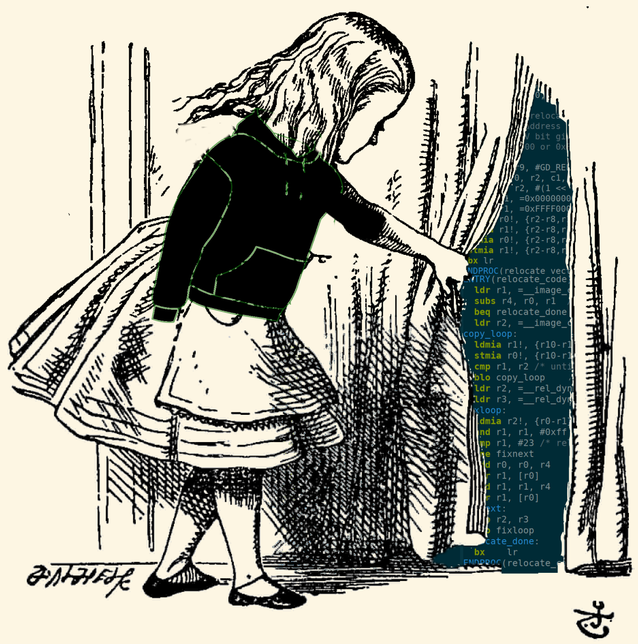
Bushwhacking your way around a bootloader

Rebecca ".bx" Shapiro

2018.11.16



Tools and techniques for traversing treacherous code bases

-or- How I managed to develop understanding of U-Boot

Blackhoodie Berlin

whoami

Dr. .bx

- Senior security researcher @ Narf Industries
- Studied w/ Sergey Bratus & the Dartmouth Trust Lab
- Commander of ELF metadata-based weird machines
- ELF, bootloaders,
- Dynamic analysis
- Defensive research (more or less) with a dash of reverse engineering
- [at]bxsays on Twitter

Photo circa Sept 2018

Meet Das U-Boot bootloader

```
U-Boot 2010.12-xes_r3 (Aug 25 2011 - 11:04:04)

CPU0:  P2020E, Version: 1.0, (0x80ea0010)
Core:  E500, Version: 4.0, (0x80211040)
Clock Configuration:
       CPU0:1066.667 MHz, CPU1:1066.667 MHz,
       CCB:533.333 MHz,
       DDR:400  MHz (800 MT/s data rate) (Asynchronous), LBC:133.333 MHz
L1:    D-cache 32 kB enabled
       I-cache 32 kB enabled
Board: X-ES XPedite5501 PMC/XMC SBC
       Rev SA, Serial# 36093001, Cfg 90015130-1
I2C:   ready
DRAM:  2 GiB (DDR3, 64-bit, CL=6, ECC on)
FLASH: Executed from FLASH1
POST memory PASSED
FLASH: 256 MiB
L2:    512 KB enabled
NAND:  4096 MiB
PCIE1: connected as Root Complex (no link)
PCIE1: Bus 00 - 00
PCIE2: disabled
PCIE3: disabled
In:    serial
Out:   serial
Err:   serial
DTT:   37 C local / 59 C remote (adt7461@4c)
DTT:   37 C local (lm75@48)
Net:   eTSEC1 connected to Broadcom BCM5482S
eTSEC2 connected to Broadcom BCM5482S
eTSEC1, eTSEC2
POST i2c PASSED
Hit any key to stop autoboot:  0
=>
```

Meet Das U-Boot bootloader

```
[user@boot-dev ~]$ cloc u-boot/
   13518 text files.
   12700 unique files.
    4701 files ignored.
github.com/AlDanial/cloc v 1.76  T=4.02 s (2196.7 files/s, 504571.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                             3958         177722         230606         911861
C/C++ Header                  3540          64684         108111         429854
Assembly                       236           5927          10632          24037
Python                         119           4380           9180          12486
Perl                             6           1660           1346           9850
make                           911           2263           4664           8500
Bourne Shell                    32            427            626           2164
C++                              1            233             58           1588
yacc                             2            169             75           1076
Glade                            1             58              0            603
lex                              2             98             41            539
NAnt script                      1             91              0            367
YAML                             1             13             25            347
Bourne Again Shell               3             75             66            316
Markdown                         1             80              0            283
DOS Batch                        3             20              0            176
CSS                              2             24             10             90
Kermit                           3              4             20             83
Tcl/Tk                           1              5              5             28
sed                              2              1             27             24
INI                              2              3              0             14
XSLT                             1              0              1              9
-------------------------------------------------------------------------------
SUM:                          8828         257937         365493        1404295
```

Meet Das U-Boot bootloader

```
[user@boot-dev ~]$ cloc u-boot/
   13518 text files.
   12700 unique files.
    4701 files ignored.
github.com/AlDanial/cloc v 1.76  T=4.02 s (2196.7 files/s, 504571.1 lines/s)
--------------------------------------------------------------------------------
Language                      files          blank        comment           code
--------------------------------------------------------------------------------
C                              3958         177722         230606         911861
C/C++ Header                   3540          64684         108111         429854
Assembly                        236           5927          10632          24037
Python                          119           4380           9180          12486
Perl                              6           1660           1346           9850
[illegible]
Bourne Shell                     32            427            626           2164
[illegible]                                                    58           1588
[illegible]                       2    [illegible]             75           1076
Glade                             1             58              0            603
Lex                               2    [illegible]             41            539
[illegible]                                                     0            367
YAML                              1             13             25            347
Bourne Again Shell                3             75             66            316
Markdown                          1             80              0            283
DOS Batch                         3             20              0            176
CSS                               2             24             10             90
Kermit                            3              4             20             83
Tcl/Tk                            1              5              5             28
sed                               2              1             27             24
INI                               2              3              0             14
XSLT                              1              0              1              9
--------------------------------------------------------------------------------
SUM:                           8828         257937         365493        1404295
```

"Only" 111 MB of code for a resource-constrained system's bootloader

```
[user@boot-dev ~]$ make -C u-boot distclean
make: Entering directory '/home/user/u-boot'
make: Leaving directory '/home/user/u-boot'
[user@boot-dev ~]$ rm -rf u-boot/.git
[user@boot-dev ~]$ du -sh u-boot/
111M  u-boot/
```

Meet Das U-Boot bootloader

```
[user@boot-dev ~]$ cloc u-boot/
   13518 text files.
   12700 unique files.
    4701 files ignored.
github.com/AlDanial/cloc v 1.76  T=4.02 s (2196.7 files/s, 504571.1 lines/s)
--------------------------------------------------------------------------------
Language                      files          blank        comment           code
--------------------------------------------------------------------------------
C                              3958         177722         230606         911861
C/C++ Header                   3540          64684         108111         429854
Assembly                        236           5927          10632          24037
Python                          119           4380           9180          12486
Perl                              6           1660           1346           9850
make                     [illegible]    [illegible]    [illegible]    [illegible]
Bourne Shell                     32            427            626           2164
C++                      [illegible]    [illegible]             58           1588
yacc                              2    [illegible]             75           1076
Glade                             1             58              0            603
Lex                               2    [illegible]             41            539
NAnt script              [illegible]    [illegible]              0            367
YAML                              1             13             25            347
Bourne Again Shell                3             75             66            316
Markdown                          1             80              0            283
DOS Batch                         3             20              0            176
CSS                               2             24             10             90
Kermit                            3              4             20             83
Tcl/Tk                            1              5              5             28
sed                               2              1             27             24
INI                               2              3              0             14
XSLT                              1              0              1              9
--------------------------------------------------------------------------------
SUM:                           8828         257937         365493        1404295
```

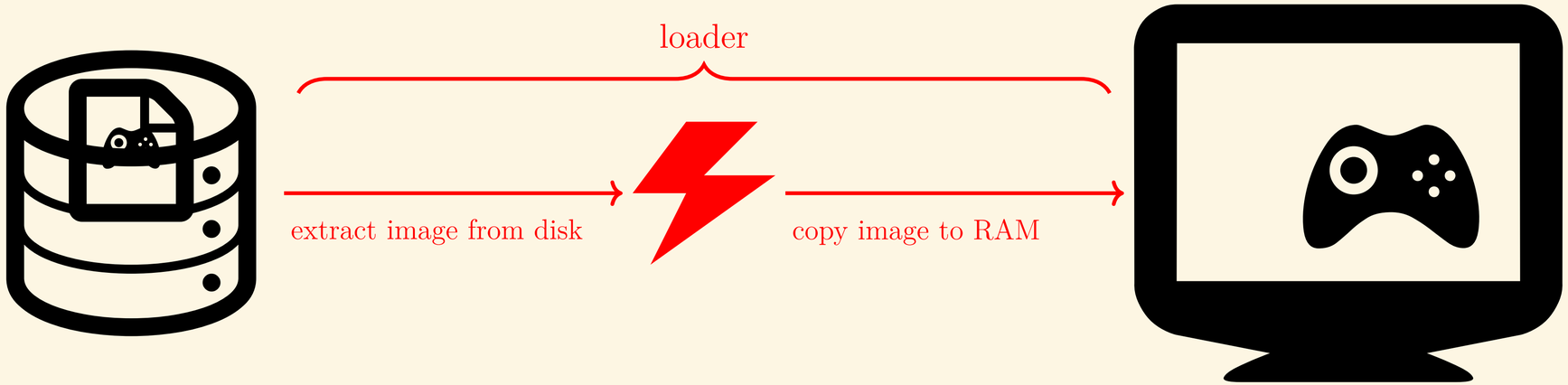
"Only" 111 MB of code for a resource-constrained system's bootloader

```
[user@boot-dev ~]$ make -C u-boot distclean
make: Entering directory '/home/user/u-boot'
make: Leaving directory '/home/user/u-boot'
[user@boot-dev ~]$ rm -rf u-boot/.git
[user@boot-dev ~]$ du -sh u-boot/
111M    u-boot/
```

Quick aside: what is a loader?

