Exposing Vulnerabilities in Media Software

David Thiel, iSEC Partners

BlackHat EU 2008
Introduction

Overview

Containers and Codecs

Fuzzbox

Fuzzing Techniques

Case study: Ogg-Vorbis

Other formats and features

Fallout

Finding root causes

Collateral damage and future directions

Summary
Hello
- I’m a consultant and researcher with iSEC Partners
- Focus on application security
- Audio hobbyist

What’s this all about?
- The attack surface and potential of media codecs, players and related devices
- Focus here is on slightly on audio, but that doesn’t matter
- Video works the same way, and uses the same container formats

Takeaways
- Understand attack surface and implications
- Understand how to fuzz and design fuzzers for media
- Help developers understand how to improve code
- Plant ideas for future research
Why this matters

- Omnipresent and always on
  - Promiscuously shared, played, streamed
  - Comes from extremely untrusted, often anonymous sources
  - Who thinks to refrain from playing “untrusted” media?
  - Most browsers will play automatically anyhow

- It’s political
  - There are people out there who don’t like you stealing music
  - Like me, for example
  - But mostly I mean the RIAA, and companies like Sony
  - Exploits here are ripe for corporate abuse - it’s happened before

- It’s “rich”
  - Media playback/parsing software is almost by definition excessively functional
  - Does tons of parsing
Modern codecs are designed to be resistant to corruption
- Bit-flipping an Ogg file, for example, will usually not work
- Example: zzuf, a popular bit-flipping fuzzer, noted VLC as being “robust” against fuzzing of Vorbis, Theora, FLAC
- As zzuf notes, this does not mean there are no bugs; we just need a targeted fuzzer

Most media software exploits thus far have been simple
- Attacks on players: long playlists, URL names, etc.
- Few attacks using media files themselves
- Even fewer targeting things on the codec level
Containers and Codecs

- Container formats organize multiple types of media streams and metadata
  - “tags”—content describing end-user relevant data
  - subtitles
  - sync data, frame ordering
  - management of separate bitstreams

- Codec data describes and contains the actual video/audio
  - sample rate
  - bitrate
  - channels
  - compressed or raw media data
Containers and Codecs

- Examples of media containers:
  - AVI
  - Ogg
  - MPEG-2
  - MP4
  - ASF

- Examples of media codecs:
  - DivX
  - Vorbis
  - Theora
  - WMV
  - Xvid
  - Sorenson
(re-)Introducing Fuzzbox

- A multi-codec audio stream fuzzer, written in Python
- Targets specific stream formats, no general file fuzzing
- Uses third party libs like py-vorbis and mutagen for metadata fuzzing
- Uses built-in frame parsing for frame fuzzing
- *Not* another fuzzing framework
- An example of a real-world fuzzer used in pen-testing: quick, dirty and targeted
- Available at https://www.isecpartners.com/tools.html
What to fuzz

Two main areas are important here

- Content metadata
  - ID3, APEv2, Vorbis comments, album art, etc.
  - Because many types allow arbitrarily large content, this is a great place to store shellcode with plenty of NOP cushion—even if the bug isn’t in metadata parsing

- Frame data
  - We’re mostly interested in the frame header
  - Contains structural data describing overall file layout: sample rate, number of frames, frame size, channels
  - Can be multiple types of frame headers in a file, especially in the case of container formats
What to fuzz it with

- Obviously, random strings
  - Repeating one random ASCII char to help us spot stack pointer overwrites
  - Throw in some random unicode, encoded in funny ways
  - Just a bunch of “%n”s to give us some memory corruption
  - Random signed ints
- Format strings
- Fencepost numbers
- HTML! More on this later
- URLs—for catching URL pingbacks
How to fuzz it

