 이 if문은 UDP 헤더가 완전한지 확인합니다. UDP 헤더 이후에 데이터가 없다면, 즉, udp + 1이 data_end보다 크다면, XDP_PASS를 반환하고, 패킷을 그대로 통과시킨다.
8. 목적지 포트 확인: if (udp->dest != htons(80)) return XDP_PASS; - 이 if문은 UDP 패킷의 목적지 포트가 80인지 확인합니다. 목적지 포트(dest)가 80이 아니라면, XDP_PASS를 반환하고, 패킷을 그대로 통과시킨다.

기능

1. 패킷 검사: 이 함수 내에서는 패킷의 Ethernet 헤더,IP 헤더, 그리고 UDP 헤더를 순차적으로 검사한다. 각 단계에서 헤더의 유효성을 검사하고, 유효하지 않다면 XDP+PASS를 반환하여 패킷을 그대로 통과시킨다.
2. UDP 패킷 차단: 만약 패킷이 모든 검사를 통과하고, UDP 패킷의 목적지 포트가 80이라면, 이 함수는 XDP_DROP을 반환하여 패킷을 차단한다.
3. 라이선스: char _license[] SEC(“license”) = “GPL”; - 이 프로그램은 GPL 라이선스를 사용한다.
4. 네트워크 트래픽을 관리하고, 특정 조건의 패킷을 차단하는 역할을 수행한다. 이는 서버의 보안을 강화하거나, 특정 트래픽을 제어하는 데 사용된다.

소스코드 출력

```
root@hanseungyeon:/xdp# nano xdp_project.c
root@hanseungyeon:/xdp# cat xdp_project.c
root@hanseungyeon:~/xdp# cat xdp_project.c
#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("prog")
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";
```

i. mkdir 명령어를 통해 xdp 폴더를 생성한 뒤, nano 명령어를 이용하여 xdp_project.c 소스코드를 작성한다.

ii. cat 명령어를 통해 xdp_project.c 소스코드를 확인한다.

컴파일 설치

```
root@hanseungyeon:/xdp# which clang
root@hanseungyeon:/xdp# sudo apt-get install clang
```

xdp 소스코드를 컴파일 진행 전 필요한 라이브러리를 설치한다.

*. which: 라이브러리 경로 확인 명령어, clang 라이브러리가 없으므로 clang 설치를 진행해야한다.

† . apt-get install clang: clang 설치 명령어

ǂ . 컴파일 명렁어:

```
root@hanseungyeon:/xdp# clang -O2 -target bpf -I/usr/src/linux-headers-$(uname -
r)/include/uapi/linux/bpf -I/usr/include/x86_64-linux-gnu -c xdp_project.c -o xd
p_project.o
```

라이브러리 설치 명령어

i. apt install gcc

ii. apt-get install libc6-dev-i386

iii. apt-get install libbpf-dev

iv. apt-get install linux-tools-common linux-tools-generic linux-tools-`uname

v. github에서 리눅스 커널 버전을 통일 시켜주는 코드를 다운 해야된다.

5-1: sudo apt-get install git

5-2: git clone https://github.com/torvalds/linux.git

컴파일 성공 출력화면

xdp_project.c eBPF 파일을 컴파일 작업을 진행한다.

이를 위해 clang 컴파일러를 사용하였고, 필요한 헤더 파일들의 위치를 -I 옵션으로 지정해 주었다.

```
root@hanseungyeon:/xdp# clang -O2 -target bpf -I/usr/src/linux-headers-$(uname -r)/inc
ude/uapi/linux/bpf -I/usr/include/x86_64-linux-gnu -c xdp_project.c -o xdp_project.o
root@hanseungyeon:/xdp# ls -l xdp_project.o
-rw-r--r-- 1 root root 912 Dec  3 00:58 xdp_project.o
root@hanseungyeon:/xdp#
```

xdp_project.o 파일이 성공적으로 생성되었음을 보여준다. 파일의 크기는 912byte 이고, 마지막으로 수정된 날짜는 12월 3일 00:47이다.

