IoT Security and Privacy
Attacks against IoT

YIER JIN

UNIVERSITY OF FLORIDA

EMAIL: YIER.JIN@ECE.UFL.EDU

SLIDES ARE ADAPTED FROM PROF. XINWEN FU @ UCF/UMASS

Learning Outcomes

Upon completion of this unit:

- Students will be able to explain attacks against IoT system (hardware + software)
- Students will be able to explain attacks against IoT network protocols
- Students will be able to explain attacks against industry IoT

Prerequisites and Module Time

Prerequisites

- Students should have taken classes on operating system and computer architecture.
- Students should know basic concepts of networking.

Module time

- Three-hour lecture
- One-hour homework

Main References

[1] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum and N. Ghani, "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations," in *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702-2733, thirdquarter 2019.

Outline

Introduction

Attacks against IoT system (hardware + software)

IoT reverse engineering and forensics

Attacks against IoT network protocols

Attacks against Industry IoT (IIoT)

Introduction

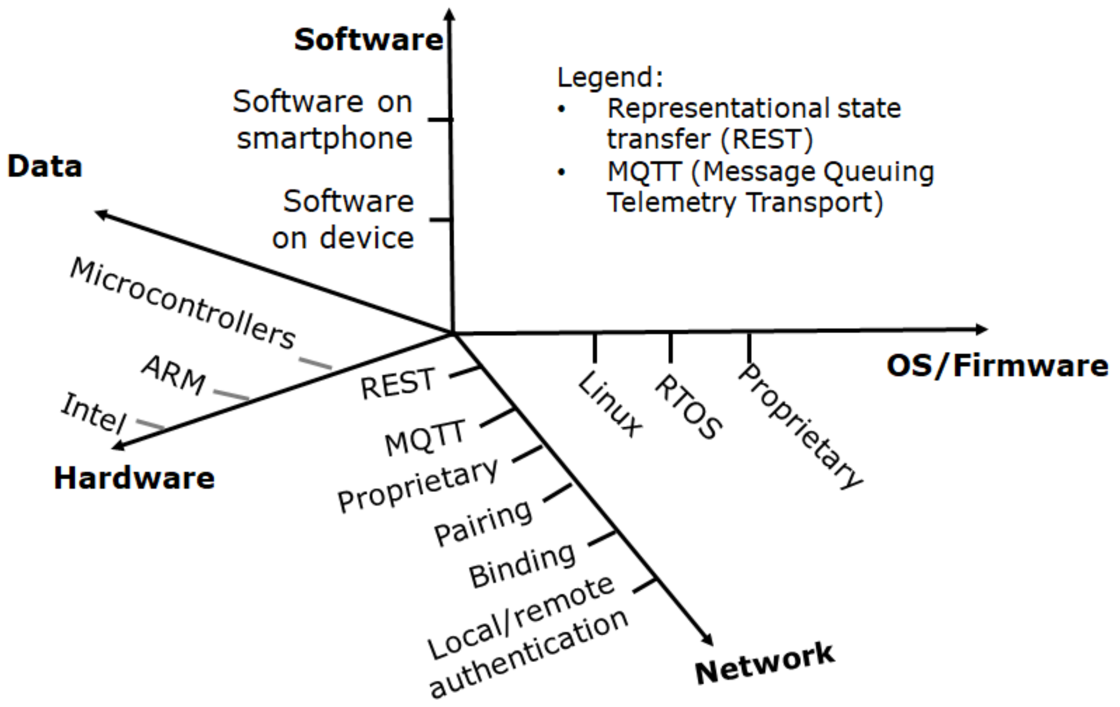
Attacks can be directed against hardware, OS/Firmware, network, software and data

Any design flaws leave vulnerabilities

- We will show examples of attacks later

Learning to think like an attacker to better design defense

- Because of attacks, defense is needed, right?



Outline

IoT Hardware Security

Hardware security is critical when attackers can physically access the IoT devices.

Many IoT devices do not disable their debugging ports after the testing and validation stage, which give attackers full access to the internal firmware.

Most IoT devices have hardware vulnerabilities which may be exploited by attackers including

- UART/JTAG debugging ports,
- Multiple boot options, and
- Unencrypted flash memory.

Through the hardware backdoors, attackers can easily bypass software level integrity checking by either disabling the checking functionality or booting the system through an injected firmware image.

Universal asynchronous receiver-transmitter (UART)

A UART controller is a microchip and a key component of serial communication of a SoC (System on a Chip).

At the transmitter side, a UART takes bytes of data and sends them out bit by bit sequentially.

At the receiver side, a UART reassemble the bits into bytes.

A debugger can hook a UART cable to the SoC and log into the operating system for debugging.

A UART port is often left on the device board for the sake of technical support in order to figure out what may go wrong when a device has problems.

Joint Test Action Group (JTAG)

JTAG serves two major roles: boundary scan and debug access.

With boundary scan, one can ensure components of a device are correctly connected.

With debug access, one can interact the SoC and access registers, memories and even pause and redirect the instruction flow.

- Including the firmware.

SPI Bus (Serial Peripheral Interface)

Flash memory is often connected via SPI to the CPU on a SoC.

We can desolder the flash and remove it from the IoT device board.

Then we can dump the content through a chip programmer or a bus pirate to dump the flash memory to a computer.

We may dump the flash when it is in circuit on the board.

In this case, care has to be taken not to interfere with the running board and cause the failure of dumping.

Example Hack – an IP camera

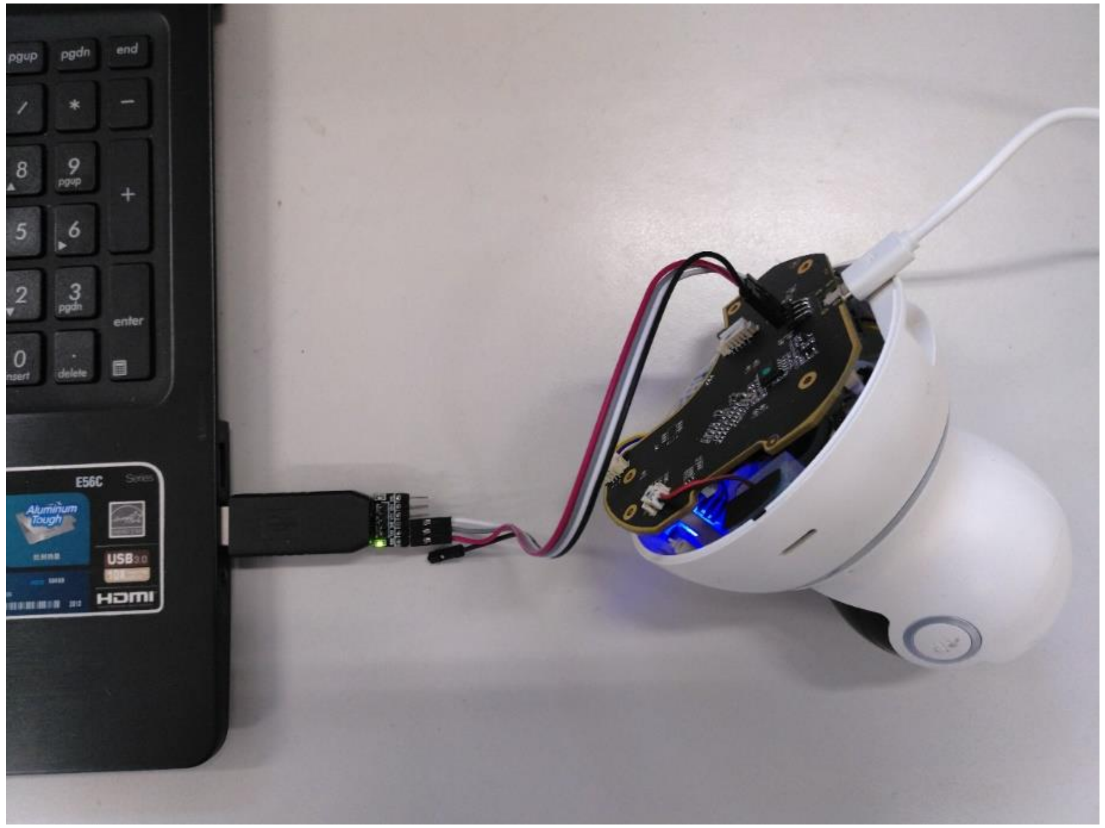
Locate the UART interface on the IoT device board such as a smart camera

Use a USB to UART bridge to connect the IoT device to the computer

Set up the terminal software (such as Putty) to communicate with the IoT device

Power on the IoT device

- If lucky, no password!

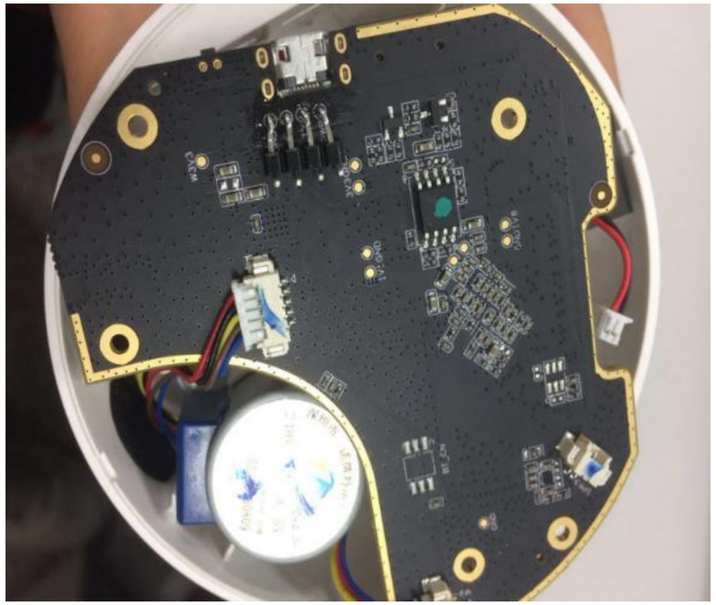


Locate the UART Interface

Carefully disassembly the IoT device

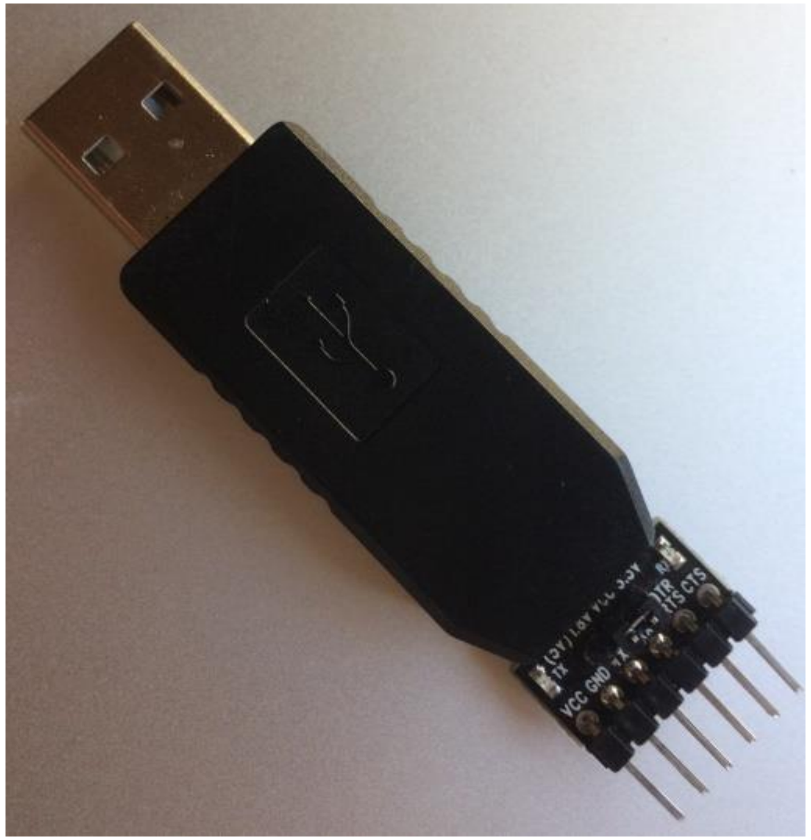
Use a voltmeter to identify VCC,TX,RX,GND of the UART interface

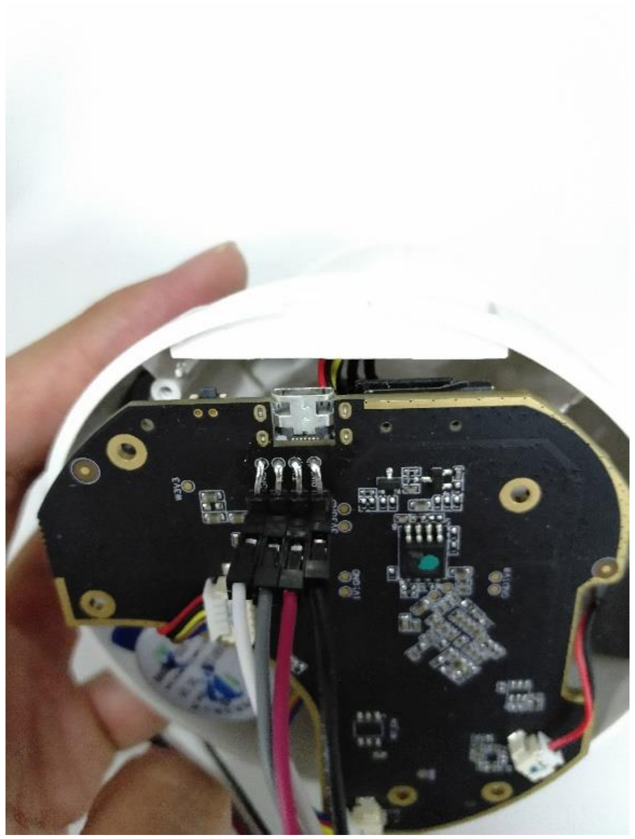
- GND – 0v
- VCC – 3.3v or 5v
- TX – peak 3.3V but vibrating
- RX – often close to TX