The magic that transforms a binary image into an running application

- **Loader:** Software that transduces binary images into memory for execution
- **Binary image:** Static representation/encapsulation of binary (machine) code
  - e.g. An ELF or PE file



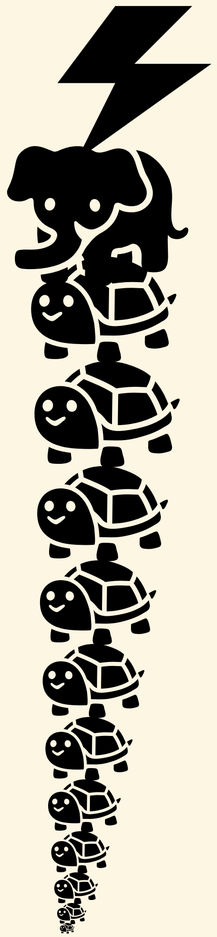
Other useful terminology

- **Address space:** general term referring to addressable memory
- **Memory map:** address space model that semantically labels memory regions

Who loads the loader?

A loader, of course

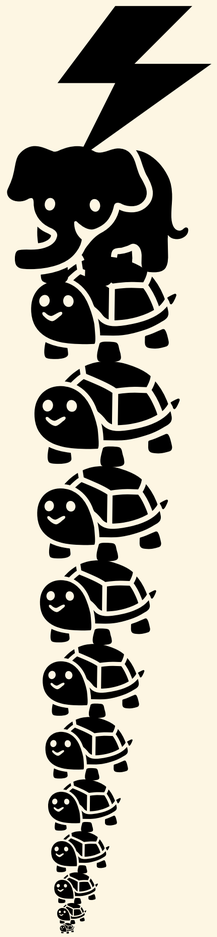
(It's turtles all the way down)



Who loads the loader?

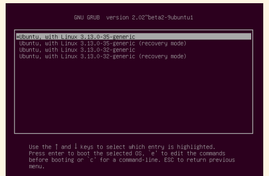
A loader, of course

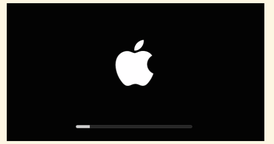
(It's turtles all the way down)



Bootloaders: a subset of loaders

that execute before the OS (or primary application) is executed

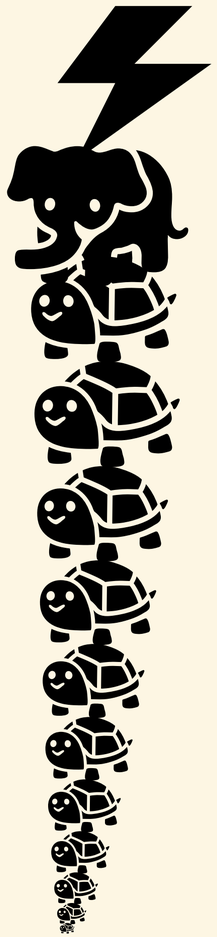




Who loads the loader?

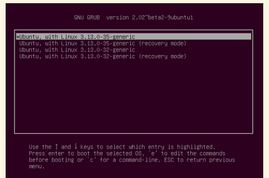
A loader, of course

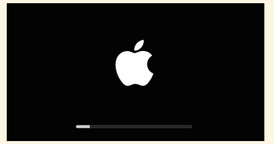
(It's turtles all the way down)



Bootloaders: a subset of loaders

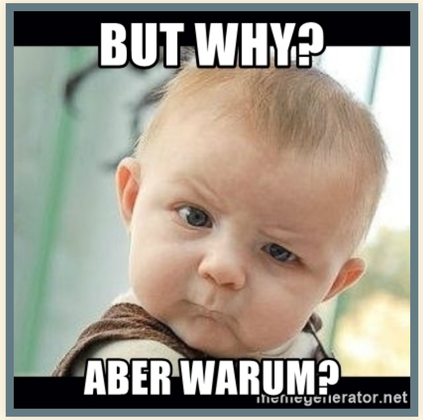
that execute before the OS (or primary application) is executed





</aside>

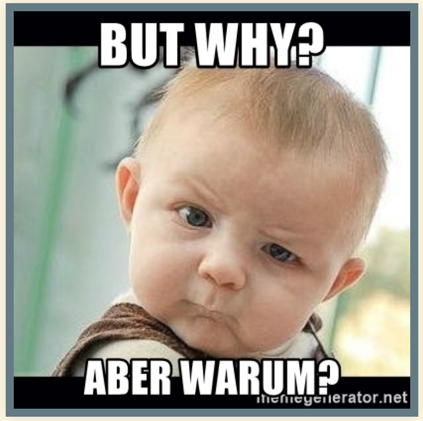
The existential question.



Overall research goals

1. **Identify weaknesses** underlying (boot)loader security
2. **Develop (boot)loader hardening techniques** that:
   - are **realistic**
   - lend themselves to **formal reasoning**
   - can be **retroactively applied** to **existing** loaders
3. Demonstrate technique **feasibility**

The existential question.



Overall research goals

1. **Identify weaknesses** underlying (boot)loader security
2. **Develop (boot)loader hardening techniques** that:
   - are **realistic**
   - lend themselves to **formal reasoning**
   - can be **retroactively applied** to **existing** loaders
3. Demonstrate technique **feasibility**

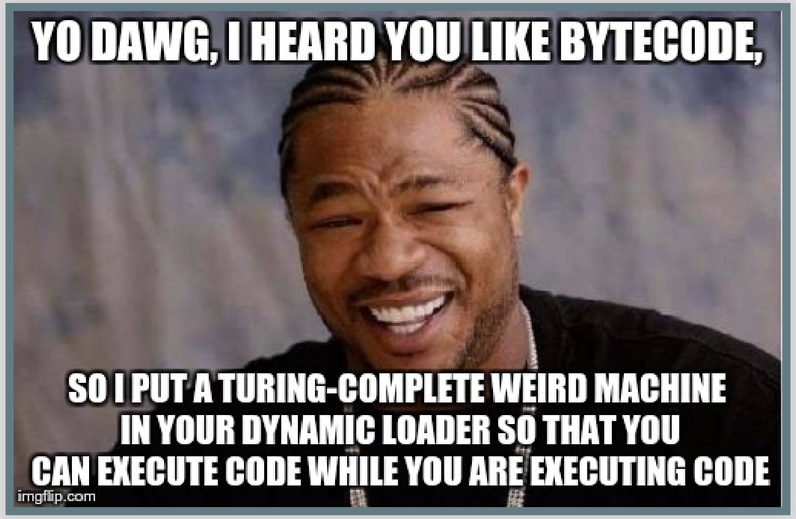
Hence the bootloader reversing

The existential question.

* *Flashback to 2012* *

The Turing-complete ELF-metadata weird machine

a brainfuck to ELF-metadata compiler @ https://github.com/bx/elf-bf-tools



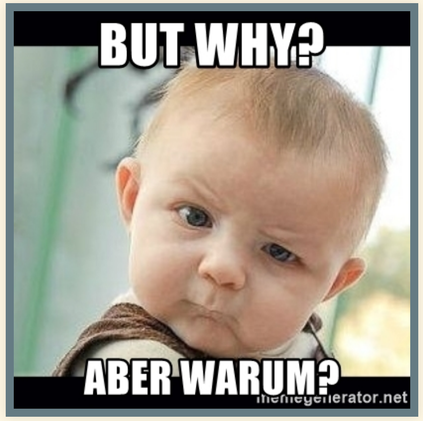
(ELF is *NIX's file format for executables, libraries, etc.)

AS SEEN ON YouTube DEF CON, 29C3, USENIX WOOT, PoC || GTFO

([1], [2], [3], [4])

Hence the bootloader reversing

The existential question.



Overall research goals

1. **Identify weaknesses** underlying (boot)loader security
2. **Develop (boot)loader hardening techniques** that:
   - are **realistic**
   - lend themselves to **formal reasoning**
   - can be **retroactively applied** to **existing** loaders
3. Demonstrate technique **feasibility**

Hence the bootloader reversing

The existential question.

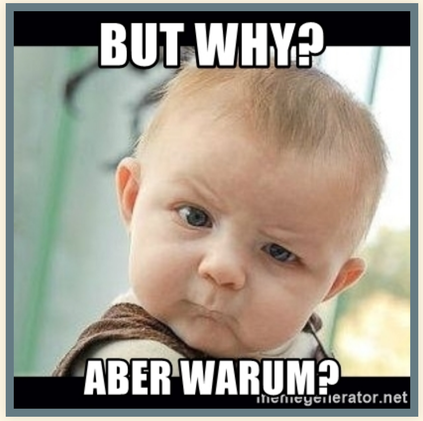
* The ***ultimate*** goal *

Ove goals

1. **Iden**tify we (U)loader security
2. **Dev**elop (b **hniques** that:
   - a**re** reali
   - l end the ng
   - c an be r existing loaders
3. Dem onstrat

Hence the bootloader rever sing

The existential question.



