

[NT] Microsoft ASN.1 Library Length Overflow And Bit String Heap Corruption

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-02/0023.html>

From: SecuriTeam (*support_at_securiteam.com*)

Date: 02/11/04

To: list@securiteam.com

Date: 11 Feb 2004 15:00:09 +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>

Microsoft ASN.1 Library Length Overflow And Bit String Heap Corruption

SUMMARY

Critical vulnerabilities in Microsoft's ASN.1 library (MSASN1.DLL) have been found that would allow an attacker to overwrite heap memory and cause the execution of arbitrary code. Because this library is widely used by Windows security subsystems, the vulnerability can be exposed in many ways, including Kerberos, NTLMv2 authentication and applications that make use of certificates (SSL, digitally-signed e-mail, signed ActiveX controls, etc.).

DETAILS

Vulnerable Systems:

- * Microsoft Windows NT 4.0
- * Microsoft Windows 2000
- * Microsoft Windows XP
- * Microsoft Windows Server 2003

Services Affected:

- * Kerberos (UDP/88)
- * Microsoft IIS using SSL

Securiteam: [NT] Microsoft ASN.1 Library Length Overflow And Bit String Heap Corruption

* NTLMv2 authentication (TCP/135, 139, 445)

CVE Information:

<<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0818>>
CAN-2003-0818

Library Length Overflow Vulnerability

A pair of arithmetic errors exist in a generic and low-level part of ASN.1 BER decoding that allow a very large swath of heap memory to be overwritten. This vulnerability affects basically any client of MSASN1.DLL, the most interesting of which are LSASS.EXE and CRYPT32.DLL (and therefore any application that uses CRYPT32.DLL).

In short, the BER encoding scheme is used for representing binary data and is often compared to "binary XML". Each piece of data is encoded as a typed value, which is constructed as a tag number that describes how to interpret the following value data, then the length of the data, and finally, the data itself. By supplying a very large value (from 0xFFFFFFFF to 0xFFFFFFFF) in this field, one can cause an integer overflow in a heap allocation routine. Although there are checks in place to ensure the validity of a value's length, a separate pointer arithmetic overflow in the verification routine gives rise to the vulnerability.

When a simple value (a value that consists of atomic data, rather than more values) is decoded by MSASN1, ASN1BERDecLength() is called to retrieve the length of the value, then passes the reported length to the ASN1BERDecCheck() function to make sure that that much data actually exists.

ASN1BERDecCheck() verifies that (pointer_to_start_of_data + reported_length_of_data), unsigned, is less than or equal to (pointer_to_start_of_BER_block + total_size_of_BER_block). If not, the function returns failure, which propagates back up the call stack and causes decoding to stop.

If the function that called ASN1BERDecLength() attempts to allocate memory and make a copy of the data (example: ASN1BERDecOctetString()), the decoded length will be passed to DecMemAlloc() which rounds it to DWORD multiples and attempts to allocate the memory, similar in fashion to the following call:

```
LocalAlloc(LMEM_ZEROINIT, (length + 3) & ~3);
```

If DecMemAlloc() succeeds, the calling function then memcpy()'s the value data into the allocated heap buffer, using the original decoded length of the value as the byte count.

If a very large length is decoded by ASN1BERDecLength() in step 1, then there will be an integer overflow when ASN1BERDecCheck() adds the length to the current data pointer in step 2, essentially causing the resulting

pointer to "wrap around" and therefore have an address that is numerically less than the pointer to the end of the buffer.

If a length in the range 0xFFFFFFFFD through 0xFFFFFFFF is given, it will pass through ASN1BERDecCheck() with no problem. However, because of the round-off in DecMemAlloc(), the three lengths in this range will all round "up" to zero. Since LocalAlloc() successfully allocates a zero-length heap block whose address gets returned to the caller and the original (very large) length is used in memcpy(), the result is a classic heap overflow, where all contiguous heap memory following the zero-length block is wiped out by arbitrary data.

The simplest way to manifest this condition is to encode a simple octet string (tag 04h) with a length-of-length set to 4 and a length of 0xFFFFFFFF, which corresponds to the bytes 04h/84h/FFh/FFh/FFh/FFh. Depending on which decoder functions the MSASN1 client uses, it is also possible to leverage this vulnerability through OIDs and character strings as well. The following is a sampling of vulnerable decoder functions:

- ASN1BerDecCharString
- ASN1BERDecChar16String
- ASN1BERDecChar32String
- ASN1BERDecEoid
- ASN1BERDecGeneralizedTime
- ASN1BERDecMultibyteString
- ASN1BERDecOctetString
- ASN1BERDecOpenType
- ASN1BERDecSXVal
- ASN1BERDecUTCTime
- ASN1BERDecUTF8String
- ASN1BERDecZeroCharString
- ASN1BERDecZeroChar16String
- ASN1BERDecZeroChar32String
- ASN1BERDecZeroMultibyteString

Bit String Heap Corruption Vulnerability

Software that uses MSASN1 directly or indirectly is vulnerable to a complete overwrite of a large portion of its heap memory due to another integer overflow bug. The attack is specific to bit string values (tags 03h and 23h), but the result is the same as with the heap corruption involving large data lengths.

