Reverse-engineering embedded MIPS devices
Case study: Draytek SOHO routers

Nikita Abdullin
nabdullin@gmail.com

25.11.2011
Reverse-engineering embedded MIPS devices

- Embedded systems: overview
- Firmware research
- MIPS: Introduction
- MIPS: Reversing and exploitation
- Case study: DrayTek SOHO routers
Reverse-engineering embedded MIPS devices

Embedded systems: overview

Firmware research

MIPS: Introduction

MIPS: Reversing and exploitation

Case study: DrayTek SOHO routers
Embedded systems: overview

• Everywhere: mobile, media, CPE, ...
• High level of integration: mostly SoC
• Compact size, low power consumption
• ARM, MIPS, xScale, ...
• A real challenge for a reverser
  – Variety of models
  – Variety of architectures, conventions and APIs
  – Lack of information
Embedded systems: overview

Why ever reverse a MIPS box?

• Ubiquitous devices – especially network & CPE
• A well-known RISC platform
• Surprisingly, a little of security research:

Google: "mips exploit"

Search

About 64 results (0.12 seconds)
Reverse-engineering embedded MIPS devices

Embedded systems: overview

Firmware research

MIPS: Introduction

MIPS: Reversing and exploitation

Case study: DrayTek SOHO routers
Firmware research

Firmware – how to get it?

• Download from device ...
  – JTAG
  – Serial / UART console / other ports
  – read EEPROM directly, etc.

• ... or simply find a firmware upgrade
  – Single file, several MB’s in size
  – Encrypted / compressed
Firmware research

Typical firmware binary:
• Bootloader
• Code
• Filesystem image

How to extract them?
• Manual review
• Tools
Firmware research :: Tools

- Signature search
  - binwalk
  - signsrch, offzip
  - trid
- Scriptable hex editor
  - <Your favourite here>
- A set of archivers / unpackers
  - <Your favourite here>
- %script_lang_name%
Firmware research :: Manual review

- Sequences of 0x00’s and 0xFF’s
- First byte/word may be block size
- Last byte/word may be block checksum
- Magic signatures
- Text strings
  - Extract, sort, guess and google
- Entropy
Firmware research :: Manual review

- Filesystem extraction
  - Use binwalk
  - Or look for magic headers
    - JFFS2 = 85 19 (for FS nodes)
    - cramfs = 45 3D CD 28
    - YAFFS = 03 00 00 00 01 00 00 00 FF FF
    - SquashFS = "hsqs"
    - VFAT, etc. ...
Reverse-engineering embedded MIPS devices

- Embedded systems: overview
- Firmware research
- MIPS: Introduction
- MIPS: Reversing and exploitation
- Case study: DrayTek SOHO routers
MIPS: Introduction

- RISC ISA (instruction set architecture)
- Fixed 4-byte length instructions
- 32-bit and 64-bit
- Both big-endian and little-endian
- Popular and even more promising with multi-core Loongson series from China
## MIPS: registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>$0</td>
<td>Const = 0</td>
</tr>
<tr>
<td>$at</td>
<td>$1</td>
<td>Temporary</td>
</tr>
<tr>
<td>$v0–$v1</td>
<td>$2–$3</td>
<td>Return values</td>
</tr>
<tr>
<td>$a0–$a3</td>
<td>$4–$7</td>
<td>Function args</td>
</tr>
<tr>
<td>$t0–$t7</td>
<td>$8–$15</td>
<td>Temporary</td>
</tr>
<tr>
<td>$s0–$s7</td>
<td>$16–$23</td>
<td>Saved values</td>
</tr>
<tr>
<td>$t8–$t9</td>
<td>$24–$25</td>
<td>Temporary</td>
</tr>
<tr>
<td>$k0–$k1</td>
<td>$26–$27</td>
<td>Reserved for OS kernel</td>
</tr>
<tr>
<td>$gp</td>
<td>$28</td>
<td>Global pointer</td>
</tr>
<tr>
<td>$sp</td>
<td>$29</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>$fp</td>
<td>$30</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>$ra</td>
<td>$31</td>
<td>Return address</td>
</tr>
</tbody>
</table>
MIPS: delay slot

• MIPS is a pipelined architecture

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
</tr>
</tbody>
</table>

• So not all instructions are safe to be executed right after others
  – branches & jumps (bne beq j jr jal)
  – Loads (lb lw ld)
MIPS: memory layout

Virtual

- Kernel Mapped Cacheable: 0.5GB
- Kernel Mapped Cacheable: 0.5GB
- Kernel Unmapped Uncacheable: 0.5GB
- Kernel Unmapped Cacheable: 0.5GB
- User Mapped Cacheable: 2GB

Physical

- Memory: 0.5GB

Anywhere arrows connecting Virtual to Physical.
MIPS: memory layout

![Memory Layout Diagram]

- **Reserved**
- **Text segment**
- **Data segment**
- **Stack segment**
Reverse-engineering embedded MIPS devices

- Embedded systems: overview
- Firmware research
- MIPS: Introduction
- MIPS: Reversing and exploitation
- Case study: DrayTek SOHO routers
MIPS: Reversing and exploitation