USB to UART bridge from device to computer

Install the driver of the bridge





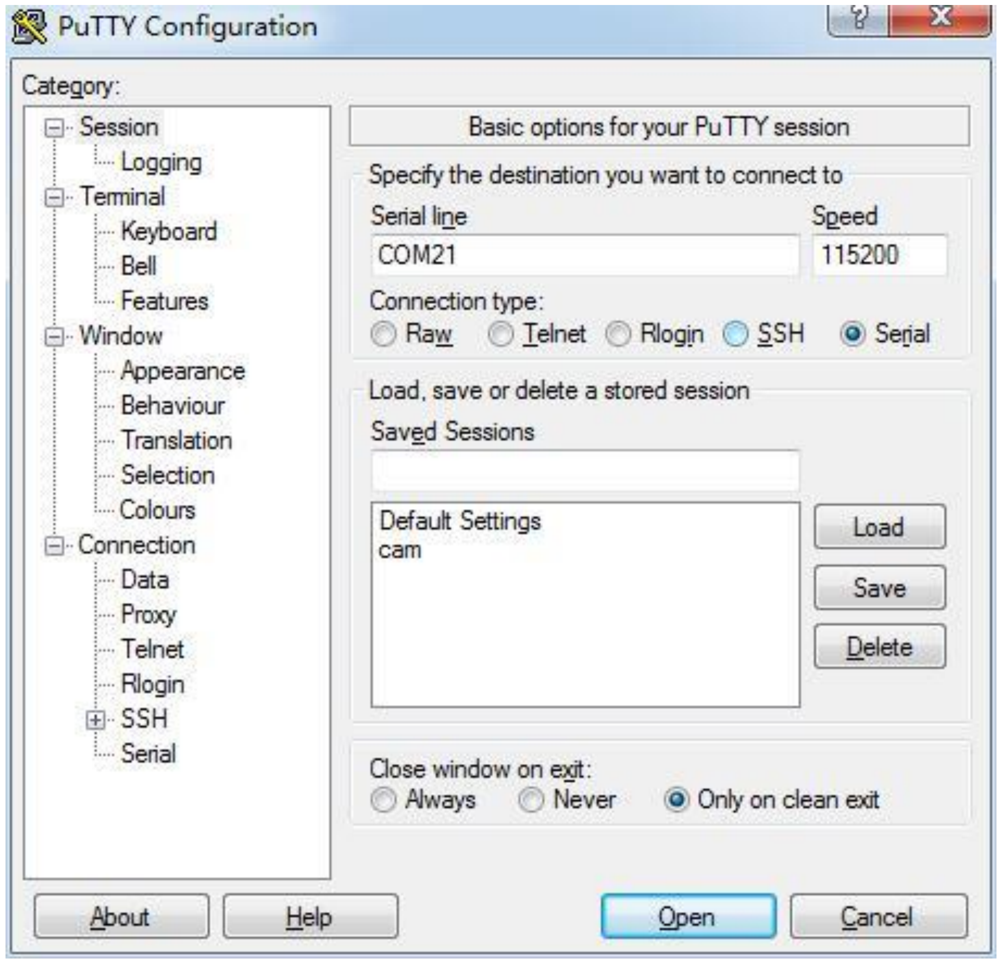
Set Up the Terminal Software

Use the device management software to identify the communication port of the USB to USRT bridge (Com or LPT)

- For example, USB SERIAL CH340 (COM21)

Configure Putty

Power on the IoT device



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM21

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

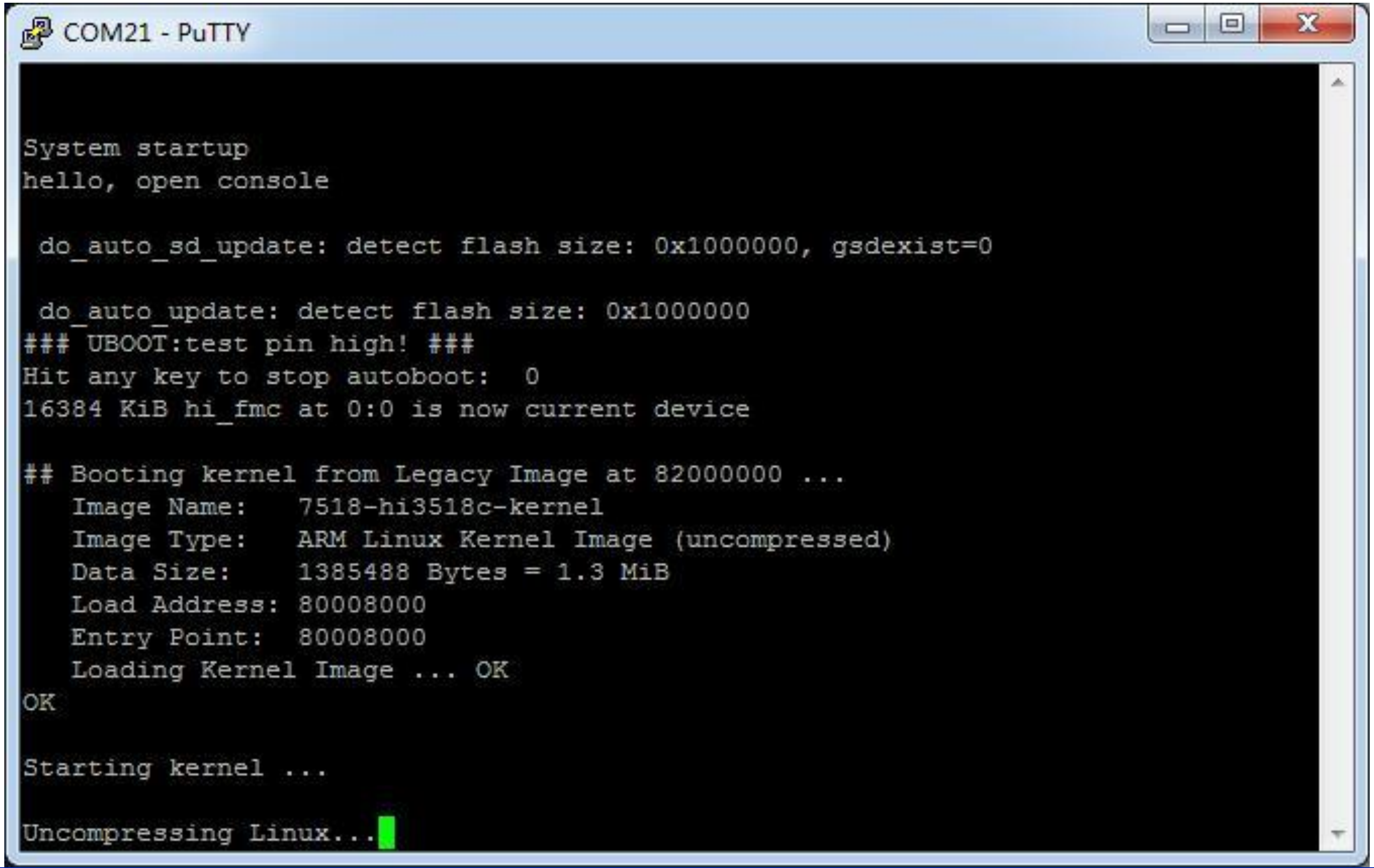
Load, save or delete a stored session

Saved Sessions:

Default Settings
cam

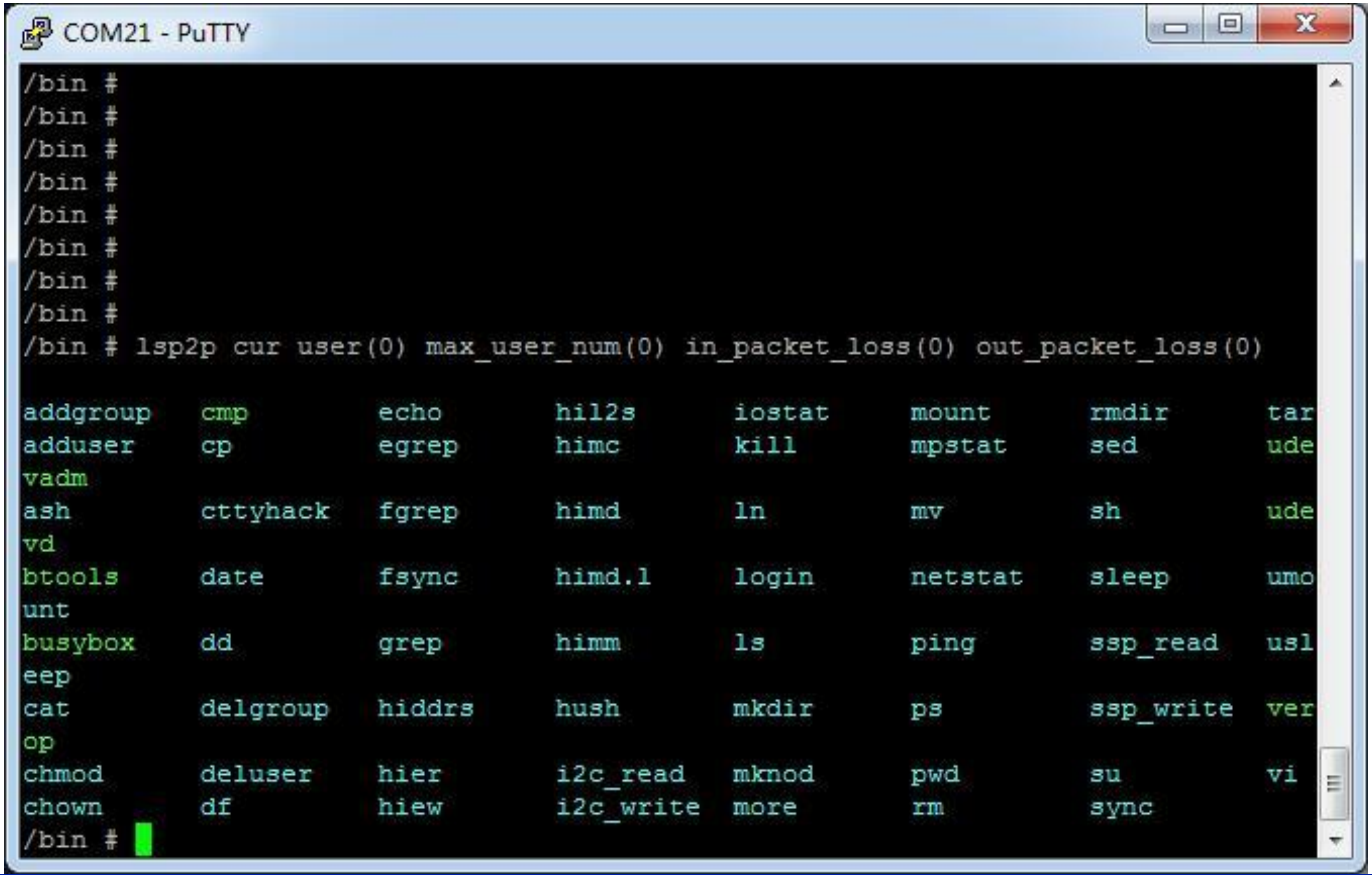
Close window on exit: Always Never [✓ Only on clean exit]

Example: a camera



```
System startup
hello, open console

 do_auto_sd_update: detect flash size: 0x1000000, gsdexist=0

 do_auto_update: detect flash size: 0x1000000
### UBOOT:test pin high! ###
Hit any key to stop autoboot:  0
16384 KiB hi_fmc at 0:0 is now current device

## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   7518-hi3518c-kernel
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1385488 Bytes = 1.3 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux...
```

Example (Cont'd)


COM21 - PuTTY

```
/bin #
/bin #
/bin #
/bin #
/bin #
/bin #
/bin #
/bin #
/bin # lsp2p cur user(0) max_user_num(0) in_packet_loss(0) out_packet_loss(0)

addgroup    cmp         echo        hil2s       iostat      mount       rmdir       tar
adduser     cp          egrep       himc        kill        mpstat      sed         ude
vadm
ash         cttyhack    fgrep       himd        ln          mv          sh          ude
vd
btools      date        fsync       himd.l      login       netstat     sleep       umo
unt
busybox     dd          grep        himm        ls          ping        ssp_read    usl
eep
cat         delgroup    hiddrs      hush        mkdir       ps          ssp_write   ver
op
chmod       deluser     hier        i2c_read    mknod       pwd         su          vi
chown       df          hiew        i2c_write   more        rm          sync
/bin #
```

Then what?

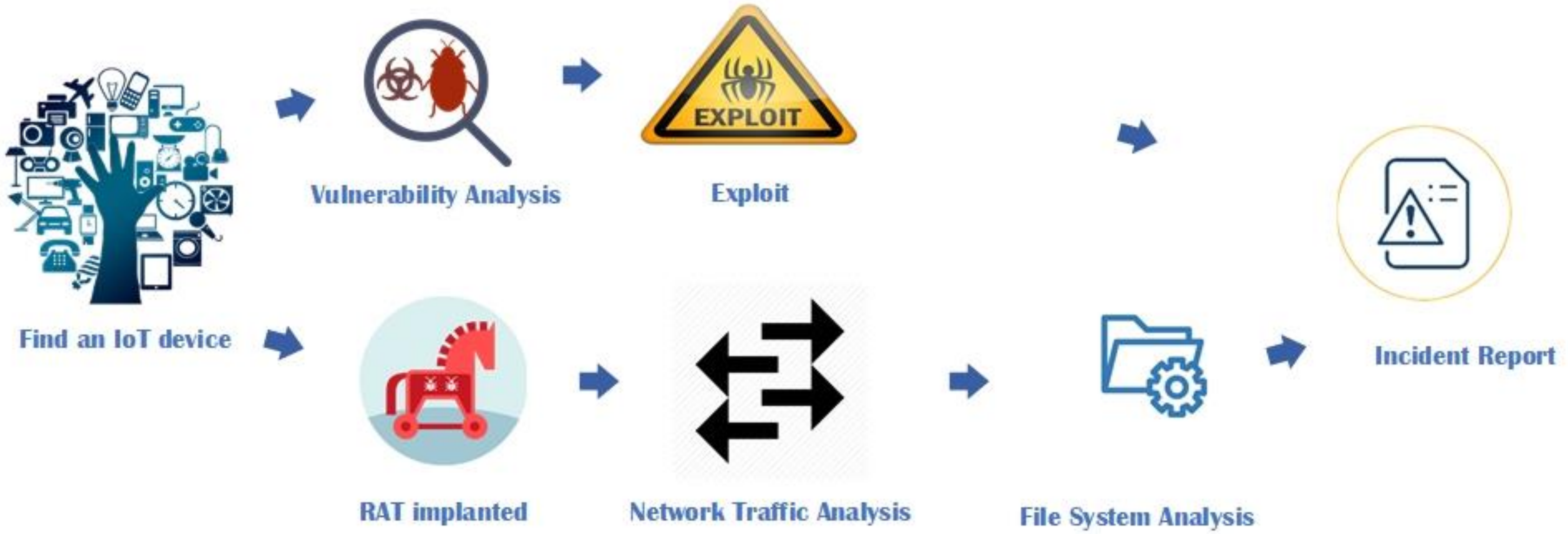
Now we can analyze the system software and see if there are vulnerabilities

We can also use the MITM to analyze the communication protocols at the side of the IoT device and identify vulnerabilities

