

# A Not-So Fuzzy Art

“Better to have fuzzed and lost than to  
have never fuzzed at all”

# Goal

- Make the *fuzzy* idea of fuzzing more concrete
- Teach you about some fuzzing tools / techniques
- Reassure you that fuzzing is a useful art

# Where do we start?

```
$ while :; do
```

```
timeout 5 ./program < /dev/urandom;
```

```
done
```

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init          ; DATA XREF: _start+1Ato  
                          ; .text:iniitro  
  
PUSH.W                   {R3-R9,LR}  
MOV                       R7, R0  
LDR                       R6, =( __do_global_dtors_aux_fini  
MOV                       R8, R1  
LDR                       R5, =( frame_dummy_init_array_entr  
MOV                       R9, R2  
ADD                       R6, PC ; __do_global_dtors_aux_fi  
BLX                       .init_proc  
ADD                       R5, PC ; __frame_dummy_init_array  
SUBS                      R6, R6, R5  
ASRS                      R6, R6, #2  
BEQ                       locret_104F8  
MOVS                      R4, #0  
  
loc_104E6  
ADDS                      R4, #1  
LDR                       R3, [R4], #4  
MOV                       R2, R9  
MOV                       R1, R8  
MOV                       R0, R7  
BLX                       R3 ; frame_dummy  
CMP                       R4, R6  
BNE                       loc_104E6  
  
locret_104F8              ; CODE XREF: __libc_csu_in  
POP.W                     {R3-R9,PC}  
; End of function __libc_csu_init  
  
; -----  
off_104FC                DCD __do_global_dtors_aux_fini_array_entry - 0x104  
                          ; DATA XREF: __libc_csu_in  
off_10500                DCD __frame_dummy_init_array_entry - 0x104E0  
                          ; DATA XREF: __libc_csu_in  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini          ; DATA XREF: _start+10to  
                          ; .text:off_10390to  
  
BX                       LR  
; End of function __libc_csu_fini  
  
; -----
```

# Where do we start?

```
$ afl-fuzz -i inputs \
-o outputs \
-- ./program
```

```
; ***** SUBROUTINE *****

EXPORT __libc_csu_init
__libc_csu_init                ; DATA XREF: _start+1Ato
                                ; .text:iniitro
                                {R3-R9,LR}
                                PUSH.W
                                MOV     R7, R0
                                LDR     R6, =( __do_global_dtors_aux_fini
                                MOV     R8, R1
                                LDR     R5, =( frame_dummy_init_array_entr
                                MOV     R9, R2
                                ADD     R6, PC ; __do_global_dtors_aux_fi
                                BLX     .init_proc
                                ADD     R5, PC ; __frame_dummy_init_array
                                SUBS     R6, R6, R5
                                ASRS     R6, R6, #2
                                BEQ     locret_104F8
                                MOVS     R4, #0

loc_104E6                      ; CODE XREF: __libc_csu_in
                                ADDS     R4, #1
                                LDR.W    R3, [R5],#4
                                MOV     R2, R9
                                MOV     R1, R8
                                MOV     R0, R7
                                BLX     R3 ; frame_dummy
                                CMP     R4, R6
                                BNE     loc_104E6

locret_104F8                   ; CODE XREF: __libc_csu_in
                                POP.W    {R3-R9,PC}
; End of function __libc_csu_init

; ***** SUBROUTINE *****

off_104FC                      DCD __do_global_dtors_aux_fini_array_entry - 0x104
                                ; DATA XREF: __libc_csu_in
off_10500                      DCD __frame_dummy_init_array_entry - 0x104E0
                                ; DATA XREF: __libc_csu_in

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini                ; DATA XREF: _start+10to
                                ; .text:off_10390to
                                BX        LR
; End of function __libc_csu_fini

; ***** SUBROUTINE *****
```

# Where do we end?

- A hypervisor with COW memory to save and restore the entire OS state (and device drivers) to roll back time for each test case
- Per-block VM-exits as coverage indicators

# Not so black-and-white

- Ok well, let's take a step back
- Do you have source?
  - No? Ok not the end of the world, we do this everyday
  - Yes? Lovely, that'll make this easier (hopefully)

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1Ato
; .text:00000000
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =(__do_global_dtors_aux_fini_array_entry)
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entry)
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry
BLX         .init_proc
ADD         R5, PC ; __frame_dummy_init_array_entry
SUBS        R6, R6, R5
ASRS        R6, R6, #2
BEQ         locret_104F8
MOVS        R4, #0

loc_104E6
; CODE XREF: __libc_csu_init
ADDS        R4, #1
LDR.W       R3, [R5], #4
MOV         R2, R9
MOV         R1, R0
BLX         .frame_dummy
CMP         R4, R6
BEQ         loc_104E6
POP.W       {R3-R9,PC}
; End of function __libc_csu_init

; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+10to
; .text:00000000
BX          LR
; End of function __libc_csu_fini
; ***** SUBROUTINE *****
```

# Binary-only Fuzzing

“Live or die by your coverage” or  
“Can I even run this?”

# Binary-only Fuzzing

- Relevant especially to CTF
- AFL havoc mode or radamsa just won't cut it
  - Don't have benefit of afl-gcc
  - At the *very* least, need a strong input corpus

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
; DATA XREF: __start+1Ato  
; .text:ini7to  
  
PUSH.W      {R3-R9,LR}  
MOV         R7, R0  
LDR         R6, =( __do_global_dtors_aux_fini  
MOV         R8, R1  
LDR         R5, =( frame_dummy_init_array_ent  
MOV         R9, R2  
ADD         R6, PC ; __do_global_dtors_aux_fi  
BLX         .init_proc  
ADD         R5, PC ; __frame_dummy_init_array  
SUBS        R6, R6, R5  
ASRS        R6, R6, #2  
BEQ         locret_104F8  
MOVS        R4, #0  
  
; 104F8  
LDR         R3, [R5],#4  
MOV         R2, R9  
MOV         R1, R8  
MOV         R0, R7  
BLX         R3 ; frame_dummy  
CMP         R4, R6  
BNE         loc_104E6  
  
locret_104F8  
POP.W       {R3-R9,PC}  
; End of function __libc_csu_init  
  
; -----  
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104  
; DATA XREF: __libc_csu_in  
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_in  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
; DATA XREF: __start+10to  
; .text:off_10390to  
  
BX          LR  
; End of function __libc_csu_fini  
  
; -----  
ALDCPU1
```



# Binary-only Fuzzing

- You need some kind of instrumentation!
  - Block/branch/path coverage metrics
  - memcmp(), strcmp()
  - Symbolic execution
- Emulation or static binary rewriting

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
; DATA XREF: __start+1Ato  
; .text:00000000  
  
PUSH.W      {R3-R9,LR}  
MOV         R7, R0  
LDR         R6, =(__do_global_dtors_aux_fini_array_entry - 0x104FC)  
MOV         R8, R1  
LDR         R5, =(__frame_dummy_init_array_entry - 0x104E0)  
MOV         R9, R2  
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry  
BLX         .init_proc  
MOV         R5, PC ; __frame_dummy_init_array_entry  
R6, R6, R5  
R6, R6, #2  
locret_104F8  
R4, #0  
  
; CODE XREF: __libc_csu_init+104E6  
ADDS        R4, #1  
LDR.W       R3, [R5],#4  
MOV         R2, R9  
MOV         R1, R8  
MOV         R0, R7  
BLX         R3 ; frame_dummy  
CMP         R4, R6  
BNE         loc_104E6  
  
locret_104F8  
POP.W       {R3-R9,PC} ; CODE XREF: __libc_csu_init+104E6  
; End of function __libc_csu_init  
  
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_init+104FC  
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_init+10500  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
; DATA XREF: __start+10to  
; .text:00000000  
  
BX          LR  
; End of function __libc_csu_fini  
  
; *****
```

# Binary-only Fuzzing

- Emulation and dynamic binary instrumentation are **tractable** but **slow**
- Static binary instrumentation is **difficult** but **fast**
  - Most solutions require private pdb's (Vulcan, Syzygy)

# Binary-only Fuzzing

- Emulation and dynamic binary instrumentation are **tractable** but **slow**
- Static binary instrumentation is **difficult** but **fast**
  - Most solutions require private pdb's (Vulcan, Syzygy)
- Hardware acceleration? - Intel PT

# Fuzzing “BC”

- Emphasis on black box fuzzing
- Extremely smart mutation engines, so we never “waste” a mutation

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init    ; DATA XREF: __start+1Ato  
                  ; .text:libc_csu_init  
  
