Exposing Vulnerabilities in Media Software

David Thiel, iSEC Partners

March 31st, 2008

Agenda

Introduction

- Hello
  - I'm a consultant and researcher with iSEC Partners
  - Focus on application security
  - Audio hobbyist (definitely no expert)
- What's this all about?
  - The attack surface and potential of media codecs, players and related devices
  - Focus here is 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
  - Most don't think to refrain from playing "untrusted" media
  - And most browsers will play automatically anyhow
- It's political
  - There are people out there who don't like you stealing music: 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

Why media security is under-explored

- 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
- Deep research not historically necessary
  - Most media software exploits thus far have been simple: long playlists, URL names, etc.
  - Few attacks using media files themselves
  - Even fewer targeting things on the codec level
  - But this is changing as of late

Terminology: 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

Examples: Containers and Codecs

- Examples of media containers:
  - AVI
  - Ogg
  - MPEG-2
  - MP4
  - ASF
- Examples of media codecs:
  - DivX
  - Vorbis
  - Theora
  - WMV
  - Xvid
  - Sorenson

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
  - 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

What to fuzz it with

- Traditional random strings
  - Repeating one random ASCII char to help us spot stack pointer overwrites
  - Random unicode, encoded in funny ways
  - Random signed ints
- Bunch of “%n” format strings to give us some memory corruption
- Fencepost numbers
- HTML, more on this later
- URLs—for catching URL pingbacks
- Perhaps other injection types—SQL, XML

How to fuzz it

- Three possible approaches
  - Reach in and just mangle
    - Might work, might not
    - Works a sad amount of the time
    - But, misses a lot of attack classes
- Use existing parsing libraries
  - Works well, but usually requires patching the libs
  - Built-in error handling will trip us up
  - Metadata editing libraries don't always allow changing of data we want
  - 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

Fuzzbox

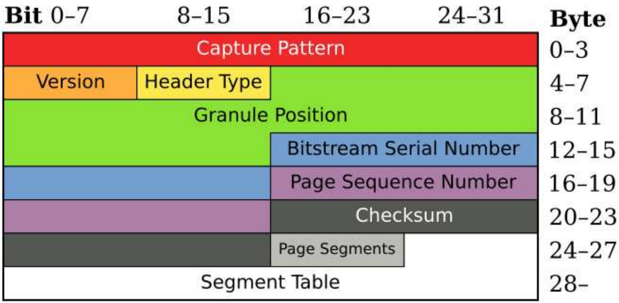
- A multi-codec audio/video stream fuzzer, written in Python
- Targets specific stream formats, no general file fuzzing
- Uses third party libs like py-vorbis and mutagen for metadata fuzzing
- Uses built-in frame parsing for frame fuzzing
- *Not* another fuzzing framework
- Just a real-world fuzzer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

Fuzzbox Supported filetypes

- Ogg
- FLAC
- ASF (*i.e.*, WMV, WMA)
- MP3
- Quicktime/MP4
- Speex
- WAV
- AIFF

Case study: Ogg-Vorbis
Ogg frame structure

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



- . . . 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

Typical tags used in Vorbis comments:

```
comments = {}

# these are the most commonly used tags by vorbis apps.
comments['COMMENT'] = 'leetleet'
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)
                                                    76,1          82%
```

iSEC PARTNERS

Case study: Ogg-Vorbis

Ogg and Vorbis frame data in Python

Mercifully 8-bit aligned—Vorbis portion starts at “12version”

```
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(6)
y['12version'] = f.read(4)
y['13channels'] = f.read(1)
y['14samplerate'] = 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()
                                                  155,0-1       25%
```

Transforms are defined in randjunk.py:

```
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 = "%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n"
        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
                                                        24,14-35        9%
```

Case study: Ogg-Vorbis

Data fuzzed, writing back out

Comments just write back in. Frame data needs to be packed:

```
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

                                                206,0-1          32%
```

Case study: Ogg-Vorbis
Fixing the CRC

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

```
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
~
~
                                                        36,0-1          Bot
```

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
    - Better for streaming, otherwise uniformly awful

Example: ID3v2's OCD

"APIC" picture attachment tag


```
  <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 (but need patched)
- 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
- ASF
  - Container format for MS WMA and WMV files
  - WMA used on portable devices, WMVs distributed widely online

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 logfile 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 ]
```

Demo

Nifty Fuzzbox features

- Autoplay mode—spawns a player of your choice under gdb
- Gathers backtraces, registers and resource usage
- Kills off runaway apps (excessive CPU/memory consumption, mangled play rate)
- iTunes anti-anti-debugging
- iTunes automation with AppleScript

Simply jump around PT_DENY_ATTACH with gdb[1]:

```
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 ptrace 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", progname], stdin=d
evnull, stdout=log, stderr=log)
        if itunes == True:
            os.system("""osascript -e 'tell application "iTunes" to play'""")
                                                              327,39-46     46%
```

