r2@snow

pancake@nopcode.org

Who am I?

- Sergi Alvarez i Capilla (also known as pancake)
- Born in Barcelona ‘83
- Mobile Security Analyst at NowSecure
- Author of r2 and several other open source projects
- Sleepless hacker and developer
- r2con organizer
  - 2nd week of September in Barcelona
  - Meeting point for all r2 devs and users
  - Only technical talks, all talks available in YouTube

What's r2?

—

- Free / Libre Reverse Engineering framework
  - Libs, apis, cmds, scripts, pipeable programs, ..
- Unix-like design, aims to be orthogonal
- Focus on API, cmdline tools and bindings
- Some GUIs already available, no one complete
- Package manager to install plugins and dependencies
- Always refactoring and releasing every 6 weeks
- Enforces test suite, code reviews and fuzz
- 11 year old project, Release every 6 weeks

What's this talk about?

- Show some of the **features** of this tool
- Practical use case **examples** of r2
- Analyzing raw **ARM** firmware images
- Focus on the **HYTERA** DMR clones
  - I don't have the device or interest in the project
  - Didn't checked the project since last week
  - Just as an example project for showing r2
  - Thanks Travis for the talks on YouTube

Target Device

Cheap Chinese clones of HYTERA (60€-100€) DMR/NFM

- Retevis RT3 / RT8 (GPS)
- TYT MD380 / MD 390 (GPS)
- Zastone D900
- Chielda D200
- VITAI VDG-385
- Juentai JD-780
- SAMCOM DP-20
- HYDX D50
- Radioddity GD-55

Not HJKL friendly

Really confusing

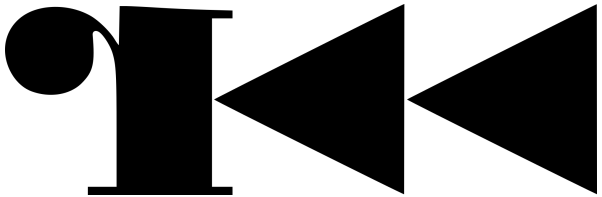
Target Architecture

- ARM **STM32F405** Microcontroller
  - **Cortex-M4**, **Fpv4-sp-d16** FPU
  - Instructions to sha, cry, md5, aes, 3des, print
- Supports ARM, Thumb and **Thumb2**
- **1MB** Flash memory
- **192KB** of RAM
- **HR C5000** Radio Baseband peripheral
- IO / Ports
  - **1x LCD, 3 SPI**, 4 UART, **3 I2C**, 2 CAN, SDIO
  - 2x **USB** OTG
  - 10/100 **Ethernet**

Radare2

Free/Libre multi - {arch, platform, paradigm, language, user} Unix-like Reverse Engineering Framework.

- Compact mnemonic commands shell
- Scriptable via bindings, pipes, batch
- CLI, Visual, WebUI and some native GUIs
- Written in C, portable and fast
- Huge and passionate community
- Implementation in separated libraries
- Extensions implemented as plugins
- Very customizable and versatile

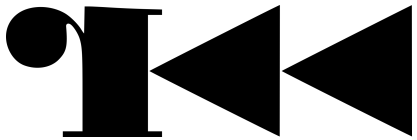


Libraries

- **RBin/RFs** parses headers (executable file formats, partt)
- **RIO** abstracts open/read/write/close (everything is a file)
- **RAsm/RAnal** implements archs (asm, disasm, analyze, emu)
- **RDebug/RReg/RBp** debuggers (native, gdb, windbg, …)
- **RSearch/RMagic** match patterns, with mask, aproximation, ..
- **RUtil** base library on top of libc

Targets

- Linux, Windows, Mac, iOS, Android, QNX
- x86, mips, arm, arm64, sparc, powerpc, avr, 6501, ..
- ELF, mach0, PE, DEX, ART, Wasm, Swf, COFF, Plan9, …

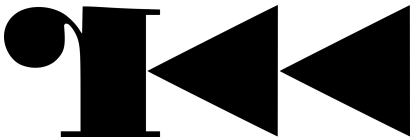


Common use cases

- **Solve** crackmes
- Cooking **ROP** payloads
- Exploiting router **vulnerabilities**
- Analyze Windows, Linux, Android, iOS **malware**
- Reverse engineer **unknown** file formats
- **Carve** disk/memory for needles
- **Recover** deleted files
- Bypass **security** protections
- Find **vulnerabilities** in software
- Debug **crashes**