- Three possible approaches
  - Reach in and just mangle
    - Might work, might not
    - Works a sad amount of the time
  - Use existing parsing libraries
    - Works well, but usually requires patching the libs
    - Built-in error handling will obviously trip us up
    - Metadata editing libraries don’t always allow changing of data we want
    - Let’s use this for basic stuff like ID3 tags and Vorbis comments
  - Make your own frame parser
    - Sometimes quick and easy, sometimes painful
    - But turns up some great bugs
The fuzzer’s toolbox
A few tools to make fuzzing and parsing easier

- hachoir: Dissects many file types visually
- mutagen: Help in mangling audio tags and understanding file layout
- vbindiff: Shows differences between fuzzed and non-fuzzed files
- bvi: A hex editor with keybindings similar to a certain one true editor
- bbe: sed for binary streams
- gdb: Love it or hate it, it’s all you get
Case study: Ogg-Vorbis

- Excellent free codec
- Well documented
- Not just for hippies
- Unencumbered status gets it into many things
- Consists of an Ogg container...

---

Ogg frame structure

<table>
<thead>
<tr>
<th>Bit</th>
<th>Segment Table</th>
<th>0-7</th>
<th>8-15</th>
<th>16-23</th>
<th>24-31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Capture Pattern</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Header Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Granule Position</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bitstream Serial Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page Sequence Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page Segments</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

The diagram above shows the structure of an Ogg frame, starting with the bit positions and segments.
Case study: Ogg-Vorbis

Vorbis frame structure

...and a Vorbis center

Also “Vorbis comments”

- Simple name/value pairs—can be any length or content, but some have special meaning
- Easiest to use existing libs for this—in this case, py-vorbis
Case study: Ogg-Vorbis

Vorbis comment structure

Typical tags used in Vorbis comments:

```python
comments = {
    # these are the most commonly used tags by vorbis apps.
    'COMMENT': 'leetleet',
    'TITLE': 'safety short',
    'ARTIST': 'Various',
    'ALBUM': 'Comp',
    'TRACKNUMBER': '1',
    'DISCNUMBER': '1',
    'GENRE': 'Experimental',
    'DATE': '2006',
    'REPLAYGAIN_TRACK_GAIN': 'trackgain',
    'REPLAYGAIN_ALBUM_GAIN': 'albumgain',
    'REPLAYGAIN_TRACK_PEAK': 'trackpeak',
    'REPLAYGAIN_ALBUM_PEAK': 'albumpeak',
    'LICENSE': 'Free as in beer',
    'ORGANIZATION': 'iSEC',
    'DESCRIPTION': 'A test file',
    'LOCATION': 'SF',
    'CONTACT': 'david@isecpartners.com',
    'ISRC': '12345'
}

vcomments = ogg.vorbis.VorbisComment(comments)
```
Case study: Ogg-Vorbis
Ogg and Vorbis frame data in Python

Mercifully 8-bit aligned—Vorbis portion starts at “12version”
Case study: Ogg-Vorbis
Comments and frame data loaded, feed to fuzzer

Transforms are defined in randjunk.py:

```python
import random

def randstring():
    thestring = ""
    chance = random.randint(0, 8)
    print "using method " + str(chance)
    if chance == 0:
        # try a random length of one random char
        char = chr(random.randint(0, 255))
        length = random.randint(0, 3000)
        thestring = char * length
        # or a format string
    elif chance == 1:
        thestring = "abcdefghijklmnopqrstuvwxyz" * 5
    elif chance == 2:
        # some garbage ascii
        for i in range(random.randint(0, 3000)):
            char = '\n'
            while char == '\n':
                char = chr(random.randint(0, 127))
            thestring += char
    elif chance == 3:
        # build up a random string of alphanumerics
```
Comments just write back in. Frame data needs to be packed:

```python
thestring = ""
letsfuzz = random.choice(y.keys())
print "fuzzing %s"%letsfuzz

thestring = randstring()
stringtype = type(thestring)
length = len(y[letsfuzz])
if str(stringtype) == "<type 'str'>":
    y[letsfuzz] = struct.pack('s', thestring[:length])
elif str(stringtype) == "<type 'int'>":
    y[letsfuzz] = struct.pack('i', thestring)
else:
    thestring = ""
    for i in range(len(y[letsfuzz])):
        thestring += "\x"%random.randint(0,15)

return y, restoffile
```
Case study: Ogg-Vorbis
Fixing the CRC

