2014년 정보보호학과 졸업작품 보고서

IDS와 Netfilter를 이용한 서버 보안

팀명 : 8조 K1L4

지도교수 : 양환석 교수님

조장 : 이유원
김병섭
이민철
이동원
이샛별

2014.5
중부대학교 정보보호학과

목 차

Ⅰ. 서론

1. 연구의 배경 및 목적

우리가 구축하고자 하는 침입탐지시스템은 실시간으로 네트워크를 모니터링하고 사용자가 정의한 보안정책을 적용해 불법적인 침입에 대응할 수 있지만 실시간으로 패킷을 분석하기 위한 처리 능력과 저장의 한계를 극복해야 하고, 네트워크 트래픽 증가에 따른 별도의 하드웨어 및 소프트웨어가 필요하다. 또한 현재 많은 연구들이 비정상적인 탐지기술을 중심으로 이루어지고 있으나 아직 많은 솔루션들은 단순한 오·남용(Misuse) 등의 비정상 행위들에 대한 탐지를 주로 사용하고 있다. 즉 개별적인 사용에 대한 탐지보다는 일반적인 사용시간, 네트워크 접속 수, 인증 실패 회수 등이 탐지를 위한 척도로 사용되는 것으로 제한되고 있다. 침입탐지시스템이 활성화되기 위해서는 거짓탐지율(false positive)의 제거, 탐지실패율(false negative)의 제거, 보안 관련 이벤트는 정의와 보고방법, 침입탐지시스템의 테스트 방법의 고안, 탐지된 공격의 피해 정도의 결정과 피해의 최소화 방법 제공, 네트워크에서 요구하는 크기에 변화할 수 있는 시스템 등과 같은 기술적인 논점들이 해결되어야 한다.

그러나 침입탐지시스템의 한계로는 침입을 탐지하여 알리기만 할 뿐 적극적 대응 방법이 없어 본 프로젝트는 이러한 문제점을 해결하기 위해 실시간 방어시스템을 도입해 특정 공격을 사전에 탐지하고 차단해주는 환경을 구현한다.

Ⅱ. 이론적 배경

1. IDS(Intrusion Detection System)

1) IDS의 정의

IDS는 대상 시스템(네트워크 세그먼트 탐지 영역)에 대한 인가되지 않은 행위와 비정상적인 행동을 탐지하고, 탐지된 불법 행위를 구별하여 실시간으로 침입을 차단하는 기능을 가진 보안시스템이다. 또한 한국인터넷진흥원 (http://www.kisa.or.kr)에서는 침입차단시스템을 '컴퓨터 시스템의 비정상적인 사용, 오용·남용 등을 가능하면 실시간으로 탐지하는 시스템'이라고 정의한다.

IDS은 일반적인 보안시스템 구현 절차의 관점에서 침입차단시스템과 더불어 가장 우선적으로 구축되었으며, IDS의 구축 목적은 해킹 등의 불법 행위에 대한 실시간 탐지 및 차단과 침입차단시스템에서 허용한 패킷을 이용하는 해킹 공격의 방어 등의 목적으로 구축된다.

2) IDS 도입 목적 및 필요성

IDS은 IT 정보자산을 공격하는 구체적인 해킹 행위를 실시간으로 탐지함으로써, 보다 능동적으로 침해 사고를 예방하고 대응할 수 있는 보안시스템이다. 또한 침입차단시스템에서 허용한IP와 Port를 이용한 불법 행위를 탐지함으로써 침입차단시스템의 단점을 보완할 수 있으며, 보안 관리자의 부재 시에도 탐지된 해킹 행위에 대한 자동대응을 수행함으로써 보다 효과적인 정보 보안을 할 수 있다.

- 해킹 행위에 대해 보다 능동적인 대응 : 실시간 Session kill을 통한 실시간 대응
- 침입차단시스템의 수동적인 보안 정책 보완 : IP주소와 서비스 port 단위 통제의 한계성을 보완하여 네트워크 패킷을 분석하여 실제 공격 행위를 탐지하여 대응
- 효과적인 보안 강화 : 설정된 보안 정책에 따라 관리자의 부재 시에도 자동 대응이 가능
- 내부사용자의 불법 행위 탐지 및 대응 : 외부 사용자의 불법 행위 뿐만 아니라 내부 사용자의 권한을 초과하는 행위와 오용 행위를 탐지

[표1] IDS의 주요기능

주요 기능	설 명
정보분석 및 실시간 모니터링기능	IDS에서 감시하는 대상 네트워크 세그먼트의 모든 패킷을 실시간으로 감시하고 분석하는 기능
공격패턴의 인식 및 탐지기능	감시대상 영역에서 일어나는 알려진 공격 행위에 대한 패턴 인식 및 탐지 기능
탐지된 공격에 대한 통지기능	탐지된 공격 행위에 대하여 IDS에서 정의한 보안 정책에 따라 Notify, Syslog, E-mail, SNMP trap경보 전송 등의 통보기능
실시간 침입대응 기능	탐지된 공격 행위에 대하여 IDS에서 정의한 보안 정책에 따라 Connection Kill, 연동 기능을 이용하여 공격행위를 차단하는 행위
로그 기반의 침입행위 사후관리기능	IDS에서 탐지된 침입 정보에 대한 사후 검토 및 데이터베이스 기록저장, 보고서작성 등의 사후관리 기능

[표2] IDS의 종료

분류기준	설 명
데이터 소스기반	단일 호스트 기반의 침입탐지 (single host Based)
	다중 호스트 기반의 침입탐지 (Multi host Based)
탐지기법 기반	비정상행위 침입탐지기법 (Anomaly Detection)
	오용행위 침입탐지기법 (Misuse Detection)

3) IDS의 보안 정책

IDS의 표준 보안 정책은 감시대상 영역에서 탐지하고자 하는 공격 패턴(Attack Signature)에 대한 정의와 탐지된 공격 행위에 대한 대응 방법을 보안 정책으로 정의한다.

- 공격 패턴(Attack Signature)

IDS에서 탐지하는 공격 패턴은 '백도어', '서비스거부 공격', '불법접근시도', '사전공격시도', '의심스러운 행위', '프로토콜 해독공격' 등으로 구분되며, 각 분류 기준별 대표적인 공격 유형은 아래와 같다.

[표3] 공격 패턴

공격 패턴 구분	공격 유형
백도어(Back Doors)	Backdoor, Net Bus
서비스 거부공격 (DoS)	Flooding, fragment, ping, Winnuke
불법접근시도 (Unauthorized Access Attempts)	Login, buffer overflow, cmd, imap
사전공격시도 (Pre Attack Probes)	DNS, ftp syst detect, Cisco ident
의심스러운 행위 (Suspicious Activity)	Finger, ident error, rpc, port map, rpc
프로토콜 해독 (Protocol Decode)	Decode, netbios session, bootparam, Cookie

- 공격 패턴의 위험도

IDS에서 탐지하는 공격 패턴은 국제 표준화되고 있는 CVE(Common Vulnerability Exposure)를 기준으로 하여, 그 위험도에 따라 High, Medium, Low의 위험도가 각 공격 패턴별로 정의되어 있다. 공격 패턴의 위험도는 탐지된 공격 패턴이 실제 공경이 아닌데도 공격으로 판단하는 오판여부(False Positive)와 공격 패턴의 중요도 및 위급상황 정도에 따라 구분된다.

4) IDS의 한계성 및 고려 사항

IDS은 다양한 공격 기법으로 이루어지는 실제 공격 행위에 초점을 맞추어, 공격을 실시간으로 탐지하고 자동으로 대응하도록 설계된 보안 시스템이다. 그러나 공격행위에 대한 자동대응 TCP포트를 이용하는 해킹공격으로 제한되며, 공격자를 근본적으로 차단할 수 없는 한계성을 가지고 있다. 따라서 IDS(Firewall)과 연동을 통하여 각 단위 보안시스템의 장단점을 상호 보완함으로써 보안 수준을 향상시킬 수 있다.

