UNIVERSITY OF AMSTERDAM

MSc System and Networking Engineering

Research Project 1

Smart TV Hacking

January 2013

Nikos Sidiropoulos Periklis Stefopoulos

Abstract

Modern TV technology demands a far more advanced and typical entertainment system. The idea behind the “Smart TV” concept is the ability of this system to interact with the Internet. However, the more “Smart” the TV is, the more network related services it provides, and the more likely it is to be vulnerable to remote attacks. Given the above statement, we focused on revealing these vulnerabilities and subsequently abusing them. Specifically, we managed to perform an online firmware upgrade by impersonating Samsung’s update servers. Additionally, we discovered that the browser’s TLS/SSL implementation was bad leading to a successful man in the middle attack. Finally, we document all the other design flaws that were found, in terms of security and formulate corresponding threat scenarios.

Acknowledgements

We would like to thank Henri Hambartsumyan, Daan Muller and Coen Steenbeek for their support and valuable assistance throughout the research project.

Nikos and Periklis

Table of Contents

List Of Figures

1 Introduction

Smart TVs are considered as a mandatory entertainment system tailored to the average user which can be placed to a typical home/office network. These systems have several functionalities:

- Content delivery like photos, movies and music from other computers or network attached storage devices
- Network interfaces (Ethernet & Wi-Fi).
- Usb interfaces for keyboard/mouse/storage devices.
- Access to Internet-based services like video and audio streaming, as well as social apps.
- Web browsing with usb keyboard/mouse support.
- Remote control apps for smartphones, tablets and PCs.
- Video/audio interaction for the high end models.

This kind of entertainment system demands to have a proper operating system (stripped down Linux based OS). It is equipped with a CPU (ARM based), RAM, Flash-storage and EEPROM. Similarly with smartphones and tablets, most of the Smart TVs can be updated via Internet or USB. As most of the latest Smartphone OSs demand an application repository, Smart TVs are no exception in this rule. Every manufacturer has created its own "app store" where the users can install, uninstall and update their apps according to their needs. Given that some of the apps deal with private information, vendors are expected to create devices that are secure enough in order to prevent unauthorized access to the TV and block active and passive attacks from hackers.

- This unauthorized access permits the TV acting in different modes:
  - ✓ As a Trojan horse to the home network.
  - ✓ As a stealth surveillance camera/mic service.
  - ✓ As a botnet.
- Passive attacks permit eavesdropping of sensitive data.
- Active attacks permit impersonation (MiTM attack).

Based on the fact that Smart TVs are rather popular in homes, offices, educational institutes and public areas (cafes), our purpose is to investigate the level of security of a Smart TV and assess the security risks. Specifically, our model of reference will be a Samsung TV, due to its large market share (almost 25%)[1]

A Smart TV is considered unsecure due to the following reasons: Firstly, there is no shell access provided to the user, which makes the identification of an attack almost impossible. Secondly, there is no antivirus created for Smart TVs and - even if existed - it wouldn't make the product more secure. In any case, it is proposed that vendors should provide secured devices without any intervention from a third party. Hence, the following question and sub-questions arise:

"What kinds of security countermeasures are implemented in a Samsung Smart TV?"

- Are they enough in order to characterize the Smart TV secure?
- Are there any vulnerabilities? If so, how can we misuse them?

Our approach is based on remote attacks and not on attacks requiring physical access to the device. A remote attack is far more challenging as a method, since it can be applied both in a single and in a multi-attack scenario.

The structure of this paper is as follows: At first, the Materials and Methods are stated, followed by the Vulnerability Analysis where the major (Firmware Attack-Browser Attack) and minor attacks (Miscellaneous Attack) are discussed. Furthermore, there is a short Discussion in which all the design flaws and their consequences are presented. The Conclusion chapter provides answers to the research question while the Future Work section gathers our ideas concerning all further research work which needs to be done.

[1] http://www.koreatimes.co.kr/www/news/biz/2012/10/309_114056.html

2 Materials and Methods

2.1 Materials

Hardware:

- Samsung Smart TV UES5500 Out of the Box
    - ARMv7 microprocessor
    - Firmware 1029
    - Linux based OS (2.6.17 - 2.6.36)
    - Flash Disk : 1 GB
    - Interfaces
        - USB, HDMI, Ethernet, CI,UART
- Linksys WRT54G → DDWRT
- Sony Vaio Laptop → Windows 7
- Samsung Laptop → Linux Ubuntu 12.04
- Samsung Galaxy S
- Samsung Galaxy Note

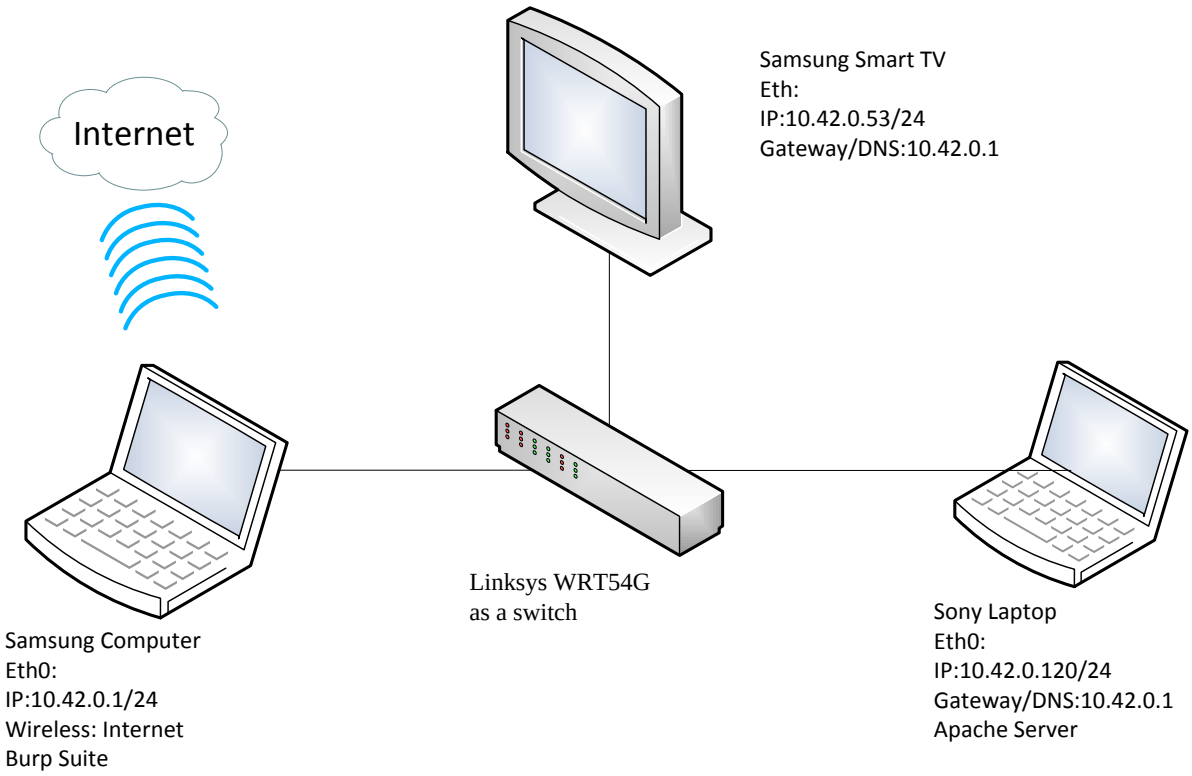
Software:

- Nessus Home Edition
- Burp Suite
- Wireshark
- Zenmap
- Txc-ssl-dos
- Nikto
- DirBuster

2.2 Methods

The following topology diagram was applied for our experiments:

Figure 1: Topology diagram



A straightforward and self-explanatory topology is applied. The Samsung Laptop is connected through Wi-Fi Access to the Internet and serves as a router DHCP and DNS. Linksys WER54G acts as a switch where the Smart TV and the Sony laptop are connected.

3 Vulnerability Analysis

The aim of this research paper is to find new vulnerabilities or expand vulnerabilities that have been already found. The scheme of black box approach was followed. Initially, based on the technical characteristics of the TV, six areas which could possibly reveal vulnerabilities were identified: Firmware, Browser, Samsung Apps, Remote Help Management, AllShare and Remote Control. Moreover, additional tools like, Zenmap and Nessus, were used to further complete our research. More emphasis was given to the first two research areas in which the major weaknesses were revealed.

3.1 Firmware Attack

3.1.1 Firmware Importance

The firmware is the core element of any electronic system comprising of persistent memory, program code and data. In the case of the Smart TV, the firmware is the operating system which is responsible for managing the TV's available hardware resources and providing common services to the programs needed for the TV.

3.1.2 About the Firmware

The running version of the TV's firmware is 1029, while 1030 and 1031 updates are available through Samsung's product support site[2]. Samsung does not provide any kind of change log to the versions it provides through its website. There are three update procedures that can be followed:

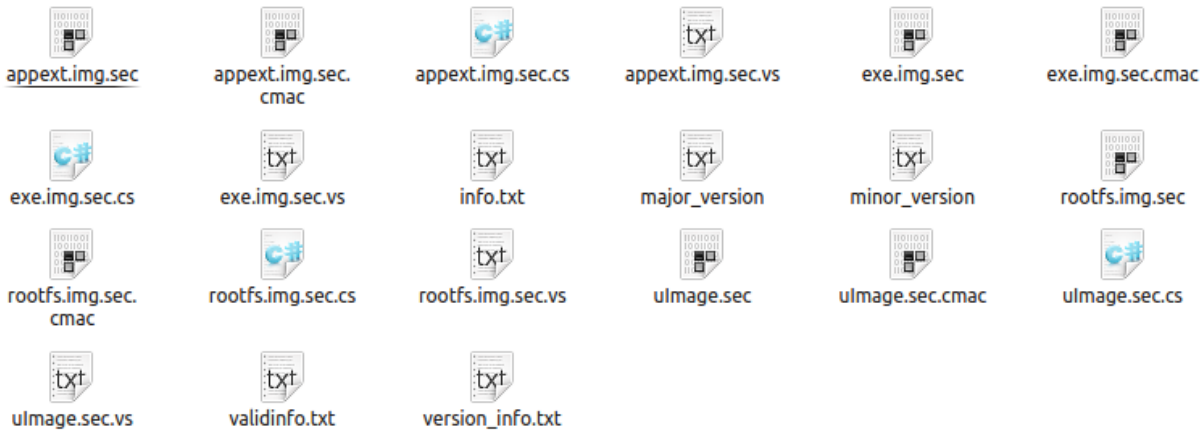
- On line (Internet)
- USB
- Broadcast signal

[2] http://www.samsung.com/uk/support/model/UE32ES5500KXXU-downloads

3.1.3 Vulnerability Assessment

The firmware does not provide any kind of shell access, so from that point of view it seems secure yet unmanageable from the user perspective. But, what if you can modify the firmware, in order to get root access[3], and install it by using any of the aforementioned update procedures? How does Samsung prevent such a thing to happen? All the firmwares available for download and installed via USB are encrypted but not in the best possible way as it turned out. The encryption method (two-layered encryption: AES + XOR) was proven insufficient. A development team from Samygo.tv has created a tool for decrypting/encrypting most of the Samsung TV firmwares[4]. Additionally, for every file in this firmware update package (USB update) there is also another file that contains a cmac signature (fugure 2). This signature guaranties the integrity and authenticity of the firmware. Hence, the TV's USB update procedure requires a firmware that is cryptographically signed by Samsung.

Figure 2 USB firmware files



[3] Root access means full access to all features that the TV provides like file system, usb ports, Ethernet, microphone, camera etc.

[4] http://wiki.samygo.tv/index.php5/Extracting_the_ES-series_firmware

Using the SammyGoFirmwarePacher.py [5] the firmware was decrypted (Figure 3) and the following files were derived.

