Finding Bugs the Rube-Goldberg Way

Ruxcon 2014

mark.brand@datacom.com.au/c01db33f@gmail.com

Me

Work

- Datacom TSS
- pentesting/code auditing/research

Play

- Same as last year :-P
- When I have time, it’s nice to try and break things.

Outline[0]

Recap

- Last year
- Concolic execution for dummies

Requirements

- What do we need to attack harder problems.
- What do we need to do to find *real* bugs?

Outline[1]

Debugger-integrated goodness

Targetting

- What makes a good target for this technique?
- What legwork do we need to do?

Demos

Recap[0]

```
[*] [0 0x8049128] Wrote 0xb00ff002L recv_0292 return_address
[*] [0 0x8049128] Wrote 0xb00ff003L recv_0293 return_address
[*] Got full control of instruction pointer
[*] Looks like we got control from a return
[*] Writing shellcode at esp
[*] Pivoting via 0x28134827
[*] Built a small zoo on this binary!
[*] Launching exploit against 192.168.91.163:7482
[*] Press any key to throw

antipasto@c01db33f-freebsd-91-x86$ id
uid=1004(antipasto) gid=1004(antipasto) groups=1004(antipasto)
antipasto@c01db33f-freebsd-91-x86$
```

Recap[1] - Last year

Basically a fun toy

- Horrific parallelism (fork())
- Static analysis to generate IL

Plus, it was PoC quality code…

Recap[2] - Concolic

So, concolic execution…

- Your fuzzer is concrete
- Symbolic is impractical
- Concolic is a bit better; you have a get-out-of-jail-free card if things get too hard.

Recap[3] - REIL

Arithmetic Instructions

ADD, SUB, MUL, DIV, MOD, BSH

Bitwise Instructions

AND, OR, XOR

Data Transfer Instructions

LDM, STM, STR

Conditional Instructions

BISZ, JCC

Other Instructions

NOP, UNDEF, UNKN

Requirements[0]

Speed

- Ditching python for C++ was not a good answer to this problem

Windows support

- Any platform on a supported CPU with a gdbstub?

Requirements[1]

Nice-to-have

- Dynamic REIL translation
- Cluster-able
- File-format aware

Targeting[0]

What are we better than a human at?

- Integer boundaries
- Complex pointer arithmetic

What are we hopeless at?

- Massively complex state-spaces
- Heavy use of string functions

Targeting[1]

What do we want to look at?

- Binary protocols/file formats
- Post-crypto or plaintext…

- Audio formats?
- Image formats?
- Fonts?

Approach

Started off writing proper, complete ELF and PE loaders.

Modern ELF is surprisingly undocumented.

Let the system ELF loader handle it…
Use LD_BIND_NOW and a debugger.

But

If we’re doing stuff dynamically…

We can’t rely on static lifting of native code to REIL using IDA and BinNavi.

That approach always had some issues anyway; so…

XREIL

Extra Comparison Instructions

BISNZ, EQU

Better Shift Instructions

LSHL, LSHR, ASHR

Sign Extension

SEX

System Calls

SYS

Still under debate

SDIV

VDB - Visigoth's Debugger

All python, supports BSD, linux, OSX, Windows and all sorts of embedded systems I hope to never see.

Two extension commands:

save_state - dump process state for analysis start-point.

save_trace - dump a trace for testing/validation

Ogg Vorbis

Why? I use it.

Ogg is the container format used to frame the Vorbis codestream.