- Mitmproxy
- May need to change CA certificate in the device

Outline

IoT Reverse Engineering and Forensics



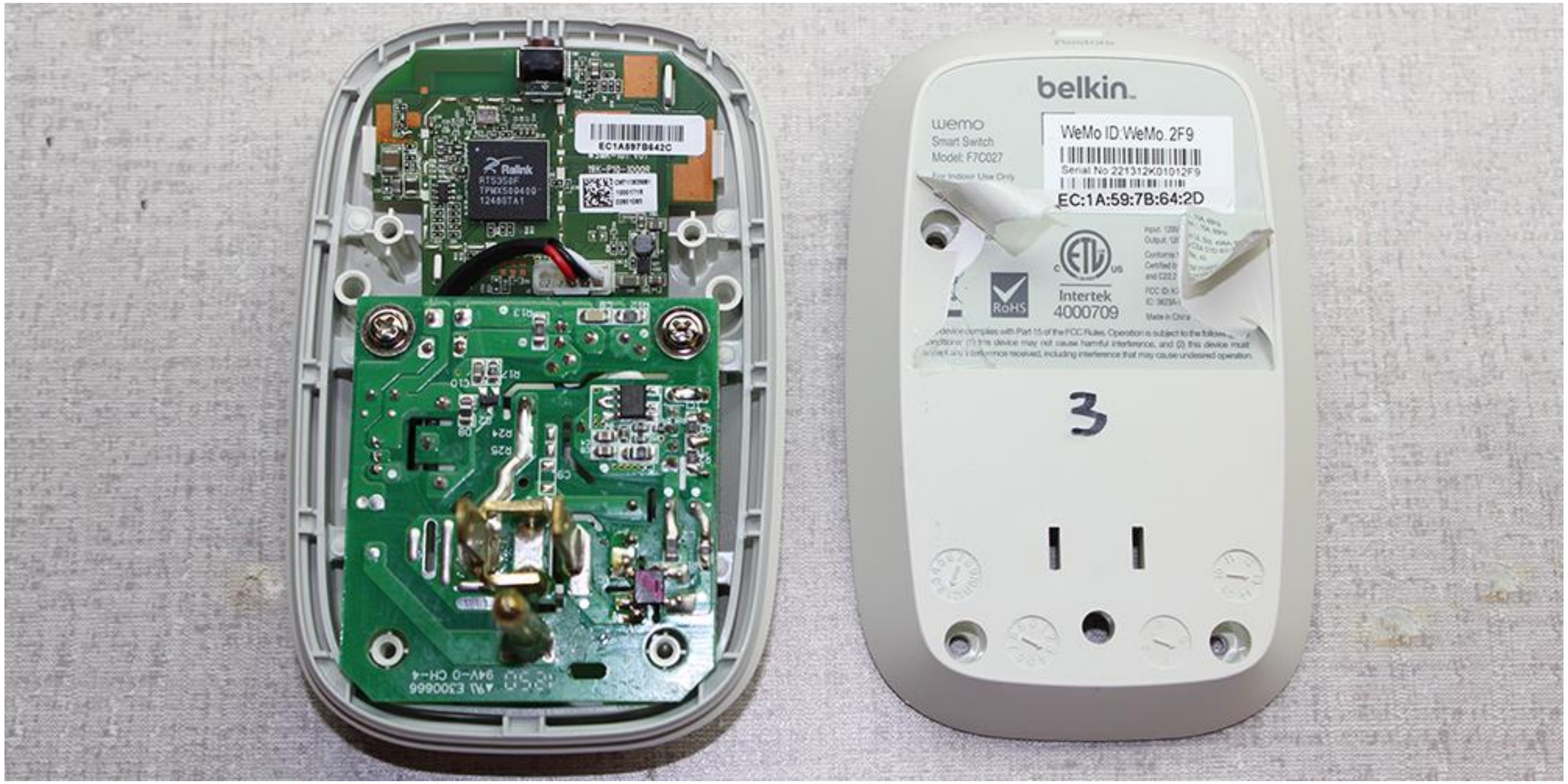
IoT Reverse Engineering

- Practical Reverse Engineering: Belkin WeMo WiFi Switch

Practical Reverse Engineering

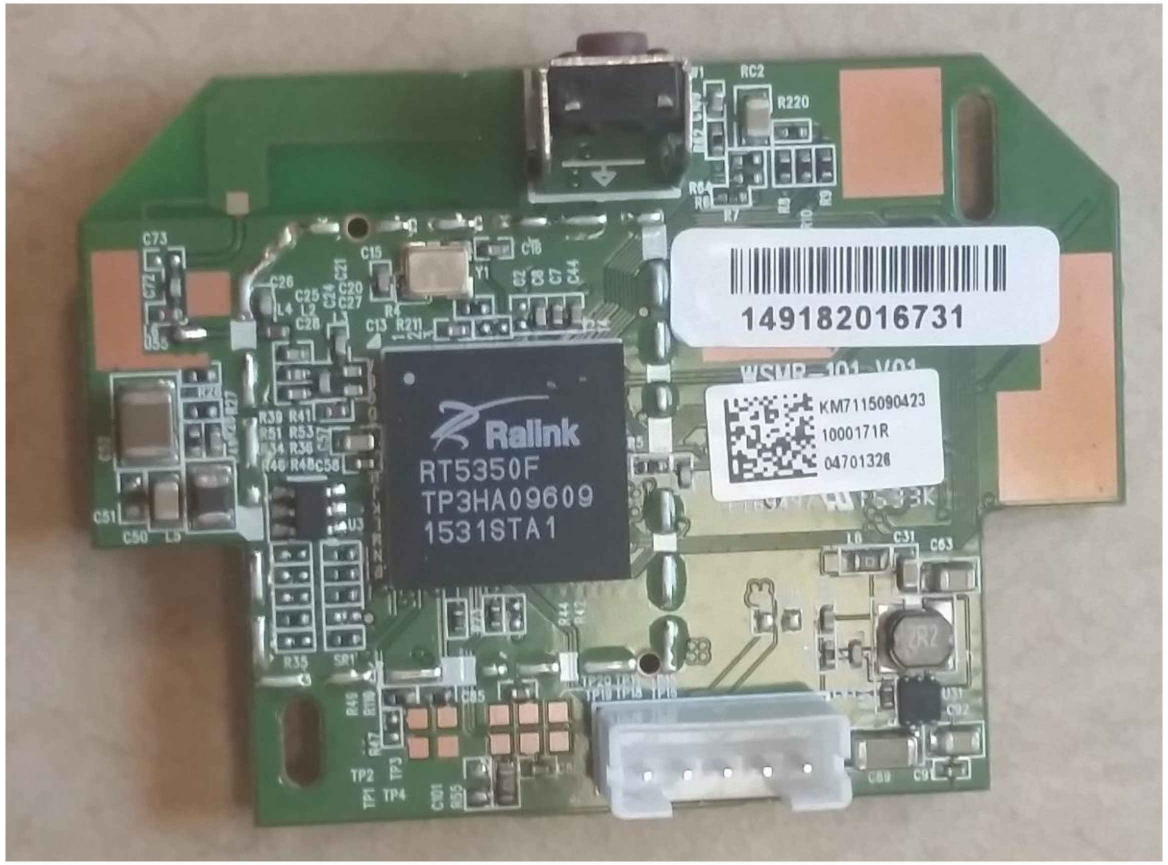
- Identify the components
- Identify debug ports
- Dump the flash
- Extract/Analyze the firmware

Identifying the Components

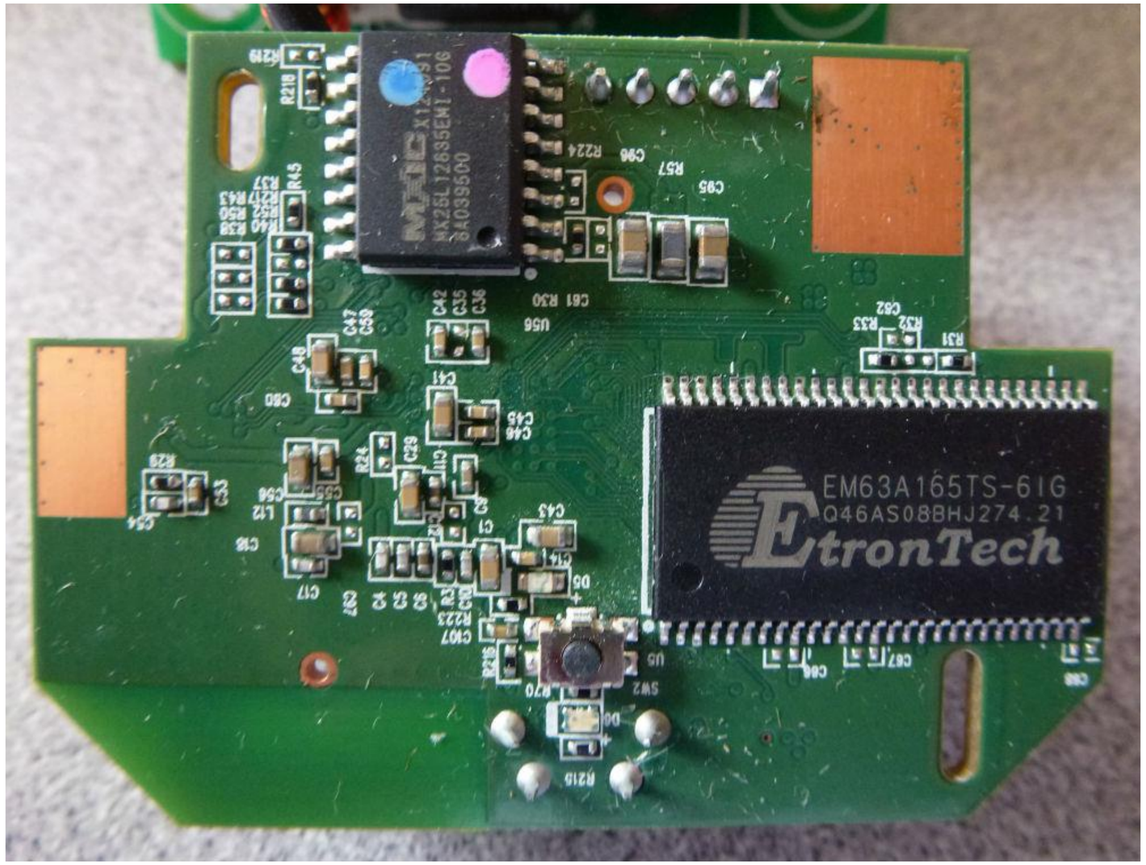


Wemo Wireless switch tear down

Identifying the Components



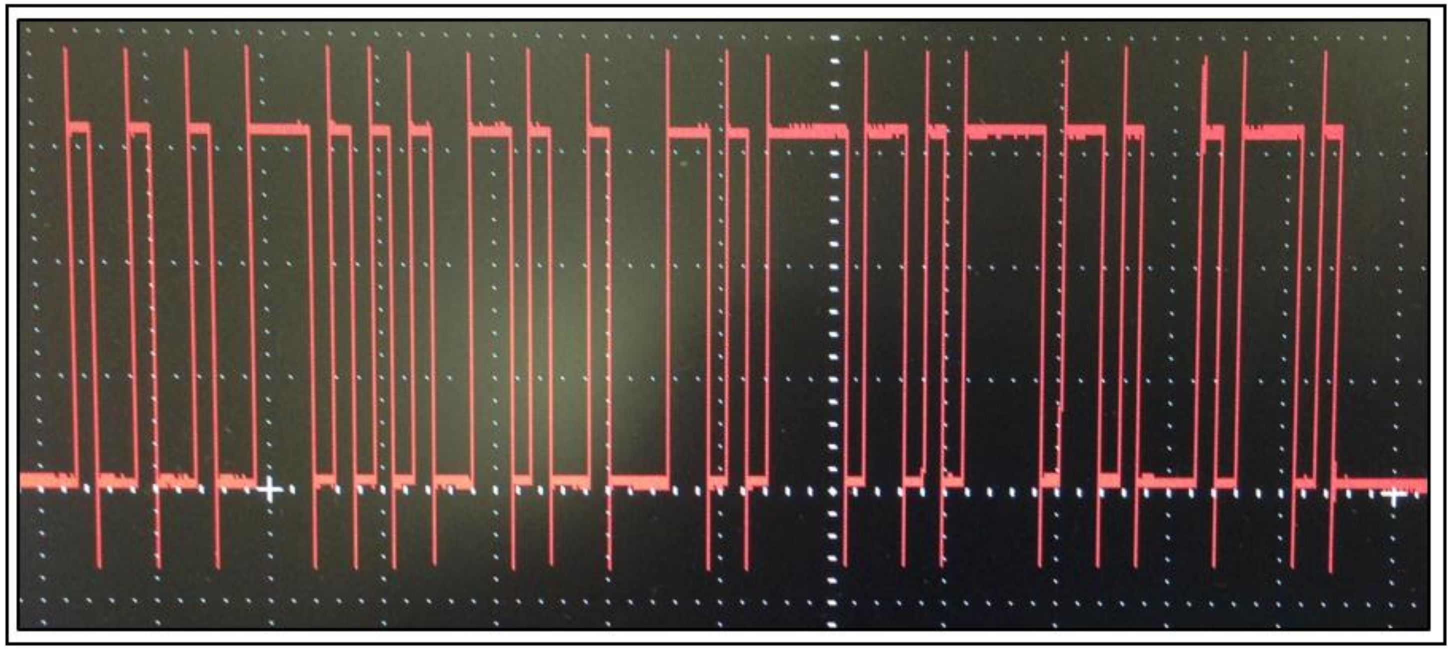
Wemo digital board top

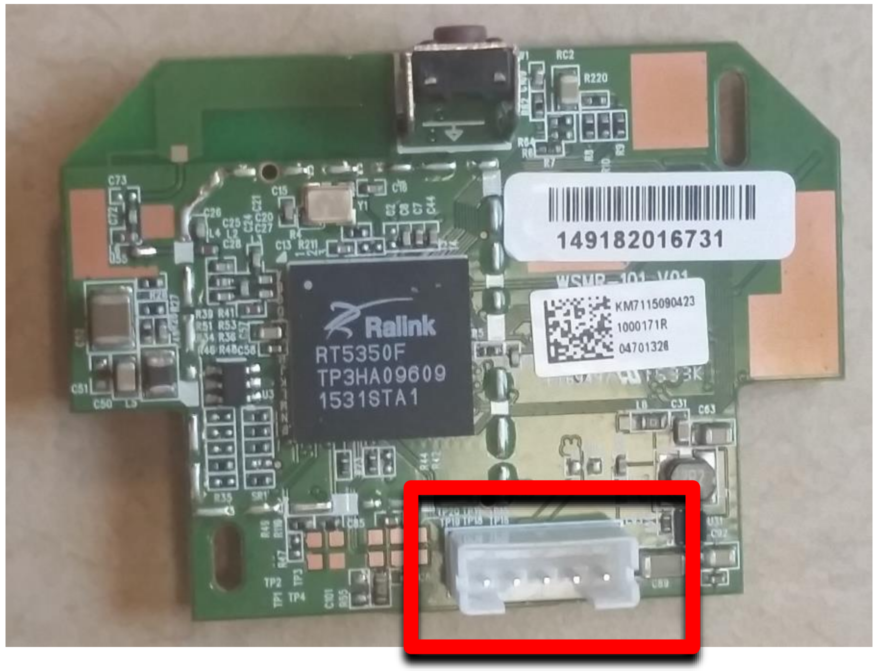


Wemo digital board bottom

Identifying Debug Ports

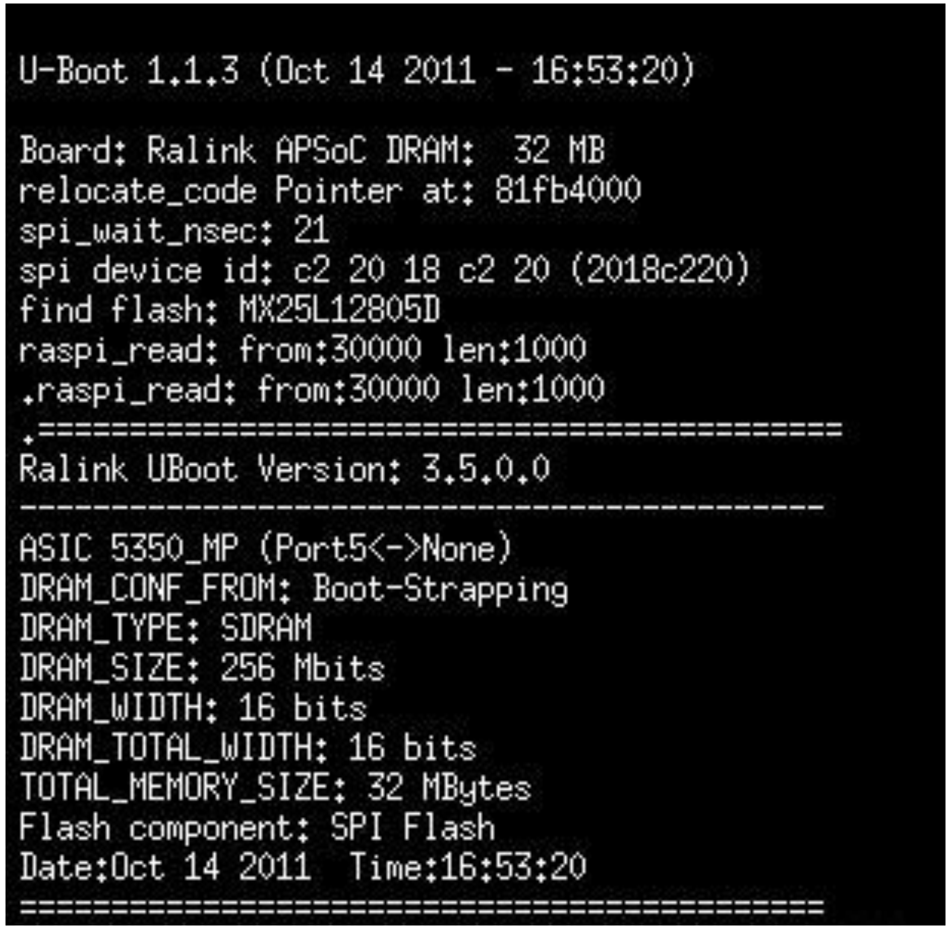
- JTAG (Joint Test Action Group)
- UART (Universal Asynchronous Receiver/Transmitter)


