

WideLeak: How Over-the-Top Platforms Fail in Android

Gwendal Patat, Mohamed Sabt, Pierre-Alain Fouque

▶ **To cite this version:**

Gwendal Patat, Mohamed Sabt, Pierre-Alain Fouque. WideLeak: How Over-the-Top Platforms Fail in Android. DSN 2022 - 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Jun 2022, Baltimore, MD, United States. hal-03637107

HAL Id: hal-03637107

https://univ-rennes.hal.science/hal-03637107

Submitted on 11 Apr 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

WideLeak: How Over-the-Top Platforms Fail in Android

Gwendal Patat
Univ Rennes, CNRS, IRISA
gwendal.patat@irisa.fr

Mohamed Sabt
Univ Rennes, CNRS, IRISA
mohamed.sabt@irisa.fr

Pierre-Alain Fouque
Univ Rennes, CNRS, IRISA
pierre-alain.fouque@irisa.fr

***Abstract*—Nowadays, most content providers rely on DRM (Digital Right Management) to protect media from illegal distribution. Becoming a major platform for streaming, Android provides its own DRM framework that does not comply with existing DRM standards. Thus, OTT (over-the-top) platforms need to adapt their apps to suit Android design, despite a fragmented ecosystem and little public documentation. Unfortunately, the security implications of how OTT apps leverage Widevine, the most popular Android DRM, have not been studied yet.**

In this paper, we report the first experimental study on the state of Widevine use in the wild. Our study explores OTT compliance with Widevine guidelines regarding asset protection and legacy phone support. With the evaluation of premium OTT apps, our experiments bring to light that most apps adopt weak and potentially vulnerable practices. We illustrate our findings by showing how to easily recover media content from many OTT apps, including Netflix.

***Index Terms*—Android, Digital Right Management, Over-The-Top, Widevine**

I. Introduction

The world almost fully moved over earlier days of owning media. Instead, many people watch videos on over-the-top platforms (OTT), such as Netflix and Amazon Prime, that distribute multimedia content over the Internet. Today, the most prevalent model is subscription-based services, where media can be played as often as the user wishes, but becomes unavailable when a user stops paying for the service. The large distribution of media in what can sometimes be hostile environments, like untrusted devices, creates challenges for content producers. The main challenge remains piracy; in 2020, the OTT market size was estimated at $13.9 billion and expected to reach $139 billion by 2028 [1].

Thus, content providers rely on DRM (Digital Right Management), which is a technology that aims to protect media from illegal distribution. Modern DRMs ship content in an encrypted form, and then control their decryption through authorized players on users' devices. DRM systems are often cried out because of their insecurity. However, the recent surge of OTT platforms has led to ongoing adoption and evolution of such systems. For instance, World Wide Web Consortium (W3C) has published, despite being criticized [2], the Encrypted Media Extensions (EME) [3], as the first official Web standard for DRM. Android devices have become a major platform for streaming content consumption. This has drawn interests from OTT platforms to develop mobile apps in order to reach a large number of audience in a thriving business ecosystem. Android has anticipated such a trend, and provided a DRM framework allowing the decryption of protected media streams. This framework supports many DRM systems; which DRM a device supports varies regarding the device manufacturer. The most dominant one is Widevine [4], which is also deployed on web browsers (eg., Chrome and Firefox), and smart TVs among others.

The ecosystem of DRM in smartphones is quite peculiar. First of all, starting from Android 7, Widevine takes advantage of hardware-backed security, such as TrustZone-based Trusted Execution Environment (TEE) [5]. In the past decade, much work has been done to evaluate the security of software-only DRM solutions based on obfuscation and cryptographic whiteboxes [6], [7]. TEE-based Widevine arguably provides stronger protection, despite some successful attacks in the literature [8]–[11]. Second, the Android DRM API, is quite different from the standardized EME specifications that are defined for the Web. Therefore, apps developers need to rewrite a large part of their code, which may lead to inconsistency in security practices. This is especially true given that Widevine public documentations are sparse, which limits community discussions and sharing. Third, any DRM system has a long history of hacking and patching. Thus, they require to be updated in a regular basis. Unfortunately, it is not common that smartphones receive a long-term support from their manufacturers. Even Google used to only guarantee a three-year support for their Pixels [12], before Pixel 6.

In order to address such concerns, Widevine has issued a set of guidelines and recommendations to follow by OTT apps. However, little has been done to explore how these apps actually implement Widevine recommendations to protect their contents. In this paper, we focus on whether the security features offered by Widevine are properly leveraged. For instance, we empirically study if apps actually encrypt their contents with Widevine keys. One might argue about the relevance of such a question, especially that OTT developers are well aware of piracy threats. However, our evaluation shows that, for example, Netflix, which is the leader app with more than one billion installations, does not even encrypt audio tracks and just delivers them in clear. Our goal is to identify missing conformance of OTT apps regarding three Widevine security guidelines: content protection, usage of encryption keys, and support of discontinued devices.