[표4] IDS의 한계성 및 고려 사항

종류	내용
침입에 대한 자동대응	침입탐지 시스템의 자동대응 기능은 TCP 를 이용한 해킹 공격으로 제한되며, DP/ICMP 등의 프로토콜에 대한 자동 기능이 적용될 수 없다.
근원적 차단	침입탐지 시스템은 네트워크에 흐르는 패킷을 실시간으로 탐지하며 해킹으로 단되는 행위에 대하여 TCP reset 를 공격자의 IP주소에 전송하여 onnection kill기능을 구현한다. 하지만 서비스 거부공격등과 병행하여 공격이 이루어질 경우 자동차단이 불가능 할 수 있다.
침입탐지 시스템 우회공격기법	IDS이 일반화 되면서 침입탐지 시스템의 탐지 기능을 우회할 수 있는 공격 기법이 다량으로 발표되고 있다. 따라서 단일 보안 시스템만으로는 안전한 보호 대책의 구현이 미흡하다.

2. Netfilter

1) Netfilter 란?

Netfilter란 Linux Kernel에 모듈로 등록된 프레임워크이다. 그 기능으로는 packet mangling(packet header의 TOS, TTL 등의 값을 변경하는 동작), NAT(Network Address Translation : 외부 네트워크에 알려진 것과 다른 IP주소를 사용하는 내부 네트워크에서 IP 주소를 변환하는 동작), packet filtering(특정 packet을 ACCEPT하거나 DROP시키는 동작)이 있다.

Netfilter는 hook이라고 불리는 논리적인 packet 처리 지점을 정의하고 있는데, 처리 시점에 따라 prerouting hook, input hook, forward hook, output hook, postrouting hook으로 나뉜다. 각 hook지점은 packet 처리를 위한 해당 모듈을 callback 방식으로 호출하고 결과값으로 패킷의 처리방식을 리턴 받는다.

[표5] Netfilter의 동작

동작 분류	설명	
PREROUTING	Interface를 통해 받은 packet은 Kernel sanity chack를 거쳐(not truncated, IP Checksum) hook 안으로 들어옴	
	Conntrack	들어오는 패킷에 관한 연결정보가 기존에 저장되어 있는지 검사
	Mangle	들어오는 패킷의 패킷 정보를 변경(TOS)
	DNAT	들어오는 패킷의 목적지 주소를 변경
IN-ROUTE	Local process로 유입될 패킷인지 네트워크상의 다른 호스트를 위해 forward랑 패킷인지 구분	
INPUT	Local process로 들어오는 패킷의 처리	
	Filter	Rule검사 후 해당 target을 실행(DROP, ACCEPT)
	Conntrack	처음 연결을 맺는 packet일 경우 conntrack정보 저장
FORWARD	같은 네트워크상의 다른 host를 위한 packet의 처리	
	Filter	Rule검사 후 해당 target을 실행(DROP, ACCEPT)
OUTPUT	Local process로 들어오는 패킷의 처리	
	Conntrack	들어오는 패킷에 관한 연결정보가 기존에 저장되어 있는 지 검사
	Mangle	들어오는 패킷의 패킷 정보를 변경(TOS, TTL)
	DNAT	사용하지 않음
	Filter	Rule검사 후 해당 target을 실행(DROP, ACCEPT)
OUT-ROUTE	Appropriate output interface로 packet의 경로 설정	
POSTROUTING	Outgoing interface로 나갈 packet의 처리	
	SNAT	Outgoing packet의 source 주소를 변경
	Masquerade	Outgoing packet의 주소, 포트 변경
	conntrack	처음 연결을 맺는 packet일 경우 conntrack정보 저장

[표6] Netfilter Table의 종류

Table	설명
Mangle	Packet 내부의 TOS, TTL, MARK의 packet정보 변경을 위한 rule과 해당 action(target)의 리스트가 저장되어 있는 테이블
Nat	Packet 내부의 source address 및 destination address의 정보 변경을 위한 rule과 해당 action(target)의 리스트가 저장되어 있는 테이블
Filter	Packet을 필터링하기 위한 rule과 해당 action(target)의 리스트가 저장되어 있는 테이블

2) Iptables의 개념과 기능

Netfilter 프로젝트에서 개발되었으며 2.4.X Kernel 배포 시점부터 리눅스의 일부분으로 제공하였고 iptables 정책은 정렬된 규칙집합으로부터 생성된다. 규칙은 특정 분류의 패킷에 대해 취해야 할 조치를 커널에게 알려주며 하나의 iptables 규칙은 테이블 내에 있는 하나의 chain에 적용된다.

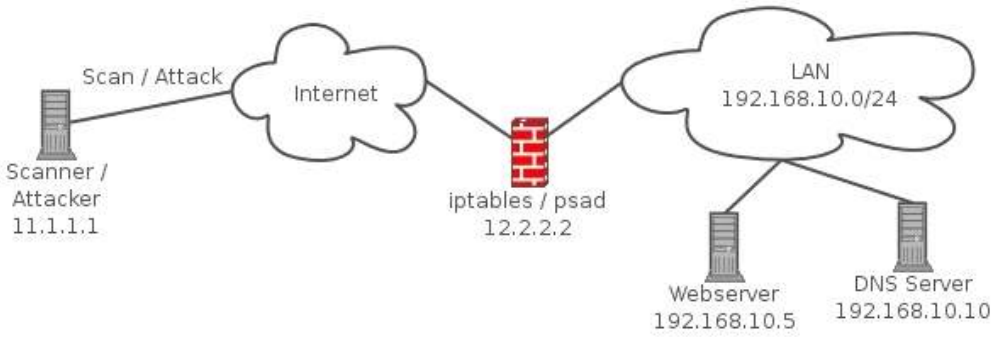
Iptables의 기능으로 상태추적 기능(Stateful Inspection), 향상된 매칭 기능, 포트 포워딩 기능(port forwarding) 등이 내장되어 있다.

상태추적 기능(Stateful Inspection)은 방화벽을 통과하는 모든 패킷에 대한 상태를 추적하여 메모리에 기억하고 기존의 연결을 가장하여 접근할 경우 메모리에 저장된 목록과 비교하여 차단시키는 지능화된 공격차단 기능을 갖는다.

향상된 매칭기능은 방화벽의 기본적인 매칭 정보인 패킷의 IP주소와 포트 번호 뿐만 아니라 추가적으로 다양한 매칭 기능을 제공하고 현재의 연결 상태, 포트 목록, 하드웨어 MAC 주소, 패킷 발신자의 유저나 그룹 프로세스, IP 헤더의 TOS 등 여러 가지 조건을 이용한 세부적인 필터링 기능을 갖는다.

포트 포워딩 기능(port forwarding)은 NAT 기능을 자체적으로 포함하고 있어 사용 가능하고 브리지 등의 다양한 네트워크 구조를 지원하여 원하는 방식대로 네트워크를 구성할 수 있다.

3) psad의 개념과 기능 및 작동방식



[그림1] psad 구상도

psad는 port scan attack detector다. 공격자가 침투할 타겟에 대한 정보를 모으기 위한 사전작업으로서 포트스캔을 하므로 바로 이 점이 포트스캔 시도를 탐지하는 명분이라 할 수 있다. 이 기능을 구현하기 위해서 iptables가 제공하는 로그를 분석한다.
커널은 허용되지 않은 패킷 즉, DROP 될 패킷에 대해서 LOG를 남긴다. log level 은 바꿀 수 있지만 기본 레벨은 warning이다. 로그를 남긴 후 그 패킷은 버린다. 이 방식을 로그 후 버리기 전략이라고 부른다. 커널이 발생시킨 warning 레벨 로그메시지는 syslog에게 넘어간다. syslog의 작동은 syslog.conf에 의해서 조작된다. syslog.conf 에 psadfifo 에 kern.=warn 을 기록하게 돼있기 때문에 syslog는 이 명명된 파이프에 warning 레벨 커널 로그를 기록한다. 물론 그 로그에는 DROP된 패킷에 대한 정보 뿐만 아니라 네트워크와 관련 없는 warning 레벨 커널 로그도 끼어 있을 것이다. psad와 함께 동작하는 도우미 데몬인 kmsgsd는 psadfifo 에서 DROP 된 패킷에 관련된 로그만 추려서 fwdata 로그 파일에 저장한다.