Figure 3 USB decryption process

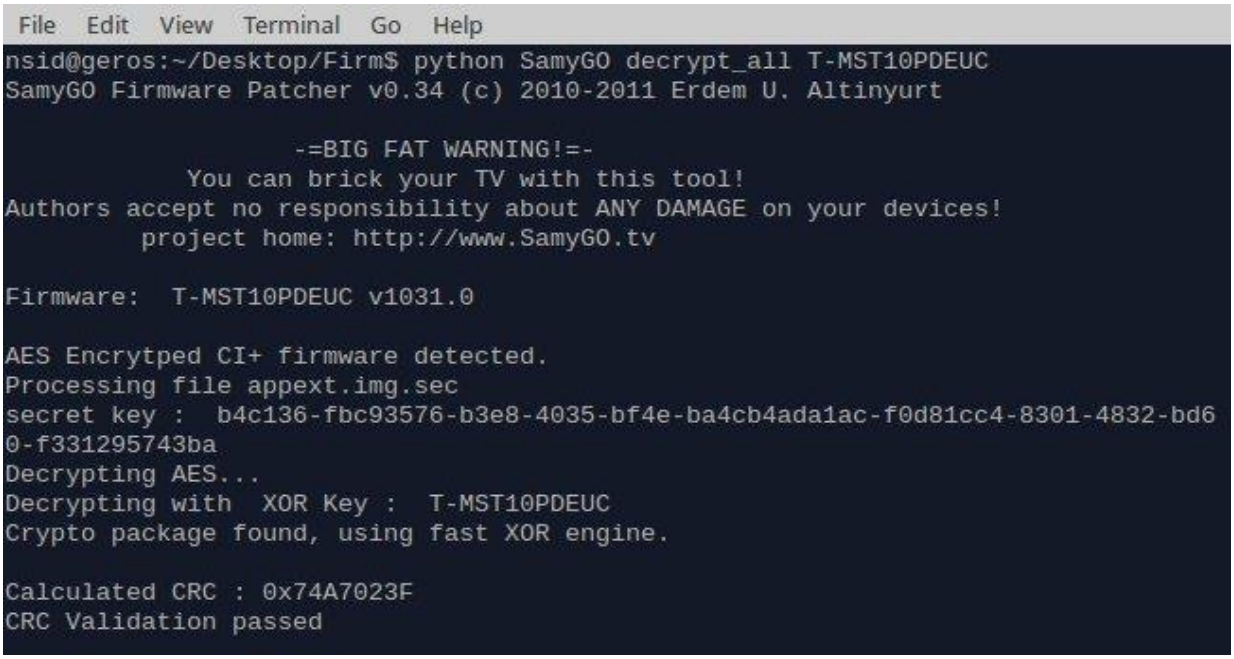
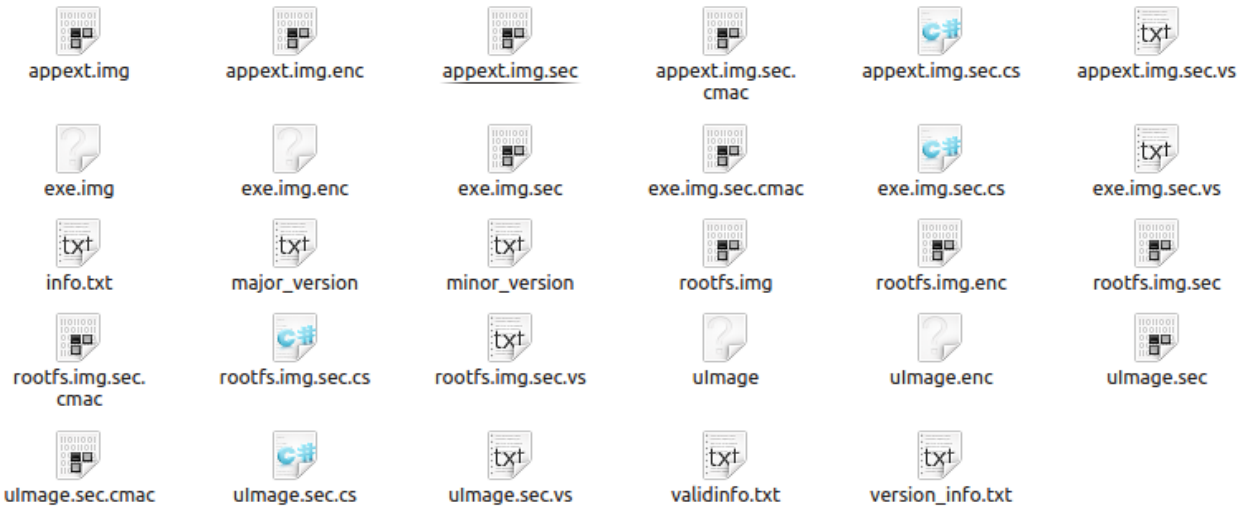


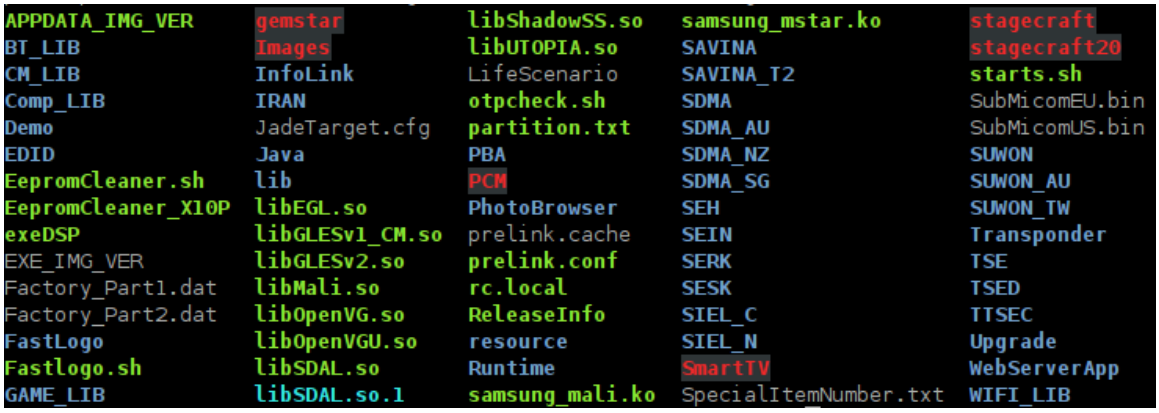
Figure 4 USB decrypted firmare



[5] http://sourceforge.net/p/samygo/code/HEAD/tree/patcher/trunk/SamyGO%20Firmware%20Patcher.py

The uImage is the “VDLinux” based kernel image, while rootfs.img and appext.img are the file system and auxiliary data/programs images respectively. The most important image is the exe.img because it contains the exeDSP which is the main executable for running the digital signal processor (DSP). The next step was to mount those images and inspect them thoroughly. Exe.img contains rc.local which defines the booting order of the system as well as the different types of libraries being used. The folder WIFI_LIB contains the drivers supported for Wifi hardware as well as the iperf tool. Moreover, the Java folder has all the supported java libraries. In the following figure (Figure 5) the whole exe.img file structure is depicted.

Figure 5 exe.img files

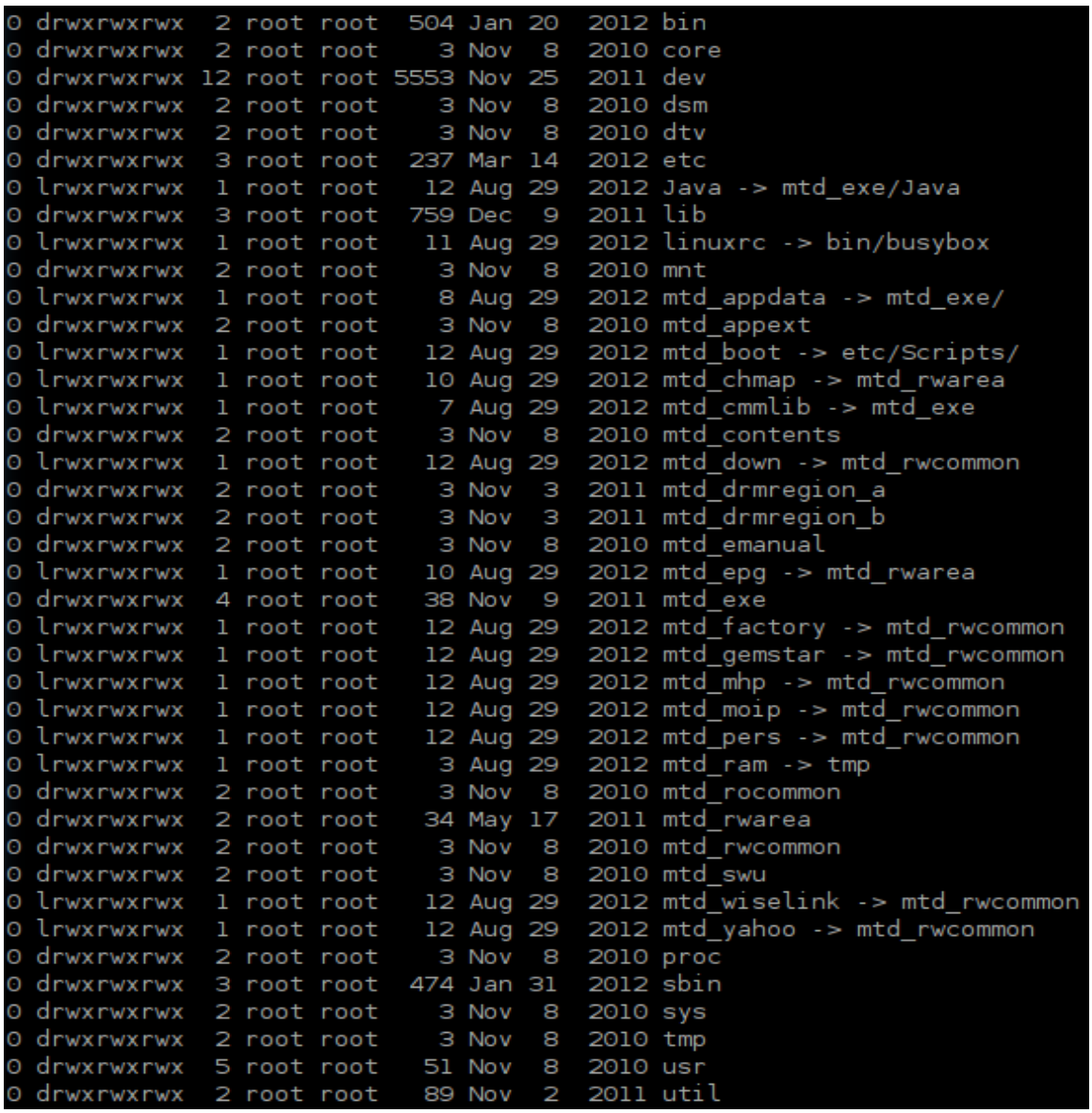


By mounting rootfs.img it was possible to view the whole filesystem structure (Figure 7). Most of the folders are symbolic links that point the common read/write area (mtd_rwcommon) which is the “sandbox” for the apps. By exploring “/etc/” we can find rc.local which reveals all the mountpoints.

Figure 6 /etc/rc.local

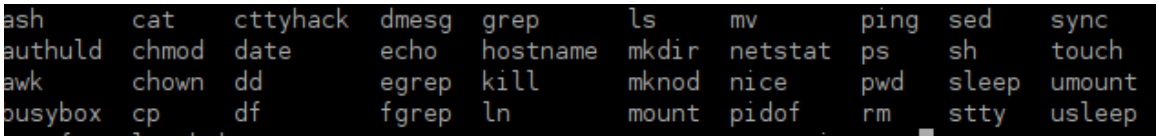
```
echo "================================================================="
echo "  ROOTFS VERSION : "$ROOTFS_VERSION
echo "================================================================="

# mount ramdisk
mount -n -t proc proc /proc
mount -n -t sysfs sysfs /sys
mount -t tmpfs tmpfs /dev/shm
mount -t tmpfs tmpfs /dtv -o size=40M,mode=1777
mount -t tmpfs tmpfs /tmp -o size=36M,mode=1777
mount -t tmpfs tmpfs /dsm -o size=12M,mode=1777
mount -t tmpfs tmpfs /core -o size=30M,mode=1777
```

Figure 7 Mounted rootfs.img



By investigating the "/bin" folder enabled us to understand what types of commands are available. All these commands are linked(symbolic) to busybox.

Figure 8 /bin

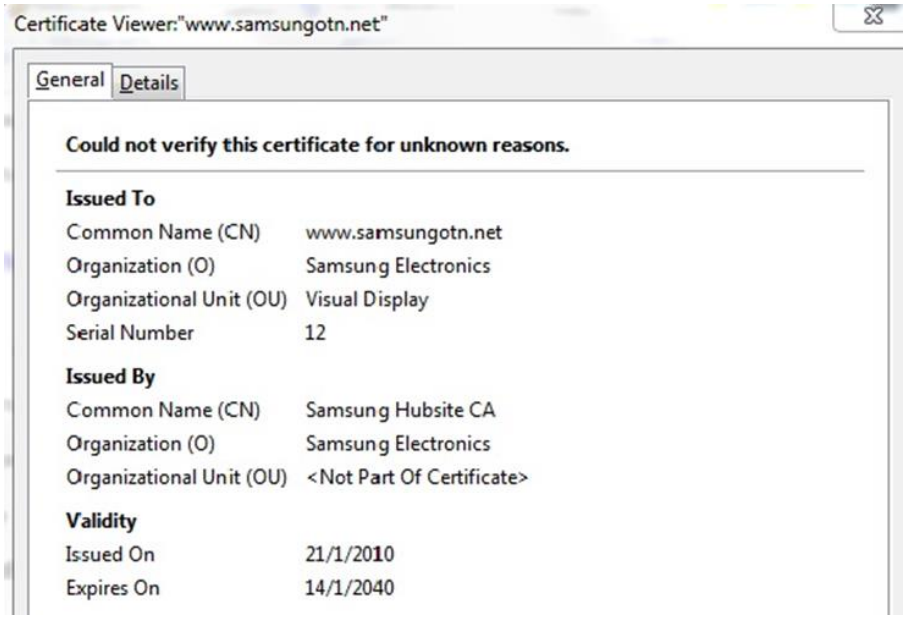


Firmware files downloaded via the online procedure are not signed. So, by decrypting, modifying and then re-encrypting the firmware someone can easily update the TVs firmware with his own customized version. However, creating a customized version is very dangerous, considering the possibility of getting your Samsung TV "bricked". "Brick" is a term used by the hacking community when a device is not operational due to a software update failure. Assuming that the customized Firmware was developed correctly, you can update the TV through the online update procedure.

3.1.4 Attack Procedure

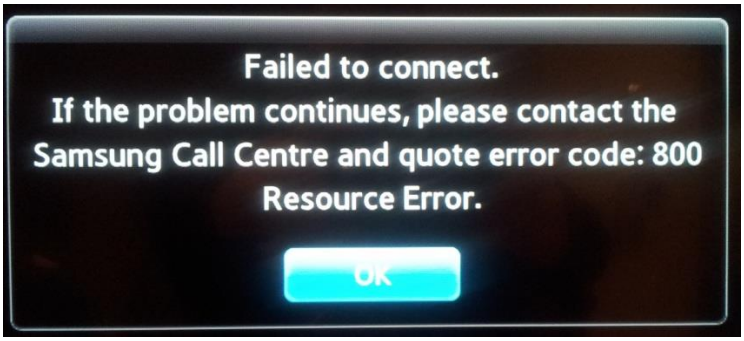
Based on Wireshark, it was found that the online update procedure starts with a TLS/SSL connection with www.samsungotn.net.