    PUSH.W        {R3-R9,LR}  
    MOV           R7, R0  
    LDR           R6, =( __do_global_dtors_aux_fini_array_entry - 0x104FC)  
    MOV           R8, R1  
    LDR           R5, =( __frame_dummy_init_array_entry - 0x10500)  
    MOV           R9, R2  
    ADD           R6, PC ; __do_global_dtors_aux_fini_array_entry  
    BLX           .init_proc  
    ADD           R5, PC ; __frame_dummy_init_array_entry  
    SUBS          R6, R6, R5  
    ASRS          R6, R6, #2  
    BEQ           locret_104F8  
    MOVS          R4, #0  
  
    LDR.W         R3, [R5], #4  
    MOV           R2, R9  
    MOV           R1, R8  
    MOV           R0, R7  
    BLX           R3 ; frame_dummy  
    CMP           R4, R6  
    BNE           loc_104E6  
  
locret_104F8    ; CODE XREF: __libc_csu_init  
    POP.W        {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC      DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                ; DATA XREF: __libc_csu_init  
off_10500      DCD __frame_dummy_init_array_entry - 0x104E0  
                ; DATA XREF: __libc_csu_init  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini    ; DATA XREF: __start+10to  
                  ; .text:libc_csu_fini  
  
    BX           LR  
; End of function __libc_csu_fini  
  
; *****
```

# Age of Enlightenment

- History of modern fuzzing begins with AFL
- Key concepts:
  - Compiler instrumentation-guided
  - Coverage informs input generation

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
; DATA XREF: _start+1Ato  
; .text:iniit7o  
  
PUSH.W      {R3-R9,LR}  
MOV         R7, R0  
LDR         R6, =(__do_global_dtors_aux_fini_array_entry)  
MOV         R8, R1  
LDR         R5, =(__frame_dummy_init_array_entry)  
MOV         R9, R2  
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry  
BLX         R5, PC ; __frame_dummy_init_array_entry  
MOV         R6, R5  
MOV         R7, R6, #2  
BEQ         locret_104F8  
MOVS        R4, #0  
  
loc_104E6  
; CODE XREF: __libc_csu_init  
ADDS        R4, #1  
LDR.W       R3, [R5],#4  
MOV         R2, R9  
MOV         R1, R8  
MOV         R0, R7  
BLX         R3 ; frame_dummy  
CMP         R4, R6  
BNE         loc_104E6  
  
locret_104F8  
; CODE XREF: __libc_csu_init  
POP.W       {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_init  
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_init  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
; DATA XREF: _start+10to  
; .text:off_10390to  
  
BX          LR  
; End of function __libc_csu_fini  
  
; *****  
ALDCM1
```

# Case Study: Radamsa

piazzt@DESKTOP-IERJHAA: ~/radamsa

```
piazzt@DESKTOP-IERJHAA:~/radamsa$ wget -q https://www.w3schools.com/XML/simple.xml -O- | radamsa
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<breakfast_menu>
```

```
<name><food>
```

```
<name>Belgian Waffles</name>
```

```
<price>$5.95</price>
```

```
<description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
```

```
<calories>650</calories>
```

```
</food></name>
```

```
<food>
```

```
<name>Strawberry Belgian Waffles</name>
```

```
<price>$7.95</price>
```

```
<description>Light Belgian waffles covered with strawberries and whipped cream</description>
```

```
<calories>900</calories>
```

```
</food>
```

# Case Study: Radamsa

- More **sophisticated** mutations, XML-like inputs aware
- **No** harness
- **No** feedback

```
$ gzip -c /bin/bash > sample.gz
$ while true; do radamsa sample.gz | gzip -d > /dev/null; done
```



# Case Study: AFL

american fuzzy lop 0.47b (readpng)			
<b>process timing</b>		<b>overall results</b>	
run time : 0 days, 0 hrs, 4 min, 43 sec		cycles done : 0	
last new path : 0 days, 0 hrs, 0 min, 26 sec		total paths : 195	
last uniq crash : none seen yet		uniq crashes : 0	
last uniq hang : 0 days, 0 hrs, 1 min, 51 sec		uniq hangs : 1	
<b>cycle progress</b>		<b>map coverage</b>	
now processing : 38 (19.49%)		map density : 1217 (7.43%)	
paths timed out : 0 (0.00%)		count coverage : 2.55 bits/tuple	
<b>stage progress</b>		<b>findings in depth</b>	
now trying : interest 32/8		favored paths : 128 (65.64%)	
stage execs : 0/9990 (0.00%)		new edges on : 85 (43.59%)	
total execs : 654k		total crashes : 0 (0 unique)	
exec speed : 2306/sec		total hangs : 1 (1 unique)	
<b>fuzzing strategy yields</b>		<b>path geometry</b>	
bit flips : 88/14.4k, 6/14.4k, 6/14.4k		levels : 3	
byte flips : 0/1804, 0/1786, 1/1750		pending : 178	
arithmetics : 31/126k, 3/45.6k, 1/17.8k		pend fav : 114	
known ints : 1/15.8k, 4/65.8k, 6/78.2k		imported : 0	
havoc : 34/254k, 0/0		variable : 0	
trim : 2876 B/931 (61.45% gain)		latent : 0	



# Case Study: AFL

- Huge emphasis on file format fuzzing
- Heavy binary-based mutations
  - Sequential bit/byte flips, random stepover
  - Incs/decs of varying bitwidths
  - Boundary values

```
; ***** SUBROUTINE *****  
  
__libc_csu_init      EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: _start+1Ato  
                    ; .text:iniit0  
  
    PUSH.W           {R3-R9,LR}  
    MOV              R7, R0  
    LDR              R6, =( __do_global_dtors_aux_fini  
    MOV              R8, R1  
    LDR              R5, =( frame_dummy_init_array_entr  
    MOV              R9, R2  
    ADD              R6, PC ; __do_global_dtors_aux_fi  
    BLX              .init_proc  
    ADD              R5, PC ; __frame_dummy_init_array  
    LDR              R6, R5  
    LDR              R6, R6, #2  
    BEQ              locret_104F8  
    MOV              R4, #0  
  
loc_104E6            ; CODE XREF: __libc_csu_in  
    ADDS             R4, #1  
    LDR.W            R3, [R5],#4  
    MOV              R2, R9  
    MOV              R1, R8  
    MOV              R0, R7  
    BLX              R3 ; frame_dummy  
    CMP              R4, R6  
    BNE              loc_104E6  
  
locret_104F8         ; CODE XREF: __libc_csu_in  
    POP.W            {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC            DCD __do_global_dtors_aux_fini_array_entry - 0x104  
                    ; DATA XREF: __libc_csu_in  
off_10500            DCD __frame_dummy_init_array_entry - 0x104E0  
                    ; DATA XREF: __libc_csu_in  
  
; ***** SUBROUTINE *****  
  
__libc_csu_fini      EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: _start+10to  
                    ; .text:off_10390to  
  
    BX              LR  
; End of function __libc_csu_fini  
  
; *****  
ALDCPU1
```

# Case Study: AFL

- Probably unhelpful for *most* menu-based CTF challenges...

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
    PUSH.W    {R3-R9,LR}  
    MOV       R7, R0  
    LDR       R6, =( __do_global_dtors_aux_fini_array_entry - 0x104FC)  
    MOV       R8, R1  
    LDR       R5, =( frame_dummy_init_array_entry - 0x10500)  
    MOV       R9, R2  
    ADD       R6, PC ; __do_global_dtors_aux_fini_array_entry  
    BLX       .init_proc  
    ADD       R6, R6, #0  
    BEQ       locret_104F8  
    MOVS      R4, #0  
  
loc_104E6  
    ADDS      R4, #1  
    LDR.W     R3, [R5],#4  
    MOV       R2, R9  
    MOV       R1, R8  
    MOV       R0, R7  
    BLX       R3 ; frame_dummy  
    CMP       R4, R6  
    BNE       loc_104E6  
  
locret_104F8  
    POP.W     {R3-R9,PC} ; CODE XREF: __libc_csu_init  
; End of function __libc_csu_init  
  
; *****  
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_init  
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_init  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
    BX        LR  
; End of function __libc_csu_fini  
  
; *****
```

# Case Study: AFL

- “Coverage-guided Fuzzing”
  - Associate each input with its corresponding coverage bitmap
  - Save inputs that have unique coverage to corpus
  - Mutator draws from corpus