■ 작동방식

- iptables가 남긴 로그에 전적으로 기반함
- 남겨진 로그에는 커널이 DROP 시킨 패킷들에 대한 정보가 상세히 기술되어 있음
- iptables가 남긴 로그를 분석해서 포트 스캔을 탐지함
- 분석된 내용을 기반으로 능동적 응답이 가능함

3. 소켓 프로그래밍

소켓은 UC버클리 대학에서 1989년에 만들어졌다. 이 소켓을 버클리 소켓(berkely sockets) 혹은 BSD소켓이라고 부른다. 당시 인터넷은 급속히 상업화되고 있었다. AT&T와 같은 회사가 소켓기술을 독점하기 위한 시도를 하게 되는데, 버클리대학에서 이를 자유롭게 사용할 수 있도록 공개했다. 그 결과 특정 기업에 종속되지 않고 자유롭게 사용할 수 있는 BSD소켓이 만들어졌다.
BSD소켓은 네트워크 프로그램을 작성하기 위한 라이브러리를 포함한 API(Application Programming Interface)를 제공한다. 이 API는 C프로그래밍 언어로 만들어졌다. 자유롭게

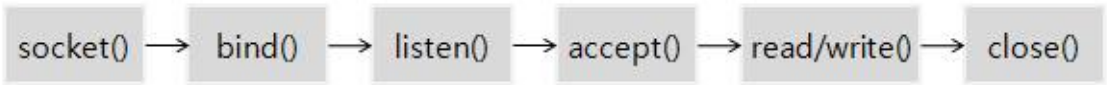
사용할 수 있으며 산업표준인 C언어를 지원한다는 점 때문에, 가장 널리 사용되는 네트워크 프로그래밍 도구가 되었다. 네트워크 프로그램의 대부분이 소켓으로 개발된다고 보면 된다.

소켓 네트워크 프로그래밍이란 소켓 계층에서 제공하는 소켓 함수로 프로그램을 개발하는 과정을 의미하며 소켓은 프로그램과 프로그램이 연결되어서 통신할 수 있도록 통신선로를 만드는 역할을 한다.

소켓 프로그래밍은 클라이언트와 서버로 나누어 프로그램을 만드는데, 고객의 입장에서 서버로 서비스 요청을 하는 프로그램을 클라이언트 프로그램이라고 부른다. 반대로 클라이언트의 요청을 받아서 서비스를 제공하는 프로그램을 서버 프로그램이라고 부른다. 서버는 여러 클라이언트의 요청을 처리할 수 있어야 하며 대량의 데이터를 유지하는 경우가 많아서 클라이언트보다 복잡하며 강력한 컴퓨터가 사용된다. 동시에 여러 클라이언트가 요청할 수 있다.

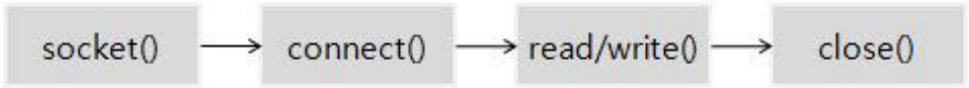
서버/클라이언트 프로그램에 대하여 작성하기 전 서버 프로그램과 클라이언트 프로그램의 흐름에 대해 살펴보도록 하고 각 함수에 대해서도 알아보자.

■ 서버 프로그램 흐름도



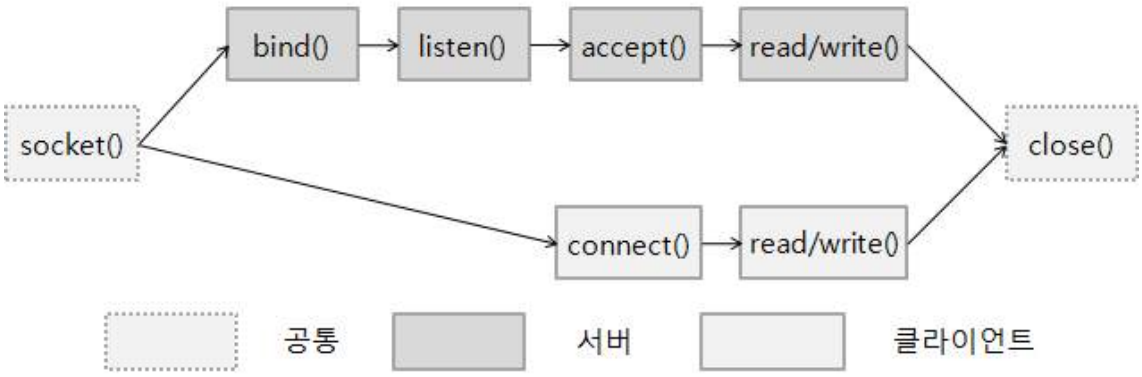
[그림2] 서버 프로그램 흐름도

■ 클라이언트 프로그램 흐름도



[그림3] 클라이언트 프로그램 흐름도

■ 서버 / 클라이언트 프로그램 흐름도



[그림4] 서버 / 클라이언트 프로그램 흐름도

1) socket 함수로 소켓 생성하기

socket 함수를 이용하면 인터넷에서 오는 연결 요청을 받아들일 수 있는 소켓을 생성할 수 있다.

다음은 socket 함수의 원형이다.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

■ 도메인 : 여기서는 AF_INET 으로 인터넷 영역에서 물리적으로 서로 멀리 떨어진 컴퓨터 사이의 통신을 위해서 사용한다. 일반적인 네트워크 프로그램은 이 값을 사용한다.
■ 타입 : SOCK_STREAM 으로 연결지향의 TCP/IP 기반 통신에서 사용한다.
■ 프로토콜 : IPPROTO_TCP 로 TCP 프로토콜로 AF_INET 도메인과 SOCK_STREAM 유형과 함께 사용한다.

int socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); 로 정의된다.

2) bind 함수로 소켓 설정하기

소켓을 생성한 다음 IP주소와 서비스를 위한 포트의 할당을 해야 한다. bind 함수를 이용해서 소켓에 IP와 포트를 할당할 수 있다.
bind 함수는 서버 프로그램에 포트를 고정시키기 위해서 사용한다. 클라이언트는 연결할 인터넷 서비스(서버)의 포트 번호만 알고 있으면 된다. 그래서 클라이언트는 bind 함수를 사용할 필요가 없다.

다음은 bind 함수의 원형이다.

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr  *my_addr, socklen_t addrlen);
```

bind 함수의 매개 변수
- sockfd : socket 함수로 생성된 듣기 소켓
- my_addr : IP 주소와 port 번호를 저장하기 위한 변수가 있는 구조체
- addrlen : 두 번째 인자의 데이터 크기
- 반환 값 : 성공하면 0, 실패하면 -1

bind 함수를 이용해서 IP와 포트를 지정하는 방법

```
struct sockaddr_in clientaddr, serveraddr, myaddr;

memser(&serveraddr, 0x00, sizeof(serveraddr)); //사용자가 지정한 값(0x00)으로 초기화
serveraddr.sin_family = AF_INET; //주소체계 지정
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY); //서버-모든 IP로부터 대기, 클라이언트-특정 IP만 대기
serveraddr.sin_port = htons(12345); //포트번호 지정

state = bind(sockfd, (struct sockaddr *)&serveraddr, sizeof(serveraddr));
if (state = -1)
{
    perror("bind error : ");
    exit(0);
}
```

3) listen 함수로 수신 대기열 생성하기

소켓은 수신 대기열을 만들어 연결 요청을 관리한다. 클라이언트 요청은 먼저 수신 대기열로 들어가서 잠시 기다리게 된다.

listen 함수를 이용해서 이러한 수신 대기열을 생성할 수 있다.

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

listen 함수의 매개 변수

- sockfd : 듣기 소켓의 소켓지정 번호
- backlog : 수신 대기열의 크기(일반적으로 5로 지정한다.)

4) accept 함수로 연결 기다리기

accept 함수는 대기 중인 연결 요청이 있는지 확인한다. 클라이언트가 연결 요청을 하면, 이 연결 요청은 먼저 수신 대기열에 들어가게 되는데, accept 함수는 수신 대기열의 가장 앞에 있는 연결요청을 가져오게 된다.

다음은 accept 함수의 원형이다.

