New Platform, Old Issues: How Web-based TV Broadcasts threaten Users' Security

Carlotta Tagliaro[1], Andrej Danis[1], Kevin Borgolte[2], and Martina Lindorfer[1]

[1] TU Wien, Vienna, Austria
{carlotta,martina}@seclab.wien, andrejdanis99@gmail.com
[2] Ruhr University Bochum, Bochum, Germany
kevin.borgolte@rub.de

Abstract. Hybrid digital TV, combining standard television broadcasts with Internet content, has become the de facto standard for delivering broadcast TV. One such web-based TV standard is the Hybrid Broadcast Broadband TV (HbbTV), which is used in Europe, Oceania, and parts of Asia. With millions of supported devices, and the trust users put into this traditional type of media, security is crucial. While prior work has examined privacy risks due to tracking, the security of the HbbTV protocol and its implementation remain underexplored.

In this paper, we present the first cross-vendor security analysis of HbbTV browsers on Smart TVs. We focus on three devices, from Toshiba (Android TV), Samsung (Tizen OS), and LG (WebOS), manufactured over the last nine years. We show that attackers can inject malicious HbbTV applications into broadcast streams and compromise TVs without user interaction through the built-in HbbTV browser. In particular, we show that all three TVs are vulnerable to *denial of service* (making the TV unusable), *spoofing* (replacing news banners with fake ones to spread misinformation), as well as *phishing* (tricking users into inputting sensitive information, like credentials) attacks, and that the Toshiba and LG TVs give attackers *local network access* to send web requests to other connected devices, potentially enabling attackers to move laterally within the user's network. We discuss that the root causes are two-fold: HbbTV application capabilities and outdated embedded browser runtimes. Addressing these risks requires systemic changes to both the HbbTV specification and its implementations.

1 Introduction

Television (TV) has been a defining consumer technology of the 20th and 21st centuries, evolving from monochrome to color and from analog to digital broadcasting. The latest stage of this evolution has been the widespread adoption of Smart TVs, which extend traditional TV functionality with web-based services, downloadable applications, and rich user interaction through built-in peripherals such as cameras, microphones, USB, and Bluetooth [2]. Smart TVs have become the dominant platform in the consumer market: already in 2021, over 94% of all TV sales in Germany were Smart TVs [53, 54].

Beyond the devices themselves, how broadcast content is being delivered to TVs has also fundamentally changed. Integrated Broadcast-Broadband (IBB), also known as *hybrid digital TV*, replaces traditional one-way broadcast protocols with a hybrid model that enables bi-directional communication between broadcasters and TVs using web technologies. Prominent standards deployed worldwide include (1) the *Hybrid Broadcast Broadband TV (HbbTV)* [29] in Europe, Oceania, and parts of Asia, (2) *ATSC 3.0* [7] in the US, South Korea, and Jamaica, (3) *Hybridcast* [23] in Japan, and (4) *Ginga* in Latin America [61].

HbbTV is particularly interesting from a security and privacy perspective, as it enables targeted advertisements, integrates third-party web content, and allows broadcasters to deploy interactive applications that run directly on Smart TVs. HbbTV applications can infer users' viewing habits, collect sensitive user information, for example, through TV shopping channels [55], and can overlay HTML-based content on top of the broadcast signal. Efforts to harmonize or translate applications across standards, such as between HbbTV and Hybridcast, further intensify the potential impact of vulnerabilities [57].

Despite the widespread adoption of Smart TVs and proliferation of HbbTV to deliver TV content, the security of the execution environments that run broadcast-delivered HbbTV applications on Smart TVs remains understudied. In particular, HbbTV applications run within *built-in HbbTV browsers*, distinct from the browsers users can install from app stores. These browsers operate outside the traditional web security and software update models.

Prior research has already demonstrated that TV broadcast signals can be hijacked, from early analog attacks by Captain Midnight in the 1980s [5] to more recent attacks on terrestrial Digital Video Broadcasting (DVB-T) broadcasts using lightweight aerial drones and Software-Defined Radios (SDRs) [8]. Subsequent work has focused on vulnerabilities in DVB transmission hardware combined with misuse of HbbTV [8, 47], as well as on privacy risks introduced by HbbTV-enabled tracking and fingerprinting [4, 18, 55, 56]. Prior work, however, leaves two key questions unanswered: Can an attacker abuse HbbTV-specific APIs? And what is the security posture of the built-in HbbTV browsers?

In short, HbbTV allows broadcasters to embed an application URL in the DVB stream, which Smart TVs parse and load in the built-in HbbTV browser as an overlay on top of the broadcast. The HbbTV browsers allow HbbTV applications to use standard web technologies, like HTML, CSS, and JavaScript. Still, they inherit any web-related vulnerabilities, as shown by several proof-of-concept exploits [9, 19, 45]. Many HbbTV implementations are based on outdated web browsers and standards, as documented in the protocol's developer guidelines, and are difficult to impossible to update independently of the TV firmware. In fact, the most recent HbbTV 2.0.4 specification from March 2023 sets the web platform baseline to 2021 [25]. As a result, in practice, vulnerabilities persist long after patches are available upstream in conventional browsers. Moreover, HbbTV-specific APIs, such as application lifecycle management or device control interfaces, introduce additional, new attack vectors that have received little to no attention. Given the large-scale deployment of HbbTV [26], flaws in these APIs

can affect millions of devices and TV viewers. While similar concerns may apply to other hybrid TV standards such as ATSC 3.0 and Hybridcast, we focus on HbbTV as a representative and widely deployed case study.

We present the first systematic, cross-vendor security analysis of the *built-in HbbTV browsers* that execute broadcast-delivered applications. Unlike prior work that focused on DVB transmission attacks or privacy implications, we examine the browser layer itself and show how the combination of outdated browsers, legacy web features, and HbbTV-specific APIs creates a new and dangerous attack surface. We prototype a "malicious" HbbTV application an attacker would use, inject it into a live broadcast in a controlled environment, and demonstrate that end-to-end attacks on three representative Smart TV platforms, Toshiba (Android TV), Samsung (Tizen), and LG (webOS), are practical. We investigate the following research questions:

RQ1: *Can an attacker exploit HbbTV applications by misusing APIs?*
We show that attackers can abuse both HbbTV-specific APIs (e.g., application management) and standard web features to gain control of the TV, render it unusable, or use it as an entry point into the local network.

RQ2: *To what extent do firmware updates on the tested Smart TVs affect the security of the embedded HbbTV browser?*
For the tested TVs, the firmware updates that we examined did not change the security of the embedded HbbTV browsers. On the Toshiba TV, multiple firmware updates included newer Android TV versions and system security patches, but they did not update the HbbTV runtime.

RQ3: *Does the security of the HbbTV browser differ across Smart TV vendors?*
Our evaluation reveals vendor-specific differences in supported HbbTV versions and API implementations, yet shows that these differences do not fundamentally prevent our attacks.

In summary, we make the following contributions:

- We introduce the first cross-vendor analysis of the built-in HbbTV browsers across three TV vendors: Toshiba (Android TV, 2021), Samsung (Tizen, 2017), and LG (webOS, 2024). We show that these browsers lag far behind modern standards, with Toshiba reporting `Chrome 55-66` (2,087 CVEs).
- We implement practical exploits targeting the TV's HbbTV runtime. We focus on four attack classes: denial of service (DoS), spoofing, phishing, and local network access.
- We identify ecosystem failures that make these vulnerabilities persistent, including frozen HbbTV browser and protocol versions across firmware updates, and vendor-specific implementations, features, and restrictions (e.g., origin/network behavior).

Artifacts. We make our HbbTV toolkit available as open source under a permissive license to foster integration with other protocols (e.g., ATSC 3.0, Hybridcast) and HbbTV APIs that were unsupported by our test devices. It is available at: `https//github.com/SecPriv/HbbTV-attack-toolkit`.