We performed our evaluation on premium OTT apps. One

might expect that Android has defined their DRM API, while taking into account Widevine guidelines, since both is owned by Google. A surprising result of our work is to uncover that there is some gap between actual implementation practices and what Widevine wants. By switching to TEE-based by default, the hope was that content piracy becomes old news. Nevertheless, we show that OTT apps might introduce new attack vectors rendering Widevine protection ineffective. We demonstrate the practical impact of our dire observations by obtaining DRM-free contents from six popular OTT apps, including Netflix, Hulu and Showtime. We insist that we do not look to introduce yet-another attack against DRM systems, but to better frame the misunderstood pitfalls of the rapidly growing Android DRM. We assess their prevalence, showing a rather concerning scenario.

Our contributions can be summarized as follows:

1) *New findings and understandings.* We conducted the first empirical study on how OTT apps leverage Widevine in Android. Notably, we automate apps monitoring by intercepting all calls to the Widevine process related to the DRM API. Our study reveals that almost no OTT app follows the Widevine recommendations in terms of cryptographic operations and revocation policy.
2) *Real-world impact.* To confirm the potential threats of our findings, we shed light on how certain practices allow an easy long-term compromise. We show that old still supported Widevine can be easily broken by recovering their software-only root of trust. Then, we reverse-engineered the proprietary cryptographic operations of Widevine to automatically obtain the decryption keys of any media. Our work results in CVE-2021-0639.
3) Our automated monitoring tool and Proof-of-Concept source code are available online[1] for reproducibility purpose and to assist future work on this topic.

Roadmap. The rest of the paper is organized as follows: section II presents the background information for our study. In section III, we present the research questions that we explore and that are related to Widevine security guidelines. Our findings and their practical impacts can be found in section IV. Section V details the limitations and future directions of this work. Section VI surveys the prior related research.

II. Background

A. DRM Systems

Modern DRM systems allow content providers to encrypt their media, and control who can decrypt them. The related architecture involves three parties: a Content Delivery Network (CDN) supplying encrypted content, a License Server providing the necessary keys to decrypt such a content, and a DRM module on the user device to retrieve these keys. For flexibility purposes, the part performing sensitive operations, such as decryption and license request, is implemented separately and called CDM or Content Decryption Module. Obviously, CDMs are required to protect the keys while using them. To this end, they mostly rely on closed-source proprietary protection mechanisms. Thus, every DRM system provides its custom CDM with its own interface.

[1]https://github.com/Avalonswanderer/widevinel3_Android_PoC

B. Android DRM API

In order to cope with this fragmented ecosystem, Android offers a unified API in Java/Kotlin for DRM systems. Starting from API level 18, this is implemented by some HAL (Hardware Abstraction Layer) module called Media DRM Server that abstracts the actual running DRM from the programming interface used by OTT apps. The Android DRM API mainly consists of two modules: Media DRM and Media Crypto. The Media DRM is used to communicate with License Servers and to manage keys for a given media. As for Media Crypto, it is used to perform decryption. The DRM APIs support the ISO/IEC 23002-7: Common Encryption standard (CENC) [13], but implement other encryption schemes.

Playing encrypted content when leveraging DASH (Dynamic Adaptive Streaming over HTTP) is summarized in Figure 1 and works as follows. First, the app constructs a Media DRM object and opens a new session. A Media Crypto object is then constructed and bound to the opened session. Next, Media DRM retrieves keys (aka licenses) from the License Server. The obtained keys are only accessible through Media Crypto. Indeed, the encrypted content is decrypted by a Media Codec instance to which the Media Crypto object was registered. Thus, MovieStealer as defined in [6] does not work anymore, since the app has never access to the decrypted buffer. In addition to the DASH mode, the DRM APIs provide the ability to establish a secure session to protect arbitrary data.

C. Google Widevine

Widevine is a DRM solution acquired by Google in 2010. Its earlier version (supported up to Android 5.1) worked only with the proprietary format *.wmv*. The current version, called Widevine Modular, implements various standards, including MPEG-DASH and CENC. Widevine Modular, or henceforth simply Widevine, is supported on several platforms: Android (4.4+), Android TVs, Smart TVs, and Web Browsers (e.g., Chrome, Firefox and Edge). Widevine defines three security levels: L1, L2 and L3, where L1 is considered for playing HD videos. At L1, all operations take place inside a TEE. L2 and L3 are used when the TEE is not available. Android Widevine does not propose. For L3, the CDM is processed outside the TEE. Being less secure, L3 can only play sub-HD content.