```
#include <sys/types.h>
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

accept 함수의 매개 변수
- sockfd : 클라이언트의 연결 요청을 받아들이는 듣기 소켓이다.
- addr : 연결한 클라이언트의 정보를 확인하거나 로그를 남기기 위한 목적 등으로 사용할 수 있다.
- addrlen : sockaddr 구조체의 크기

5) connect 함수로 연결 요청하기

서버가 accept 함수를 통해서 연결 요청을 기다리면, 클라이언트는 connect 함수를 이용해서 연결 요청을 한다.

다음은 connect 함수의 원형이다.

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

connect 함수의 매개변수
- sockfd : socket 함수로 생성된 소켓 지정 번호
- serv_addr : 연결한 서버의 IP주소와 포트 등의 정보를 알려주기 위해 사용된다.
- addrlen : server_addr의 크기
- 반환 값 : 연결에 성공하면 0, 실패하면 -1을 반환하다.

6) read/write 함수로 데이터 읽고 쓰기

■ read 함수를 이용한 데이터 읽기

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

read 함수의 매개 변수
- fd : 클라이언트 프로그램에서는 socket 함수로 생성된 소켓이고, 서버 프로그램에서는 accept로 생성된 소켓이다.
- buf : 읽어 들인 데이터가 저장될 버퍼 변수이다.
- count : 읽어 들일 데이터의 count 크기이다.
- 반환 값 : 수행 결과가 성공이면 읽어 들인 데이터의 크기(byte단위)를 반환하고, 실패하면 -1을 반환한다.

■ write 함수를 이용한 데이터 쓰기

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

write 함수의 매개 변수
- fd : 연결된 소켓 지정 번호

- buf : 보낼 데이터가 저장되어 있는 버퍼
- count : 보낼 데이터의 크기
- 반환 값 : 성공하면 쓴 데이터의 크기를 반환하고, 실해하면 -1을 반환한다.

7) close 함수로 연결 종료하기

read 함수와 write 함수를 이용해서 데이터 통신을 모두 끝냈다면, 이제 더 사용하지 않는 소켓은 종료시켜야 한다. 종료시키지 않는다면, 만들어진 소켓은 시스템 자원을 계속 소모하게 될 것이다. 이는 시스템 자원을 낭비하게 되므로 반드시 종료하여 사용하지 않는 소켓 자원을 시스템에 반환해야 한다.

다음은 close 함수의 원형이다.

```
include <unistd.h>
int close(int sockfd);
```

III. 연구 내용

1. IDS구축 과정과 실행

1) Snort 설치

(1) snort 설치에 필요한 libpcap, libpcap-devel, pcre, pcre-devel, libdnet, gcc, flex, bison, libdnet-devel, zlib을 yum을 이용하여 설치한다.

(2) 그 후 snort를 다운 받는다.

```
root # tar xvfz /hoem/centos/snort-2.8.6.1.tat.gz
root # cd snort-2.8.6.1
root # ./configure && make && make install  // snort 설치
```

(3) snort에서 제공하는 기본 룰 설치

```
root # mkdir /etc/snort
root # mkdir /var/log/snort
root # cd /etc/snort
root # tar zxvf /home/centos/snortrules-smapshot-2860.tar.gz -C /etc/snort
root # cp etc/* /etc/snort
root # groupadd snort
root # useradd -g snort snort
root # chown snort:snort /var/log/snort
root # mkdir /usr/local/lib/snort_dynamicrules
root # cp /etc/snort/so_rules/precompiled/Centos-5-4/i386/2.9.1.0/*.so
       /usr/local/lib/snort_dynamicrules
```

(4) snort를 기본 룰 설치 후, 정상적으로 동작하는지 확인한다.

```
[root@localhost centos]# snort -V

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.8.6.1 (Build 39)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-t
eam
           Copyright (C) 1998-2010 Sourcefire, Inc., et al.
           Using PCRE version: 6.6 06-Feb-2006
```

[그림5] Snort 정상 동작 확인

2) Rule 추가

(1) ICMP 공격

```
root@bt:~# hping3 --icmp 10.26.0.105 -p 80 -d 65000 --flood
HPING 10.26.0.105 (eth1 10.26.0.105): icmp mode set, 28 headers + 65000 data byt
es
hping in flood mode, no replies will be shown
```

[그림6] ICMP 공격

ICMP 데이터가 65000바이트로 데이터 최대치로 전송이 되었다. 이 공격이 잠깐만 동작해도 타겟 시스템이 다운이 될 수 있는 공격이다. 패킷을 분석해보면 X 문자열을 65000바이트를 채워 보내는 것을 볼 수 있다.

```
59979 7.34169300 10.26.0.185    10.26.0.87      ICMP    1514 Echo (ping) request  id=0x7632, seq=33556/5251, ttl=64

⊞ Frame 59979: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
⊞ Ethernet II, Src: Vmware_fa:e9:38 (00:0c:29:fa:e9:38), Dst: SamsungE_c8:10:47 (00:13:77:c8:10:47)
⊞ Internet Protocol Version 4, Src: 10.26.0.185 (10.26.0.185), Dst: 10.26.0.87 (10.26.0.87)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4f09 [correct]
    Identifier (BE): 30258 (0x7632)
    Identifier (LE): 12918 (0x3276)
    Sequence number (BE): 33556 (0x8314)
    Sequence number (LE): 5251 (0x1483)
  ⊟ Data (65000 bytes)
      Data: 5858585858585858585858585858585858585858585858...
      [Length: 65000]

0000  00 13 77 c8 10 47 00 0c  29 fa e9 38 08 00 45 00   ..w..G.. )..8..E.
0010  05 dc 00 76 2e 74 40 01  30 f4 0a 1a 00 b9 0a 1a   ...v.t@. 0.......
0020  00 57 58 58 58 58 58 58  58 58 58 58 58 58 58 58   .WXXXXXX XXXXXXXX
0030  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0040  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0050  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0060  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0070  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0080  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0090  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00a0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
```

[그림7] ICMP 공격 패킷 분석

(2) ICMP 룰 정책

어느 IP 에서 오던지 타겟 IP 에 58585858 문자열을 가진 패킷이 10 초안 에 20 번 들어오면 탐지를 하는 정책이다.

```
alert ip !10.26.0.105 any -> any any (msg:"ICMP Redirect"; content:"|58 58 58 58
 58 58 58 58|";threshold: type both, track by_src, count 20, seconds 10;  sid:10
00055;)
```

[그림8] ICMP 룰 정책

(3) SYN Flooding 공격

```
root@bt:~# hping3 -S --fast -p 80 10.26.0.105
HPING 10.26.0.105 (eth1 10.26.0.105): S set, 40 headers + 0 data bytes
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=0 win=0 rtt=71.7 ms
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=1 win=0 rtt=2.5 ms
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=2 win=0 rtt=0.3 ms
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=3 win=0 rtt=0.4 ms
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=4 win=0 rtt=0.4 ms
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=5 win=0 rtt=0.3 ms
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=6 win=0 rtt=0.4 ms
len=46 ip=10.26.0.105 ttl=64 DF id=0 sport=80 flags=RA seq=7 win=0 rtt=0.3 ms
```

[그림9] SYN Flooding 공격

(4) SYN Flooding 룰 정책

Rule을 설명하면 옵션으로 flags를 S(Syn) 으로 설정하고 threshold 옵션을 추가한 이유는 여러 번 핑 이 발생해야 스캔으로 감지하기 때문이다. IP 설정은 Snort 네트워크영역을 제외한 영역에서 오는 패킷들을 탐지 한다는 것이다.

```
alert tcp !10.26.0.105 any -> any any (msg:"Syn Flooding";sid:10000006;flags:S;t
hreshold:type both, track by_src, count 10, seconds 5;)
```