Overall research goals

1. **Identify weaknesses** underlying (boot)loader security
2. **Develop (boot)loader hardening techniques** that:
   - are **realistic**
   - lend themselves to **formal reasoning**
   - can be **retroactively applied** to **existing** loaders
3. Demonstrate technique **feasibility**

Hence the bootloader reversing

This talk for those in a hurry

- ~~Introduce goals and case study~~
- Pontificate about bootloaders in general
- Debugging U-Boot (as according to U-Boot)
- My instrumentation toolsuite
  - An attempt at something better
- Techniques for identifying code <–> data
- A test drive of a simple example

Properties of a bootloader

- Load images and prepare its address space
- Initialize resources/hardware
- Sometimes self-relocate

In general, (boot)loaders:

- Allocate **non-overlapping** addresses
- Manage address **alignment requirements**
- **Prepare memory map** for target
  - **Load** target image into memory
  - **Patch** (link) loaded images
- **Extract** and **enforce** requirements and restrictions imposed by both resources and target

A bit about self-relocation

Bootloaders self-relocate, this may sound crazy
but when RAM is small, you can't be lazy

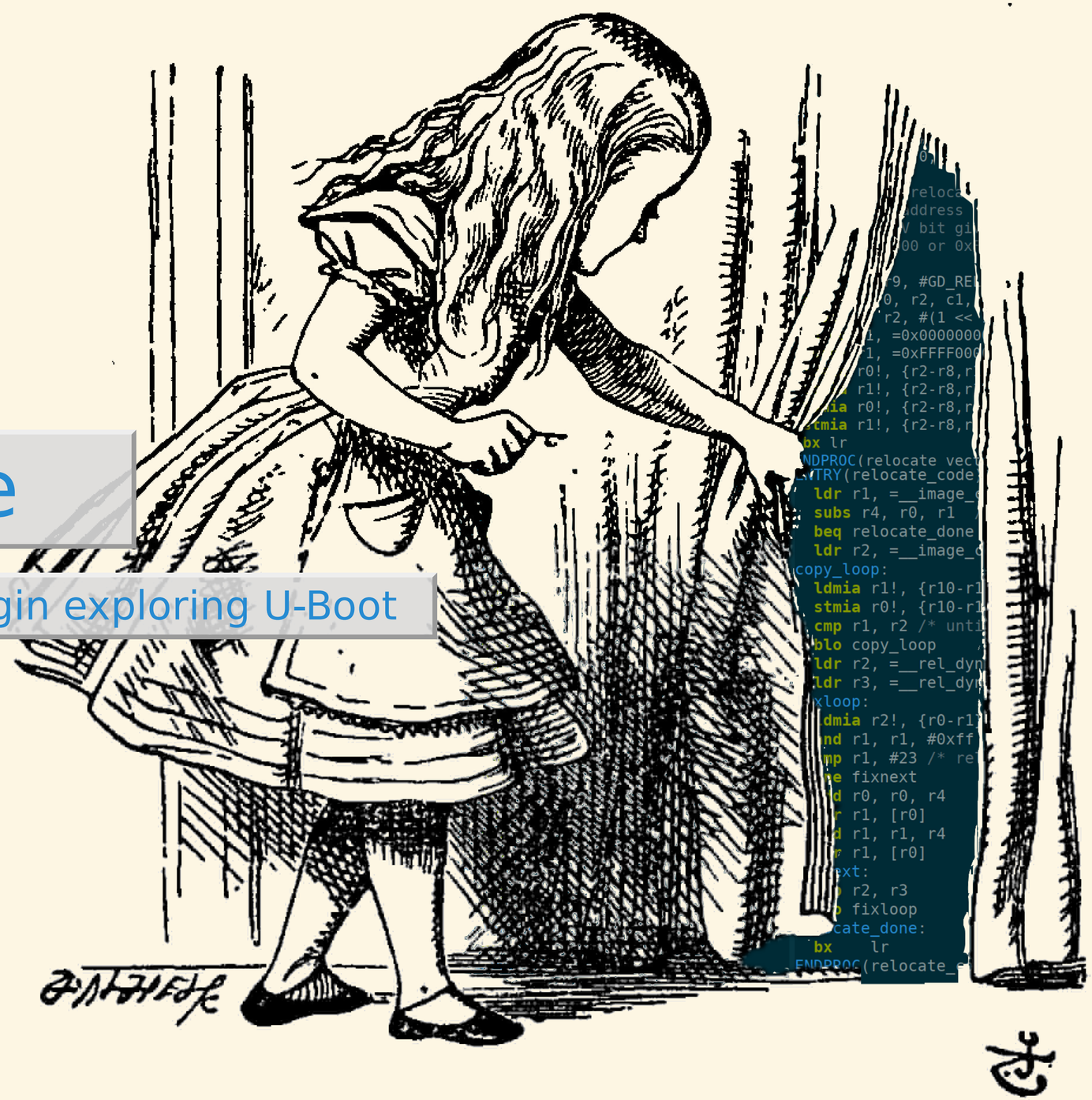
```
ENTRY(relocate_code)
  ldr r1, =__image_copy_start /* r1 <- &__image_copy_start
*/
  subs    r4, r0, r1      /* r4 <- relocation offset */
  beq relocate_done       /* skip relocation */
  ldr r2, =__image_copy_end /* r2 <- &__image_copy_end */
copy_loop:
  ldmia   r1!, {r10-r11}  /* copy from source address [r1]
*/
  stmia   r0!, {r10-r11}  /* copy to   target address [r0]
*/
  cmp r1, r2      /* until source end address [r2]    */
  blo copy_loop
   /* fix .rel.dyn relocations */
  ldr r2, =__rel_dyn_start /* r2 <- SRC &__rel_dyn_start */
  ldr r3, =__rel_dyn_end /* r3 <- SRC &__rel_dyn_end */
fixloop:
  ldmia   r2!, {r0-r1} /* (r0,r1) <- (SRC location,fixup) */
  and r1, r1, #0xff
  cmp r1, #23  /* relative fixup? */
  bne fixnext
  /* relative fix: increase location by offset */
  add r0, r0, r4
  ldr r1, [r0]
  add r1, r1, r4
  str r1, [r0]
fixnext:
  cmp r2, r3
  blo fixloop
relocate_done:
  bx  lr
ENDPROC(relocate_code)
```

Questions to answer when you are exploring a bootloader

- On what **architecture** does it run? (ARM? x86?)
- **Where** can it be **stored**? (First disk sector? Flash memory?)
- **Where** in memory can it be **loaded**?
- **How** is its image **packaged**? (MBR? ELF?)
- **What** runtime **arguments** does it expect and how does it **locate** them?
- From **what devices** can it load its **next stage**? (Disk? Network?)
- **What formats** can it handle as it locates the next stage? (FAT? GPT?)
- How can the **next stage** be **packaged**? (ELF? Compressed? Signed?)
- **What arguments** does it pass to the next stage and how?

Phase one

Let's begin exploring U-Boot



Bootloader instrumentation: tricky bits

1. Relocation
2. Magic/hardcoded numbers
3. Major characteristics of bootloader's address space may change
4. Possible "holes" in its address space
5. Operations or peripherals may silently fail (e.g., nothing may happens if a read-only register has been written)
6. Incorrect documentation-based bugs
7. Operating mode of CPU may change (e.g., privileged, unprivileged)

Bootloader instrumentation: tricky bits

1. Relocation
2. Magic/hardcoded numbers
3. Major characteristics of bootloader's address space may change
4. Possible "holes" in its address space
5. Operations or peripherals may silently fail (e.g., nothing may happens if a read-only register has been written)
6. Incorrect documentation-based bugs
7. Operating mode of CPU may change (e.g., privileged, unprivileged)