Scripting

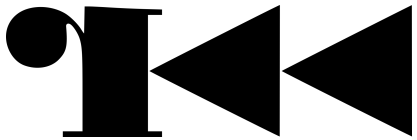
- **Mnemonic** and **compact** command shell
- Supports #! hashbang with **r2pipe** and **rlang**
- Emscripten **r2core.js**
- **Native** bindings with valabind and swig
  - **Python**, **NodeJS**, Perl, Java, C#, Ruby, ...
- r2pipe / api
  - Faster, support **sync**/**async**, in/out/ versatile
  - Multiple **transports** (pipe, tcp, http, dlsym, …)
  - r2pipe-api is **wip** high level api on top of r2pipe



Packages and plugins!

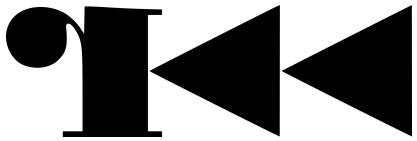
Available via r2pm

- **Frida** tracing and code injection (r2frida)
- **Keystone** assembler
- **Unicorn** emulator
- **Kaitai** struct visualizer
- **Yara** plugin
- **RetDec/SnowMan** decompilers
- Many **Web** User Interfaces
- ...



Documentation

- Fully documented in **C**
- An **IRC** channel bridged with **Telegram**
  - About 700 online users
- An official **book** for r1 and r2
- Code **snippets** and **examples**
- Huge **testsuite** and growing
- Several **talks** and **presentations**
  - Official website (pdf)
  - YouTube
  - Blog posts



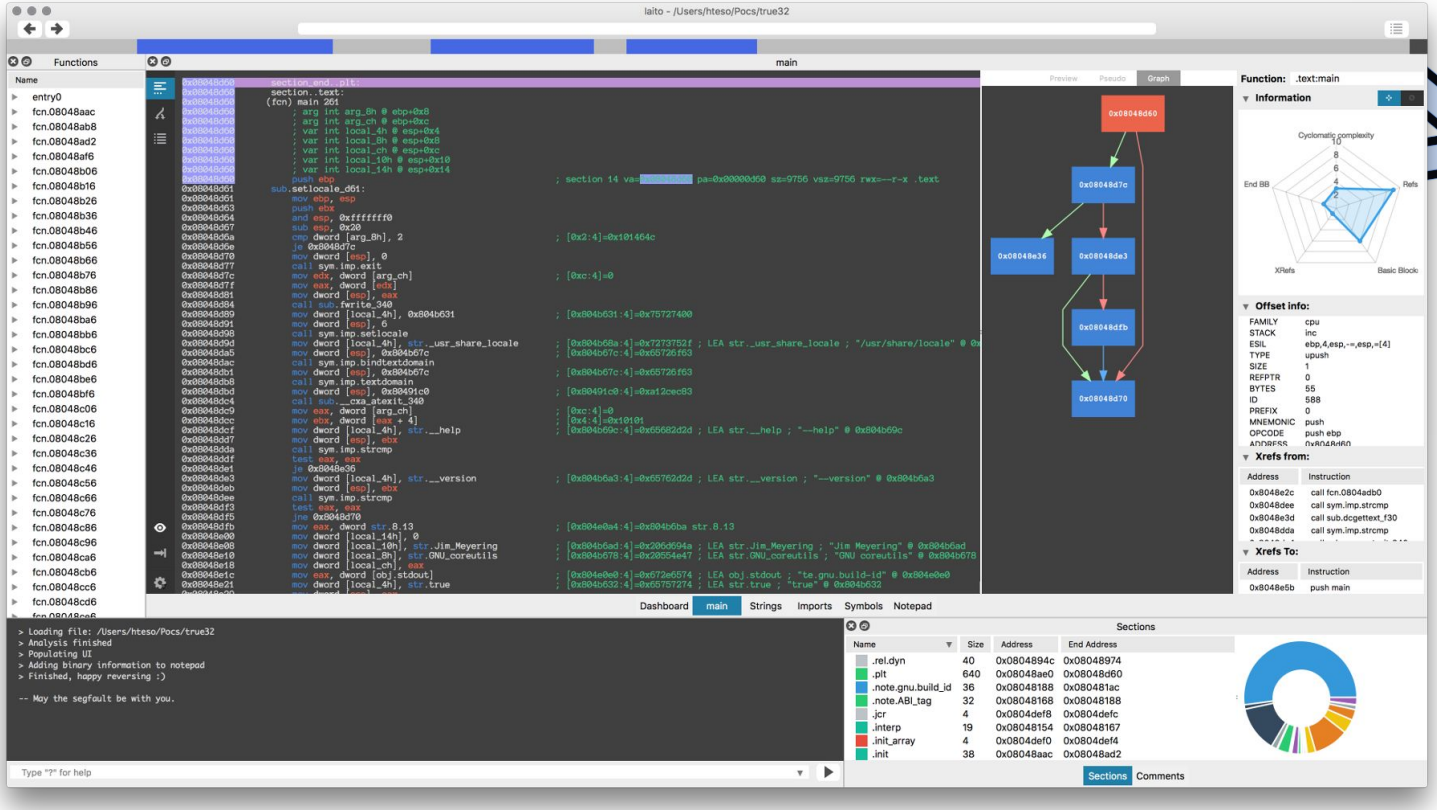
And now also in QT!

