[48Bits Advisory] QuickTime Panorama Sample Atom Heap Overflow
Mario Ballano Bárcena (mballano[_at_]gmail.com)

Abstract

QuickTime is prone to a heap overflow vulnerability when parsing malformed Panorama Sample Atoms, which are used in QuickTime Virtual Reality Movies. This Vulnerability allows attackers to execute code on vulnerable installations. Successful exploitation via Web Browser requires that the attacker should trick the user into visiting a specially crafted webpage.

Affected versions :

Tested with QuickTime VR extension 7.2.0.240 included with QuickTime Player 7.2

Analysis :

QuickTime file format structure :

QuickTime files are built using atom containers, which are the basic unit of information. Each Atom has an atom header containing the following fields:

- Atom Size. (uint32)
- Type. (uint32)
- Extended size. (uint64, not always present)

Before the header each atom has its own structure.

QuickTime VR technology :

From Wikipedia :

"QuickTime VR (virtual reality) (also known as QTVR) is a type of image file format supported by Apple's QuickTime. It allows the creation and viewing of photographically captured panoramas and the exploration of objects through images taken at multiple viewing angles. It functions as a plugin for the standalone QuickTime Player, as well as working as a plugin for the QuickTime Web browser plugin. QuickTime VR will play on Windows computers as well as Apple Macintosh computers.

Apple continues to include QuickTime VR in its QuickTime technology. Many software companies create authoring applications to create QuickTime VR content."

Panorama Sample Atom structure :

From Apple's QuickTime File Format Reference :

"A panorama sample atom has an atom type of kQTVRPanoSampleDataAtomType ('pdat'). It describes a single panorama, including track reference indexes of the scene and hot spot tracks and information about the default viewing angles and the source panoramic image. The structure of a panorama sample atom is defined by the QTVRPanoSampleAtom data type:

```
typedef struct VRPanoSampleAtom {
   UInt16 majorVersion;
   UInt16 minorVersion;
   UInt32 imageRefTrackIndex;
   UInt32 hotSpotRefTrackIndex;
   Float32 minPan;
   Float32 maxPan;
   Float32 minTilt;
   Float32 maxTilt;
   Float32 minFieldOfView;
   Float32 maxFieldOfView;
   Float32 defaultPan;
   Float32 defaultTilt;
   Float32 defaultFieldOfView;
   UInt32 imageSizeX;
   UInt32 imageSizeY;
   UInt16 imageNumFramesX;
   UInt16 imageNumFramesY;
   UInt32 hotSpotSizeX;
   UInt32 hotSpotSizeY;
   UInt16 hotSpotNumFramesX;
   UInt16 hotSpotNumFramesY;
   UInt32 flags;
   OSType panoType;
   UInt32 reserved2;
} VRPanoSampleAtom, *VRPanoSampleAtomPtr; "
```

Disassemblies and analysis:

QuickTimeVR.qtx parses VR Atoms :

```
.text:67635DC0              mov     ebp, [edi+0C2h]
.text:67635DC6              push    ebp
.text:67635DC7              mov     [esp+40h+var_18], ebp
.text:67635DCB              call    sub_6765DA90
.text:67635DD0              push    0
.text:67635DD2              push    1
.text:67635DD4              push    'pdat'        ; <- Panorama Sample Atom
.text:67635DD9              push    0
.text:67635DDB              push    ebp
.text:67635DDC              call    GetAtom
.text:67635DE1              mov     ebx, eax
.text:67635DE3              add     esp, 18h
.text:67635DE6              test    ebx, ebx
.text:67635DE8              jz      loc_67636775
.text:67635DEE              push    0
.text:67635DF0              lea     edx, [esp+40h+arg_0]
.text:67635DF4              push    edx
.text:67635DF5              push    ebx
.text:67635DF6              push    ebp
.text:67635DF7              call    QT_dispatch_f1
.text:67635DFC              add     esp, 10h
.text:67635DFF              test    ax, ax
.text:67635E02              mov     [esp+3Ch+var_28], eax
.text:67635E06              jnz     loc_6763677D
.text:67635E0C              mov     eax, [esp+3Ch+arg_0] <- Size of Atom
.text:67635E10              push    0
.text:67635E12              push    esi
.text:67635E13              push    eax
.text:67635E14              push    0
.text:67635E16              push    ebx
.text:67635E17              push    ebp
.text:67635E18              call    QT_dispatch_f2  ; ends copying Atom with a funcion inside QuickTime.qts with wrong size
```