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
(gdb) si
0x80100000 in ?? ()
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
(gdb) si
0x80100000 in ?? ()
(gdb) file u-boot
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "u-boot"? (y or n) y
Reading symbols from u-boot...done.
Error in re-setting breakpoint 1: Function "jump_to_image_no_args" not defined.
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
(gdb) si
0x80100000 in ?? ()
(gdb) file u-boot
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "u-boot"? (y or n) y
Reading symbols from u-boot...done.
Error in re-setting breakpoint 1: Function "jump_to_image_no_args" not defined.
(gdb) break relocate_done
Breakpoint 2 at 0x801020b4: file arch/arm/lib/relocate.S, line 134.
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
(gdb) si
0x80100000 in ?? ()
(gdb) file u-boot
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "u-boot"? (y or n) y
Reading symbols from u-boot...done.
Error in re-setting breakpoint 1: Function "jump_to_image_no_args" not defined.
(gdb) break relocate_done
Breakpoint 2 at 0x801020b4: file arch/arm/lib/relocate.S, line 134.
(gdb) continue
Continuing.
Breakpoint 2, relocate_done () at arch/arm/lib/relocate.S:134
134 in arch/arm/lib/relocate.S
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
(gdb) si
0x80100000 in ?? ()
(gdb) file u-boot
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "u-boot"? (y or n) y
Reading symbols from u-boot...done.
Error in re-setting breakpoint 1: Function "jump_to_image_no_args" not defined.
(gdb) break relocate_done
Breakpoint 2 at 0x801020b4: file arch/arm/lib/relocate.S, line 134.
(gdb) continue
Continuing.
Breakpoint 2, relocate_done () at arch/arm/lib/relocate.S:134
134 in arch/arm/lib/relocate.S
(gdb) break board_init_r
Breakpoint 3 at 0x80104e7c: file common/board_r.c, line 957.
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
(gdb) si
0x80100000 in ?? ()
(gdb) file u-boot
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "u-boot"? (y or n) y
Reading symbols from u-boot...done.
Error in re-setting breakpoint 1: Function "jump_to_image_no_args" not defined.
(gdb) break relocate_done
Breakpoint 2 at 0x801020b4: file arch/arm/lib/relocate.S, line 134.
(gdb) continue
Continuing.
Breakpoint 2, relocate_done () at arch/arm/lib/relocate.S:134
134 in arch/arm/lib/relocate.S
(gdb) break board_init_r
Breakpoint 3 at 0x80104e7c: file common/board_r.c, line 957.
(gdb) continue
Continuing.
```

Bootloader instrumentation in action

(with GDB and QEMU)

```
(gdb) target remote | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
Remote debugging using | qemu-system-arm -gdb stdio -M beaglexm -sd sd.img -S
  0x40014000 in ?? ()
(gdb) continue
Continuing.

Breakpoint 1, jump_to_image_no_args (spl_image=0x80000000 )
at arch/arm/cpu/armv7/omap-common/boot-common.c:229
(gdb) si
0x80100000 in ?? ()
(gdb) file u-boot
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "u-boot"? (y or n) y
Reading symbols from u-boot...done.
Error in re-setting breakpoint 1: Function "jump_to_image_no_args" not defined.
(gdb) break relocate_done
Breakpoint 2 at 0x801020b4: file arch/arm/lib/relocate.S, line 134.
(gdb) continue
Continuing.
Breakpoint 2, relocate_done () at arch/arm/lib/relocate.S:134
134 in arch/arm/lib/relocate.S
(gdb) break board_init_r
Breakpoint 3 at 0x80104e7c: file common/board_r.c, line 957.
(gdb) continue
Continuing.
^Comap_gpmc_write: bad SDRAM idle mode 3
omap_i2c_write: Bad register 0x0000cc
```

How to debug a self-relocating bootloader

As according to U-Boot's doc/README.arm-relocation

start debugger

```
[hs@pollux u-boot]$ arm-linux-gdb u-boot
```

How to debug a self-relocating bootloader

As according to U-Boot's doc/README.arm-relocation

start debugger

```
[hs@pollux u-boot]$ arm-linux-gdb u-boot
```

connect to target

```
(gdb) target remote localhost:4444
```

How to debug a self-relocating bootloader

As according to U-Boot's doc/README.arm-relocation

start debugger

```
[hs@pollux u-boot]$ arm-linux-gdb u-boot
```

connect to target

```
(gdb) target remote localhost:4444
```

execute until relocation complete

```
(gdb) break _relocation_done
(gdb) c
```

How to debug a self-relocating bootloader

As according to U-Boot's doc/README.arm-relocation

start debugger

```
[hs@pollux u-boot]$ arm-linux-gdb u-boot
```

connect to target

```
(gdb) target remote localhost:4444
```

execute until relocation complete

```
(gdb) break _relocation_done
(gdb) c
```

discard symbol-file

```
(gdb) symbol-file
Discard symbol table from `/home/hs/celf/u-boot/u-boot'? (y or n) y
No symbol file now.
```

How to debug a self-relocating bootloader

As according to U-Boot's doc/README.arm-relocation

start debugger

```
[hs@pollux u-boot]$ arm-linux-gdb u-boot
```

connect to target

```
(gdb) target remote localhost:4444
```

execute until relocation complete

```
(gdb) break _relocation_done
(gdb) c
```

discard symbol-file

```
(gdb) symbol-file
Discard symbol table from `/home/hs/celf/u-boot/u-boot'? (y or n) y
No symbol file now.
```

load new symbol table at relocated address

```
gdb) add-symbol-file u-boot 0x8ff08000
add symbol table from file "u-boot" at
  .text_addr = 0x8ff08000
(y or n) y
Reading symbols from /home/hs/celf/u-boot/u-boot...done.
```

How to debug a self-relocating bootloader

As according to U-Boot's doc/README.arm-relocation

start debugger

```
[hs@pollux u-boot]$ arm-linux-gdb u-boot
```

connect to target

```
(gdb) target remote loc
```

execu ete

```
(gdb) break _relocation
(gdb) c
```


```
(gdb) symbol-file
Discard symbol table fr                    r n) y
No symbol file now.
```

load new s address

```
gdb) add-symbol-file u-
add symbol table from file "u-boot" at
  .text_addr = 0x3ff08000
(y or n) y
Reading symbols from /home/hs/celf/u-boot/u-boot...done.
```

"Demystifying" magic numbers

As suggested by the U-Boot developers

```
Program received signal SIGSTOP, Stopped (signal).
0x8ff17f18 in serial_getc () at serial_mxc.c:192
192     while (__REG(UART_PHYS + UTS) & UTS_RXEMPTY);
(gdb) add-symbol-file u-boot 0x8ff08000
```

"Demystifying" magic numbers

As suggested by the U-Boot developers

```
Program received signal SIGSTOP, Stopped (signal).
0x8ff17f18 in serial_getc () at serial_mxc.c:192
192     while (__REG(UART_PHYS + UTS) & UTS_RXEMPTY);
(gdb) add-symbol-file u-boot 0x8ff08000
```

"Demystifying" magic numbers

As suggested by the U-Boot developers

```
Program received signal SIGSTOP, Stopped (signal).
0x8ff17f18 in serial_getc () at serial_mxc.c:192
192     while (__REG(UART_PHYS + UTS) & UTS_RXEMPTY);
(gdb) add-symbol-file u-boot 0x8ff08000
```

"Get this address from u-boot bdinfo command or get it from gd->relocaddr in gdb. Interrupt execution by any means and re-load the symbols at the location specified by gd->relocaddr -- this is only valid after board_init_f."

"Demystifying" magic numbers

As suggested by the U-Boot developers

```
Program received signal SIGSTOP, Stopped (signal).
0x8ff17f18 in serial_getc () at serial_mxc.c:192
192     while (__REG(UART_PHYS + UTS) & UTS_RXEMPTY);
(gdb) add-symbol-file u-boot 0x8ff08000
```

"Get this address from u-boot bdinfo command or get it from gd->relocaddr in gdb. Interrupt execution by any means and re-load the symbols at the location specified by gd->relocaddr -- this is only valid after board_init_f."

```
=> bdinfo
arch_number = XXXXXXXXXX
boot_params = XXXXXXXXXX
DRAM bank   = XXXXXXXXXX
-> start    = XXXXXXXXXX
-> size     = XXXXXXXXXX
ethaddr     = XXXXXXXXXX
ip_addr     = XXXXXXXXXX
baudrate    = XXXXXXXXXX
TLB addr    = XXXXXXXXXX
relocaddr   = 0x8ff08000
reloc off   = XXXXXXXXXX
irq_sp     = XXXXXXXXXX
sp start    = XXXXXXXXXX
FB base     = XXXXXXXXXX
```

"Demystifying" magic numbers

As suggested by the U-Boot developers