Figure 9: samsungotn.net certificate



The certificate is signed with Samsung Hubsite private key. Using Burp Suite, a Man in The Middle attack (MiTM) was performed, yet the TV popped up a network error due to the denial of Burp Suite's self-signed certificate.

Figure 10: Certificate Denial



Looking up closely at the captured communication between the two entities (Samsung TV – Samsung Web Server), it was discovered that after this initial HTTPS connection with www.samsungotn.net, a HTTP (unsecure) connection was established with the same webserver for checking the availability of a new update (Figure5). Afterwards, another HTTP connection was established with "az43064.vo.msecnd.net" to actually download the firmware files (Figure 6).

Figure 11: Unsecure firmware update check

```
▶ Frame 54: 283 bytes on wire (2264 bits), 283 bytes captured (2264 bits) on interface 0
▶ Ethernet II, Src: 1c:5a:3e:e3:f1:4b (1c:5a:3e:e3:f1:4b), Dst: Wistron_6a:26:93 (00:1f:16:6a:26:93)
▶ Internet Protocol Version 4, Src: 10.42.0.53 (10.42.0.53), Dst: 157.55.184.57 (157.55.184.57)
▶ Transmission Control Protocol, Src Port: 43813 (43813), Dst Port: http (80), Seq: 1, Ack: 1, Len: 217
▼ Hypertext Transfer Protocol
  ▶ GET /openapi/tv/T-MST10PDEUC/SWU_T-MST10PDEUC_001029_I04_KK000RK000EK000DK000_121015/m_notice HTTP/1.1\r\n
    Host: www.samsungotn.net\r\n
    Accept: */*\r\n
    DUID: CPCB3EXSMRCXQ\r\n
    If-Modified-Since:  Mon, 28 Jan 2013 10:37:48 GMT\r\n
    \r\n
```

Figure 12: Unsecure firmware download

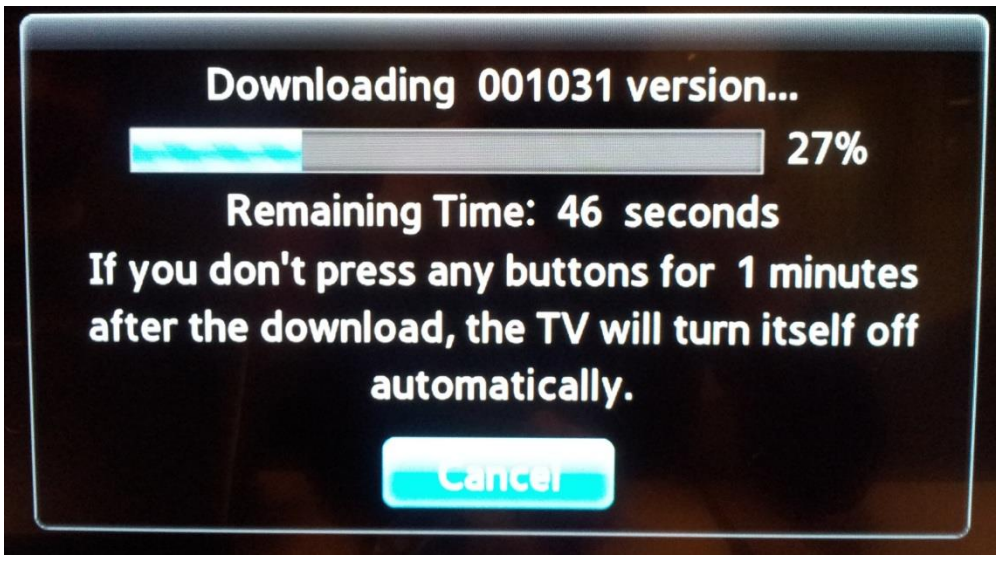
```
▶ Frame 75: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits) on interface 0
▶ Ethernet II, Src: 1c:5a:3e:e3:f1:4b (1c:5a:3e:e3:f1:4b), Dst: Wistron_6a:26:93 (00:1f:16:6a:26:93)
▶ Internet Protocol Version 4, Src: 10.42.0.53 (10.42.0.53), Dst: 65.54.88.173 (65.54.88.173)
▶ Transmission Control Protocol, Src Port: 60201 (60201), Dst Port: http (80), Seq: 1, Ack: 1, Len: 142
▼ Hypertext Transfer Protocol
  ▶ GET /firmware/tv/154/SWU_T-MST10PDEUC_001031_I04_KS000RS000ES000DS000_121129/appext.img HTTP/1.1\r\n
    Host: az43064.vo.msecnd.net\r\n
    Accept: */*\r\n
    \r\n
```

Based in the above facts we added a static DNS entry to the "hosts" file in order to forward the HTTP GET request to a local Web Server, running on the other laptop, instead of Samsung update Web Server (az43064.vo.msecnd.net). Furthermore, by using the download link [6] provided by Wireshark captured file, the first file (appext.img) needed for the upgrade process was downloaded. The only thing missing was to test whether the download procedure would accept the file that was provided by our Apache Web Server. As expected the file was successfully accepted. (Figure 7 – 8)

Figure 13: Local server update

```
Frame 440: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits) on interface 0
Ethernet II, Src: SamsungE_e3:f1:4b (1c:5a:3e:e3:f1:4b), Dst: Sony_22:af:b9 (54:53:ed:22:af:b9)
Internet Protocol Version 4, Src: 10.42.0.53 (10.42.0.53), Dst: 10.42.0.120 (10.42.0.120)
Transmission Control Protocol, Src Port: 38716 (38716), Dst Port: http (80), Seq: 1, Ack: 1, Len: 142
Hypertext Transfer Protocol
  GET /firmware/tv/154/SWU_T-MST10PDEUC_001031_I04_KS000RS000ES000DS000_121129/appext.img HTTP/1.1\r\n
  Host: az43064.vo.msecnd.net\r\n
  Accept: */*\r\n
  \r\n
```

Figure 14: Local firmware download



[6] http://az43064.vo.msecnd.net/firmware

3.2 Browser Attack

3.2.1 Browser Importance

Browsers are the fundamental applications that any "Smart" OS can provide, considering that the terminology Smart derives from the fact that it interacts with the internet. On the other hand, most of the services that web browsers provide (for Smart OSs) have been replaced by the apps in terms of speed and user-friendliness. Samsung Apps (store) is still at an early stage of development and either popular apps are missing or they exist with limited functionality. Hence, most of the interaction with internet has to be done through the browser and with the ease provided by a USB keyboard/mouse.

3.2.2 About the browser

The browser is based on an outdated version of Firefox (version 5, based on user strings) and supports basic extensions like Java, flash and can also manage cookies. Moreover, it supports SSL/TLS over HTTP.

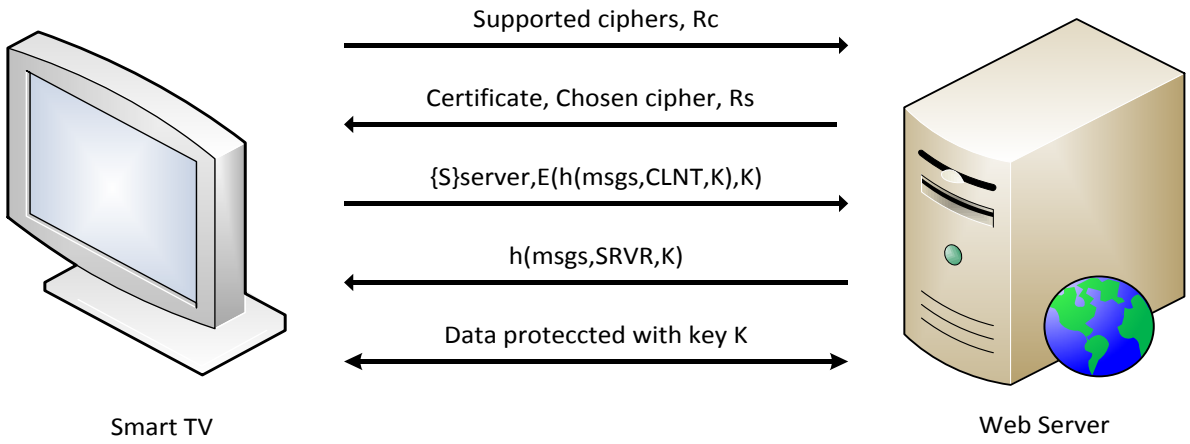
3.2.3 SSL/TLS background

SSL is designed to run on top of layer 4 and provides a reliable encrypted and integrity protected TCP connection to the application. The main operation is described below:

1. The client contacts the server, gives out the list of cryptographic algorithms supported and a random number Rc.
2. The server sends his certificate, plus the highest algorithm that they both support and also a random number Rs.
3. If the client accepts (trusts) the certificate (contains server's public key), the client chooses a random number S and encrypts it by using the public key. Along with the encrypted random number S, the client also sends an encrypted keyed hash, of the master secret K=h(S,Rc,Rs) and the handshake messages.
4. The server proves that it knows the session keys and ensures that the previous messages arrived intact by sending back a keyed hash of all the previous messages plus the master secret K.

5. The connection is established and protected with master key K until the session is terminated.

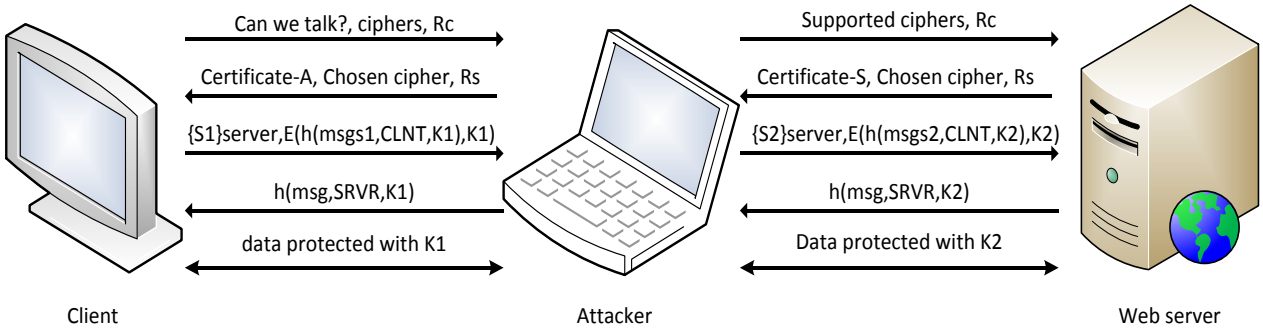
Figure 15: TLS/SSL



3.2.4 Man in the Middle Attack (MiTM)

The most common attack for the SSL is Man in the Middle attack. The method is described in the following figure:

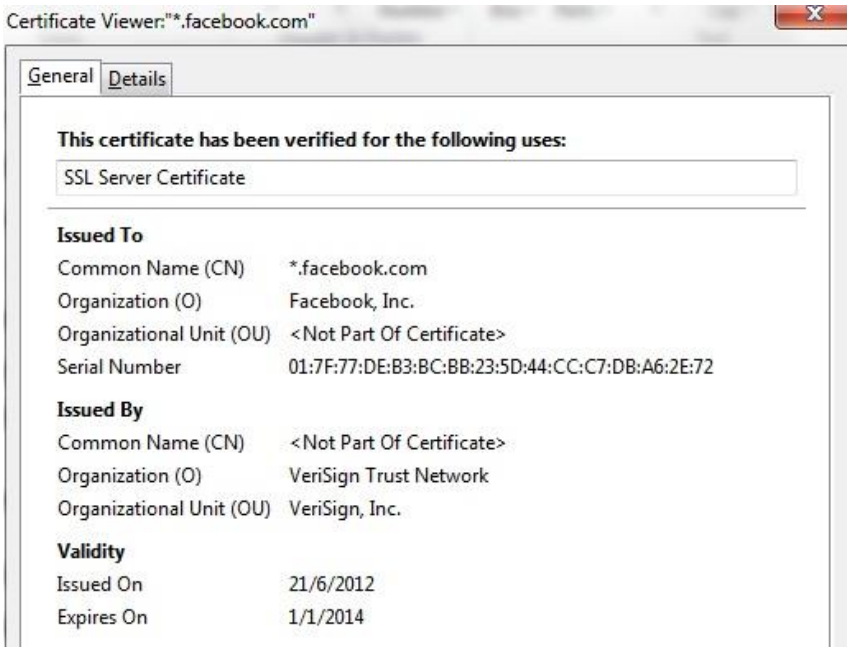
Figure 16: TLS/SSL MiTM



The client establishes an SSL connection with the attacker, by using master key K1 after its certificate is accepted by the client. In the meantime the attacker establishes a SSL connection to

the server that the client has intention to connect using a master key K2. Now, the attacker can read or modify any conversation exchanged between the client and the server. This type of attack can be prevented by automatically checking the validity of the certificate. If this is not the case the user must be always asked whether the certificate should be trusted or not.