2 Background

Following, we provide background on HbbTV, covering its specification, application distribution model, and supported features. Because HbbTV applications run in an embedded browser runtime (as HTML/JavaScript content), our analysis focuses on the security of the HbbTV browser and its web features and HbbTV-specific APIs, rather than the broader Smart TV app-store ecosystem.

2.1 HbbTV Specifications and Security

The HbbTV developer guidelines [27] state that TVs typically remain tied to their initial HbbTV version and receive limited updates over their 6–8 years lifespan. TV vendors have no economic incentive to maintain the embedded HbbTV runtime, because many models are sold at very low margins or even at a loss [38]. However, missing updates create significant security risks, as attackers can exploit known vulnerabilities in older software versions, or due to missing security features. For example, `HbbTV 1.5` [24] finally introduced requirements for HTTP over TLS, but only since `HbbTV 2.0.4` [15] is HTTPS actually required.

To separate specification functionalities from actual behavior, Table 1 summarizes the main security differences across the HbbTV versions that we investigate in this paper and connects them to our attacks. The HbbTV version defines the supported features (in theory), while the attack's feasibility depends on the vendor-specific implementation and the embedded browser. Not all TVs implement all features of their HbbTV version though (e.g., none of our TVs support the Content Download API despite it being defined in `HbbTV 1.0` [11]).

Application Information Table (AIT). The AIT is an application signaling table in the DVB stream providing key details about the HbbTV application [28] (see Listing 1 in the Appendix). It includes properties such as `version`, the application's declared version, which the TV uses to assess compatibility, and `application_type`, which indicates the type of the HbbTV application. The `url` element with the `base` property defines the application's base URL, while `initial_path` specifies its entry point. The `control_code` determines how the application is launched (see Table 3 in the Appendix).

2.2 Supported Features

HbbTV combines functionalities from the "web" and the "TV" worlds, such as:

Native JavaScript Features (Web). The HbbTV consortium does not mandate specific JavaScript APIs, leaving the implementation to Smart TV vendors [27]. Developers must account for delayed software updates (see Section 2.1), which often means targeting older software versions and ensuring backward compatibility. As a result, HbbTV specifications typically lag behind current browser platforms by roughly two years. While devices can support newer browser features, they still need to retain legacy features for compatibility, potentially exposing them to security and privacy risks. One reason for this delay

Table 1: Security-relevant HbbTV version differences in relation to our attacks. The attack columns report our results (✔ feasible, ◐ partial, ✖ not observed, ? not evaluated) on the evaluated TVs for denial of service (DoS), spoofing, phishing, and local network access.

	Specification	Observed Impact	DoS	Spoof	Phish	LNet
1.5.1	Core app-management model, Keyset input control, Origin/CORS support, TLS support (optional)	*Attacks possible* Local network access is implementation-dependent	✔	✔	✔	✖
2.0.2	Same core model, HDR video and next-gen audio, Implementation fixes	*Attack surface unchanged*	✔	✔	✔	✔
2.0.3	Same core model, Web baseline update 2013→2018, MSE/CMAF support, TLS 1.3 support	*Attacks still possible* HbbTV browser runtime shapes outcome	◐	◐	◐	✔
2.0.4	Same core model, HTTPS required for transport, New accessibility/voice APIs	Transport security may improve with HTTPS, New APIs may increase exposure due to new risks	?	?	?	?

is that several stakeholders are involved in the design of the new specification, currently 75 active members from different branches of the industry. Across our three TVs, the embedded HbbTV browser versions and supported JavaScript APIs differed, largely matching the browser version-specific APIs (see Section 4), but there might be additional limitations to these APIs. For example, our Samsung TV (see Section 4.2) blocks requests to resources of different origins.

HbbTV Specific Features (TV). HbbTV offers APIs for deeper integration with Smart TVs and broadcasts [30]. API availability and behavior vary across HbbTV versions, for example, the voice assistant API is only available in HbbTV 2.0.4. Since some APIs are optional, applications must check for their availability at run time. For instance, as mentioned earlier, none of the TVs we tested supported the optional Content Download API, introduced in HbbTV 1.0.

We rely on the following HbbTV APIs and web features for our attacks:

- **Application Management API** (*oipfApplicationManager*, HbbTV 1.0+): The Application Management API enables an application to manage its own lifecycle and visibility and, depending on the implementation, to launch or interact with other HbbTV applications. One can access the `Application` object, which provides data about the TV via `ApplicationPrivateData`, containing broadcast-related information and a debugging function for fetching available memory, and also access the `Keyset` object. The `Keyset` object defines a set of buttons (via a bit mask) that the application will respond to, overriding the TV's remote control's original behavior.
- **Configuration and Settings API** (*oipfConfiguration*, HbbTV 1.0+): The Configuration and Settings API provides access to user and system configuration. The application can access and modify some user configuration, but the

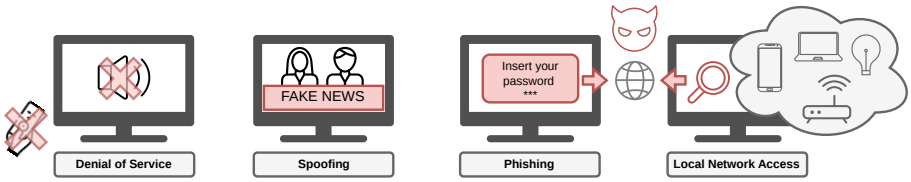

Fig. 1: Overview of our threat model and setup. The attacker injects their HbbTV application into the DVB broadcast stream, to launch denial of service, spoofing, or phishing attacks, or to access devices on the local network.

modifiable settings differ by vendor. For example, it can include the user's preferred language, country, whether subtitles are enabled, and a device ID.

– **Media Playback API** (`HbbTV 1.0`; extended in `HbbTV 1.5`): The Media Playback API builds on standard HTML5 `audio` and `video` elements but specifies TV-specific behavior and adds extensions, such as parental controls and digital rights management (DRM) integration. Using these objects, an application can replace the current broadcast presentation: a `video` element replaces the broadcast video (and associated audio), whereas an `audio` element can replace only the broadcast audio while leaving the video unchanged.

3 Threat Model

In our threat model (see Figure 1), we consider all Smart TVs supporting the HbbTV protocol, regardless of version, as targets. We assume the target TV receives the DVB stream over an unprotected medium, such as terrestrial antenna (DVB-T), a cable (DVB-C), or a satellite (DVB-S), all of which are vulnerable to signal-hijacking attacks. Attackers can then use lightweight aerial drones and software-defined radio (SDR) to hijack signals, as shown by Cabrera [8].

We adopt a similar approach to Oren et al. [47] for replacing the HbbTV URL within the DVB stream, but focus on attacks targeting the HbbTV application itself and not the web servers delivering the content. We assume the Smart TV is operating normally and is not compromised (e.g., has no malicious app installed).

We assume the attacker is aware of which TV channel the victim is watching, for example, a channel covering a major news or sports event. This is necessary, as launching an attack without user interaction requires injecting a malicious AIT into the active broadcast channel. Of course, an attacker can also launch the attack on multiple individual channels simultaneously. Further, the attacker can obtain basic information about the TV, such as the operating system, HbbTV version, and browser version, to adapt the attack (e.g., via firmware analysis).

Importantly, we assume that signal injection is feasible. Outside a lab environment, an attacker would need suitable RF equipment, a transmission setup compatible with the target, knowledge of the channel's frequency, and sufficient signal strength to overpower the legitimate broadcast. Thus, we do not claim that every attacker can mount such an attack, but rather that it is realistic under

conditions already shown in prior work. Furthermore, emerging standards, such as DVB streams delivered over the Internet (DVB-I), might provide new attack vectors for signal hijacking once they are widely deployed.

4 Smart TVs Under Test

We conduct our experiments on TVs using Tizen OS (market share 12.9% in June 2024 [10]), webOS (7.4%), and Android TV (5.9%). We initiated our evaluation with the Android-based Smart TV as the first target, as its open source nature enables deeper analysis than the other two proprietary operating systems.

