Cast Away: On the Security of DLNA Deployments in the SmartTV Ecosystem

Guangwei Tian*†, Jiongyi Chen‡(✉), Kailun Yan*†, Shishuai Yang*†, and Wenrui Diao*†(✉)
*School of Cyber Science and Technology, Shandong University
{gwtian, kailun, shishuai}@mail.sdu.edu.cn, diaowenrui@link.cuhk.edu.hk
†Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University
‡National University of Defense Technology, jiongyi_chen@126.com

***Abstract*—The casting service on SmartTV has been increasingly used for home entertainment and business, given the convenience offered in media broadcast and screen sharing. Among the underlying protocols that support TV cast, DLNA (Digital Living Networking Alliance) – established by a group of tech giants – has become a prevailing standard in the consumer market. Although DLNA has launched the market for years, concerns may arise about whether its real-world deployment has been clearly understood.**

In this work, we systematically evaluate the security of DLNA deployments in the SmartTV ecosystem. Specifically, we identify a series of critical security issues in the interactions between SmartTVs and casting apps on the smartphone, ranging from non-mandatory encryption to unauthorized file access. The identified security risks can be exploited by a malicious app on the victim's phone, without requesting sensitive permissions, to launch multiple attacks, including arbitrary command execution, data theft, MITM (man-in-the-middle) attack, and DoS (denial-of-service) attack. To measure the impact of the identified security issues, we designed semi-automated analysis solutions to facilitate the measurements and conducted real-world experiments on 10 on-shelf TV boxes. The results show that most DLNA implementations of products and apps in the wild are insecure. In the end, we provide immediate improvement solutions to mitigate the identified security issues.

I. Introduction

As a new instance of the Internet of Things, the Internet-connected SmartTV with essential features (like media broadcast, content sharing, and projection) has been increasingly owned by households and offices. A recent report shows that more than 665 million homes worldwide owned a SmartTV by the end of 2020, which will rise to 1.1 billion by 2026 [29].

The growing popularity of SmartTV could be attributed to the rich functionalities backed by the wide interconnection and the board integration of the ecosystem, compared with traditional TVs that only allow access to limited channels. A prominent and unique feature is the cast service, which enables the sharing of digital media between the SmartTV and other multimedia devices like the smartphone. Users can push local streaming videos and photos on their smartphones for a larger and clearer display.

One of the most representative protocols that support the cast service of SmartTV is DLNA (Digital Living Network Alliance). It is a set of interoperability guidelines for digital media sharing in multimedia devices developed and promoted by a group of tech giants (including Microsoft, Intel, Samsung, HP, Sony, and Philips) in June 2003 [3]. The latest version – DLNA 4.0, was released in June 2016. Nowadays, considerable SmartTV and set-up box manufacturers have pre-installed the DLNA-based TV casting services on their devices.

DLNA Security. Though the DLNA service is widely deployed with SmartTVs, its security has not been brought to the forefront. As a result, we have seen some real-world security incidents. For example, an X-rated film was cast on the SmartTV in a famous hotpot restaurant by pranksters [5], but the root cause of this incident is still unknown to the public. Regarding the protection provided by the casting protocol, although the DLNA guideline details the specification about security configurations, it does not provide clear implementation and configuration instructions for developers. For instance, implementing secure authentication and authorization are non-mandatory, which makes such implementations vulnerable. On the other hand, DLNA is designed for the LAN (local area network) environment with a coarse-grained trust model – all devices on the same LAN are trusted. Different from the previous security concern on the device-level access control [30], [37], the SmartTV ecosystem brings the new security risks to the communicated devices when a malicious app is installed on a trusted smartphone [25], [21]. Further, the security of DLNA deployments in the wild has not been systematically evaluated.

Our Work. In this study, we evaluated the security of DLNA deployments in the SmartTV ecosystem by investigating the interaction between the casting app installed on the smartphone and the SmartTV device. After investigation, we identified four critical security issues: (1) *no message content protection*; (2) *unauthorized file access on phone*; (3) *inadequate authentication for SmartTV*; (4) *inadequate authentication for casting app*. Exploiting those security issues, attackers can steal confidential data, gain access to sensitive files, or perform unauthorized actions, by only installing a malicious app without sensitive permissions on the victim's smartphone.

To measure the impacts of the identified security issues, we designed semi-automated analysis solutions to analyze SmartTVs and casting apps, and conducted a series of experiments. The results show that, out of 15 devices we purchased,

10 have DLNA casting service built-in, and all of them (100%) have at least one security issue. Of the 117 casting apps that we crawled from app markets, 73 (62.4%) have at least one security issue. Also, we present two concrete attack cases to demonstrate the consequences of identified vulnerabilities.

Responsible Disclosure. The discovered vulnerabilities have been reported to the corresponding vendors and vulnerability management organizations. Currently, two reports have been confirmed with assigning `CNVD-2022-54667` (rated as **high severity**) and `CNVD-2022-34589` (rated as low severity).

Contributions. Here we summarize the main contributions:

- *New understanding.* We performed the first systematic study on the security of DLNA deployments in the SmartTV ecosystem and identified four widespread security issues. Also, we provided several practical mitigation measures.
- *Concrete attacks.* Exploiting the identified security issues, a malicious app without sensitive permissions can execute commands, control the TV, and even steal sensitive files from the victim user's smartphone.
- *Real-world measurement.* We designed a new analysis tool and conducted real-world measurement experiments. The results show that most DLNA implementations are insecure.

Roadmap. The rest of this paper is organized as follows. Section II provides the necessary background of TV cast and DLNA. Section III gives the threat model used in this work. Section IV elaborates the four security issues we identified. In Section V, we introduce the detailed design of our analysis solutions and presents the findings. Section VI presents two case studies of real-world attacks. In Section VII, we discuss some mitigation and limitations of our work. Section VIII reviews related work, and Section IX concludes this paper.

II. Background

In this section, we provide the necessary background of TV cast and the DLNA protocol.

A. SmartTV & TV Cast

SmartTV is a traditional TV with integrated Internet and interactive features, allowing users to stream music and videos, browse the Internet, and view photos [15]. The smart features could also be implemented by deploying a smart set-top box[1]. The systems of most SmartTVs and set-top boxes are built based on the Android TV OS [9] with customized features. Like smartphones, SmartTVs also allow users to install apps from TV app markets.

SmartTVs have larger screen sizes than smartphones, usually from 32 $\sim$ 75 inches. Pushing the multimedia contents like videos or photos from a small-screen smartphone to a nearby large-screen TV for display is called TV cast (also known as screen mirroring or screen projection), as shown in Figure 1. Usually, the smartphone and the SmartTV should be connected to the same LAN. On the one hand, there is a need for users to share with others in the same room or to view personal multimedia files on a larger screen. On the other hand, some video software does not have a universal VIP service on different platforms (TV and mobile phone), or the accessible content is not the same on different platforms. These reasons create a demand for TV cast.

Fig. 1: Cast smartphone to TV.