XDP를 eBPF 프로그램으로 확인하는 방법

```
root@hanseungyeon:/home/hanseungyeon# ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
 group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
e DEFAULT group default qlen 1000
    link/ether 00:0c:29:d3:b2:78 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
```

1. ip link show 명령어를 통해 네트워크 인터페이스 확인

네트워크 인터페이스 이름이 ens33 인 것을 확인

```
root@hanseungyeon:/xdp# sudo ip link set dev ens33 xdp obj xdp_project.o
Error: XDP program already attached.
root@hanseungyeon:/xdp#
```

2. ip link set dev [인터페이스 이름] xdp obj xdp_protect.o 명령어를 통해 네트워크 인터페이스에 xdp 적용

해당 에러 메시지는 xdp 네트워크 인터페이스가 이미 적용이 되어있어서 에러가 출력하는 것

```
root@hanseungyeon:/xdp# ip link show dev ens33
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 xdpgeneric qdisc fq_codel state UP
 mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:d3:b2:78 brd ff:ff:ff:ff:ff:ff
    prog/xdp id 74 name block_udp tag 74ccc25185b0cb65 jited
    altname enp2s1
root@hanseungyeon:/xdp# ^C
root@hanseungyeon:/xdp#
```

3. ip link show dev ens33 명령어를 통해 eBPF 프로그램의 이름은 block_udp이고, 프로그램의 ID는 74 인 것을 확인

jited 명칭은 eBPF 프로그램이 JIT(Just-In-Time) 컴파일러를 통해 기계 코드로 변환 되었음을 의미

eBPF 프로그램이 ens33 인터페이스에 성공적으로 적용되었음을 확인

4. bpftool prog show id 74

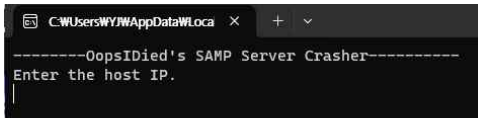
```
root@hanseungyeon:/linux/tools/bpf/bpftool# ./bpftool prog show id 74
74: xdp  name block_udp  tag 74ccc25185b0cb65  gpl
        loaded_at 2023-12-03T01:06:13-1000  uid 0
        xlated 184B  jited 121B  memlock 4096B
```

5. make

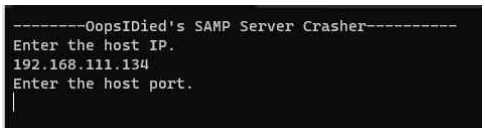
```
root@hanseungyeon:/linux/tools/bpf/bpftool# make

Auto-detecting system features:
...                         clang-bpf-co-re: [ on  ]
...                                    llvm: [ on  ]
...                                  libcap: [ OFF ]
...                                  libbfd: [ OFF ]

  CC       jit_disasm.o
  GEN      pid_iter.skel.h
  CC       pids.o
  CLANG    profiler.bpf.o
  GEN      profiler.skel.h
  CC       prog.o
  CC       struct_ops.o
  CC       tracelog.o
  CC       xlated_dumper.o
  CC       disasm.o
  LINK     bpftool
```