```
; ***** SUBROUTINE *****  
  
__libc_csu_init      EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: _start+1Ato  
                    ; .text:00000000  
                    PUSH.W      {R3-R9,LR}  
                    MOV         R7, R0  
                    LDR         R6, =( __do_global_dtors_aux_fini_array_entry  
                    MOV         R8, R1  
                    LDR         R5, =( frame_dummy_init_array_entry  
                    MOV         R9, R2  
                    ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry  
                    BLX         .init_proc  
                    ADD         R5, PC ; __frame_dummy_init_array_entry  
                    SUBS         R6, R6, R5  
                    ASRS         R6, R6, #2  
                    BEQ         locret_104FB  
                    MOVS         R4, #0  
                    ALD         R4, #0  
                    LDR.W       R3, [R5],#4  
                    MOV         R2, R9  
                    MOV         R1, R8  
                    MOV         R0, R7  
                    BLX         R3 ; frame_dummy  
                    CMP         R4, R6  
                    loc_104E6  
locret_104FB      ; CODE XREF: __libc_csu_init  
                    POP.W       {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC      DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_init  
off_10500      DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_init  
  
; ***** SUBROUTINE *****  
  
__libc_csu_fini      EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: _start+10to  
                    ; .text:00000000  
                    BX          LR  
; End of function __libc_csu_fini  
  
; *****
```

# Case Study: AFL

- Employs compiler or Qemu instrumentation to compute “path coverage”
- Saves coverage as 4k bitmap

```
cur_location = (block_address >> 4) ^ (block_address << 8);
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

# Case Study: AFL

- In general: coverage can be anything from edge or basic blocker counters, to wrapping compare / memcmp instructions

```
; ***** SUBROUTINE *****  
  
__libc_csu_init    EXPORT __libc_csu_init  
__libc_csu_init    ; DATA XREF: __start+1Ato  
                    ; .text:00000000  
                    PUSH.W      {R3-R9,LR}  
                    MOV         R7, R0  
                    LDR         R6, =(__do_global_dtors_aux_fini_array_entry - 0x104FC)  
                    MOV         R8, R1  
                    LDR         R5, =(__frame_dummy_init_array_entry - 0x104E0)  
                    MOV         R9, R2  
                    ADD         R6, PC, #4  
                    BLX         .init_proc  
                    ADD         R5, PC, #4  
                    SUB         R6, R6, #4  
                    CSR         R6, R6  
                    BEQ         locret_104FB  
                    MOV         R4, #0  
                    MOV         R3, [R5], #4  
                    MOV         R2, R9  
                    MOV         R1, R8  
                    MOV         R0, R7  
                    BLX         R3, #frame_dummy  
                    CMP         R4, R6  
                    BNE         loc_104E6  
  
locret_104FB    ; CODE XREF: __libc_csu_init+1Ato  
                POP.W      {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_init+1Ato  
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_init+1Ato  
  
; ***** SUBROUTINE *****  
  
__libc_csu_fini    EXPORT __libc_csu_fini  
__libc_csu_fini    ; DATA XREF: __start+10to  
                    ; .text:00000000  
                    BX         LR  
; End of function __libc_csu_fini  
  
; *****
```

# Case Study: AFL

- Coverage informs the mutator about inputs interesting to this target
  - Typically Block/Path coverage
- We probably haven't scratched the surface of instrumentation

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init                                ; DATA XREF: __start+1Ato  
                                                ; .text:00000000  
  
PUSH.W                                        {R3-R9,LR}  
MOV     R7, R0  
LDR     R6, =(__do_global_dtors_aux_fini  
MOV     R8, R1  
LDR     R5, =(__frame_dummy_init_array_e  
MOV     R9, R2  
ADD     R6, PC ; __do_global_dtors_aux_f  
BLX     .init_proc  
ADD     R5, PC ; __frame_dummy_init_array  
SUB     CSR, R5  
CSR     R5, #2  
BEQ     locret_104F8  
MOV     R4, #0  
  
loc_104E6                                     ; CODE XREF: __libc_csu_in  
ADD     R4, #1  
LDR.W   R3, [R5],#4  
MOV     R2, R9  
MOV     R1, R8  
MOV     R0, R7  
BLX     R3 ; frame_dummy  
CMP     R4, R6  
BNE     loc_104E6  
  
loc_104E6                                     ; CODE XREF: __libc_csu_in  
POP.W   {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                                                ; DATA XREF: __libc_csu_in  
off_10500 DCD __frame_dummy_init_array_entry - 0x104E0  
                                                ; DATA XREF: __libc_csu_in  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini                                ; DATA XREF: __start+10to  
                                                ; .text:00000000  
  
BX      LR  
; End of function __libc_csu_fini  
  
; *****  
ALDCM4
```

# Source Instrumentation

- Other kinds of instrumentation to aid fuzzing:
  - *ASAN*
  - *UBSAN*
  - *TSAN*
- Source pretty much required for these

# Binary Instrumentation

- AFL-Qemu - As accurate as qemu, as slow as qemu
- AFL-Pin/honggfuzz-pin - As fast as your CPU, as slow as your disassembler
- WinAFL - DynamoRIO, Windows only
- These are just off the shelf tools



# Binary Instrumentation

- mesos-style<sup>1</sup> - A mix of both; Temporary breakpoints that are removed after each hit.
  - Block coverage only (currently)
- Snapshot fuzzing - Emulated

```
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: _start+1Ato
; .text:ini7to
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =( __do_global_dtors_aux_fini_array_entry
MOV         R8, R1
LDR         R5, =( frame_dummy_init_array_entry
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fini
BLX         .init_proc
ADD         R5, PC ; frame_dummy_init_array
; R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15
; R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31
BEQ         locret_104F8
MOVS        R4, #0
loc_104E6
; CODE XREF: __libc_csu_init
ADDS        R4, #1
LDR.W       R3, [R5],#4
MOV         R2, R9
MOV         R1, R8
MOV         R0, R7
BLX         R3 ; frame_dummy
CMP         R4, R6
BNE         loc_104E6
locret_104F8
; CODE XREF: __libc_csu_init
POP.W       {R3-R9,PC}
; End of function __libc_csu_init
; -----
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init
; -----
; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:off_10390to
BX          LR
; End of function __libc_csu_fini
; -----
```

# More non-source fuzzing

- If a bug doesn't crash does it make a sound?
- Poor man's alternative to clang sanitizers:
  - Page Heap
  - Application Verifier
  - LibDislocator
  - DrMemory
- Binpatching...? :'(

```
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1Ato
; .text:00000000
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =(__do_global_dtors_aux_fini_array_entry)
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entry)
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry
BLX         .init_proc
ADD         R5, PC ; __frame_dummy_init_array_entry
SUB         R5, R5, #2
BEQ         locret_104F8
MOVS        R4, #0

; CODE XREF: __libc_csu_init+00000000
LDR.W       R4, #1
LDR.W       R3, [R5],#4
MOV         R2, R9
MOV         R1, R8
MOV         R0, R7
BLX         R3 ; frame_dummy
CMP         R4, R6
BNE         loc_104E6

locret_104F8
; CODE XREF: __libc_csu_init+00000000
POP.W       {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init+00000000
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init+00000000

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+10to
; .text:00000000
BX          LR
; End of function __libc_csu_fini

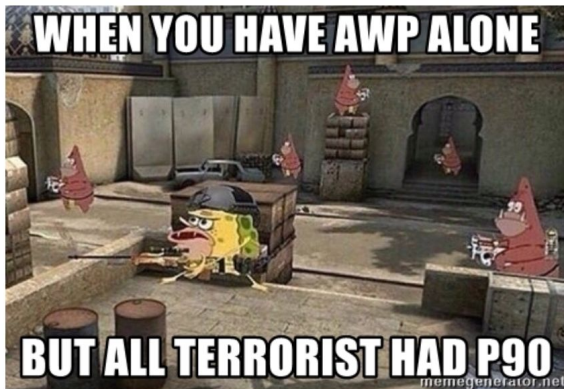
; -----
```

# A Case Study: RealWorldCTF

## Fuzzing Counter-Strike: Global Offensive maps files with AFL

Aug 26, 2018 • By [niklasb](#)

[RealWorldCTF 2018](#) had a really fun challenge called “P90 Rush B”, an allusion to a desperate tactic employed in the Valve game “Counter-Strike: Global Offensive”. It was about finding and exploiting a bug in the map file loader used by a CS:GO server. During the CTF, I exploited a stack buffer overflow that was later described well [in a writeup by another team](#).



# A Case Study: RealWorldCTF

- CSGO Map File Format Fuzzing (niklasb) - <https://phoenixhex.re/2018-08-26/csgo-fuzzing-bsp>
- Literally dlopen'd csgo engine and repeatedly called the CModelLoader::GetModelForName and had AFL+Qemu do the rest

# A Case Study: RealWorldCTF

- Not always that easy, sometimes the input is not just a binary blob
  - Prefer not to waste mutations...
  - Write a generator or stamp in magic values by hand after some REing

# Modern Techniques

The Future is *Now*

# Emulation

- Environment can be completely specified
  - In-vitro
  - *You specify the environment* (hooks)
- Can instrument programs via hooks

```
cur_location = (block_address >> 4) ^ (block_address << 8);
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

# Emulation

- Qemu linux i386 user:
  - Library **dependencies** (need correct ld.so)
  - **Unsupported** system calls (sys\_memfd\_create, etc)
- Emulating from the first instruction is a **pain**
  - Hold this thought...