Every Ogg frame has a CRC to prevent corruption. Also hides bugs, but easy enough to fix:

```python
from optparse import OptionParser

vcomments = ogg.vorbis.VorbisComment(comments)

totaltags = len(vcomments)

# this is to reset the CRC after mangling of the header.
def ogg_page_checksum_set(page):
    crc_reg = 0

    # This excludes the CRC from being part of the new CRC.
    page = page[0:22] + '\x00\x00\x00\x00' + page[26:]

    for i in range(len(page)):
        crc_reg = ((crc_reg<<8) & 0xffffffff) ^ crc_lookup[((crc_reg >> 24) & 0xff) ^ ord(page[i])]

    # Install the CRC.
    page = page[0:22] + struct.pack('I', crc_reg) + page[26:]

    return page
```
Other supported formats

- **FLAC**
  - Lossless audio—uses Vorbis comments for metadata, can use Ogg as a container (and usually does)

- **MP3**
  - Metadata with ID3
  - ID3v1
    - Length limited
    - Stored at end of file
    - Great for rewriting, awful for streaming
  - ID3v2
    - Massively structured and complex
    - Incompletely supported
    - Obsessively detailed
    - I hope it dies
Example: ID3v2’s OCD

```
<Header for 'Attached picture', ID: "APIC">
  Text encoding       $xx
  MIME type           <text string> $00
  Picture type        $xx
  Description         <text string according to encoding> $00 (00)
  Picture data        <binary data>

  Picture type: $00 Other
                 $01 32x32 pixels 'file icon' (PNG only)
                 $02 Other file icon
                 $03 Cover (front)
                 $04 Cover (back)
                 $05 Leaflet page
                 $06 Media (e.g. label side of CD)
                 $07 Lead artist/lead performer/soloist
                 $08 Artist/performer
                 $09 Conductor
                 $0A Band/Orchestra
                 $0B Composer
                 $0C Lyricist/text writer
                 $0D Recording Location
                 $0E During recording
                 $0F During performance
                 $10 Movie/video screen capture
                 $11 A bright coloured fish
                 $12 Illustration
                 $13 Band/artist logotype
                 $14 Publisher/Studio logotype
```
Even more supported formats

- WAV and AIFF
  - What's to attack in “raw” audio?
  - Not a lot, but it still works
  - Sample width, framerate, frame number; all things that can expose integer bugs
  - WAV and AIFF parsing libraries are included with Python

- Speex
  - Optimized for speech
  - Used in several high-profile third-party products
  - Uses Vorbis comments for metadata
  - Often stored in an Ogg container
And yet more formats

- **MP4**
  - Often used for AAC, but can also contain many other video and audio types
  - Comprised of a series of FOURCC “atoms”
  - Combines functionality of tags/comments and lower level descriptions like sample rate, positional info
  - In true Apple fashion, not officially documented
Setting up a fuzzer run
Basic usage of Fuzzbox

```
[lx@dt apps/fuzzers/fuzzbox 669 ] python ./fuzzbox.py
ERROR: You need to define at least the source file.
usage: fuzzbox.py [options]

options:
  --version  show program's version number and exit
  -h, --help  show this help message and exit
  -r REPS, --reps=REPS  Number of files to generate/play
  -p PROGNAME, --program=PROGNAME  Path to the player you'd like to test
  -l LOGFILE, --logfile=LOGFILE  Path to the log file to record results
  -s SOURCEFILE, --source=SOURCEFILE  Path to a source file to fuzz
  -t TIMEOUT, --timeout=TIMEOUT  How long to wait for the player to crash
  --itunes  Work around iTunes anti-debugging
  --filetype=FILETYPE  Type of file to fuzz: wav, aiff, mp3 or ogg
```