```
Program received signal SIGSTOP, Stopped (signal).
0x8ff17f18 in serial_getc () at serial_mxc.c:192
192     while (__REG(UART_PHYS + UTS) & UTS_RXEMPTY);
(gdb) add-symbol-file u-boot 0x8ff08000
```

"Get this address from u-boot bdinfo command or get it from gd->relocaddr in gdb. Interrupt execution by any means and re-load the symbols at the location specified by gd->relocaddr -- this is only valid after board_init_f."

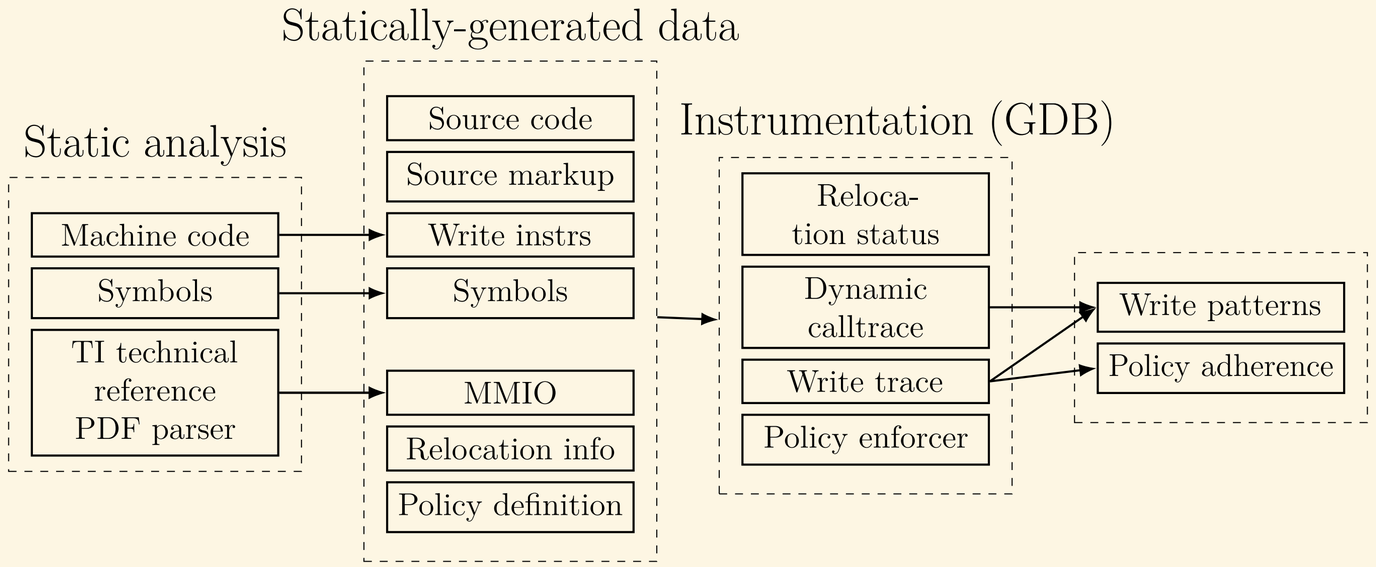
```
=> bdinfo
arch_number = XXXXXXXXXX
boot_params = XXXXXXXXXX
DRAM bank   = XXXXXXXXXX
-> start    = XXXXXXXXXX
-> size     = XXXXXXXXXX
ethaddr     = XXXXXXXXXX
ip_addr     = XXXXXXXXXX
baudrate    = XXXXXXXXXX
TLB addr    = XXXXXXXXXX
relocaddr   = 0x8ff08000
reloc_off   = XXXXXXXXXX
irq_sp     = XXXXXXXXXX
sp start    = XXXXXXXXXX
FB base     = XXXXXXXXXX
```

Can we do better?

The Fiddle toolsuite

Featuring:

- Static and dynamic analysis
- Instrumentation (via GDB)
  - Mediates **all** memory writes
- Language to express (and enforce) memory write polices
- Supports 32-bit ARM & amd64 (some support for arm64 & x86)
- Not just for bootloaders



Source code at https://typedregions.com

Phase two

Relocation reconnaissance

Featuring the U-Boot SPL for BeagleBoard-xM

Tools at https://typedregions.com

Identifying relocation phases

Hypothesis:

The relocation process makes use of memcpy-like operations

(Copying a sequence of adjacent bytes in a tight loop)

I call these **block write operations**

Block write operation

($ip, offset in image, destination, size, call stack)

Identifying relocation phases

Hypothesis:

The relocation process makes use of memcpy-like operations

(Copying a sequence of adjacent bytes in a tight loop)

I call these **block write operations**

Block write operation

($ip, offset in image, destination, size, call stack)

Corollary:

We should be able to identify relocation phases from large block write operations

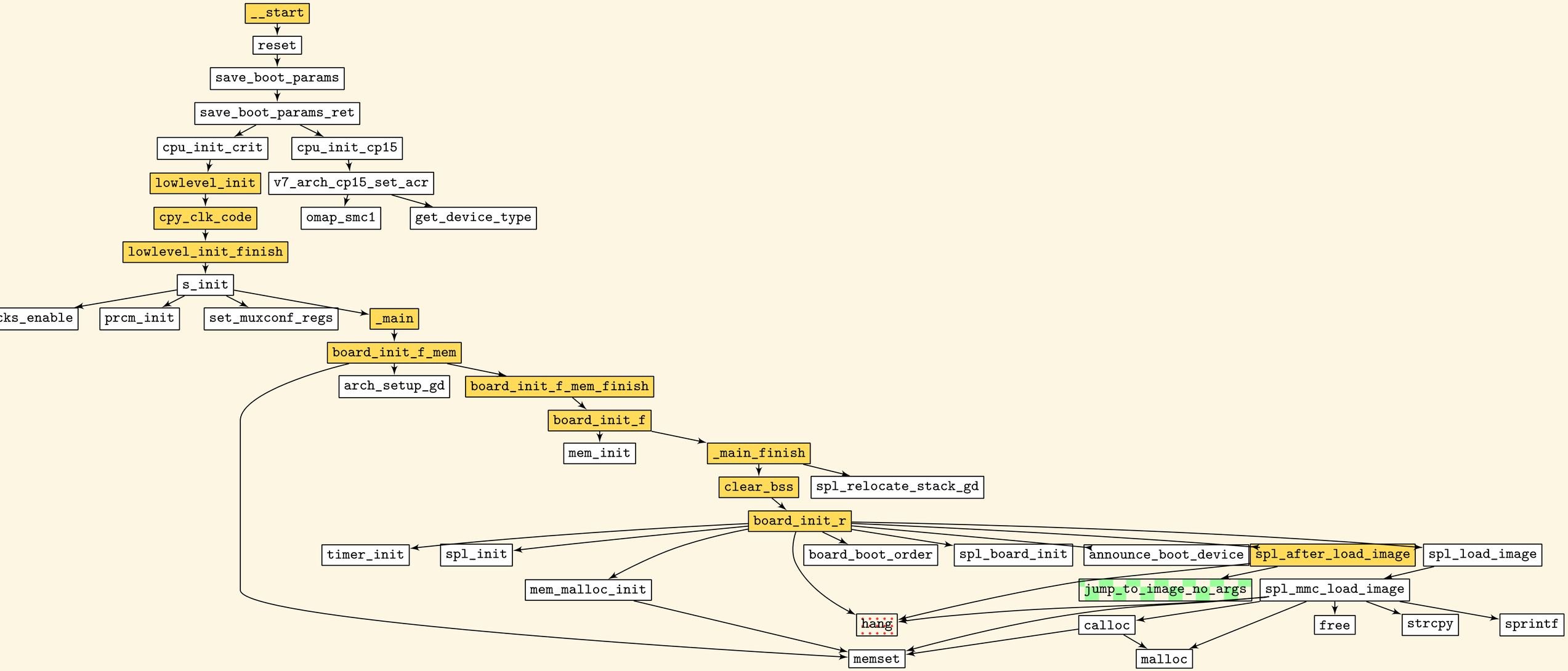
Block writes in U-Boot