The TV cast function can be implemented through the *casting protocols*, such as AirPlay [7], Chromecast built-in (formerly Google Cast) [12], Miracast [13], and DLNA [3]. AirPlay was designed by Apple, and its deployment was originally limited to Apple devices (e.g., Apple TV) until 2018 when Apple opened up the license for the protocol. There are currently over 70 TV devices [8] that support AirPlay. Similarly, Chromecast built-in is mainly used by Google devices (e.g., Google TV) as well as some mainstream Android TVs (e.g., Xiaomi Mi TVs, Sony TVs). Over 4,000 certified Android TVs or devices currently available [11]. Miracast is a screen mirroring protocol introduced by the Wi-Fi Alliance in 2012. It currently has over 6,400 certified TV or set-top box devices [14]. The DLNA (Digital Living Network Alliance) protocol is designed to share digital media among multimedia devices, introduced in 2003. Given the first-mover advantage, openness, and support from a group of tech giants, DLNA has been recognized as one of the most popular TV cast protocols. According to the DLNA official statistics [4], there are more than 13,000 DLNA-certified TVs and set-top boxes around the world. Therefore, in this study, we focus on the security of DLNA deployments in the SmartTV ecosystem.

B. DLNA Protocol

The DLNA protocol enables media file sharing and playing between different devices within the same LAN, which is well suited for casting on TVs. Therefore, in the smart home era, DLNA finds its own place. Below we elaborate on the details of the roles and workflow in DLNA.

DLNA Roles. There are four roles involved in the DLNA deployment – DMR, DMC, DMP, and DMS, as listed below.

- *Digital Media Renderer (**DMR**)* is like a traditional TV, waiting for media data to play, and executing commands sent by DMC.
- *Digital Media Controller (**DMC**)* acts as a remote controller to control the playback like stop/play and adjusting

[1]In this paper, we do not strictly distinguish between SmartTVs and set-top boxes, because SmartTV can be treated as [set-top box + traditional TV]. For convenience, we refer to them as "SmartTV" in subsequent sections.

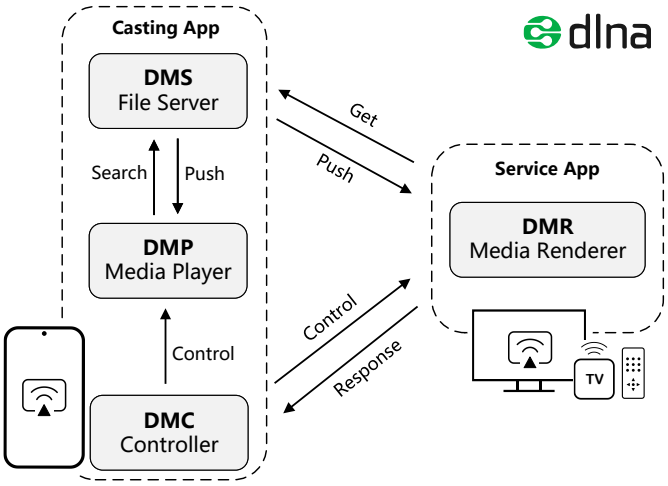

Fig. 2: Roles of DLNA in the SmartTV ecosystem.

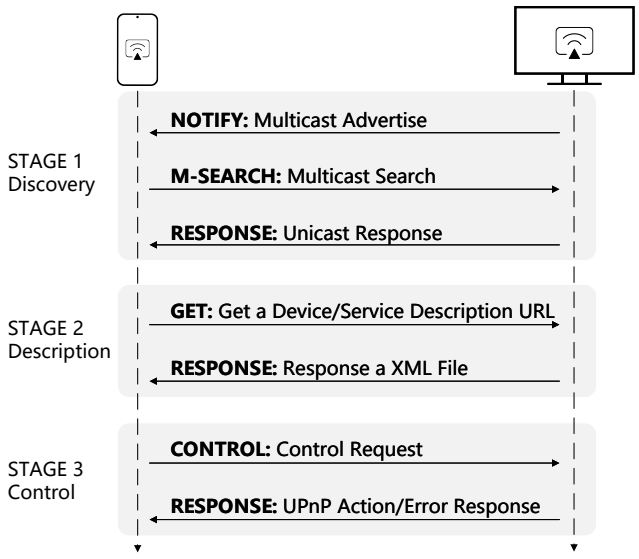

Fig. 3: Workflow of DLNA.

volumes. It is also in charge of discovering media content on DMS and sending the file to DMR.

- *Digital Media Player (**DMP**)* is the media player on the control side. Compared with DMR, DMP can actively search and play media content on DMS.
- *Digital Media Server (**DMS**)* is a file server that can provide photos, videos, or audios to DMP and DMR. Those files are accessed from the file system of the smartphone.

As shown in Figure 2, from the user's perspective, if she wants to cast a local video file (on the phone) to the TV, she needs to install a **casting app** on her phone. In fact, this casting app serves the roles of DMS (for setting up a local multimedia file server), DMP (for finding multimedia files), and DMC (for pushing files and controlling the playback of media). The **service app** on SmartTV, on the other hand, only implements the role of DMR, which is responsible for receiving media files and executing control commands. The service apps are usually pre-installed by vendors or installed by users.

Workflow. The DLNA protocol is developed based on the UPnP protocol [2]. Therefore, as shown in Figure 3, the interaction between two communication parties are similarly modeled, including the stages of *discovery*, *description*, and *control*.

- *Stage1 – Discovery.* Before casting a video to TV, the casting app needs to discover other devices in the same LAN. Usually, there are two ways to do so: the service app can send a `NOTIFY` message from time to time, actively sending out its basic information (such as device type, unique identifier, current status, and description URI) via multicast; alternatively, for the casting app, it can send an `M-SEARCH` message via multicast. The SmartTV listens to the message and responds with its basic information via unicast. With IPv4, the multicast address is `239.255.255.250`, and the port number is `1900`. All UPnP-enabled devices (or services) in the LAN listen to this port, but only the devices (or services) that match the request type in `M-SEARCH` message will reply.
- *Stage 2 – Description.* After the discovery stage, the casting app needs to request more detailed information (e.g., supported actions and the format of control command) about the discovered device and select an appropriate device. This is achieved by accessing the description URL returned by the SmartTV.
- *Stage 3 – Control.* After selecting the appropriate device, the casting app can then cast contents to it. At this stage, users can preview the media files on the DMS via DMP. The DMC then pushes the file URL to DMR through commands such as `SetAVTransportURI`. DMR can display the corresponding media files by accessing the URL. While playing, the DMC can perform various controls on the DMR through the actions acquired during the Description stage.

III. Threat Model

In the traditional threat model on LAN (local area network) security research [22], the adversary (usually as a device) can connect to the Wi-Fi AP and access other devices on this LAN. It means the adversary has obtained the password of this LAN or compromised this AP's authentication mechanism.

Our Model. Compared with previous work [30], [37], our study considers a weaker threat model, and ***all attacks are launched through a malicious app installed on the victim's Android phone***. Also, this malicious app ***does not claim any sensitive permissions***, such as storage access. Since the user owns a SmartTV, she uses a casting app on her phone to facilitate TV control and video sharing. Restricted by the Android app isolation mechanism, the malicious app cannot interact with the casting app or directly access its data. The attacker's goal is to control the TV and steal the data stored

in the phone by leveraging the malicious app and exploiting the DLNA security issues.

IV. Security Analysis of DLNA

In this work, we systematically study the security of DLNA deployments in the SmartTV ecosystem. Combined with the field study and the official technical documents [1], we identify a series of security issues related to TV casting. These issues can result in various kinds of attacks, like device control, data leakage, DoS (denial-of-service) attack, and MITM (man-in-the-middle) attack. In this section, we illustrate these identified security issues.