Figure 17: Certificate example



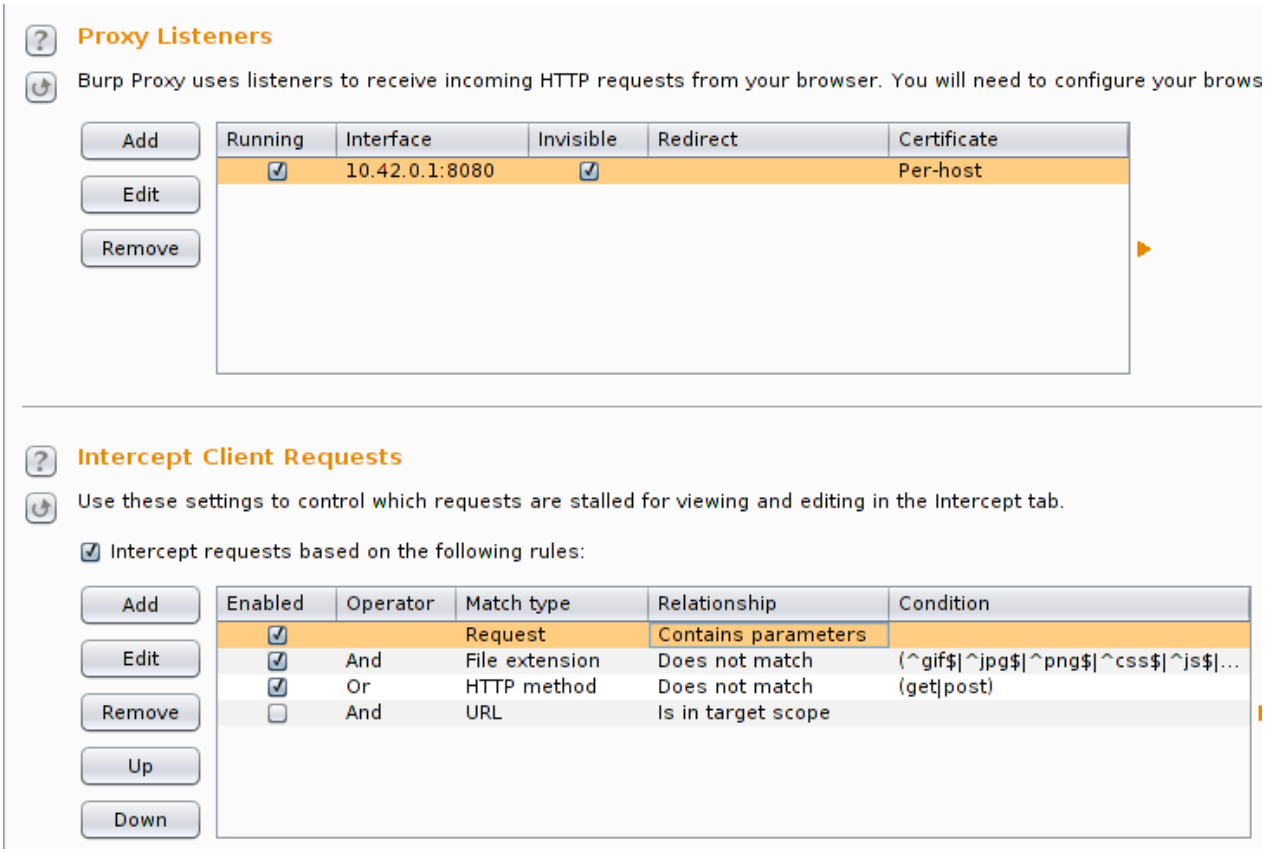
A SSL server certificate contains the public key of the server, the common name of the server, the starting day of the validity and the expiration day, all signed by a Certificate Authority. A common browser has pre-installed certificates of trusted Certificate Authorities in order to check the validity of any given certificate. Furthermore, it verifies if the requested domain matches the CN that the certificate is signed for and also checks whether the duration that the certificate is valid for, matches the local time.

3.2.5 Attack Procedure (SSL MiTM)

In order to apply such a MiTM attack, a proxy server (Burp Suite) with a self-signed certificate was used. As it is depicted, the Samsung Laptop plays the role of a router. Through

iptables configuration[7] packets whose source IP is equal to the TV's IP and whose destination port is equal to "443" are forwarded to port 8080, on "eth0" that Burp Suite listens.

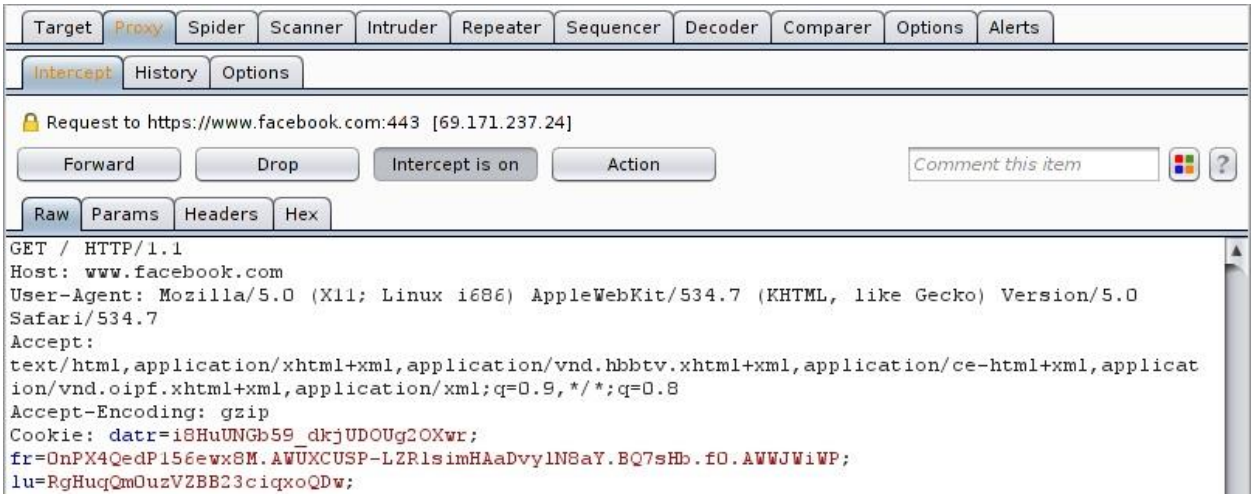
Figure 18: Burp Suite proxy



The idea behind this MiTM attack is to check whether Samsung's browser checks for trusted SSL certificates when going to a secure HTTPS website. So, a site that supports HTTPS connection and its certificate is signed by a trusted authority (VeriSign) was typed in: https://www.facebook.com/. The TV's browser, establishes an HTTPS connection with Burp Suite by accepting (without user verification) the Burp Suite's self-signed certificate. Then, the specific HTTPS GET request is being forwarded by Burp Suite to Facebook.

[7] sudo iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 443 -s 10.42.0.53 -j REDIRECT --to-port 8080

Figure 19: Decrypted HTTP GET message

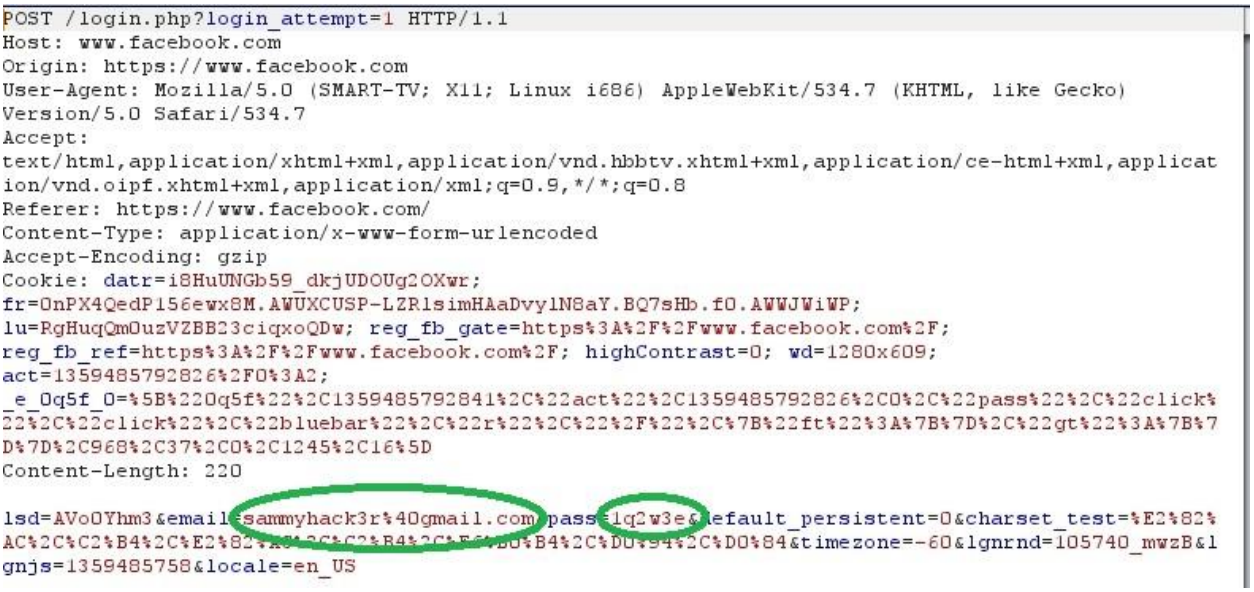


The response also passes through Burp Suite and the login page appears in Samsung's browser.

Figure 20: Facebook login page

Furthermore, a small lock badge shows up in the toolbar, indicating a false notification that a secure connection is established with the specific site. Next, the demo credentials are typed in, and Burp Suite intercepts all the traffic that is being sent.

Figure 21: Decrypted HTTP POST message



It is possible to modify the request/response, sent/received through Burp Suite and enhance the attack. As depicted from the above figure, both the user name and password appear, verifying a successful MiMT.

3.3 Miscellaneous Attacks

3.3.1 Samsung Apps

The third most important part of the TV in terms of security is the applications. Samsung has its own app-store called Samsung Apps through which you can install, update and uninstall the apps. Regarding the security, there are two questions that can be raised: What are the privileges of the apps over the system? How difficult is it for an outsider to create and install a malicious application in the TV?

Samsung only permits apps written in JavaScript/HTML/Flash[8] through its own SDK[9]. In JavaScript there are several APIs that are offered for I/O, video, audio, camera and also open/close/delete/create operations, which all work in a shared directory "/mtd_rwcommon". There is only one user found, namely the root user. So, based on the shared folder concept, every app can actually have access to the files that other apps create. Hence, by taking advantage what Samsung permits in terms of file access, an attacker can create a malicious app and affect every other app installed in the same system[10].

Recently SammyGo community released an application that gives you remote shell access using an app and a usb stick containing scripts[11] . In order to install this app you have to be logged in as a developer and configure the TV to synchronize user apps from SammyGo webserver (149.154.159.134). After this synchronization, the app *Test2* is installed to the TV and as illustrated in Figure 22, a user indication exists to point out that the application is installed in developer mode. Then you have to run the application with the usb stick plugged in and open the browser. After those steps a shell access is provided by using ***nc*** on port 23.

---

[8] http://www.samsungdforum.com/Devtools/Spec

[9] http://www.samsungdforum.com/Devtools/SdkReleaseNote

[10] http://www.samsungdforum.com/Guide/ref00001/index.html

[11] http://forum.samygo.tv/download/file.php?id=1509

Figure 22 Test2 app

The app essentially copies from the usb stick a modified library "***libm.so.6***" to the filesystem of the TV by using the following method "*eval("FilePlugin.Copy(\"/dtv/usb/sda1/libm.so.6\", \"/dtv/libm.so.6\")");*". This command proves that the sandbox of the "*mtd_rwcommon*" area is broken given that it can actually write on the restricted "/dtv" folder. When you open the browser, the script "*/dtv/usb/sda1/bin/run.sh*" (Figure 23) runs and provides shell and ftp access. This is done by installing a busybox (richer than the one installed on the TV) on the /tmp/bin folder. Finally by adding the following lines to the script it is possible to write the results of commands "netstat", "ps", "dmesg" to log files on the usb stick. The results are provided to the Appendix. Also by adding " **kill all 68**" (68 process is exeDSP) the TV **freezes** and **reboots**.

