Watch and be Watched: Compromising All Smart TV Generations

Benjamin Michéle
Security in Telecommunications
Berlin Institute of Technology
Email: ben@sec.t-labs.tu-berlin.de

Andrew Karpow
Security in Telecommunications
Berlin Institute of Technology
Email: andy@mailbox.tu-berlin.de

***Abstract*—Smart TVs are slowly becoming ubiquitous in households and offices, offering an ever-growing number of features such as Internet access, media players, and built-in cameras and microphones. They are physically placed in sensitive locations and connected to trusted home and business networks.**

These TVs use the same operating systems and software stacks as regular PCs, leaving them vulnerable to similar software-based attacks. Even worse, security updates are provided much less frequently and stop completely after the TV has reached end-of-life. Furthermore, as these systems are closed, it is nearly impossible for end users to examine if the TV is vulnerable or if it has been compromised.

This paper demonstrates that Smart TVs in their current state must not be considered trustworthy and therefore pose a severe security and privacy threat. We show that the integrated media player — a feature offered on nearly every Smart TV on the market, ranging from entry level to high end models and regardless of the vendor — is highly vulnerable. We developed a practical proof-of-concept attack using a malicious video file that gives an attacker permanent, full control over the device, yet is completely undetectable by the user. Furthermore, we provide fully functional payloads for stealthily tapping into a TV's camera and microphone.

I. Introduction

Half a decade ago, TVs slowly started to incorporate additional features, turning them into so-called Smart TVs. While conventional TVs were only capable of displaying broadcasted material, Smart TVs added features such as direct media playback, Internet access, and built-in cameras. This eliminated the need to hook up computers or other additional hardware to TV sets, greatly enhancing user experience.

Apart from offering a myriad of new possibilities, Smart TVs introduce new risks to users' security and privacy. As always with new technology, most users are not aware of the associated risks. Back in the nineties, people connected directly to the Internet without firewalls, leading to millions of infected computers. Later, with the rise of wireless networks, it took years until these networks were protected adequately. Right now, Smart TVs are becoming a valuable target for criminals. This is mainly due to the following reasons:

- **Distribution:** Smart TVs are becoming increasingly widespread, rising from 67 million units shipped worldwide in 2012, to an estimated 141 million in 2015 [12].
- **Features:** The drawback of a large set of features is an increased attack surface. The use of standard libraries with known vulnerabilities makes it even easier for attackers to develop exploits.
- **Trust:** Users' lack of distrust leads to Smart TVs being connected to trusted local area networks without further security measures. This allows an attacker to connect to otherwise unreachable systems or access sensitive files on plugged in USB sticks or network-attached storage.
- **Hardware:** Smart TVs with built-in cameras and microphones are placed in sensitive environments, such as private living and bedrooms, but also conference rooms and lobbies. Permanently available powerful processing hardware on fast Internet connections further increases the attractiveness of this target.

Contribution: In this paper, we present a new practical attack vector against Smart TVs. We exploit a feature that is actually used by many TV owners: Direct media playback from (network-) attached storage. This feature is available on the vast majority of Smart TV models, regardless of age, price class, or vendor. We present a fully functional proof-of-concept (POC) attack against a popular Samsung TV model. The POC installs a permanent backdoor on the TV, giving the remote attacker full control while being invisible to the user. Furthermore, we demonstrate in detail how Samsung's 2012 mid-range and high-end models can be converted into stealthy surveillance devices, tapping into the built-in camera and microphone. Lastly, we provide an overview of different possible payloads and how an attacker could benefit from them. To the best of our knowledge, we are the first to publish a practical attack vector that can be exploited on nearly all Smart TVs and does not require physical proximity to the device.

The paper is structured as follows: First, we provide some background on the Smart TVs we compromised; this is followed by a section on Smart TV features and associated risks. Then we explain how to get local root shells on our target devices for debugging and exploit development. Section V describes our attack scenario and Section VI explains in detail how to practically launch this attack. This is followed by novel payloads such as camera and microphone tapping in Section VII. We end with related work and our conclusions.

II. Background

This section provides background information on the software and hardware of the devices we attacked.