4.1 Toshiba TV (Android)

We analyze a Toshiba 24WA2063DA [60] TV based on the Android TV Operating System (OS), released in 2021. We evaluated firmware versions `0.29.19.0` (based on `Android 9` with security patches until December 1st, 2021) to `2.11.0.0` (based on `Android 11` with security patches until January 1st, 2024).

To fingerprint the embedded HbbTV browser, we combined user-agent extraction with feature validation using selected JavaScript APIs introduced in different Chrome releases. This method does not identify an exact browser build. Rather, it restricts the browser to a reasonable version range coherent with the available features. Since HbbTV browsers are built-in execution environments, vendor-specific feature disabling, backporting, or additional restrictions may affect the APIs. We consider the inferred browser version as a good approximation of the underlying browser. The HbbTV web browser reports itself as `Chrome 55.0.2883.91`. This version was released on December 9th, 2016 [20], more than seven years before the security patches for the other TV applications. Without backported security patches, using an outdated Chrome version poses a significant security risk, as the missing security patches leave the browser vulnerable to known exploits. In fact, according to the Common Vulnerabilities and Exposures (CVE) database [52], this Chrome version has 2,087 known vulnerabilities. We verify the version claim by checking whether asynchronous functions [40], introduced in `Chrome 55`, are supported on our TV, which indeed they are. Conversely, the `AbortController` [39] object introduced in `Chrome 66` is not available. This narrows the likely version range down to `Chrome 55-66`. The HbbTV version reported in the user agent is `HbbTV 2.0.2`, released in 2018.

4.2 Samsung TV (Tizen)

Next, we analyze a Samsung UE75MU6170U [50] Smart TV, released in 2017, running Tizen OS. Our TV is running firmware version `T-KTMDEUC-1121.9`, the latest available at the time of writing, released in 2017.

In contrast to our Toshiba TV, the user agent does not report the Chrome version, but only the HbbTV version `HbbTV 1.5.1`, released in 2012, five years before the TV and firmware versions. Notably, `HbbTV 2.0.0` was released in 2015,

two years before the TV. We expect the HbbTV browser to be based on an even older version of Chrome than the one on our Toshiba TV. We again verify different JavaScript functionalities. We find that asynchronous functions [40] are not available, meaning the version is lower than `Chrome 55`, but the fetch API [42], introduced in `Chrome 44`, is available. Thus, the HbbTV browser is likely in the range of `Chrome 44-54`. Unfortunately, this again suggests a highly outdated version (2016 or earlier) as the base for the HbbTV browser.

Generally, this browser's functionality is very limited. We discover that the browser blocks any requests to resources from different origins, regardless of the request type or whether we use HTML to load an image or the `fetch` API. Similarly, the browser blocks redirects to URLs of different origins and web sockets. Interestingly, Errata 2 of `HbbTV 1.5.1` explicitly introduces the concept of an application origin and discusses cross-origin access and CORS semantics [12], despite the Samsung TV blocking cross-origin requests completely.

4.3 LG TV (webOS)

Finally, we test an LG UR75006LK [35] Smart TV, released in 2024, running webOS. We analyze the firmware versions `03.31.82` (webOS `8.3.1-3607`, released in June 2024) and `13.40.96` (webOS `8.4.0-2001`, released in October 2024).

The HbbTV browser's user agent reports `Chrome 94.0.4606.128`, released December 10th, 2021 [21]. While it is much newer than the other TVs' HbbTV browsers, it was still three years old when LG released the TV in August 2024. We again validate this claim with feature-based fingerprinting using JavaScript APIs expected to be present in newer Chrome versions. While such checks cannot exclude vendor-specific variations, the observed behavior is compatible with a newer browser baseline than on the Toshiba and Samsung TVs. The HbbTV version reported in the user agent is `HbbTV 2.0.3`, released in 2021 [27].

5 Web-based Attacks on Smart TVs

In this section, we detail how we can (mis)use HbbTV features (see Section 2) to implemented the attacks given our threat model (see Section 3 and Figure 1). We implemented two of our attacks using the HbbTV APIs as defined in the specifications, while the remaining attacks use plain JavaScript. Thus, we both assess which vulnerabilities are introduced by new HbbTV features, but we also consider issues that remain exploitable due to outdated browser versions. We show that, while the platform itself is new, the vulnerabilities remain the same.

Our attacks target two layers of the HbbTV ecosystem. Some attacks originate from the HbbTV execution model and APIs, such as application launch, screen overlays, media replacement, and remote-control interception via Keyset handling. Other attacks are enabled by the embedded browser, including outdated web versions and permissive networking behavior. We differentiate between attacks caused by HbbTV-related behavior and APIs, and those that depend on browser obsolescence or vendor runtime behavior.

5.1 Denial of Service (DoS)

The goal of DoS is to limit or block the viewing of content and interaction with the Smart TV. There are several approaches to achieving a DoS that we investigate, including native web features and dedicated HbbTV features:

Blocking Visual Content. We block visual content by creating a large image element that covers the entire screen using HTML or JavaScript. Because HbbTV applications are overlays on the broadcast, the image will cover the whole screen.

Blocking of Audio Content. Like visual content, we block audio content by creating audio elements with HTML or JavaScript. Because HbbTV applications are automatically launched and their audio tracks have priority over the broadcast audio, the TV will play our audio instead of the broadcast audio.

Blocking of Controls. We block controls using the Application Management API of HbbTV (see Section 2.2), specifically the `Keyset` object, which defines which key events it requests to receive. By defining a bit mask that covers all possible buttons, we can hook each button's press event and override its default behavior. While the behavior can differ across Smart TVs, it can still substantially inconvenience the TV viewer.

Infinite Switching between Channels. For our last DoS attack, we misuse the HbbTV Hybrid Tuner API (see Section 2.2) to switch TV channels. If we hijack two channels simultaneously, we can cause the Smart TV to switch between them in an infinite loop, resulting in a DoS.

5.2 Spoofing (Spreading Disinformation)

Spoofing attacks can prompt victims to believe false and incorrect information, most notoriously disinformation. In our case, this can, for example, be done by spoofing the audio or video content of the broadcast without the user noticing. There are, in fact, multiple ways to accomplish this. We analyze switching the audio, switching the video, and creating an overlay over the existing news bars seen on various news channels. We consider an attack on the news bar of a news channel to be particularly inconspicuous and effective, as victims typically trust the information presented by TV news channels [16, 33, 44] and news channels are often played at low volume in the background (e.g., at airports). Creating an overlaying news bar could be highly effective for spreading disinformation.

Spoofing Audio Content. Similar to our audio DoS attack, we create an audio element using HTML or JavaScript.

Spoofing Video Content. To spoof video content, we create a video element using HTML or JavaScript, just as we do with an audio element. The Smart TV automatically switches the visual content to the video element, which is prioritized over the broadcast. As not all Smart TVs behave the same for displaying video content, some additional engineering effort might be necessary to ensure the attacks succeed on specifically targeted Smart TVs.

Spoofing News Bar. Spoofing the news bar is similar to spoofing video content. Still, it requires adapting to the specific news bar the attacker wants to spoof, ensuring the styling matches the original news bar's style. In practice, additional monitoring may be needed to ensure the news bar is not displayed in TV segments that do not have one, as this could raise viewer suspicion.

5.3 Phishing

Phishing attacks deceive victims into revealing private information to attackers or into executing malicious code that compromises their devices' security. For HbbTV, we investigate the disclosure of private information. In many cases, Smart TVs display different pop-ups that require user input, such as input forms for credit card numbers on shopping channels [55]. For our attack we display a JavaScript pop-up that requests the victim's wireless network password and then exfiltrates it via an HTTP POST request. Note that our phishing evaluation is purely technical: we verify that malicious prompts render, accept input, and allow an attacker to exfiltrate data. Such an attack is realistic because HbbTV applications often display prompts and on-screen keyboards for legitimate interactions, such as login flows or shopping, which reduces the likelihood that a viewer immediately recognizes the prompt as malicious. We leave the assessment of real-world TV viewers' susceptibility to phishing for future work.