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: _start+1Ato  
                    ; .text:00000000  
  
PUSH.W              {R3-R9,LR}  
MOV                 R7, R0  
LDR                 R6, =(__do_global_dtors_aux_fini_array_entry)  
MOV                 R8, R1  
LDR                 R5, =(__frame_dummy_init_array_entry)  
MOV                 R9, R2  
ADD                 R6, PC ; __do_global_dtors_aux_fini_array_entry  
BLX                 .init_proc  
ADD                 R5, PC ; __frame_dummy_init_array_entry  
SUBS                 R6, R6, R5  
ASRS                 R6, R6, #2  
BEQ                 locret_104F8  
MOVS                 R4, #0  
  
                    ; CODE XREF: __libc_csu_init+00000000  
ADDS                 R4, #1  
LDR.W               R3, [R5],#4  
MOV                 R2, R3  
MOV                 R1, R2  
BLX                 R1  
CMP                 R3, R6  
R4, R6  
locret_104F8         ; CODE XREF: __libc_csu_init+00000000  
POP.W               {R3-R9,PC}  
; End of function __libc_csu_init  
  
;-----  
off_104FC           DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                    ; DATA XREF: __libc_csu_init+00000000  
off_10500           DCD __frame_dummy_init_array_entry - 0x104E0  
                    ; DATA XREF: __libc_csu_init+00000000  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: _start+10to  
                    ; .text:00000000  
  
BX                  LR  
; End of function __libc_csu_fini  
  
;-----
```



# Emulation



# Snapshot Fuzzing

- Start emulation from a “snapshot” in time
  - Completely repeatable continued executions
- **Perfect** state restoration
  - \*cough\* AFL forking-server mode
  - \*cough\* WinAFL
- No source required

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: _start+1Ato  
                    ; .text:iniTo  
  
    PUSH.W          {R3-R9,LR}  
    MOV             R7, R0  
    LDR             R6, =( __do_global_dtors_aux_fini  
    MOV            R8, R1  
    LDR             R5, =( frame_dummy_init_array_ent  
    MOV            R9, R2  
    ADD            R6, PC ; __do_global_dtors_aux_fi  
    BLX            .init_proc  
    ADD            R5, PC ; __frame_dummy_init_array  
    LDR             R6, R5  
    LDR             R6, R5  
    BEQ            locret_104FB  
    MOV            R4, #0  
    MOV            R4, #1  
    ADDS           R4, #1  
    LDR.W          R3, [R5],#4  
    MOV            R2, R9  
    MOV            R1, R8  
    MOV            R0, R7  
    BLX            R3 ; frame_dummy  
    CMP            R4, R6  
    BNE            loc_104E6  
  
locret_104FB      ; CODE XREF: __libc_csu_in  
    POP.W          {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC      DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                ; DATA XREF: __libc_csu_in  
off_10500      DCD __frame_dummy_init_array_entry - 0x104E0  
                ; DATA XREF: __libc_csu_in  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: _start+10to  
                    ; .text:off_10390to  
  
    BX             LR  
; End of function __libc_csu_fini  
  
; *****
```

# Snapshot Fuzzing

- “Human in the loop”
  - a. Fuzz for a while
  - b. Analyze coverage
    - i. Take new snapshot
    - ii. Or update mutator/generator
  - c. Repeat

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init                                ; DATA XREF: _start+1A0  
                                                ; .text:00000000  
  
PUSH.W                                        {R3-R9,LR}  
MOV     R7, R0  
LDR     R6, =(__do_global_dtors_aux_fini_array_entry)  
MOV     R8, R1  
LDR     R5, =(__frame_dummy_init_array_entry)  
MOV     R9, R2  
ADD     R6, PC ; __do_global_dtors_aux_fini_array_entry  
BLX     .init_proc  
ADD     R5, PC ; __frame_dummy_init_array_entry  
SUBS    R6, R6, R5  
ASRS    R6, R6, #2  
BEQ     locret_104F8  
MOVS    R4, #0  
  
loc_104E6                                     ; CODE XREF: __libc_csu_init+00000000  
ADDS    R4, #1  
LDR.W   R3, [R5],#4  
MOV     R2, R9  
MOV     R1, R8  
MOV     R0, R7  
BLX     R3 ; frame_dummy  
CMP     R4, R6  
BNE     loc_104E6  
  
locret_104F8                                 ; CODE XREF: __libc_csu_init+00000004  
POP.W   {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                                                ; DATA XREF: __libc_csu_init+00000008  
off_10500 DCD __frame_dummy_init_array_entry - 0x104E0  
                                                ; DATA XREF: __libc_csu_init+0000000C  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini                                ; DATA XREF: _start+100  
                                                ; .text:00000000  
  
BX      LR  
; End of function __libc_csu_fini  
  
; *****
```

# Snapshot Restore

- Optimized snapshot restore ensures we spend more time fuzzing and less time loading all program state into the emulator

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
    PUSH.W      {R3-R9,LR}  
    MOV         R7, R0  
    LDR         R6, =(__do_global_dtors_aux_fini_array_entry - 0x104FC)  
    MOV         R8, R1  
    LDR         R5, =(__frame_dummy_init_array_entry - 0x104E0)  
    MOV         R9, R2  
    ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry  
    BLX         .init_proc  
    ADD         R5, PC ; __frame_dummy_init_array_entry  
    ASR         R7, R7, #1  
    BEQ         locret_104F8  
    MOVS        R4, #0  
    LDR.W       R3, [R5], #4  
    MOV         R2, R9  
    MOV         R1, R8  
    MOV         R0, R7  
    BLX         R3 ; frame_dummy  
    CMP         R4, R6  
    BNE         loc_104E6  
  
locret_104F8  
    POP.W       {R3-R9,PC} ; CODE XREF: __libc_csu_init  
; End of function __libc_csu_init  
  
; *****  
off_104FC      DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_init  
off_10500      DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_init  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
    BX         LR  
; End of function __libc_csu_fini  
  
; *****
```

# Snapshot Restore

1. Dirty page tracking - only restore pages that have been written to
2. Abuse COW pages using OS facilities
  - a. AFL-forkserver mode
  - b. Can easily do this with file-backed pages and unicorn

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
    ; DATA XREF: _start+1Ato  
    ; .text:ini7to  
    PUSH.W    {R3-R9,LR}  
    MOV       R7, R0  
    LDR       R6, =( __do_global_dtors_aux_fini  
    MOV       R8, R1  
    LDR       R5, =( frame_dummy_init_array_enti  
    MOV       R9, R2  
    ADD       R6, PC    ; __do_global_dtors_aux_fi  
    BLX       .init_proc  
    ADD       R5, PC    ; __frame_dummy_init_array  
    MOV       R5, R0, #2  
    BEQ       locret_104F8  
    MOV       R4, #0  
  
loc_104E6  
    ; CODE XREF: __libc_csu_in  
    ADDS      R4, #1  
    LDR.W     R3, [R5],#4  
    MOV       R2, R9  
    MOV       R1, R8  
    MOV       R0, R7  
    BLX       R3        ; frame_dummy  
    CMP       R4, R6  
    BNE       loc_104E6  
  
locret_104F8  
    ; CODE XREF: __libc_csu_in  
    POP.W     {R3-R9,PC}  
; End of function __libc_csu_init  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
    ; DATA XREF: _start+10to  
    ; .text:off_10390to  
    BX        LR  
; End of function __libc_csu_fini  
  