We focus on the communication process between the casting app and SmartTV. The establishment of secure communication relies on three critical mechanisms, namely authentication, data encryption, and authorization. Following this trail, we identify **four security issues**[2] violating these requirements, as listed in Table I.

- For *authentication*, two communication parties must authenticate each other's identity before exchanging any information. Before communication can take place, both sides of the communication need to be securely and effectively authenticated. However, most vendors do not take sufficient authentication measures, which leads to **SI#3** and **SI#4**.
- For *data encryption*, once the identities are verified, both communication parties need to encrypt the message content before it is sent out. The absence of an encryption mechanism leads to **SI#1**.
- For *authorization*, accessing resources in a remote system is also a necessary step during communication, which requires effective authorization mechanisms to guarantee legal access to file resources. The lack of an authorization mechanism presents **SI#2**.

A. *SI#1: No Message Content Protection*

Encryption is an effective solution to guarantee message confidentiality in communication.

DLNA Guideline. In the guideline, message encryption is not strictly forced and no specific encryption scheme is mentioned. It is only mentioned in the authentication section, say if the device needs to be authenticated, it can use the TLS handshake for authentication. In the UPnP specifications [2] (used in the underlying implementation of DLNA), it is mentioned that the privacy and integrity of the service can be guaranteed by the HTTPS protocol based on the TLS standard. However, it just is a recommendation, not a mandatory requirement.

Finding. The identified security issue in the wild is that DLNA messages are all transmitted in plaintext during the entire casting process. The manufacturer does not effectively secure the message contents.

Security Risk: Privacy Leakage. Due to plaintext communication, any device on the same LAN can capture the private data contained in messages. At each interaction stage, the leaked information is illustrated as follows.

At *Stage 1 – Discovery*, the device broadcasts its brief information on port 1900, and other devices on the same LAN can also actively detect all DLNA-enabled devices with the `M-SEARCH` command. Therefore, with such messages, an attacker can obtain basic information about the device and its active time.

At *Stage 2 – Description*, devices can access the description URL to obtain detailed information about the casting device. Thus, with such messages, an attacker can obtain the formats and parameters of the control commands supported by the device. Exploiting such information, the attacker can further actively forge and execute control commands at Stage 3.

At *Stage 3 – Control*, similarly, the attacker can obtain the control instructions by traffic analysis, which may contain the information of cast multimedia files, such as the file name, file description, and file resource address. Listing 1 gives an example of the partial information about the currently played video via the `GetMediaInfo` command. It means the attacker can obtain the video's title, the casting app, and even the user account information.

```
<dc:title>"Fraidy Cat"</dc:title>
<dc:creator>"youku"</dc:creator>
...
<yunos os="android"
  ver="10.1.24"
  name="youku video"
  pkg="com.youku.phone"
  yk_showtitle="Tom and Jerry"
  definitionStr="540P"
  device_model="Pixel+2"
  user_info=\'{"isVip":"0","ytid":""}\'
  ...
/>
```

Listing 1: Part of obtained video information.

B. *SI#2: Unauthorized File Access on Phone*

Authorization is needed to protect the resource files from unauthorized access. For casting, only the cast media files can be accessed by the DLNA service.

DLNA Guideline. The guideline does not explain how to access media files before they are transferred. It is only mentioned in the link protection section that the content stream in transmission need to be protected. The casting app provider should consider how to access the files by themselves.

Finding. The identified security issue in the wild is that, at Stage 3 – Control, the casting app needs to generate a file URL for DLNA service (i.e., SmartTV) accessing. However, many app developers generate the file URLs in insecure ways, resulting in unauthorized file access on the phone. Note that, such access exploits the DLNA service and bypasses the restriction of mobile OS, e.g., the `READ_EXTERNAL_STORAGE` permission on Android.

Security Risk: File Leakage. As mentioned in **SI#1**, the cast file address is transmitted in plaintext. As a result, an attacker

[2] **SI#X** for short. Therefore, we have **SI#1** $\sim$ **SI#4**.

TABLE I: Security issues related to DLNA deployments in the wild.

No.	Security Issue	Components Involved	Consequences / Possible Threats
SI#1	No message content protection	Both sides	Privacy leakage
SI#2	Unauthorized file access on phone	Casting app on phone	File leakage
SI#3	Inadequate authentication for SmartTV	Casting app on phone	Man-in-the-middle attack
SI#4	Inadequate authentication for casting app	DLNA service app on TV	Command execution, denial of service

can easily obtain the address, resulting in file leakage. Note that, in this process, the casting app sets a file server, and the DLNA service requests accessing a file by providing the corresponding file URL. It means an attacker can construct valid URLs for other files on the phone, even resulting in arbitrary file access. Therefore, how to generate the file URL is crucial for preventing URL guessing. We discovered the following URL generation schemes in the wild:

Scheme#1: *Generating URL based on file path.* File URLs can be formed by using the absolute / relative paths of files. In this case, the attacker can construct URLs for non-designated files by exhausting file paths in a directory. For example, for the casting app EasyCast[3], the attacker can access a non-casting file under `/sdcard/` directory via the URL `http://ip_addr:port/Pictures/Screenshots/Screenshot_20200220-022515.png`.

Scheme#2: *Generating URL based on file serial number.* All files on Android are assigned a serial number, and some casting apps use this number to generate the file URL. In this case, it is also possible for the attacker to access other files by traversing all numbers. Note that the upper limit of this serial number is determined by the number of multimedia files on the phone, which usually does not exceed 10^4 or 10^5. Hence, it only takes a relatively short time to traverse all the multimedia files. For example, FastCast[4], `http://ip_addr:port/image-item-40`.

Scheme#3: *Generating URL with encoding.* Some casting apps generate the URLs by encoding the absolute path to the file. The implementation can be based on Base64 or hash algorithms (e.g., MD5) on the path and then intercepting a section in the middle. In theory, for URLs of this form, if we can identify which encoding algorithm is being used, we can use the same approach to generate valid URLs for accessing other files. However, in practice, due to the various customized implementations, it is difficult to tell the accurate encoding approach. For example, Cast to TV[5], `http://ip_addr:port/Y29udGVudDovL21lZGlhL2V4dGVybmFsL2ltYWdlcy9tZWRpYS80MA==`.

Scheme#4: *Static URL.* In our study, only one casting app uses this approach to generate URLs for local files. In this case, all files share the same URL, and the returned file depends on which file is currently being pushed. That is, a fine-grained file access control mechanism is implemented. This approach is effective in preventing attacks from accessing other non-pushed files. For example, Stream Phone To TV[6], `http://ip_addr:port/`.

To sum up, **Scheme#1** & **2** are insecure and face the risk of file leakage. **Scheme#3** & **4** can resist the brute-force (file traversal) attack to some extent.

C. *SI#3: Inadequate Authentication for SmartTV*

To prevent device forgery, authentication is an effective solution in the interactions.

DLNA Guideline. Guideline provides an optional authentication mechanism. Developers can distribute credentials through CA (Credential Authority) and use the credentials for authentication, preventing attackers from forging devices. However, this feature is optional, and developers would not implement it if they believe the authentication operation is unnecessary.

Finding. The identified security issue is that most casting apps under investigation do not implement the authentication feature. It means they may establish a connection with a malicious or fake device.