- objdump / disasmips / radare
- IDA
  - IDAPython
  - QEMU-MIPS
- MIPS emulator
  - SPIM
  - OVPsim
MIPS: Reversing and exploitation

• Delay slots

```assembly
# SUBROUTINE

schedule_insns_bug:
    lw     $v0, dword_80559A10    # Load Word
    addiu  $sp, -0x10             # Add Immediate Unsigned
    sltiu  $v0, 1                 # Set on Less Than Immediate Unsigned
    jr     $ra                    # Jump Register
    addiu  $sp, 0x10              # Add Immediate Unsigned

# End of function schedule_insns_bug
```
MIPS: Reversing and exploitation

• Finding code patterns
  – 03 E0 00 08 (jr $ra -- return)
  – 0C xx xx xx (jal)
  – 3C xx xx xx (load – lui, la, ...)
  – 1x xx xx xx (branches)

```assembly
 alloc_etherdev:

 var_8 = -8

27 BD FF E8 addiu $sp, -0x18 # Add Immediate Unsigned
3C 05 80 60+ la $a1, aEthD # "eth%d"
3C 06 80 2F+ la $a2, ether_setup # Load Address
AF BF 00 10 sw $ra, 0x18+var_8($sp) # Store Word
0C 0B D8 B8 jal alloc_netdev # Jump And Link
00 00 00 00 nop
8F BF 00 10 lw $ra, 0x18+var_8($sp) # Load Word
03 E0 00 08 jr $ra # Jump Register
27 BD 00 18 addiu $sp, 0x18 # Add Immediate Unsigned

# End of function alloc_etherdev
```
MIPS: Reversing and exploitation

• Finding more details about the target system is a key to understanding kernel procedures

14000000 14000FFF  external bus general purpose i/o
18000300 180003FF  general purpose timer
1E100400 1E1007FF  first serial port
...
1E103100 1E1031FF  hardware crypto
1E104100 1E1041FF  dma controller
1E105300 1E1053FF  external bus controller
1E105400 1E1057FF  PCI bus controller
1E116000 1E117000  DSL interface
1E180000 1E1BFFFF  ethernet controller
1F101000 1F101FFF  external interrupt controller
...
1F203000 1F203FFF  reset controller
1F880200 1F8802FF  interrupt controller
1F8803F0 1F8803FF  watchdog timer
MIPS: Reversing and exploitation

• Basic buffer overflow
  – Overwrite the old $ra stored on stack
  – ret2libc
  – ...

<table>
<thead>
<tr>
<th>addr</th>
<th>Stack contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>new $sp→</td>
<td>X-28  allocated by the compiler</td>
</tr>
<tr>
<td></td>
<td>X-24  allocated by the compiler</td>
</tr>
<tr>
<td></td>
<td>X-20  local_buf[0] ... local_buf[3]</td>
</tr>
<tr>
<td></td>
<td>X-12  local_buf[8], local_buf[9] ...</td>
</tr>
<tr>
<td></td>
<td>X-8   Global pointer ($gp) from main()</td>
</tr>
<tr>
<td></td>
<td>X-4   Return address ($ra) back into main()</td>
</tr>
<tr>
<td>old $sp→</td>
<td>X      ...</td>
</tr>
</tbody>
</table>
MIPS: Reversing and exploitation

• Exploitation is more difficult than on x86 and ARM
  – Fixed-length instructions with a lot of null bytes
  – Word-aligned instructions (32 bit)
  – Word-aligned memory words (32 bit)
  – I-Cache vs. D-Cache (exploit has no effect, data & code is read & executed from cache)

• But protection techniques are still rather new
  – ASLR (MIPS Linux > 2.6.23)
  – Stack canaries (GCC > 4.5)
  – W ^ X not standardized and rarely implemented
Reverse-engineering embedded MIPS devices

Embedded systems: overview
Firmware research
MIPS: Introduction
MIPS: Reversing and exploitation
Case study: DrayTek SOHO routers
Case study: DrayTek SOHO routers
Case study: DrayTek SOHO routers

• DrayTek Vigor V2xxx series – network devices
  – Over 35 models, widely deployed by ISP’s over Australia, Germany, UK
    • Routers
    • ADSL-modems
    • CPE & combo devices (dual wan + VoIP + ... )

• MIPS and ARM platforms
  – POI: Infineon Danube-S (MIPS, dual-core, HW crypto)

• Proprietary closed-source OS: DrayOS
  – Single static non-relocatable binary
  – Capable of loading Linux drivers
Case study: DrayTek SOHO routers

• Router Web UI allows to download obviously obfuscated binary config for further reloading
• Firmware upgrades available from vendor, but they are compressed using unknown algorithm
• Rumours are that a master key based on MAC address exist
Case study: DrayTek SOHO routers

Research steps.