Example UART oscilloscope capture

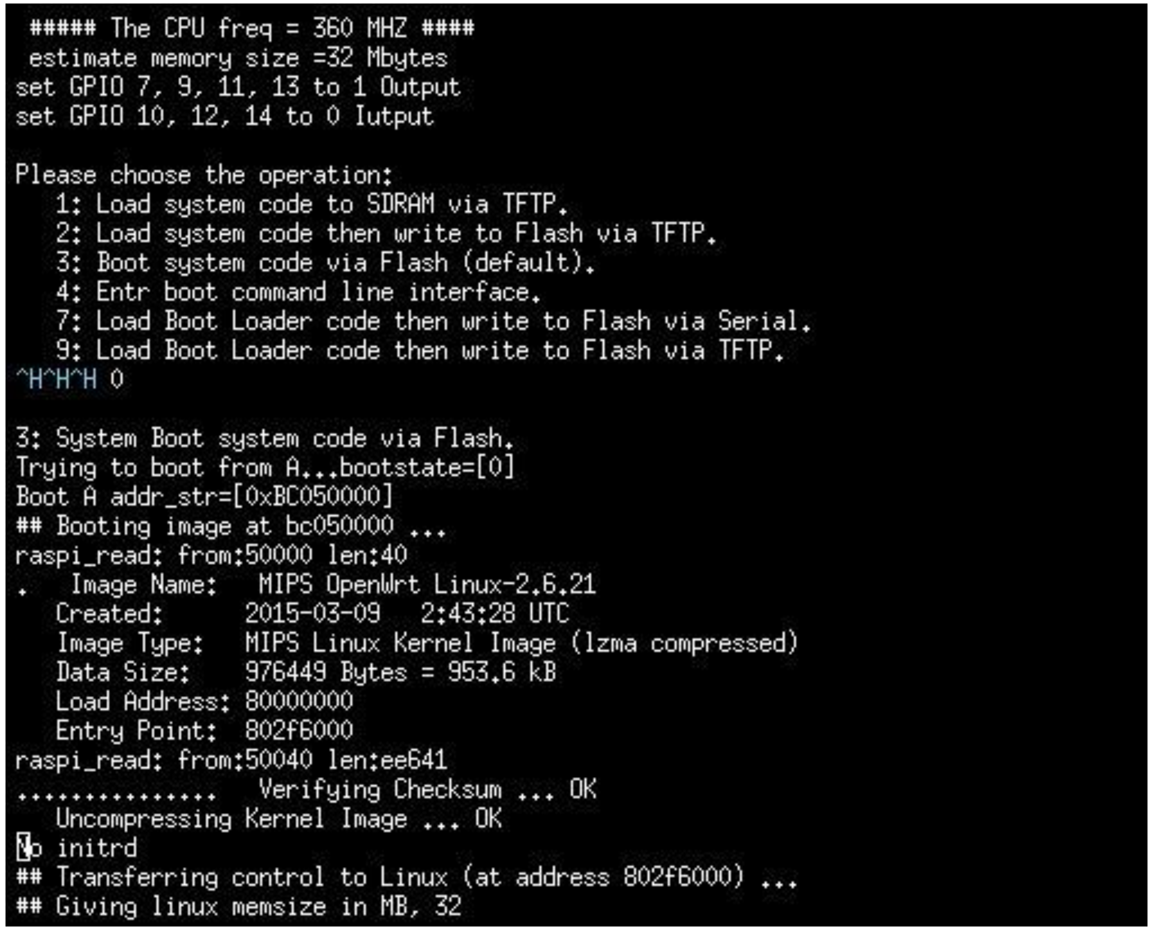


UART Ports

Identifying Debug Ports



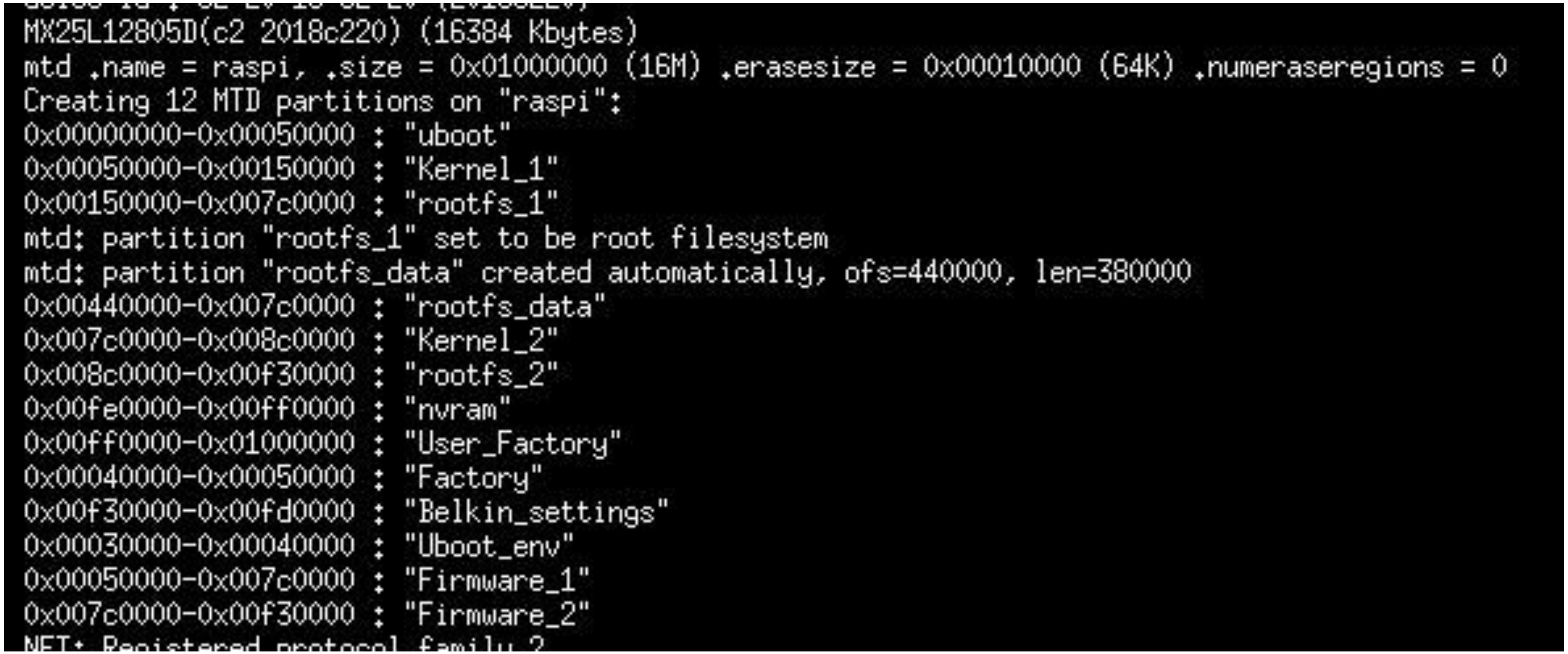
```
U-Boot 1.1.3 (Oct 14 2011 - 16:53:20)

Board: Ralink APSoC DRAM:  32 MB
relocate_code Pointer at: 81fb4000
spi_wait_nsec: 21
spi device id: c2 20 18 c2 20 (2018c220)
find flash: MX25L12805D
raspi_read: from:30000 len:1000
.raspi_read: from:30000 len:1000
.============================================
Ralink UBoot Version: 3.5.0.0
--------------------------------------------
ASIC 5350_MP (Port5<->None)
DRAM_CONF_FROM: Boot-Strapping
DRAM_TYPE: SDRAM
DRAM_SIZE: 256 Mbits
DRAM_WIDTH: 16 bits
DRAM_TOTAL_WIDTH: 16 bits
TOTAL_MEMORY_SIZE: 32 MBytes
Flash component: SPI Flash
Date:Oct 14 2011  Time:16:53:20
============================================
```



```
 ##### The CPU freq = 360 MHZ ####
 estimate memory size =32 Mbytes
set GPIO 7, 9, 11, 13 to 1 Output
set GPIO 10, 12, 14 to 0 Iutput

Please choose the operation:
   1: Load system code to SDRAM via TFTP.
   2: Load system code then write to Flash via TFTP.
   3: Boot system code via Flash (default).
   4: Entr boot command line interface.
   7: Load Boot Loader code then write to Flash via Serial.
   9: Load Boot Loader code then write to Flash via TFTP.
^H^H^H 0

3: System Boot system code via Flash.
Trying to boot from A...bootstate=[0]
Boot A addr_str=[0xBC050000]
## Booting image at bc050000 ...
raspi_read: from:50000 len:40
.   Image Name:   MIPS OpenWrt Linux-2.6.21
   Created:      2015-03-09   2:43:28 UTC
   Image Type:   MIPS Linux Kernel Image (lzma compressed)
   Data Size:    976449 Bytes = 953.6 kB
   Load Address: 80000000
   Entry Point:  802f6000
raspi_read: from:50040 len:ee641
...............    Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
No initrd
## Transferring control to Linux (at address 802f6000) ...
## Giving linux memsize in MB, 32
```

Boot log from UART port

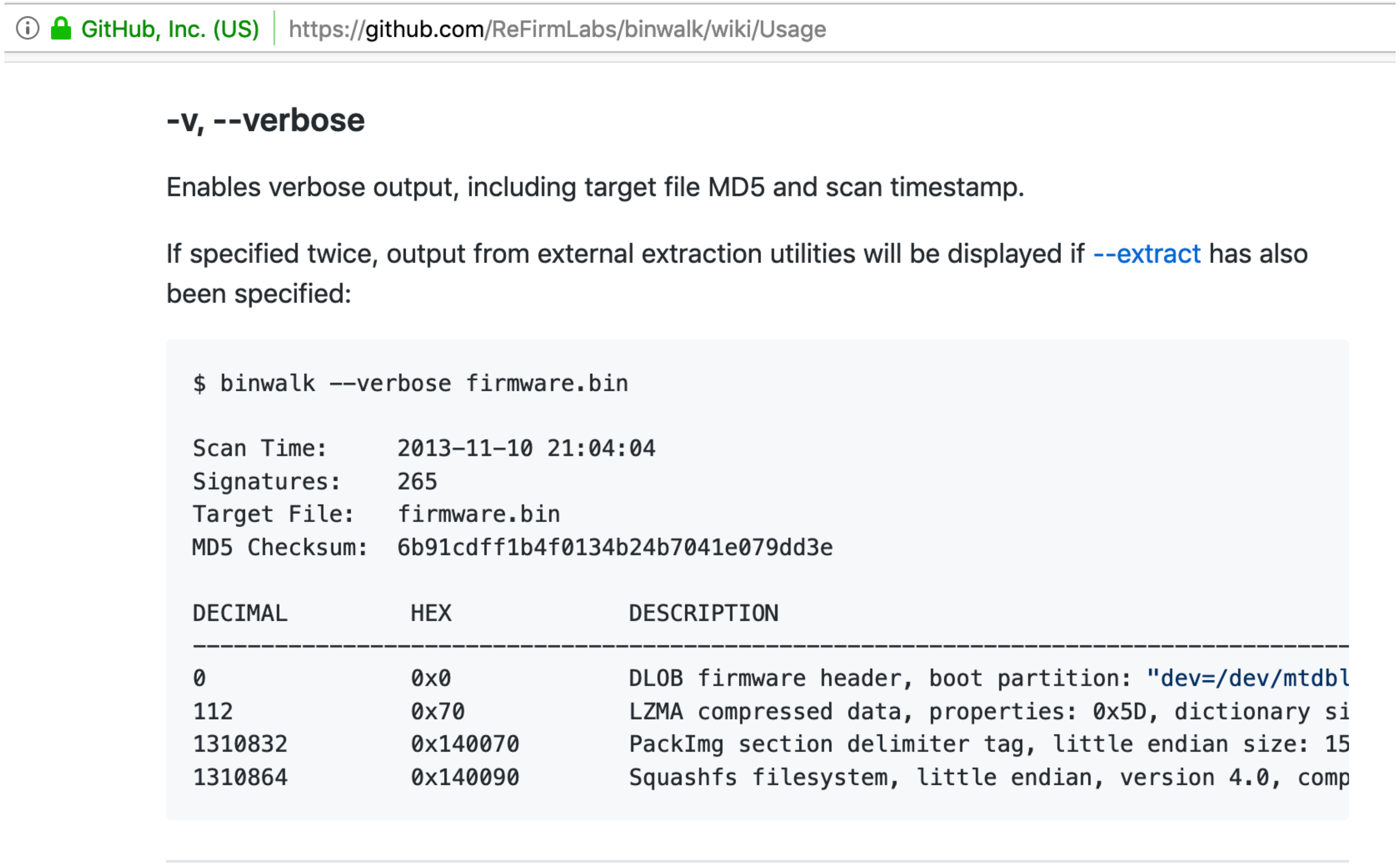
Identifying Debug Ports



```
MX25L12805D(c2 2018c220) (16384 Kbytes)
mtd .name = raspi, .size = 0x01000000 (16M) .erasesize = 0x00010000 (64K) .numeraseregions = 0
Creating 12 MTD partitions on "raspi":
0x00000000-0x00050000 : "uboot"
0x00050000-0x00150000 : "Kernel_1"
0x00150000-0x007c0000 : "rootfs_1"
mtd: partition "rootfs_1" set to be root filesystem
mtd: partition "rootfs_data" created automatically, ofs=440000, len=380000
0x00440000-0x007c0000 : "rootfs_data"
0x007c0000-0x008c0000 : "Kernel_2"
0x008c0000-0x00f30000 : "rootfs_2"
0x00fe0000-0x00ff0000 : "nvram"
0x00ff0000-0x01000000 : "User_Factory"
0x00040000-0x00050000 : "Factory"
0x00f30000-0x00fd0000 : "Belkin_settings"
0x00030000-0x00040000 : "Uboot_env"
0x00050000-0x007c0000 : "Firmware_1"
0x007c0000-0x00f30000 : "Firmware_2"
NET: Registered protocol family 2
```

Bootloader debug messages: Complete memory map for the external flash ROM.