공격 PC에서 UDP Flooding 공격 진행

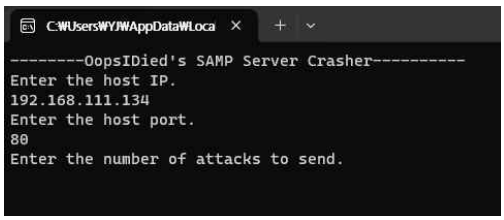
6. 대상 IP 입력



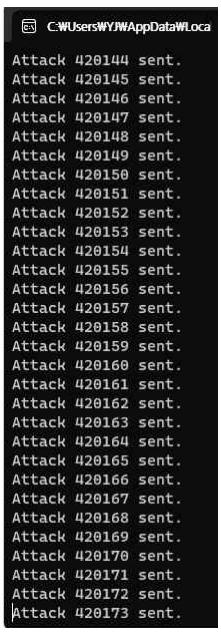
7. Port 번호 입력

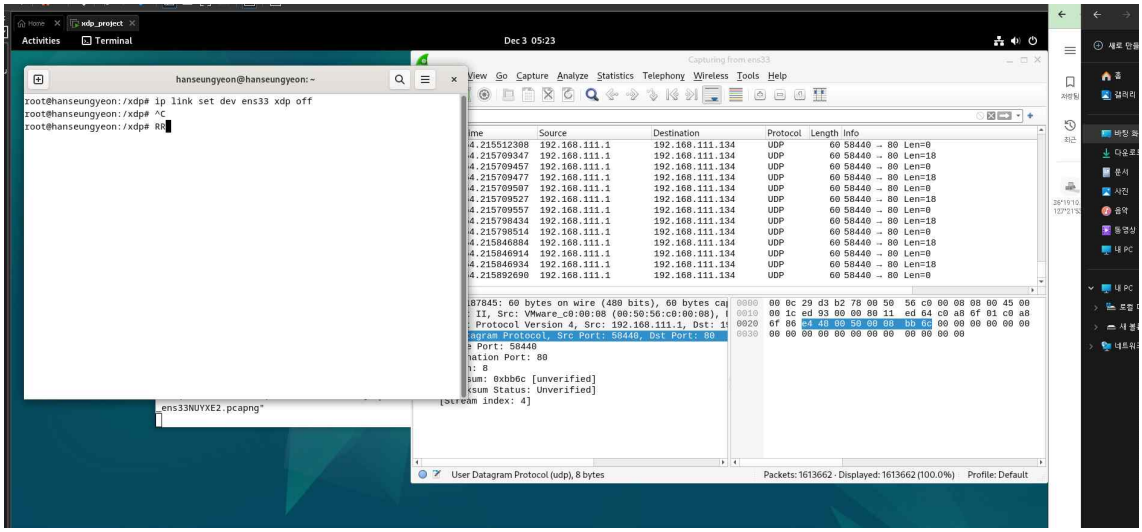


8. 공격횟수 입력(0은 무제한)

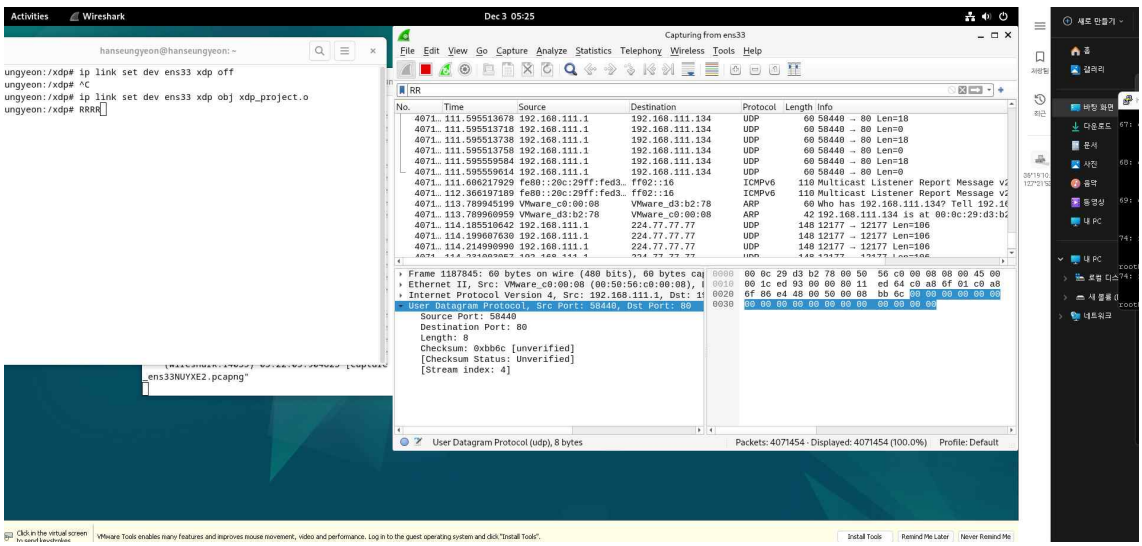


9. 공격 진행





XDP를 비활성화 했을때 WireShark에서 UDP 80포트로 패킷이 유입되는 것을 확인



XDP를 활성화 했을 때 WireShark에서 패킷이 차단되는 것을 확인할 수 있음

테스트 영상

https://clipchamp.com/watch/mN0GX4eRKr7