1. Config file – decrypt
2. UART console and debug menu
   1. Firmware dumped through UART hexdump 100 bytes at a time 😊
3. Firmware reversing
4. Config file – decompress
5. Firmware – decompress
6. Filesystem – decompress & extract
7. Master password generation algorithm discovered
Config file

1. A way to get encrypted and unencrypted configs simultaneously was discovered
   1. Immediate result – a cipher used is definitely in ECB mode, bytewise substitution table is made
   2. Substitution table shows strong symmetry, the cipher is extremely weak
2. Now try to uncompress – firmware research suggests to try LZO algorithm
3. Success!
4. Passwords offsets are found to be constant in the config file
Config file

```c
struct CFG_HEADER {
    uint32   file_size;     // Total file size
    byte    magic0 [8];    //
    uint16  modelid;       // Big-endian BCD model number, e.g. \x27\x10 = 2710
    byte    signature1 [18]; //
    uint32  magic1;        //
    uint32  cfg_size;      // Config data size, without header and trailing bytes
    byte    flags [8];     //
    uint32  v2k_checksum;  // V2kChecksum checksum of unencrypted data
    byte    reserved [204]; //
};
```

- The config file content itself is a slice of memory
Config file

```
loc_8038EDF4:     # Set on Less Than Unsigned
    sltu    $v0, $a2, $a0
    addu    $v1, $a1, $a2   # Add Unsigned
    beqz    $v0, loc_8038EDF0  # Branch on Zero
    addiu   $a2, 1   # Add Immediate Unsigned

loc_8038EDF0:     # Branch on Less Than or Equal to Zero
    move    $a2, $zero
    blez    $s0, loc_8038EE2C  # Branch on Less Than or Equal to Zero
    andi    $a3, 0xFF   # AND Immediate

loc_8038EDFC:     # Add Unsigned
    addu    $a0, $s1, $a2
    lbu     $v0, 0($a0)  # Load Byte Unsigned
    addiu   $a2, 1   # Add Immediate Unsigned
    slt     $a1, $a2, $s0  # Set on Less Than
    xor     $v0, $a3  # Exclusive OR
    subu    $v0, $a3  # Subtract Unsigned
    andi    $v0, 0xFF  # AND Immediate
    srl     $v1, $v0, 5  # Shift Right Logical
    sll     $v0, 3  # Shift Left Logical
    or      $v0, $v1  # OR
    bnez    $a1, loc_8038EDFC  # Branch on Not Zero
    sb      $v0, 0($a0)  # Store Byte
```
def make_key(modelstr):
    sum = 0
    for c in modelstr:
        sum += ord(c)
    return (0xFF & sum)

# Make a key out of model string
# sum character codes of the string
# take only lower byte of the sum (= mod 256)

def enc(c, key):
    c ^= key
    c -= key
    c = 0xFF & (c >> 5 | c << 3)
    return c

# Encrypt a byte of data
# XOR a byte with a key
# Subtract a key from the previous step result
# Swap 3 leftmost bits and 5 rightmost bits
Firmware file

[4 bytes   ] = the size of (loader code + main code).
[4 bytes   ] = currently unknown purpose
[248 bytes ] = rest of header, zeroes
[varies    ] = firmware loader code (uncompressed)
[8 bytes   ] = loader end signature - 8 bytes ending in A55AA55A, XXA5A5A55AA55A or XX5A5AA55AA55A

[4 bytes   ] = compressed code size
[varies    ] = compressed firmware code
[4 bytes   ] = compressed filesystem image total size
[4 bytes   ] = compressed filesystem image data size
[varies    ] = compressed filesystem image
[varies    ] = garbage or padding at the end
Filesystem

[16 bytes   ] = filesystem header block
[N*44 bytes ] = file nodes block, 44 bytes in each node
[N*variable ] = compressed files block, all files are simply
    concatenated, no padding or alignment

struct FS_HEAD {
    char magic  [8];   // "PFS/1.0\x00"
    byte unused [6];   // zeroes
    uint16 num_files; // number of files
}
struct FS_ENTRY {
    char fname [32];   // file name, right zero-padded
    uint32 hash;       // some hash, not researched yet
    uint32 offset;     // offset from the beginning of common compressed
        file block
    uint32 size;       // compressed file size
}
Master key

• Derived from local interface MAC address
• 8-byte, alternating case letters, e.g. ‘AaBbCcDd’
• Login allowed only from local addresses to FTP
• The same algorithm for all V2xxx devices (from 2002 to 2012)
  – Compute a simple polynom over MAC address bytes
  – Divide by 26 and map result to reordered alphabet
  – Repeat until all chars filled
<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Whether this will happen ???? I hope it would never happen ---- Fanny</td>
</tr>
<tr>
<td>C</td>
<td>===&gt;Please Tell Fanny : When will this happen.....?????????</td>
</tr>
<tr>
<td>C</td>
<td>N-Channel nkt size &gt; %d (Eyreeds N-Channel buffer size = %d)</td>
</tr>
</tbody>
</table>
Keywords & links

binwalk, signsrch, SPIM, disasmips, OVPsim, QEMU-mips
MIPS linux wiki www.linux-mips.org

Case study reference:
Draytools github.com/ammonium