The atom gets copied using a wrapper for an statically compiled memcpy function inside QuickTime.qts :

```
.text:668DB8A0 ; int __cdecl memcpy_wrapper(void *Src,void *Dst,size_t Size)
.text:668DB8A0 memcpy_wrapper  proc near
.text:668DB8A0
.text:668DB8A0
.text:668DB8A0 Src           = dword ptr  4
.text:668DB8A0 Dst           = dword ptr  8
.text:668DB8A0 Size          = dword ptr  0Ch
.text:668DB8A0
.text:668DB8A0              mov     eax, [esp+Size]
.text:668DB8A4              mov     ecx, [esp+Src]
.text:668DB8A8              mov     edx, [esp+Dst]
.text:668DB8AC              push    eax             ; Size
.text:668DB8AD              push    ecx             ; Src
.text:668DB8AE              push    edx             ; Dst
.text:668DB8AF              call    _memcpy
.text:668DB8B4              add     esp, 0Ch
.text:668DB8B7              retn
.text:668DB8B7 memcpy_wrapper  endp
```

Destination buffer has a fixed size, we can build a malformed Panorama Sample Atom specifying a big value in its size field at the header, resulting in a heap corruption when atom gets copied.

There is a problem when trying to exploit it, because the heap gets overwritten with random data values, perhaps some kind of heap feng-shui could allow a smart way to control this, but I think exploitation could be more reliable using other way :

Within the values we can ovewrite in the heap we can control a pointer which will be used later as parameter for a statically compiled free() :

```
.text:668DC4E0 free           proc near
.text:668DC4E0
.text:668DC4E0
.text:668DC4E0 arg_0          = dword ptr  4
.text:668DC4E0
.text:668DC4E0                mov     eax, [esp+arg_0]
.text:668DC4E4                test    eax, eax
.text:668DC4E6                jnz     short not_null
.text:668DC4E8                mov     ax, 65427
.text:668DC4EC                retn
.text:668DC4ED ; ---------------------------------------------------------------------------
.text:668DC4ED
.text:668DC4ED not_null:
.text:668DC4ED                mov     edx, [eax+4]    ; edx = lpmem->snd_field
.text:668DC4F0                xor     ecx, ecx
.text:668DC4F2                cmp     edx, eax        ; edx == edx->snd_filed ?
.text:668DC4F4                setnz   cl
.text:668DC4F7                push    esi
.text:668DC4F8                dec     ecx
.text:668DC4F9                and     ecx, eax
.text:668DC4FB                mov     esi, ecx
.text:668DC4FD                jz      short loc_668DC537
.text:668DC4FF                test    byte ptr [esi+0Dh], 8
.text:668DC503                jnz     short loc_668DC537
.text:668DC505                push    esi
.text:668DC506                call    sub_668DBDB0
.text:668DC50B                mov     eax, [esi]
.text:668DC50D                add     esp, 4
.text:668DC510                test    eax, eax
.text:668DC512                jz      short loc_668DC529
.text:668DC514                push    eax
.text:668DC515                call    sub_668DBAB0
.text:668DC51A                mov     edx, [esi+8]
.text:668DC51D                mov     eax, [esi]
.text:668DC51F                push    edx             ; int
.text:668DC520                push    eax             ; lpMem
.text:668DC521                call    sub_668DB590
.text:668DC526                add     esp, 0Ch
.text:668DC529
.text:668DC529 loc_668DC529:
.text:668DC529                push    esi
.text:668DC52A                call    sub_668DB5E0
.text:668DC52F                add     esp, 4
.text:668DC532                xor     ax, ax
.text:668DC535                pop     esi
.text:668DC536                retn
.text:668DC537 ; ---------------------------------------------------------------------------
.text:668DC537
.text:668DC537 loc_668DC537:
.text:668DC537
.text:668DC537                mov     ax, 0FF91h
.text:668DC53B                pop     esi
.text:668DC53C                retn
.text:668DC53C free           endp
```

This pointer is located at AtomStart + 0x9C . In order to exploit this we must use a specially crafted structure and overwrite this pointer with its address. I've done it using heap spraying as a Proof of Concept.

Attack Vectors :

- Quicktime WebBrowser Plugin (remote!)
- Quicktime Player
- Quicktime PictureViewer

References:

- QuickTime File Format : http://developer.apple.com/documentation/QuickTime/QTFF/index.html
- QuickTime VR Technology (Apple) : http://www.apple.com/quicktime/technologies/qtvr/
- QuickTime VR Technology (Wikipedia) : http://en.wikipedia.org/wiki/QuickTime_VR
- iDefense Advisory : http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=620
- CVE : http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4675
- Apple Advisory : http://docs.info.apple.com/article.html?artnum=306896

Credit:

Analysis performed and vulnerability discovered by Mario Ballano Bárcena from 48bits.com