A. FFmpeg

FFmpeg is a set of open source tools for handling multimedia data. Its main components are `ffmpeg`, a tool for transcoding, `libavcodec`, an audio/video (A/V) codec library, and `libavformat`, an A/V mux and demux library.

The `libavformat` library is used by Samsung Smart TVs to identify media files before they are passed to the proprietary player software. It is also used by some LG models.

B. TV Hardware

We thoroughly analyzed the firmware of two Samsung TVs, an LE40B650 from 2009 and a UE40ES7000 from 2012. We chose these particular models because the former model is one of the first popular Smart TVs to feature a usable built-in media player, and the latter was the most recent model to include a built-in camera and microphone when this research started. Samsung was chosen because they continue to lead the global Smart TV market, with a share of 26% in Q1 2013 [10].

The LExxB650 is a mid-range model from Samsung's 2009 line of devices. It features a 600MHz ARMv6 CPU, 292MB of RAM, and 1GB of flash memory. The TV is controlled by a large application called `exeDSP`, which runs on a Linux 2.6.18 kernel. `libavformat` version 52.23.1 is used for detection of media formats, which is a part of FFmpeg dating back to the end of 2008. The TV uses the original firmware T-CHLCIPDEUC_002007 from January 2010.

The UExxES7000 is a mid-range to high-end 3D-capable model from Samsung's 2012 line of devices. It features a 1GHz dual core ARMv7 Cortex-A9 CPU, 584MB RAM, and 2GB eMMC flash memory. In addition, it has a built-in camera and microphone, as well as Bluetooth and WiFi. It runs a Linux 2.6.35 kernel and uses `libavformat` version 52.104.0, from an FFmpeg version dating back to March 2011. The TV uses the original firmware version T-ECPDEUC-1021.1 from October 2012.

C. Comparison to Other Vendors

Nearly all Smart TVs on the market come with built-in media players. There is a high probability that these players contain vulnerabilities, as they are required to support a multitude of complex media formats. Samsung and LG use FFmpeg for media file parsing, which facilitates the process of identifying and exploiting vulnerabilities for an attacker. Other vendors are not safe, however; using completely proprietary media players complicates our attack, but does not prevent it.

D. ELF, PLT and GOT

The *System V Application Binary Interface* [8] defines the *Executable and Linkable Format* (ELF) for executable files and shared libraries. It can contain many sections, e.g., `.text` for code and `.data` for initialized data. At runtime, shared libraries are loaded to random addresses within the process address space. Functions from these libraries can be called from within the executable by jumping to a small code fragment in the *Procedure Linkage Table* (`.plt`), which jumps to the real function address loaded from the corresponding entry of the *Global Offset Table* (`.got`).

Section VI-A explains how our POC exploit overwrites a function pointer in the `.got` to hijack the program flow.

III. Smart TV Features and Risks

TVs have been around for nearly a century, posing threats to safety due to high voltage and implosion rather than to security or privacy. This changed half a decade ago, when TVs became Smart TVs and were connected to the Internet. Depending on the model and vendor, Smart TVs offer various features that could become valuable targets for attackers.

A. Apps

Functionality enhancements are provided through apps, similar to the smart phone world. Popular examples are social network as well as shopping or streaming apps. They can be installed either via online app stores or from attached USB storage. In general, apps run in a sandbox and are thus not easily exploitable. They can, however, store sensitive information, which an attacker may steal once the TV has been compromised through other means.

B. Web Browser

Previous Smart TV generations suffered from weak processing power, hence disqualifying them for serious web browsing. This has changed in the meantime, and many models come equipped with web browsers and even built-in WiFi. According to their open source web sites, Samsung, Sony, and LG use the open source WebKit engine to provide a web browser. Usability, however, has not improved significantly. As a consequence only a 10% minority of connected TV owners uses this feature [11]. Users find it more convenient to surf the web on tablets, smart phones, or notebooks. It is unlikely that criminals will target browser vulnerabilities in Smart TVs unless the user base increases.

C. Media Playback

A feature found on nearly all models from all vendors is a built-in media player for music, photos, and videos. Files can be played from attached USB mass storage or from network storage via UPNP/DLNA. Samsung has included a movie player feature[1] in nearly all Smart TV models since 2008; the same holds true for the other major vendors.