; ***** SUBROUTINE *****
```

# Snapshotting (usermode)

- Userland snapshots don't have a kernel (core dump, minidump)
  - Sockets
  - Files
  - Registry
- Implement a “mock OS”

# Snapshotting (reduced kernel)

- Full kernel+usermode memory snapshot, but no device state
- Able to run a few syscalls accurately

linuxbochs commented on Jan 16, 2016

Author

Contributor

...

@aquynh yes, I have a legitimate reason to run a reduced Linux kernel under Unicorn. I've already confirmed I seem to be able to switch between user/kernel mode as well.

# Snapshotting (full system)

- Entire CPU and device state is captured
- a. Ia. VMWare/Bochs snapshots
  - Cache reads, throw away writes to disk
  - State of networking stack?

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
libc_csu_init
    ; DATA XREF: _start+1Ato
    ; .text:ini7to
    {R3-R9,LR}
    R7, R0
    R6, =(__do_global_dtors_aux_fini
    R8, R1
    R5, =(__frame_dummy_init_array_entr
    MOV     R9, R2
    ADD     R6, PC ; __do_global_dtors_aux_fi
    BLX     .init_proc
    ADD     R5, PC ; __frame_dummy_init_array
    R6, R6, R5
    R6, R6, #2
    locret_104F8
    BEQ     R4, #0
    MOV     R4, #0
    ; CODE XREF: __libc_csu_in
    ADDS     R4, #1
    LDR.W    R3, [R5],#4
    MOV     R2, R9
    MOV     R1, R8
    MOV     R0, R7
    BLX     R3 ; frame_dummy
    CMP     R4, R6
    BNE     loc_104E6
    locret_104F8
    POP.W    {R3-R9,PC}
; End of function __libc_csu_init
; *****
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104
; DATA XREF: __libc_csu_in
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_in
; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
    ; DATA XREF: _start+10to
    ; .text:off_10390to
    BX      LR
; End of function __libc_csu_fini
; *****

```



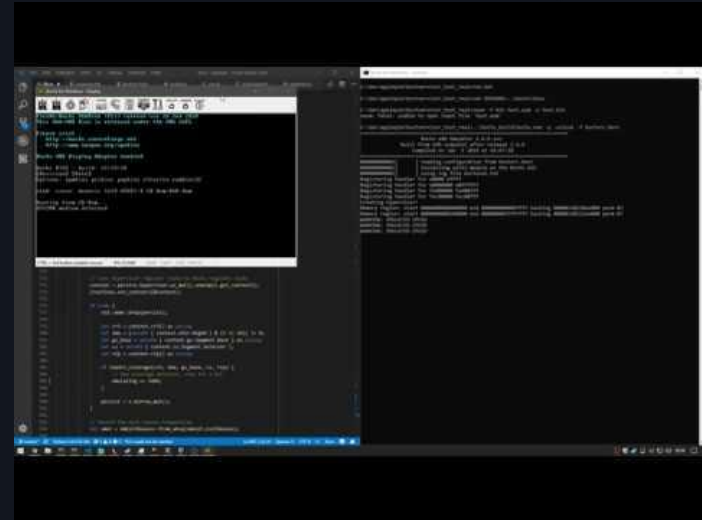
# Snapshotting (full system)

- Device restoration is **expensive** and **hard to do deterministically**
- Emulating a full system (i.e. up to the snapshot point) takes **forever**

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
    ; DATA XREF: __start+1Ato  
    ; .text:00000000  
    {R3-R9,LR}  
    R7, R0  
    R6, =(__do_global_dtors_aux_fini_array_entry - 0x104FC)  
    R8, R1  
    R5, =(__frame_dummy_init_array_entry - 0x10500)  
    MOVW R9, R2  
    ADD R6, PC ; __do_global_dtors_aux_fini_array_entry  
    BLX .init_proc  
    ADD R5, PC ; __frame_dummy_init_array_entry  
    MOV R6, R0  
    BEQ locret_104F8  
    MOVW R4, #0  
  
loc_104E6  
    ; CODE XREF: __libc_csu_init+00000000  
    ADDS R4, #1  
    LDR.W R3, [R5], #4  
    MOV R2, R9  
    MOV R1, R0  
    MOV R0, R3  
    CMP R4, R6  
    BNE loc_104E6  
  
locret_104F8  
    ; CODE XREF: __libc_csu_init+00000000  
    POP.W {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_init+00000000  
off_10500 DCD __frame_dummy_init_array_entry - 0x10500  
; DATA XREF: __libc_csu_init+00000000  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
    ; DATA XREF: __start+10to  
    ; .text:00000000  
    BX LR  
; End of function __libc_csu_fini  
  
; *****
```

# Snapshotting (full system)

- Make full-system snapshotting fast: throw a hypervisor into bochs?



# Snapshot Fuzzing

- (Almost) no tools for this
- Can somewhat replicate with off-the-shelf tools
  - AFL and honggfuzz fork-server
  - LibFuzzer
  - Need to walk up to the desired state often

```
; ***** SUBROUTINE *****  
  
__libc_csu_init      EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: __start+1Ato  
                    ; .text:libc_csu_init  
                    PUSH.W      {R3-R9,LR}  
                    MOV         R7, R0  
                    LDR         R6, =(__do_global_dtors_aux_fini_array_entry - 0x104FC)  
                    MOV         R8, R1  
                    LDR         R5, =(__frame_dummy_init_array_entry - 0x10500)  
                    MOV         R9, R2  
                    ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry  
                    BLX         .init_proc  
                    ADD         R5, PC ; __frame_dummy_init_array_entry  
                    SUBS         R6, R6, R5  
                    ASRS         R6, R6, #2  
                    BEQ         locret_104F8  
                    MOVS        R4, #0  
  
locret_104F8      ; CODE XREF: __libc_csu_init  
                    POP.W        {R3-R9,PC}  
; End of function __libc_csu_init  
; ***** SUBROUTINE *****  
  
__libc_csu_fini      EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: __start+10to  
                    ; .text:libc_csu_fini  
                    BX           LR  
; End of function __libc_csu_fini  
; ***** SUBROUTINE *****
```

# Snapshot Fuzzing

- Hopefully some are released soon ;-)





“These go to 11”

```

; ***** SUBROUTINE *****
it
; DATA XREF: _start+1Ato
; .text:inittro
R9,LR}
R0
=(__do_global_dtors_aux_fini
R1
=(__frame_dummy_init_array_ent
R2
PC ; __do_global_dtors_aux_fi
t_proc
PC ; __frame_dummy_init_array
R6, R5
R6, #2
et_104F8
#0
; CODE XREF: __libc_csu_in
#1
[R5],#4
R9
R8
R7
; frame_dummy
R6
104E6
; CODE XREF: __libc_csu_in
R9,PC}

-----
s_aux_fini_array_entry - 0x104
; DATA XREF: __libc_csu_in
it_array_entry - 0x104E0
; DATA XREF: __libc_csu_in

; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:off_1039to
BX LR
; End of function __libc_csu_fini
; *****
```

# Hypervisor Snapshot Fuzzing

- Not for the faint of heart
  - Maximum performance
  - Extreme effort
- Fun application of snapshot fuzzing!

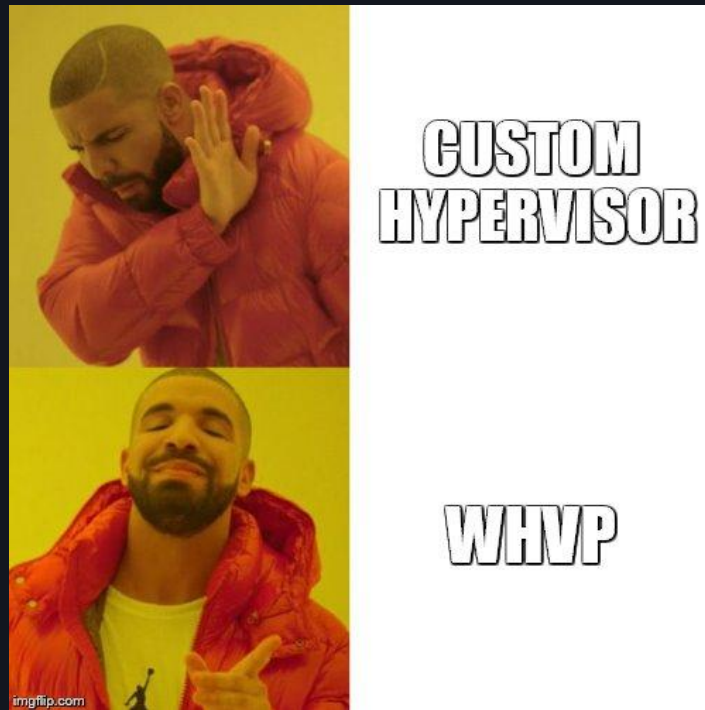
[https://github.com/gamozolabs/falkervisor\\_grilled\\_cheese](https://github.com/gamozolabs/falkervisor_grilled_cheese)

