ME AND YOU AND EVERYONE WE DEBUG

with the same trace

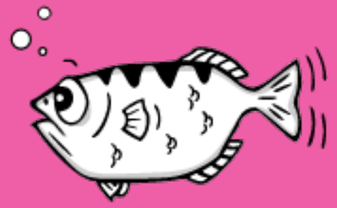

))<>((
forever

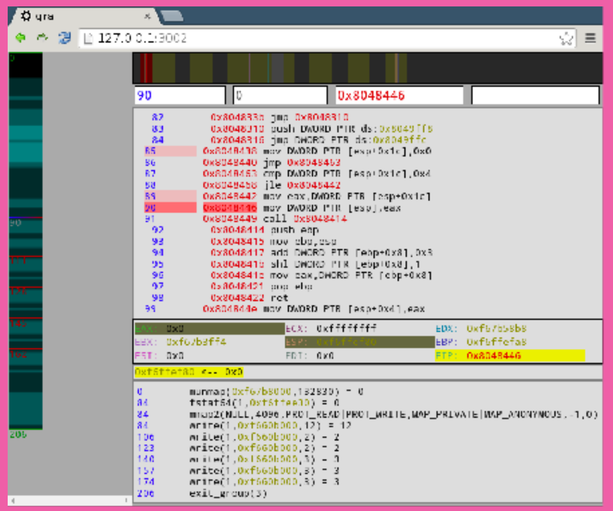
```
$ id
uid=65534(nobody)
gid=65534(nogroup)
groups=65534(nogroup)
```

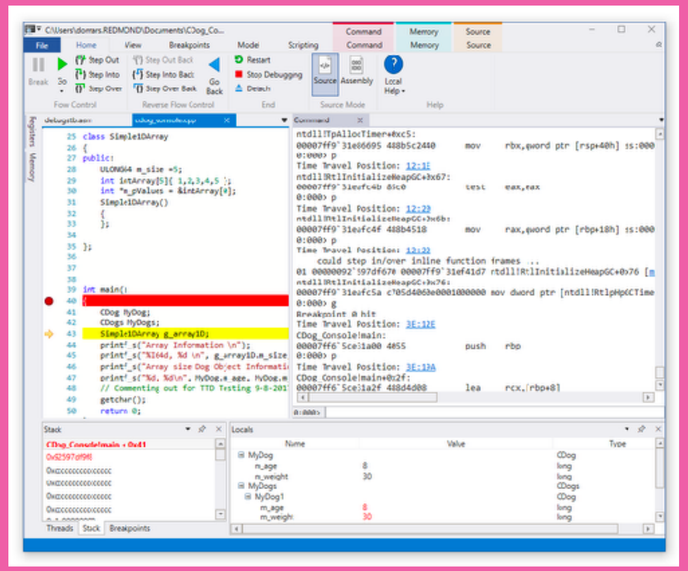
- only 🇺🇸 r2 dev?
- air djibouti 🇩🇯 passenger 🛬💥
- recent apple convert
- fuzzing every day
- wishing for better debugger tools
- macOS, x86_64 only, to make things easy
- radare2 user/dev for 6+ years
  - computer user for 10 years
- stupidly told my manager i could build a reversable debugger in a weekend

(d  r) - 

existing tech/strategies



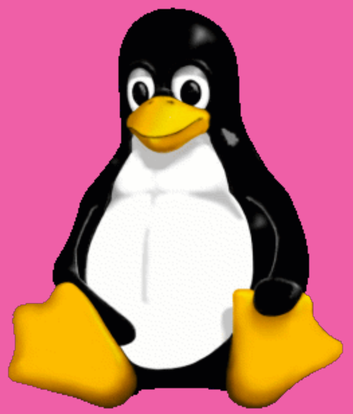




rr

rr (almost like r2 ;))