[1] http://www.steike.com/code/debugging-itunes-with-gdb/

Results: VLC

Format string issues in Vorbis comments (CVE-2007-3316)

Also CDDA, SAP/SDP—broadcast exploitation!

```
Breakpoint 2, 0x28469625 in vasprintf () from /lib/libc.so.6
(gdb) where
#0  0x28469625 in vasprintf () from /lib/libc.so.6
#1  0x080d1d93 in input_vaControl (p_input=0x87d4000, i_query=142491908,
    args=0x87cbbcc "%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n")
    at input/control.c:192
#2  0x080d3aab in input_Control (p_input=0x87e4104, i_query=142491908)
    at input/control.c:50
#3  0x294d6825 in DecodeBlock (p_dec=0x87b1800, pp_block=0xbf1f6f84)
    at vorbis.c:625
#4  0x080d4eaa in DecoderDecode (p_dec=0x87b1800, p_block=0x87db300)
    at input/decoder.c:662
#5  0x080d5d85 in DecoderThread (p_dec=0x87b1800) at input/decoder.c:494
#6  0x28428168 in pthread_create () from /lib/libpthread.so.2
#7  0x284f1983 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
```

Results: 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, opb=0xbfbfe710)
    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  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  0x080501d4 in ovf_init (source=0x805c430, ogg123_opts=0x8059840,
    audio_fmt=0xbfbfe8b0, callbacks=0xbfbfe8d8, callback_arg=0x8096000)
```

Results: flac-tools

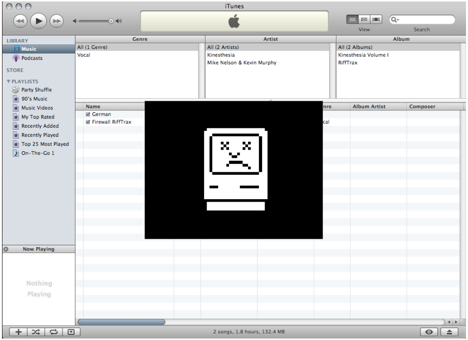
Overflow in metadata parsing, flac123 (CVE-2007-3507)


```
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=0xbfbfeb31 "27272727flac123.flac") at vorbiscomment.c:69
#3  0x08049564 in print_file_info (filename=0xbfbfeb31 "27272727flac123.flac")
    at flac123.c:121
#4  0x08049a97 in decoder_constructor (
    filename=0xbfbfeb31 "27272727flac123.flac") at flac123.c:245
#5  0x08049b2d in play_file (filename=0xbfbfeb31 "27272727flac123.flac")
    at flac123.c:269
#6  0x08049520 in main (argc=2, argv=0xbfbfe9fc) 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);
```

Results: iTunes

- Heap overflow in “COVR” MP4 atom parsing (CVE-2007-3752)
- Normally to store album art, can store evil 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:

```
08_safety_short.ogg
0000 0000: 4F 67 67 53 00 02 00 00  00 00 00 00 00 00 FA 80  OggS.... ........
0000 0010: E1 1B 00 00 00 00 8C 9F  4D 9F 01 1E 01 76 6F 72  ........ M....vor
0000 0020: 62 69 73 00 00 00 00 02  44 AC 00 00 00 00 00 00  bis..... D.......
0000 0030: 00 EE 02 00 00 00 00 00  B8 01 4F 67 67 53 00 00  ........ ..OggS..
0000 0040: 00 00 00 00 00 00 00 00  FA 80 E1 1B 01 00 00 00  ........ ........
0000 0050: BB 2F D9 B9 10 4E FF FF  FF FF FF FF FF FF FF FF  ./...N.. ........
0000 0060: FF FF FF FF 71 03 76 6F  72 62 69 73 1D 00 00 00  ....q.vo rbis....
0000 0070: 58 69 70 68 2E 4F 72 67  20 6C 69 62 56 6F 72 62  Xiph.Org  libVorb
0000 0080: 69 73 20 49 20 32 30 30  35 30 33 30 34 02 00 00  is I 200 50304...
ogg4.ogg
0000 0000: 4F 67 67 53 00 02 00 00  00 00 00 00 00 00 FA 80  OggS.... ........
0000 0010: E1 1B 00 00 00 00 A2 B2  2D 10 01 1E 01 76 6F 72  ........ -....vor
0000 0020: 62 69 73 00 00 00 00 42  44 AC 00 00 00 00 00 00  bis....B D.......
0000 0030: 00 EE 02 00 00 00 00 00  B8 01 4F 67 67 53 00 00  ........ ..OggS..
0000 0040: 00 00 00 00 00 00 00 00  FA 80 E1 1B 01 00 00 00  ........ ........
0000 0050: BB 2F D9 B9 10 4E FF FF  FF FF FF FF FF FF FF FF  ./...N.. ........
0000 0060: FF FF FF FF 71 03 76 6F  72 62 69 73 1D 00 00 00  ....q.vo rbis....
0000 0070: 58 69 70 68 2E 4F 72 67  20 6C 69 62 56 6F 72 62  Xiph.Org  libVorb
0000 0080: 69 73 20 49 20 32 30 30  35 30 33 30 34 02 00 00  is I 200 50304...