Security Risk: MITM Attack. At Stage 2 – Description, the casting app does not validate the discovered devices. It only checks the `deviceType` field to confirm whether the service is a DMR, that is providing the casting service. Therefore, an attacker can provide a forged DLNA service with the same service name and description file as the origin service. From the user's perspective, she cannot distinguish which service is forged or not. Note that such a service can be constructed through a malicious app running on the user's smartphone. No physical device is needed.

Further, this malicious app can also forward the incoming messages between the casting app and the real device, say man-in-the-middle (MITM) attack. In this case, since the DLNA service works well, the user cannot find any irregularity. Also, the malicious app can record and tamper with the forwarded control commands. For example, by parsing the `SetAVTransportURI` command issued by the casting app, the attacker can access the currently played content and insert an advertisement before forwarding the command, thereby gaining revenue.

D. *SI#4: Inadequate Authentication for Casting App*

Since the device may receive malicious connection requests and control commands, authentication to the casting app (controller) is also necessary.

[3]Package name: `com.tv.cast.screen.mirroring.remote.control`
[4]Package name: `com.creative.fastscreen.phone`
[5]Package name: `com.casttotv.screenmirroring.castwebbrowser`
[6]Package name: `com.miracast.smartthing.tv.airplay.screenmirroring`

DLNA Guideline. Guideline provides an optional authentication mechanism. As described in Section IV-C, this authentication is not mandatory for device vendors to implement.

Finding. The identified security issue is that most devices under investigation do not implement the authentication feature. Only a few manufacturers implemented customized authentication mechanisms, such as whitelist.

Security Risk: Command Execution. At Stage 2 – Description, the casting app obtains the information of the device and control commands (including the parameters, formats, and control address). Afterward, the casting app can generate and send control commands to the device (SmartTV), as showen in Listing 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope s:encodingStyle="http://schemas
    .xmlsoap.org/soap/encoding/" xmlns:s="
    http://schemas.xmlsoap.org/soap/envelope
    /">
  <s:Body>
    <u:SetVolume xmlns:u="urn:schemas-upnp-
    org:service:RenderingControl:1">
      <InstanceID>0</InstanceID>
      <Channel>Master</Channel>
      <DesiredVolume>48</DesiredVolume>
    </u:SetVolume>
  </s:Body>
</s:Envelope>
```

Listing 2: The `SetVolume` command.

The control commands supported by most SmartTVs can be classified into the following three types – `AVTransport`, `Rendering Control`, and `Connection Manager` [26], as illustrated as follows.

1) `AVTransport`. This type of command enables the control of the currently played content, like: `GetMediaInfo`, `GetPositionInfo`, `GetTransportInfo`, `Pause`, `Play`, `Seek`, `SetAVTransportURI`, and `Stop`.
2) `Rendering Control`. This type of command enables the change to the TV state, like volume and image. However, most available commands are volume related (`GetMute`, `GetVolume`, `SetMute`, and `SetVolume`), and nearly no device we investigated implements the ability of image property modification, like `SetBrightness`.
3) `Connection Manager`. This type of command can get the information about the current connection or get the current protocols supported by the device. The commands of this type do not affect the casting process, like `GetCurrentConnectionInfo`, `GetProtocolInfo`, and `GetCurrentConnectionIDs`.

Most manufacturers do not introduce effective authentication mechanisms for DLNA-based casting services. Therefore, an attacker can send the above commands to a device directly. Then the device will perform the corresponding actions, such as playing any local or online media content and device volume control. As mentioned in Section I, exploiting this approach, someone cast an X-rated film to the SmartTV in a famous hotpot restaurant [5].

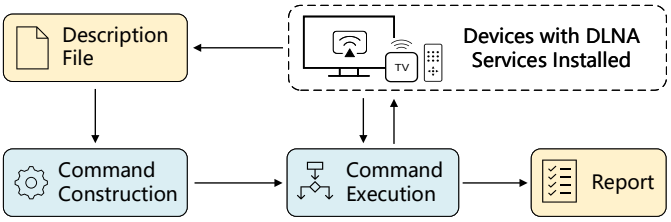

Fig. 4: Analysis of SmartTVs

Fig. 5: Devices used in our experiments.

In addition, the built-in casting services on most SmartTVs keep the ports of the casting service open permanently, including the non-screen casting periods. Some manufacturers also prefer to set their services as boot-up items. This brings a better experience to the user. However, at the same time, always-open ports inevitably result in a long attack window. An attacker can launch attacks at any time the TV is on.

V. Measurement in the Wild

To understand the real-world impacts of our reported security threats, we designed semi-automated analysis solutions to identify the security issues for the service apps of SmartTV and the casting apps of smartphone. On the one hand, we evaluate the service apps of SmartTV by testing whether control commands sent by illegitimate users can be executed (**SI#4**). On the other hand, we evaluate the casting apps on phones by testing whether they can be connected with a malicious app (**SI#3**) and whether this malicious app can further access files in smartphones via brute-forcing file URLs (**SI#2**).

A. Analysis Solutions for SmartTVs

On the SmartTV side, we focused on evaluating **SI#4** – inadequate authentication for casting app. The analysis process is illustrated in Figure 4.

Setup. To evaluate the real-world implementation, we purchased 15 TV boxes produced by mainstream vendors, as shown in Figure 5. These TV boxes were selected based on the recommendations [18]. We setup the experiments to simulate

how a normal user operates on the device: launching all the service apps on each TV box, returning to the main UI, and waiting for 5 minutes. The purpose of leaving such time is to test whether the port of casting service is closed or not when the service app is not in use. We believe 5 minutes are sufficient for a normal app to finish the procedure of closing the port. If the port is still open after 5 minutes, the casting service of the SmartTV is considered not closed on time.

Command Construction. In this step, we request the description file of the cast service and construct commands from it, which contains action name, action parameters, allowed value range, etc. The description file is requested by invoking the API `discover()` of the package `upnpy` [20] to interact with the SmartTV. `upnpy` is a lightweight Python library that implements a range of functions of UPnP protocol like device discovery and control. We show an example of the collected description file in Listing 3. It specifies the description of the `SetVolume` command, including several fields like `argument` and `allowedValueRange`. We reconstruct the command according to the description file, by specifying the action name, the arguments, and the values of arguments. The values of arguments are determined by randomly choosing a value that is within the range specified by the label `allowedValueRange`.