In Android, Widevine comes as a dynamically loadable HAL plugin. Widevine is not open-source; only provided as binary code. In a top down architecture, Widevine-backed Android DRM works as follows. All calls to the DRM API go through some Java Native Interface (JNI) layer via the `libmedia_jni.so` library. Instantiated by the `mediadrmserver` process, the Media DRM Server receives these calls, and accordingly, reaches Widevine using HAL. Any communication with Widevine first goes to its specific library such as `libwvdrmengine.so` or `libwvhidl.so`. In L3, no further component is involved. As for L1, whenever

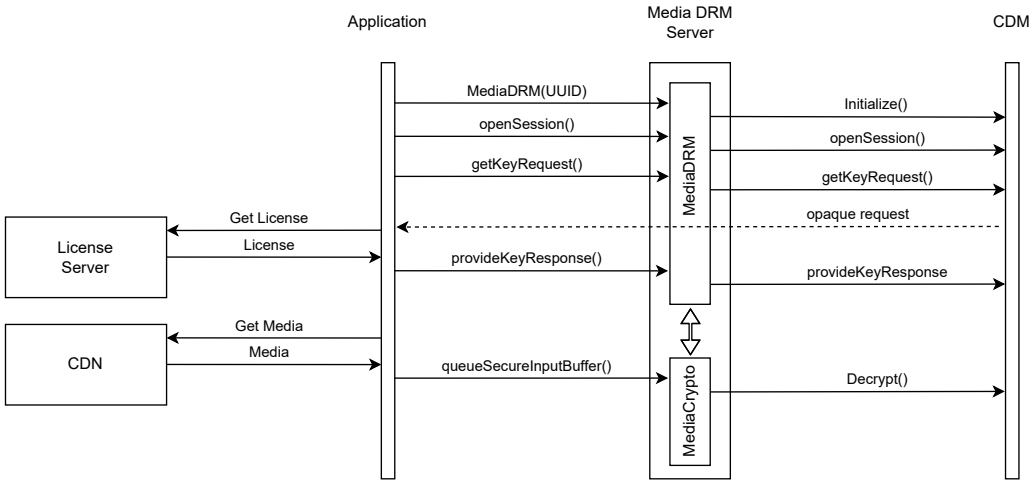


Fig. 1. Encrypted Content Playback in Android

CDM is required, this library calls `liboemcrypto.so` that sends the related requests to the Widevine TEE trustlet.

III. Research Question

Despite the recent surge of streaming on Android devices, little has been done to understand how Android most popular DRM, namely Widevine, is used in practice within the Android DRM framework. Our paper takes the first step in this direction and explores the implementation choices of OTT apps regarding Widevine security guidelines. To better frame the scope of our study, we also quantify the usage of Widevine in the wild. Our goal is to gain insights into *how do OTT apps follow Widevine recommendations to protect content?* By addressing this question, we show OTT apps would introduce new attacks vectors that render DRM protection ineffective while leveraging Widevine. Accordingly, we aim to answer the following four questions:

Q1: How often do OTT apps rely on Widevine? The main benefit of the unified Android DRM API is that developers can effortlessly rely on Widevine, or any Android integrated DRM, to protect content. The first question we intend to answer is how effective this API is for Widevine adoption. In particular, we intend to quantify the OTT apps leveraging Widevine or custom DRM implementation like in Indian music industry [14]. Because most apps do not disclose whether Widevine is supported (an official list can be found in [4]), we need to fill this gap by conducting several experiments for both L1 and L3. The support for L1 is an important security trait, since it resists most attacks against software-based DRM.

Q2: Do OTT apps encrypt their media contents using Widevine? In any DRM ecosystem, CDN servers deliver media contents to OTT apps. In a standard work flow, these contents are encrypted, and DRM modules in users' devices (aka CDM) require to decrypt them first. Nevertheless, the prevailing way of media consumption makes this operation less straightforward. Indeed, in most OTT apps, users are allowed to change the language (i.e., audio and subtitles) at any moment without negatively impacting users' experience. Thus, video tracks are obtained only once, while subtitles or audio are downloaded again for each language selection. In practice, these parts are sent separately, and thus might be differently protected depending on the OTT app that might choose for instance to only protect the video and not the audio. Moreover, the Android documentation of DRM API does not clearly explain this common scenario, which might inadvertently lead to unprotected contents. Our work investigates whether all media assets are DRM-protected for OTT apps: video, audio and subtitle tracks.

Q3: Do OTT apps use multiple keys for content encryption? Widevine has issued a set of recommendations concerning DRM cryptographic key usage [15] (similar ones are defined by W3C DRM standard EME [16] adopted by Widevine [17]). These recommendations state that video with different quality settings should be encrypted using a different key, and that audio files should be protected by an additional key. Obviously, such guidelines are defined to minimize the impact of a content key recovery, and therefore should be enforced. However, the use of multiple keys might make the code to manage more complex if no external media playback library is used, such as ExoPlayer [18]. Consequently, there might be some tension between Widevine recommendations about using multiple keys and apps implementations. In this paper, we intend to quantify the DRM keys received for each media. If an app does not encrypt audio tracks, or does not use distinct keys, we note such a practice as minimal.