```
[262144x] memset @ pc=0x40208a3e wrote 1048576 B to 0x80208000 str
[49233x] clbss_l @ pc=0x402025c4 wrote 196932 B to 0x80000000 strcc   <-- Zero BSS
[128x] mmc_read_data @ pc=0x40206cfa wrote 512 B to 0x4020f2c0
str.w
[128x] mmc_read_data @ pc=0x40206cfa wrote 512 B to 0x4020f2c0     <-- Read target image to memory
str.w
[128x] mmc_read_data @ pc=0x40206cfa wrote 512 B to 0x80104bc0
str.w                                                              <-- Relocate "go_to_speed" function
[128x] mmc_read_data @ pc=0x40206cfa wrote 512 B to 0x80104dc0
str.w
[83x] memset @ pc=0x40208a3e wrote 332 B to 0x80208038 str         <-- Relocate bookkeeping data
[9x] next2 @ pc=0x402009c4 wrote 288 B to 0x4020f840 stmia         <-- Function call/stack push
[54x] memset @ pc=0x40208a3e wrote 216 B to 0x4020fe10 str
[30x] memcpy @ pc=0x40208a72 wrote 120 B to 0x800200c0 str
[16x] memcpy @ pc=0x40208a72 wrote 64 B to 0x4020f118 str
[1x] omap_smc1 @ pc=0x40200970 wrote 40 B to 0x40200234 push
```

# write operations	~400,000
# block writes	10,000

U-Boot's static call graph (for reference)

Generated using IDA Pro (and then simplified by hand)



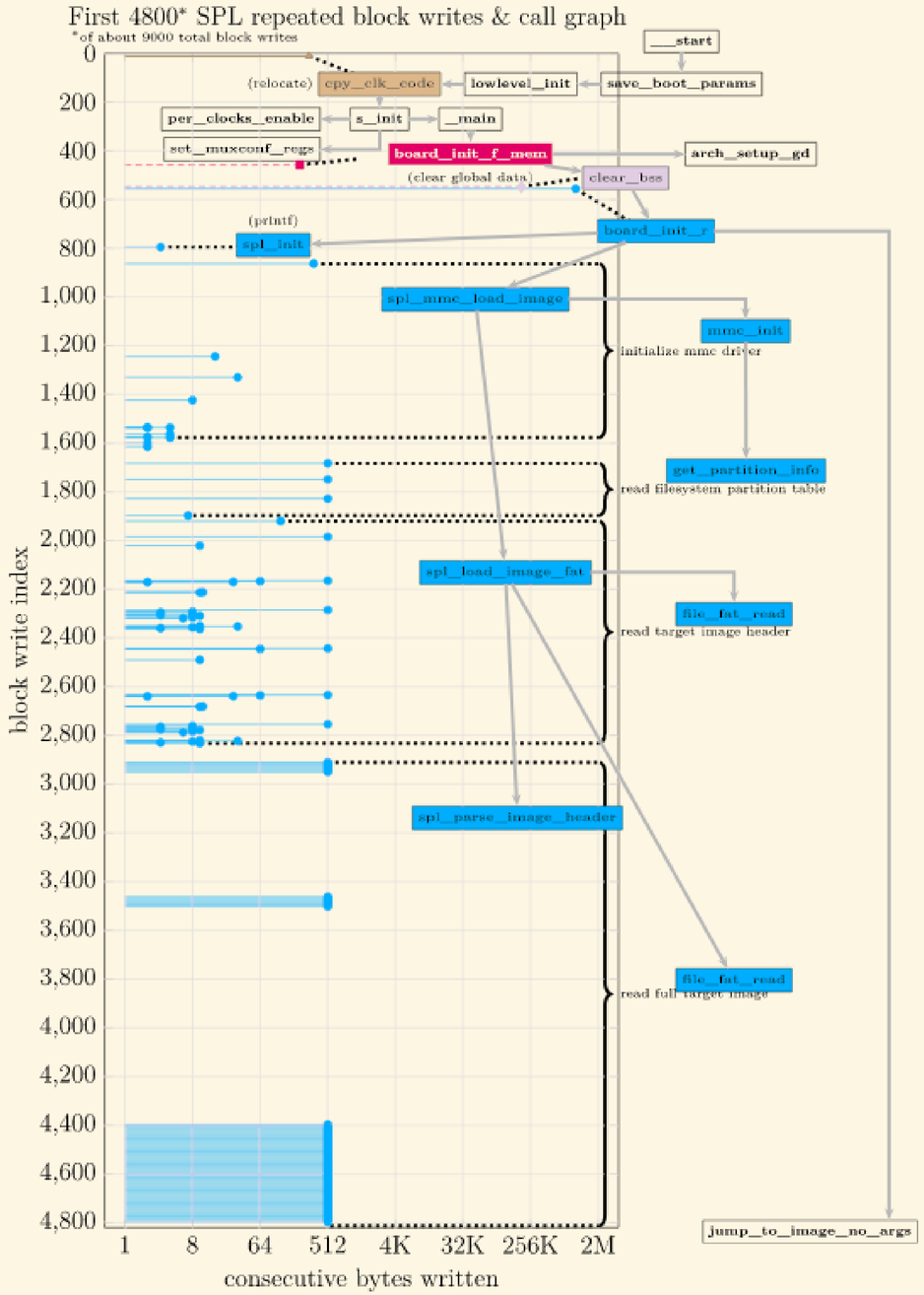
Calltrace of successful U-Boot execution

Example output from U-Boot SPL execution

- > save_boot_params {arch/ahrm/cpu/armv7/omap-common/lowlevel_init.S::22}
    - > cpu_init_cp15 {arch/arm/cpu/armv7/start.S::120}
        - > v7_arch_cp15_set_acr {arch/arm/cpu/armv7/omap3/board.c::388}
            - > get_device_type {arch/arm/cpu/armv7/omap3/sys_info.c::247}
            - < get_device_type
            - > omap_smc1 {arch/arm/cpu/armv7/omap-common/lowlevel_init.S::31}
            - < omap_smc1
        - < v7_arch_cp15_set_acr
        - > v7_arch_cp15_set_acr {arch/arm/cpu/armv7/omap3/board.c::388}
            - > get_device_type {arch/arm/cpu/armv7/omap3/sys_info.c::247}
            - < get_device_type
            - > omap_smc1 {arch/arm/cpu/armv7/omap-common/lowlevel_init.S::31}
            - < omap_smc1
        - < v7_arch_cp15_set_acr
        - > v7_arch_cp15_set_acr {arch/arm/cpu/armv7/omap3/board.c::388}
            - > get_device_type {arch/arm/cpu/armv7/omap3/sys_info.c::247}
            - < get_device_type
            - > omap_smc1 {arch/arm/cpu/armv7/omap-common/lowlevel_init.S::31}
            - < omap_smc1
        - < v7_arch_cp15_set_acr
    - < cpu_init_cp15
    - > cpu_init_crit {arch/arm/cpu/armv7/start.S::273}
        - > lowlevel_init {arch/arm/cpu/armv7/omap3/lowlevel_init.S::187}
            - > cpy_clk_code {arch/arm/cpu/armv7/omap3/lowlevel_init.S::49}
                - > lowlevel_init_finish {arch/arm/cpu/armv7/omap3/lowlevel_init.S::203}
                    - > s_init {arch/arm/cpu/armv7/omap3/board.c::190}
                        - > watchdog_init {arch/arm/cpu/armv7/omap3/board.c::256}
                            - > wait_on_value {arch/arm/cpu/armv7/syslib.c::37}
                            - < wait_on_value
                            - > wait_for_command_complete {arch/arm/cpu/armv7/omap3/board.c::247}
                            - < wait_for_command_complete
                        - < watchdog_init
                        - > try_unlock_memory {arch/arm/cpu/armv7/omap3/board.c::152}
                            - > is_running_in_sdram {arch/arm/cpu/armv7/omap3/sys_info.c::226}
                                - > get_base {arch/arm/cpu/armv7/omap3/sys_info.c::191}
                                - < get_base
                            - < is_running_in_sdram
                            - > get_device_type {arch/arm/cpu/armv7/omap3/sys_info.c::247}
                            - < get_device_type
                            - > secure_unlock_mem {arch/arm/cpu/armv7/omap3/board.c::91}

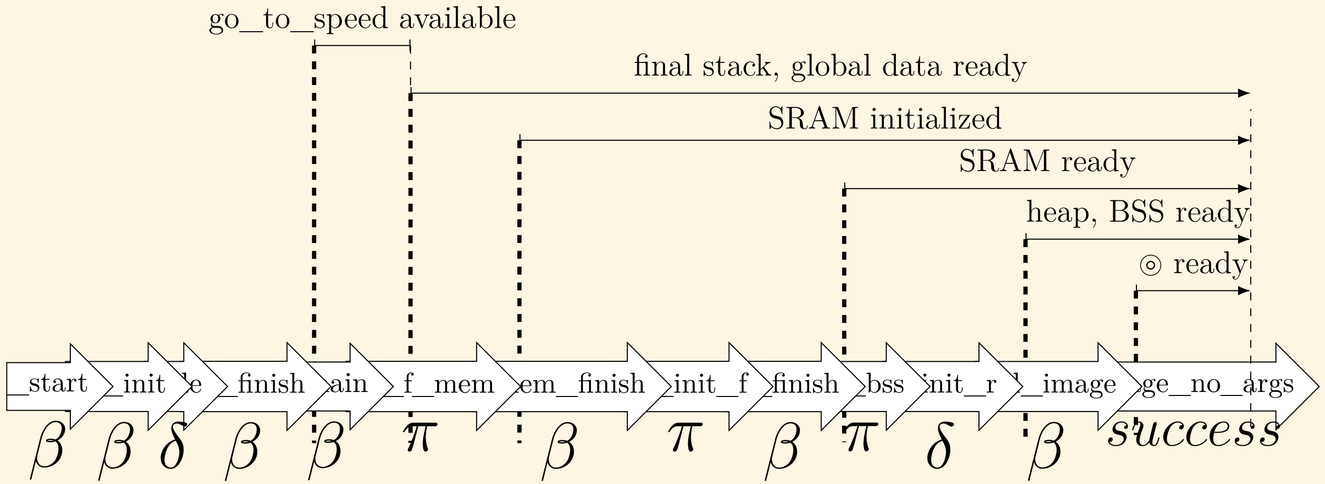
This is a javascript-enabled demo

Calltrace and write data combined

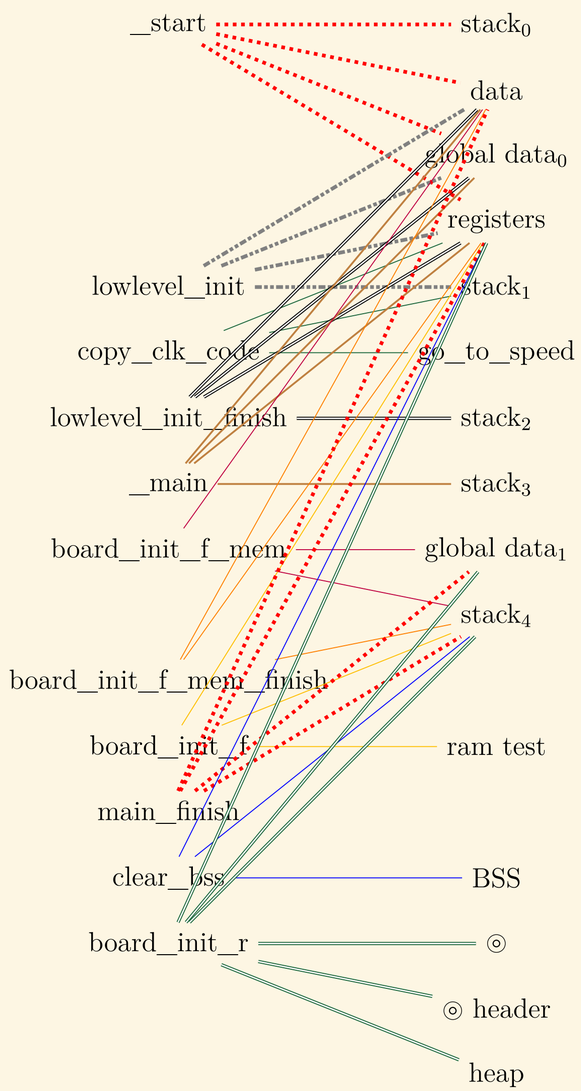


Understanding U-Boot SPL execution

as a sequence of phases



Devising code and data relationships



Other magic numbers