- "rr records nondeterministic executions and debugs them deterministically" : awesome!
  - mozilla project from 2014 for debugging difficult to reproduce firefox bugs.
- built on several clever linux kernel hacks, performance counters for replay, etc. There's a whitepaper, it's open source.
- requires linux kernel, not really portable.



microsoft ttt + windbg

windows only ofc... i asked some msftie friends, but didn't do much digging myself

radare2

- ren kimura's gsoc project in 2017
- record and replay (rnr), with diffs, checkpoints, and all sorts of great features.
- https://asciinema.org/a/nR8jzJwHj9bkzo8q6hTwHOZL4
- unfortunately it's either WAY too slow, or doesn't work on macOS.
  - several minutes for a single `dsb`
  - i didn't really want to go digging (sorry!)

gdb

- on some platforms "process record" exists.
- if an instruction is in the record log, take the effect of the log, otherwise execute. perfect for both forward and reverse debugging!
  - macOS is not one of these platforms
  - i ported this in a weekend, super easy to do.
    - 500 line patch to gdb for basic support.
    - many instructions are unsupported, not a big deal, right?

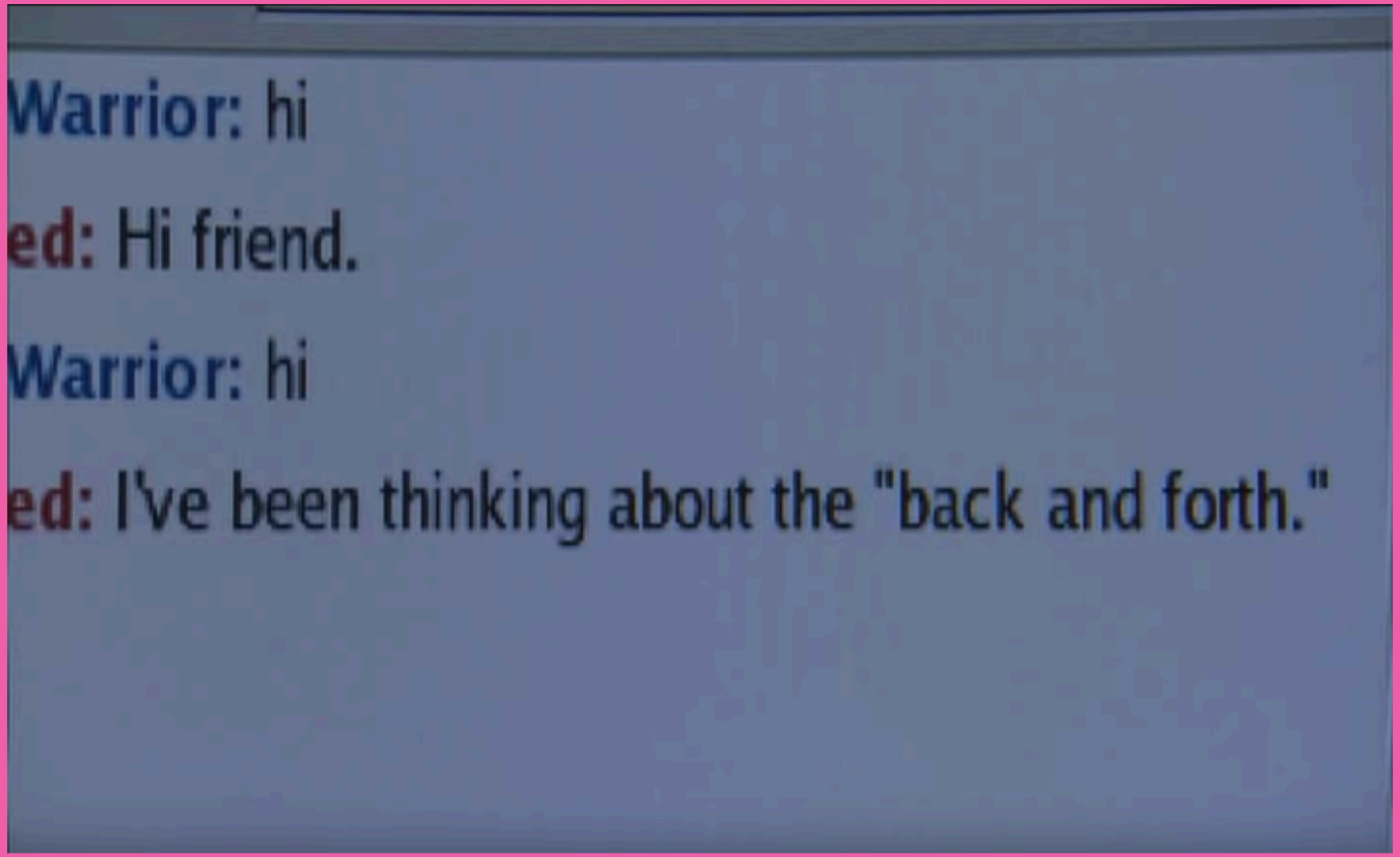
gdb

- xsave/xrstor not supported. used heavily in libSystem. Everything links against libSystem...
- xrstor required a similar length patch. as did a few other instructions before i gave up
- doing this so many times sucks, it's really tough to make sure you do it right, intel manual is very very dense.
- (ask me and i'll give you the patches), don't think they apply cleanly anymore

```
445 +       case 4:    /* xsave */
446 +         {
447 +             if (ir.mod == 3) {
448 +                 opcode = (opcode << 8) | ir.modrm;
449 +                 goto no_support;
450 +             }
451 +             ULONGEST rfbm_eax, rfbm_edx;
452 +
453 +             regcache_raw_read_unsigned (regcache, I386_EAX_REGNUM, &rfbm_eax);
454 +             rfbm_eax &= (1ULL << 32) - 1;
455 +
456 +             regcache_raw_read_unsigned (regcache, I386_EDX_REGNUM, &rfbm_edx);