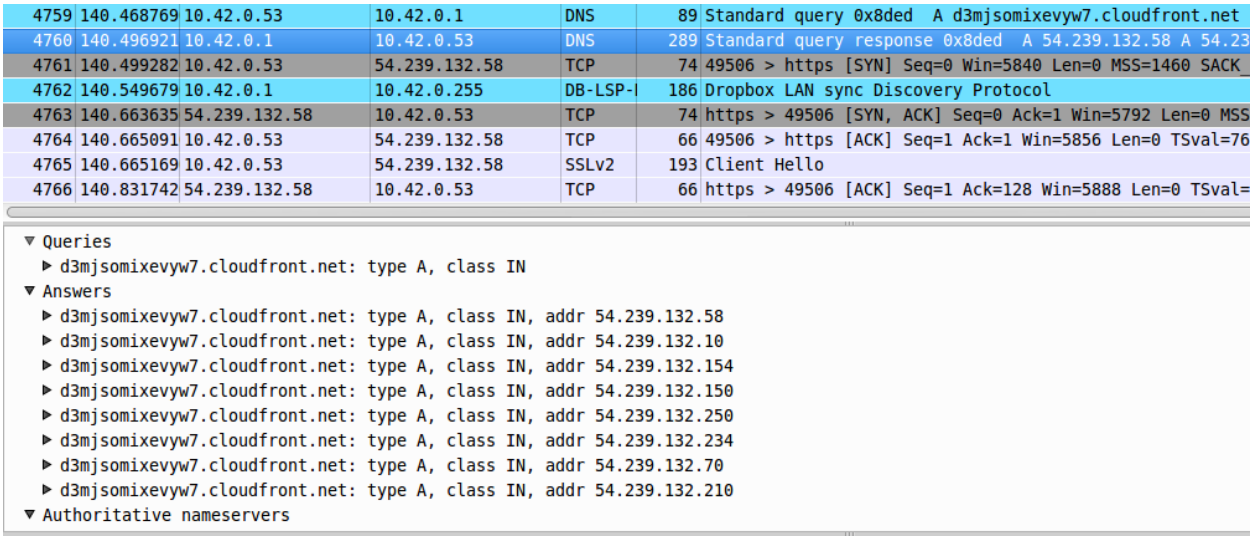
```
mkdir /tmp/os3
chmod 777 /tmp/os3
netstat -tulpn > /tmp/os3/netstat.log
ls -alt / > /tmp/os3/ls_root.log
ls -alt /bin >> /tmp/os3/bin.log
ps >> /tmp/os3/ps.log
dmesg > /dtv/usb/sda1/from_tv/dmesg.log
cp -r /tmp/os3/* /dtv/usb/sda1/from_tv/
```

Figure 23 go.sh

```
#!/bin/sh

cd /tmp
mkdir /tmp/bin
cp /dtv/usb/sda1/bin/* /tmp/bin/
chmod 777 /tmp/bin/*
/tmp/bin/busybox --install -s /tmp/bin
sync
export PATH=/tmp/bin:$PATH
export LD_LIBRARY_PATH=/tmp/bin:$LD_LIBRARY_PATH
sync
/tmp/bin/busybox tcpsvd -vE 0.0.0.0 21 /tmp/bin/busybox ftpd -w / &
/tmp/bin/remshd &
sync
sleep 10
```

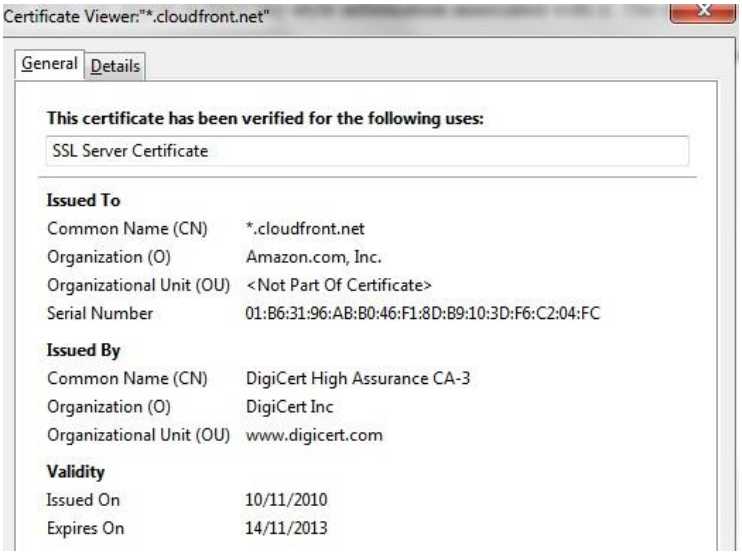
However, this cannot be considered as a remote attack because the attacker has to have physical access in order to use the usb ports of the TV and login as a developer. There is only one (legit) way to install an app to a Samsung Smart TV is by downloading it from the Samsung store. In this case, Samsung needs to approve that your app is secure before it can be publicly available. It was impossible to determine how Samsung does this evaluation and in what depth the application is checked for security flaws. The last thing that needs to be checked is how the install/update procedure from Samsung apps is performed.

Figure 24: HTTPS Download App



From the below picture it can be derived that use of “Samsung apps” demand an HTTPS connection.

Figure 25: TLS/SSL certificate



The certificate is signed with DigiCert’s private key. Using Burp Suite, a Man in The Middle attack (MiTM) was attempted, yet the TV popped up a “Network interference” error due to the denial of Burp Suite’s self-signed certificate.

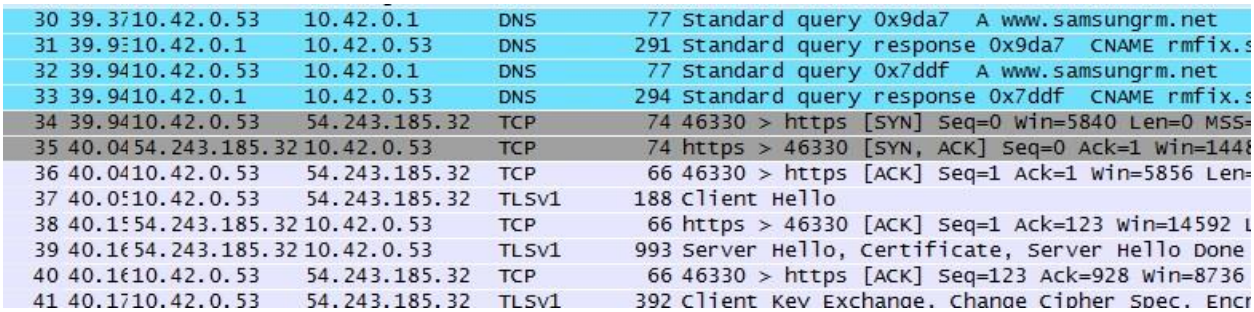
Figure 26: Network interference error

So a hacker cannot impersonate “Samsung apps” Web service unless he possesses the private key.

3.3.2 Remote help service

Samsung's OS provides a special service for remote management to help customers with troubleshooting. This is done by manually selecting this service from the support menu and only after the customer accepts the privacy agreement. Then a random pin number pops up to be verbally passed to the engineer who will check for any malfunctions. The specific service was started, while Zenmap rescanned the ports. It was noticed that there were no additional ports opened. Likewise, another session was initiated, but this time the network traffic was captured. It showed that there was a secure TCP connection with Samsung headquarters.

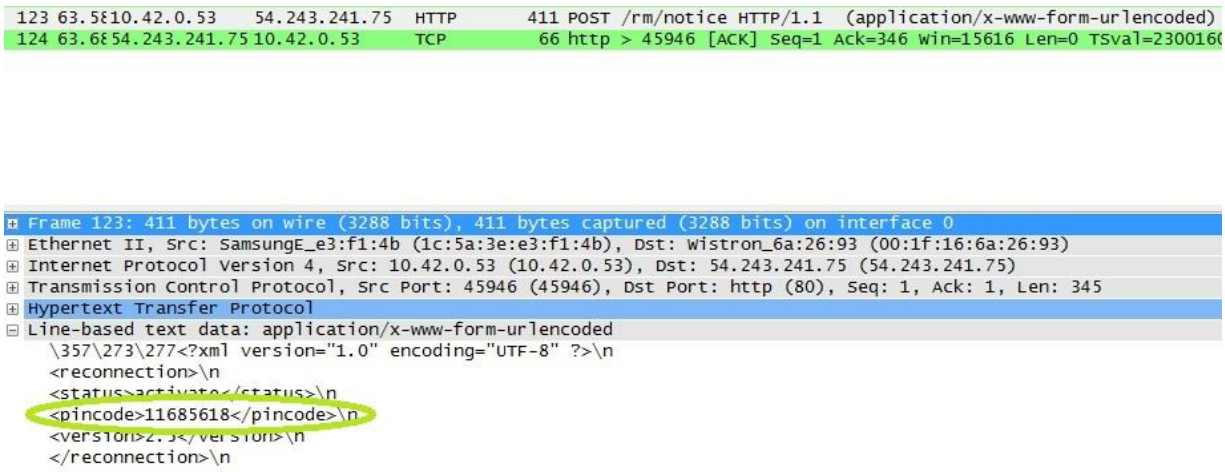
Figure 27: HTTPS connection with samungrm.net



Once again, the Web Server's certificate was signed by Samsung Hubsite CA and we verified that the TV checks the validity of the certificate that is provided.

Our hypothesis is that this secure connection, which is a reverse connection, works in the same way as various remote control services that have to bypass any firewall. We suspect that the pin-code is an identifier for the Samsung engineer to distinguish the customer's request for help.

Figure 28: HTTP POST pin-code



This pin though is sent in clear text, but this would not imply that this is a bad implementation in terms of security. Unfortunately, it cannot be stated that is secure either. It was impossible to communicate with Samsung's remote assistant service in order to establish a session; due to its high demand and lack of availability. Hence, capturing the network traffic data was infeasible. This remote service might hide a vulnerability that must be researched in depth.

3.3.3 AllShare

AllShare is a service that is used to share contents within a Local Area Network (LAN) with DLNA compliant devices. DLNA stands for Digital Living Network Alliance and uses UPnP for media management discovery and control.

For this service, the TV uses the 55000 and 55001 ports and the traffic is unencrypted since the DLNA protocol does not support encryption[12]. This cannot be considered as a security issue but a design feature. It should be mentioned that a typical authentication (hostname, ip-address) is being used, so when a media server wants to connect to the TV for the first time a pop up dialog box appears on the TV screen to accept or reject access.

[12] http://www.dlna.org/

3.3.4 Remote control

There are several apps for Smart Phones, Tablets and PCs which act as virtual remote control that operate through the network. For this service, the TV uses DLNA protocol at port 55001. As previously mentioned, this protocol uses primitive authentication and does not support encryption. Thus, someone can easily impersonate an already accepted device.

Our purpose was to investigate the type of security that the TV provides while adding a new software remote control device. Every time the user adds a new remote device to the TV, the system via AllShare sends a notification-confirmation message (Allow/Deny) along with the device's hostname. It is up to the user to accept or deny the device. After this authentication mechanism the software remote control identifies itself to the AllShare control panel. In order to demonstrate the vulnerability of the authentication mechanism, SSRemote software was used. The first thing to do is to configure the SSRemote (v0.5) to connect to the TV and then just accept it through AllShare confirmation message. As it is depicted on Figure 29 the SSRemote identifies itself with the strings "SSRemote_V05" and "SSRemote_IP" for hostname and IP respectively. By repeating the same procedure from another host we could easily remote control the TV simultaneously without the need of verification. This occurs due to the fact that the second instance provides the same strings as credentials which bypass the authentication procedure. Provided that the hostname and the IP of an authenticated device can easily be sniffed, it is possible to spoof any remote control device. (Video of Attack Demonstration)

Figure 29 AllShare settings

3.3.5 Web Server

For no apparent reason there is a Web Server (lighttpd) running on ports 80, 443 and 4443. During the black box approach different kinds of tools (web crawlers, web vulnerability

scanners) were used to stress test the lighttpd daemon. Nikto and DirBuster did not give any valuable results yet the following ssl-dos attack did.

thc-ssl-dos -n 800 10.42.0.53 443

```
….
Handshakes 48 [47.98 h/s], 462 Conn, 0 Err
….
Handshakes 219 [40.89 h/s], 474 Conn, 0 Err
Handshakes 297 [78.16 h/s], 474 Conn, 0 Err
….
Handshakes 16500 [77.36 h/s], 536 Conn, 0 Err
Handshakes 16569 [69.50 h/s], 536 Conn, 2 Err
Handshakes 16648 [78.95 h/s], 536 Conn, 2 Err
Handshakes 16719 [70.30 h/s], 536 Conn, 2 Err
Handshakes 16796 [77.66 h/s], 536 Conn, 2 Err
Handshakes 16867 [71.25 h/s], 535 Conn, 4 Err
….
Handshakes 19294 [76.06 h/s], 542 Conn, 12 Err
Handshakes 19372 [75.22 h/s], 543 Conn, 12 Err
Handshakes 19452 [82.50 h/s], 543 Conn, 12 Err
….
Connection timed out
Connection timed out
Connection timed out
```

In the above script we set an upper limit of parallel connections equal to 800. The script starts with the minimum of ~460 parallel connections and increments up to 800, while keeping an average rate of 65 handshakes per second for the whole session. As it is depicted, after the 16500th connection lighttpd starts returning errors (2 Err). At some point (after 60s) the server cannot handle any of the incoming connections; hence returning ”connection time out”. At this point the server becomes non-operational.

When the decrypted firmware was available, it was possible to read the configuration file of the Web Server.

```
#
# Security hardening
#

# Give another name to the server, we dont want hackers to know which server we use
server.tag="Swift1.0"

# Prevent access to all files except the virtual folders which are supported
$HTTP["url"] !~ "^/($|www/|ws($|/)|api($|/)|test($|/)|dtv/usb($|/)|dev($|/))" {
        url.access-deny = ("")
}
```

With the above regular expression the Web Server prevent us from having access to anything except folders “/”,“/ws”, “/api”, “/test”, “/dtv/usb” and “/dev”. A tcpdump session was scheduled to run for eight hours, yet we did not observe any local traffic.