Firmware Analysis



GitHub, Inc. (US) https://github.com/ReFirmLabs/binwalk/wiki/Usage

-v, --verbose

Enables verbose output, including target file MD5 and scan timestamp.

If specified twice, output from external extraction utilities will be displayed if --extract has also been specified:

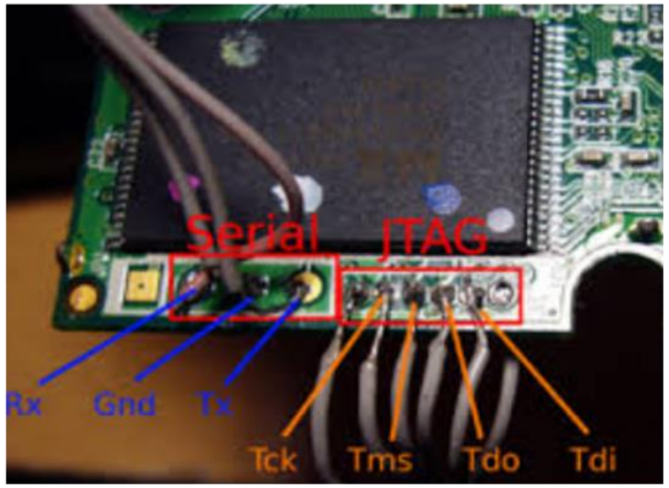
```
$ binwalk --verbose firmware.bin

Scan Time:     2013-11-10 21:04:04
Signatures:    265
Target File:   firmware.bin
MD5 Checksum:  6b91cdff1b4f0134b24b7041e079dd3e

DECIMAL       HEX           DESCRIPTION
--------------------------------------------------------------------------------
0             0x0           DLOB firmware header, boot partition: "dev=/dev/mtdbl
112           0x70          LZMA compressed data, properties: 0x5D, dictionary si
1310832       0x140070      PackImg section delimiter tag, little endian size: 15
1310864       0x140090      Squashfs filesystem, little endian, version 4.0, comp
```

Robotic Vacuum Attack

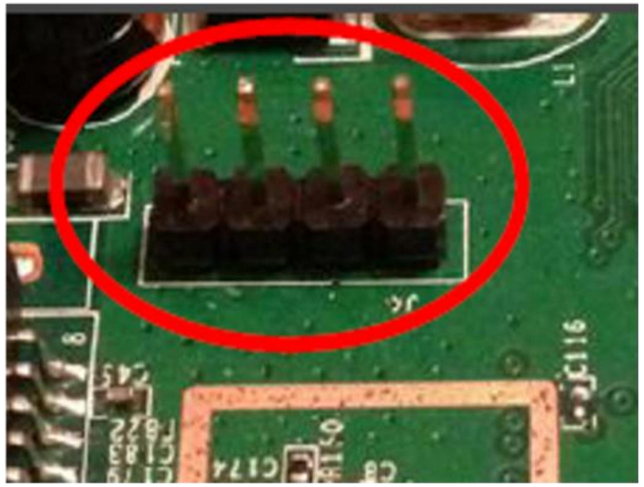
- A real case from Moonbeom Park's IoT security research, reflected all the aforementioned steps while you are doing IoT security research.
- Robotic Vacuum
  - Image capture(Video) using camera
  - Remote control function
  - Voice recoding function

Device Attack Surface

- Acquire the firmware of IoT device
  - Open the manufacture's official firmware download webpage.
  - Setup a hotspot and sniffer the wireless traffic while updating the firmware of IoT device.
  - Open the case and dump the flash memory.
- Locate debug port
  - JTAG Port
  - UART Port