```
switch(beagle_revision()) {
 case REVISION_C4:
  if (identify_xm_ddr() == NUMONYX_MCP) {
    /* 512MB/bank */
   __raw_writel(0x4, SDRC_CS_CFG);

__raw_writel(SDP_SDRC_MDCFG_0_DDR_NUMONYX_XM,
     SDRC_MCFG_0);
```

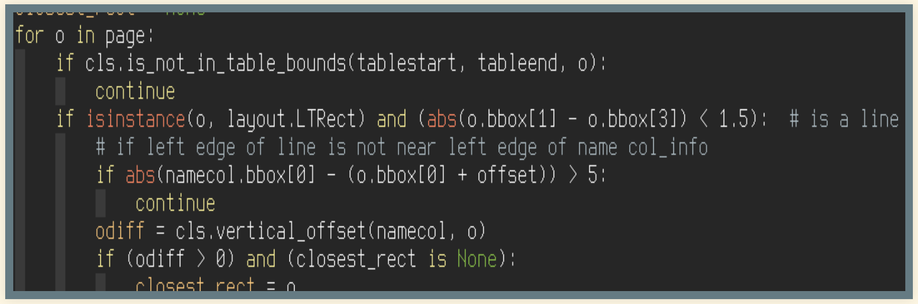
```
argument = 0x0000 << 16;
err = mmc_send_cmd(MMC_CMD55, argument, resp);
if (err == 1) {
 mmc_card_cur->card_type = SD_CARD;
} else {
 mmc_card_cur->card_type = MMC_CARD;
```

Other magic numbers

```
switch(beagle_revision()) {
 case REVISION_C4:
  if (identify_xm_ddr() == NUMONYX_MCP) {
    /* 512MB/bank */
   __raw_writel(0x4, SDRC_CS_CFG);

__raw_writel(SDP_SDRC_MDCFG_0_DDR_NUMONYX_XM,
      SDRC_MCFG_0);
```

```
argument = 0x0000 << 16;
err = mmc_send_cmd(MMC_CMD55, argument, resp);
if (err == 1) {
 mmc_card_cur->card_type = SD_CARD;
} else {
 mmc_card_cur->card_type = MMC_CARD;
```

So I built a PDF scraper



Other magic numbers

```
switch(beagle_revision()) {
 case REVISION_C4:
  if (identify_xm_ddr() == NUMONYX_MCP) {
    /* 512MB/bank */
   __raw_writel(0x4, SDRC_CS_CFG);

__raw_writel(SDP_SDRC_MDCFG_0_DDR_NUMONYX_XM,
      SDRC_MCFG_0);
```

```
argument = 0x0000 << 16;
err = mmc_send_cmd(MMC_CMD55, argument, resp);
if (err == 1) {
 mmc_card_cur->card_type = SD_CARD;
} else {
 mmc_card_cur->card_type = MMC_CARD;
```

So I built a PDF scraper

that transforms

```
for o in page:
    if cls.is_not_in_table_bounds(tablestart, tableend, o):
        continue
    if isinstance(o, layout.LTRect) and (abs(o.bbox[1] - o.bbox[3]) < 1.5):  # is a line
        # if left edge of line is not near left edge of name col_info
        if abs(namecol.bbox[0] - (o.bbox[0] + offset)) > 5:
            continue
        odiff = cls.vertical_offset(namecol, o)
        if (odiff > 0) and (closest_rect is None):
            closest_rect = o
```

Table 10-28. GPMC Registers Mapping Summary

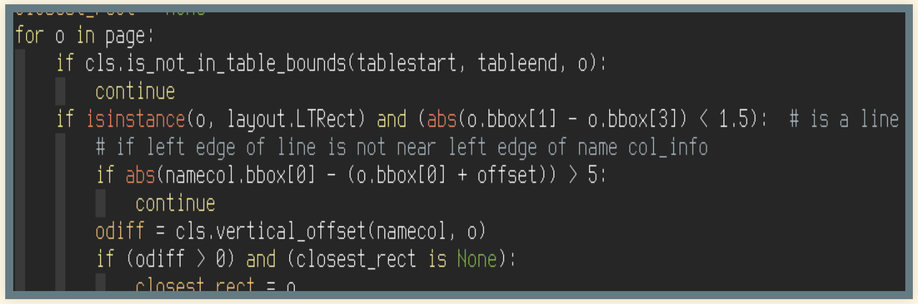
Register Name	Type	Register Width (Bits)	Address Offset	Physical Address
GPMC_REVISION	R	32	0x0000 0000	0x6E00 0000
GPMC_SYSCONFIG	RW	32	0x0000 0010	0x6E00 0010
GPMC_SYSSTATUS	R	32	0x0000 0014	0x6E00 0014
GPMC_IRQSTATUS	RW	32	0x0000 0018	0x6E00 0018

Other magic numbers

```
switch(beagle_revision()) {
 case REVISION_C4:
  if (identify_xm_ddr() == NUMONYX_MCP) {
    /* 512MB/bank */
   __raw_writel(0x4, SDRC_CS_CFG);

__raw_writel(SDP_SDRC_MDCFG_0_DDR_NUMONYX_XM,
      SDRC_MCFG_0);
```

```
argument = 0x0000 << 16;
err = mmc_send_cmd(MMC_CMD55, argument, resp);
if (err == 1) {
 mmc_card_cur->card_type = SD_CARD;
} else {
 mmc_card_cur->card_type = MMC_CARD;
```

So I built a PDF scraper



```
for o in page:
    if cls.is_not_in_table_bounds(tablestart, tableend, o):
        continue
    if isinstance(o, layout.LTRect) and (abs(o.bbox[1] - o.bbox[3]) < 1.5):  # is a line
        # if left edge of line is not near left edge of name col_info
        if abs(namecol.bbox[0] - (o.bbox[0] + offset)) > 5:
            continue
        odiff = cls.vertical_offset(namecol, o)
        if (odiff > 0) and (closest_rect is None):
            closest_rect = o
```

that transforms

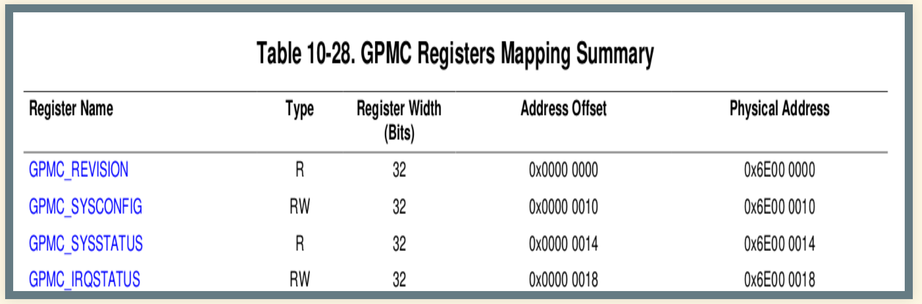


Table 10-28. GPMC Registers Mapping Summary

Register Name	Type	Register Width (Bits)	Address Offset	Physical Address
GPMC_REVISION	R	32	0x0000 0000	0x6E00 0000
GPMC_SYSCONFIG	RW	32	0x0000 0010	0x6E00 0010
GPMC_SYSSTATUS	R	32	0x0000 0014	0x6E00 0014
GPMC_IRQSTATUS	RW	32	0x0000 0018	0x6E00 0018

into

```
CM_FCLKEN_IVA2,RW,W,32,0x00000000,0x48004000,Table 3-93. IVA2_CM Register Summary
CM_CLKEN_PLL_IVA2,RW,W,32,0x00000004,0x48004004,Table 3-93. IVA2_CM Register Summary
CM_IDLEST_IVA2,R,C,32,0x00000020,0x48004020,Table 3-93. IVA2_CM Register Summary
CM_IDLEST_PLL_IVA2,R,C,32,0x00000024,0x48004024,Table 3-93. IVA2_CM Register Summary
CM_AUTOIDLE_PLL_IVA2,RW,W,32,0x00000034,0x48004034,Table 3-93. IVA2_CM Register Summary
```

A simpler example

Hello, world!