```
<actionList>
  ...
  <action>
    <name>SetVolume</name>
    <argumentList>
    ...
      <argument>
        <name>DesiredVolume</name>
        ...
      </argument>
    </argumentList>
  </action>
</actionList>
<serviceStateTable>
  ...
    <allowedValueRange>
      <minimum>0</minimum>
      <maximum>100</maximum>
      ...
    </allowedValueRange>
  ...
</serviceStateTable>
```

Listing 3: Partial description of the `SetVolume` command

Command Execution. We send the constructed commands to the SmartTV and receive the replied messages. When the replied message (based on the HTTP protocol) contains status code "200", the service is responding to the requests. On that basis, if the replied message indicates "error", it means that the request has been processed by the cast service but the format or the fields of the command are incorrect. For example, when a response contains error description "`Invalid Channel`", we replace the command field "`Channel`" with other options like "`LF`" and "`RF`" in the description file. When the traversal of command fields is done, and the response still indicates an error, we discard the command request and construct requests

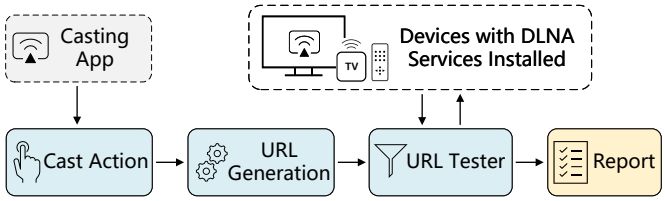

Fig. 6: Analysis of casting apps

for the next action. A request can be categorized into the following, according to its replied message:

- *Action executed.* When the replied message contains status code `200` and does not contain the keywords like "error", "fail", "invalid", "abort", or "not implemented", we consider the previous request triggers a action on the server side.
- *Action failed.* On the other hand, if the status code of the replied message is "`200`" and the message contains failure keywords like "action failed", "action not implemented", it indicates that the constructed command was received and processed, but the action was not executed for some reason.
- *Request Denied.* When the status code of the response is not "`200`", the control command is denied and not processed at all. This is attributed to the fact that the service app in the SmartTV requests authentication for the sender before acting any control actions.

We determine whether **SI#4** exists in the service app with *Action executed*: as long as the replied messages contain this indicator, there is a possibility that the device could be controlled by an unauthenticated user.

B. Analysis Solutions for Casting Apps

On the smartphone side, we evaluate the DLNA-based casting apps downloaded from Android app markets. The main idea of our analysis is to trigger the screen cast function of the casting app, extract the URLs of casting files, generate new URLs on the same smartphone and access them. The analysis process is illustrated in Figure 6.

Setup. We design a malicious Android app based on Cling [19], which is a UPnP stack for Java and Android development. This app runs on the smartphone, has a similar description file, and provides similar functions to the cast service of SmartTV. It connects to the casting app to perform unauthorized actions such as accessing files.

Using "DLNA" and "cast" as the keywords, we found a total of 337 apps from four app stores (Google Play, Wandoujia [17], Anzhi [10], and Yingyongbao [16]). Then we selected 117 apps by specifying the updated date (from 2020 to 2022) and the number of downloads (more than 1,000). Among them, 14 apps have more than 10 million downloads. Next, we ran the 117 apps one by one and manually examined the app interfaces to confirm whether they support casting local contents.

Triggering of Cast Function. Before we extract and access the URLs of casting files, we need to trigger the cast function from the casting app. As UI designs are largely varied from each other, it is difficult to trigger the cast function for each app automatically. Therefore, we manually operate the app to trigger such a function by following the steps to cast local media to a SmartTV: selecting a device to display, browsing local files, and pushing the file.

This procedure determines whether the casting app under test authenticates the malicious app (i.e., a fake device) or not. We test whether an app can successfully discover the malicious app, establish a connection with it, and push files to it. If the malicious app accepts the URL of the file to be played, it indicates that the casting app under test does not authenticate the malicious app, resulting in **SI#3**. Furthermore, if the file can be accessed by the malicious app, there is a **SI#2**.

URL Generation. In this step, we extract the file URL from `SetAVTransportURI` – a command sent by the casting app to specify file location on the smartphone. In addition to the specified casting files, we also generate new URLs to access other files, by mutating the parameters of the `SetAVTransportURI` command. The generation of new URLs is based on **Scheme#1** and **Scheme#2**[7] described in Section IV-B. For **Scheme#1**, since the files on the smartphone were available to the tester, we directly specify the file paths in the `SetAVTransportURI` command for casting. For **Scheme#2**, By increasing/decreasing the number at the end of the filename, a series of new filenames are produced. We then make access to them to confirm the existence. The access does not extend outside the `/sdcard/` directory, because the malicious app is "delegating" permissions from the casting app. The casting app does not have permissions to access other locations like the underlying Android filesystem (e.g., the `/system/` directory).

URL Access. Through the malicious app, we push the contents of the URLs to the service app on the SmartTV and check whether the contents can be accessed. The SmartTV here is used to display the contents to the tester. However, during testing, some casting apps returned the same content for the requests with different URLs. As such, we need to check if the contents on the SmartTV are changed when traversing URLs.

C. Findings on the SmartTVs

Results. As mentioned in Section V-A, the measurement was conducted on 15 TV boxes. 10 out the 15 TV boxes have DLNA-based cast service apps built-in. We found that security issues were presented in the 10 devices. None of the 10 devices authenticate casting apps (**SI#4**). The results are summarized in Table II.

For the DLNA service apps in SmartTVs, we invoke and traverse all the actions it provides to check whether the actions could be executed without authentication. Table III details the number of extracted actions and the number of actions that can be actually executed. We can see that most service apps can directly execute unauthorized actions. Those actions include `SetAVTransportURI`, `Pause`, `SetVolume` and other actions that are directly related to the current playback status.

[7]The URL generation does no cover **Scheme#3** and **Scheme#4**, as those two schemes provide a certain level of security and it is difficult to correctly specify the URL.

TABLE II: Evaluation results of SmartTVs

Device	Model	SI#3
DiyoMate	K3	✓
Mi Box	4SE	✓
Webox	WE60C	✓
Tmall Box	M20_A	✓
Tencent Aurora	Q0102	✓
Dangbei Box	DBH1A	✓
Skyworth	Q0102	✓
Mifon	HG680-KA	✓
Bell Tree	6108	✓
Magicsee	N5	✓
Chromecast	GZRNL	`N/A`
TiVo	Stream 4K	`N/A`
Fire TV	E9L29Y	`N/A`
Mecool	KM6	`N/A`
Ematic	AGT419	`N/A`

✓: security issue found; ✗: security issue not found; `N/A`: not applicable.

TABLE III: Amounts of executed and total commands

Device	Service App	AVT	RCS	CMS
DiyoMate	HiMedia Render	11/13	2/6	3/3
	KuMiao Video	11/18	9/11	3/3
Mi Box	Wireless Share	8/14	4/35	3/3
Webox	QiYiGuo TV	13/14	4/8	3/3
	KuMiao Video	11/18	9/11	3/3
Tmall Box	KuMiao Video	11/18	9/11	3/3
Tencent Aurora	System Service	8/14	4/35	3/3
	Aurora TV	13/14	35/35	3/3
Dangbei Box	System Service	12/13	2/7	3/3
	LeBo Cast	8/14	4/35	3/3
Skyworth	System Service	8/14	4/35	3/3
	KuMiao Video	11/18	9/11	3/3
Mifon	System Service	8/14	4/35	3/3
	KuMiao Video	11/18	9/11	3/3
Bell Tree	KuMiao Video	11/18	9/11	3/3
	System Service	7/13	2/6	3/3