Q4: Do OTT apps still support L3 outdated devices? Widevine regularly updates their software. For instance, the release cadence for Android CDM is annual (per Android release). In order to keep away some vulnerable implementations, Widevine may revoke devices due to non-compliance

with their security rules, e.g. no longer receiving security update. Nevertheless, OTT apps can ignore such revocation and deliver content keys to users [15]. This practice is not recommended, since old Widevine implementations could be broken at some point without any security patch. The problem of outdated devices is not straightforward, because following revocation rules restricts the number of OTT potential clients. Here, we identify to which extent OTT apps still support potentially vulnerable Android devices to underline the delicate trade-off between availability and security. We emphasize that this problem is quite different from the fact that Widevine, by design, supports L3 that is inherently bypassable [19]. Indeed, breaking into legacy L3 should be easier than the latest L3 version. In addition, it should have longer impact, because no vulnerability will be patched.

IV. Experimental Evaluation

As stated previously, our goal is to analyze the implementation choices of OTT apps while leveraging Widevine. In this section, we first describe our experiment setup to address our research questions, and review the methodology that we used to perform our evaluation. Then, we present our findings. Finally, we show how some of the existing practices can be abused to bypass DRM by targeting its weakest link.

A. Evaluated Apps

In our work, we evaluate the following Android OTT apps (with their number of installation in millions at the time of writing): Netflix (1,000M+), Disney+ (100M+), Amazon Prime Video (100M+), Hulu (50M+), HBO Max (10M+), Starz (10M+), myCANAL (10M+), Showtime (5M+), OCS (1M+), and Salto (1M+).

The selection was based on the list of Widevine clients [4] as well as their popularity on the Google Play Store. Unfortunately, we could not extend our study to more apps because of platforms restricting their client to register with a bank account in a particular country. However, our code can be easily applied for other apps.

B. Methodology

We could have answered all our research questions by looking directly into OTT apps. However, we did not consider such an option for two main reasons. First, they are closed-source, and their design and implementation differ considerably. This would have required substantial amount of reverse engineering that is specific for each app. Second, most evaluated OTT apps apply anti-debugging techniques such as time checking [20], or rely on SafetyNet [21] to hinder any dynamic analysis. Therefore, we preferred to focus on the Widevine HAL plugin. In particular, we precisely identify the work flow for each app regarding Figure 1, by monitoring all Widevine activities involved in the DRM API. This approach presents multiples advantages. Indeed, it bypasses all protection mechanisms implemented by apps to block dynamic analysis. Moreover, it can be easily automated. Nevertheless, in order to address some of our questions, we will still need to note OTT-specific information. Accordingly, we design our methodology considering a human in the loop, which also implies to limit the number of evaluated apps. Furthermore, the lack of documentation on Widevine made the manual analysis necessary before implementing our monitoring tool.

DRM API Monitoring. Here, we take a two-pronged methodology. First, we decompile the Java classes of the evaluated OTT apps to identify some of the included Android classes. More specifically, we scan all calls to `MediaDrm` and `MediaCrypto` methods that are required within a Widevine session. However, we are aware that some apps might include some dead code. Thus, in order to err on the side of soundness (i.e., low false positives), we monitored Widevine component functions linked to the Android Media DRM framework while playing protected content. Indeed, for both L1 and L3 security levels, we intercept and note any function called within the CDM process linked to the Widevine protocol (namely `_oeccXX` functions), loaded in `mediadrmserver` starting from Android 7 and `mediaserver` otherwise. To this end, we leverage Frida [22] to hook CDM calls. The use of L1 is confirmed whenever the control flow reaches `liboemcrypto.so`, since L3 is characterized by the fact that all calls stay within `libwvdrmengine.so` (the Widevine library name can differ between devices). In our Github[2], we provide a Frida script that automates OTT app monitoring. Moreover, to allow more in-depth analysis, we dumped input and output buffers related to various functions, including non DASH mode (e.g., generic encrypt and decrypt).

Content Protection. By protection, we mean cryptographic encryption by Widevine. When an OTT app asks for some media, the related CDN server does not deliver it directly. Instead, it sends several URI links that are used by apps to download all required video, audio and subtitle tracks. Contents might come encrypted or clear. Our approach mainly consists of obtaining these URI links for each OTT app, to determine whether the downloaded contents are protected or not. For this purpose, we just rely on video or audio players to read the downloaded files. As for subtitles, we check whether they contain ascii characters for English ones.

The main challenge in our approach is that apps also protect these URI links, and thus we have to find them. First, we keep monitoring all function calls to the CDM process, and more particularly non DASH mode, as it might be used as a secure channel to protect arbitrary data. In addition, we intercept the received network packets when selecting and displaying protected content to recover media URI and Media Presentation Description (MPD) files. Because of the use of TLS, we require to implement SSL repinning technique using Burp [23] and Frida. Moreover, we note the used key IDs for each content by parsing the MPD files and their related OTT-specific metadata. Finally, we perform our experiments for L1 and L3 to assess that it does not depend on security level.