In contrast to built-in web browsers, users accept and utilize this feature. In general, videos are downloaded from the web or received from friends, placed on USB disks or network storage, and played back directly on the TV using the built-in player and interface.

Multimedia codecs are complex and every movie player has vulnerabilities. In Section V and VI we show that it is indeed feasible to exploit these vulnerabilities on Smart TVs.

[1] WiselinkPro (2008), USB2.0 Movie (2009), Connect Share Movie (2010+)

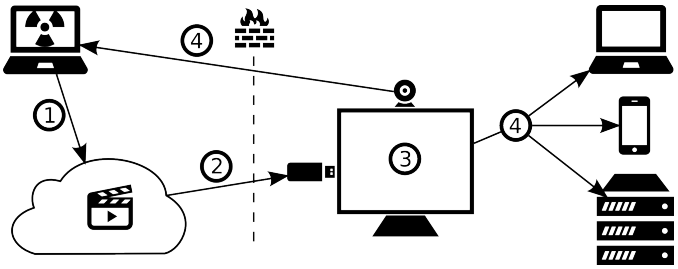

Fig. 1. Attack scenario: Distribution (1) and download (2) of exploit video, compromise (3) of TV and execution of payload (4)

D. Skype, Voice Commands, and Motion Sensing

High-end and mid-range models feature built-in or add-on cameras and microphones. They can be used for Skype conversations and for controlling the TV via voice commands and gestures. Being able to eavesdrop on conversations in living and bedrooms as well as conference rooms or offices poses a serious privacy threat.

IV. Root Shell

Having a local root shell is very important for analyzing the system and developing exploits and payloads. User accessible shells generally do not exist on Smart TVs, so obtaining them is the first step.

A. Samsung LExxB650

On this and possibly other models, a TOCTTOU attack [6] against the *Content Library* installation feature can be used to gain a root shell. Another method using nested *Content Library* folders is described on the SamyGO forum [7].

B. Samsung UExxES7000

The TV allows the installation of web applications. If they contain calls to privileged functions of Samsung's JavaScript API such as `FilePlugin.Copy`, they have to be signed by Samsung. A static JavaScript analyzer, which is run only once during installation, denies installation to unsigned applications containing privileged function calls. The analyzer, however, can be bypassed by generating the code at runtime using JavaScript's `eval()` function.

Furthermore, the `FilePlugin.Copy` function passes its arguments unfiltered to `sh -c cp -rf $a $b`, except for the characters {`;'&|"`}. This allows us to run the following JavaScript code, which in turn executes our own native code with root privileges from an attached USB drive:

```
eval("FilePlugin.Copy(
  \"\$(sh /dtv/usb/sda1/script.sh), \"\")");
```

V. Attack Scenario

Our attack exploits the Smart TV movie player feature. From an attacker's point of view, it has many desirable properties. This feature is available on virtually every Smart TV sold in the last few years. Equally important, it is not only available but actually used by many users. This yields a large amount of potentially vulnerable devices, ranging in the millions. Furthermore, the attacker does not need physical proximity to the device.

Figure 1 illustrates our attack. First, the attacker places a manipulated popular video file on the Internet or targets it directly at a specific victim (1). The victim downloads and places the file on storage connected to the TV (2). The TV is compromised as the victim starts to play back the video on the TV (3). Then the attacker's payload is executed on the TV, which, e.g., attacks other systems on the local network or transmits data from the built-in camera and microphone (4).

VI. Exploit

Playback of media files on Samsung Smart TVs works as follows. The user opens up the media player feature and selects a media file source. This can either be connected USB mass storage or a DLNA streaming server. The TV presents available movies with legitimate file extensions. The user selects one of these movies, which is then played back.