5.4 Local Network Access

Access to the victim's local network is critical, as it can enable fingerprinting and allow attackers to move laterally to other devices [51]. Due to the nature of HbbTV, where applications run automatically without user interaction, we can use the Smart TV to make web requests (e.g., HTTP, HTTPS, WebSocket, etc.) to other devices, including on the local network, if cross-origin request filtering is not in place, or we might even be able to launch cross-protocol attacks [6].

Without prior knowledge of the local network, an attacker does not know which active devices are on the victim's network and which hosts to target. We ported the browser-based network scanner by Kamkar [32] to a HbbTV application that scans private subnets and infers active hosts based on the success or failure of network interactions. In practice, the scan identifies hosts reachable from the HbbTV browser. Thus, the scan does not find all connected devices, but only those that respond. An attacker can then use the discovered hosts as candidates for next web requests. Not all web requests are possible, though, due to the Cross-Origin Resource Sharing (CORS) policy [41]. This policy blocks cross-origin web requests, meaning requests sent to origin URLs other than the website's original one are blocked. This, however, is often disabled in IoT devices to enable interaction via RESTful APIs [49], making them a viable target. In fact, the HbbTV specification itself requires TVs to relax these protections for some TV services: For the UPnP device description and the DIAL REST service, terminals *shall* allow cross-origin requests and authorize requests from any origin (e.g., by returning `Access-Control-Allow-Origin: *`) [14].

It is important to emphasize that this local network access attack is covert and occurs entirely in the background without the victim noticing. This gives the attacker a much larger time window for the attack, namely the duration that the victim watches the broadcast, than for other, overt attacks. The attack would only be detectable by analyzing the Smart TV's outgoing network traffic.

6 Exploiting Smart TVs via HbbTV

6.1 Hijacking DVB Signals

Assembling the Attacking Broadcast. We use TSDuck [34] to create our DVB attack broadcast, building on a recording of an existing DVB stream. First, we extract the Service Description Table (SDT) from the broadcast recording to list the available services (channels) in the DVB stream. Next, we select a target service and use the Program Association Table (PAT) to obtain the PID of its Program Map Table (PMT), which we then extract from the stream. The PMT provides information about the selected service and its associated elementary streams. From the PMT, we determine the PID of the AIT for the chosen service and extract the corresponding AIT (see Listing 1 in the Appendix).

After we extract the AIT, we set the `url base` property to the URL of our malicious HbbTV application, and the `initial_path` property to the path where the application should start. We then inject the modified AIT into the broadcast stream. We have now assembled a new DVB stream recording with injected custom AIT pointing to our malicious HbbTV application.

Modulating the DVB Stream. For our modified DVB stream to play on a TV, we need to modulate it and send the signal to the TV via antenna, cable, or satellite. Various DVB modulators are readily available, but most are relatively expensive. Luckily, in recent years, there has been substantial development in the less expensive amateur radio market [58]. We use a *UT-100 USB DVB-T Modulator* [31], which is available online (e.g., eBay) for approximately US$200. We then stream our broadcast from a file and loop it indefinitely. We then only need to connect the transmission cable to the TV antenna port and run a channel scan, after which our channel can be viewed on the TV. In our test environment, we do not receive any other streams. In practice, launching such an attack outside a lab environment requires additional effort, as an attacker must transmit on the target broadcast's frequency and compete with the legitimate signal (e.g., by leveraging features like Single Frequency Networks in DVB-T2 [13]).

6.2 Hosting HbbTV Applications

We host our HbbTV attack application on a web server, allowing the TV to access the entry-point HTML file directly through a previously defined URL. To correctly parse the HbbTV application, we send the HTML file with the `Content-Type` header set to `application/vnd.hbbtv.xhtml+xml`. Other than this, the process is akin to typical web development and deployment.

6.3 HbbTV Attack Toolkit

Our modular HbbTV attack toolkit consists of a server module written in Python and Flask, and a client module written in HTML, CSS, and JavaScript. To run the client on the TV, we create a custom AIT with the URL to our server. After switching to our modulated channel on the TV, we observe an incoming connection on the server. We then communicate via WebSockets. Our server provides a console-based interface for interacting with the Smart TV to:

1. **Show Client Info:** Show information about the client, including the client number, session ID, IP address, information about the client's browser, and information about the network (if a local network scan was performed).
2. **Request Client Configuration:** Fetch the HbbTV configuration object, `oipfConfiguration`, return it to the server, and store it.
3. **Request Client appObject**: Get the `oipfApplicationManager` object (see Section 2.2), send it to the server, and store it.
4. **Start DoS:** Launch a DoS attack on the client TV by blocking all remote control buttons, displaying a TV test pattern on the whole screen, showing a rolling banner with a message about the broadcast blockage, and playing a pre-recorded audio message on repeat (see Section 5.1).
5. **Show Fake Banner:** Show false information using a fake news banner with custom text overlaying the actual broadcast (see Section 5.2).
6. **Start Phishing:** Show a phishing pop-up on the client, which can be customized. The submitted sensitive data is then sent over the WebSocket to the server and recorded (see Section 5.3).
7. **Start Local Network Scan:** Scan the local network for hosts accessible from the Smart TV. The scan can use RTC for localhost detection and be restricted to specific subnets. As the scan usually takes longer, we execute it asynchronously on the TV. The client sends regular status messages over the WebSocket to the server, which are logged. After the network scan finishes, the results are sent to the server over the WebSocket (see Section 5.4).
8. **Switch Channel:** Switch to the previous/next channel.
9. **Send Web Request:** Send web requests from the Smart TV to any URLs or devices on the local network. The list of devices is pre-populated from the results of a prior local network scan. For example, for HTTP, one can specify the request method (GET, POST, PUT, etc.), set the request body, and the request headers. The Smart TV then sends the request, and the response is sent to the server over the WebSocket. The server parses and stores the response. If the request fails, we record an error.
10. **JS Evaluation:** Run arbitrary JavaScript code directly on the target TV. The result is returned over the WebSocket.
11. **Redirect to URL:** Redirect the Smart TV to another URL. This can be useful to launch a different HbbTV or web application from a different URL.
12. **Reload Target:** Reload the HbbTV application on the Smart TV.

We emphasize the modularity of our toolkit, and adding new attacks or features only requires limited engineering effort. We make our prototype available as open source: https//github.com/SecPriv/HbbTV-attack-toolkit.

Table 2: Vulnerability of the tested Smart TVs to different types of attacks (● successful, ◐ partially successful). The local network access failed on the Samsung TV due to a lack of features in its HbbTV browser.

Attack Type	**Toshiba TV** Android (2021)	**Samsung TV** Tizen (2017)	**LG TV** webOS (2024)
Denial of Service (DoS)	●	●	◐
Spoofing	●	●	●
Phishing	●	●	●
Local Network Access	●	N/A	●

7 Empirical Results

In this section, we discuss our results on the vulnerability of the Smart TVs under test (see Section 4) to DoS, spoofing, and phishing attacks, as well as whether they facilitate access to devices on the local network. Table 2 summarizes our results. We further analyze how different firmware versions affect the security of the HbbTV implementation. We limit our scope to the updates available for the tested devices and do not characterize vendor update policies in general.

7.1 Toshiba TV - Android OS (2021)

We successfully launch and interact with our malicious HbbTV application on the Toshiba TV (supporting `HbbTV 2.0.2`) using plain HTTP.