[2]https://github.com/Avalonswanderer/widevinel3_Android_PoC

TABLE I
WIDEVINE USAGE AND ASSET PROTECTIONS BY OTTS

OTT	Widevine used (Q1)	Content Protection (Q2) Video	Audio	Subtitles	Widevine Key Usage (Q3)	Playback on L3 discontinued phones (Q4)
Netflix	●	Encrypted	Clear	Clear	Minimum	●
Disney+	●	Encrypted	Encrypted	Clear	Minimum	◐
Amazon Prime Video	●†	Encrypted	Encrypted	Clear	Recommended	●†
Hulu	●	Encrypted	Encrypted	-	-	●
HBO Max	●	Encrypted	Encrypted	Clear	-	◐
Starz	●	Encrypted	Encrypted	-	Minimum	◐
myCanal	●	Encrypted	Clear	Clear	Minimum	●
Showtime	●	Encrypted	Encrypted	Clear	Minimum	●
OCS	●	Encrypted	Encrypted	Clear	Minimum	●
Salto	●	Encrypted	Clear	Clear	Minimum	●

◐ Widevine fails during provisioning phase.
† using custom DRM if only Widevine L3 is available.
Minimum: Audio in clear or using the same encryption key as the video.
Recommended: Audio and Video are encrypted with different keys.

Outdated Device. Our approach is straightforward: we use Nexus 5 phone to display content. Released in 2013, it did not receive an official Android 7.0 in 2016, and thus Android 6.0.1 becomes its last update. It runs Widevine in L3 mode with CDM version 3.1.0. This is quite old, as the current CDM version is 15.0 in 2021. In our experiments, we attempt to display some media content on Nexus 5. We also keep monitoring all calls to Widevine. We distinguish two cases: (1) the app can display Widevine protected content, and (2) the app uses Widevine, but no content can be displayed.

C. Results

Our results are summarized in Table I.

Q1. All the evaluated apps depend on Widevine for content playback, and not on some custom app-specific DRM solution. One exception is Amazon Prime Video using an embedded Widevine library when just the L3 software-only mode is available. We also find that the L1 TEE-based mode is popular, unlike what was predicted in [7]. This is an important trend, since TEE becomes mandatory starting from Android 7.

Q2. Based on our findings about Widevine usage, we check whether OTT apps protect their contents. We first look for URI links related to content assets: video, audio and subtitles. First, using public Frida resources, we succeeded in bypassing SSL repinning on all OTT apps, which shows how ineffective such a security mechanism is. Then, we analyzed the network packets for each app. Once we found the URI links for a given media, we downloaded the associated assets and checked their protection status. A notable exception is Netflix that creates a secure channel between Widevine and the CDN to protect URIs through the non DASH Widevine API. This protection does not prevent us from recovering Netflix links by intercepting the output of some Widevine functions. Overall, we obtained all URI links for all OTT apps, except the subtitles URI for Hulu and Starz. Our investigation finds that video tracks are always encrypted, contrary to subtitles that are always in clear (we could not confirm this for Hulu and Starz). The case of audio tracks is more peculiar. Most OTT apps encrypt their audio, except for Netflix, myCanal and Salto. In fact, for these apps, audio in any language can be played anywhere without any OTT account.

Q3. As noted in Q2, most OTT apps protect their video and audio. Here, we explore whether they are encrypted with different keys. Of course, we cannot tell by just looking into the encrypted files. Thus, as explained previously, we analyzed some metadata indicating the identifier for every decryption keys. Our observations show that all evaluated OTT apps properly encrypt their videos with different keys depending on the resolution. This implies that we cannot trivially recover HD resolution, while breaking L3 mode. However, given the same resolution and the same content, many OTT apps are confirmed to use the same key for both video and audio, which is not recommended for DRM solutions by Widevine or EME. As for apps that do not protect audio, namely Netflix, myCanal and Salto, Widevine also considers this practice as a minimal recommendation setting. Only Amazon Prime Video strictly follows Widevine recommendation by always using distinct keys. For Hulu and HBO Max, we were unfortunately not able to conclude our analyses due to some regional restrictions.

Q4. On our Nexus 5 with its Android 6, the installation process succeeds for all OTT apps. Connected to a valid OTT account, we attempted to display some media. We notice that Disney+, HBO Max and Starz reject to provision Widevine with valid licenses, implying that Nexus 5 is actually revoked. Thus, media contents and their decryption keys were not even delivered to our Nexus 5. However, the remaining seven apps do provision Nexus 5 as a valid device, and therefore display the required media. One restriction was on the video quality, since Nexus 5 only supports L3, hence no HD video.