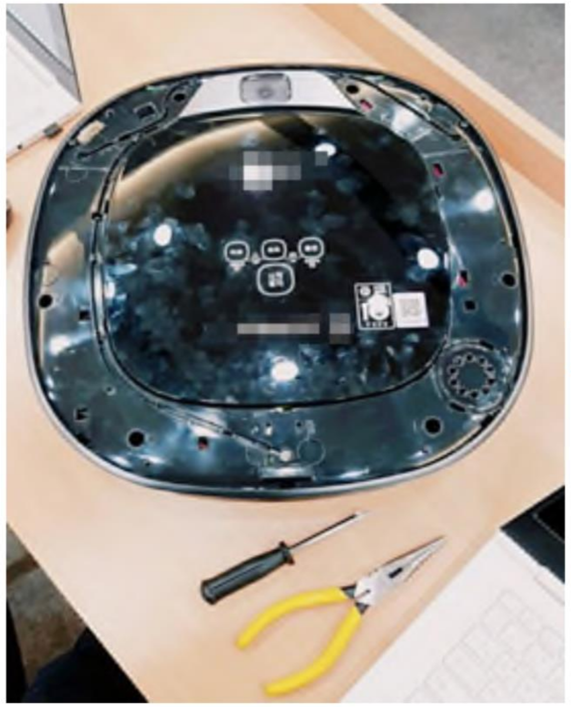


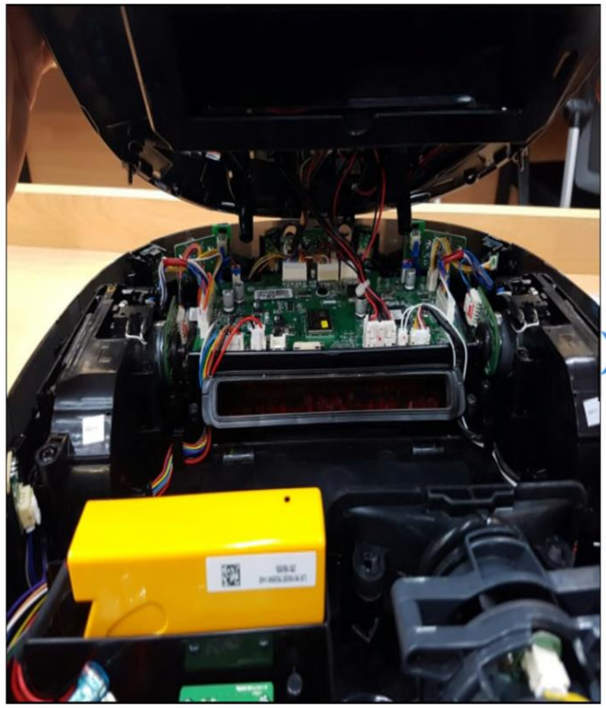
JTAG Port

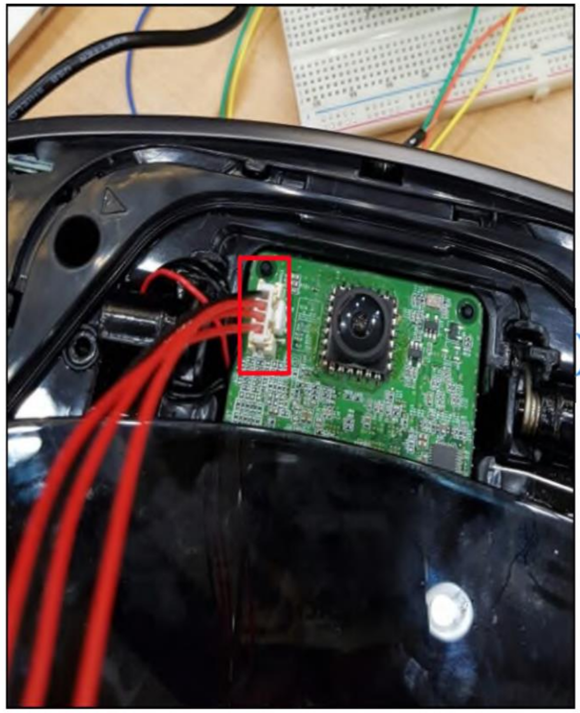


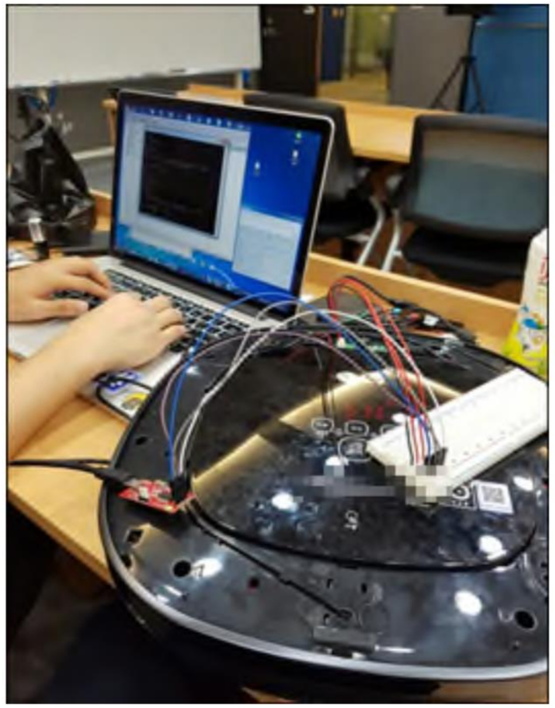
UART Interface

UART Port Connect

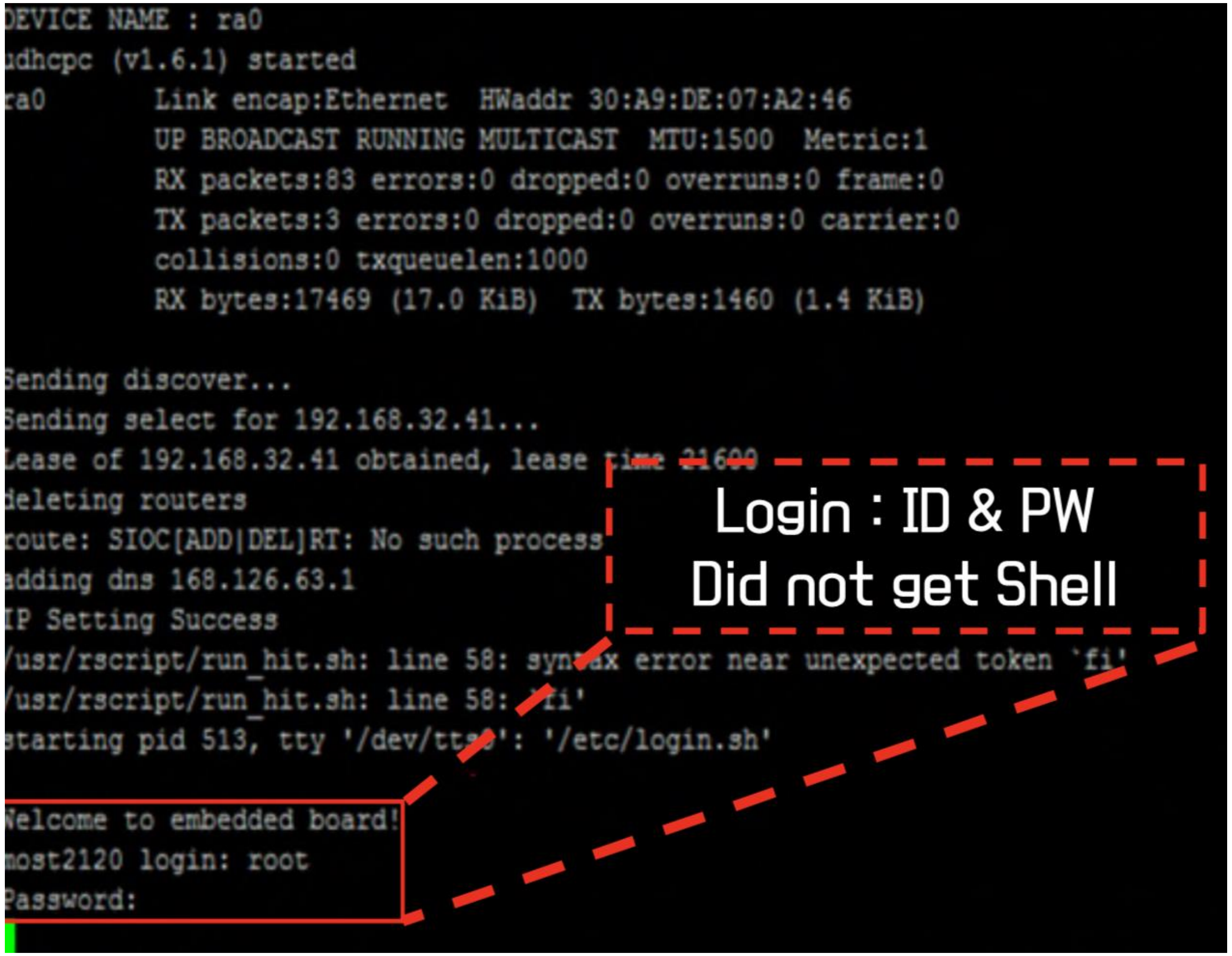

1. Take to pieces


2. Check UART


3. Identify UART pin (Vcc, Tx, Rx, Gnd)

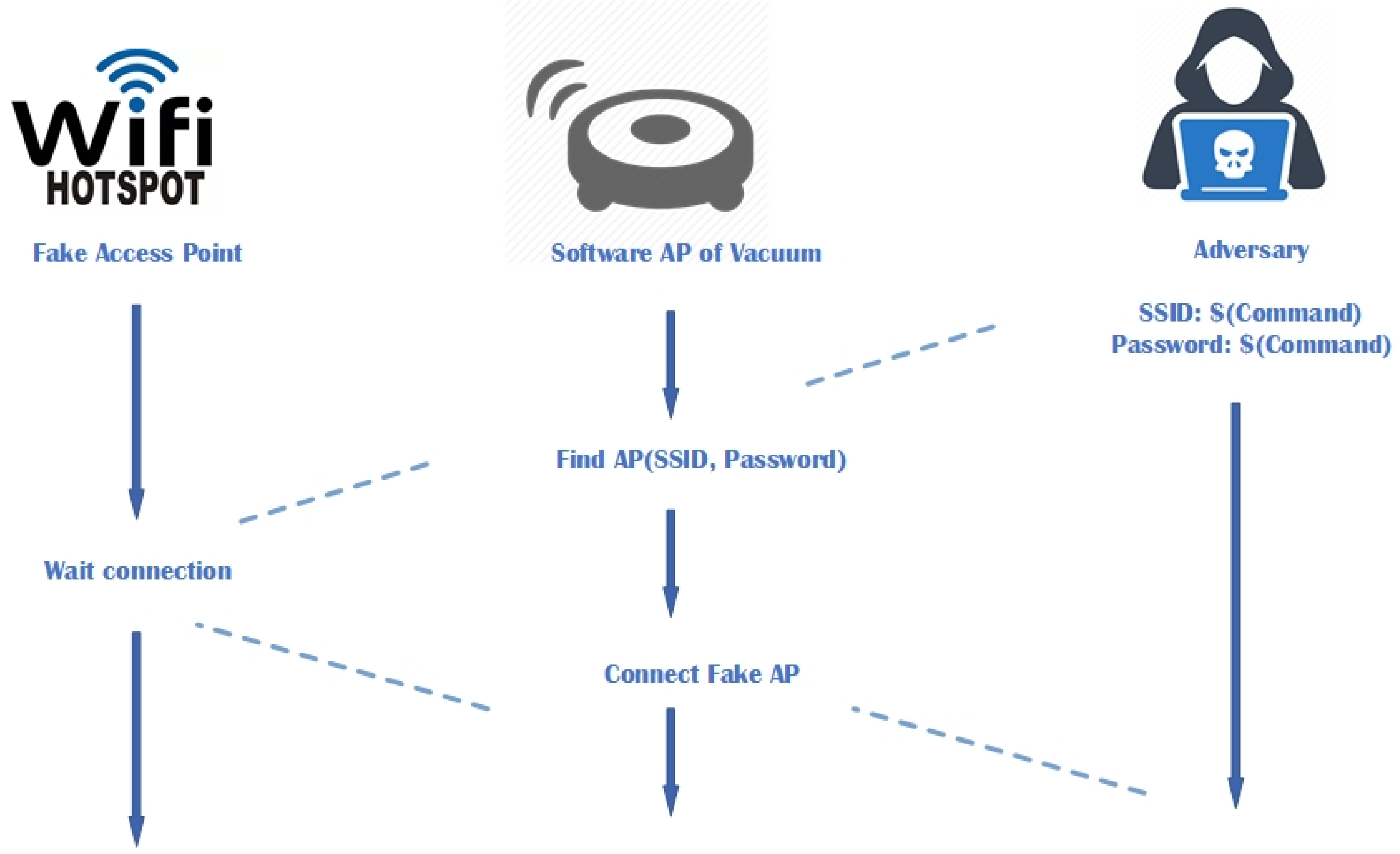

4. Connect UART

UART Port Connect



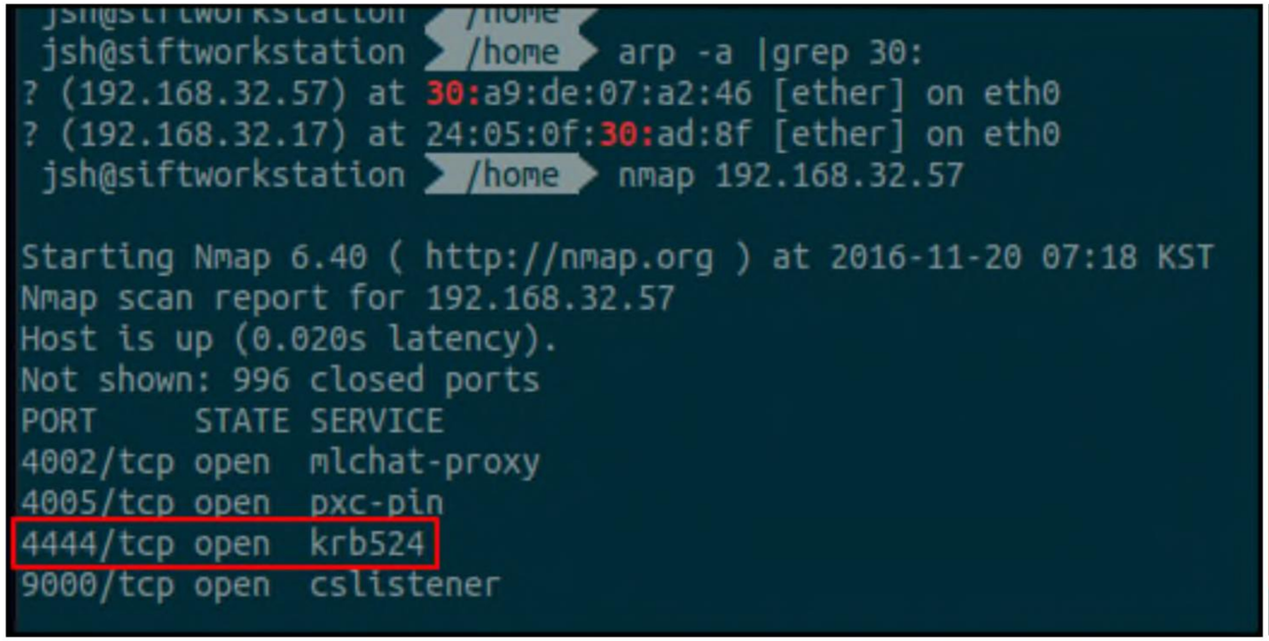
```
DEVICE NAME : ra0
udhcpc (v1.6.1) started
ra0       Link encap:Ethernet  HWaddr 30:A9:DE:07:A2:46
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:83 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17469 (17.0 KiB)  TX bytes:1460 (1.4 KiB)

Sending discover...
Sending select for 192.168.32.41...
Lease of 192.168.32.41 obtained, lease time 21600
deleting routers
route: SIOC[ADD|DEL]RT: No such process
adding dns 168.126.63.1
IP Setting Success
/usr/rscript/run_hit.sh: line 58: syntax error near unexpected token `fi'
/usr/rscript/run_hit.sh: line 58: `fi'
starting pid 513, tty '/dev/tts0': '/etc/login.sh'

Welcome to embedded board!
most2120 login: root
Password:
```

Attack Scenario



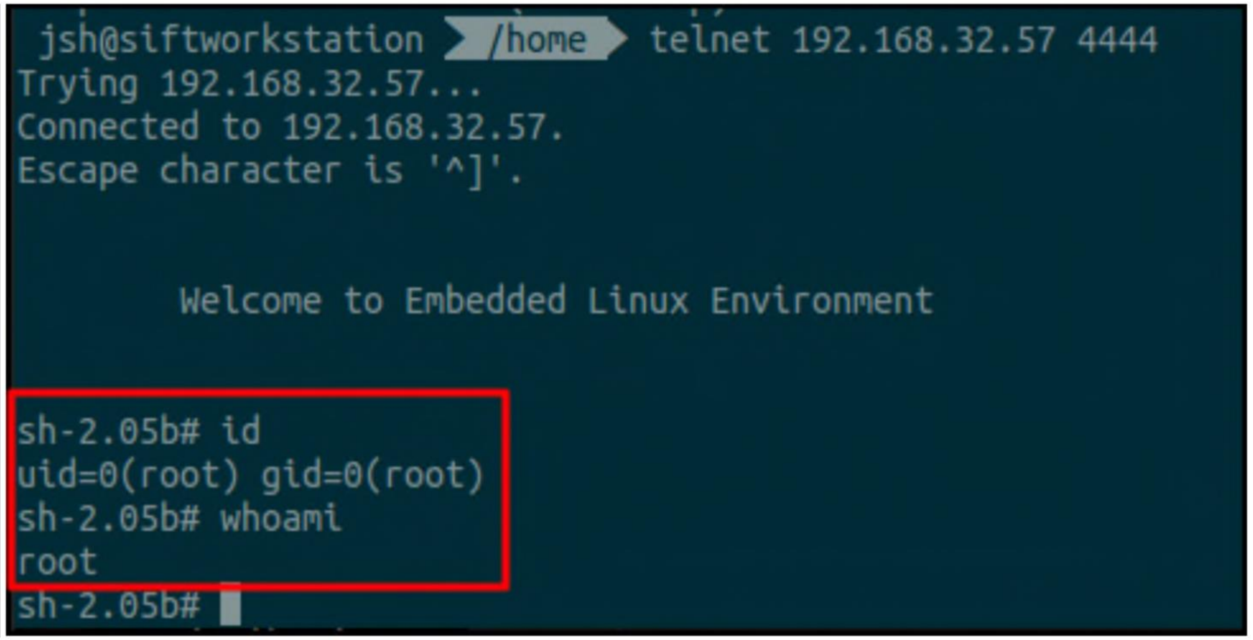
Telnet service started && Remote shell

Get root privileges



```
 jsh@siftworkstation  /home  arp -a |grep 30:
? (192.168.32.57) at 30:a9:de:07:a2:46 [ether] on eth0
? (192.168.32.17) at 24:05:0f:30:ad:8f [ether] on eth0
 jsh@siftworkstation  /home  nmap 192.168.32.57

Starting Nmap 6.40 ( http://nmap.org ) at 2016-11-20 07:18 KST
Nmap scan report for 192.168.32.57
Host is up (0.020s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
4002/tcp open  mlchat-proxy
4005/tcp open  pxc-pin
4444/tcp open  krb524
9000/tcp open  cslistener
```

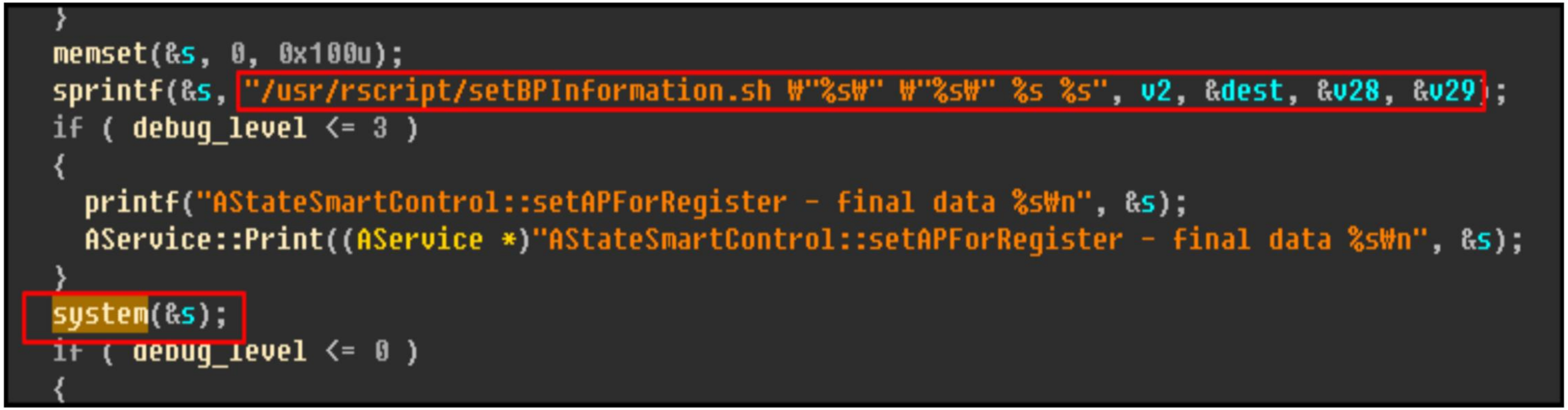
Open Telnet service



```
 jsh@siftworkstation  /home  telnet 192.168.32.57 4444
Trying 192.168.32.57...
Connected to 192.168.32.57.
Escape character is '^]'.


        Welcome to Embedded Linux Environment


sh-2.05b# id
uid=0(root) gid=0(root)
sh-2.05b# whoami
root
sh-2.05b#
```

Get root !!

Analysis to rpmain.axf



```
  }
  memset(&s, 0, 0x100u);
  sprintf(&s, "/usr/rscript/setBPInformation.sh ₩"%s₩" ₩"%s₩" %s %s", v2, &dest, &v28, &v29);
  if ( debug_level <= 3 )
  {
    printf("AStateSmartControl::setAPForRegister - final data %s₩n", &s);
    AService::Print((AService *)"AStateSmartControl::setAPForRegister - final data %s₩n", &s);
  }
  system(&s);
  if ( debug_level <= 0 )
  {
```

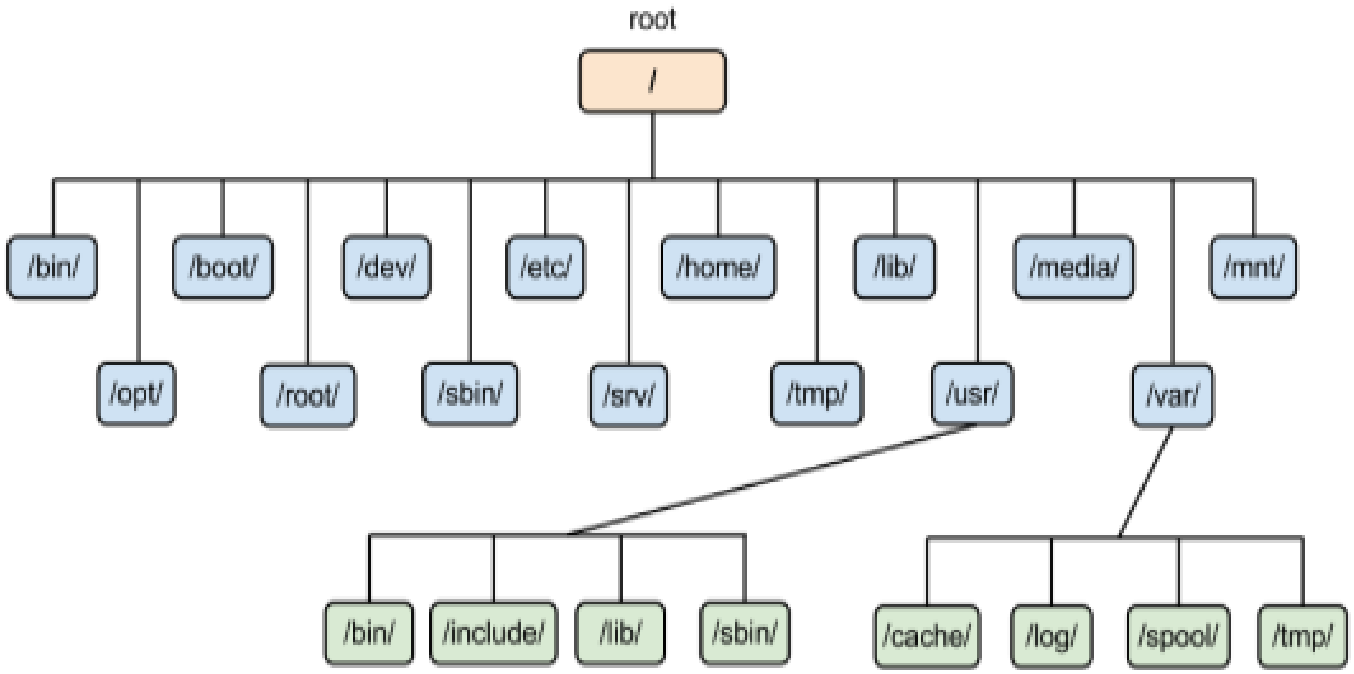
- Binary Patch
- Exploit Proof Of Code
- Searching additional vulnerability

IoT Devices Forensics Research

- Why IoT Forensics?
  - In case of a security incident .
  - IoT device might be the portal of enterprise network.
  - Mostly wireless connected, more attack surface, no physical contact.
- Difference between IoT and Regular Computer Forensics
  - Hardware is different, no complex mother board design.
  - No hard drive, use Flash memory instead, often needs universal programmer.
  - Often use Linux as OS, Windows CE sometimes.
  - Different file system, no ext4, use UBI/Sqaush/JFFS2/Yaffs
  - Firmware often compressed

IoT Forensic

- Linux Forensic & Embedded Linux Forensic
  - Linux commands to collect information.
  - Should know where leave logs.
  - Should know where leave logs.