Denial of Service. All DoS attacks are successful. We can completely block the broadcast's video content, and we can also block audio content by playing arbitrary audio. Moreover, we can block controls, disabling buttons like "RED-GREEN-YELLOW-BLUE," video controls, the directional selector, and for the next/previous channel. However, numeric buttons, volume controls, and the power button retain their functions. Last, we can perform the attack to switch channels indefinitely. After some iterations, the Smart TV stops displaying the actual broadcast and only shows basic channel information.

Spoofing. All spoofing attacks are successful. We can replace the broadcast's audio and video, and overlay custom bars and banners via HTML or JavaScript.

Phishing. The TV correctly focuses on our input fields, enabling phishing and allowing victims to type using an on-screen keyboard and submit data that we can exfiltrate. However, phishing attacks are complex social engineering attacks, and their real-world effectiveness is highly situational. Our evaluation focuses on technical feasibility rather than user susceptibility.

Local Network Access. We can successfully scan the local network, identify reachable hosts, and then make HTTP requests to those local devices.

Post Firmware Update. For all firmware updates available for this TV, including an upgrade from `Android 9` to `Android 11`, all attacks remained successful. Despite changes to the underlying operating system and security patch level, we did not observe any changes in the HbbTV runtime behavior and our fingerprinting did not show a change in the HbbTV browser version.

7.2 Samsung TV - Tizen (2017)

Our older Samsung Smart TV only supports `HbbTV 1.5.1`, limiting some of our attacks. Our attack toolkit requires WebSockets, which are not supported. Thus, we craft custom HTML and JavaScript pages for launching our attacks against the Samsung TV. Generally, because the HbbTV browser is based on a severely outdated version of Chrome, we had to use multiple outdated, partially deprecated APIs for this TV, which are of limited use for analyzing more recent TVs. Like the Toshiba TV, this TV runs HbbTV application over HTTP.

Denial of Service. All DoS attacks are successful. We can block visual and audio content, and we can turn off the "extras," next/previous channel, and directional selector buttons. The infinite channel-switching attack is also effective.

Spoofing. We can replace broadcast audio and video and create fake news bars and banners using HTML or JavaScript.

Phishing. The attack is successful, we can display an arbitrary phishing pop-up prompting the victim to enter sensitive data, which is then sent to our server.

Local Network Access. We are unable to launch attacks on the local network. The webscan [32] library we adapted fails due to missing APIs in older Chrome versions, and strict same-origin policies block HTTP requests to local devices. This behavior is in contrast with the HbbTV specification, which discusses application origin and cross-origin access, as it requires that TVs implement permissive CORS policies for specific services (e.g., UPnP/DIAL) [12, 14]. As we cannot create HTTP requests to servers other than the one hosting the HbbTV application, HTTP requests to devices on the local network are blocked. Thus, this TV cannot be misused to interact with local network devices.

7.3 LG TV - webOS (2024)

This Smart TV supports `HbbTV 2.0.3` and runs the most recent Chrome-based HbbTV browser of our TVs, which is based on `Chrome 94` from December 2021 (Section 4.3). As with previous cases, launching an HbbTV application over an unprotected HTTP connection posed no issues.

Denial of Service. Blocking visual content by overlaying an image works as intended, as does blocking remote control buttons, including numeric buttons, and the same ones as for the Toshiba TV. By forcing focus on an input field with JavaScript, we can turn off all buttons except the power button, which restores control. The infinite channel-switching attack also works. However, we cannot

readily block audio via the HbbTV application, possibly due to `Chrome 66`'s autoplay policy change [3], which now requires user interaction to play media.

Spoofing. Spoofing attacks are partially successful. We can overlay fake news bars and banners. However, directly replacing broadcast audio or video with native HTML5 `audio`/`video` elements was not successful, likely due to autoplay policy restrictions. Alternative embedding mechanisms can still achieve spoofing though. For example, we were able to load a YouTube player in an `iframe` [22] with the autoplay parameter enabled, which starts playback automatically. This playback replaced the broadcast audio and interrupted the broadcast video.

Phishing. As with the other TVs, our phishing attack works as we intended.

Local Network Access. Access to the local network is entirely unrestricted from the embedded HbbTV browser's perspective. We can scan the local network and send arbitrary web requests to all reachable hosts.

Post Firmware Update. We observed no differences between firmware versions and the TV still reports the same HbbTV browser version across updates.

8 Responsible Disclosure & Ethics

We conducted our research in accordance with our institutions' ethical guidelines, aiming to improve the security and privacy of Smart TV users. We conducted all experiments in controlled environments using devices we owned. We did not interact with production broadcast infrastructure, consumer devices outside our lab, or end-users in any way, ensuring no harm during our study. After we identified the vulnerabilities we described in this paper, we initiated a coordinated, responsible disclosure process with the vendors: In early September 2025, we emailed coordinated reports to Samsung, LG, and Toshiba, providing full technical details. Toshiba redirected us to Hisense, the company now responsible for Toshiba-branded TVs, and we continued disclosure with Hisense's security team.

9 Related Work

Security of Digital Video Broadcasting (DVB). DVB is the standard for digital TV streaming in large parts of the world. Researchers have successfully hijacked signals, forcing Smart TVs to play hijacked streams and interact with malicious HbbTV apps injected into them. Claverie et al. [9] discuss how easy and inexpensive it can be to hijack a DVB signal caused by the lack of authentication in terrestrial DVB streams. Likewise, Oren et al. [47] demonstrated that DVB signal hijacking enables a large-scale exploitation technique with a minimal budget (around US$450 in 2015) and is difficult to detect.

Security and Privacy of HbbTV. HbbTV is part of DVB and allows content providers to define an HbbTV application URL in the DVB stream, which is then parsed and automatically opened by the TV's built-in HbbTV browser as

an intentional overlay over the original broadcast. HbbTV has previously been exploited by Claverie et al. [9] and Oren et al. [46, 47], and we adapted the HbbTV hijack technique of Oren et al. to replace the URL in the broadcast stream. Ghiglieri et al. [18] reviewed the security and privacy impact of HbbTV, highlighting vulnerabilities in older devices, insecure data transmissions, and unauthorized data collection. They also examined broadcasters' methods for tracking users and collecting viewing data through third-party analytics services. Tagliaro et al. [55, 56] investigated the privacy risks of HbbTV in Europe, focusing on its use for Internet-based content delivery and user tracking. In their study, they found that HbbTV's bi-directional nature allows broadcasters to request and receive sensitive data without user consent. Cabrera [8] showed how an attacker can inject their own HbbTV URL, for example, to mine cryptocurrency in the background of a broadcast.

Security of Smart TVs. Tileria and Blasco [59] analyzed the security and privacy of the Android TV ecosystem by examining over 4,500 apps across app stores. They found that many apps collect sensitive data and share it with tracking and ad services, and that TV apps generally have poorer data protection practices than mobile apps. In a similar study on over 3,100 Android TV apps from the Google Play Store, Liu et al. [37] also urged for more scrutiny of Android TV apps. On the firmware level, Aafer et al. [1] proposed a dynamic fuzzing technique to detect anomalies in Smart TV systems. Analyzing 11 popular Android TV boxes, they discovered 37 vulnerabilities, including memory corruptions, boot environment issues, and sensitive data leaks. Yiwei et al. [62], who introduced EvilScreen, show that vulnerabilities in TVs' remote-control communications can allow attackers to mimic remote controls, bypass authentication, and access or control TVs.

Web Browser Security. Lim et al. [36] analyzed the security of modern web browsers, focusing on Chrome, Firefox, Safari, and Edge, and compared their designs, vulnerabilities, exploitation techniques, and defenses, such as sandboxing. Nicula and Zota [43] highlight the need for comprehensive testing, including fuzzing, to uncover non-trivial vulnerabilities. Pradeep et al. [48] studied Android browsers on smartphones from global app stores and found that many browsers disclose sensitive data, such as geolocation and browsing history. Likewise, Franken et al. [17] studied integrated web browsers across a broad range of consumer products, including Smart TVs. They found that many rely on outdated browser engines, in some cases already obsolete at the time of product release. Similar to our work, they combine user-agent information with feature-based fingerprinting to infer browser versions and show that browser updates are often coupled to larger software releases.