Insights and Learned Lessons. Our evaluation highlights the large adoption of Widevine by OTT apps on Android devices. More importantly, we find that TEE-backed Widevine is widely leveraged thanks to the unified Android DRM API. This shows that OTT apps are aware of piracy threats, and therefore look to rely on modern security technologies, such as TEE. However, we also find that this does not prevent developers from opting for bad security practices. Indeed, OTT

apps do not protect subtitles. There might be two reasons for this. First, OTT frameworks do not consider subtitles as a valuable asset to protect. Second, there is no API to deal with encrypted subtitles in the Android DRM API. This might also uncover the rational behind why most apps do not follow Widevine recommendation to use distinct keys for video and audio. The case of subtitles is more special though. Indeed, we notice that many apps call DRM API through ExoPlayer as recommended by Widevine [17]. This playback library proposes some API allowing developers to provide encrypted audio and video, but not subtitles. However, this API is not yet widely used with different keys. We highlight an unexpected result in our study: Netflix does not protect audio tracks. During our responsible disclosure, we discovered that Netflix was not even aware of that, because they believed that non-Dash mode was sufficient. App developers should not take all the blame, since the Android API does not make it easy to implement Widevine guidelines. However, we think that our results are surprising, since Widevine recommends ExoPlayer providing a demo app with best practices [18].

Finally, we shed light into the problem of license distribution. We notice that many OTT apps care more about reaching more clients than applying the revocation rules of Widevine: they allow the playback of protected content on devices no longer receiving security update. Note that most Android devices have short security update support period with for instance three years for Google Pixels [12] and five only for the latest model. This might seem anecdotal, but we will show in subsection IV-D how an attacker might exploit a weak Widevine implementation to recover media contents.

D. Practical Impact

We argue that discontinued phones constitute one of the weakest link in the DRM ecosystem. Indeed, the threat model of DRM includes attackers that have full control on their devices. This is due to the fact that the same media, with different resolution, is played everywhere, and the only incentive of an attacker is to recover that media, regardless of the targeted device. We demonstrate the practical impact of our results by obtaining DRM-free contents from all OTT apps still supporting old devices (except Amazon). To do so, we first analyzed Widevine internal key ladder from its root of trust to content decryption key. By reverse engineering, we found three main keys:

- **Keybox**: 128-byte structure including a magic number and a 128-bit AES Device Key. This key is installed by the manufacturer, and constitutes the root of trust (RoT).
- **Device RSA Key**: 2048-bit private RSA key. This key is installed when needed by the Provisioning Server. The installation process is protected by the keybox. This RSA key protects the Content Key sent by the License Server.
- **Content Key**: 128-bit AES key for media decryption.

During our reverse engineering. we find that the keybox is used to initiate the key ladder. By dynamically monitoring memory regions that are used during obfuscated cryptographic operations within `libwvdrmengine.so`, we searched for specific keybox structure (e.g., magic number). Thus, we succeeded in recovering the L3 keybox on our deprecated Nexus 5 due to an insecure storage of sensitive information (CWE-922), that lead to our CVE-2021-0639.[3]

Once we recovered the keybox, we were able to obtain the provisioned *Device RSA Key*. Then, we mimic the rest of the key ladder by intercepting Widevine function arguments to recover derivation buffers and encrypted keys. We implement this key ladder to automatically recover the media-related *Content Key*. During our experiments, we found out that OTT apps use the same keys for all their subscribers for a given media. Finally, we use MPEG-CENC [13] to decrypt all protected contents. With some processing, we reconstruct the pirated media and play it on another device (i.e., personal computer) without any OTT account. Since keys depend on the required quality and all media in our experiments was requested from Widevine L3, the best quality that we get is unsurprisingly 960x540, which is qHD (quarter high definition).

Our PoC (proof of concept) of the key ladder can be found in our GitHub[4]. Following our ethical approach, we only provide the implementation of the key ladder without the associated *Device RSA Key*, nor keybox. Thus, nobody can use it to actually pirate OTT contents. Note that our PoC works for both L1 and L3. However, we only applied it on Widevine L3. Our purpose is not to define yet another attack against DRM systems, but to show that some outdated L3 security protection can lead to a simple long-term compromise because of discontinued devices. We stress the fact that there is currently no clear solutions to the upgradeability of deprecated phones, even if the situation might be improved in the future. For this reason, we believe that OTT apps must strictly abide to Widevide revocation rules to avoid piracy.

V. Discussion

A. Responsible Disclosure

Our findings have been timely reported to all concerned parties following their responsible disclosure process. Netflix was quite responsive and we got rewarded via their bug bounty program. Regarding Google Widevine, our security report was assigned with the highest priority within the Google Vulnerability Reward Program (VRP). The Widevine security team investigated our findings and issued a patch to mitigate our identified flaws. Google assigned the CVE ID 'CVE-2021-0639' for us, and acknowledged us in the Google Hall of Fame and the Android Security Acknowledgments. Our goal is to improve the state-of-the-art knowledge of DRM internals and not to provide copyright infringement tools.

B. Not Just Another Attack

We insist that our work does not present a new attack against DRM solutions. Instead, we show that bad implementation practices imply new attack vectors that can be abused to recover DRM-free contents in Android smartphones.