Everything on the TV is controlled by the `exeDSP` process, which consists of ~260 threads, some of which are responsible for media playback. One of them loads the FFmpeg library, which is used to identify the format of the chosen media file. Although not required by FFmpeg, the file is probed for all formats known to FFmpeg. The proprietary media player code takes FFmpeg's output and checks if the format is supported by the TV. If so, the file is analyzed again, this time by the media player itself. If no errors occur, the movie is displayed using the TV's hardware-accelerated playback. Listing 1 shows the code path in the native player.

```
av_register_all();
av_open_input_file();
puts("open␣successful");
dump_format();
seek(beginning_of_file);
start_playback_natively();
```

Listing 1. Vulnerable code path in native player

Media formats are complex, so in general there will always be exploitable bugs. The more media formats a TV supports, the more vulnerabilities will exist. Each TV uses a system on chip (SOC) responsible for media playback. In many cases the vendor of the SOC will provide the TV vendor with source code to interface to the media playback part of the SOC. It is not uncommon for this code to leak to the Internet and thereby aid attackers in finding vulnerabilities in the player software.

There is, however, an even easier way: Exploiting vulnerabilities in FFmpeg. FFmpeg is open source, so attackers can review the source code to find new vulnerabilities. Alternatively, they can leverage public bug trackers and git logs to find known vulnerabilities. Time works in favor of the attacker, as firmware updates for TVs will eventually cease and new bugs will appear, yielding all of these TVs vulnerable.

We proved that our attack is feasible by developing a POC exploit for the LExxB650. First, we identified the version of the binary FFmpeg libraries used on the target TV model. Then we searched the public bug tracker to select a vulnerability that was discovered after the release of this library version.

```
 1 int cur           = -1; // current track
 2 int track_count   =  0; // total tracks
 3 AudioTrack *tracks= NULL;
 4 for (i=0; i<header_size-8; i++) {
 5  if (fourcc_tag == strk_TAG) {
 6   cur = RL32(&header[i+8]);
 7   if (cur+1 > track_count) {
 8    track_count = cur + 1;
 9    tracks=av_realloc(tracks,track_count*20);
10   }
11   tracks[cur].adpcm     =RL32(&header[i+12]);
12   tracks[cur].channels =RL32(&header[i+36]);
13   tracks[cur].rate      =RL32(&header[i+40]);
14   tracks[cur].bits      =RL32(&header[i+44]);
15  }
16 }
```

Listing 2. `fourxm_read_header` with `strk` parsing

The next step was to write an exploit and payloads. An important task for the exploit is to provide the media player with valid movie data. Otherwise, the user might become suspicious. When the movie starts to play, payloads as described in Section VII are executed stealthily in the background.

The main process `exeDSP` runs as `root` with full privileges. The FFmpeg library is loaded into the same process space and is hence executed with the same privileges. As a result, the exploit has full privileges, too, and can, e.g., reflash the OS and applications to persist throughout power cycles.

A. 4xm File Format

4xm is a media file format that can transport a video and multiple audio streams [5]. It was developed for computer and console games, but is rarely used. The TV does not support it and should therefore not be vulnerable. However, as explained in Section VI, the TV's media player will consult FFmpeg about the file type.

1) Probing: FFmpeg will probe the file for every registered file format. To support all formats and codecs, `register_all` can be called. Alternatively, only those formats and codecs explicitly required can be registered using `register_input_format` and `register_av_codec`. From a security point of view, this is the better option. Samsung TVs register all formats, including many unused formats such as 4xm. Probing is triggered by calling `open_input_file`, which in turn calls the `read_probe` function of every registered file format. If the format is recognized, `open_input_file` calls `open_input_stream`, which fills in file information from the header by calling the file format's `read_header` function.

2) Structure: A 4xm file consists of a number of chunks. The header contains chunks defining the properties of every video and sound track. These chunks start with the four character ASCII codes `'vtrk'` and `'strk'`, respectively.

The `fourxm_read_header` function parses the file header for any occurrence of a `strk` chunk and fills in the corresponding fields. Listing 2 shows the relevant code part.

3) Vulnerability: The vulnerability was found by Tobias Klein, listed as CVE-2009-0385 and described in [4]. In line 6 of Listing 2, the current track number is read from the 4xm file header to the signed int variable `cur`. The following line contains a type conversion error: If `cur` $\geq$ `INT_MAX`, a change of sign occurs resulting in `cur` being negative. The comparison between this negative value and the zero value from `track_count` will always yield false. Therefore, the code to allocate memory in line 9 is never reached, leaving `fxm.tracks` initialized to `NULL`. This leads to an exploitable `NULL` pointer dereference in lines $11-14$. User-supplied data can be written to (`NULL` + `cur`) $*$ 20 + `x`, where `x` is the offset of the corresponding field within the tracks structure. As `cur` is user-controlled, too, arbitrary data can be written to a wide range of memory addresses.