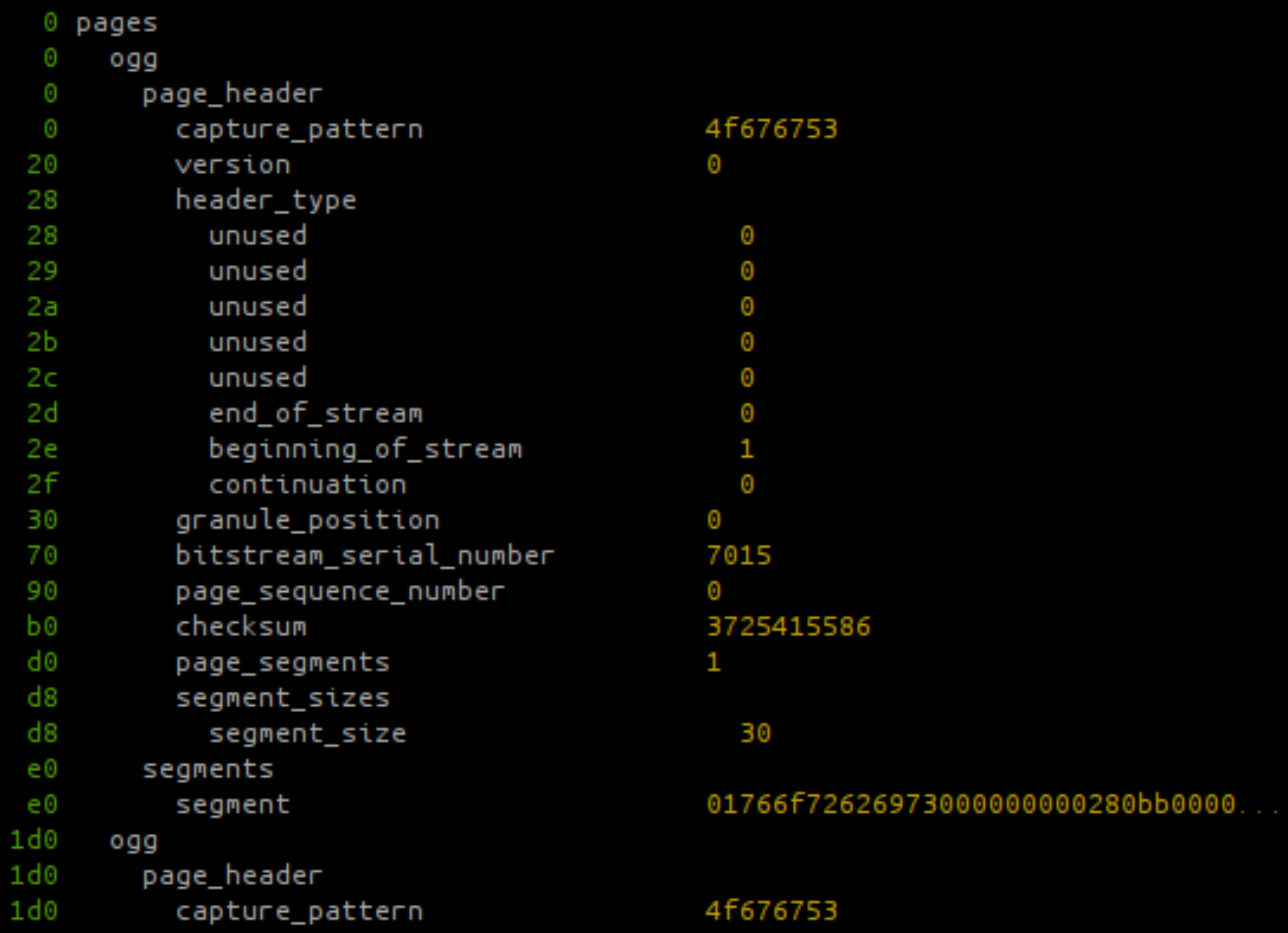
Naively trying to run the tools on a fully symbolic file goes nowhere - Ogg format is *very* simple. We want to mess with the metadata and the Vorbis codestream

Hybrid Concolic Fuzzing?

Idea - parse the input files, mark the parts that we think are interesting as symbolic, leave the boring stuff as concrete.