[lx@dt apps/fuzzers/fuzzbox 669 ]
Nifty Fuzzbox features

- Autoplay mode—spawns a player of your choice under gdb
- Gathers backtraces, registers and resource usage
- Kills off runaway apps
- iTunes anti-anti-debugging
- iTunes automation with AppleScript
Introduction

Overview

Containers and Codecs

Fuzzbox

Fuzzing Techniques

Case study: Ogg-Vorbis

Other formats and features

Fallout

Finding root causes

Collateral damage and future directions

Summary

iTunes-specific functionality

Avoiding iTunes anti-debugging

Simply jump around PT_DENY_ATTACH with gdb¹:

```python
def playit(filename, timeout):
    log = open(logfile, "a")
gdbfile = open("/tmp/gdbparams", "w")
gdbfile.write("set args %s\n"%filename)
if itunes == True:
    gdbfile.write("break pttrace if $r3 == 31\n")
gdbfile.write("run\n")
gdbfile.write("bt\n")
if itunes == True:
    gdbfile.write("return\n")
gdbfile.write("cont\n")
gdbfile.write("bt\n")
gdbfile.write("info reg\n")
gdbfile.write("quit\n")
gdbfile.close()
# this is stupid. stdin=None causes the program to suspend
# when gdb is killed.
devnull = open("/dev/null", "r")
log.write("---- Playing %s\n"%filename)
gdb = Popen(['gdb', "--batch", "-x", "/tmp/gdbparams", progsname], stdin=devnull, stdout=log, stderr=log)
if itunes == True:
    os.system("osascript -e 'tell application "iTunes" to play'\n")
```

¹http://www.steike.com/code/debugging-itunes-with-gdb/
Fallout: VLC
Format string issues in Vorbis comments (CVE-2007-3316)

Also CDDA, SAP/SDP—broadcast exploitation!
Fallout: libvorbis
Bug in invalid mapping type handling (CVE-2007-4029)

Function pointer to an invalid memory address offset by an attacker-controlled value

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x8063000 (LWP 100138)]
0x280a6c14 in vorbis_info_clear (vi=0x805a260) at info.c:165
165 _mapping_P[ci->map_type[i]]->free_info(ci->map_param[i]);
(gdb) bt
#0 0x280a6c14 in vorbis_info_clear (vi=0x805a260) at info.c:165
#1 0x280a758c in _vorbis_unpack_books (vi=0x805a260, op=0xbfbfe770) at info.c:327
#2 0x280a770f in vorbis_synthesis_headerin (vi=0x805a260, vc=0x805c440, op=0xbfbfe770) at info.c:380
#3 0x2808d1ef in _fetch_headers (vf=0x806f000, vi=0x805a260, vc=0x805c440, serialno=0x806f05c, og_ptr=0xbfbfe790) at vorbisfile.c:262
#4 0x2808dfab in _ov_open1 (f=0x8066180, vf=0x806f000, initial=0x0, ibytes=0, callbacks=
   {read_func = 0x805058c <vorbisfile_cb_read>, seek_func = 0x80505b8 <vorbisfile_cb_seek>, close_func = 0x80505e4 <vorbisfile_cb_close>, tell_func = 0x80505f0 <vorbisfile_cb_tell>}) at vorbisfile.c:666
#5 0x2808e266 in _ov_open_callbacks (f=0x8066180, vf=0x806f000, initial=0x0, ibytes=0, callbacks=
   {read_func = 0x805058c <vorbisfile_cb_read>, seek_func = 0x80505b8 <vorbisfile_cb_seek>, close_func = 0x80505e4 <vorbisfile_cb_close>, tell_func = 0x80505f0 <vorbisfile_cb_tell>}) at vorbisfile.c:731
#6 0x80501d4 in ovf_init (source=0x805c430, ogg123_opts=0x8059840, audio_fmt=0xbfbfe8b0, callbacks=0xbfbfe8d8, callback_arg=0x8096000)
Fallout: flac-tools
Stack overflow in metadata parsing, flac123 (CVE-2007-3507)

```
Starting program: /crypt/usr/local/bin/flac123 27272727flac123.flac
flac123 version 0.0.9  'flac123 --help' for more info

Program received signal SIGSEGV, Segmentation fault.
0x27272727 in ?? ()
(gdb) bt
#0 0x27272727 in ?? ()
#1 0x0804a811 in local_vcentry_matches (field_name=0x804afaf "artist", entry=0x8268038) at vorbiscomment.c:32
#2 0x0804a9ac in get_vorbis_comments (filename=0xbfbf631 "27272727flac123.flac") at vorbiscomment.c:69
#3 0x08049564 in print_file_info (filename=0xbfbf631 "27272727flac123.flac") at flac123.c:121
#4 0x08049a97 in decoder_constructor (filename=0xbfbf631 "27272727flac123.flac") at flac123.c:245
#5 0x08049b2d in play_file (filename=0xbfbf631 "27272727flac123.flac") at flac123.c:269
#6 0x08049520 in main (argc=2, argv=0xbfbf69fc) at flac123.c:108
(gdb) up
#1 0x0804a811 in local_vcentry_matches (field_name=0x804afaf "artist", entry=0x8268038) at vorbiscomment.c:32
   32 const FLAC__byte *eq = memchr(entry->entry, '=', entry->length);
```
Fallout: iTunes