```
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: _start+1Ato
; .text:00000000
; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:00000000
; ***** SUBROUTINE *****
; End of function __libc_csu_fini
BX LR
; ***** SUBROUTINE *****
loc_104E6
; CODE XREF: __libc_csu_in
ADDS R4, #1
LDR.W R3, [R5], #4
MOV R2, R9
MOV R1, R8
MOV R0, R7
BLX R3
CMP R4, R6
BNE loc_104E6
locret_104F8
; CODE XREF: __libc_csu_in
POP.W {R3-R9, PC}
; End of function __libc_csu_init
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_in
off_10500 DCD __frame_dummy_init_array_entry - 0x10500
; DATA XREF: __libc_csu_in
; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:00000000
; ***** SUBROUTINE *****
; End of function __libc_csu_fini
BX LR
; ***** SUBROUTINE *****
```

# Hypervisor Snapshot Fuzzing

- Load full memory dump into address space of VM
  - + mesos-style coverage using vmexits
  - + fast snapshot restore using COW memory

# Hypervisor Snapshot Fuzzing



```
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1Ato
; .text:ini7to
PUSH.W R9,LR
MOV R0
R6,=(__do_global_dtors_aux_fini_array_entry)
LDR R8,R1
MOV R5,=(frame_dummy_init_array_entry)
MOV R9,R2
ADD R6,PC ; __do_global_dtors_aux_fini_array_entry
BLX __init_proc
ADD R5,PC ; __frame_dummy_init_array_entry
SUBS R6,R6,R5
ASRS R6,R6,#2
BEQ locret_104F8
MOVS R4,#0

; CODE XREF: __libc_csu_init
ADDS R4,#1
LDR.W R3,[R5],#4
MOV R2,R9
MOV R1,R8
MOV R0,R7
BLX R3 ; frame_dummy
CMP R4,R6
BNE loc_104E6

4F8 ; CODE XREF: __libc_csu_init
POP.W {R3-R9,PC}
function __libc_csu_init
DCD __do_global_dtors_aux_fini_array_entry - 0x104E6
; DATA XREF: __libc_csu_init
DCD __frame_dummy_init_array_entry - 0x104E8
; DATA XREF: __libc_csu_init
***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+10to
; .text:off_10398to
BX LR
; End of function __libc_csu_fini
*****
```



# Hypervisor Snapshot Fuzzing

```
// Run the hypervisor until exit, returning the exit context
pub fn run(&mut self) -> WHV_RUN_VP_EXIT_CONTEXT {
    let mut context: WHV_RUN_VP_EXIT_CONTEXT =
        unsafe { std::mem::zeroed() };
    let res = unsafe { WHVRunVirtualProcessor(self.partition, 0,
        &mut context as *mut WHV_RUN_VP_EXIT_CONTEXT as *mut c_void,
        std::mem::size_of_val(&context) as u32) };
    assert!(res == 0, "WHVRunVirtualProcessor() error: {:#x}", res);

    // Mark that memory may be dirty
    self.memory_dirty = true;

    context
}
```

# Source-Based Fuzzing

For Real People Doing Real Things and  
Not CTFs

# Source-Based Fuzzing

- The compiler is your friend, let it help you
- So step 1: Instrumentation
  - GCC has gcov
  - LLVM+Clang have SanitizerCoverage
  - AFL has a similar compiler pass
  - Build your own birdfeeder

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
libc_csu_init
; DATA XREF: _start+1Ato
; .text:ini7to
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =(__do_global_dtors_aux_fini_array_entry)
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entry)
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry
BLX         .init_proc
ADD         R5, PC ; __frame_dummy_init_array_entry
MOV         R6, R5
MOV         R6, R6, #2
BEQ         locret_104F8
MOVS        R4, #0

loc_104E6
; CODE XREF: __libc_csu_init
ADDS        R4, #1
LDR.W       R3, [R5],#4
MOV         R2, R9
MOV         R1, R8
MOV         R0, R7
BLX         R3 ; frame_dummy
CMP         R4, R6
BNE         loc_104E6

locret_104F8
; CODE XREF: __libc_csu_init
POP.W       {R3-R9,PC}
; End of function __libc_csu_init

;-----
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:off_10390to
BX          LR
; End of function __libc_csu_fini
; *****
```

# Source-Based Fuzzing

- Step 2: Harness the code you want to target
  - Either embed a harness in the codebase, and behave like AFL's forkserver
  - Simple python harness will do to delivery fuzz test case to program

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
libc_csu_init
; DATA XREF: _start+1Ato
; .text:ini7to
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =(__do_global_dtors_aux_fini_array_entry)
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entry)
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry
BLX         .init_proc
ADD         R5, PC ; __frame_dummy_init_array_entry
; R5
; R5
BEQ         locret_104F8
MOV         R4, #0
ADDS        R4, #1
LDR.W       R3, [R5],#4
MOV         R2, R9
MOV         R1, R8
MOV         R0, R7
BLX         R3 ; frame_dummy
; R4
; R6
locret_104F8
POP.W       {R3-R9,PC}
; End of function __libc_csu_init

; *****
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init

; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:off_10390to
BX          LR
; End of function __libc_csu_fini
; *****

```

# Source-Based Fuzzing

- Step 3: Work on your mutator / fuzz test case generator
  - Do this as appropriate when coverage shows deficiencies
  - Will find *much* deeper bugs

```
EXPORT __libc_csu_init
libc_csu_init
; DATA XREF: _start+1Ato
; .text:iniit70
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =(__do_global_dtors_aux_fini_array_entry)
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entry)
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry
BLX         .init_proc
APR         R5, PC ; __frame_dummy_init_array_entry
; .text:iniit70
BEQ         locret_104F8
MOV         R4, #0

loc_104E6
; CODE XREF: __libc_csu_init
ADDS        R4, #1
LDR.W       R3, [R5],#4
MOV         R2, R9
MOV         R1, R3
BLA         R3, R1 ; __frame_dummy_init_array_entry
CMP         R4, R6
BNE         loc_104E6

locret_104F8
; CODE XREF: __libc_csu_init
POP.W       {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC    DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init
off_10500    DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init

; ***** SUBROUTINE *****

EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:off_10390to
BX          LR
; End of function __libc_csu_fini

; -----
```

# Source-Based Fuzzing

```
266 try { someTypedArray1.reduce(function(acc, cval, c_index, c_array) { try{ c_array[c_index] = c_array.reduce((acc, cval, c_index, c_array) => { try{
    y.lastIndexOf(new Object(), -32760);c_array[c_index] = c_array.filter((arg) => { return (arg == new Object()) }); } catch(e){ } });c_array.reverse()
267 try { someRegex1[Symbol.search]("WUtazcQunqxEnKAbPkeIfNoQnSpOwQULMUoDVf") } catch (e) { }
268 try { someSet1.entries() } catch (e) { }
269 try { someWeakSet1.delete(function() {})} catch (e) { }
270 try { Object.getOwnPropertyNames(someWeakSet1) } catch (e) { }
271 try { someString1.hasOwnProperty("toString") } catch (e) { }
272 try { for (var element in someObject1) { try{ someObject1[element] = someObject1[element].concat(); } catch(e) { } } } catch (e) { }
273 try { someTypedArray1 = new Uint16Array(someArrayBuffer1, 114) } catch (e) { }
274 try { someIntlNumberFormat1.formatToParts(+17064) } catch (e) { }
275 try { Math.sinh(11582) } catch (e) { }
276 try { someWeakSet1.add(someTypedArray1) } catch (e) { }
277 try { someDataView1.getFloat64(250, false) } catch (e) { }
278 try { Intl.NumberFormat.supportedLocalesOf("fi-FI") } catch (e) { }
279 try { someArray1[0] = someRegex1.test(String.fromCharCode(669014) + "prAmHEKKXgdQBgytdTnyQnd") } catch (e) { }
280 try { someWeakSet1.delete(someObject1) } catch (e) { }
281 try { Intl.DateTimeFormat.supportedLocalesOf("ar-LB-u-hc-h11-nu-beng") } catch (e) { }
282 try { Intl.NumberFormat.supportedLocalesOf("es-PA-u-nu-kali") } catch (e) { }
283 try { for(var index=0; index < 7; index++){ someArray1[index] = someArray1.entries(); } } catch (e) { }
284 try { for(var index=0; index < 8; index++){ someArray1[index] = someArray1.join("iRq"); } } catch (e) { }
```

```
libcsu_init EXPORT __libc_csu_init
; DATA XREF: _start+1Ato
; .text:ini7to
PUSH.W {R3-R9,LR}
MOV R7, R0
LDR R6, =( __do_global_dtors_aux_fini
MOV R8, R1
LDR R5, =( frame_dummy_init_array_end
MOV R9, R2
ADD R6, PC ; __do_global_dtors_aux_fi
BLX .init_proc
ADD R5, PC ; __frame_dummy_init_array
SUBS R6, R6, R5
```

```
__libc_csu_fini EXPORT __libc_csu_fini
; DATA XREF: _start+10to
; .text:off 10398to
BX LR
; End of function __libc_csu_fini
```

# Effective Mutation

- Don't need neural networks or genetic mutations really
  - Just take an input that you know has caused coverage, and add it to your corpus
  - If a mutation on that test case causes new coverage, add it to corpus