[그림10] SYN Flooding 룰 정책

2. Netfilter 구축 과정과 실행

1) 커널 컴파일과 설치

(1) 커널 컴파일시, 메뉴를 화면에 뿌려주기 위한 ncurses-devel를 설치.

root # yum install ncurses-devel

(2) 컴파일하고 싶은 커널 버전 다운로드 및 경로 지정

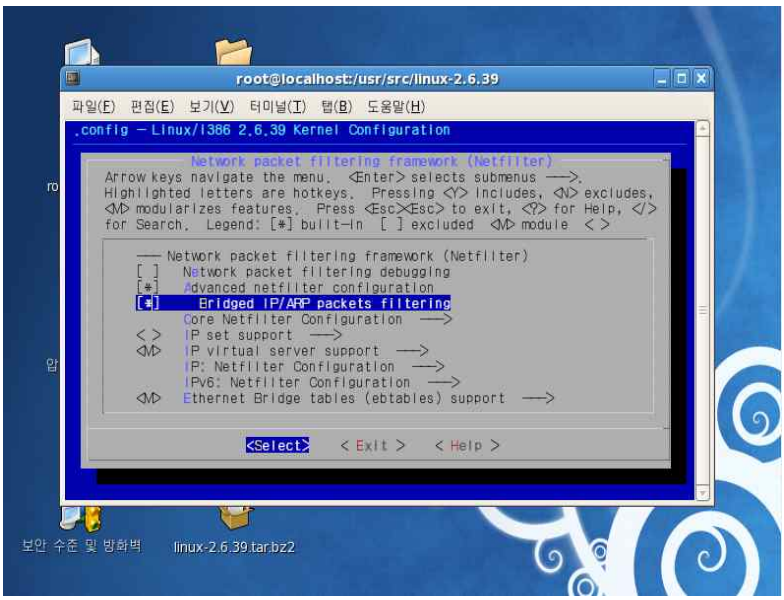
root # wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.36.tar.bz2

```
[root@localhost iptables-1.2.11]# uname -r
2.6.36-default
```

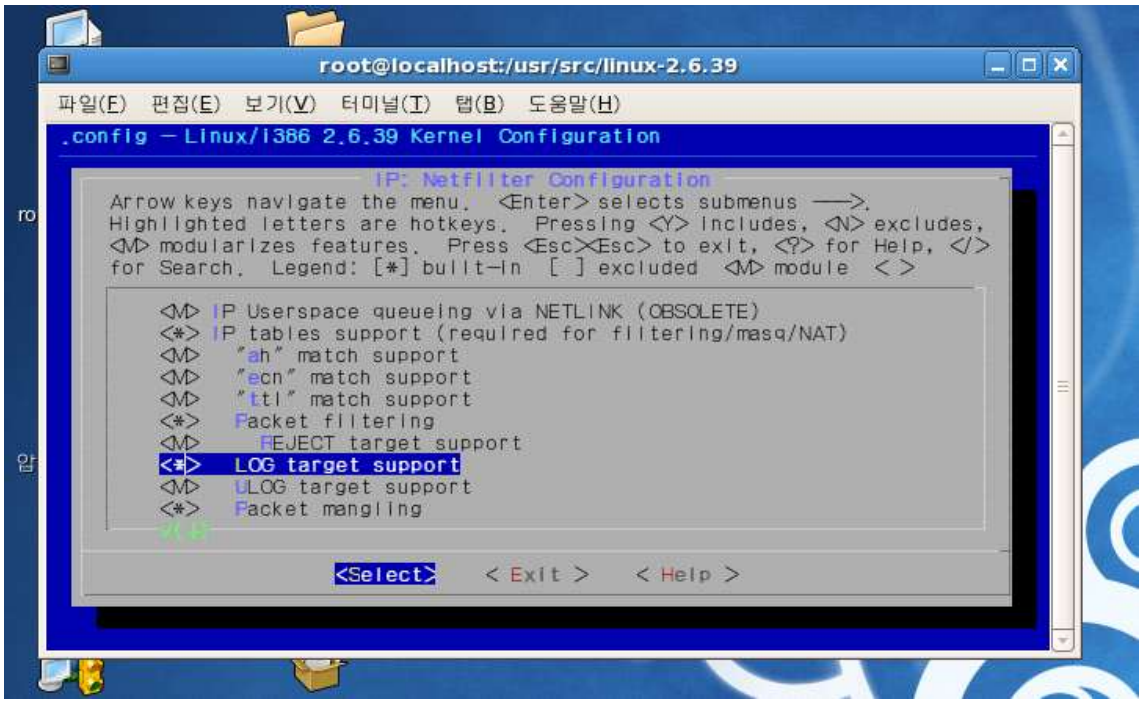
[그림11] 커널 버전 확인하는 명령어

root # tar xvf linux-2.6.36.tar.bz2
root # ln -s linux-2.6.36 linux
root # cd /usr/src/linux
root # make mrproper
root # make clean
root # cp /boot/config-`uname -r` ./.config
root # make menuconfig

- make menuconfig 명령어를 통해 모듈과 커널의 체크 후 config를 통한 컴파일



[그림12] 커널 컴파일 시 방화벽의 브릿지 연결 선택



[그림13] 방화벽 룰을 설정 할 때 로그 기록에 target 대상을 로드할 수 있게끔 체크

root # make all;make modules_install;make install
재부팅 후 linux-2.6.36-default 재설정.

2) iptables 방화벽 설치

기존의 방화벽 확인 후, 최신 버전이 아니라면 기존 것을 지우고 새로 설치 할 준비.

```
root # rpm -qa | grep iptables
root # rpm -e iptables --nodeps

root # wget http://www.iptables.org/files/iptables-1.2.11.tar.bz2
root # bunzip2 -d iptables-1.2.11.tar.bz2
root # tar xvf iptables-1.2.11.tar.bz2
root # cd iptables-1.2.11
root # make install KERNEL_DIR=/boot
```

```
[root@localhost iptables-1.2.11]# iptables -V
iptables v1.2.11
```

[그림14] iptables -v 명령으로 iptables 버전 확인

3) psad 설치 및 구동

```
root # cd /usr/local/src
root # wget http://cipherdyne.org/psad/download/psad-2.1.7.tar.bz2
root # wget http://cipherdyne.org/psad/download/psad-2.1.7.tar.bz2.md5
root # wget http://cipherdyne.org/psad/download/psad-2.1.7.tar.bz2.asc
root # wget  wget http://cipherdyne.org/public_key
root # md5sum -c psad-2.1.7.tar.bz2.md5
root # gpg --import public_key
root # gpg --verify psad-2.1.7.tar.bz2.asc
root # tar xfj psad-2.1.7.tar.bz2
root # cd psad-2.1.7
root # ./install.pl
root # /etc/init.d/psad start
```

```
root@localhost src]# ls
ptables                 psad-2.1.5              psad-2.1.5.tar.bz2.md5
ptables-1.4.3           psad-2.1.5.tar.bz2      public_key
ptables-1.4.3.tar.bz2   psad-2.1.5.tar.bz2.asc  snort_dynamicsrc
root@localhost src]# psad -V
+] psad v2.1.5 (file revision: 2253)
     by Michael Rash <mbr@cipherdyne.org>
```