Arrow keys move   F find       RET next difference   ESC quit   T move top
C ASCII/EBCDIC    E edit file    G goto position       Q quit   B move bottom
```

Finding root causes

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

```
0) file:./ogg4.ogg: Ogg multimedia container (305.3 KB)
 + 0) page[0] (58 bytes)
      0) capture_pattern= "OggS" (4 bytes)
      4) stream_structure_version= 0 (1 byte)
      5.0) continued_packet= False (1 bit)
      5.1) first_page= True (1 bit)
      5.2) last_page= False (1 bit)
      5.3) unused= <null> (5 bits)
      6) abs_granule_pos= 0 (8 bytes)
      14) serial= 0x1be180fa (4 bytes)
      18) page= 0 (4 bytes)
      22) checksum= 0x102db2a2 (4 bytes)
      26) lacing_size= 1 (1 byte)
    + 27) lacing (1 byte)
    - 28) segments (30 bytes)
         -> next= /page[1]/segments
       - 0) vorbis_hdr (30 bytes)
            0) type= 1 (1 byte)
            1) codec= "vorbis" (6 bytes)
            7) vorbis_version= 0 (4 bytes)
            11) audio_channels= 66 (1 byte)
            12) audio_sample_rate= 44100 (4 bytes)
            16) bitrate_maximum= 0 (4 bytes)
 0 root                                     log: 0/0/0  |  F1: help
```

iSEC PARTNERS

Finding root causes

- With the cause identified, you can start manipulating rather than fuzzing
- Play with values in a hex editor or with bbe
  - bbe -e 's:ZZZZ:\xdd\xc5\x04\x08:' < crashy.flac > evil.flac
- In the case of Ogg-contained formats, the included oggcrc.py will recalculate CRC after editing

Collateral damage and future directions

- Non-player apps, or "nobody uses Vorbis!"
- As mentioned before, some of these codecs get around
- Used in games—custom sounds downloaded with maps, etc.
- The Asterisk PBX does
  - 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

Collateral damage and future directions

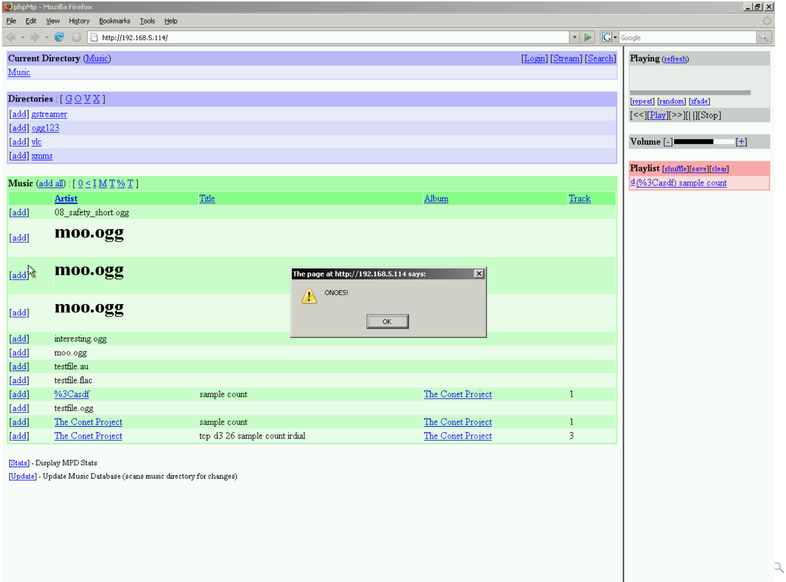
- Also potential for VOIP-related attacks in WAV/PCM modules
  - Good potential for active network attacks; see RTPInject (Lackey, Garbutt)
- “Embedded” devices
  - Phones and other portables play lots of audio and video formats.
  - So do home multimedia devices, game consoles, in-car systems. . . but no one will let me test their car

Collateral damage and future directions

- Total speculation: indexing services and other parsers
  - Some software relies on existing 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
- Web Applications
  - Some apps aren't real careful about data parsed from media
  - Good for CSRF, XSS or JavaScript intranet scanning, etc.

Collateral damage and future directions

Cheesy example: phpMp, front-end for MPD



Demo

Recommendations

- Write paranoid—for every specification, test for violation.
- Vendors should fuzz their own software.
- Media parsing should be done sandboxed when possible.
- Use, but don't rely on, source analysis.
- Users should treat media streams as potentially malicious content.

Questions?

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

david@isecpartners.com