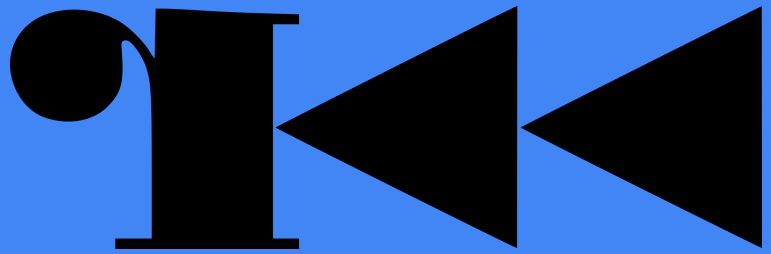
Iaito is the name of the new GUI by Hugo Teso.

- Author of Bokken
- Released two days ago
- Multiplatform (Win/Mac/Lin)
- Free/Libre/OpenSource

https://github.com/hteso/iaito

Iaito (https://github.com/hteso/iaito)





Let’s go practice!

Decrypting the Firmware

- **Initial overview with r2**
  - **Hexdump, visual mode, disassembly, ...**

```
> wtf sys.img $s-0x200 @ 0x100
```

- **RBin** plugin
  - Identify file format
  - Load sections, requires IO to decrypt
  - Parsing header and dumping
- **Xoring**
  - rahash2 -E xor -S - < a > b
  - Extract key from pattern

Overview

- Zoom view with pz and pxA (available in VPP)
  - Instruction type map
- Per-block Hash (rahash2)
  - rahash2 -a entropy -Bb 512 jailbreak.bin
- Code/Data Block Statistics with p=
  - Number of printable chars
  - Strings per block
  - Invalid instructions per block
  - Call/Jump
  - Entropy
  - ...

Extracting Strings

- Strings is not able to catch any kanji.
- Supports Ascii, WideChar, UTF8 strings
  - `rabin2 -zz newfw.bin | grep =wide`
- We can overwrite with code in all that chinese fonts and text regions

```
vaddr=0x000ed590 paddr=0x000ed590 ordinal=13783 sz=12 len=5 section=unknown type=wide string=Clock
vaddr=0x000ed59c paddr=0x000ed59c ordinal=13784 sz=12 len=5 section=unknown type=wide string=Clock
vaddr=0x000ed5a8 paddr=0x000ed5a8 ordinal=13785 sz=10 len=4 section=unknown type=wide string=Date
vaddr=0x000ed5b4 paddr=0x000ed5b4 ordinal=13786 sz=10 len=4 section=unknown type=wide string=Date
vaddr=0x000ed5c0 paddr=0x000ed5c0 ordinal=13787 sz=10 len=4 section=unknown type=wide string=Menu
vaddr=0x000ed5cc paddr=0x000ed5cc ordinal=13788 sz=10 len=4 section=unknown type=wide string=Name
vaddr=0x000ed5d8 paddr=0x000ed5d8 ordinal=13789 sz=12 len=5 section=unknown type=wide string=Clear
vaddr=0x000ed5e4 paddr=0x000ed5e4 ordinal=13790 sz=10 len=4 section=unknown type=wide string=Mode
vaddr=0x000ed5f0 paddr=0x000ed5f0 ordinal=13791 sz=12 len=5 section=unknown type=wide string=T CTC
vaddr=0x000ed5fc paddr=0x000ed5fc ordinal=13792 sz=12 len=5 section=unknown type=wide string=R CTC
vaddr=0x000ed608 paddr=0x000ed608 ordinal=13793 sz=12 len=5 section=unknown type=wide string=T DCS
vaddr=0x000ed614 paddr=0x000ed614 ordinal=13794 sz=12 len=5 section=unknown type=wide string=R DCS
vaddr=0x000ed62c paddr=0x000ed62c ordinal=13795 sz=12 len=5 section=unknown type=wide string=USB模式
vaddr=0x000ed7e8 paddr=0x000ed7e8 ordinal=13808 sz=10 len=4 section=unknown type=wide string=CP版本
vaddr=0x000edac4 paddr=0x000edac4 ordinal=13822 sz=12 len=5 section=unknown type=wide string=CTC编码
vaddr=0x000edad0 paddr=0x000edad0 ordinal=13823 sz=12 len=5 section=unknown type=wide string=CTC解码
vaddr=0x000edadc paddr=0x000edadc ordinal=13824 sz=12 len=5 section=unknown type=wide string=DCS编码
vaddr=0x000edae8 paddr=0x000edae8 ordinal=13825 sz=12 len=5 section=unknown type=wide string=DCS解码
```