```
; ***** SUBROUTINE *****  
  
__libc_csu_init      EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: __start+1Ato  
                    ; .text:00000000  
                    PUSH.W      {R3-R9,LR}  
                    MOV         R7, R0  
                    LDR         R6, =(__do_global_dtors_aux_fini_array_entry - 0x104FC)  
                    MOV         R8, R1  
                    LDR         R5, =(__frame_dummy_init_array_entry - 0x104E0)  
                    MOV         R9, R2  
                    ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry - 0x104FC  
                    BLX         .init_proc  
                    MOV         R5, PC ; __frame_dummy_init_array_entry - 0x104E0  
                    R6, R6, R5  
                    R6, R6, #2  
                    BEQ         locret_104F8  
                    MOV         R4, #0  
  
loc_104E6             ; CODE XREF: __libc_csu_init+00000000  
                    ADDS         R4, #1  
                    LDR.W        R3, [R5],#4  
                    MOV         R2, R9  
                    MOV         R1, R8  
                    BLA         R3, R4, .frame_dummy  
                    CMP         R4, R6  
                    BNE         loc_104E6  
  
locret_104F8         ; CODE XREF: __libc_csu_init+00000000  
                    POP.W        {R3-R9,PC}  
; End of function __libc_csu_init  
  
off_104FC            DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                    ; DATA XREF: __libc_csu_init+00000000  
off_10500            DCD __frame_dummy_init_array_entry - 0x104E0  
                    ; DATA XREF: __libc_csu_init+00000000  
  
; ***** SUBROUTINE *****  
  
__libc_csu_fini      EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: __start+10to  
                    ; .text:00000000  
                    BX          LR  
; End of function __libc_csu_fini  
  
; *****
```



# LibFuzzer

- LLVM Fuzzer
- Can combine the LLVM Sanitizers for an excellent in-memory coverage based fuzzer
- No processes spawning, no recv() calls, less transitions away from the code you want to target



# LibFuzzer

- Implements techniques already described
  - SanitizerCoverage module increments into a per-module bitmap (edge coverage)
  - Table of recent comparisons to record literal cmp's
- Ultimately very fast, and very precise

# LibFuzzer Example

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    uint8_t Uncompressed[100];
    size_t UncompressedLen = sizeof(Uncompressed);
    if (Z_OK != uncompress(Uncompressed, &UncompressedLen, Data, Size))
        return 0;
    if (UncompressedLen < 2) return 0;
    if (Uncompressed[0] == 'F' && Uncompressed[1] == 'U')
        abort(); // Boom
    return 0;
}
```

# LibFuzzer Example

```
extern "C" size_t LLVMFuzzerCustomMutator(uint8_t *Data, size_t Size,
                                         size_t MaxSize, unsigned int Seed) {

    uint8_t Uncompressed[100];
    size_t UncompressedLen = sizeof(Uncompressed);
    size_t CompressedLen = MaxSize;
    if (Z_OK != uncompress(Uncompressed, &UncompressedLen, Data, Size)) {
        // The data didn't uncompress. Return a dummy...
    }
    UncompressedLen =
        LLVMFuzzerMutate(Uncompressed, UncompressedLen, sizeof(Uncompressed));
    if (Z_OK != compress(Data, &CompressedLen, Uncompressed, UncompressedLen))
        return 0;
    return CompressedLen;
}
```

# If all you have is a hammer...

- LibFuzzer has mostly a binary-file format esque mutator (understandable)
- To get around this, generate a grammar that maps raw bytes -> your input structure and mutate that
- **LibProtobuf-Mutator** makes this faster

# Case Study: Fuzzing Clang

- Structure-aware fuzzing for Clang and LLVM with libprotobuf-mutator

```
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1Ato
; .text:00000000
MOVW R3-R9,LR
R7, R0
R6, =(__do_global_dtors_aux_fini_array_entry)
R8, R1
R5, =(__frame_dummy_init_array_entry)
MOV R9, R2
ADD R6, PC ; __do_global_dtors_aux_fini_array_entry
BLX __init_proc
ADP R5, #0
MOV R8, #0
BEQ loc_104F8
MOV R4, #0
loc_104E6
; CODE XREF: __libc_csu_init
ADDS R4, #1
LDR.W R3, [R5],#4
MOV R2, R9
MOV R1, R8
MOV R0, R7
BLX R3 ; frame_dummy
CMP R4, R6
BNE loc_104E6
locret_104F8
; CODE XREF: __libc_csu_init
POP.W {R3-R9,PC}
; End of function __libc_csu_init
; -----
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init
off_10500 DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init
; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+10to
; .text:00000000
BX LR
; End of function __libc_csu_fini
; -----
```

# Case Study: Fuzzing Clang

- Protobuf syntax:

```
message Person {  
    required string name = 1;  
    required int32 id = 2;  
    optional string email = 3;  
}
```

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init  
; DATA XREF: __start+1Ato  
; .text:00000000  
{R3-R9,LR}  
R7, R0  
R6, =(__do_global_dtors_aux_fini  
R8, R1  
MOV  
LDR  
R5, =(__frame_dummy_init_array_entr  
MOV  
R9, R2  
ADD  
R6, PC ; __do_global_dtors_aux_fi  
BLX  
.init_proc  
ADD  
R5, PC ; __frame_dummy_init_array  
SUBS  
R6, R6, R5  
ASRS  
R6, R6, #2  
BEQ  
locret_104F8  
MOVS  
R4, #0  
  
loc_104E6  
; CODE XREF: __libc_csu_in  
ADDS  
R4, #1  
LDR.W  
R3, [R5],#4  
MOV  
R2, R9  
MOV  
R1, R8  
MOV  
R0, R7  
BLX  
R3 ; frame_dummy  
CMP  
R4, R6  
BNE  
loc_104E6  
  
locret_104F8  
; CODE XREF: __libc_csu_in  
POP.W  
{R3-R9,PC}  
; End of function __libc_csu_init  
; -----  
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_in  
off_10500 DCD __frame_dummy_init_array_entry - 0x10500  
; DATA XREF: __libc_csu_in  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini  
; DATA XREF: __start+10to  
; .text:00000000  
BX  
LR  
; End of function __libc_csu_fini  
; -----  
ALDCPU
```

# Case Study: Fuzzing Clang

1. Define proto that “looks like” C/C++ AST
2. Map arbitrary byte array -> libprotobuf
3. Mutate protobuf object with LPM
4. Turn protobuf into C-like syntax
5. Run test case

# Case Study: Fuzzing Clang

## clang-proto-fuzzer trophies

[null deref in llvm::ScalarEvolution::getMulExpr](#)

```
void foo(int *a) {  
    while (1) {  
        a[60] = ((1 + a[60]) + a[0]);  
        while ((a[60] + a[0])) {  
            a[0] = (a[0] + 1);  
        }  
    }  
}
```





# Case Study: Fuzzing Clang

- Related: Attacking Chrome IPC: Reliably finding bugs to escape the Chrome sandbox

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: __start+1Ato
; .text:00000000
{R3-R9,LR}
R7, R0
R6, =(__do_global_dtors_aux_fini_array_entry)
R8, R1
R5, =(__frame_dummy_init_array_entry)
MOV R9, R2
ADD R6, PC ; __do_global_dtors_aux_fini
BLX .init_proc
R5, PC ; __frame_dummy_init_array
R6, R0
R6, R0
BEQ locret_104F8
MOV R4, #0
; CODE XREF: __libc_csu_init
ADD R4, #1
LDR.W R3, [R5],#4
MOV R2, R9
MOV R1, R8
MOV R0, R7
BLX R3 ; frame_dummy
CMP R4, R6
BNE locret_104E6

locret_104F8
; CODE XREF: __libc_csu_init
POP.W {R3-R9,PC}
; End of function __libc_csu_init

; -----
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init
off_10500 DCD __frame_dummy_init_array_entry - 0x104E0
; DATA XREF: __libc_csu_init

; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: __start+10to
; .text:00000000
BX LR
; End of function __libc_csu_fini
; -----
```