Magicsee	System Service	9/14	2/8	3/3

AVT: AV Transport; RCS: Rendering Control; CMS: Connection Manager.

Failed Command Executions. For the actions that are not successfully executed, we summarize the cause of failures as follows:

- *Incorrect parameters or incorrect state during command execution.* Some apps require a specific value of parameter, like `DesireMute` in command `SetMute` to be 0 or 1. Filling it with other integers would lead to errors. Besides, instructions like `Previous` and `Next` require a specific status to run. For example, the `Next` instruction needs to be executed in the presence of the next playable item. Otherwise, the execution may fail.
- *Unimplemented actions.* Certain service apps recognize and accept some unimplemented actions (e.g., the actions of adjusting brightness), by simply replying with "action unimplemented".

TABLE IV: Evaluation results of casting apps

	SI#2	SI#3
# of confirmation	31	73
Percentage	55.4%	62.4%

The evaluation on **SI#3** is based on 117 casting apps, and the evaluation for **SI#2** is based on 56 casting apps that can cast local files.

TABLE V: Casting apps with different file accessing schemes.

Privilege of file access	**URL generation schemes**			
	Sch.#1	**Sch.#2**	**Sch.#3**	**Sch.#4**
Can not access any files	0	0	2	1
Can access casting file	13	0	8	1
Can access unauthorized files	24	7	0	0

- *Denied actions.* A few casting service apps (e.g., Bilibili for TV, or pre-installed casting service in Bell Tree) have authentication mechanisms and close the open port in time. As a result, the command requests are refused.

In general, there are two types of built-in service apps. The first type is the streaming app like Youku Video and Tencent Video that integrates cast service functions. The cast service of this type comes as an additional feature of streaming apps and tends not to offer abundant specialized settings. The other type is the dedicated screen casting app. For example, a customized service app called Lebo Cast is pre-installed in Mi Box and Dangbei Box. It offers more specialized features like manually confirming the connection status.

In the service app developed by Lebo Cast, mitigation to authorized access is provided. The service app identifies the casting app that is attempting to cast content to the SmartTV and requires the user to manually choose to allow or deny the connection. However, we found that this feature only restricts some of the actions in `AVTransport` class (e.g., `SetAVTransportURI`), while some actions in `RenderingControl` (e.g., `SetVolume`) can still be executed without authentication.

False Positives & False Negatives. False positives occur in the detection of **SI#4**, for QiYiGuo TV on WeBox and Aurora TV on Tencent Aurora. This is mainly attributed to the fact that some service apps on TVs do not report any errors for the received command, even if the command is not executed. The actions/commands that are not executed but do not correspond to an error reply are incorrectly classified as "action executed" by the tool. We manually confirm whether the execution is successful by checking the status and settings of the SmartTV to eliminate the false positives of the tool. Also, there is no service app that reports errors for executed commands. Thus, the tool gives no false negatives for the detection of **SI#4**.

D. Findings on the Casting Apps

Results. As mentioned earlier, the test is carried out on a total of 117 apps. We tested the 117 apps for **SI#3**. Out of the 117 apps, 56 apps can cast local multimedia files to the TV via DLNA. Thus, the 56 apps are used for the evaluation of **SI#2**. As shown in Table IV, 73 of the 117 casting apps suffer from **SI#3**, by accepting the URL from the malicious app in the control stage and successfully replying with acceptance messages. 31 out of the 56 casting apps suffer from **SI#2**, by allowing file access on the smartphone.

Table V shows that more than half of the apps (31/56) use risky file URLs, leading to unauthorized file access on the smartphone. That means a malicious app installed on the user's phone can access the files on external storage without any permissions by exhausting the URLs. There are also 13 apps that cannot access unauthorized files because only the current file's URL is set to be valid.

Only a small number of apps (10/56) use hashes or random numbers to generate file URLs (**Scheme#3**). We test it by detecting the URL format and do not generate new URLs based on the existing URL. This mitigation makes it more difficult for malicious apps to exhaust the file URLs. As for **Scheme#4**, which uses a static URL as the file URL, we cannot enumerate it, given that all files share the same URL. Therefore, only the current casting file can be accessed.

Failed File Access. File access is closely related to the configuration of the casting app on phone. There are few apps that set a very short expiry time (a few seconds) for the URL, which can cause us to time out when accessing it and thus not be able to access the casting file.

False Positives & False Negatives. As long as the malicious app can accept the URL, the casting app must have established a connection with the malicious app. Therefore there is no false positive in this experiment. False negatives in the experiment are mainly caused by the rudimentary implementation of the "malicious app". The improper implementation for app-SmartTV interaction causes the connection to interrupt due to command execution exceptions. In this situation, the casting app was classified by the tool as not having **SI#2**. In reality, the casting app does not implement authentication mechanisms.

VI. Practical Attack Cases

Exploiting the identified vulnerabilities, we present two concrete attack cases against SmartTV and smartphone, respectively.

A. Attack Case 1: Command Execution and DoS

As mentioned in **SI#4**, the DLNA services of SmartTVs tend to lack effective authentication for the control command sender, resulting in the risk of device control. Also, what is worse, after casting, the DLNA service port may not be closed on time, facilitating the attack at any time. Here we demonstrate a practical attack case.

Attack Setup. We choose Mi Box 4SE as the attack target. It has 2 pre-installed apps providing the DLNA service, LeBo Cast for TV and Tencent Video for TV. Following our threat model, we developed a malicious Android app MalApp1 without any dangerous-level permission. It achieves various DLNA control functions, and the code implementation is based on Cling [19], an open-source UPnP/DLNA library for Java and Android.

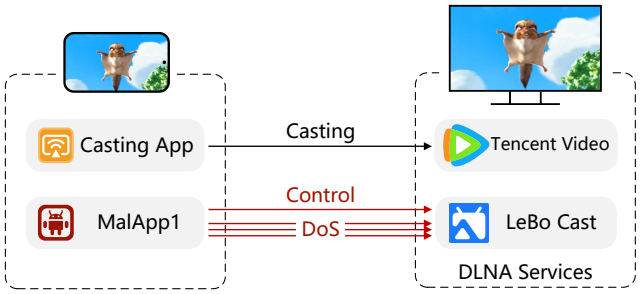

Fig. 7: Attack case: command execution.

Attack Process. We assume MalApp1 has been installed on the victim user's phone and runs in the background. At a certain time, the user is using a casting app (e.g., Youku Video[8]) on her phone to cast an online movie to her TV. If MalApp1 detects an available DLNA device appears, it starts to launch the command execution attack. To both DLNA service apps[9], MalApp1 can execute 7 kinds of control commands (i.e., `Pause`, `Play`, `Seek`, `SetAVTransportURI`, `Stop`, `SetMute`, and `SetVolume`) directly. For example, it can execute the `SetAVTransportURI` command to make the TV play another video, `SetVolume` for increasing the TV volume.

Furthermore, in addition to executing a single control command, MalApp1 also can launch a DoS attack. The DLNA service ports of 49153 (LeBo Cast for TV) and 39520 (Tencent Video for TV) are always open. Therefore, MalApp1 can send the `SetAVTransportURI` and `SetMute` commands repeatedly and frequently (e.g., at 5s intervals), resulting in the normal service not being provided. The former command brings the TV to the video playback screen repeatedly, and the latter one keeps the TV muted at all times.