457 +             rfbm_edx &= (1ULL << 32) - 1;
458 +
459 +             uint64_t rfbm = tdep->xcr0 & (rfbm_eax | rfbm_edx << 32);
460 +
461 +             if (rfbm & ~X86_XSTATE_ALL_MASK) {
462 +                 printf_unfiltered (_("Process record does not support XSAVE "
463 +                                      "instruction with RFBM=%" PRIx64 ".\n"),
464 +                                    rfbm);
465 +                 opcode = (opcode << 8) | ir.modrm;
466 +                 goto no_support;
467 +             }
468 +
469 +             uint64_t tmpu64;
470 +             if (i386_record_lea_modrm_addr (&ir, &tmpu64)) {
471 +                 return -1;
472 +             }
473 +
474 +             uint64_t legacy_size = 160; /* x87 xsave size */
475 +             if (rfbm & X86_XSTATE_SSE) {
476 +                 if (ir.regmap[X86_RECORD_R8_REGNUM])
477 +                     legacy_size = 416; /* 64 bit sse xsave size */
478 +                 else
479 +                     legacy_size = 288; /* 32 bit sse xsave size */
480 +             }
481 +             if (record_full_arch_list_add_mem (tmpu64, legacy_size))
482 +                 return -1;
483 +
484 +             uint64_t size = X86_XSTATE_SIZE (rfbm) - 512;
485 +             if (record_full_arch_list_add_mem (tmpu64 + 512, size))
486 +                 return -1;
487 +         }
488 +         break;
489 +
```

gdb

- i gave this demo to my team at work, only showed code that was supported, won my "bet", ignored this topic for a few months.


Warrior: hi
ed: Hi friend.
Warrior: hi
ed: I've been thinking about the "back and forth."

qira

- qemu-user powered trace recorder and replayer
- some suboptimal webui with crazy colors i don't quite understand, but not a big dealbreaker
- qemu-user only works on linux applications on linux hosts, so that doesn't really work.
- but...
  - there's an outdated pin tracer, i386 only, but pin is cross platform. perfect place to start. my work is mostly built by reading qira sources

so, what'd i build?