# Sanitizers

- ASAN, UBSAN, MSAN, TSAN, COVSAN
- Runtime checking for memory corruption, undefined behavior, race conditions, uninitialized memory, etc
- You want these.

```
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_init  
__libc_csu_init                                ; DATA XREF: _start+1Ato  
                                                ; .text:ini7to  
  
PUSH.W                                         {R3-R9,LR}  
MOV                                            R7, R0  
LDR                                            R6, =( __do_global_dtors_aux_fini  
MOV                                            R8, R1  
LDR                                            R5, =( frame_dummy_init_array_enti  
  
MOV                                            R9, R2  
ADD                                            R6, PC ; __do_global_dtors_aux_fi  
BLX                                           .init_proc  
MOV                                            R5, PC ; __frame_dummy_init_array  
R6, R6, R5  
R6, R6, #2  
BEQ     locret_104F8  
MOV                                            R4, #0  
  
; CODE XREF: __libc_csu_in  
R4, #1  
LDR.W                                         R3, [R5],#4  
MOV                                            R2, R9  
MOV                                            R1, R8  
MOV                                            R0, R7  
BLX                                           R3 ; frame_dummy  
CMP                                            R4, R6  
BNE                                           loc_104E6  
  
locret_104F8                                ; CODE XREF: __libc_csu_in  
POP.W                                         {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
; DATA XREF: __libc_csu_in  
off_10500 DCD __frame_dummy_init_array_entry - 0x104E0  
; DATA XREF: __libc_csu_in  
  
; ***** SUBROUTINE *****  
  
EXPORT __libc_csu_fini  
__libc_csu_fini                                ; DATA XREF: _start+10to  
                                                ; .text:off_10390to  
  
BX                                             LR  
; End of function __libc_csu_fini  
  
; *****  
ALDCM1
```

# Address Sanitizer

- Catches buffer-out-of-bounds accesses (reads and writes)
- Usually the go-to as the bare minimum for the bugs you probably want anyways

# Ripping Out Code

- May be useful to just rip out the function
  - Definitely perf to gain
- Don't jump through hurdles to get to vulnerable functions
- Make sure to keep the same assumptions as it would have in the original program

```

; ----- SUBROUTINE -----
__libc_csu_init
EXPORT __libc_csu_init
; DATA XREF: _start+1Ato
; .text:00000000
__libc_csu_init
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =(__do_global_dtors_aux_fini
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entr
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fi
BLX         .init_proc
ADD         R5, PC ; __frame_dummy_init_array
MOV         R6, R5
MOV         R6, R5
BEQ         locret_104F8
MOVS        R4, #0

loc_104E6
; CODE XREF: __libc_csu_in
ADDS        R4, #1
LDR.W       R3, [R5],#4
MOV         R2, R9
MOV         R3, R2
BLX         R3 ; frame_dummy
CMP         R4, R6
BNE         loc_104E6

locret_104F8
; CODE XREF: __libc_csu_in
POP.W       {R3-R9,PC}
; End of function __libc_csu_init

; ----- SUBROUTINE -----
__libc_csu_fini
EXPORT __libc_csu_fini
; DATA XREF: _start+10to
; .text:00000000
__libc_csu_fini
BX          LR
; End of function __libc_csu_fini
; ----- SUBROUTINE -----

```

# Documenting the source(??)

- Add assumptions that aren't enforced as runtime asserts
- Log state to check for invalid transitions
- Harden the source as you fuzz it

# Big Questions (1)

- How do I improve fuzzing performance?
  - Remove syscalls? Seriously, all of them
  - Skip code that you don't care about... (easier said than done of course)

```
; ***** SUBROUTINE *****  
  
__libc_csu_init      EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: _start+1Ato  
                    ; .text:00000000  
                    PUSH.W      {R3-R9,LR}  
                    MOV         R7, R0  
                    LDR         R6, =( __do_global_dtors_aux_fini_array_entry - 0x104FC)  
                    MOV         R8, R1  
                    LDR         R5, =( __frame_dummy_init_array_entry - 0x10500)  
                    MOV         R9, R2  
                    ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry  
                    BLX         .init_proc  
                    ADD         R5, PC ; __frame_dummy_init_array_entry  
                    MOV         R6, R5  
                    MOV         R6, R6, #2  
                    BEQ         locret_104FB  
                    MOV         R4, #0  
                    ; CODE XREF: __libc_csu_init+00000000  
                    ADDS         R4, #1  
                    LDR.W        R3, [R5], #4  
                    MOV         R2, R0  
                    MOV         R0, R3  
                    MOV         R0, R7  
                    BLX         R3 ; frame_dummy  
                    CMP         R4, R6  
                    BNE         loc_104E6  
  
locret_104FB      ; CODE XREF: __libc_csu_init+00000000  
                    POP.W        {R3-R9,PC}  
; End of function __libc_csu_init  
  
; *****  
off_104FC      DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                    ; DATA XREF: __libc_csu_init+00000000  
off_10500      DCD __frame_dummy_init_array_entry - 0x104E0  
                    ; DATA XREF: __libc_csu_init+00000000  
; ***** SUBROUTINE *****  
  
__libc_csu_fini      EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: _start+10to  
                    ; .text:00000000  
                    BX          LR  
; End of function __libc_csu_fini  
; *****
```

# Big Questions (2)

- At what point does my mutator output this?
  - Useful for scenarios like JS engines
  - Can design a fuzzer after-the-fact to find this bug, and hope it finds new bugs. Chicken-And-Egg problem

```
; ***** SUBROUTINE *****  
  
__libc_csu_init      EXPORT __libc_csu_init  
__libc_csu_init      ; DATA XREF: _start+1Ato  
                    ; .text:00000000  
                    PUSH.W      {R3-R9,LR}  
                    MOV         R7, R0  
                    LDR         R6, =(__do_global_dtors_aux_fini_array_entry)  
                    MOV         R8, R1  
                    LDR         R5, =(__frame_dummy_init_array_entry)  
                    MOV         R9, R2  
                    ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry  
                    BLX         .init_proc  
                    ADD         R5, PC ; __frame_dummy_init_array_entry  
                    LDR         R3, [R5, #0]  
                    BEQ         locret_104F8  
                    MOV         R4, #0  
loc_104E6:           ADDS         R4, #1  
                    LDR.W       R3, [R5], #4  
                    MOV         R7, R0  
                    MOV         R8, R1  
                    BLX         .init_proc  
                    CMP         R3, R6  
                    BEQ         locret_104E6  
locret_104F8:       POP.W       {R3-R9,PC} ; CODE XREF: __libc_csu_init  
; End of function __libc_csu_init  
  
; *****  
off_104FC:         DCD __do_global_dtors_aux_fini_array_entry - 0x104FC  
                    ; DATA XREF: __libc_csu_init  
off_10500:         DCD __frame_dummy_init_array_entry - 0x104E0  
                    ; DATA XREF: __libc_csu_init  
  
; ***** SUBROUTINE *****  
  
__libc_csu_fini      EXPORT __libc_csu_fini  
__libc_csu_fini      ; DATA XREF: _start+10to  
                    ; .text:00000000  
                    BX          LR  
; End of function __libc_csu_fini  
  
; *****
```

# Fuzzing as Exploit Assistance

- Can't find a good primitive? Search space is relatively small?
- Let fuzzer run with crashing input and analyze the different crashes
- Cross fingers and pray for rip = 0x41414141



# Fuzzing as Exploit Assistance

- Chainsawing is smarter:
  - Write a harness to trigger different allocations
  - Groom LFH, different bucket sizes, different objects, etc
  - Trigger bug at end
  - Cross fingers and pray for deref \*0x41414141

# Fuzzing Take-Away

- Make every mutation count
- Make every iteration fast, only run the code you want to target, don't waste cpu cycles
- But before you hyper-optimize, get your dumb fuzzer up and running.
  - It doesn't cost you anything!

```

; ***** SUBROUTINE *****
EXPORT __libc_csu_init
__libc_csu_init
; DATA XREF: _start+1Ato
; .text:00000000
PUSH.W      {R3-R9,LR}
MOV         R7, R0
LDR         R6, =(__do_global_dtors_aux_fini_array_entry)
MOV         R8, R1
LDR         R5, =(__frame_dummy_init_array_entry)
MOV         R9, R2
ADD         R6, PC ; __do_global_dtors_aux_fini_array_entry
BLX         .init_proc
ADD         R5, PC ; __frame_dummy_init_array_entry
SUBS       R6, R6, R5
ASRS       R6, R6, #2
BEQ        locret_104F8
MOVS       R4, #0

; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:00000000
BX         LR
; End of function __libc_csu_fini
; ***** SUBROUTINE *****

```

```

loc_104FC: DCD __do_global_dtors_aux_fini_array_entry - 0x104FC
; DATA XREF: __libc_csu_init
off_10500: DCD __frame_dummy_init_array_entry - 0x10500
; DATA XREF: __libc_csu_init
; ***** SUBROUTINE *****
EXPORT __libc_csu_fini
__libc_csu_fini
; DATA XREF: _start+10to
; .text:00000000
BX         LR
; End of function __libc_csu_fini
; ***** SUBROUTINE *****

```