Note that the TV remote control handset operations still work well during the attacks. However, even if the user exits the DLNA service apps or reboots the device, the casting service will still run in the background. On the other hand, due to heavy OS customization, the TV OS of Mi Box does not provide the function of forcing closing an app. Therefore, as a general user, she has no way to stop the DoS attack.

Impact. This case has been confirmed by CNVD with rating high severity, and an ID has been assigned: `CNVD-2022-54667`.

B. Attack Case 2: Phone Data Theft

As mentioned in **SI#2** and **SI#3**, the casting app does not authenticate the devices for connection, and the file URL generation mechanism may not be secure. It brings the security risk of data theft.

Attack Setup. We choose Fast Cast[10] as our attack target. It is a popular casting app on Google Play with over 1 million installations. Following our threat model, the attack is launched through a malicious app MalApp2 without any dangerous-level permission. Similar to the previous attack case, MalApp2 is developed based on Cling. Exploiting **SI#3**, it can provide the DLNA service like a SmartTV. This means that MalApp2 can [receive – record or modify – forward] a control command, say MITM attack. The test phone is Google Pixel 2 with Android 12.

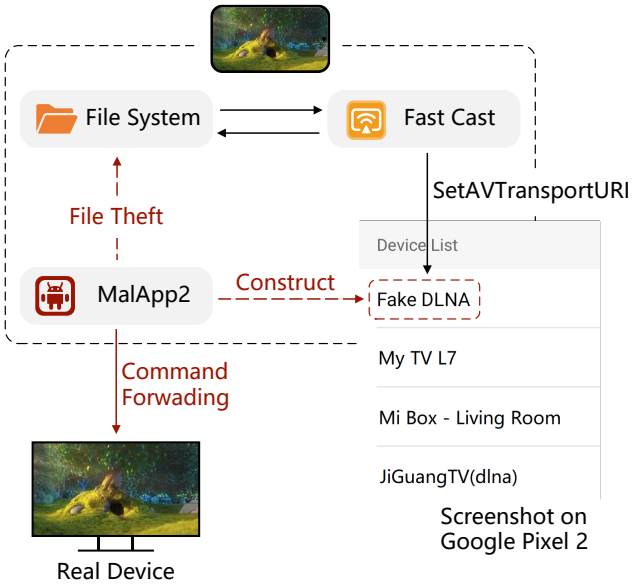

Fig. 8: Attack case: data theft.

Attack Process. As illustrated in Figure 8, at the moment of connection establishment between the user's casting app and the SmartTV, she cannot distinguish whether the listed DLNA service is provided by her TV or by a malicious app on her phone. Therefore, the user may pick the fake service provided by MalApp2 for connection. After that, during user operations, when Fast Cast sends a casting command, MalApp2 can extract the file URL from the command to steal the cast media file.

Also, other files that are not be cast still can be theft through file name traversal. Taking Fast Cast as an example, if the user wants to cast an image file named `Screenshot_20200220-022515.png` in the folder `/sdcard/Pictures/Screenshots/`, this casting app will generate a file URL based on the sequence number, like `http://192.168.123.167:8090/image-item-40`. MalApp2 can traverse the last number of this URL to access other files, like [`image-item-1` ∼ `image-item-999`]. Note that, this kind of file access is based on DLNA, that is, exploiting the `Internet` permission (normal-level), not the `READ_EXTERNAL_STORAGE` permission (dangerous-level). It is still Fast Cast that has access to the files directly. What MalApp2 needs to do is to make a network request and receive the files over the network. Therefore, it does not break the isolation mechanism of the Android OS. Following this approach, MalApp2 can access all image files on the phone.

Impact. This case has been confirmed by CNVD with rating low severity, and an ID has been assigned: `CNVD-2022-34589`.

[8]Package name: `com.youku.phone`

[9]LeBo Cast for TV provides the "anti-harassment" feature which is a kind of white-list mechanism. However, by default, this option is switched off.

[10]Package name: `com.creative.fastscreen.phone`

VII. Discussions

In this study, we identified four DLNA-related security issues in the SmartTV ecosystem and designed analysis solutions to detect them in devices and apps. Here we discuss how to mitigate these security issues and some limitations of this study.

A. Mitigation

Though the DLNA guidelines [26] design some protection mechanisms, most of them are optional and do not affect the core functions. On the other hand, the threat model of DLNA is outdated and does not consider the scenario of malicious apps on smartphones. A LAN does not mean all components are trusted, such as malicious apps on the user's phone. As a result, most vendors did not implement these protections. For each security issue, we propose the corresponding mitigation solutions with the least modifications.

To **SI#1**, a straightforward and standard solution is message encryption. PIN codes or QR codes can be used. By displaying a PIN code or QR code on the TV side, the user enters or scans the code on the mobile phone side. Then we can use the code for key generation and negotiation, and later for encrypted communication. For the exact process, we can refer to the Bluetooth pairing and authentication process [6]. Also, since the PIN codes or QR codes are displayed directly on the TV screen, the malicious app on the phone has no way of knowing the exact contents of these codes.

To **SI#2**, app developers can use secure file URL generation schemes (such as **Scheme#3** and **Scheme#4** mentioned in Section IV-B) to avoid brute-forcing file URLs. In addition, denying any requests that do not match the URL of the cast file can also prevent the leakage of non-cast files.

To **SI#3**, when the casting app establishes the connection with the discovered device, it should deny the local DLNA services (with local IPs) to prevent the attacks launched from malicious apps on the same phone. Besides, using a PIN code or scanning a QR code, as mentioned before, can also achieve the effect of authentication.

To **SI#4**, it is necessary to add an extra connection confirmation on the TV side. After receiving the request to establish a connection, a connection confirmation window can be popped up on the TV side for the user to mannualy confirm, in order to avoid malicious users taking control of the TV. In addition, whitelisting and blacklisting mechanisms are also necessary to block frequent malicious requests. For example, we can use user agents [30] to construct an access control list.

B. Limitations

Fully-automated Analysis. In Section V, we designed two analysis solutions to detect our identified security issues in casting apps and SmartTVs. However, some manual actions are still required to trigger the operations of casting apps due to the diverse code implementations. Though the workload of manual actions is slight, it affects the analysis efficiency.

For the casting app analysis, we intended to use static analysis to find the execution path of how the file URLs are generated. The endpoint of the path is easy to determine. However, there is no unified interface for DLNA, and the software vendor's code implementations are quite different. Therefore, tracing the path back to the starting point is challenging. Finally, we launched our experiments by triggering a casting action to capture the file URL. We faced a similar challenge to the automated UI click triggering – the UI designs of casting apps are quite different. Therefore, we have to trigger the casting actions in the experiments by manual clicks.

Attack Consequences. The security impacts of some identified issues are not quite serious. For example, **SI#4** can result in the command execution attack, but this attack affects the availability of services, not affecting the user's private data security. The main reason is that the DLNA is a file-sharing protocol, and the user generally does not use the TV to process sensitive data.

Other Casting Protocols. As described in Section II-B, there are various casting protocols. In this study, we focus on DLNA for its popularity. Other protocols are also deployed widely, and we plan to study them in our future work.

VIII. Related Work

Security of Smart Home. The security of the smart home is a trending topic and has attracted attention from researchers. For example, Fu et al. [28] proposed a semantics-aware anomaly detection system for smart homes. Trimananda et al. [35] presented a tool that can automatically extract packet-level signatures for device events from network traffic. Fernandes et al. [27] and Tian et al. [34] illustrate the security issues associated with weak or lack of authentication in the smart home. Zhou et al. [38] investigated the security issues in the interaction between different entities in a smart home platform. On the DLNA security, the previous work focused on designing device-level access control schemes [30], [37]. However, these schemes are not suitable for our threat model in the SmartTV ecosystem.