Finding the Base Address

- Required to properly read the disassembly
- That's where the code and data is mapped
- Fixed addresses, no heap or ASLR

- Firmware header contains it
- Pointers in code (code/data references)
- Pointers in data (dwords)
- In the strings of the bootloader

```
$ rabin2 -qzz bootloader.bin | grep Intern
```

Memory Layout

- 0xe0000000 - Cortex peripherals
- 0x0800c000 - flash app
- 0x08000000 - boot loader (mirrored at 0)
- 0x40000000 - IO serial/SPI/i2c/USB ports
- 0x20000000 - SRAM
- 0x10000000 - Fast TCRAM (non-executable)
- 0x00000000 - Flash (null deref exploits for fw dumping)

Loading the image

Most people will do:

```
$ r2 -a arm -b 16 -m 0x0800c000
```

But we are missing a lot of things in here..

Loading the image

- Loading all images into memory
- Setting the cortex CPU
- Setup two RAM regions
- Emulating memory mapped devices
- ESIL emulation of Thumb2 code
- Force filter search hits aligned to 4 bytes
- Configure sections with iS, S, S=, o.

```
e asm.section.sub = true
```

Check load.r2 script

Disassembling

- Thumb2 and Cortex caveats
  - Useful to find points of interest (STI/CLI)
- Symbol information from RBin
  - Not available in raw firmware images
- Code Analysis information
  - Call cross-references
- Data Analysis
  - Pointer dereferences
- Analysis hints
  - `ahb 16`
  - `afb 16`
  - `e asm.bits=16`
  - `e asm.cpu=cortex`

```
[0x08003048]> pd 5
┌ (fcn) fcn.08003048 2
│   fcn.08003048 ();
│              ; CALL XREF from 0x08003ca2 (fcn.08003c70)
│              ; CALL XREF from 0x08003cf4 (fcn.08003c70)
│              ; CALL XREF from 0x08003d68 (fcn.08003d58)
│              ; CALL XREF from 0x08001ab2 (fcn.08001aa0)
│              ; CALL XREF from 0x08003238 (fcn.08003234)
└           0x08003048  ~   eff31080       Invalid
│           0x0800304c      72b6          Cpsid i
└           0x0800304e      7047          Bx    lr
┌ (fcn) fcn.08003050 2
│   fcn.08003050 ();
│              ; XREFS: CALL 0x08003cec  CALL 0x08003d44  CALL
│              ; XREFS: CALL 0x08003da2  CALL 0x08003d84  CALL
│              ; XREFS: CALL 0x08001ace  CALL 0x0800328a
└           0x08003050  ~   80f31088       Invalid
└           0x08003054      7047          Bx    lr
[0x08003048]> e asm.cpu = cortex
[0x08003048]> pd 5
┌ (fcn) fcn.08003048 2
│   fcn.08003048 ();
│              ; CALL XREF from 0x08003ca2 (fcn.08003c70)
│              ; CALL XREF from 0x08003cf4 (fcn.08003c70)
│              ; CALL XREF from 0x08003d68 (fcn.08003d58)
│              ; CALL XREF from 0x08001ab2 (fcn.08001aa0)
│              ; CALL XREF from 0x08003238 (fcn.08003234)
│           0x08003048  ~   eff31080       Mrs   r0,   primask
│           0x0800304c      72b6          Cpsid i
└           0x0800304e      7047          Bx    lr
┌ (fcn) fcn.08003050 2
│   fcn.08003050 ();
│              ; XREFS: CALL 0x08003cec  CALL 0x08003d44  CALL
│              ; XREFS: CALL 0x08003da2  CALL 0x08003d84  CALL
│              ; XREFS: CALL 0x08001ace  CALL 0x0800328a
│           0x08003050  ~   80f31088       Msr   primask, r0
└           0x08003054      7047          Bx    lr
[0x08003048]>
```