- Heap overflow in “COVR” MP4 atom parsing (CVE-2007-3752)
- Normally used for album art, but works for arbitrary code execution too
Note about static analysis

- At least one of these vendors was actually using a commercial static analysis tool
- It missed all of the bugs found with Fuzzbox
- These tools are useful, but not a complete solution
- Fuzzing is necessary too—and cheaper
Finding root causes

Checking diffs between source file and crasher, We can see the difference in CRC and one other byte:
Finding root causes

Located just after the Vorbis version—a silly number of audio channels
Finding root causes

- With the cause identified, you can start manipulating rather than fuzzing
- Play with the value in a hex editor or with bbe
- In the case of Ogg, the included oggcrc.py will recalculate CRC after editing
Non-player apps, or “nobody uses Vorbis!”
- As mentioned before, some of these codecs get around
- Used in games—custom sounds downloaded with maps...
- Asterisk does—(O_o);
  - It also supports Speex, which is structurally very similar...
  - In other words, any DoS or code execution in Ogg/Vorbis or Speex can mean the same for Asterisk

Also potential for VOIP-related attacks in WAV/PCM modules
- Good potential for active network attacks; see RTPInject (Lackey, Garbutt)

Embedded devices!
- My phone plays lots of audio and video formats.
- So do a bunch of other portables, in-car systems, home multimedia devices...
Collateral damage and future directions

- Total speculation: indexing services and other parsers
  - Some software (e.g. Beagle) relies on media libraries to index
  - Exploits in these libraries affect the indexer
  - Can also be a venue for finding bugs in the indexer itself
  - Or its web interface
  - Some apps aren’t real careful about data parsed from media
  - Cool for CSRF, XSS or JavaScript intranet scanning, etc.
Collateral damage and future directions
Example: phpMp, frontend for the Music Player Daemon

Collateral damage and future directions
Example: phpMp, frontend for the Music Player Daemon
Demo
flac code execution
• Vendors should fuzz their own software.
• Users should treat media streams as potentially malicious content.
• Use, but don’t rely on, source analysis.
Questions?

- Thanks for coming!
- Thanks to:
  - Chris Palmer, Jesse Burns, Tim Newsham
  - Xiph.org, the VLC team and Apple product security

  david@isecpartners.com