File system hierarchy of Linux

IoT Forensic

- IoT File System Structure
- Root FS
  - Major Linux directory structure, /bin, /sbin, /etc …
  - Use /tmp to collect system log, often extracted and built from read only memory.
- User FS
  - User data by IoT devices : mount to /usr
  - Can be modified and wrote back to flash so be persisted.

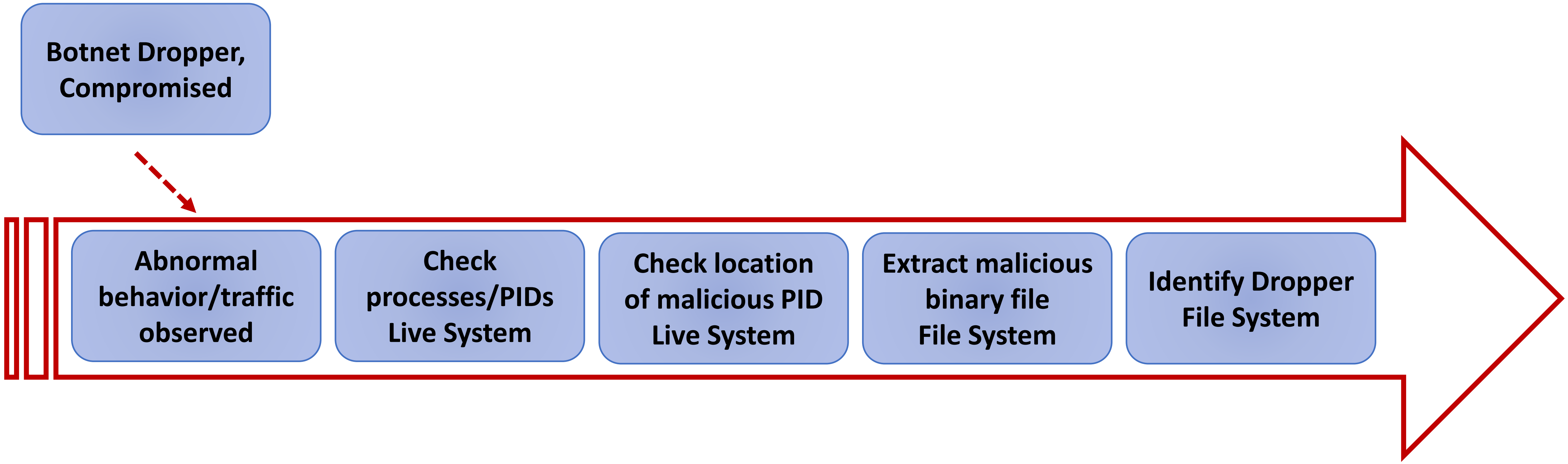
IoT Forensics Plan (Regular)

- Investigate on IoT devices' state
- Collect to artifact of IoT Forensics
  - Extract Flash Memory image for analysis
  - Memory dump by nc and dd
  - Extract firmware by JTAG, Flash Memory
    - Proceed after agreement with being damaged
    - Be careful integrity might be damaged
- Analyze artifact of IoT Forensics
  - Analyze flash memory dump
  - Analyze FS
  - Analyze activity system/logs
  - Analyze File Format
- Preserve evidence and write report

IoT Forensics Plan (Compromised)

- Incident Forensics
  - When/Where/What/How?
- Information gathering for crime scene
  - Check for Device model and H/W spec info
  - Obtain the identified IoT device manual
  - Check firmware version
- Extract firmware
  - Locate malicious file (Does the file still exist after reboot?)

IoT Forensics Plan (Compromised)



Botnet Dropper, Compromised
Abnormal behavior/traffic observed
Check processes/PIDs Live System
Check location of malicious PID Live System
Extract malicious binary file File System
Identify Dropper File System

IoT Forensics Summary

- Existence of manufacturer program accessible to the IoT device? Check for hidden file (ex. Backdoor)
- Dumping Memory Data from IoT device (using UART, JTAG)
- Identify the volume structure and file system for the dump image
- Information gathering about system info (e.g., OS info)
- Collecting Data generated by specific IoT devices

Outline

Introduction

Attacks against IoT system (hardware and software)

IoT reverse engineering and forensics

Attacks against IoT network protocols

Attacks against Industry IoT (IIoT)

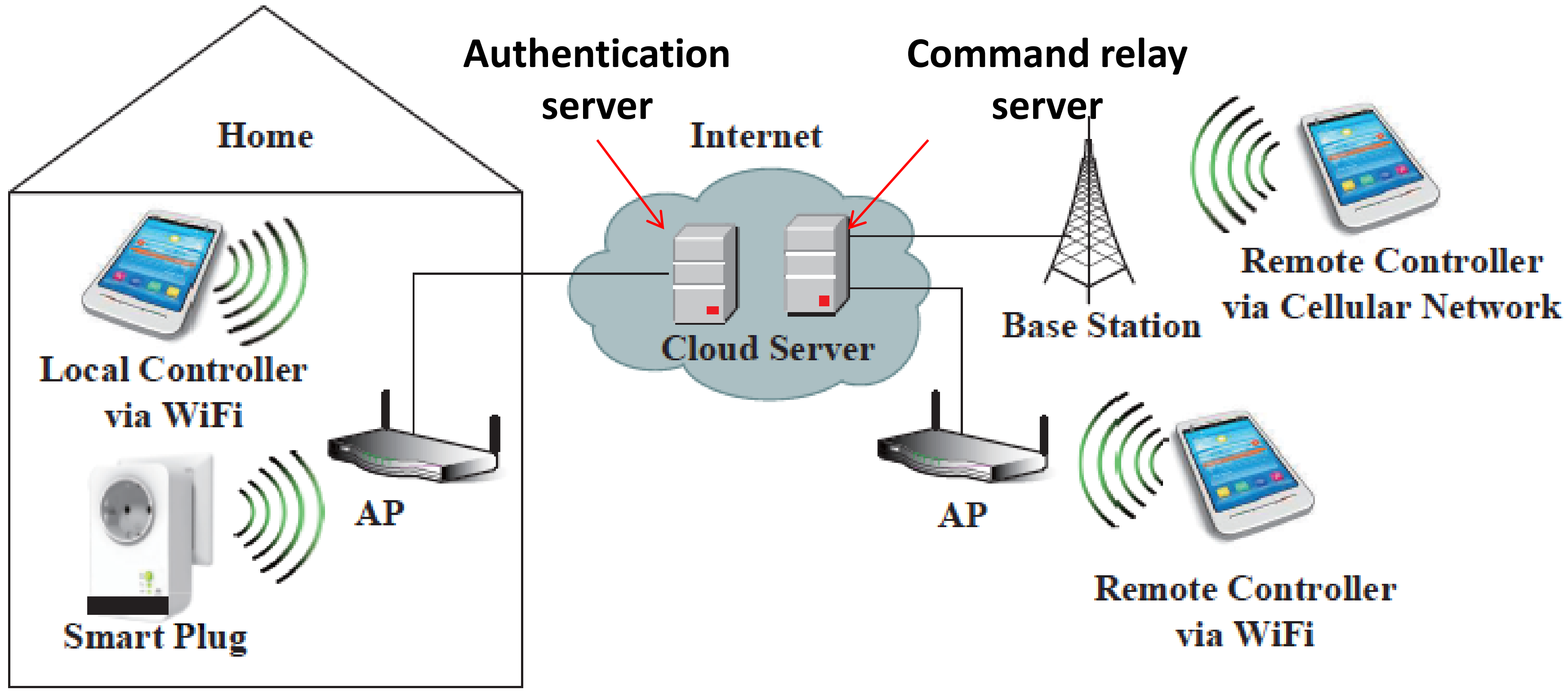
Issues

Weak authentication protocols

Unsanitized user input

Various programming bugs

Edimax Smart Plug



Insecure Communication Protocols

No cryptographic mechanisms for the communication protocols

- No encryption
- Obfuscation based on a bit shifting strategy

Reverse engineering attack

- Communication protocol details

Traffic analysis attacks

- Password, user name if the traffic can be monitored

Device Scanning Attack

Password based user authentication

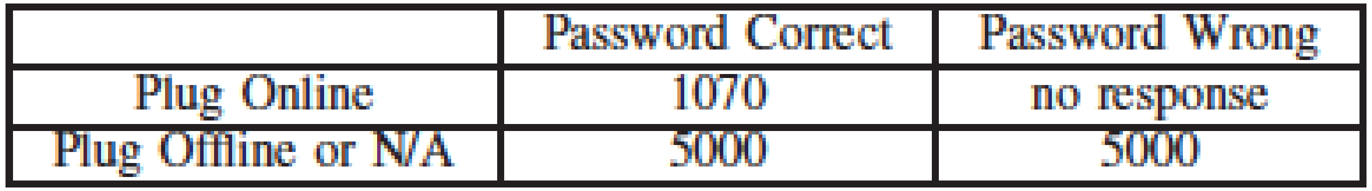
- User name: MAC address
- Password: default “1234”

Scanning the vendor’s MAC address space

- Find the online status of all smart plugs
- Reveal the use of default password
- Many users do not change default passwords!!!

Brute force attack against non-default passwords

- No intrusion detection



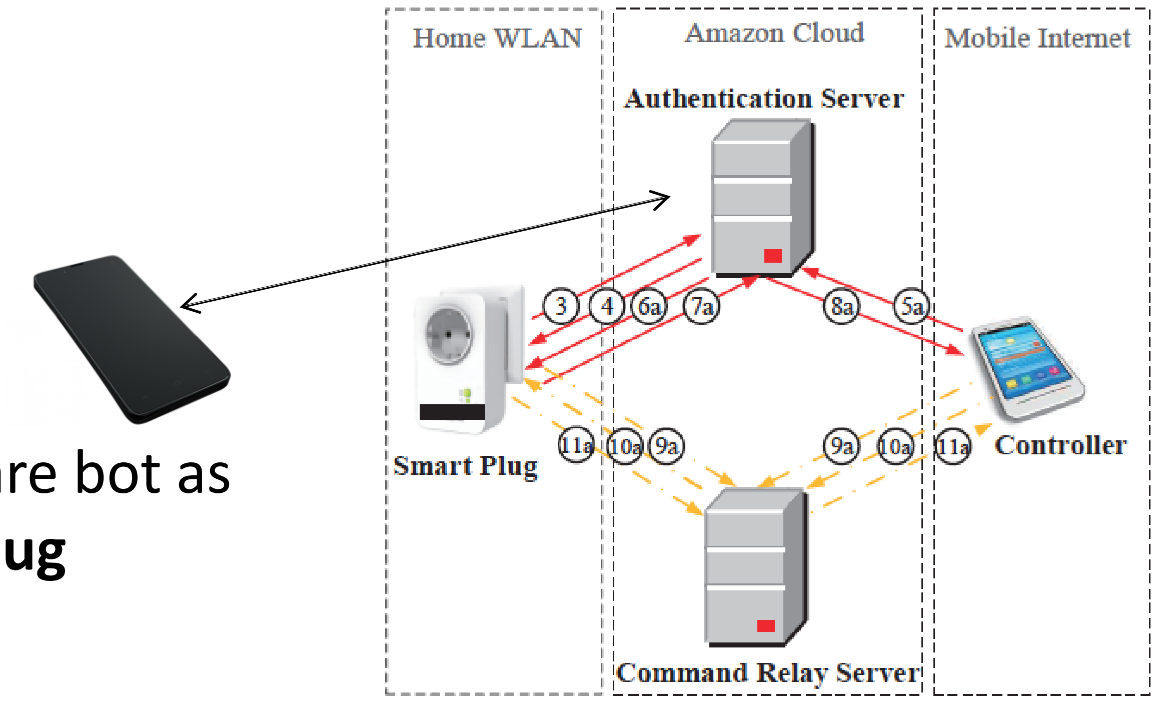
	Password Correct	Password Wrong
Plug Online	1070	no response
Plug Offline or N/A	5000	5000