Jailbreaking

- Modifying bytes to unlock jtag
  `$ radiff2 bootloader.bin jailbreak.bin`
- Can be done by modifying a byte..
  `> r2 -nw -c 'wx aa @ 0x080044a8' boot.bin`

```
[0x080044a0 35% 150 (0x8:-1=1)]> pd $r @ flag.jailbreak
|      0x080044a0      fdf7a0fd       Bl    sym.rdp_is_not_locked
|      0x080044a4      0028           Cmp   r0,   0
|   ,=< 0x080044a6      04d1           Bne   0x80044b2
|   |   ;-- flag.jailbreak:
|   |   0x080044a8    * aa20           Movs  r0,   0xaa
|   |   0x080044aa      fdf781fd       Bl    sym.rdp_lock
|   |   0x080044ae      fdf78bfd       Bl    sym.rdp_apply_lock
|   `-> 0x080044b2      fdf776fd       Bl    fcn.08001fa2
|      0x080044b6      00f097fa       Bl    sym.bootloader_pin_test
|      0x080044ba      dff8a404       Ldr.w r0,   [0x08004966]
|      0x080044be      0021           Movs  r1,   0
|      0x080044c0      0160           Str   r1,   [r0]
```

DFU dumping

- Flash memory mapped at 0, null deref bugs can be used to dump flash memory.
- 48KB of flash to dump the bootloader (..c000)
- DFU protocol available via USB.
- Some tools available in md380re for that
- Write r2 IO plugin

```
$ r2 dfu://0483:df11
```

Analysing code

- Don’t use auto analysis
- Basic block graph (VVn)
- Finding functions
- Finding memory accesses
- Finding pointers
- Special instructions
- Identify code / data
- Manual tweaks