- plugin for r2 for deterministically replaying a single execution trace.

so, what'd i build?

- replay trace back on any machine, on any architecture
- (but must be recorded on intel x86)

so, what'd i build?

- full record and replay, the entire program execution is recorded and nothing is emulated
- support of sse/avx/mmx/other fancy instruction with zero effort

pin

- intel's tool for binary instrumentation
- support linux, macOS, and windows
- pin runtime lacks rtti, tools cannot use exceptions
- non-free PinCRT, but tools can be open source
- i'd prefer dynamorio, but doesn't work on macOS, so let's dig in

how does it work

- actually very simple
  - ∀ images loaded -> record base addresses
  - ∀ instruction executed -> record register set
  - ∀ memory write -> record address and value (qword granularity)

how does it work

- all register/memory access/writes logged to text file.
- single exection of /bin/ls on macOS -> ~20mb log
- totally not-optimized, but overhead is quite high on the initial execution. once the data has been preprocessed, it's not so bad to work with though

hooking at runtime

- Pin allows you to insert hooks on certain events, so I insert several.

hooking at runtime (loader)

```
IMG_AddInstrumentFunction(ImageLoad, 0);
```

- if it's the first image, mark the highest and lowest addresses. we can use this for later filtering*
- otherwise just list all sections mapped, high and low addresses
    - macOS has non-contiguous image loading for some reason (performance maybe?

hooking at runtime (loader)

```
0bb78000-0bb82000 0 /bin/ls
0d293000-0d35e000 0 /usr/lib/dyld
66206000-66208000 0 /usr/lib/libSystem.B.dylib
9f1be000-9f1be2d8 38fb8000 /usr/lib/libSystem.B.dylib
d1eca000-dfd023d6 6bcc4000 /usr/lib/libSystem.B.dylib
66449000-6649d000 0 /usr/lib/libc++.1.dylib
9f1ea000-9f1f1cf8 38da1000 /usr/lib/libc++.1.dylib
d1eca000-dfd023d6 6ba81000 /usr/lib/libc++.1.dylib
6649d000-664b3000 0 /usr/lib/libc++abi.dylib
9f1f2000-9f1f5210 38d55000 /usr/lib/libc++abi.dylib
d1eca000-dfd023d6 6ba2d000 /usr/lib/libc++abi.dylib
67487000-674b8000 0 /usr/lib/libncurses.5.4.dylib
9f4cd000-9f4d0e4c 38046000 /usr/lib/libncurses.5.4.dylib
d1eca000-dfd023d6 6aa43000 /usr/lib/libncurses.5.4.dylib
67a3d000-681c3000 0 /usr/lib/libobjc.A.dylib
9f66a000-9f7f61d0 37c2d000 /usr/lib/libobjc.A.dylib
d1eca000-dfd023d6 6a48d000 /usr/lib/libobjc.A.dylib
6878a000-6878e000 0 /usr/lib/libutil.dylib
9f973000-9f973410 371e9000 /usr/lib/libutil.dylib
d1eca000-dfd023d6 69740000 /usr/lib/libutil.dylib
690a2000-690a7000 0 /usr/lib/system/libcache.dylib
9fa67000-9fa67128 369c5000 /usr/lib/system/libcache.dylib
d1eca000-dfd023d6 68e28000 /usr/lib/system/libcache.dylib
690a7000-690b2000 0 /usr/lib/system/libcommonCrypto.dylib
9fa68000-9fa694b0 369c1000 /usr/lib/system/libcommonCrypto.dylib
d1eca000-dfd023d6 68e23000 /usr/lib/system/libcommonCrypto.dylib
690b2000-690ba000 0 /usr/lib/system/libcompiler_rt.dylib
9fa6a000-9fa6b0b0 369b8000 /usr/lib/system/libcompiler_rt.dylib
d1eca000-dfd023d6 68e18000 /usr/lib/system/libcompiler_rt.dylib
690ba000-690c4000 0 /usr/lib/system/libcopyfile.dylib
9fa6c000-9fa6c6f0 369b2000 /usr/lib/system/libcopyfile.dylib
```

hooking at runtime (loader)