[그림15] psad 설치된 목록

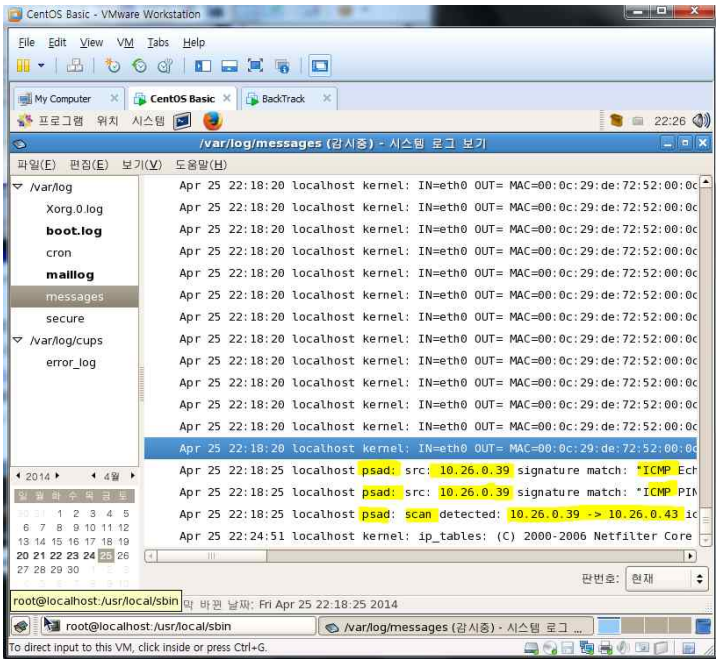
4) SYN 플러딩과 ICMP 공격에 대한 룰 설정

```
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j DROP
-A INPUT -p icmp -j LOG
-A INPUT -p tcp -j LOG --log-prefix "SYN flooding"
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j DROP
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG
-A INPUT -p tcp
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j DROP
-A INPUT -p tcp -j DROP
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG
-A INPUT -p tcp -j LOG
-A INPUT -p tcp -m tcp --dport 80 -j DROP
COMMIT
# Completed on Fri Apr 25 22:50:46 2014
```

[그림16] 공격에 대한 룰 적용 목록

5) 공격 툴을 통해 모의침투 공격시 centos 운영체제 방화벽에서 psad 시스템을 이용한 로그 기록

■ ICMP



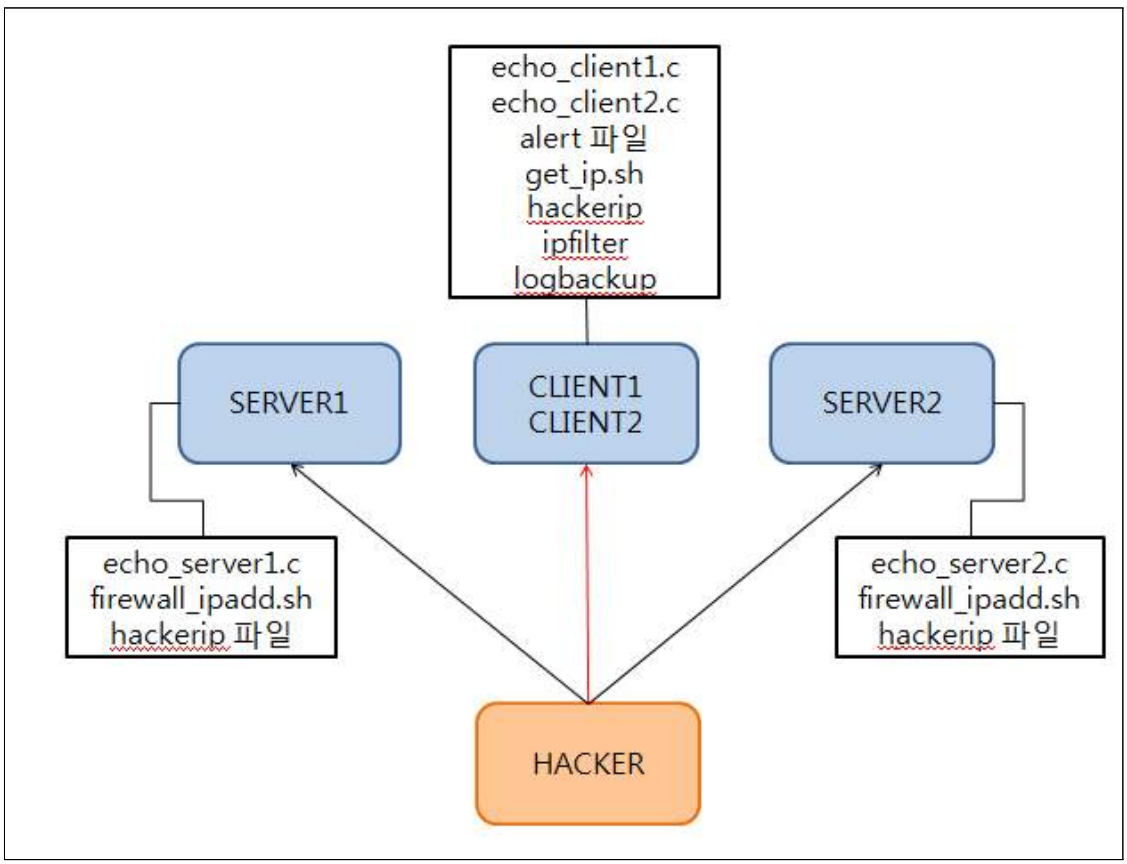
[그림17] ICMP 로그 기록

■ SYN Flooding

```
pr 25 22:40:12 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:13 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:14 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:15 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:16 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:17 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:18 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:19 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:20 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:21 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:22 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:23 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:24 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:25 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:26 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
pr 25 22:40:27 localhost kernel: SYN floodingIN=eth0 OUT= MAC=00:0c:29:de:7
```

[그림18] SYN Flooding 로그 기록

3. 소켓 프로그래밍 소스



[그림19] 소켓 프로그래밍과 쉘 프로그래밍 구상도

[SERVER1, SERVER2]

| echo_server1.c & echo_server2.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>  //소켓 관련 함수
#include <unistd.h>  //각종 시스템 함수
#include <netinet/in.h>
#include <arpa/inet.h>  //소켓 지원을 위한 각종 함수
#include <stdio.h>  //표준 입출력 관련
#include <stdlib.h>
#include <string.h>  //문자열 관련

int main(int argc, char **argv)  //int argc, char **argv 는 main()함수의 매개 변수
{
    int server_sockfd, client_sockfd;
    int client_len;

    struct sockaddr_in clientaddr, serveraddr;  //ip와 port를 저장하는 구조체(서버, 클라이언트)

    char buf[255];
    char test[50];

    if (argc != 2)  //argc가 두 개의 인자를 갖지 않으면
    {
        printf("Usage : ./zipcode [port]₩n");
        printf("예  : ./zipcode 4444₩n");
        exit(0);
    }

    // internet 기반의 소켓 생성 (INET)
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);

    // struct sockaddr_in 값 설정
    bzero(&serveraddr, sizeof(serveraddr));  //무조건 0x00으로 초기화
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons(atoi(argv[1]));  //atoi(argv[1])에 포트번호가 들어간다.
```

```
    //바인드 설정 bind(듣기 소켓, ip와 port저장하는 구조체, 두 번째 인자의 데이터 크기)
    bind(server_sockfd , (struct sockaddr *)&serveraddr, sizeof(serveraddr));

    //리슨 설정 listen(socket함수 수행한 결과 얻은 듣기 소켓 지정 번호, 수신 대기열의 크기)
    listen(server_sockfd, 5);

    //어셉트 설정 accept(클라이언트의 연결 요청을 받아들이는 듣기 소켓, 연결된 클라이언트의 주소와 포트번호 구조체에 저장, 구조체의 크기)

while(1) {
       client_sockfd = accept(server_sockfd, (struct sockaddr *)&clientaddr, &client_len);

     FILE *file = fopen("hackerip", "wt");
     if(file == NULL) {
          printf("File open Error!");
          return 1;
     }

     if( read(client_sockfd, buf, 255) != NULL ) {

            sprintf(test, "./firewall_ipadd.sh %s", buf);
       int ret = system(test);
     }
 }
     close(client_sockfd);
    return 0
}
```

| firewall_ipadd.sh