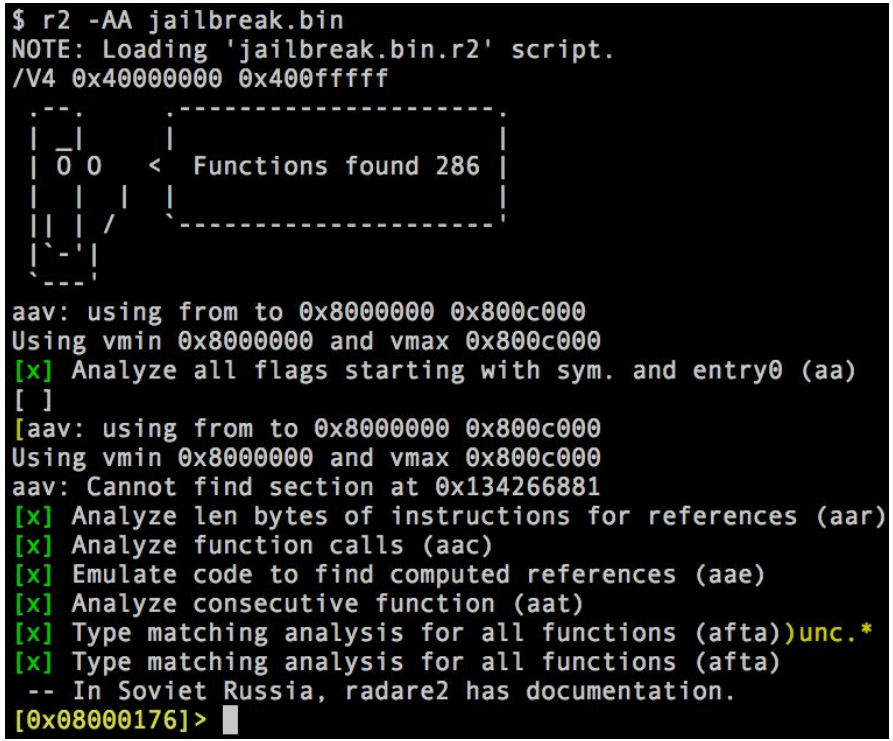
```
[0x08000000]> /A cmp
0x0800021c 2 cmp r2, 0
0x08000220 2 cmp r2, 2
0x0800022c 2 cmp r2, 6
0x08000268 2 cmp r4, 6
0x0800026c 2 cmp r4, 0xa
0x08000270 2 cmp r4, 0xb
0x08000288 2 cmp r2, 0xa
0x080002ac 2 cmp r2, 2
0x080002cc 2 cmp r1, 4
0x080002d8 2 cmp r1, 0
0x080002e4 2 cmp r1, 0
0x08000314 2 cmp r1, 5
0x08000378 2 cmp r1, 5
0x080003d0 2 cmp r0, 0x40
0x080003e8 2 cmp r1, 0xa2
0x080004e0 2 cmp r0, 0x71
0x0800053c 2 cmp r1, 0xc4
0x08000550 2 cmp r0, 0x70
0x0800056c 2 cmp r0, 0x88
0x08000590 2 cmp r1, 0xb2
0x080005a4 2 cmp r0, 0x70
0x080005bc 2 cmp r3, 0xb0
0x080005c4 2 cmp r0, 0x20
0x080005d4 2 cmp r1, 0xb3
0x080005e8 2 cmp r0, 0x70
0x0800060c 2 cmp r0, 0x18
0x0800061c 2 cmp r1, 0xd5
0x08000634 2 cmp r0, 0x70
0x0800064c 2 cmp r3, 0x24
0x08000654 4 cmp.w r0, 0x200
0x08000684 2 cmp r1, 0x23
0x0800069c 2 cmp r0, 0x40
0x080006bc 2 cmp r2, 0xb8
0x080006c0 2 cmp r1, r2
0x080006e0 2 cmp r4, r0
```

Finding Functions

- af
- afr
- e anal.hasnext
- pdf / pdr
- aab / aac / aar / aae

`$ r2 -A`

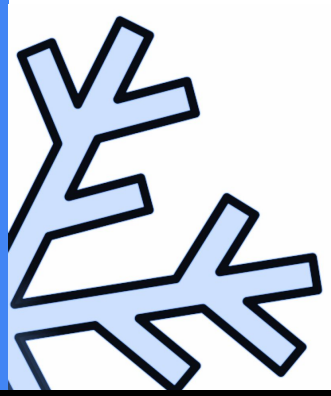
- aa
- aaa
- aaaa
- aaaaahhh!



Zignatures

- ‘Z’ chosen because ‘s’ was already taken for seek
- Identify common functions across multiple firmware versions or builds.
- Supports search.{from,to,in}
- Types of zignatures
  - Array of bytes
  - Analysis bytes with masks
  - Analysis function metrics
- FLIRT is supported.

Thanks Nibble!



```
$ r2 -c 'aac;zaF;z* > sig.r2' jailbreak.bin

[0x08000000]> z~graph?
286
[0x08000000]> z~bytes?
207

$ r2 -i sig.r2 -c z/ jailbreak.bin
```

Projects and scripts

- Projects are just r2 scripts and k=v dbs

- Run scripts with -i or .
- Save and load projects with P
- Export as r2 commands with *
  - There’s also json output if commands ends with ‘j’
- Xrefs and other heavy info is saved in SDB (my own k=v db)

String references

Probably the most useful thing when RE to find interesting blocks of code in your bins.

- Thumb2 instructions are 2 or 4 byte long
- No relocatable code
- Find dwords with /v
- Then find refs with /r

```
[0x08000000]> izzq~USB mode
0x5528 26 25 Digital Radio in USB mode
[0x08000000]> s..5528
[0x08005528]> /v $$
Searching 4 bytes in [0x8000000-0x800c000]
hits: 1
0x0800535c hit0_0 28550008
[0x08005528]> pxw 32 @ hit0_0
0x0800535c, 0x08005528 0x080055d0 0x080055f4 0x08005604  (U..U..U..V..
0x0800536c, 0x08005560 0x08005578 0x746e4940 0x616e7265  `U..xU..@Interna
[0x08005528]> /r 0x0800535c
[0x0800bfd0-0x0800c000] data 0x80052d0 ldr r0, [0x0800535c] in fcn.080052b8
data 0x80052c4 ldr r0, [0x0800535c] in fcn.080052b8
[0x08005528]>
```