- this allows us to load each dependency into the r2 session during the trace replay for visualization.
- o [file]; om fd vaddr [size] [paddr] [rwx] [name]
  - do this for each library loaded
- (note, there's the downside if an image gets unmapped and then something else gets mapped back into our section, then, we're totally fucked [workarounds planned])

hooking at runtime (instruction)

```
INS_AddInstrumentFunction(Instruction, 0);
```

- callback on every instruction called. the overhead isn't actually that bad as pin uses a JIT (beyond the scope of my understanding of how pin works...)
- trace lots of stuff...

hooking at runtime (instruction)

```
if (!filtered) INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)DumpRegsI,
        IARG_INST_PTR,
        IARG_REG_VALUE, REG_RAX,
        IARG_REG_VALUE, REG_RBX,
        IARG_REG_VALUE, REG_RCX,
        IARG_REG_VALUE, REG_RDX,
        IARG_REG_VALUE, REG_RSI,
        IARG_REG_VALUE, REG_RDI,
        IARG_REG_VALUE, REG_RBP,
        IARG_REG_VALUE, REG_RSP,
        IARG_REG_VALUE, REG_R8,
        IARG_REG_VALUE, REG_R9,
        IARG_REG_VALUE, REG_R10,
        IARG_REG_VALUE, REG_R11,
        IARG_REG_VALUE, REG_R12,
        IARG_REG_VALUE, REG_R13,
        IARG_REG_VALUE, REG_R14,
        IARG_REG_VALUE, REG_R15,
        IARG_END);
```