4xm header		
strk	@unused_mem	payload[0..15]
strk	@unused_mem+16	payload[16..31]
strk	@got[puts]	@unused_mem
stage 2 payload		
benign matroska video data		

Fig. 2. Malicious 4xm file: `strk` chunks with destination address and stage 1 payload, then the address of the `.got` entry to be overwritten with pointer to stage 1 payload, finally stage 2 payload and benign matroska video data

4) Exploitation: Figure 2 illustrates a malicious video file consisting of a manipulated 4xm header, the payload, and the benign movie file the victim was expecting.

The exploit is fully reliable and can support a number of different TV models and firmware versions in a single malicious file. It leverages that `fourxm_parse_header` scans the header for all `strk` chunks. For each hit, an `AudioTrack` structure is filled with values from the malicious file. This can be exploited by placing the payload in the file in such a way that `parse_header` reassembles it to memory by filling in these structures. The final `strk` chunk overwrites the `puts` entry in the `.got`. The great advantage here is that the payload is written to a fixed address in memory, not to a variable address on the heap. The entry in the `.got` can point directly to this fixed address, resulting in a fully reliable exploit. Furthermore, multiple distinct addresses can be overwritten, providing the flexibility of having a single malicious file that is able to compromise different TV models and firmware versions. The version of FFmpeg used has a limitation of supporting a maximum of 20 audio tracks. For each audio track five consecutive 32-bit values are written to memory; four of them are user-controlled. The fifth field, `stream_index`, is initialized with zero and increased by one for each audio track. Our POC exploit is divided into two stages. The first stage payload is small and embedded into the `strk` chunks as explained above. Its sole task is to load and execute the second stage payload from the media file, which can be arbitrary in size. The first stage payload can be stripped down to 20 opcodes, which fit into five `strk` chunks. This leaves 15 chunks to overwrite `.got` entries for at least 15 different TV models or firmware versions, if needed.

5) Control Integrated Player: From an attacker's point of view, it is very important that the movie, which the user wanted

to see, is actually played back. The exploit takes this into account. `open_input_file` fills in a structure containing information on the file format, including a pointer to the demuxer. The exploit changes the demuxer from the unsupported 4xm to, e.g., matroska and seeks to the beginning of the benign movie within the file. To fill in missing information about the benign movie, `open_input_file` is called again with the modified structure. A last step is needed for playback to actually work. The integrated player, after verifying that the media format detected by FFmpeg is actually supported, seeks to the beginning of the file. Obviously, this does not work, as the file starts with the unsupported 4xm header, which would crash the TV. The exploit therefore hooks the read function call in FFmpeg's library's `.got`. The first attempt of the player to read the file results in a call that seeks to the beginning of the benign movie data, removes the hook, and subsequently reads the requested data.

6) Summary: To sum up, the following steps are necessary to construct a malicious media file that is able to compromise Smart TVs using FFmpeg. First, the FFmpeg version used by the target system has to be figured out. Using this information, FFmpeg's bug tracker, git log, or source code can be used to find exploitable vulnerabilities. As explained in Section VI-A, this can be any format supported by FFmpeg, but not necessarily by the TV. An exploit can then be developed and tested in qemu-arm and ported to the TV. A small portion of the integrated player interface to FFmpeg has to be disassembled. Then the exploit can be adapted to convince the player to actually play back the benign part of the file.

VII. Payload

This section presents some interesting payloads that can be executed after exploiting a vulnerability in the media player.

A. Camera

Starting in 2010, Samsung added camera support for their Smart TV lineup that allows users to make Skype calls or control the TV via physical gestures. The camera hardware is based on current webcam solutions using the standard Video4Linux API interface, providing h264-encoded frames at a rate of 25 frames per second.

The video stream cannot be captured directly using the Linux camera device, as it is used exclusively by the `exeDSP` process for gesture recognition. To capture the video stream unnoticed by the user, the full feature set of the TV has to be maintained. We achieved this by injecting the payload to the camera thread running on the TV with the `injectso` framework written by Sebastian Krahmer.

