Efficient Reverse Engineering of Automotive Firmware

Alyssa Milburn
Security Analyst, Riscure
milburn@riscure.com / @noopwafel

(with Niek Timmers)
Reverse Engineering

- Getting Firmware
- Reverse Engineering
- Understanding

- Tuning / manipulation
- IP
- Hacking
- ???

Efficient Reverse Engineering of Automotive Firmware
Automotive Firmware?
Instrument Cluster

- Speedometer/gauges
- Display (screen)
- Speaker!
- Blinky lights!
- 32-bit CPU
- CAN bus
- I2C bus
  - EEPROM
How can we get the firmware?

- External flash
- Leaks
- Debug interfaces
- Software vulnerabilities
- Hardware attacks
What makes this challenging?

• “Non-standard” platforms
• New concepts
• Complexity
What makes this challenging?

- **Static analysis** (disassembly): too complicated
- **Dynamic analysis** (emulation / debugging): no tools?

No tools?! Let’s make some!
What do we **need**?

- Processor (instruction set) emulator
- Timers, interrupts
- CAN controller
- I2C controller
  - EEPROM
  - Display controller
Emulating the CPU architecture

case:

    INSTX(or, "r%d, r%d", low, high);
    assert(high != 0);
    if (high != 0) {
        m_registers[high] |= m_registers[low];
        TAINt_REG_OR(high, low);
        ZERO_FLAG(m_registers[high]);
        NEG_FLAG(m_registers[high]);
        updatePSW(false, PSW_OV);
    }
    pc += 2;
    break;
“Implementing” peripherals

```c
    case 0x0000:
        //
        // not implemented yet
        break;

    case 0x0001:
        //
        break;

    case 0x0002:
        //
        // for now, we just pretend the clock initializes instantly
        printf("** clock init **\n");
        *(uint8_t *)&m_memory[addr] = 0;
        break;
```
How difficult was it?

~ 1 man-week of work

~ 3000 lines of (terrible) code (excluding support tooling)
Dynamic analysis
Debugging

- Step!
- gdb
  - "stub"
- Watch!
- Break!

Efficient Reverse Engineering of Automotive Firmware
Debugging

(gdb) hbreak *0x11032
Hardware assisted breakpoint 1 at 0x11032

(gdb) c
Continuing.

0x00011032 in ?? ()
(gdb)
Execution tracing

call getChecksumChunkSize, lp
mov r10, r7
mov r27, r6

call calculateChecksum, lp  -- r6 is pointer (note: skips first 2 bytes)
   -- r7 is size to check (in bytes)
   -- returns checksum in r10

cmp r10, r29
bz ret
xor 0xAAAA, r29, r0
bz ret
mov 0xFFFF, r0, r1

set 3, (g_globalIntegrityState - 0x3FF0000)[r1]
mov 1, r28  -- checksum was invalid (manipulation)

ret:
   -- CODE XREF: performChecksumVerification+1C↑j
   -- performChecksumVerification+22↑j

mov r28, r10
z r10
call pop_r26tor29_lp

-- End of function performChecksumVerification
Execution tracing

0x02920
0x02922 (jump)
0x02926
0x02928
0x0292c
0x02930
Execution tracing

0x02920
0x02922 (jump)
0x02926
0x02928
0x0292c
0x02930
Execution tracing

call getChecksumChunkSize, lp
mov r10, r7
mov r27, r6
call calculateChecksum, lp -- r6 is pointer (note: skips first 2 bytes)
  -- r7 is size to check (in bytes)
  -- returns checksum in r10
cmp r10, r29
bz ret
xor 0xA AAA, r29, r0
bz ret
mov 0xFFFF, r0, r1
set 3, (g_globalIntegrityState - 0x3FF0000)[r1]
mov 1, r28 -- checksum was invalid (manipulation)

ret:
  -- CODE XREF: performChecksumVerification+1C↑j
  -- performChecksumVerification+22↑j
mov r28, r10
z r10
call pop_r26tor29_lp
-- End of function performChecksumVerification
Execution tracing

call  getChecksumChunkSize, lp
mov   r10, r7
mov   r27, r6
call  calculateChecksum, lp  -- r6 is pointer (note: skips first 2 bytes)
   -- r7 is size to check (in bytes)
   -- returns checksum in r10

cmp   r10, r29
bz    ret

xor   0xAADA, r29, r0
bz    ret
mov   0xFFFF, r0, r1
set   3, (g_globalIntegrityState - 0x3FF0000)[r1]
mov   1, r28  -- checksum was invalid (manipulation)

ret:   -- CODE XREF: performChecksumVerification+1C↑j
       -- performChecksumVerification+22↑j

mov   r28, r10
z     r10
call  pop_r26tor29_lp

-- End of function performChecksumVerification
Hacks!
Hacks!

```c
if (m_pc == 0x\_\_\_\_\_) {
    // end of message display: print tmp buffer
    printf("\n");
    hexdump(&m_memory[0x\_\_\_\_], 30);
    printf("\n");
}

if (m_pc == 0x\_\_\_\_\_) {
    // segments on/off
    if (m_registers[7])
        printf("[on %02x: %02x] ", m_registers[6] >> 3, m_registers[6] & 0x7);
    else
        printf("[off %02x: %02x] ", m_registers[6] >> 3, m_registers[6] & 0x7);
}
```
State rewinding

Initial state

100ms boot time

Running (booted)

Send CAN message

Observe CAN response
Taint tracking

<table>
<thead>
<tr>
<th>1</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>CAN message</strong></td>
</tr>
<tr>
<td>3</td>
<td>??</td>
</tr>
<tr>
<td>4</td>
<td>??</td>
</tr>
<tr>
<td>5</td>
<td>??</td>
</tr>
<tr>
<td>6</td>
<td>??</td>
</tr>
<tr>
<td>7</td>
<td><strong>CAN message</strong></td>
</tr>
<tr>
<td>8</td>
<td>??</td>
</tr>
</tbody>
</table>

Data[2] = CAN.read()


Data[7] == Y?
Fuzzing

CAN message

Memory

Memory[5] == 0xc7?

Path 1

Path 2
UDS

./cc.py dcm discovery

CARING CARIBOU v0.1

-------------------
Starting diagnostics service discovery
Found diagnostics at arbitration ID 0x????, reply at 0x????
UDS: security access

Random key == calculateKey(seed)?

We found calculateKey!
UDS: security access

sending requestSeed (0x3)
CAN0: RCV [id ______] 02 27 03 aa aa aa aa aa
CAN0: TRQ [id ______] 06 67 03 47 2e 8e 70 aa

sending sendKey
CAN0: RCV [id ______] 06 27 04 41 9b 35 42 aa

comparison at 0002f390 (419b3542 vs 419b3542) is **tainted** with 000000c0

CAN0: TRQ [id ______] 02 67 04 aa aa aa aa aa
EEPROM contents

- Identification (VIN)
- Features/configuration
- (UDS) security state
- Odometer 😞

Reverse engineering is hard work!

$updateEEPROM(id, value)$
Takeaways

• Reverse engineering is not so hard!
• Lots of other “tricks” to try:
  • Symbolic execution
  • Deobfuscation (if necessary)
  • Smarter fuzzing
• You can’t hide secrets in firmware:
  • Use asymmetric cryptography (i.e. public keys)
  • Use the secure hardware inside modern processors
Thanks to...

Eloi Sanfelix
Santiago Cordoba
Ramiro Pareja
Efficient Reverse Engineering of Automotive Firmware

Alyssa Milburn
Security Analyst, Riscure
milburn@riscure.com / @noopwafel

Challenge your security

- Training
- Tools
- Services