In this paper, we bridge the gap between web browser and Smart TV security. We have shown that the way HbbTV browsers and HbbTV-specific APIs operate makes Smart TVs vulnerable to new attacks, as well as to well-known, largely mitigated web-based vulnerabilities and attacks, and that this is endemic to the HbbTV browser ecosystem.

10 Discussion

Our study shows that HbbTV-enabled Smart TVs are vulnerable to a variety of attacks, new and old, for two main reasons: (1) the HbbTV ecosystem exposes powerful TV functionalities and APIs, and (2) vendors implement these on top of outdated and modified browser runtimes. As a consequence, HbbTV applications enable DoS, spoofing, and phishing attacks, simply because the applications can abuse standard overlay, media, and input-control functionalities of HbbTV. On the other hand, the feasibility of other attacks, like local network access, depends on the HbbTV browser behavior and vendor implementation, as shown by the differences between the Samsung TV and the other TVs. Importantly yet unfortunately, we also observed that firmware updates often fail to update the HbbTV browser, leaving devices persistently exposed, even after the upstream browser developers addressed the security vulnerabilities.

10.1 Limitations and Future Work

One of the biggest challenges in HbbTV security is the inconsistency between the HbbTV specification and the actual implementation running on Smart TVs. The specification defines a plethora of APIs and functions, but not all of them are necessarily implemented (or correctly). One example is the Content Download API (see Section 2.2), which is mandatory if the video-on-demand feature is supported, but was unavailable on our TVs. Various others are "partially" mandatory but have multiple exceptions. Unfortunately, there is no way to know a priori which HbbTV APIs are *truly* supported by a Smart TV. Another challenge is the time it takes between the release of a specification update until it is actually implemented by vendors. For example, voice assistant integration is optional since the latest HbbTV version, `HbbTV 2.0.4` from March 2023. However, no Smart TV supporting this version was commercially available at the time of writing. Future work should investigate the implementation of the voice assistant HbbTV APIs and how commands can be registered to identify potential for misuse.

Another challenge is broader coverage of the Smart TV ecosystem. Naturally, our work is limited by the Smart TVs that we tested and their available firmware versions. The three TVs we tested are from leading vendors (Toshiba, Samsung, and LG) and cover the major Smart TV operating systems (Android TV, Tizen, and webOS), but the ecosystem's diversity means our results may not generalize more broadly. Future work should extend the evaluation to a broader set of devices, in particular from other vendors, with other operating systems, and for newer TV models. Additionally, longitudinal monitoring of firmware updates could be valuable for assessing whether vendors eventually address these issues or if they persist indefinitely. While we cannot be certain that the same issues persist for newer models, our findings suggest that they are unlikely to be limited to our devices: HbbTV specifications intentionally lag behind mainstream browsers, and the embedded HbbTV runtimes are typically not updated, independently of firmware updates. Moreover, currently, users have no mechanism to update the embedded HbbTV browser, meaning browser security depends on the version

initially shipped with the TV. Even the newest TV we tested, which was released in 2024, uses an HbbTV browser based on `Chrome 94` from 2021. Thus, we assume that newer models may remain vulnerable whenever they inherit similarly outdated browser baselines and permissive HbbTV capabilities.

Third, our work focuses on HbbTV. We did not empirically test other related protocols, such as Hybridcast in Japan or ATSC 3.0 in the US or South Korea. Given that these systems are architecturally similar to HbbTV, we expect similar vulnerabilities to affect them too. Future systematic studies of these standards would provide further insight into the vulnerabilities of hybrid TV.

While we successfully demonstrated the technical feasibility of spoofing and phishing attacks, we did not conduct human factors research to evaluate their effectiveness against real users. Future work performing such studies could yield new insights into users' susceptibility to and awareness of TV-based attacks, that is, how users perceive and respond to malicious overlays and prompts, and then guide the design of user-centric mitigations. Related work has previously shown that users trust the information presented by traditional news media, such as TV broadcasts, more than information online, for example through social media [16, 33, 44]. Future work should also examine how this perception changes once users become more aware of the shift of how TV content is delivered.

Finally, in addition to the four attack classes we already evaluated, the embedded HbbTV browsers may enable further web-based attacks. For example, the misuse of other HbbTV APIs that were not supported on our devices, or more advanced variants of local network interactions or cross-origin request abuse enabled by permissive browser behavior. Additionally, given the outdated HbbTV browser versions, there is also the immediate danger of exploiting known vulnerabilities in the HbbTV browser itself to perform remote code execution or local privilege escalation on the Smart TV itself. This could then lead to achieving persistence on the Smart TV, which enables a variety of other malware-like attacks, such as participating in a botnet, key/input logging for legitimate HbbTV applications on shopping channels, etc. Considered how outdated the HbbTV browsers are, it is highly likely that proof-of-concept exploits for the browser version exists that then merely need to be ported to the Smart TVs architecture (typically ARM or ARM64) and operating systems (Android TV, Tizen, and webOS are all Linux-based). We leave a systematic investigation and study of HbbTV browser exploitation for future work.

10.2 Mitigations and Recommendations

Our findings clearly indicate that securing the HbbTV ecosystem will require both specification safeguards and systematic changes in how vendors implement HbbTV. At the specification level, HbbTV should enforce secure-by-default behavior, rather than relying on optional best practices and allowing insecure exceptions. Specifically, the specification should mandate the use of HTTPS/TLS for all application resources, including initial entry points and dynamically loaded content, to eliminate injection attacks that can occur through plain HTTP. Any deviation from secure transport should be explicitly prohibited.

Furthermore, the API surface exposed to HbbTV applications should be reduced and restructured in accordance with the principle of least privilege. Security-sensitive features, such as remote-control interception, local network access, and microphone integration, should be accessible only through an explicit permission system, as on smartphones. This system should require applications to declare their intended use in advance and obtain explicit user authorization. Additionally, the specification should define enforcement rules, including permission revocation, and secure defaults when user consent is absent.

At the implementation level, manufacturers should decouple the HbbTV browser engine from the TV firmware lifecycle, treating it as an independently updatable component. This architectural change is urgently needed to ensure that browser security patches and compliance updates can be deployed promptly, rather than being delayed for years. Franken et al. [17] make a similar recommendation for the TV's "user browser," as updates for it are also commonly tied to major software releases rather than delivered independently with minor updates, which can leave it outdated even when the platform itself receives updates. Vendors must also strengthen the HbbTV runtime environment by enforcing strict origin isolation, applying content security policies that limit script execution and resource loading, and minimizing the exposure of device-specific APIs to untrusted applications. To address the lack of authenticity in broadcast delivery, vendors should require integrity and origin verification for broadcast-delivered applications. This could be achieved through digital signatures in a vendor- or consortium-managed trust root store, like the CA/Browser Forum, or through runtime verification mechanisms. Without these measures, attackers who can control or manipulate the broadcast channel will continue to be able to inject arbitrary applications into the HbbTV runtime.

11 Conclusion

In this paper, we bridged the gap between web and Smart TV security by showing that the HbbTV ecosystem inherits outdated web standards while simultaneously introducing powerful TV-specific APIs, creating a unique, large, and neglected attack surface. We demonstrated that attackers can hijack Smart TVs via HbbTV applications and launch web-based attacks, ranging from DoS and phishing to disinformation, without requiring any user interaction. With millions of HbbTV-enabled Smart TVs in Europe, Oceania, and Asia, these devices create an attack surface vulnerable to large-scale exploitation.

We introduced our new HbbTV attack toolkit and highlighted systemic weaknesses across TV vendors. Our results raise the question of whether compatibility with older Smart TVs should take precedence over ensuring the security and privacy of TVs. Stronger patching policies, application verification mechanisms, and updated specifications are urgently needed. Finally, our methodology provides a guide for investigating similar risks in other hybrid TV protocols in other regions, like Hybridcast or ATSC 3.0.