Device Spoofing Attack

A fake plug (program) registers itself with the cloud

- The real device is pushed offline temporarily

Credentials leak once users open the app

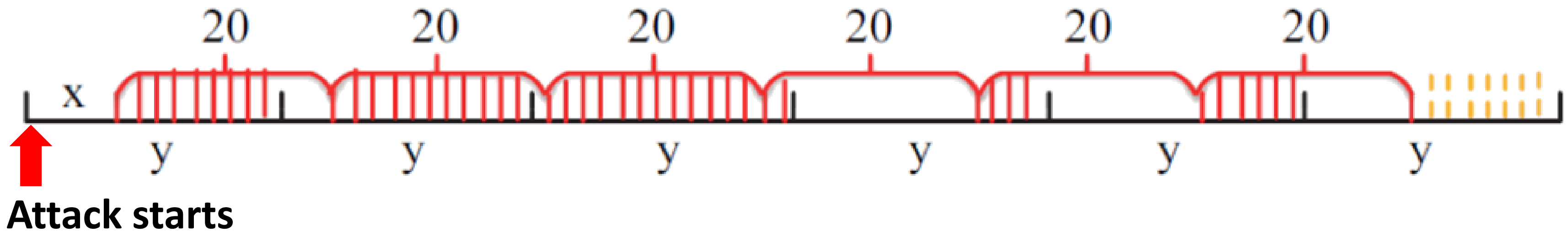


Software bot as **fake plug**

Success Rate of Device Spoofing Attack

Keep-alive messages from real plug every 20 minutes

Keep-alive messages from fake plug every *y* minutes

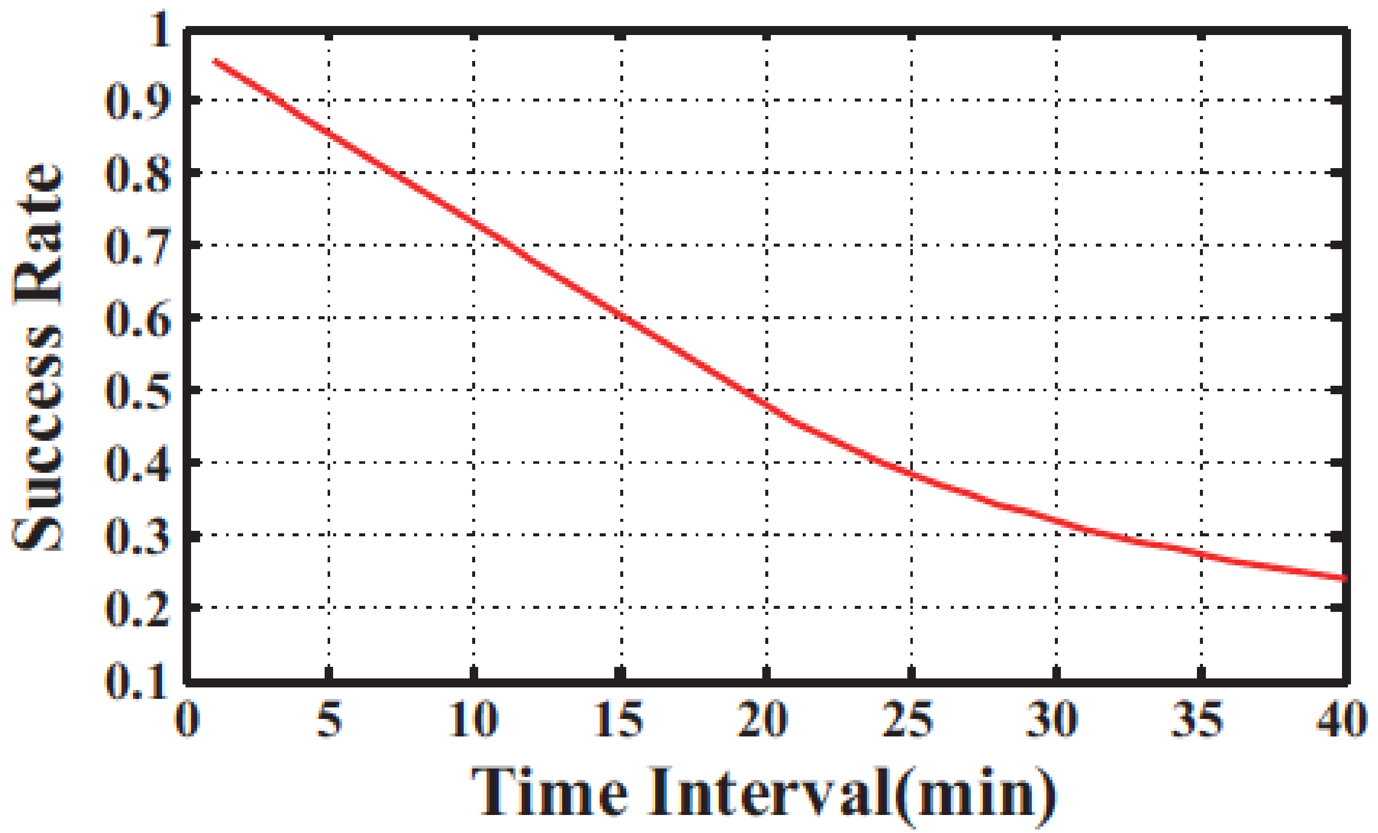


Success rate $S = 1 - \int_{x=0}^{20} \frac{\sum_{i=0}^{n-1} T(i) + T(n)}{20\lceil (x+20n)/y \rceil y} dx$, n>1

- x: the time between first attack packet and real plug's first keep-alive message after the attack starts
- T(i): the period in which real plug is active in ith 20 minutes

Evaluation of Device Spoofing Attack

Success rate vs fake registration packet time interval (y)



Local Firmware Attack

Open port for firmware update in local networks

No integrity checking and authentication for firmware upgrading/downgrading

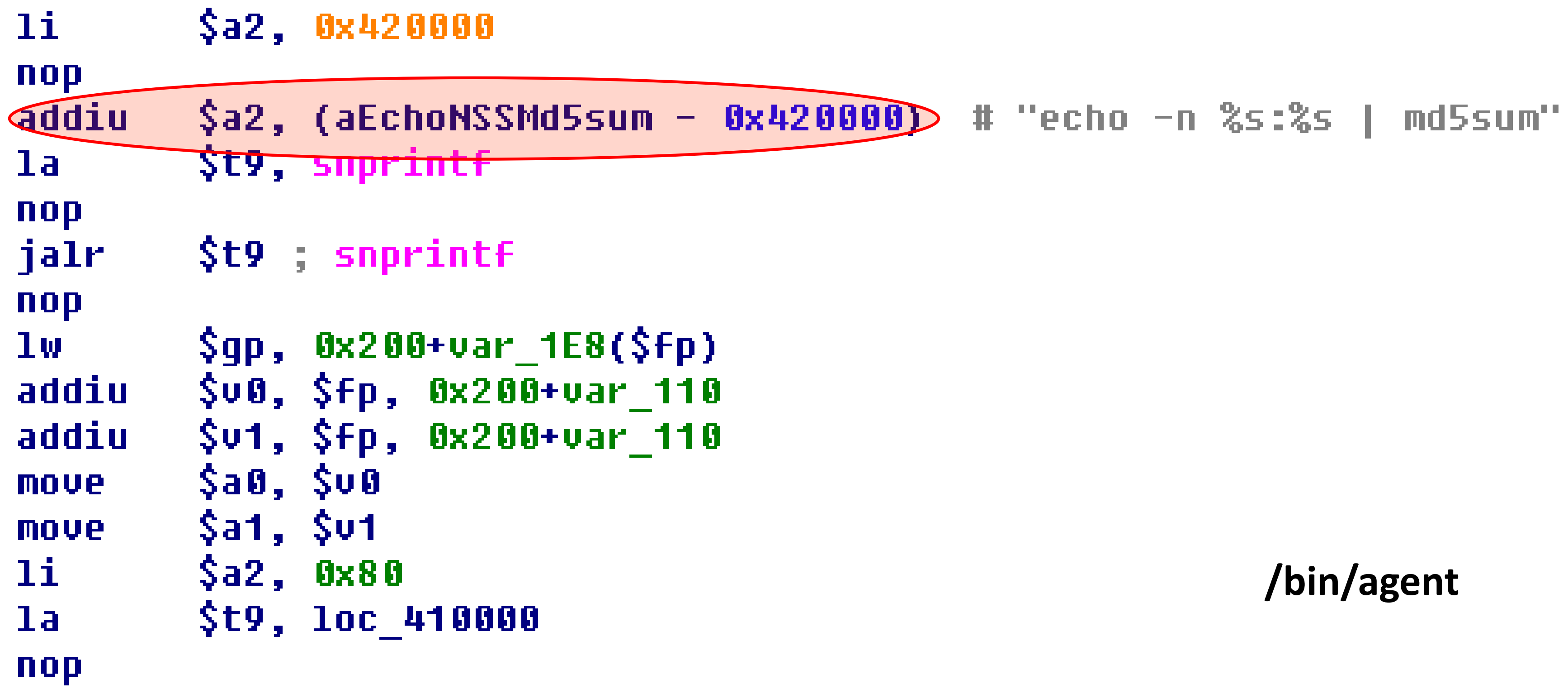
Installation of any malicious firmware

- Reverse tunnel back to the attacker
- Reverse root shell can be opened
- Full control of the plug OS

Remote Command Injection Attack

Vulnerability in password update

- Calls a local md5 hash command to directly work on the user provided password with no sanitization



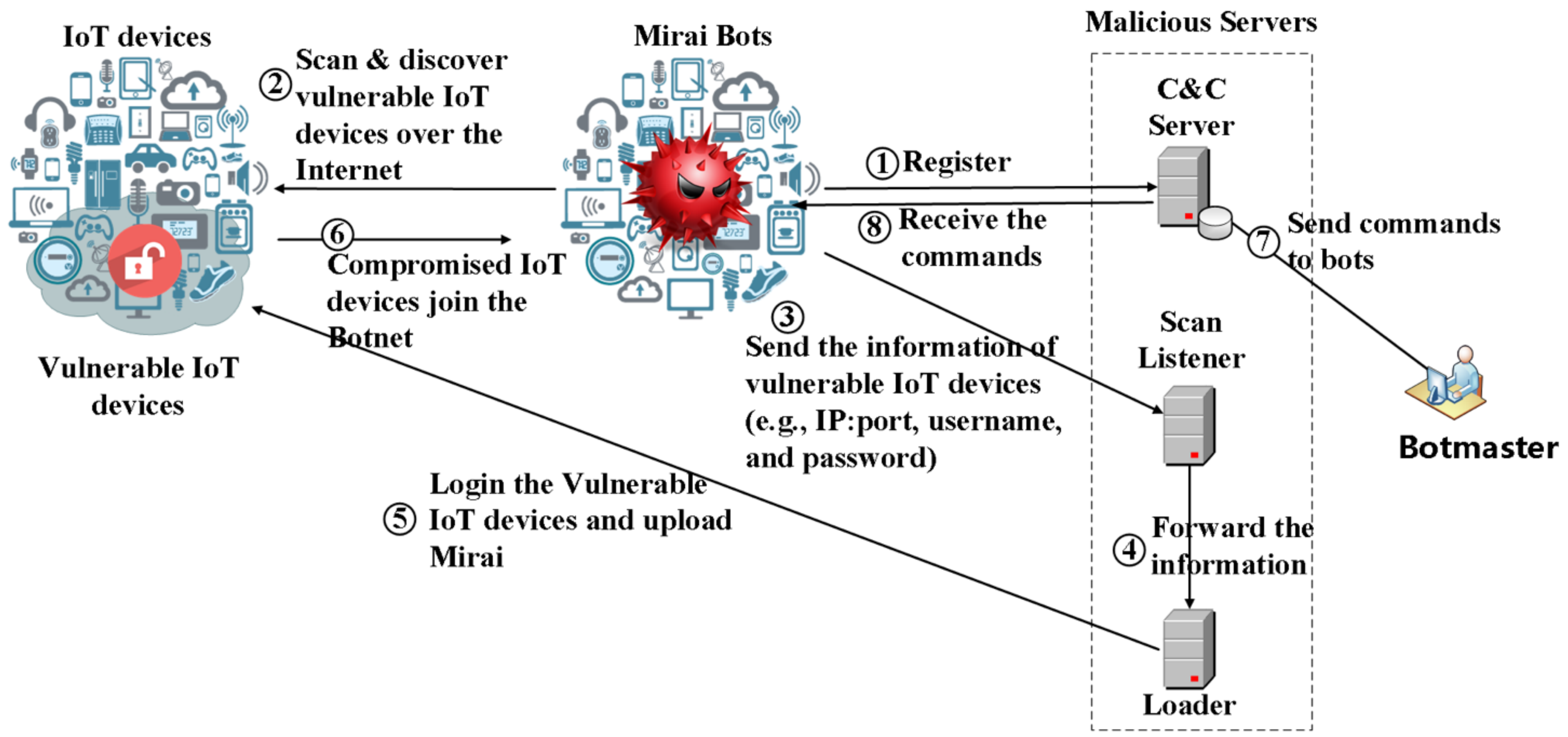
```
li      $a2, 0x420000
nop
addiu   $a2, (aEchoNSSMd5sum - 0x420000)  # "echo -n %s:%s | md5sum"
la      $t9, snprintf
nop
jalr    $t9 ; snprintf
nop
lw      $gp, 0x200+var_1E8($fp)
addiu   $v0, $fp, 0x200+var_110
addiu   $v1, $fp, 0x200+var_110
move    $a0, $v0
move    $a1, $v1
li      $a2, 0x80
la      $t9, loc_410000
nop
```

/bin/agent

Mirai Botnet over IoT

Device spoofing attack + remote command inject attack

- A new wave of Mirai DDoS!!!


IoT devices
Vulnerable IoT devices
Mirai Bots
Malicious Servers
C&C Server
Scan Listener
Loader
Botmaster
② Scan & discover vulnerable IoT devices over the Internet
⑥ Compromised IoT devices join the Botnet
① Register
⑧ Receive the commands
⑦ Send commands to bots
③ Send the information of vulnerable IoT devices (e.g., IP:port, username, and password)
④ Forward the information
⑤ Login the Vulnerable IoT devices and upload Mirai

Outline

Introduction

Attacks against IoT system (hardware and software)

IoT reverse engineering and forensics

Attacks against IoT network protocols

Attacks against Industry IoT (IIoT)

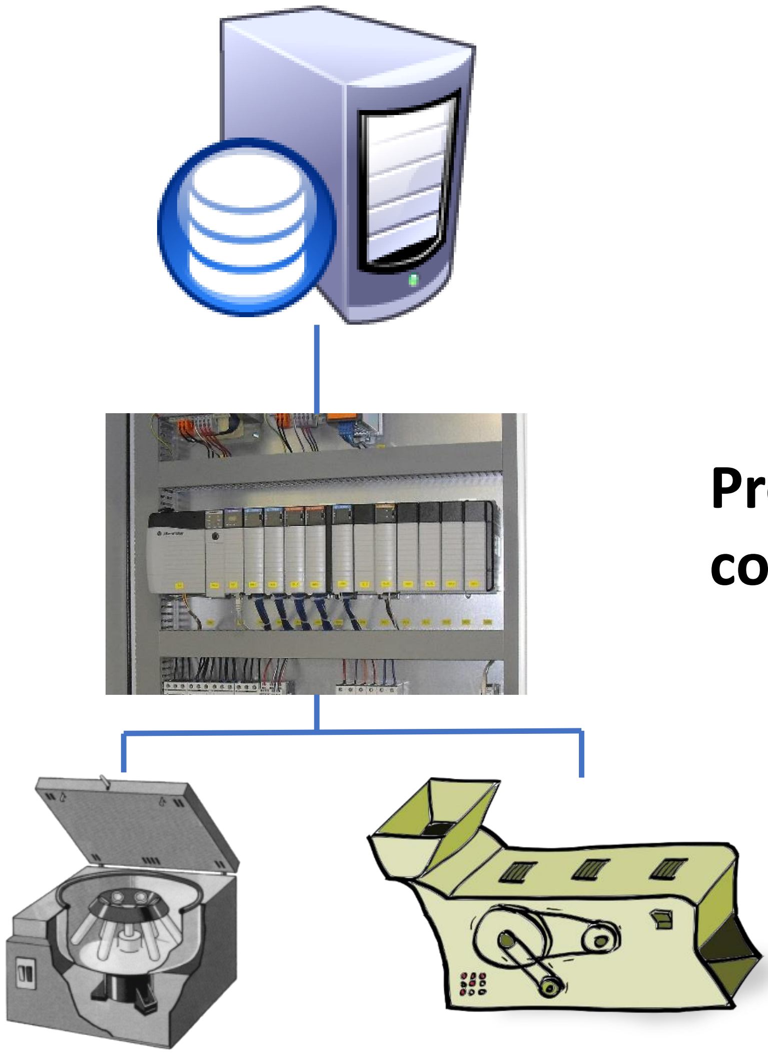
Issues

If any device is connected to the Internet, they are in danger

Same thing goes to any industry machinery

Industry IoT (IIoT)

Supervisory control and data acquisition (SCADA)



Programmable logic controller (PLC)

Stuxnet

Stuxnet was computer worm

It targets SCADA systems and caused serious damage to Iran's nuclear program.

It utilized four zero-day flaws, targeted Microsoft Windows operating system and networks

- Locating Siemens Step7 software, “The programming software for the controller families S7-300, S7-400, C7 and WinAC”

Propagation

introduced to the target networked system via an infected USB flash drive

Propagates across the network, searching for Siemens Step7 software - industrial control systems - on computers

Infects PLC and Step7 software with rootkits

Challenges of IoT

Fully interoperate a large number of heterogeneous interconnected devices, which may

- Have low computation and energy capacity
- Require resource efficient solutions
- Require scalable solutions

Integrates high degree of smartness with adaptation and autonomous-ness

Guarantee trustworthiness with security and privacy

References

[1] Zhen Ling, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, Xinwen Fu, "Security Vulnerabilities of Internet of Things: A Case Study of the Smart Plug System", accepted to appear in IEEE Internet of Things Journal (IoT-J), 2017.

[2] Stuxnet, 2018