[3]https://www.cve.org/CVERecord?id=CVE-2021-0639

[4]https://github.com/Avalonswanderer/widevinel3_Android_PoC

In particular, we show that OTT apps do not use Widevine protection to the fullest, and this might be due to lack of public documentation (therefore community discussion) and complex proprietary design. Moreover, we show that DRM protection is quite brittle if we include all the supported, yet outdated, versions. We successfully demonstrate that deprecated phones create more risks than benefits. Note that no SafetyNet or anti-screen recording techniques can be of any use, since attackers only need to monitor Widevine that runs in a different process.

C. Limitation & Future Work

As any study, our work is not exempt from limitations. Our paper focuses on the most popular DRM solution on Android smartphones, namely Widevine. However, Huawei's devices offer their custom DRM solution, called WisePlay. Moreover, outside the smartphone ecosystem, Widevine is present on Android TVs, web browsers on different operating systems, and small devices (e.g., Chromecast). Studying similarities and differences among these different implementations constitutes the main future direction of this work. We hope that our tools will encourage more work on these topics to reach other platforms and CDMs with fully automated approches.

Another limitation is that our PoC can only recover video with sub-HD quality. It is not clear how Widevine enforces the control on video transmission regarding the security level. On PCs, the Github project `netflix-1080p` [24] explains how to get HD quality on L3 by just modifying the profiles to be sent to the CDN. This implies that there is no strong verification for web browsers. An interesting future work is to adapt this exploit to Android in order to get the license keys of HD contents without breaking into the Widevine L1.

VI. Related Work

A. DRM Researches and DMCA

Despite the widespread of Widevine, surprisingly, no much attention has been given to the security of its underlying protocol. The main reason behind such a lack of public security analysis is the DMCA's 1201 clause that makes it illegal to circumvent controls preventing access to copyrighted material. Consequently, researchers cannot investigate and discover security vulnerabilities if doing so requires reverse engineering or circumventing controls such as obfuscated code. This law has already been used against security researchers to censor their work, as shown by Hewlett-Packard against Snosoft in 2002 [25]. Unfortunately, over fifty court-cases have been launched against security research as of 2016 [26].

Nevertheless, restrictions have been partially lifted recently. Indeed, in October 2018, the American Library of Congress and the Copyright Office have expanded the exemptions to DMCA. Therefore, in theory, researchers can now freely investigate and publish security flaws on DRM solutions.

B. Bypassing DRM & Widevine Keys Recovery

In the past decades, multiple techniques have been developed to circumvent now outdated DRM solutions without having to break the underling scheme, like MovieStealer [6] targeting content decryption to recover decrypted buffers within the decoder when no TEE is available. Analog loophole was also used to capture analog signals of the protected content by screen-recording or through connectors.

In 2019, David Buchanan claimed to have broken Widevine L3 DRM on Linux Chrome browsers. In a tweet [27], which is the only available information about this attack, Buchanan mentioned that Widevine L3 relies on AES-128 Whitebox to protect the keybox. This attack gets a large number of articles on the web, all of which relied merely on the tweet of Buchanan who has never provided any further detail.

In October 2020, Tomer Hadad released widevine-l3-decryptor in Github. This project is a Chrome extension on Windows that contains a hard-coded value of an RSA key pair used by Widevine L3 unique for all Windows devices. Hadad mentioned that he extracted the RSA private key "*by applying some mathematical tricks to Arxan's whitebox algorithm*". In November 2020, Google issued a DMCA takedown request against widevine-l3-decryptor and all its forks [19]. This proves that Google still views L3 security as a serious matter.

In May 2021, Zhao [8] broke into the Widevine L1 trusted app to recover the Widevine keybox of a Pixel 4. However, he did not show how a recovered keybox can be used to decrypt protected contents. In our work, we took this further step and implemented the cryptographic mechanisms of Widevine.

C. Closed Source API Analysis

Our work inspects the misusage of the Widevine API by OTT apps. Similarly, other works have been made to analyze closed source proprietary API on Android. For instance, Bianchi et al. [28] have inspected the fingerprint API and its misuage by app developers, leading to multiple attacks ranging from confused deputy problem to full fingerprint bypass. In their work, Ibrahim et al. [29] looked at the SafetyNet Attestation API used to detect if an app is running in a non compromised Android environment. Their findings point out that more than half of analyzed apps were using trivially bypassable checks while none were correctly using the SafetyNet API.

VII. Conclusion

In this paper, we report on an empirical evaluation of the protection mechanisms used by content providers within Android to protect their media assets. Our study shows that OTT apps relies on Widevine using TEE protection when available but do not comply with Widevine recommendations in encrypting video and audio with distinct keys or regarding revocation of old devices. We investigate the OTT apps support of Android legacy devices running the software-only Widevine DRM no longer receiving security updates. We expose that by choosing large content distribution over security, OTT apps have extended the device attack surface. We show that such vector can be trivially exploited to recover DRM-free movies from six OTT apps including Netflix, Hulu and Showtime. We hope that this paper will push research forward in this domain and encourage furture works.

