Exploiting Vulnerabilities in Media Software
Agenda

• Introduction
  – Why media software?
  – Why bugs are still out there
  – How we're going to bang them out

• Fuzzing techniques
  – Why/What/How
  – Fuzzbox

• Codecs to attack
  – Ogg Vorbis
  – MP3
  – FLAC
  – Speex
  – Raw formats: PCM/WAV, AIFF
Agenda

- Case studies: blown up software
- Demo
- Q&A
Introduction

• Hello
  – I'm a consultant for iSEC Partners
  – Focus on application security
  – UNIX grump
  – Audio hobbyist

• What's this all about?
  – The attack surface and potential of media codecs
  – Focus here is on audio, but that doesn't matter
  – Video works the same way, and uses the same container formats
Why this matters

• Omnipresent, and always on
  – Promiscuously shared, played, streamed
  – Come from extremely untrusted, often anonymous sources
  – Who thinks to refrain from playing “untrusted” sounds?
  – 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
  – Ripe for corporate abuse

• It's “rich”
  – Media playback software is excessively functional
  – Does tons of parsing

• It's underexplored!

iSEC Partners
https://www.isecpartners.com
Why underexplored?

• 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 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
Fuzzing techniques: what to fuzz

- Two main areas are important here
  - Content metadata
    - ID3, APEv2, Vorbis comments, album art, etc.
  - 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
Fuzzing techniques: what to fuzz 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
  – Format strings
  – Just a bunch of %ns to give us some memory corruption
  – Random signed ints
  – Fencepost numbers
• HTML! More on this later.
• URLs – maybe we can catch some URL pingbacks
Fuzzing techniques: how to fuzz it

• Three possible approaches
  – Reach in and just mutate
    • 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 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
  • gdb
Fuzzbox

- A multi-codec audio stream fuzzer, written in Python
- Targets specific codecs, 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 real-world fuzzers used in pen-testing: quick, dirty and targeted
Ogg Frame Structure

- 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:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 | Byte
+-----------------------------------------------| 0-3
| capture_pattern: Magic number for page start "OggS" |
+-----------------------------------------------|
| version | header_type | granule_position | 4-7 |
+-----------------------------------------------| |
| bitstream_serial_number | page_sequence_number | 12-15 |
+-----------------------------------------------|
| CRC_checksum | 20-23 |
+-----------------------------------------------|
| page_segments | segment_table | 24-27 |
+-----------------------------------------------|
| ... | 28- |
+-----------------------------------------------|
```
Vorbis Frame Structure

- ...with a chewy Vorbis center
  - Contains channels, sample rate, etc
  - 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

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

vcomments = ogg.vorbis.VorbisComment(comments)
```
Ogg and Vorbis frame in Python

- Mercifully 8-bit aligned

```python
y = {}
#
# Ogg structure
y['01magic'] = f.read(4)
y['02version'] = f.read(1)
y['03headertype'] = f.read(1)
y['04granulepos'] = f.read(8)
y['05serial'] = f.read(4)
y['06pageseq'] = f.read(4)
y['07crc'] = f.read(4)
y['08numsegments'] = f.read(1)
y['09segtable'] = f.read(1)
y['10packettype'] = f.read(1)
y['11streamtype'] = f.read(1)
y['12version'] = f.read(1)
y['13channels'] = f.read(1)
y['14samprate'] = f.read(4)
y['15maxbitrate'] = f.read(4)
y['16nominalbitrate'] = f.read(4)
y['17minbitrate'] = f.read(4)
y['18blocksize'] = f.read(1)
```

# should be 58 bytes
headerlength = f.tell()
Data loaded, feed to fuzzer

- Now we have comments and frame data
- Time to mangle them up
- Transforms are defined in randjunk.py:

```python
import random

def randstring():
    thestring = ""
    chance = random.randint(0, 0)
    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 = "\n\n\n\n\n\n"
    elif chance == 2:
        # some garbage ascii!
        for i in range(random.randint(0, 30000)):
            char = '\n'
            while char == '\n':
                char = chr(random.randint(0, 127))
        thestring += char
    elif chance == 3:
        # build up a random string of alphanumerics
```

24.14.35 9%
Data fuzzed, writing back out

• In the case of comments, we just write them back in
• For our frame data, we need to pack it:

```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('a', theestring[:length])
elif str(stringtype) == "<type 'int'>":
    y[letsfuzz] = struct.pack('1', theestring)
else:
    theestring = ""
    for i in range(len(y[letsfuzz])):
        theestring += "%x" % random.randint(0, 15)

return y, restoffile
```
Fix 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