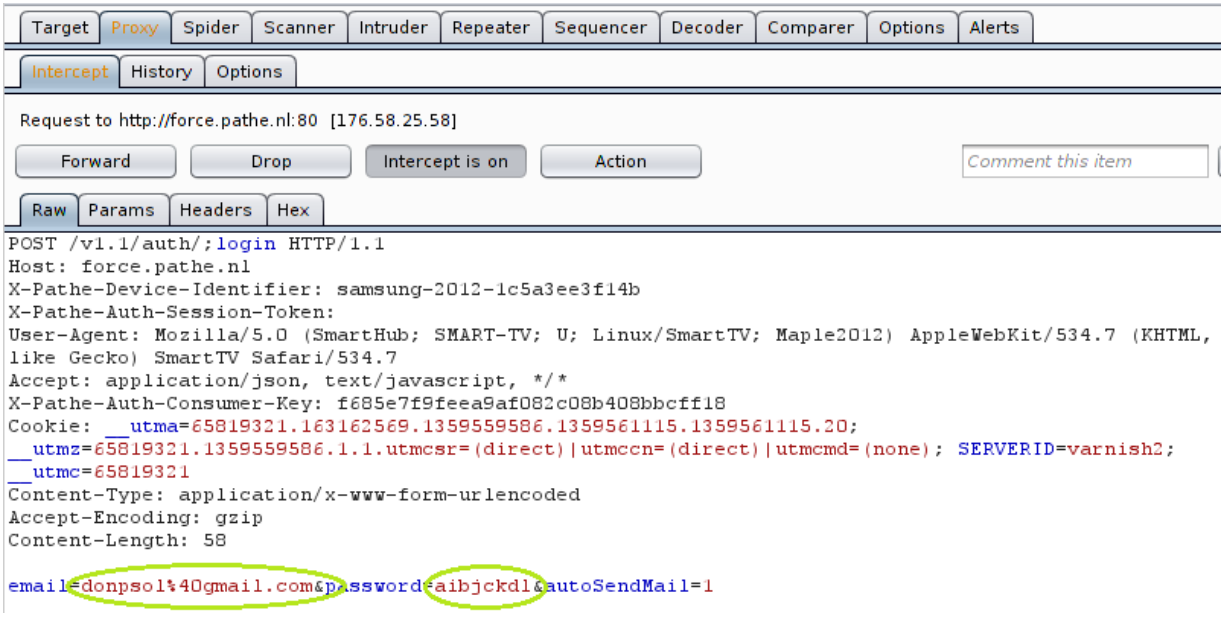
3.3.6 Other Network related daemons

There are other 4 tcp ports open (7676, 6000, 3697 and 9090) besides the ports used for lighttpd and DLNA (80, 443, 4443, 55000 and 55001). Likewise, a tcpdump session was scheduled to run for eight hours, yet no connection was established in these ports. We did not observe any local traffic in this case either.

3.3.7 Popular installed apps

The most important and popular apps were checked for flaws. In our perspective, these would be all the social apps, Skype, Facebook, Twitter and some popular apps in the Netherlands like Pathé. For all the aforementioned applications, it was checked whether the communication protocol that they use was secure. All of them use a SSL/TLS connection and after thorough checking it was determined that SSL/TLS was implemented properly. Pathé app does not use an SSL/TLS and sends sensitive data in clear text (email=donpsol%40gmail.com & password=aibjckdl) as it is depicted from the following figure.

Figure 30: Pathé HTTP POST



4 Discussion

During the assessment of Samsung TV, we identified the following design flaws.

1st Design flaw

It has already been stated that by changing only a DNS response you can upgrade the firmware of the TV from a non-authentic Samsung Web Server. Due to the simplicity of this attack and the fact that it can be performed in a single or in a larger scale (i.e. DNS poisoning, Rogue DHCP Server) the consequences can be severe. Indeed by creating a custom firmware a potential hacker can deliver it through the update procedure without the user noticing it. **A custom firmware might be either a modified firmware that gives the attacker access to the TV (mic/camera/sensitive data) or a firmware that has been deliberately misconfigured to force the TV in a non-operational state.**

2nd Design flaw

For a smart device, a browser is indispensable let alone when it's pre-installed (no uninstall option) and the only available for Samsung Smart TVs. It is proven that the browser does not check the validity of the SSL certificates. Going to a secure site (HTTPS) there is a lock indicating that the user's data are under a secure channel, yet this is not the case. Indeed, **not only you can read but also modify the exchanged packets without the user noticing it.** We are talking about Samsung branded, Firefox based browser that should be updated to the latest version, or at least the next firmware should include the CA certificates pre-installed.

3rd Design flaw

The fact that there is only one user in the system, "root" and also that all application files share a common folder makes the TV vulnerable. Considering the situation that there is one vulnerable application running, **an attacker can exploit this application and compromise the whole TV**. The only countermeasure is the application certification procedure (beforehand) where Samsung checks the application for malicious content. At what depth does Samsung check the security risks that one app might carry is really unknown.

4th Design flaw

Samsung uses DLNA protocol for both video/audio streaming and remote control feature. DLNA was designed to be unencrypted, but the idea of using an unsecure protocol for the Remote control service is considered inadequate. A possible negative outcome of this design flaw would be for example to present inappropriate video content in a common public area like café/office/class. A new secured protocol should be designed for remote control service thus providing the user a more risk-free remote session.

5 Conclusion

There were several countermeasures that were implemented by Samsung for many of the services provided, in order to offer a secure environment for the users. Yet in some cases, these were proven to be insufficient. In particular, the firmware by itself is encrypted by AES+XoR, but the keys have already been found, making the firmware modification possible. In addition, based on the attack method described, an upgrade with a modified firmware can be done remotely.

Moreover, this Smart TV system supports SSL/TLS protocol which, in most cases, is successfully implemented with one exception, the browser. The browser accepts any SSL certificate that makes it susceptible to a Man in The Middle Attack without the user's notice.

Regarding the Samsung Apps repository, the downloading procedure is performed through a secure connection with Samsung's servers that provide only apps which are already approved by Samsung. It wasn't possible to determine the level of security check that is being applied to apps, given that there was no time creating a malicious one and waiting to be approved or rejected. The security policy applied by Samsung was to create a Sandbox with which a app can only write/read from a specific folder and not outside of it. This policy however, was proven to be violated by using a Javascript method which copies files from a source to a destination outside the sandbox.

Finally, some popular and preinstalled apps were checked in terms of the communication protocol being used and all of them were found secure except "Pathé".

6 Future Work

There are quite a lot of areas that can be investigated and considered as a future work, considering we are talking about a new commercial product equipped with various network services.

Create a Custom Firmware

Create a custom Firmware that provides remote shell access to the TV and distribute it through our "Attack Procedure".

Check Remote Help Service

It is intriguing to actually eavesdrop the communication between the TV and the Samsung support center during this Remote Help Service, since it could reveal a serious vulnerability.

Root ES models in a remote way.

The possibility of rooting the TV remotely and enabling shell access would mean taking the full control of the TV in a stealthy way.

Check Broadcast update.

It is very interesting that Samsung has enabled this feature and moreover it would be no surprise if there is weak or no security at all for this procedure. With the current available tools, someone can create a legitimate look-alike firmware, where a hacker can create a multi-attack scheme through broadcasting.

Further investigation about daemons

There were 4 different ports open whose existence and functionality were unclear. Ports 9090, 3697, 7676, 6000.

Samsung TV apps security check assessment.

Develop a Trojan horse and check whether Samsung accepts it for the Samsung TV App store.

Unclear Web Server existence

Find what lighttpd is used for. It runs in 80 & 443/4443 ports.

Appendix

Nessus

Figure 31: Nessus Summary

Summary

Critical	High	Medium	Low	Info	Total
0	0	4	2	21	27

Details

Severity	Plugin Id	Name
Medium (6.4)	51192	SSL Certificate Cannot Be Trusted
Medium (4.3)	42873	SSL Medium Strength Cipher Suites Supported
Medium (4.3)	53491	SSL / TLS Renegotiation DoS
Medium (4.0)	35291	SSL Certificate Signed using Weak Hashing Algorithm
Low (3.2)	50686	IP Forwarding Enabled
Low (2.6)	10407	X Server Detection
Info	10107	HTTP Server Type and Version
Info	10114	ICMP Timestamp Request Remote Date Disclosure
Info	10287	Traceroute Information
Info	10386	Web Server No 404 Error Code Check
Info	10863	SSL Certificate Information
Info	11219	Nessus SYN scanner
Info	11936	OS Identification
Info	19506	Nessus Scan Information
Info	21643	SSL Cipher Suites Supported
Info	22964	Service Detection
Info	24260	HyperText Transfer Protocol (HTTP) Information
Info	25220	TCP/IP Timestamps Supported
Info	35716	Ethernet Card Manufacturer Detection
Info	43111	HTTP Methods Allowed (per directory)
Info	45590	Common Platform Enumeration (CPE)
Info	50845	OpenSSL Detection
Info	51891	SSL Session Resume Supported
Info	54615	Device Type
Info	56984	SSL / TLS Versions Supported
Info	57041	SSL Perfect Forward Secrecy Cipher Suites Supported

Zenmap

Starting Nmap 6.00 (http://nmap.org) at 2013-01-08 11:27 CET
NSE: Loaded 93 scripts for scanning.
NSE: Script Pre-scanning.
Initiating ARP Ping Scan at 11:27
Scanning 10.42.0.53 [1 port]
Completed ARP Ping Scan at 11:27, 0.04s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 11:27
Completed Parallel DNS resolution of 1 host. at 11:27, 0.09s elapsed
Initiating SYN Stealth Scan at 11:27
Scanning 10.42.0.53 [65535 ports]
Discovered open port 443/tcp on 10.42.0.53
Discovered open port 80/tcp on 10.42.0.53
Discovered open port 7676/tcp on 10.42.0.53
Discovered open port 4443/tcp on 10.42.0.53
Discovered open port 55000/tcp on 10.42.0.53
Discovered open port 55001/tcp on 10.42.0.53
Discovered open port 6000/tcp on 10.42.0.53
Discovered open port 3697/tcp on 10.42.0.53
Discovered open port 9090/tcp on 10.42.0.53
Completed SYN Stealth Scan at 11:27, 4.63s elapsed (65535 total ports)
Initiating Service scan at 11:27
Scanning 9 services on 10.42.0.53
Completed Service scan at 11:29, 126.12s elapsed (9 services on 1 host)
Initiating OS detection (try #1) against 10.42.0.53
NSE: Script scanning 10.42.0.53.
Initiating NSE at 11:29
Completed NSE at 11:30, 30.02s elapsed
Nmap scan report for 10.42.0.53
Host is up (0.0015s latency).
Not shown: 65526 closed ports
PORT STATE SERVICE VERSION
80/tcp open http lighttpd
|_http-methods: OPTIONS GET HEAD POST
|_http-title: 404 - Not Found
443/tcp open ssl/http lighttpd
|_http-title: 404 - Not Found
|_http-methods: OPTIONS GET HEAD POST
| ssl-cert: Subject: commonName=106.1.9.39/organizationName=Samsung SERI/stateOrProvinceName=|
Issuer: commonName=Root CA/organizationName=Samsung SERI/stateOrProvinceName=Surrey/|
Public Key type: rsa
| Public Key bits: 1024
| Not valid before: 1970-01-01 00:00:00
| Not valid after: 2030-01-01 00:00:00
| MD5: 3cc6 e4b0 203c fa68 5adf 3808 a651 9549
|_SHA-1: a387 008c bf7a 3745 fc7e 9ada 9200 df31 7bcb 65b7
3697/tcp open nw-license?
4443/tcp open ssl/pharos?
6000/tcp open X11 (access denied)
7676/tcp open imqbrokerd?
9090/tcp open zeus-admin?
55000/tcp open unknown
55001/tcp open tcpwrapped

2 services unrecognized despite returning data. If you know the service/version, please
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port7676-TCP:V=6.00%I=7%D=1/8%Time=50EBF49C%P=x86_64-unknown-linux-gnu%
SF:r(GetRequest,52,"HTTP/1\.1\x20400\x20Bad\x20Request\r\nContent-Type:\x2
SF:0\r\nContent-Length:\x200\r\nConnection:\x20close\r\n\r\n")%r(HTTPOptio
SF:ns,52,"HTTP/1\.1\x20400\x20Bad\x20Request\r\nContent-Type:\x20\r\nConte
SF:nt-Length:\x200\r\nConnection:\x20close\r\n\r\n")%r(RTSPRequest,52,"HTT
SF:P/1\.1\x20400\x20Bad\x20Request\r\nContent-Type:\x20\r\nContent-Length:
SF:\x200\r\nConnection:\x20close\r\n\r\n")%r(FourOhFourRequest,52,"HTTP/1\
SF:.1\x20400\x20Bad\x20Request\r\nContent-Type:\x20\r\nContent-Length:\x20
SF:0\r\nConnection:\x20close\r\n\r\n")%r(SIPOptions,52,"HTTP/1\.1\x20400\x
SF:20Bad\x20Request\r\nContent-Type:\x20\r\nContent-Length:\x200\r\nConnec
SF:tion:\x20close\r\n\r\n");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port9090-TCP:V=6.00%I=7%D=1/8%Time=50EBF497%P=x86_64-unknown-linux-gnu%
SF:r(GetRequest,54,"HTTP/1\.0\x20200\x20OK\r\nContent-type:\x20application
SF:/octet-stream\r\nCache-Control:\x20no-cache\r\n\r\n")%r(HTTPOptions,54,
SF:"HTTP/1\.0\x20200\x20OK\r\nContent-type:\x20application/octet-stream\r\
SF:nCache-Control:\x20no-cache\r\n\r\n")%r(FourOhFourRequest,54,"HTTP/1\.0
SF:\x20200\x20OK\r\nContent-type:\x20application/octet-stream\r\nCache-Con
SF:trol:\x20no-cache\r\n\r\n");
MAC Address: 1C:5A:3E:E3:F1:4B (Unknown)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:kernel:2.6
OS details: Linux 2.6.17 - 2.6.36, Linux 2.6.19 - 2.6.35
Uptime guess: 0.006 days (since Tue Jan 8 11:21:05 2013)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=197 (Good luck!)
IP ID Sequence Generation: All zeros
Service Info: OS: Unix
TRACEROUTE
HOP RTT ADDRESS
1 1.50 ms 10.42.0.53
NSE: Script Post-scanning.
Read data files from: /usr/bin/../share/nmap
OS and Service detection performed. Please report any incorrect results at http://nmap.Nmap done: 1 IP address (1 host up) scanned in 162.94 seconds
Raw packets sent: 65555 (2.885MB) | Rcvd: 65551 (2.623MB)