```
#include <stdio.h>
#include <string.h>

#define SIZE 512
char memory[SIZE];

void do_nothing() {}

void say_hello() {
  printf("Hello, world\n");
}

void modify_memory() {
  for (int i = 0; i <= SIZE; i++) {
    memory[i] = 'A';
  }
}

int main(int argc, char *argv[]) {
  say_hello();
  modify_memory();
  do_nothing();
  return 0;
}
```

(Built with -static,-no-pie)

Calltrace of Hello, world!

- > _libcstartmain {arm-linux-gnueabihf-glibc/src/glibc-2.27/csu/libc-start.c::137}
  - > dlauxinit {arm-linux-gnueabihf-glibc/src/glibc-2.27/elf/dl-support.c::226}
  - < dlauxinit
  - > _libcinitsecure {arm-linux-gnueabihf-glibc/src/glibc-2.27/elf/enbl-secure.c::32}
  - < _libcinitsecure
  - > _tunablesinit {arm-linux-gnueabihf-glibc/src/glibc-2.27/elf/dl-tunables.c::289}
  - < _tunablesinit
  - > _libcsetuptls {arm-linux-gnueabihf-glibc/src/glibc-2.27/csu/../csu/libc-tls.c::107}
    - > _udivsi3
      - > _aeabiuidiv
    - < _udivsi3
      - < _aeabiuidiv
      - > sbrk {arm-linux-gnueabihf-glibc/src/glibc-2.27/misc/sbrk.c::32}
        - > _sbrk {arm-linux-gnueabihf-glibc/src/glibc-2.27/misc/sbrk.c::32}
          - > _brk {arm-linux-gnueabihf-glibc/src/glibc-2.27/misc/../sysdeps/unix/sysv/linux/arm/brk.c::28}
            - > brk {arm-linux-gnueabihf-glibc/src/glibc-2.27/misc/../sysdeps/unix/sysv/linux/arm/brk.c::28}
          - < _brk
            - < brk
            - > _brk {arm-linux-gnueabihf-glibc/src/glibc-2.27/misc/../sysdeps/unix/sysv/linux/arm/brk.c::28}
              - > brk {arm-linux-gnueabihf-glibc/src/glibc-2.27/misc/../sysdeps/unix/sysv/linux/arm/brk.c::28}
            - < _brk
              - < brk
      - < sbrk
        - < _sbrk
        - > memcpy {arm-linux-gnueabihf-glibc/src/glibc-2.27/string/../sysdeps/arm/memcpy.S::63}
        - < memcpy
        - > _udivsi3
          - > _aeabiuidiv
        - < _udivsi3
          - < _aeabiuidiv
  - < _libcsetuptls
  - > dldiscoverosversion {arm-linux-gnueabihf-glibc/src/glibc-2.27/elf/../sysdeps/unix/sysv/linux/dl-sysdep.c::45}
    - > uname {arm-linux-gnueabihf-glibc/src/glibc-2.27/posix/../sysdeps/unix/syscall-template.S::78}
      - > _uname {arm-linux-gnueabihf-glibc/src/glibc-2.27/posix/../sysdeps/unix/syscall-template.S::78}
    - < uname
      - < _uname
  - < dldiscoverosversion
  - > _libcinitfirst {arm-linux-gnueabihf-glibc/src/glibc-2.27/csu/../csu/init-first.c::44}
    - > dlnondynamicinit {arm-linux-gnueabihf-glibc/src/glibc-2.27/elf/dl-support.c::309}
      - > dlgetorigin {arm-linux-gnueabihf-glibc/src/glibc-2.27/elf/../sysdeps/unix/sysv/linux/dl-origin.c::36}

This is a javascript-enabled demo

Checking for unexpected writes

1. Configure instrumentation suite to work with sample
2. Run sample through instrumentation suite's **static analysis**
3. Construct policy to target "substages" and memory regions of interest
4. Import policy
5. Execute **dynamic analysis**
6. Use **post-analysis** to highlight policy violations

Checking for unexpected writes

1. Configure instrumentation suite to work with sample
2. Run sample through instrumentation suite's **static analysis**
3. Construct policy to target "substages" and memory regions of interest
4. Import policy
5. Execute **dynamic analysis**
6. Use **post-analysis** to highlight policy violations

Region definitions

```
regions:
 ALL:
  type: "global"
  addresses: [0, 0xFFFFFFFF]
  subregions:
   buffer:
     type: "global"
     addresses: [0x8bb98, 0x8bd98]
   stack:
     type: "stack"
     addresses: [0xfffe0000, 0xffff0000]
   ro:
     type: "global"
     addresses: [[0x0, 0x8bb898],
       [0x8bd98,0xfffe0000],
       [0xffff0000,0xffffffff]]

stagename: "_single"
```

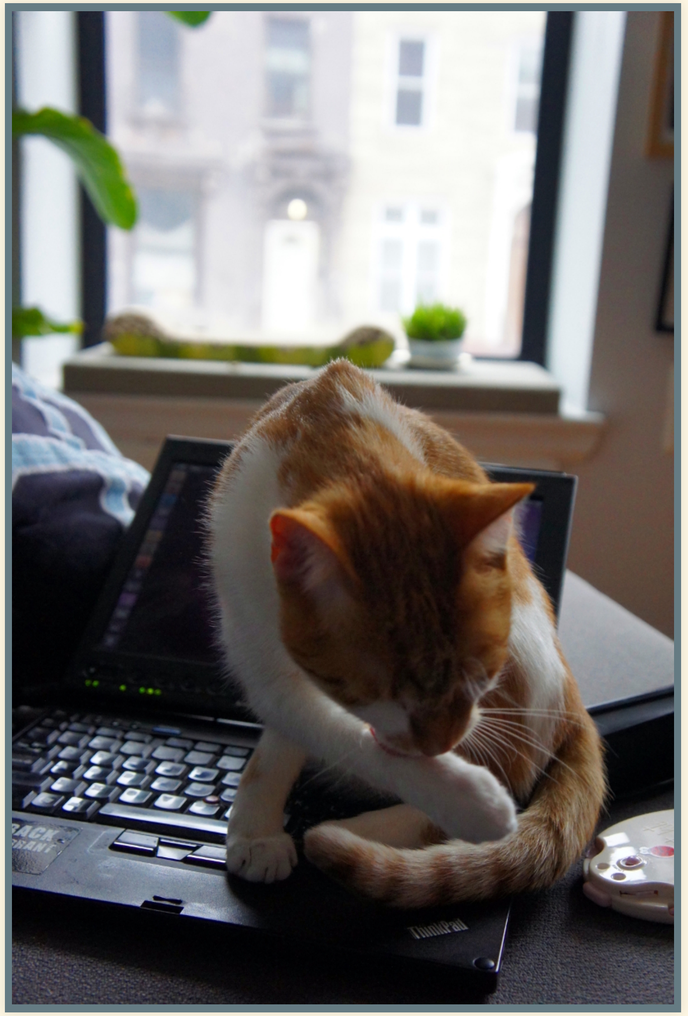
Substage/policy definitions

```
_start:
  substage_type: "bookkeeping"
  new_regions: ["ALL.buffer", "ALL.stack",
"ALL.ro"]
  reclassified_regions:
    ALL.ro: "global"
    ALL.stack: "stack"
    ALL.buffer: "global"

main:
  substage_type: "bookkeeping"

modify_memory:
  substage_type: "bookkeeping"
  reclassified_regions:
    ALL.ro: "readonly"

do_nothing:
  substage_type: "bookkeeping"
  reclassified_regions:
    ALL.ro: "global"
```

Phase three

An attempt at a demo

Thank you

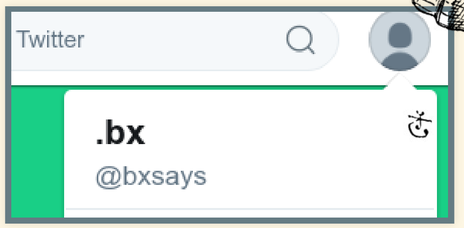


More details and tools at: https://typedregions.com

Blog post on bootloaders at: https://bit.ly/2zbentY

Many thanks **Sergey Bratus**, my PhD advisor

Thank you



bx@narfindustries.com

More details and tools at: https://typedregions.com

Blog post on bootloaders at: https://bit.ly/2zbentY

Many thanks **Sergey Bratus**, my PhD advisor

References

See also typedregions.com

[1]. R. .. Shapiro, “The care and feeding of weird machines found in executable metadata,” in *Chaos communication congress*, 2012.

[2]. R. Shapiro, S. Bratus, and S. W. Smith, “Weird machines in ELF: A spotlight on the underappreciated metadata,” in *Workshop on offensive technologies*, Washington, D.C., 2013.

[3]. R. .. Shapiro, “Returning from ELF to libc,” in *POC or GTFO*, vol. 0x0, 2013.

[4]. R. .. Shapiro, “Calling putchar() from an ELF weird machine,” in *POC or GTFO*, vol. 0x02, 2013.

[5]. J. Bangert, S. Bratus, R. Shapiro, and S. W. Smith, “The page-fault weird machine: Lessons in instruction-less computation,” in *Workshop on offensive technologies*, Washington, D.C., 2013.

[6]. J. Bangert, S. Bratus, R. Shapiro, M. E. Locasto, J. Reeves, S. W. Smith, and A. Shubina, “ELFbac: Using the loader format for intent-level semantics and fine-grained protection,” Dartmouth College, Computer Science, Hanover, NH, TR2013-727, May 2013.

[7]. S. Bratus, M. E. Locasto, and M. L. Patterson, “Exploit programming: From buffer overflows to "weird machines” and theory of computation,” in *USENIX; login*, 2011, pp. 13–21.

[8]. S. Bratus and J. Bangert, “ELFs are dorky, elves are cool,” in *POC or GTFO*, vol. 0x0, 2013.

[9]. J. Oakley and S. Bratus, “Exploiting the hard-working DWARF: Trojan and exploit techniques with no native executable code,” in *Workshop on offensive technologies*, 2011, p. 11.