Identifying Memory Mapped Devices

- Find RW refs /A load

```
[0x08000000]> pd 1 @e:asm.flags=0 @@= `/A load~[0]`
```

- Identify Cortex instructions

- 0x40000000 - IO peripherals (serial port, gps)
- 0xe0000000 - other memory mapped devices

Use /V4 0x40000000 0x4000f000

Identifying Memory Mapped Devices

```
[0x08000000]> pv4 @@= `C*~[3] `|sort -u
0x40000000
0x40007000
0x40013000
0x40013808
0x40016940
0x400169c0
0x40020000
0x40020400
0x40020800
0x40020c00
0x40021000
0x40023000
0x40023008
0x40023800
0x40023804
0x40023808
0x4002380c
0x40023830
0x40023834
0x40023840
0x40023844
0x40023c00
0x40023c04
0x40023c08
0x40023c0c
0x40023c10
0x40023c14
0x40023c15
0x40040000
0x40084910
0x40084944
[0x08000000]>
```

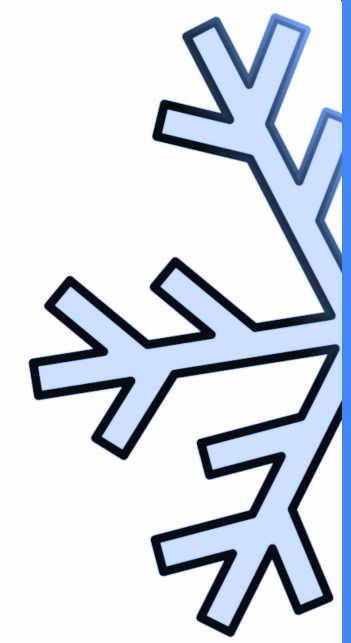
```
[0x08002058 16% 205 jailbreak.bin]> pd $r @ mdev.flash_protection
            ;-- mdev.flash_protection:
               ; DATA XREF from 0x08001fc2 (sym.rdp_lock)
               ; DATA XREF from 0x08001fe6 (sym.rdp_is_not_locked)
            0x08002058      .dword 0x40023c15
               ; DATA XREF from 0x08001ffc (fcn.08001ffa)
               ; DATA XREF from 0x08002008 (fcn.08001ffa)
               ; DATA XREF from 0x08002014 (fcn.08001ffa)
               ; DATA XREF from 0x08002022 (fcn.08001ffa)
            0x0800205c      .dword 0x40023c0c
┌ (fcn) fcn.08002060 40
|   fcn.08002060 ();
|              ; XREFS: CALL 0x08001eb8  CALL 0x08001f12  CALL 0x08001f42  CALL 0x08001f74  CALL 0x08001fb6
|              ; XREFS: CALL 0x08001fd8
|            0x08002060      80b5            Push {r7, lr}
|            0x08002062      0820            Movs r0,  8
|            0x08002064      8df8 ...        Strb.w r0,  [sp]
|            0x08002068      fff7c7ff        Bl    fcn.08001ffa          ;[1]
|            0x0800206c      8df8 ...        Strb.w r0,  [sp]
|        ┌─< 0x08002070      03e0            B     0x800207a             ;[2]
|        ┌──> 0x08002072      fff7c2ff        Bl    fcn.08001ffa          ;[1]
|        ||  0x08002076      8df8 ...        Strb.w r0,  [sp]
|        ↑|      ; JMP XREF from 0x08002070 (fcn.08002060)
|        |└─> 0x0800207a      9df8 ...        Ldrb.w r0,  [sp]
|        |    0x0800207e      0128            Cmp  r0,  1
|        └──< 0x08002080      f7d0            Beq  0x8002072             ;[3]
|            0x08002082      9df8 ...        Ldrb.w r0,  [sp]
└            0x08002086      02bd            Pop  {r1, pc}
```