Acknowledgments

This work is based on research supported by the Vienna Science and Technology Fund (WWTF) and the City of Vienna [Grant ID: 10.47379/ICT19056 and 10.47379/ICT22060], the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, the Austrian Science Fund (FWF) [Grant ID: 10.55776/F8515-N], and SBA Research (SBA-K1 NGC), a COMET Center within the COMET – Competence Centers for Excellent Technologies Programme and funded by BMIMI, BMWET, and the federal state of Vienna. The COMET Programme is managed by FFG. We would also gratefully acknowledge the support of netidee (Internet Stiftung Austria). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the respective funding agencies.

Appendix

```
<AIT version="6" current="true" test_application_flag="false" application_type="0x0010">
  <metadata PID="5,601" />
  <application control_code="0x01">
    <application_identifier organization_id="0x0000001D" application_id="0x0006" />
    <transport_protocol_descriptor transport_protocol_label="0x00">
      <http>
        <url base="http://192.168.x.x:5000/" />
      </http>
    </transport_protocol_descriptor>
    <application_descriptor service_bound="true" visibility="3" application_priority="1">
      <profile application_profile="0x0000" version="1.2.1" />
      <transport_protocol label="0x00" />
    </application_descriptor>
    <application_name_descriptor>
      <language code="eng" application_name="HbbTV - Hello World" />
    </application_name_descriptor>
    <simple_application_location_descriptor initial_path="hbbtv/entry.html" />
  </application>
</AIT>
```

Listing 1: Example Application Information Table (AIT).

Table 3: HbbTV application start modalities and their codes.

Code	Meaning	Consequences
0x01	AUTOSTART	The application starts automatically
0x02	PRESENT	The application will not start automatically, may continue running
0x03	DESTROY	The application should be terminated if running
0x04	KILL	Like DESTROY but termination is immediate
0x07	DISABLED	The application should not start, and attempts to start will fail

References

[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 Security)*. Aug. 2021. URL: https://www.usenix.org/system/files/sec21-aafer.pdf.

[2] I. Alam, S. Khusro, and M. Naeem. "A Review of Smart TV: Past, Present, and Future". In: *Proceedings of the 11th International Conference on Open Source Systems and Technologies (ICOSST)*. Dec. 2017. DOI: 10.1109/ICOSST.2017.8279002.

[3] F. Beaufort. *Autoplay policy in Chrome*. Sept. 13, 2017. URL: https://developer.chrome.com/blog/autoplay (visited on 01/08/2025).

[4] C. Böttger, H. Hosseini, C. Utz, N. Demir, J. Hörnemann, C. Wressnegger, T. Hupperich, N. Pohlmann, M. Große-Kampmann, and T. Urban. "Privacy from 5 PM to 6 AM: Tracking and Transparency Mechanisms in the HbbTV Ecosystem". In: *Proceedings of the 55th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. June 2025. DOI: 10.1109/DSN64029.2025.00017.

[5] P. J. Boyer. "HBO Piracy Incident Stuns Other Satellite Users". In: *New York Times* (Apr. 29, 1984). URL: https://www.nytimes.com/1986/04/29/arts/hbo-piracy-incident-stuns-other-satellite-users.html (visited on 10/01/2024).

[6] M. Brinkmann, C. Dresen, R. Merget, D. Poddebniak, J. Müller, J. Somorovsky, J. Schwenk, and S. Schinzel. "ALPACA: Application Layer Protocol Confusion - Analyzing and Mitigating Cracks in TLS Authentication". In: *Proceedings of the 30th USENIX Security Symposium (USENIX Security)*. Aug. 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/brinkmann.

[7] Broadcast Standards Association. *Spotlight ATSC 3.0*. Apr. 1, 2020. URL: https://www.atsc.org/nextgen-tv/ (visited on 04/07/2025).

[8] P. Cabrera Camara. *SDR Against Smart TVs; URL and Channel Injection Attacks*. Aug. 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 (visited on 10/01/2024).

[9] T. Claverie, J. Esteves, and C. Kasmi. "Smart TVs: Security of DVB-T". In: *Proceedings of the 16th Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*. June 13–15, 2018. URL: https://www.sstic.org/media/SSTIC2018/SSTIC-actes/smart_tvs_security_of_dvb-t/SSTIC2018-Article-smart_tvs_security_of_dvb-t-kasmi_lopes-esteves_claverie.pdf.

[10] Z. Comeau. *Which Smart TV Operating Systems are the Most Popular?* June 19, 2024. URL: https://www.cepro.com/audio-video/displays/which-smart-tv-operating-systems-are-the-most-popular/ (visited on 12/11/2024).

[11] European Telecommunications Standards Institute (ETSI). *Hybrid Broadcast Broadband TV*. Technical Specification 102 796. Version 1.1.1. June 2010. URL: https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.01.01_60/ts_102796v010101p.pdf (visited on 12/27/2024).

[12] European Telecommunications Standards Institute (ETSI). *Hybrid Broadcast Broadband TV*. Technical Specification 102 796. Version 1.2.1 Errata 2. Aug. 2014. URL: https://www.hbbtv.org/wp-content/uploads/2015/07/TS102796-v121-errata-21.pdf (visited on 02/18/2026).

[13] European Telecommunications Standards Institute (ETSI). *Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)*. European Standard

302 755. Version 1.4.1. July 2015. URL: https://www.etsi.org/deliver/etsi_en/302700_302799/302755/01.04.01_60/en_302755v010401p.pdf (visited on 04/09/2026).

[14] European Telecommunications Standards Institute (ETSI). *Hybrid Broadcast Broadband TV*. Technical Specification 102 796. Version 1.5.1. Sept. 2018. URL: https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.05.01_60/ts_102796v010501p.pdf (visited on 02/18/2026).

[15] European Telecommunications Standards Institute (ETSI). *Hybrid Broadcast Broadband TV*. Technical Specification 102 796. Version 1.7.1. Sept. 2023. URL: https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.07.01_60/ts_102796v010701p.pdf (visited on 12/17/2024).

[16] R. Fletcher, S. Andı, S. Badrinathan, K. A. Eddy, A. Kalogeropoulos, C. Mont'Alverne, C. T. Robertson, A. Ross Arguedas, A. Schulz, B. Toff, and R. K. Nielsen. "The link between changing news use and trust: longitudinal analysis of 46 countries". In: *Journal of Communnication (JOC)* 75.1 (Feb. 2025). DOI: 10.1093/joc/jqae044.

[17] G. Franken, P. Claeys, T. Van Goethem, and L. Desmet. "Shiny Shells, Rusty Cores: A Crowdsourced Security Evaluation of Integrated Web Browsers". In: *Proceedings of the 21st Symposium On Usable Privacy and Security (SOUPS)*. Aug. 2025. URL: https://www.usenix.org/system/files/soups2025-franken.pdf.

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

[19] D. Goodin. "Smart TV hack embeds attack code into broadcast signal–no access required". In: *Ars Technica* (Mar. 31, 2017). URL: https://arstechnica.com/information-technology/2017/03/smart-tv-hack-embeds-attack-code-into-broadcast-signal-no-access-required/ (visited on 04/07/2025).

[20] Google. *Chrome for Android Update*. Dec. 9, 2016. URL: https://chromereleases.googleblog.com/2016/12/chrome-for-android-update_9.html (visited on 12/16/2024).

[21] Google. *Publish DEPS for 94.0.4606.128*. URL: https://chromium.googlesource.com/chromium/src/+/refs/tags/94.0.4606.128 (visited on 12/23/2024).

[22] Google. *YouTube IFrame-API – YouTube Player-Parameter*. URL: https://developers.google.com/youtube/player%5C_parameters (visited on 01/08/2025).

[23] H. Hamada. "Overview of the Hybridcast System". In: *Broadcast Technology* 53 (Winter 2013). URL: https://www.nhk.or.jp/strl/english/publica/bt/51/2.html (visited on 04/07/2025).