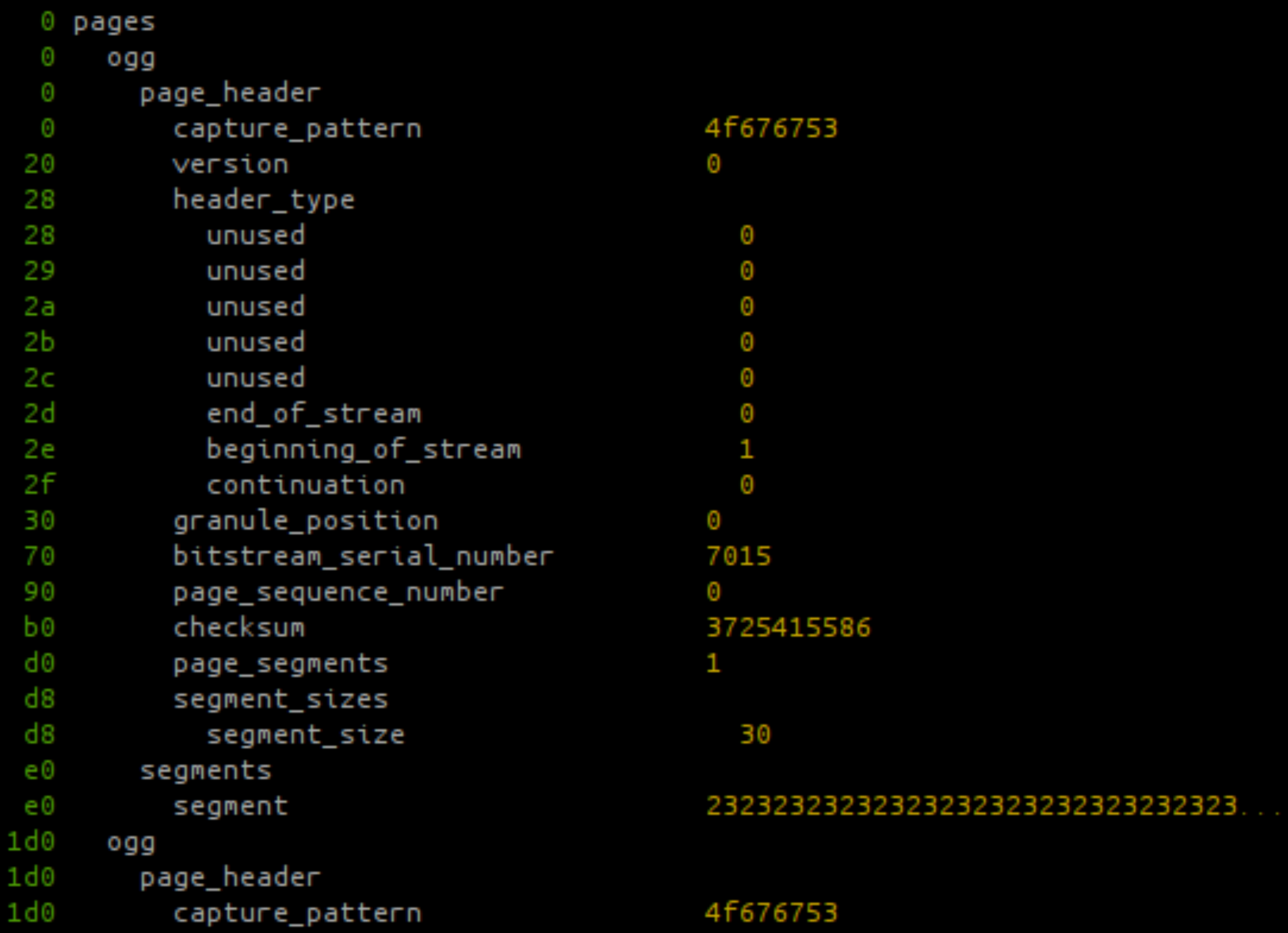
I was going to do this properly, but time limitations...

Input file…



```
   0 pages
   0   ogg
   0     page_header
   0       capture_pattern                    4f676753
  20       version                            0
  28       header_type
  28         unused                             0
  29         unused                             0
  2a         unused                             0
  2b         unused                             0
  2c         unused                             0
  2d         end_of_stream                      0
  2e         beginning_of_stream                1
  2f         continuation                       0
  30       granule_position                   0
  70       bitstream_serial_number            7015
  90       page_sequence_number               0
  b0       checksum                           3725415586
  d0       page_segments                      1
  d8       segment_sizes
  d8         segment_size                       30
  e0     segments
  e0       segment                            01766f7262697300000000280bb0000...
 1d0   ogg
 1d0     page_header
 1d0       capture_pattern                    4f676753
```

Output file…



```
  0 pages
  0   ogg
  0     page_header
  0       capture_pattern                 4f676753
 20       version                         0
 28       header_type
 28         unused                          0
 29         unused                          0
 2a         unused                          0
 2b         unused                          0
 2c         unused                          0
 2d         end_of_stream                   0
 2e         beginning_of_stream             1
 2f         continuation                    0
 30       granule_position                0
 70       bitstream_serial_number         7015
 90       page_sequence_number            0
 b0       checksum                        3725415586
 d0       page_segments                   1
 d8       segment_sizes
 d8         segment_size                    30
 e0     segments
 e0       segment                         23232323232323232323232323232323...
1d0   ogg
1d0     page_header
1d0       capture_pattern                 4f676753
```

Any Questions?

mark.brand@datacom.com.au
c01db33f@gmail.com

Grab the code…

https://github.com/c01db33f