Identifying Memory Mapped Devices

```
[0x08001fb0 16% 225 jailbreak.bin]> pd $r @ sym.rdp_lock
┌ (fcn) sym.rdp_lock 24
|   sym.rdp_lock ();
|           ; CALL XREF from 0x080044aa (fcn.080043bc)
|           0x08001fb0      38b5           Push {r3, r4, r5, lr}
|           0x08001fb2      04             Movs r4, r0
|           0x08001fb4      0825           Movs r5, 8
|           0x08001fb6        f053f8       Bl   fcn.08002060           ;[1]
|           0x08001fba      05             Movs r5, r0
|           0x08001fbc      edb2           Uxtb r5, r5
|           0x08001fbe      082d           Cmp  r5, 8
|       ┌─< 0x08001fc0      01d1           Bne  0x8001fc6              ;[2]
|       |   0x08001fc2      2548           Ldr  r0, [0x0800205a]       ; [0x8002058:4]=0x40023c15 ; LEA mdev.flash_protection ; mdev.flash_protection
|       |   0x08001fc4      0470           Strb r4, [r0]
└       └─> 0x08001fc6      31bd           Pop  {r0, r4, r5, pc}
┌ (fcn) sym.rdp_apply_lock 28
|   sym.rdp_apply_lock ();
|           ; CALL XREF from 0x080044ae (fcn.080043bc)
|           0x08001fc8      10b5           Push {r4, lr}
|           0x08001fca      0824           Movs r4, 8
|           0x08001fcc      1e48           Ldr  r0, [0x08002048]       ; [0x8002048:4]=0x40023c14
|           0x08001fce        78           Ldrb r0, [r0]
|           0x08001fd0      50f002         Orrs r0, r0, 2
|           0x08001fd4      1c49           Ldr  r1, [0x08002048]       ; [0x8002048:4]=0x40023c14
|           0x08001fd6      0870           Strb r0, [r1]
|           0x08001fd8        f042f8       Bl   fcn.08002060           ;[1]
|           0x08001fdc      04             Movs r4, r0
|           0x08001fde      20             Movs r0, r4
|           0x08001fe0      c0b2           Uxtb r0, r0
└           0x08001fe2      10bd           Pop  {r4, pc}
┌ (fcn) sym.rdp_is_not_locked 22
|   sym.rdp_is_not_locked ();
|           ; CALL XREF from 0x080044a0 (fcn.080043bc)
|           0x08001fe4        20           Movs r0, 0
|           0x08001fe6      1c49           Ldr  r1, [0x0800205a]       ; [0x8002058:4]=0x40023c15 ; LEA mdev.flash_protection ; mdev.flash_protection
|           0x08001fe8      0978           Ldrb r1, [r1]
|           0x08001fea      c9b2           Uxtb r1, r1
|           0x08001fec      aa29           Cmp  r1, 0xaa
|       ┌─< 0x08001fee      01d0           Beq  0x8001ff4              ;[3]
|       |   0x08001ff0      0120           Movs r0, 1
|      ┌──< 0x08001ff2        e0           B    0x8001ff6              ;[4]
|      |└─> 0x08001ff4        20           Movs r0, 0
|      |       ; JMP XREF from 0x08001ff2 (sym.rdp_is_not_locked)
|      └──> 0x08001ff6      c0b2           Uxtb r0, r0
└           0x08001ff8      7047           Bx   lr
```

Emulating Code

- Whats ESIL (Evaluable String Intermediate Language) ?
  - Stack-based Forth-like VM
  - Catch computed references
  - See values of registers at any time.
- Native debugger with ARM target.
  - Supports backstep, snapshots, threads, ...
  - Can’t run in MD380, needs a gdbstub or so



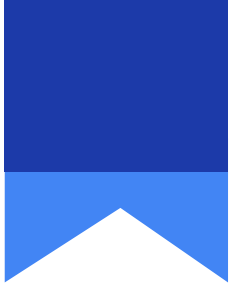
Extra Resources

—

- Travis Goodspeed talks at YouTube
- https://www.exploit-db.com/docs/pocorgtfo10.pdf
- https://github.com/roelandjansen/md380tools/

R2

- http://rada.re
- https://github.com/radareorg/r2con/



Questions?

Thanks for watching!

Thanks

—

- **Travis Goodspeed** and **DD4CR**
  - for the md380 jb, tools and research
- **NowSecure** for supporting me
- **R2 devs**
  - All r2 contributors, specially to:
  - **Nibble** (@jroimartin)
  - **Alvaro Felipe** (@alvarofe)
  - **Maijin** (@maijin)
  - **Skuater** (@sanguinawer)