The vendor's shared libraries use position independent code to relocate the library on the system. It is therefore possible to hook desired functions by overwriting absolute function addresses used in the `.got` as described in Section II-D. Our injection code gets the `.got` entry of the `V4L2Capture::ReadFrame()` function and replaces it with our own shared library function. By hooking the `V4L2Capture::ReadFrame()` function, we are able to freely process the buffer contents, e.g., by sending the video stream over the Internet to achieve stealthy surveillance of the TV user. The original camera functionality of the TV is not impeded by this.

B. Microphone

Alongside the gesture support, Samsung introduced a voice control feature. The TV continuously captures the surrounding sounds, even if the voice control feature is deactivated by the user.

Linux provides a generic framework for audio input and output, called Advanced Linux Sound Architecture (ALSA). The TV software continuously polls the microphone input by executing the `snd_pcm_readi()` ALSA function with a buffer of 1200 bytes. The ALSA library function accesses the kernel's ALSA implementation and fills the buffer with data from the recording device, which is connected through generic USB sound card support.

By hooking the `snd_pcm_readi()` function that reads uncompressed audio frames from the microphone to a designated buffer, we are able to send live audio streams from the TV's microphone without quality loss or noticeable delay.

In practice, we copied the buffer passed to the `snd_pcm_readi()` function to a network socket and returned the result back to the caller. The buffer outputs pulse-code modulated audio frames with 16 kHz sample rate and 16-bit resolution in stereo. This can easily be decoded with the ALSA PCM decoder.

C. Reflash

In many cases an attacker will be interested in installing his payload permanently on the victim's TV, so that it survives a power cycle. After compromising the TV, the attacker's code runs with `root` privileges and is therefore able to reflash the TV's software. Samsung has built in a protection mechanism, a process called `authuld`. During startup, `authuld` reads a kernel-supplied random value and starts to hash each flash partition using this value as a NONCE. The resulting hash value is written back to the kernel. The kernel compares this value with the expected value and, if they don't match, reboots the TV. The expected value is read from a flash partition that contains values for all partitions. If no value is reported back to the kernel before a timeout occurs, the TV is rebooted.

An attacker can provide a modified firmware for the TV by appending it to the movie file. This firmware can then be written to flash memory by the exploit. To not brick the device, the exploit will also have to update the flash partition containing the hash keys. Details about this process are available in the SamyGO wiki [7].

D. Various

Having a compromised Smart TV in a trusted network enables the following attacks, extending the list in Section III.

1) 24/7 Permanent Backdoor: As explained in Section VII-C, the malware can be made permanent by writing a modified image to flash. The TV will then execute the attacker's payload only while the user has it turned on; however, a malicious firmware can ignore shutdown requests and instead simulate them by turning off the screen and setting the power LED appropriately.

2) Spy: A remote attacker is able to receive a live A/V stream from Smart TVs with built-in or add-on cameras and microphones, as shown in Section VII-A and VII-B. In addition to the obvious massive privacy issues, criminals can leverage this data for profit. This includes selling information to burglars, setting up video portals where people can watch other people for money, corporate espionage, or using acoustic emanations to steal passwords entered on nearby keyboards.

3) Attack Local Systems in a Trusted Network: Since the Smart TV is normally attached to a trusted local network, all systems on the same network can be attacked.

4) Offer Malicious Services: Some of the services offered by the TV are used by Smart phones, e.g., Airplay or Allshare, to send content from the phone to the TV or vice versa. If the TV is compromised, these services can exploit vulnerabilities on the smart phone using the service.

5) Exfiltrate Data: The TV has access to connected USB drives, file shares offered on the local network, and media files sent via Airplay/Allshare. Sensitive data can be collected and sent to the attacker, possibly over a long period of time.

VIII. Related Work

So far, there has only been a single academic publication [2] on Smart TV security. The authors demonstrate flaws in the implementation of Hybrid Broadcast Broadband TV (HbbTV), threatening users' privacy by leaking information about the currently watched program.

SamyGO [7] is a popular wiki and forum for owners of Samsung TVs, providing a lot of information on flash layout and protection, shell access for different TV models, etc.