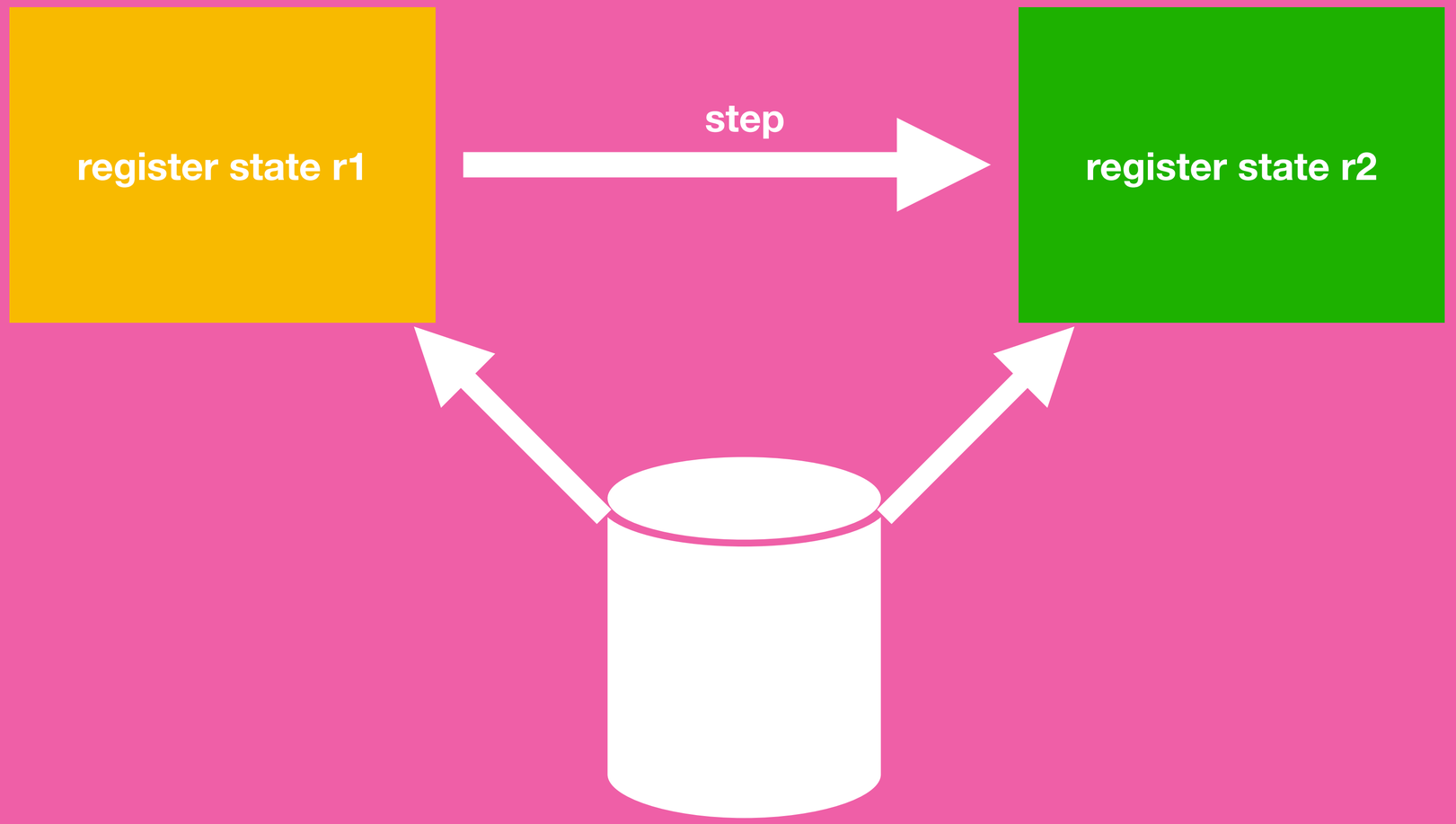
```
    fprintf(pFile2, "I %llx %llx %llx %llx %llx %llx %llx %llx %llx %llx %llx %llx %llx %llx %llx
 %llx %llx\n", (ADDRINT)ip, rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp, r8, r9, r10, r11, r12, r13, r
14, r15);
```

- for every instruction executed, log all registers
  - literally just logging them out to a file
  - every register, on every instruction

hooking at runtime (instruction)

- this is second iteration. first was delta diffing,
- stored all diffs (only changed registers) in a messagepack blob, smaller, but actually took up a lot more compute time later, so abandoned.
- can filter to just one particular binary, in case we only want to trace in a certain lib or the main image (probably only want to record/replay in the crashing image or two)

hooking at runtime (instruction)



BUT IT ISN'T ENOUGH!

hooking at runtime (memory)

- dumping the stack and heap is also important to see how heap or stack could get corrupted in the case of a fuzzing crash.
- it's stupid/impossible to dump and restore the entired mapped memory space on each instruction executed.
  - too expensive in terms of both disk and memory for sure
  - can we do better?

hooking at runtime (memory)

```
INS_InsertPredicatedCall(
    ins, IPOINT_BEFORE, (AFUNPTR)RecordMemWrite1, IARG_THREAD_ID,
    IARG_MEMORYOP_EA, i,
    IARG_REG_VALUE, writeea_scratch_reg,
    IARG_RETURN_REGS, writeea_scratch_reg,
    IARG_END
);
```

- track all memory writes, do it at the qword level
- in pseudocode explanation
- if (memory_is_written) {
  if (size > 8) pad_with_rest_of_value();
  store_to_database(address, value);
  }

```
if (flags & (IS_MEM|IS_WRITE)) {
    fprintf(pFile2, "M %llx %llx\n", addr, data);
}
```

so what do we have now

- a kind of super useless giant file 20+MB on single exection of /bin/ls on my system

- over ~1.4M lines long (filtered to main binary!)