```
#!/bin/bash
echo "`iptables -A INPUT -s $1 -j DROP`"
echo "`service iptables save`“
```

[CLIENT1, CLIENT2]

| echo_client1.c & echo_client2.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
    int client_len;
    int client_sockfd;

    char buf_in[255];

    struct sockaddr_in clientaddr;

    if (argc != 2)  //argc가 두 개의 인자를 갖지 않으면
     {
        printf("Usage : ./zipcode [port]₩n");
        printf("예  : ./zipcode 4444₩n");
        exit(0);
     }

 //소켓 생성
    client_sockfd = socket(AF_INET, SOCK_STREAM, 0);

 //struct sockaddr_in 값 설정
    clientaddr.sin_family = AF_INET;
    clientaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    clientaddr.sin_port = htons((atoi(argv[1])));

    client_len = sizeof(clientaddr);

  //connect 설정
    connect(client_sockfd, (struct sockaddr *)&clientaddr, client_len);
```

```
    FILE *file = fopen("ipfilter", "rt");
    if(file == NULL) {
    printf("File open Error!₩n₩n");
    return 1;
    }

    while( fgets(buf_in, 255, file) != NULL ) {
        write(client_sockfd, buf_in, 255);
    }
    close(client_sockfd);
    return 0;
}
```

| get_ip.sh

```
#!/bin/bash
ALERT=/var/log/snort/alert
HACKERIP=/root/eight/hackerip
BACKUPFILE=/root/eight/logbackup
IPFILTER=/root/eight/ipfilter
MYIP=`ifconfig -a | grep "inet " | grep "Bcast:" | awk '{print $2}' | cut -d: -f2`

while [ : ]
do
    while [ : ]
    do
        if [ -s $ALERT ]
        then
            echo "`awk /^03/ $ALERT | awk '{print $2}' | cut -d ':' -f1`" >>
$IPFILTER
            echo "`sort $IPFILTER | uniq`" > $IPFILTER

            while read line
            do
                if [ $MYIP != $line ]
                then
                    echo $line >>$HACKERIP
                fi
            done <$IPFILTER
```

```
                echo "`cat $HACKERIP`" >>$BACKUPFILE
                `cat /dev/null` >$ALERT

                echo "`./echo_client1 4444`"
                echo "`./echo_client2 4444`"
                break
            else
                continue
            fi
        done
done
```

Ⅳ. 결론

본 프로젝트의 가장 큰 목표는 IDS를 이용하여 공격을 탐지하고 Netfilter를 이용해 실시간 방어를 목적으로 서론에서 서술한 바와 같이 악성코드와 해킹기법들이 날로 증가하고 다양화되고 있어 보안에 위협을 가하는 요소들을 연구하도록 하였다.

이러한 문제점을 해결하기 위해 실시간 방어시스템을 도입해 특정 공격을 사전에 탐지하고 차단해주는 환경을 구현하였고, 첫 공격의 해커 ip는 IDS를 통해 정상 탐지 후 방화벽에 추가, 두 번째 공격부터 해커의 ip는 차단 및 공격을 거부 하도록 하였다.

세계에는 무수히 많은 기업들이 있다. 그 무수히 많은 기업들 중 보안을 중요시하지 않는 기업은 없다고 봐도 무방하다. 그만큼 보안영역의 의존도가 높고 보안을 중요시하는 기업이 점차 늘고있는 추세이다. 이 프로젝트를 통하여 보안에 대한 경각심을 심어주고 더욱 더 관심을 가질 수 있는 계기가 되었고 이 작품에서 IDS구축, Netfilter 방화벽 구축 및 프로그램 구현 등 구축과정에서의 오류 해결 능력을 향상시켰다.

Ⅴ. 참고 문헌

[1] 윤상배 지음
뇌를 자극하는 TCP/IP 소켓 프로그래밍

[2] 마이클 래쉬 지음 / 민병호 옮김
오픈소스를 활용한 철통같은 보안 리눅스 방화벽

[3] Netfilter 및 iptables 분석
http://mclab.hufs.ac.kr/mediawiki/images/3/36/Iptables.pdf

[4] 한국정보보호진흥원 KISA - 인프라보호단 / 보안관리팀
Snort를 이용한 IDS구축 (2005. 5)

[*] 사이트 참고
위키백과 (침입 탐지 시스템)
http://dlrudwo.tistory.com/47 (netfilter 와 iptables)

백트랙5r3
http://parkya.tistory.com/951

스노트설치 및 실습
http://kthan.tistory.com/58
http://blog.daum.net/ccman82/9
http://semidntmd.tistory.com/search/snort

Snort Rule Header
http://blog.naver.com/PostView.nhn?blogId=white00one&logNo=40088342315

구글 (psad란?)
http://marcof.tistory.com/47

IDS와 Netfilter를 이용한 서버 보안

2014.5.27

지도교수 : 양환석 교수님

8조 K1L4

목 차

조원 소개 및 역할

조장 : 이유원 – 졸업작품 전체 총괄 및 보완

조원 : 이동원 – Netfilter(방화벽) 구축
이민철 – Netfilter(방화벽) 구축
김병섭 – IDS(Snort) 구축
이샛별 – C언어 및 쉘 스크립트 구축

주제 선정 및 작품구성

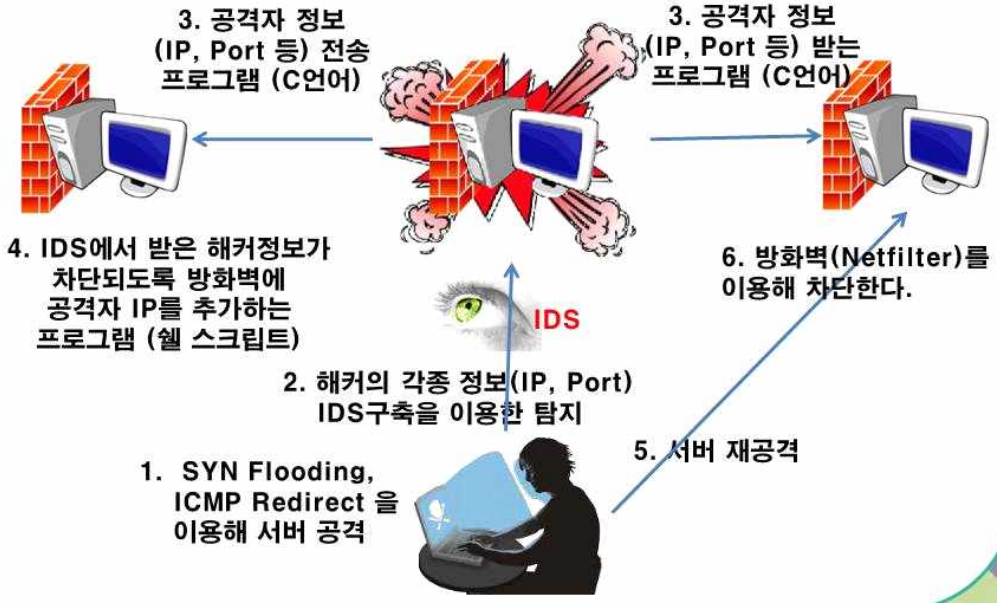
주제 선정

◆ 최근 악성코드와 해킹기법들이 날로 증가 · 다양화됨에 따라 보안에 위협을 가하는 요소들 또한 증가

◆ 이러한 문제점을 해결하기 위해 특정 공격을 사전 탐지하고 차단해 주는 실시간 방어시스템을 구현

※ 사용자가 안전하게 사용할 수 있도록 보안을 제공

작품 구성



개발 내용

개발 환경

- ❖ 운영체제 : 리눅스(CentOs 5.5)
  Window 7 Ultimate K
- ❖ 개발언어 : 쉘 스크립트, C언어
- ❖ 개발도구 : vi, gcc

IDS(Snort) & 공격

1. snort 실행 ➡ `[root@localhost rules]# snort -c local.rules`

```
[ Port Based Pattern Matching Memory ]
+-[AC-BNFA Search Info Summary]------------------------------
| Instances         : 4
| Patterns          : 4
| Pattern Chars     : 36
| Num States        : 36
| Num Match States  : 4
| Memory            :   5.67Kbytes
|   Patterns        :   0.12K
|   Match Lists     :   0.20K
|   Transitions     :   4.39K
+-------------------------------------------------

        --== Initialization Complete ==--

           -*> Snort! <*-
  o"  )~   Version 2.8.6.1 (Build 39)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort
-team
           Copyright (C) 1998-2010 Sourcefire, Inc., et al.
           Using PCRE version: 6.6 06-Feb-2006

Not Using PCAP_FRAMES
```