SeungJin Lee recently gave talks [9] about Smart TV security, focusing mostly on software bugs that can be exploited by a local attacker or as a way for the user to get a root shell on the TV. Multiple locally exploitable bugs were presented, e.g., in the Smart TV app store installer, as well as a live video stream from the TV. In contrast, we present the first practical attack vector that does not require physical proximity to the device. Furthermore, we fully describe and provide a running POC, along with various new payloads.

Another talk [3] was given recently on vulnerabilities in the web browser and other applications of a Samsung Smart TV. There is, however, no publicly available documentation.

IX. Conclusion

In this paper we have demonstrated a new serious threat for Smart TVs. By using malicious media files as an attack vector we exploit a widely used Smart TV feature. From an attacker's point of view, this approach is very promising, as nearly all Smart TVs sold in the last half decade offer a built-in media player. Furthermore, the attack can be launched remotely and does not require physical proximity to the target. To the best of our knowledge, this is the first publication that demonstrates a practical remote attack against a wide range of Smart TVs. These devices are equipped with powerful hardware and are placed in sensitive environments, thus providing a valuable target for criminals. The number of Smart TVs in private households as well as corporate environments is rising rapidly, thereby further increasing the attractiveness of this target. Last but not least, most TV models are provided with firmware updates only for a few years. This, however, is much shorter than the average lifetime of a TV, hence eventually exposing all Smart TVs to newly discovered vulnerabilities.

We leverage that vendors have chosen to use the open source library FFmpeg, which, compared to proprietary media player software, facilitates the process of identifying and exploiting vulnerabilities. The Common Vulnerabilities and Exposures (CVE) database lists 48 vulnerabilities for FFmpeg in 2013 [1], posing a threat even to the most current TV models.

Due to a missing shell on the device, it is almost impossible for users to determine if their TV has been compromised. Our approach can be used to provide this shell access, and also to add features or fix bugs on discontinued models.

So far, the best protection for users is to examine the media file in a sandboxed environment using an up-to-date version of FFmpeg. If errors or warnings are exposed, the file may contain malware. In general, users should not trust media files to be safe, especially if they are played back on systems with vulnerable software such as Smart TVs.

We urge vendors to pay more attention to the secure handling of media files. Vendors should apply all available security fixes by using recent versions of libraries. Furthermore, instead of running with full system privileges, the corresponding code should follow the principle of least privilege.

References

[1] FFmpeg. FFmpeg security. http://www.ffmpeg.org/security.html.

[2] M. Ghiglieri, F. Oswald, and E. Tews. HbbTV - I know what you are watching. In *13. Deutscher IT-Sicherheitskongress*. SecuMedia Verlags-GmbH, May 2013.

[3] A. Grattafiori and J. Yavor. The outer limits: Hacking the Samsung smart tv. In *BlackHat USA*, July 2013. https://www.blackhat.com/us-13/archives.html#Grattafiori.

[4] T. Klein. FFmpeg type conversion vulnerability. http://www.trapkit.de/advisories/TKADV2009-004.txt.

[5] M. Melanson. 4xm format. http://wiki.multimedia.cx/index.php?title=4xm_Format.

[6] C. Mulliner and B. Michéle. Read it twice! A mass-storage-based TOCTTOU attack. In *Proceedings of the 6th Workshop on Offensive Technologies*, WOOT '12. USENIX Association, 2012.

[7] SamyGO. SamyGO, Samsung firmware on the go. http://wiki.samygo.tv.

[8] SCO. System V application binary interface. http://www.sco.com/developers/devspecs/gabi41.pdf.

[9] L. SeungJin and S. Kim. Smart tv security - #1984 in 21st century. In *CanSecWest*, March 2013. http://cansecwest.com/slides/2013/SmartTV%20Security.pdf.

[10] Strategy Analytics. Global smart tv vendor market share Q1 2013. http://www.digitaltvnews.net/content/?p=22852, July 2013.

[11] The NPD Group Blog. Internet connected tvs are used to watch tv, and that's about all. https://www.npdgroupblog.com/internet-connected-tvs-are-used-to-watch-tv-and-thats-about-all.

[12] Twice.com. IHS: Smart tvs rise to 27% of tv shipments. http://www.twice.com/articletype/news/ihs-smart-tvs-rise-27-tv-shipments/105108.