let's do something useful

- convert it into a database for fast lookups! using sqlite3 here, so it's easy/portable

```
sqlite> .tables
memory     registers
sqlite> .schema registers
CREATE TABLE registers (
    idx int,
    rip int,
    rax int,
    rbx int,
    rcx int,
    rdx int,
    rsi int,
    rdi int,
    rbp int,
    rsp int,
    r8 int,
    r9 int,
    r10 int,
    r11 int,
    r12 int,
    r13 int,
    r14 int,
    r15 int
  );
```

- each register state

let's do something useful

- but memory writes aren't as easy, there can be multiple memory writes in a single instruction

- or writes between unfiltered instructions

```
sqlite> .schema memory
CREATE TABLE memory(
    lastidx int,
    address int,
    value int
  );
```

convert to this data?

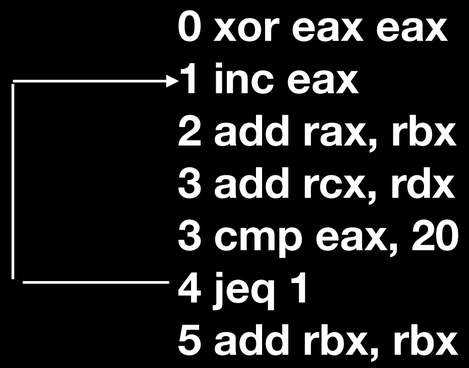
- the trace has memory writes interleaved with register traces, so preprocess the entire log into a database
- idx is the instruction index, lastidx is the previous instruction that a memory state is related to. so when we step, execute all memory diffs where lastidx < targetidx

dst [count]

what does it do?

dst [count]

- naive approach
- registers are always easy, just load the previous one from the database
- for memory, we don't store the previous memory set, so... how
  - set "breakpoint" -> execute from 0 until breakpoint is hit
  - this doesn't work imagine a loop


0 xor eax eax
1 inc eax
2 add rax, rbx
3 add rcx, rdx
3 cmp eax, 20
4 jeq 1
5 add rbx, rbx

dst [count]

```
stm = db.prepare "SELECT * FROM memory where lastidx = #{idx}"
rs = stm.execute
ncnt = db.prepare("SELECT COUNT(*) from memory where lastidx = #{idx}").execute.next[0]
unless ncnt > too_many_mems
  while (row = rs.next) do
    addr = row[1]
    s2 = db.prepare "SELECT * FROM memory where lastidx < #{idx} and address = #{addr}"
    rs2 = s2.execute
    while (r2 = rs2.next) do
      addr2 = row[1]
      val2 = bin_to_hex([row[2]].pack("Q"))
      call2 = "http://localhost:9090/cmd/wx #{val} @ #{addr}"
      cu2 = URI::encode(call2)
      HTTParty.get(cu2)
    end
  end
end
```

dst [count]

- a little more clever i think
- on each step backwards, just write the previous value of all memory writes to the address that was written in this "rewinded" instruction.
- there can be "stale" writes laying around, but this is pretty good i think. that can also be worked around (not yet done)

demo time

so what do we have/not have

- have
  - trace replayer
  - step forward and backwards
  - r2 integration, inspect memory/print structs just how you normally would
  - support for all intel x86_64 platforms supported by PIN
    - macOS, Linux, Windows

so what do we have/not have

- DON'T have
  - "true" debugger
  - must have trace recorded to be useful (eg. have to have caught a rare crash under the tracer to load into r2).
  - ability to see stack frames (no backtrace for example)
  - thread race/interaction usability
    - possible in the future, if multiple traces are done, one for each thread (pin supports this, i don't)

//TODO

- cutter plugin
- coalesce memory writes for efficiency
- rm ruby, rewrite in c (it's slow as is)
- add tunables for what parts to trace/not trace
- dwarfdump integration for source level debugging
- release (soon, need to cleanup/rm trash parts, sorry intended to do this earlier)


QUESTIONS?