In short, the BER encoding scheme is used for representing binary data and is often compared to "binary XML". Each piece of data is encoded as a typed value, which is constructed as a tag number that describes how to interpret the following value data, then the length of the data, and finally, the data itself. If a value consists of other values, then it is considered constructed (or compound). If it contains only raw data, then the value is described as

simple. If bit 5 (20h) of the tag byte is set, this indicates that the value is constructed, and MSASN1 will decode the following data as its own BER-encoded block.

In the case of a bit string, the first byte of data is the number of bits (from 0 to 7) to exclude from the end of the bit string value data, since the data is naturally given in bytes. The remaining bytes, then, contain the $(8 * (\text{value_length} - 1) - \text{number_of_unused_bits})$ bits that compose the bit string.

When a bit string is given a length of one byte (only the "number of unused bits" field, with no data bits following) and a non-zero number of unused bits, ASN1BERDecBitString() and ASN1BERDecBitString2() will both report that the length in bits of such a bit string is $(0 - \text{number_of_unused_bits})$, a number that can fall in the range $0xFFFFFFFF9$ (-7) to $0xFFFFFFFF$ (-1), although neither will attempt to copy an amount of data based on this count.

The former function will attempt to copy the length of the original data minus one byte – in this case, zero – which is ok. The latter just returns a pointer into the original BER-encoded block and the length in bits of the data, and is also harmless.

While it's possible that some client application somewhere might misuse this number of bits and create an exploitable condition, it doesn't really matter because there's another integer overflow in MSASN1 that definitely will. ?ASN1BERDecBitString() has a special way of handling constructed bit strings (tag 23h), in that it concatenates each of the simple bit strings that the compound one comprises. ?By supplying a valid constructed bit string that contains a single, simple bit string with length 1 and 7 unused bits, a second integer overflow occurs while adding the number of bits in the bit string to the cumulative total.

The following code from BERDecBitString() performs the vulnerable arithmetic:

```
76195338 mov eax, [ebp-18h] ; = length of simple bit string
7619533B cmp eax, ebx ; (EBX = 0)
7619533D jz short 7619539A ; skip this bit string if empty
7619533F cmp [ebp+14h], ebx ; = no-copy flag
76195342 jnz short 761953AF ; don't concatenate if no-copy
76195344 mov ecx, [esi] ; = count of accumulated bits
76195346 lea eax, [ecx+eax+7] ; *** INTEGER OVERFLOW ***
7619534A shr eax, 3 ; div by 8 to get size in bytes
7619534D push eax
7619534E push dword ptr [esi+4]
76195351 push dword ptr [ebp-4]
76195354 call DecMemReAlloc ; allocates a zero-byte block
```

If the first simple bit string encountered has a length of $0xFFFFFFFF9$ (-7) bits, then the arithmetic at $0x76195346$ will add the total number of

Securiteam: [NT] Microsoft ASN.1 Library Length Overflow And Bit String Heap Corruption

accumulated bits (0), the length of the bit string being concatenated (-7), and then an additional 7 for the purpose of rounding up, to arrive at a total length of zero. This sum is passed to DecMemReAlloc() to allocate a zero-length heap block, but then the bit strings' original lengths in [ESI] and [EBP-18h] are passed on to a function named ASN1bitcpy() (not shown here), which in this case performs a typical memcpy() and overwrites a whole bunch of heap memory as a result.

To demonstrate this vulnerability, all that's necessary is a constructed bit string with length 3, then a simple bit string with length 1 and an unused bits field set to 7, all of which BER-encodes to the following bytes:

23h/03h ; constructed bit string (tag bit 5 = 1), length = 3
03h/01h/07h ; simple bit string, length = 1, 7 unused bits, no data

Normal Kerberos packets already have bit strings available, but to get LSASS to accept a bit string within SPNEGO, it takes just a bit of crafting. Providing a NegTokenInit token (tag A0h) containing a ContextFlags value (tag A1h), then we can pass a bit string that does get decoded using the vulnerable function.
(See RFC 2478 Section 3.2.1 for more details.)

This leaves us with the byte sequence below:

A0h/09h ; NegotiationToken: negTokenInit, length = 9
30h/07h ; sequence, length = 7
A1h/05h ; reqFlags (ContextFlags), length = 5
23h/03h ; constructed bit string, length = 3
03h/01h/07h ; simple bit string, length = 1, 7 unused bits, no data

Vendor Status:

Microsoft has released both a bulletin and an update to solve both issues.

It is available at

<<http://www.microsoft.com/technet/security/bulletin/MS04-007.asp>>
<http://www.microsoft.com/technet/security/bulletin/MS04-007.asp>.

ADDITIONAL INFORMATION

The information has been provided by <<mailto:info@eEye.com>> eEye Digital Security.

=====

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

=====

=====

Securiteam: [NT] Microsoft ASN.1 Library Length Overflow And Bit String Heap Corruption

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.