ls_bin.log

```
drwxrwxrwx   26 root     0           682 Aug 29  2012 ..
lrwxrwxrwx    1 root     0             7 Aug 29  2012 ash -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 awk -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 cat -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 chmod -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 chown -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 cp -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 cttyhack -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 date -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 dd -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 df -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 dmesg -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 echo -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 egrep -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 fgrep -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 grep -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 hostname -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 kill -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 ln -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 ls -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 mkdir -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 mknod -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 mount -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 mv -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 netstat -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 nice -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 pidof -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 ping -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 ps -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 pwd -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 rm -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 sed -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 sh -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 sleep -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 stty -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 sync -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 touch -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 umount -> busybox
lrwxrwxrwx    1 root     0             7 Aug 29  2012 usleep -> busybox
-rwxrwxrwx    1 root     0         37636 Jan 26  2012 authuld
drwxrwxrwx    2 root     0           504 Jan 20  2012 .
-rwxrwxrwx    1 root     0       1429908 Aug 26  2011 busybox
```

ls_root.log

```
drwxr-xr-x   39 root     0            1381 Oct 15  2012 mtd_exe
drwxr-xr-x   10 root     0             174 Oct 15  2012 mtd_appext
drwxr-xr-x   14 root     0             306 Oct 15  2012 mtd_rocommon
drwxrwxrwx   26 root     0             682 Aug 29  2012 .
drwxrwxrwx   26 root     0             682 Aug 29  2012 ..
-rwxrwxrwx    1 1009     1009           81 Aug 29  2012 .version
lrwxrwxrwx    1 root     0              14 Aug 29  2012 .info -> /mtd_exe/.info
lrwxrwxrwx    1 root     0              12 Aug 29  2012 Java -> mtd_exe/Java
lrwxrwxrwx    1 root     0              11 Aug 29  2012 linuxrc -> bin/busybox
lrwxrwxrwx    1 root     0               8 Aug 29  2012 mtd_appdata -> mtd_exe/
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_boot -> etc/Scripts/
lrwxrwxrwx    1 root     0              10 Aug 29  2012 mtd_chmap -> mtd_rwarea
lrwxrwxrwx    1 root     0               7 Aug 29  2012 mtd_cmmlib -> mtd_exe
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_down -> mtd_rwcommon
lrwxrwxrwx    1 root     0              10 Aug 29  2012 mtd_epg -> mtd_rwarea
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_factory -> mtd_rwcommon
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_gemstar -> mtd_rwcommon
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_mhp -> mtd_rwcommon
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_moip -> mtd_rwcommon
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_pers -> mtd_rwcommon
lrwxrwxrwx    1 root     0               3 Aug 29  2012 mtd_ram -> tmp
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_wiselink -> mtd_rwcommon
lrwxrwxrwx    1 root     0              12 Aug 29  2012 mtd_yahoo -> mtd_rwcommon
drwxrwxrwx    3 root     0             237 Mar 14  2012 etc
drwxrwxrwx    3 root     0             474 Jan 31  2012 sbin
drwxrwxrwx    2 root     0             504 Jan 20  2012 bin
drwxrwxrwx    3 root     0             759 Dec  9  2011 lib
drwxrwxrwx   12 root     0            5553 Nov 25  2011 dev
drwxrwxrwx    2 root     0              89 Nov  2  2011 util
drwxrwxrwx    2 root     0               3 Nov  8  2010 mnt
drwxrwxrwx    5 root     0              51 Nov  8  2010 usr
drwxr-xr-x    1 root     0               0 Jan  1  1980 mtd_drmregion_a
drwxr-xr-x    1 root     0               0 Jan  1  1980 mtd_drmregion_b
drwxr-xr-x    1 root     0               0 Jan  1  1980 mtd_emanual
drwxr-xr-x    1 root     0               0 Jan  1  1980 mtd_swu
drwxrwxrwt    3 root     0             140 Jan  1 00:01 dtv
drwxrwxrwt    5 root     0             260 Jan  1 00:01 tmp
drwxr-xr-x    1 root     0               0 Jan  1 00:00 mtd_rwcommon
drwxr-xr-x    1 root     0               0 Jan  1 00:00 mtd_rwarea
drwxr-xr-x    1 root     0               0 Jan  1 00:00 mtd_contents
drwxrwxrwt    2 root     0              40 Jan  1 00:00 core
drwxrwxrwt    2 root     0              40 Jan  1 00:00 dsm
dr-xr-xr-x   57 root     0               0 Jan  1 00:00 proc
drwxr-xr-x   11 root     0               0 Jan  1 00:00 sys
```

netstat.log

Active Internet connections (only servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	127.0.0.1:58912	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:50242	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	0.0.0.0:9090	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:54211	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:46340	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:45253	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:48869	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:41351	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:47111	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:58600	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:34761	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:57417	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:40521	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:42122	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:53068	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:55692	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:42445	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:45069	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:53485	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:49389	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:44430	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:51982	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:36111	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:54735	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:52112	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	0.0.0.0:80	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:50000	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	0.0.0.0:6000	0.0.0.0:*	LISTEN	236/X
tcp	0	0	127.0.0.1:45425	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:41969	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	0.0.0.0:3697	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:45138	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:53938	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:37523	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:49811	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:36979	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:37715	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:41364	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:33012	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:58868	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:46484	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	0.0.0.0:21	0.0.0.0:*	LISTEN	1091/busybox
tcp	0	0	127.0.0.1:38197	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	0.0.0.0:23	0.0.0.0:*	LISTEN	1092/remshd
tcp	0	0	0.0.0.0:55000	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	0.0.0.0:55001	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:36730	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:53115	0.0.0.0:*	LISTEN	68/exeDSP
tcp	0	0	127.0.0.1:38555	0.0.0.0:*	LISTEN	261/WidgetEngine
tcp	0	0	127.0.0.1:57947	0.0.0.0:*	LISTEN	261/WidgetEngine

```
tcp        0      0 0.0.0.0:4443            0.0.0.0:*               LISTEN      68/exeDSP
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      68/exeDSP
tcp        0      0 127.0.0.1:38268         0.0.0.0:*               LISTEN      261/WidgetEngine
tcp        0      0 10.42.0.53:7676         0.0.0.0:*               LISTEN      68/exeDSP
tcp        0      0 127.0.0.1:58430         0.0.0.0:*               LISTEN      68/exeDSP
tcp        0      0 127.0.0.1:49566         0.0.0.0:*               LISTEN      68/exeDSP
tcp        0      0 127.0.0.1:45822         0.0.0.0:*               LISTEN      261/WidgetEngine
tcp        0      0 127.0.0.1:54623         0.0.0.0:*               LISTEN      68/exeDSP
tcp        0      0 127.0.0.1:40959         0.0.0.0:*               LISTEN      261/WidgetEngine
tcp        0      0 127.0.0.1:49087         0.0.0.0:*               LISTEN      261/WidgetEngine
udp        0      0 0.0.0.0:1900            0.0.0.0:*                           68/exeDSP
udp        0      0 10.42.0.53:24234        0.0.0.0:*                           68/exeDSP
udp        0      0 127.0.0.1:2002          0.0.0.0:*                           68/exeDSP
udp        0      0 0.0.0.0:7900            0.0.0.0:*                           68/exeDSP
```

ps.log

```
  PID USER       VSZ STAT COMMAND
    1 root      1688 S    init
    2 root         0 SW   [kthreadd]
    3 root         0 SW   [ksoftirqd/0]
    4 root         0 SW   [migration/0]
    5 root         0 SW   [migration/1]
    6 root         0 SW   [ksoftirqd/1]
    7 root         0 SW   [events/0]
    8 root         0 SW   [events/1]
    9 root         0 SW   [khelper]
   10 root         0 SW   [async/mgr]
   11 root         0 SW   [sync_supers]
   12 root         0 SW   [bdi-default]
   13 root         0 SW   [kblockd/0]
   14 root         0 SW   [kblockd/1]
   15 root         0 SW   [kmmcd]
   16 root         0 SW   [kdtvlogd]
   17 root         0 SW   [kswapd0]
   18 root         0 SW   [xfs_mru_cache]
   19 root         0 SW   [xfslogd/0]
   20 root         0 SW   [xfslogd/1]
   21 root         0 SW   [xfsdatad/0]
   22 root         0 SW   [xfsdatad/1]
   23 root         0 SW   [xfsconvertd/0]
   24 root         0 SW   [xfsconvertd/1]
   25 root         0 SW   [mmcqd]
   36 root      1692 S    -/bin/sh
   56 root      1692 S    /bin/sh /mtd_exe/rc.local
   67 root     1192m S    ./exeDSP
  146 root         0 SW   [flush-179:0]
  236 root      393m S    /mtd_cmmlib/Runtime/bin/X -logfile /mtd_rwarea/Xlog.
  417 root      628m R    /mtd_appdata/InfoLink/lib/WidgetEngine 67 51982
  517 root         0 SW   [khubd]
  599 root      1688 S    udhcpc -i eth0 -T 2 -b
  602 root         0 SW   [scsi_eh_0]
  603 root         0 SW   [usb-storage]
  609 root         0 DW   [scsi-poller]
  679 root         0 SW   [usbhid_resumer]
  708 root         0 SW   [flush-8:0]
  877 root      3272 S N  /mtd_exe/Comp_LIB/UEP.b
 1061 root     78856 S    /mtd_down/emps/empWebBrowserDRI/bin/BrowserLauncher
 1064 root      1688 S    sh -c /bin/sh /dtv/usb/sda1/run.sh
 1065 root      1688 S    /bin/sh /dtv/usb/sda1/run.sh
 1068 root      1688 S    /bin/sh -x /dtv/usb/sda1/bin/go.sh
 1075 root      4148 R    ps
```

dmesg.log

```
ts in Zone order, mobility grouping on.  Total pages: 126212
Kernel command line: console=tty1,115200 root=/dev/mmcblk0p3 rootfstype=squashfs
LX_MEM=0x40200000,0x14900000 LX_MEM2=0xA4E00000,0xB200000
EMAC_MEM=0x40000000,0x100000 SELP_ENABLE=1198282 Onboot  : 1003 quiet
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 329MB 178MB = 507MB total
Memory: 508960k/508960k available, 10208k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    DMA     : 0xffc00000 - 0xffe00000   (   2 MB)
    vmalloc : 0xe0000000 - 0xf8000000   ( 384 MB)
    lowmem  : 0xc0000000 - 0xdfb00000   ( 507 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .init : 0xc0008000 - 0xc0025000   ( 116 kB)
      .text : 0xc0025000 - 0xc02f7000   (2888 kB)
      .data : 0xc02f8000 - 0xc031b460   ( 142 kB)
SLUB: Genslabs=11, HWalign=32, Order=0-3, MinObjects=0, CPUs=2, Nodes=1
Hierarchical RCU implementation.
        RCU-based detection of stalled CPUs is disabled.
        Verbose stalled-CPUs detection is disabled.
NR_IRQS:256
Global Timer Frequency = 450 MHz
CPU Clock Frequency = 900 MHz
Console: colour dummy device 80x30
console [tty1] enabled
[VDLP] preset_lpj manual set to 3588096
Calibrating delay loop (skipped) preset value.. 1794.04 BogoMIPS (lpj=3588096)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
Calibrating local timer... 450.50MHz.
CPU1: Booted secondary processor
[VDLP] preset_lpj manual set to 3588096
Brought up 2 CPUs
SMP: Total of 2 processors activated (3588.09 BogoMIPS).
NET: Registered protocol family 16
#_# Mstar_ehc_platform_init 0xFD200DE0 => 4e00
L310 cache controller enabled
l2x0: 8 ways, CACHE_ID 0x410000c8, AUX_CTRL 0x32460000, Cache Size: 524288 B
bio: create slab <bio-0> at 0
vgaarb: loaded
SCSI subsystem initialized
Switching to clocksource timer1
NET: Registered protocol family 2
IP route cache hash table entries: 4096 (order: 2, 16384 bytes)
TCP established hash table entries: 16384 (order: 5, 131072 bytes)
TCP bind hash table entries: 16384 (order: 5, 196608 bytes)
TCP: Hash tables configured (established 16384 bind 16384)
TCP reno registered
```

```
UDP hash table entries: 256 (order: 1, 8192 bytes)
UDP-Lite hash table entries: 256 (order: 1, 8192 bytes)
NET: Registered protocol family 1
PCI: CLS 0 bytes, default 32
squashfs: version 4.0 (2009/01/31) Phillip Lougher
SGI XFS with ACLs, security attributes, large block/inode numbers, no debug enabled
msgmni has been set to 994
io scheduler noop registered (default)
io scheduler deadline registered
[drm] Initialized drm 1.1.0 20060810
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
serial8250: ttyS0 at I/O 0xfd201300 (irq = 192) is a 16550
serial8250: ttyS1 at I/O 0xfd220c00 (irq = 231) is a 16550
serial8250: ttyS2 at I/O 0xfd220d00 (irq = 242) is a 16550
brd: module loaded
loop: module loaded
SD Card Init: mstar_mci_init.
Probe MCI devices
ramzswap: Creating 4 devices ...
Netfilter messages via NETLINK v0.30.
nf_conntrack version 0.5.0 (7952 buckets, 31808 max)
ip_tables: (C) 2000-2006 Netfilter Core Team
TCP cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 3
FCIE3 set 300KHz...
FCIE3 set 300KHz...
FCIE3 set 300KHz...
FCIE3 set 300KHz...
FCIE3 set 300KHz...
FCIE3 set 300KHz...
FCIE3 set 300KHz...
FCIE3 set 300KHz...
FCIE3 set 32MHz...
FCIE3 set 32MHz...
mmc0: new high speed MMC card at address 0001
mmcblk0: mmc0:0001 M2G1HF 1.86 GiB
 mmcblk0: p1 p2 p3 p4 < p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 p16 p17 p18 p19 p20 p21 >
================================================================================
 SAMSUNG VDLP Kernel
 Version : 0064, release
 Applied Last Patch Number : 0716, DTV, X10P, release, DEU_BRANCH
================================================================================
VFS: Mounted root (squashfs filesystem) readonly on device 179:3.
Freeing init memory: 116K
##### send signal from USER, SIG : 0, init(1)->???(27) sys_kill
rfs_glue: module license 'Samsung, Proprietary' taints kernel.
Disabling lock debugging due to kernel taint
rfs_glue mod ld
rfs_fat mod ld
********** UART1 Request IRQ **********
==================================
FCR c1
==================================
```