2. 공격

- ICMP 공격에 해당되는 명령어

```
root@bt:~# hping3 --icmp 10.26.0.207 -d 65000 --flood
HPING 10.26.0.207 (eth1 10.26.0.207): icmp mode set, 28 headers + 65000 data byt
es
hping in flood mode, no replies will be shown
```

- SYN 공격에 해당되는 명령어

```
root@bt:~# hping3 -S --fast -p 80 10.26.0.207
HPING 10.26.0.207 (eth1 10.26.0.207): S set, 40 headers + 0 data bytes
len=46 ip=10.26.0.207 ttl=64 DF id=0 sport=80 flags=RA seq=0 win=0 rtt=0.6 ms
len=46 ip=10.26.0.207 ttl=64 DF id=0 sport=80 flags=RA seq=1 win=0 rtt=0.3 ms
len=46 ip=10.26.0.207 ttl=64 DF id=0 sport=80 flags=RA seq=2 win=0 rtt=0.3 ms
len=46 ip=10.26.0.207 ttl=64 DF id=0 sport=80 flags=RA seq=3 win=0 rtt=0.3 ms
len=46 ip=10.26.0.207 ttl=64 DF id=0 sport=80 flags=RA seq=4 win=0 rtt=0.3 ms
```

◎ SYN Flooding 공격이 로그에 쌓이는 모습

```
[**] [1:10000006:0] Syn Flooding [**]
[Priority: 0]
03/17-17:08:58.051706 10.26.0.205:2124 -> 10.26.0.207:80
TCP TTL:64 TOS:0x0 ID:33795 IpLen:20 DgmLen:40
******S* Seq: 0x75C92CA5  Ack: 0x7B5416F9  Win: 0x200  TcpLen: 20

[**] [1:10000006:0] Syn Flooding [**]
[Priority: 0]
03/17-17:09:13.241098 10.26.0.205:1930 -> 10.26.0.207:80
TCP TTL:64 TOS:0x0 ID:6066 IpLen:20 DgmLen:40
******S* Seq: 0x67F1FE36  Ack: 0x16B979CB  Win: 0x200  TcpLen: 20
```

◎ ICMP Redirect 공격이 로그에 쌓이는 모습

```
[**] [1:1000055:0] ICMP Redirect [**]
[Priority: 0]
03/17-17:08:42.540342 10.26.0.205 -> 10.26.0.207
ICMP TTL:64 TOS:0x0 ID:176 IpLen:20 DgmLen:1500 MF
Frag Offset: 0x0DBB   Frag Size: 0x05C8

[**] [1:1000055:0] ICMP Redirect [**]
[Priority: 0]
03/17-17:08:52.000637 10.26.0.205 -> 10.26.0.207
ICMP TTL:64 TOS:0x0 ID:176 IpLen:20 DgmLen:1500 MF
Frag Offset: 0x194B   Frag Size: 0x05C8
```

C언어 및 쉘스크립트 실행 과정

1. 서버에서 클라이언트 연결 요청 대기

```
[root@localhost eight]# ./echo server 4444
```

2. 클라이언트에서 쉘 스크립트 실행

```
[root@localhost eight]# ./get ip.sh
```

3. 서버에서 firewall_ipadd.sh 쉘 스크립트 실행 & 결과

```
[root@localhost eight]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
           all  --  102.26.0.128         anywhere
DROP       all  --  10.26.0.87           anywhere
```

C언어 및 쉘스크립트 소스 코드

[서버] **echo_server1.c & echo_server2.c(샘플)**

```
                                                    :
                                                    :
while(1) {
    client_sockfd = accept(server_sockfd, (struct sockaddr *)&clientaddr, &client_len);

    FILE *file = fopen("hackerip", "wt");
    if(file == NULL) {
        printf("File open Error!");
        return 1;
    }

    if( read(client_sockfd, buf, 255) != NULL ) {
        sprintf(test, "./firewall_ipadd.sh %s", buf);
        int ret = system(test);
    }
}
                                                    :
                                                    :
```

[해커 ip 획득] **get_ip.sh**

```
#!/bin/bash
ALERT=/var/log/snort/alert
HACKERIP=/root/eight/hackerip
BACKUPFILE=/root/eight/logbackup
IPFILTER=/root/eight/ipfilter
MYIP=`ifconfig -a | grep "inet " | grep "Bcast:" | awk '{print $2}' | cut -d: -f2`

while [ : ]
do
    while [ : ]
    do
        if [ -s $ALERT ]
        then
            echo "`awk /^03/ $ALERT | awk '{print $2}' | cut -d ':' -f1`" >> $IPFILTER
            echo "`sort $IPFILTER | uniq`" > $IPFILTER
```

[해커 ip 획득]

```
            while read line
            do
                if [ $MYIP != $line ]
                then
                    echo $line >>$HACKERIP
                fi
            done <$IPFILTER

            echo "`cat $HACKERIP`" >>$BACKUPFILE
            `cat /dev/null` >$ALERT

            echo "`./echo_client1 4444`"
            echo "`./echo_client2 4444`"
            break
        else
            continue
        fi
    done
done
```

[클라이언트] **echo_client1.c & echo_client2.c**

```
                                        :
                                        :
FILE *file = fopen("ipfilter", "rt");
if(file == NULL) {
    printf("File open Error!₩n₩n");
    return 1;
    }

while( fgets(buf_in, 255, file) != NULL ) {
    write(client_sockfd, buf_in, 255);
}
                                        :
                                        :
```

[방화벽에 추가] **firewall_ipadd.sh**

```
#!/bin/bash
echo "`iptables -A INPUT -s $1 –j DROP`"
echo "`service iptables save`"
```

Nefilter 방화벽

iptables(방화벽) INPUT(외부 → 내부)설정 확인

```
[root@localhost ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
RH-Firewall-1-INPUT  0    --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
RH-Firewall-1-INPUT  0    --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain RH-Firewall-1-INPUT (2 references)
target     prot opt source               destination
ACCEPT     0    --  anywhere             anywhere
ACCEPT     icmp --  anywhere             anywhere            icmp any
ACCEPT     esp  --  anywhere             anywhere
ACCEPT     ah   --  anywhere             anywhere
ACCEPT     udp  --  anywhere             224.0.0.251         udp dpt:mdns
ACCEPT     udp  --  anywhere             anywhere            udp dpt:ipp
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ipp
ACCEPT     0    --  anywhere             anywhere            state RELATED,ESTAB
LISHED
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:s
sh
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:h
ttp
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:h
ttps
REJECT     0    --  anywhere             anywhere            reject-with icmp-ho
st-prohibited
```

→ iptables -L (iptables -list)

Nefilter 방화벽

◎ 공격자 ping 테스트

```
[root@localhost ~]# ping 10.26.0.53
PING 10.26.0.53 (10.26.0.53) 56(84) bytes of data.
64 bytes from 10.26.0.53: icmp_seq=1 ttl=64 time=16.0 ms
64 bytes from 10.26.0.53: icmp_seq=2 ttl=64 time=5.67 ms
64 bytes from 10.26.0.53: icmp_seq=3 ttl=64 time=0.377 ms
64 bytes from 10.26.0.53: icmp_seq=4 ttl=64 time=0.682 ms
```

◎ 공격자에 대한 방화벽 룰 정책 추가

```
[root@localhost ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
RH-Firewall-1-INPUT  0    --  anywhere             anywhere
DROP       0    --  10.26.0.137          anywhere
```

◎ 재공격시 차단

```
[root@localhost ~]# ping 10.26.0.53
PING 10.26.0.53 (10.26.0.53) 56(84) bytes of data.
```

결론 및 의견

❖ 서버보안 체제 구축 성과

- IDS와 방화벽을 연동하는 시스템을 자체 개발, 해커 신호 탐지

- IDS에서 최초 탐지한 해커 ip를 방화벽에 자동 추가,
  이후 공격을 완전하게 거부

❖ 프로젝트 수행과정에서 IDS 및 Netfilter 방화벽 구축, 프로그램 개발 등을 통해 시스템 구현 및 프로그래밍 능력을 향상

Q & A

Thank You