Security of SmartTV. Prior studies have traditionally focused on the security of physical system and firmware for SmartTVs. For example, Aafer et al. [21] developed a log-guided dynamic fuzzing technique to evaluate Android SmartTV API additions. The vulnerabilities they revealed could cause cyber threats, memory corruption, and even visual and auditory disturbances. Sitterer et al. [33] shared an approach to extract Android TV application data without root access nor any authentication. Bachy et al. [23] investigated the specific case of SmartTVs, presented a new attack path that allows remote vulnerability exploitation on smart devices, and discussed several methods to extract and analyze the embedded firmware. Privitera et al. [32] explored various security threats for SmartTV, followed by the design and development of an asset protector by considering inexpensive hardware and open-source software. Bachy et al. [24] focused on the security of communication channels for aerial TV broadcasts or between smartTVs and their service providers. Niemietz et al. [31] conducted studies on smart TV apps and found that these apps suffer from

security risks such as data leakage. Varmarken et al. [36] proposed FingerprinTV, a method for automatically extracting and evaluating network fingerprints of smart TV apps. In contrast, our work focuses on security issues when performing DLNA-based casting from smartphones to SmartTVs.

IX. Conclusion

In this paper, we conducted a systematic study on the security of DLNA deployments in the SmartTV ecosystem, focusing on the interaction between casting apps and SmartTVs. After investigation, we discovered four widely existed security issues in the wild. Further, based on the new analysis solutions, we conducted a series of experiments to measure the scope of identified issues. The results are not encouraging: 100% TVs and 62.4% casting apps have at least one security issue. The current security risks should be mitigated immediately, and further studies are needed.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. This work was partially supported by National Natural Science Foundation of China (Grant No. 61902148), Shandong Provincial Natural Science Foundation (Grant No. ZR2020MF055, ZR2021LZH007, ZR2020LZH002, and ZR2020QF045), and Taishan Scholar Program of Shandong Province, China. Jiongyi Chen was supported in part by Natural Science Foundation of Hunan Province, China (Grant No. 2022JJ40553).

References

[1] (2016) DLNA Guidelines. [Online]. Available: https://spirespark.com/dlna/guidelines/
[2] (2016) UPnP Standards & Architecture. [Online]. Available: https://openconnectivity.org/developer/specifications/upnp-resources/upnp/
[3] (2017) DLNA. [Online]. Available: https://www.dlna.org/
[4] (2017) DLNA Product Search. [Online]. Available: https://spirespark.com/dlna/products
[5] (2019) When Hotpot Gets Really Hot: Haidilao Customers Shocked by Steamy TV. [Online]. Available: https://www.whatsonweibo.com/when-hotpot-gets-really-hot/
[6] (2021) Bluetooth Core Specification. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/
[7] (2022) AirPlay. [Online]. Available: https://www.apple.com/airplay/
[8] (2022) AirPlay Enabled TVs and Devices. [Online]. Available: https://www.apple.com/ios/home/accessories/#section-tv
[9] (2022) Android TV. [Online]. Available: https://www.android.com/tv/
[10] (2022) Anzhi. [Online]. Available: http://www.anzhi.com/
[11] (2022) Certified Android TV. [Online]. Available: https://docs.google.com/spreadsheets/d/1kdnHLt673EjoAJisOal2uIpcmVS2Defbgk1ntWRLY3E/edit
[12] (2022) Chromecast built-in. [Online]. Available: https://www.google.com/chromecast/built-in/
[13] (2022) Miracast. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/miracast
[14] (2022) Miracast Certified Products. [Online]. Available: https://www.wi-fi.org/product-finder-results?sort_by=certified&sort_order=desc&capabilities=100
[15] (2022) Smart TV. [Online]. Available: https://en.wikipedia.org/wiki/Smart_TV
[16] (2022) Tencent Yingyongbao. [Online]. Available: https://sj.qq.com/
[17] (2022) Wandoujia. [Online]. Available: https://www.wandoujia.com/
[18] (2022) What Android TV Buy? Comparison, Best 2022 Android TV-Box. [Online]. Available: https://androidpctv.com/best-android-tv-box/comment-page-1/
[19] 4thline. (2018) Cling. [Online]. Available: https://github.com/4thline/cling
[20] 5kyc0d3r. (2020) upnpy. [Online]. Available: https://github.com/5kyc0d3r/upnpy
[21] 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-SEC), August 11-13, 2021*, 2021.
[22] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security Evaluation of Home-Based IoT Deployments," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy (Oakland), San Francisco, CA, USA, May 19-23, 2019*, 2019.
[23] Y. Bachy, F. Basse, V. Nicomette, E. Alata, M. Kaâniche, J. Courrège, and P. Lukjanenko, "Smart-TV Security Analysis: Practical Experiments," in *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 22-25, 2015*, 2015.
[24] Y. Bachy, V. Nicomette, M. Kaâniche, and E. Alata, "Smart-TV Security: Risk Analysis and Experiments on Smart-TV Communication Channels," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 1, pp. 61–76, 2019.
[25] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS), February 24-27, 2019*, 2019.
[26] *DLNA guidelines June 2016 release*, Digital Living Network Alliance, 2016.
[27] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy (Oakland), May 22-26, 2016*, 2016.
[28] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-Aware Anomaly Detection for Appified Smart Homes," in *Proceedings of the 30th USENIX Security Symposium (USENIX-SEC), August 11-13, 2021*, 2021.
[29] Ians. (2021) Over 665 million households own smart TVs globally, says report. [Online]. Available: https://www.business-standard.com/article/current-affairs/over-665-million-households-own-smart-tvs-globally-says-report-121072500630_1.html
[30] M. Z. Islam, M. M. Hossain, S. Haque, J. Lahiry, S. A. Bonny, and M. N. Uddin, "User-agent based Access Control for DLNA Devices," in *Proceedings of the 6th International Conference on Knowledge and Smart Technology (KST), Chonburi, Thailand, January 30-31, 2014*, 2014.
[31] M. Niemietz, J. Somorovsky, C. Mainka, and J. Schwenk, "Not so smart: On smart TV apps," in *2015 International Workshop on Secure Internet of Things, SIoT 2015, Vienna, Austria, September 21-25, 2015*, 2015.
[32] D. Privitera and H. Shahriar, "Design and Development of Smart TV Protector," in *Proceedings of the National Cyber Summit (NCS), June 5-7, 2018*, 2018.
[33] A. Sitterer, N. Dubois, and I. M. Baggili, "Forensicast: A non-intrusive approach & tool for logical forensic acquisition & analysis of the google chromecast TV," in *Proceedings of the 16th International Conference on Availability (ARES), August 17-20, 2021*, 2021.
[34] Y. Tian, N. Zhang, Y. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "SmartAuth: User-Centered Authorization for the Internet of Things," in *Proceedings of the 26th USENIX Security Symposium (USENIX-SEC), August 16-18, 2017*, 2017.
[35] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-Level Signatures for Smart Home Devices," in *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS), February 23-26, 2020*, 2020.
[36] J. Varmarken, J. A. Aaraj, R. Trimananda, and A. Markopoulou, "FingerprinTV: Fingerprinting Smart TV Apps," *Proceedings on Privacy Enhancing Technologies (PoPETs)*, vol. 2022, no. 3, pp. 606–629, 2022.
[37] Y. Wu and X. Zhi, "ARP Spoofing Based Access Control for DLNA Devices," in *Proceedings of the 2015 International Conference on Cloud and Big Data Computing, Beijing (CBDCom), China, August 10-14, 2015*, 2015.
[38] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and Understanding the Security Hazards in the Interactions between IoT Devices, Mobile Apps, and Clouds on Smart Home Platforms," in *Proceedings of the 28th USENIX Security Symposium (USENIX-SEC), August 14-16, 2019*, 2019.