[24] HbbTV Association. *HbbTV 1.5 Specification with Errata #4 Integrated*. Tech. rep. May 30, 2017. URL: https://www.hbbtv.org/wp-content/uploads/2018/03/HbbTV-SPEC15-00001-001-specification_with_errata-integrated.pdf (visited on 12/17/2024).

[25] HbbTV Association. *HbbTV 2.0.4 Explained*. Tech. rep. SPEC-01235-002. Mar. 2023. URL: https://www.hbbtv.org/wp-content/uploads/2023/03/HbbTV-SPEC-01235-002-hbbtv204-explained.pdf (visited on 02/19/2026).

[26] HbbTV Association. *Deployment Information*. URL: https://www.hbbtv.org/deployments/ (visited on 12/09/2024).

[27] HbbTV Association. *HbbTV versions*. URL: https://developer.hbbtv.org/guide/introduction/hbbtv-versions/ (visited on 12/16/2024).

[28] HbbTV Association. *Introduction to AIT and its role in HbbTV*. URL: https://developer.hbbtv.org/guide/launching-hbbtv-applications-from-a-broadcast-channel/introduction-to-ait-and-its-role-in-hbbtv/ (visited on 12/20/2024).

[29] HbbTV Association. *Overview | HbbTV*. URL: https://www.hbbtv.org/overview/ (visited on 10/01/2024).
[30] HbbTV Association. *Programming Reference*. URL: https://developer.hbbtv.org/references-api/ (visited on 12/17/2024).
[31] HiDes,Inc. *UT-100 USB DVB-T Modulator Adaptor*. URL: https://hides.com.tw/product_cg74469_eng.html (visited on 12/20/2024).
[32] S. Kamkar. *webscan*. URL: https://github.com/samyk/webscan (visited on 12/26/2024).
[33] P. Kurz. *TVB: Local TV News Remains Valuable To Viewers, Advertisers*. Sept. 29, 2025. URL: https://www.tvtechnology.com/news/tvb-local-tv-news-remains-valuable-to-viewers-advertisers (visited on 02/09/2026).
[34] T. Lelégard. *TSDuck, The MPEG Transport Stream Toolkit*. URL: https://tsduck.io/ (visited on 12/20/2024).
[35] LG. *LG UR75 75 inch 4K Smart UHD TV 2024*. URL: https://www.lg.com/uk/tvs-soundbars/4k-uhd-tvs/75ur75006lk/ (visited on 12/23/2024).
[36] 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". Dec. 31, 2021. arXiv: 2112.15561 [cs.CR].
[37] Y. Liu, L. Li, P. Kong, X. Sun, and T. F. Bissyandé. "A First Look at Security Risks of Android TV Apps". In: *Proceedings of the 4th International Workshop on Advances in Mobile App Analysis (A-Mobile)*. Nov. 15, 2021. DOI: 10.1109/ASEW52652.2021.00023.
[38] C. Marshall. "Wondering why your smart TV has so many ads? Manufacturers are struggling to make money on hardware". In: *TechRadar* (Nov. 22, 2024). URL: https://www.techradar.com/televisions/wondering-why-your-smart-tv-has-so-many-ads-manufacturers-are-struggling-to-make-money-on-hardware (visited on 07/14/2025).
[39] Mozilla. *AbortController*. URL: https://developer.mozilla.org/en-US/docs/Web/API/AbortController (visited on 12/16/2024).
[40] Mozilla. *async function*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function (visited on 12/16/2024).
[41] Mozilla. *Cross-Origin Resource Sharing (CORS)*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (visited on 12/26/2024).
[42] Mozilla. *Fetch API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (visited on 12/17/2024).
[43] S. Nicula and R.-D. Zota. "An Analysis of Different Browser Attacks and Exploitation Techniques". In: *Proceedings of the 20th International Conference on Informatics in Economy (IE)*. May 14–15, 2021. DOI: 10.1007/978-981-16-8866-9_3.
[44] Ofcom. *TV loses its crown as main source for news*. Sept. 24, 2024. URL: https://www.ofcom.org.uk/media-use-and-attitudes/attitudes-to-news/tv-loses-its-crown-as-main-source-for-news (visited on 02/09/2026).
[45] Oneconsult AG. *Smart TV Hacking*. Mar. 22, 2017. URL: https://www.youtube.com/watch?v=bOJ_8QHX6OA (visited on 05/28/2026).
[46] Y. Oren and A. D. Keromytis. "From the Aether to the Ethernet—Attacking the Internet using Broadcast Digital Television". In: *Proceedings of the 23rd USENIX Security Symposium (USENIX Security)*. Aug. 2014. URL: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-oren.pdf.
[47] Y. Oren and A. D. Keromytis. "Attacking the Internet Using Broadcast Digital Television". In: *ACM Transactions on Information and System Security (TISSEC)* 17.4 (Apr. 2015). DOI: 10.1145/2723159.

[48] 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)*. July 10–15, 2023. DOI: 10.56553/popets-2023-0003.

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

[50] Samsung. *75" Flat UHD TV MU6170*. URL: https://www.samsung.com/ch/tvs/uhd-4k-tv/mu6170-75-inch-crystal-uhd-tv-ue75mu6170uxzg/ (visited on 12/17/2024).

[51] D. Schmidt, A. Ponticello, M. Steinböck, K. Krombholz, and M. Lindorfer. "Analyzing the iOS Local Network Permission from a Technical and User Perspective". In: *Proceedings of the 45th IEEE Symposium on Security & Privacy (S&P)*. May 2024. DOI: 10.1109/SP61157.2025.00045.

[52] SecurityScorecard. *Google Chrome 55.0.2883.87 security vulnerabilities, CVEs*. URL: https://www.cvedetails.com/version/1038448/Google-Chrome-55.0.2883.87.html (visited on 01/11/2025).

[53] Statista. *Number of households that own internet-connectable televisions in Japan in fiscal year 2023 with a forecast until 2030*. Dec. 2024. URL: https://www.statista.com/statistics/1020130/japan-internet-connected-tv-household-penetration/ (visited on 04/07/2025).

[54] Statista. *Share of internet-enabled television sets (smart TVs) in total TV sales in Germany from 2010 to 2024*. Apr. 2024. URL: https://www.statista.com/statistics/485443/smart-tv-share-of-total-tv-sales-germany/ (visited on 04/07/2025).

[55] 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)*. Feb. 2023. DOI: 10.14722/ndss.2023.24102.

[56] C. Tagliaro, F. Hahn, R. Sepe, A. Aceti, and M. Lindorfer. "Investigating HbbTV Privacy Invasiveness Across European Countries". In: *Proceedings of the 12th Workshop on Learning from Authoritative Security Experiment Results (LASER)*. Feb. 2023. DOI: 10.14722/laser-ndss.2023.24102.

[57] M. Takechi. *Harmonization between HbbTV2 and Hybridcast*. Nov. 2018. URL: https://www.hbbtv.org/wp-content/uploads/2018/11/27_Masaru-Takechi_Harmonization-between-HbbTV-2-and-Hybridcast-_Berlin-2018-R1.pdf (visited on 04/07/2025).

[58] The Great Scott Gadgets Team. *HackRF One*. URL: https://greatscottgadgets.com/hackrf/one/ (visited on 12/20/2024).

[59] 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)*. July 11–15, 2022. DOI: 10.56553/popets-2022-0092.

[60] Toshiba. *Toshiba WA20 Series*. URL: https://toshiba-tv.com/de-de/pdf/24wa2063da (visited on 12/16/2024).

[61] Wikipedia. *Ginga (middleware)*. URL: https://en.wikipedia.org/wiki/Ginga_(middleware) (visited on 02/19/2026).

[62] 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 (TDSC)* 21.4 (July–Aug. 2024). DOI: 10.1109/TDSC.2023.3286182.