UART DLL or UART_IER is being modified by value:0xbf
UART DLL or UART_IER is being modified by value:0x2
==========
FCR c1
==========
UART DLL or UART_IER is being modified by value:0xbf
UART DLL or UART_IER is being modified by value:0x2
Mali DRM initialize, driver name: mali_drm, version 2.1
[drm] Initialized mali_drm 2.1.1 20101111 on minor 0
Mali DRM initialize, driver name: mali_drm, version 2.1
[drm] Initialized mali_drm 2.1.1 20101111 on minor 1
UMP: UMP device driver ïN½ÍZχZ®µK loaded
Mali: Mali device driver ïN½ÍZχZ®µK loaded
samsung_mali mod ld
GOP driver inits
REG_ARM_BASE 0xFD200000
mstar_fb_init

atach component abnormal patch

samsung_mstar mod ld
X10P FASTLOGO ver : 0014
FastLogo On
SD_MISC_LVDS_FMT_VESA_8
FastLogo mod ld
Exit Show Logo Successfully
FastLogo mod uld
ioctl: GET_PNL_INIT_STATUS
********** UART1 Request IRQ **********
==========
FCR c1
==========
UART DLL or UART_IER is being modified by value:0xbf
UART DLL or UART_IER is being modified by value:0x2
==========
FCR c1
==========
==========
FCR c1
==========
Adding 102396k swap on /dev/ramzswap0. Priority:-1 extents:1 across:102396k SS
[CIP_KERNEL] /bin/authuld can read (after=0)
[CIP_KERNEL] >>> (/bin/authuld) file is successfully authenticated <<<
[CIP_KERNEL] (0)th waiting.
[PHY Addr] ==> :0
alloRAM_PA_BASE= 0x40000000 alloRAM_SIZE= 0x100000

```
alloRAM_VA_BASE= 0xE0400000  alloRAM_SIZE= 0x100000
high_memory:0xdfb00000
RAM_VA_BASE:e0404000
RAM_PA_BASE:40004000
RX_BUFFER_BASE:40004200
RBQP_BASE:40004000
TX_BUFFER_BASE:40024200
RAM_VA_PA_OFFSET:a0400000
TX_SKB_BASE:40024400
mdrv_emac mod ld
----------PHY Connect again----------
- KCONNECT: 0=,0
--------------------------------------
##### send signal from USER, SIG : 10, WidgetEngine(417)->exeDSP(67) sys_kill
 ##### call default signal (10) handler
Tuxera NTFS driver 3012.1.8 [Flags: R/W MODULE].
tntfs mod ld
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
usbcore mod ld
ehci_hcd: block sizes: qh 60 qtd 128 itd 256 sitd 96
In ehci_hcd_mstar_drv_probe
Mstar-ehci-2 H.W init
Titania3_series_start_ehc start
Mstar-ehci-2 Mstar-ehci-2.1: Mstar EHCI
drivers/mstar/usb/core/inode.c: creating file 'devices'
drivers/mstar/usb/core/inode.c: creating file '001'
Mstar-ehci-2 Mstar-ehci-2.1: new USB bus registered, assigned bus number 1
Mstar-ehci-2 Mstar-ehci-2.1: park 0
Mstar-ehci-2 Mstar-ehci-2.1: irq 226, io mem 0xfd201a00
Mstar-ehci-2 Mstar-ehci-2.1: reset command 080b02 park=3 ithresh=8 period=1024 Reset HALT
Mstar-ehci-2 Mstar-ehci-2.1: init command 010001 (park)=0 ithresh=1 period=1024 RUN
usb usb1: default language 0x0409
usb usb1: udev 1, busnum 1, minor = 0
usb usb1: New USB device found, idVendor=1d6b, idProduct=0002
usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb1: Product: Mstar EHCI
usb usb1: Manufacturer: Linux 2.6.35.13 ehci_hcd
usb usb1: SerialNumber: mstar
usb usb1: usb_probe_device
usb usb1: configuration #1 chosen from 1 choice
usb usb1: adding 1-0:1.0 (config #1, interface 0)
hub 1-0:1.0: usb_probe_interface
hub 1-0:1.0: usb_probe_interface - got id
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
hub 1-0:1.0: standalone hub
hub 1-0:1.0: no power switching (usb 1.0)
hub 1-0:1.0: individual port over-current protection
hub 1-0:1.0: power on to power good time: 20ms
hub 1-0:1.0: local power source is good
hub 1-0:1.0: trying to enable port power on non-switchable hub
hub 1-0:1.0: trying to enable port power on non-switchable hub
```

```
drivers/mstar/usb/core/inode.c: creating file '001'
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ehci_hcd: block sizes: qh 60 qtd 128 itd 256 sitd 96
In ehci_hcd_mstar_drv_probe
Mstar-ehci-1 H.W init
Titania3_series_start_ehc start
Mstar-ehci-1 Mstar-ehci-1.0: Mstar EHCI
drivers/mstar/usb/core/inode.c: creating file '002'
Mstar-ehci-1 Mstar-ehci-1.0: new USB bus registered, assigned bus number 2
Mstar-ehci-1 Mstar-ehci-1.0: park 0
Mstar-ehci-1 Mstar-ehci-1.0: irq 199, io mem 0xfd204800
Mstar-ehci-1 Mstar-ehci-1.0: reset command 080b02 park=3 ithresh=8 period=1024 Reset HALT
Mstar-ehci-1 Mstar-ehci-1.0: init command 010001 (park)=0 ithresh=1 period=1024 RUN
usb usb2: default language 0x0409
usb usb2: udev 1, busnum 2, minor = 128
usb usb2: New USB device found, idVendor=1d6b, idProduct=0002
usb usb2: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb2: Product: Mstar EHCI
usb usb2: Manufacturer: Linux 2.6.35.13 ehci_hcd
usb usb2: SerialNumber: mstar
usb usb2: usb_probe_device
usb usb2: configuration #1 chosen from 1 choice
usb usb2: adding 2-0:1.0 (config #1, interface 0)
hub 2-0:1.0: usb_probe_interface
hub 2-0:1.0: usb_probe_interface - got id
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 1 port detected
hub 2-0:1.0: standalone hub
hub 2-0:1.0: no power switching (usb 1.0)
hub 2-0:1.0: individual port over-current protection
hub 2-0:1.0: power on to power good time: 20ms
hub 2-0:1.0: local power source is good
hub 2-0:1.0: trying to enable port power on non-switchable hub
hub 1-0:1.0: state 7 ports 1 chg 0000 evt 0000
hub 2-0:1.0: trying to enable port power on non-switchable hub
drivers/mstar/usb/core/inode.c: creating file '001'
ehci_hcd: block sizes: qh 60 qtd 128 itd 256 sitd 96
In ehci_hcd_mstar_drv_probe
Mstar-ehci-3 H.W init
Titania3_series_start_ehc start
Mstar-ehci-3 Mstar-ehci-3.2: Mstar EHCI
drivers/mstar/usb/core/inode.c: creating file '003'
Mstar-ehci-3 Mstar-ehci-3.2: new USB bus registered, assigned bus number 3
Mstar-ehci-3 Mstar-ehci-3.2: park 0
Mstar-ehci-3 Mstar-ehci-3.2: irq 229, io mem 0xfd227200
Mstar-ehci-3 Mstar-ehci-3.2: reset command 080b02 park=3 ithresh=8 period=1024 Reset HALT
Mstar-ehci-3 Mstar-ehci-3.2: init command 010001 (park)=0 ithresh=1 period=1024 RUN
usb usb3: default language 0x0409
usb usb3: udev 1, busnum 3, minor = 256
usb usb3: New USB device found, idVendor=1d6b, idProduct=0002
usb usb3: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb3: Product: Mstar EHCI
usb usb3: Manufacturer: Linux 2.6.35.13 ehci_hcd
usb usb3: SerialNumber: mstar
```

```
usb usb3: usb_probe_device
usb usb3: configuration #1 chosen from 1 choice
usb usb3: adding 3-0:1.0 (config #1, interface 0)
hub 3-0:1.0: usb_probe_interface
hub 3-0:1.0: usb_probe_interface - got id
hub 3-0:1.0: USB hub found
hub 3-0:1.0: 1 port detected
hub 3-0:1.0: standalone hub
hub 3-0:1.0: no power switching (usb 1.0)
hub 3-0:1.0: individual port over-current protection
hub 3-0:1.0: power on to power good time: 20ms
hub 3-0:1.0: local power source is good
hub 3-0:1.0: trying to enable port power on non-switchable hub
hub 2-0:1.0: state 7 ports 1 chg 0000 evt 0000
hub 3-0:1.0: trying to enable port power on non-switchable hub
drivers/mstar/usb/core/inode.c: creating file '001'
ehci_hcd mod ld
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
usb_storage mod ld
Mstar-ehci-3 Mstar-ehci-3.2: GetStatus port 1 status 000803 sig=j CSC CONNECT
hub 3-0:1.0: port 1: status 0401 change 0001
hub 3-0:1.0: state 7 ports 1 chg 0002 evt 0000
hub 3-0:1.0: port 1, status 0401, change 0000, 480 Mb/s
==9==> hub_port_init 1
Plug in USB Port3:20
Mstar-ehci-3 Mstar-ehci-3.2: GetStatus port 1 status 00000d sig=se0 PEC PE CONNECT
usb 3-1: new high speed USB device using Mstar-ehci-3 and address 2
[irq_proc_open][347] -16
usb 3-1: default language 0x0409
usb 3-1: udev 2, busnum 3, minor = 257
usb 3-1: New USB device found, idVendor=05e3, idProduct=0608
usb 3-1: New USB device strings: Mfr=0, Product=1, SerialNumber=0
usb 3-1: Product: USB2.0 Hub
usb 3-1: usb_probe_device
usb 3-1: configuration #1 chosen from 1 choice
usb 3-1: adding 3-1:1.0 (config #1, interface 0)
hub 3-1:1.0: usb_probe_interface
hub 3-1:1.0: usb_probe_interface - got id
hub 3-1:1.0: USB hub found
hub 3-1:1.0: 4 ports detected
hub 3-1:1.0: standalone hub
hub 3-1:1.0: individual port power switching
hub 3-1:1.0: individual port over-current protection
hub 3-1:1.0: Single TT
hub 3-1:1.0: TT requires at most 32 FS bit times (2664 ns)
hub 3-1:1.0: Port indicators are supported
hub 3-1:1.0: power on to power good time: 100ms
hub 3-1:1.0: local power source is good
hub 3-1:1.0: enabling power on all ports
rtutil5572sta mod ld
hub 3-1:1.0: enabling power on all ports
drivers/mstar/usb/core/inode.c: creating file '002'
```

```
hub 3-0:1.0: state 7 ports 1 chg 0000 evt 0002
Mstar-ehci-3 Mstar-ehci-3.2: GetStatus port 1 status 00000d sig=se0 PEC PE CONNECT
hub 3-0:1.0: port 1 enable change, status 00000403
hub 3-1:1.0: port 3: status 0101 change 0001
usb 3-1: link qh256-0001/d56dd000 start 1 [1/0 us]
hub 3-1:1.0: state 7 ports 4 chg 0008 evt 0000
hub 3-1:1.0: port 3, status 0101, change 0000, 12 Mb/s
==9==> hub_port_init 3
usb 3-1.3: new high speed USB device using Mstar-ehci-3 and address 3
rt5572sta mod ld
rtusb init rtusbSTA --->
usbcore: registered new interface driver rtusbSTA
rtnet5572sta mod ld
usb 3-1.3: default language 0x0409
usb 3-1.3: udev 3, busnum 3, minor = 258
usb 3-1.3: New USB device found, idVendor=0781, idProduct=5577
usb 3-1.3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 3-1.3: Product: Firebird USB Flash Drive
usb 3-1.3: Manufacturer: SanDisk
usb 3-1.3: SerialNumber: 4C532000030128123255
usb 3-1.3: usb_probe_device
usb 3-1.3: configuration #1 chosen from 1 choice
usb 3-1.3: adding 3-1.3:1.0 (config #1, interface 0)
usb-storage 3-1.3:1.0: usb_probe_interface
usb-storage 3-1.3:1.0: usb_probe_interface - got id
scsi0 : usb-storage 3-1.3:1.0
drivers/mstar/usb/core/inode.c: creating file '003'
hub 3-1:1.0: state 7 ports 4 chg 0000 evt 0008
scsi 0:0:0:0: Direct-Access     SanDisk  Cruzer Pop       1.26 PQ: 0 ANSI: 5
sd 0:0:0:0: [sda] 31266816 512-byte logical blocks: (16.0 GB/14.9 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Mode Sense: 43 00 00 00
sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA
 sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
Poller thread started for sda, PID = 609
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case sensitive!
hid mod ld
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
usbhid mod ld
evdev mod ld
mice: PS/2 mouse device common for all mice
mousedev mod ld
hid_microsoft mod ld
[CIP_KERNEL] (1)th waiting.
[CIP_KERNEL] authentication success!!!
[CIP_KERNEL] Success!! Authuld is successfully completed.
##### send signal from USER, SIG : 15, sh(1068)->???(68) sys_kill
```