REFERENCES

[1] Fortune Business Insights, "Over The Top services market to reach USD 139.00 billion in 2028; emergence of Smart TVs by various companies to bolster growth." https://www.globenewswire.com/news-release/2021/08/17/2281647/0/en/Over-The-Top-Services-Market-to-Reach-USD-139-00-Billion-in-2028-Emergence-of-Smart-TVs-by-Various-Companies-to-Bolster-Growth-states-Fortune-Business-Insights.html, 2021.

[2] H. Halpin, "The crisis of standardizing DRM: the case of W3C encrypted media extensions," in *SPACE*, vol. 10662 of *Lecture Notes in Computer Science*, pp. 10–29, Springer, 2017.

[3] D. Dorwin, J. Smith, M. Watson, and A. Bateman, "Encrypted media extensions." https://www.w3.org/TR/encrypted-media/, 2019.

[4] Google Widevine, "Widevine." https://widevine.com/.

[5] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *TrustCom/BigDataSE/ISPA (1)*, pp. 57–64, IEEE, 2015.

[6] R. Wang, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Steal this movie: Automatically bypassing DRM protection in streaming media services," in *22nd USENIX Security Symposium (USENIX Security 13)*, (Washington, D.C.), pp. 687–702, USENIX Association, Aug. 2013.

[7] S. Y. Chau, B. Wang, J. Wang, O. Chowdhury, A. Kate, and N. Li, "Why johnny can't make money with his contents: Pitfalls of designing and implementing content delivery apps," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, (New York, NY, USA), p. 236–251, Association for Computing Machinery, 2018.

[8] Q. Zhao, "Wideshears: Investigating and breaking widevine on QTEE," *BlackHat Asia*, 2021.

[9] G. Beniamini, "QSEE privilege escalation vulnerability and exploit (CVE-2015-6639)." https://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html, 2016.

[10] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted TEE systems," in *IEEE Symposium on Security and Privacy*, pp. 1416–1432, IEEE, 2020.

[11] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y. R. Choe, C. Kruegel, and G. Vigna, "BOOMERANG: exploiting the semantic gap in trusted execution environments," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, The Internet Society, 2017.

[12] Google, "Learn when you'll get software updates on Google Pixel phones." https://support.google.com/pixelphone/answer/4457705, 2021.

[13] ISO/IEC, "Common encryption in ISO base media file format files - 2nd edition," 2015.

[14] A. Dabholkar, S. Kakarla, and D. Saha, "Looney tunes: Exposing the lack of DRM protection in indian music streaming services," *CoRR*, vol. abs/2103.16360, 2021.

[15] Google Widevine, "Widevine News." https://web.archive.org/web/20210903150435/https://www.widevine.com/news, 2021.

[16] Can I Use, "Encrypted media extensions." https://caniuse.com/eme, 2021.

[17] Google Widevine, "Widevine DRM Overview." https://developers.google.com/widevine/drm/overview, 2021.

[18] Google, "Exoplayer." https://github.com/google/ExoPlayer, 2021.

[19] Github, "DMCA." https://github.com/github/dmca/blob/master/2020/11/2020-11-09-Google.md, 2020.

[20] L. Xue, H. Zhou, X. Luo, Y. Zhou, Y. Shi, G. Gu, F. Zhang, and M. H. Au, "Happer: Unpacking android apps via a hardware-assisted approach," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pp. 1641–1658, IEEE, 2021.

[21] Android, "Protect against security threats with SafetyNet." https://developer.android.com/training/safetynet, 2021.

[22] O. A. V. Ravnås, "Frida." https://frida.re/.

[23] PortSwigger, "Burp suite." https://portswigger.net/burp, 2021.

[24] truedread, "netflix-1080p." https://github.com/truedread/netflix-1080p, 2021.

[25] D. McCullagh, "Security warning draws DMCA threat." https://www.cnet.com/news/security-warning-draws-dmca-threat/, 2002.

[26] Electronic Frontier Foundation, "Reported case list." https://www.eff.org/files/2016/03/17/1201_reported_case_list_revised.xls, 2017.

[27] David Buchanan, "Breaking Widevine L3 on Linux Chrome browser." https://twitter.com/david3141593/status/1080606827384131590, 2019.

[28] A. Bianchi, Y. Fratantonio, A. Machiry, C. Kruegel, G. Vigna, S. P. H. Chung, and W. Lee, "Broken fingers: On the usage of the fingerprint API in android," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, The Internet Society, 2018.

[29] M. Ibrahim, A. Imran, and A. Bianchi, "Safetynot: on the usage of the safetynet attestation API in android," in *MobiSys '21: The 19th Annual International Conference on Mobile Systems, Applications, and Services, Virtual Event, Wisconsin, USA, 24 June - 2 July, 2021* (S. Banerjee, L. Mottola, and X. Zhou, eds.), pp. 150–162, ACM, 2021.