- **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
    - I hope it dies
- **FLAC**
  - Lossless audio – uses Vorbis comments for metadata, can use Ogg as a container
Even more supported formats

- WAV and AIFF
  - What's to attack in raw audio?
  - Not much, 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
  - Can be stored in an Ogg container
Setting up a fuzzer run

- Basic usage of fuzzbox

```bash
[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
- `--help`     show this help message and exit
- `--reps=REPS` Number of files to generate/play
- `--program=PROGRAM` Path to the program you'd like to test
- `--logfile=LOGFILE` Path to the logfile to record results
- `--source=SOURCEFILE` Path to a source file to fuzz
- `--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

```bash
[lx@dt apps/fuzzers/fuzzbox 669 ]
```
Demo
Nifty features

• Autoplay mode – kicks off a player of your choice under gdb
• Gathers backtraces, registers and resource usage
• iTunes anti-anti-debugging
• iTunes automation with AppleScript
• Kills off runaway apps
Fallout: VLC

- Format string issues in Vorbis comments
  - Also CDDA, SAP/SDP – broadcast exploitation!

```c
Breakpoint 2, 0x28469625 in vsprintf () from /lib/libc.so.6
(gdb) where
#0 0x28469625 in vsprintf () from /lib/libc.so.6
#1 0x80d1d93 in input.vaControl (p_input=0x87d4000, i_query=142491908,
     args=0x87cbbcc "\n\n\n\n\n\n\n\n\n\n\n")
at input/control.c:192
#2 0x80d3a9b in input.Control (p_input=0x87e4104, i_query=142491908)
at input/control.c:56
#3 0x294c6825 in DecodeBlock (p_dec=0x87b1800, pp_block=0xbf1f6f84)
at vorbis.c:625
#4 0x80d4eaa in DecoderDecode (p_dec=0x87b1800, p_block=0x87db300)
at input/decoder.c:662
#5 0x80db5d85 in DecoderThread (p_dec=0x87b1800) at input/decoder.c:494
#6 0x284281e8 in pthread_create () from /lib/libpthread.so.2
#7 0x2841f93 in __ctx_start () from /lib/libc.so.6

(gdb) delete 2
(gdb) cont
Continuing.
[New Thread 0x9418000 (LWP 100189)]
Program received signal SIGSEGV, Segmentation fault.

[Switching to Thread 0x9418000 (LWP 100189)]
0x28502243 in __vfprintf () from /lib/libc.so.6
```
Fallout: libvorbis

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 0x2808e758c in _vorbis_unpack_books (vi=0x805a260, opb=0xbfbfe710) at info.c:327
#2 0x2808e770f 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, ug_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 0x2808e206 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
Demo
Collateral Damage

• 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 means the same for Asterisk

• Web applications
  – Some apps aren't real careful about data parsed from media
  – Cool for CSRF, XSS or Javascript intranet scanning

• Indexing services and other parsers
  – Software like 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
phpMp

---

**Current Directory**

Music

**Directories:**
- Music
- index

**Music (add to):**

<table>
<thead>
<tr>
<th>Artist</th>
<th>Title</th>
<th>Album</th>
<th>Track</th>
</tr>
</thead>
<tbody>
<tr>
<td>moo</td>
<td>moo-ogg</td>
<td>The Coast</td>
<td></td>
</tr>
<tr>
<td>moo</td>
<td>moo-ogg</td>
<td>The Coast</td>
<td></td>
</tr>
<tr>
<td>moo</td>
<td>moo-ogg</td>
<td>The Coast</td>
<td></td>
</tr>
<tr>
<td>moo</td>
<td>moo-ogg</td>
<td>The Coast</td>
<td></td>
</tr>
<tr>
<td>moo</td>
<td>moo-ogg</td>
<td>The Coast</td>
<td></td>
</tr>
<tr>
<td>moo</td>
<td>moo-ogg</td>
<td>The Coast</td>
<td></td>
</tr>
</tbody>
</table>

**Playing Options**

- Stop
- Play
- Pause

**Volume:**

- 0%

**Playlist (shuffle|repeat|dual)**

- 0:00:00 sample count

---

iSEC Partners
https://www.isecpartners.com
Questions?

- Thanks for coming!
- Thanks to:
  - Chris Palmer, Jesse Burns, Tim Newsham

Q&A

david@isecpartners.com