Looney Tunes: Exposing the Lack of DRM Protection in Indian Music Streaming Services

Ahaan Dabholkar  
ahaand@iitbhilai.ac.in  
de.ci.phe.red Lab, Indian Institute Of Technology Bhilai

Sourya Kakarla  
sourya4@iitkgp.ac.in  
Indian Institute Of Technology Kharagpur

Dhiman Saha  
dhiman@iitbhilai.ac.in  
de.ci.phe.red Lab, Indian Institute Of Technology Bhilai

Abstract
Numerous studies have shown that streaming is now the most preferred way of consuming multimedia content and this is evidenced by the proliferation in the number of streaming service providers as well as the exponential growth in their subscriber base. Riding on the advancements in low cost electronics, high speed communication and extremely cheap data, Over-The-Top (OTT) music streaming is now the norm in the music industry and is worth millions of dollars. This is especially true in India where major players offer the so called freemium models which have active monthly user bases running in to the millions. These services namely, Gaana[33], Airtel Wynk[32] and JioSaavn[34] attract a significantly bigger audience than their 100% subscription based peers like Amazon Prime Music, Apple Music etc.[35] Given their ubiquity and market dominance, it is pertinent to do a systematic analysis of these platforms so as to ascertain their potential as hotbeds of piracy. This work investigates the resilience of the content protection systems of the four biggest music streaming services (by subscriber base) from India, namely Airtel Wynk, Gaana, JioSaavn and Hungama. By considering the Digital Rights Management (DRM) system employed by Spotify as a benchmark, we analyse the security of these platforms by attempting to steal the streamed content efficiently. Finally, we present a holistic overview of the flaws in their security mechanisms and discuss possible mitigation strategies. To the best of our knowledge, this work constitutes the first attempt to analyze security of OTT music services from India. Our results further confirm the time tested belief that security through obscurity is not a long term solution and leaves such platforms open to piracy and a subsequent loss of revenue for all the stakeholders.

Keywords: Digital Rights Management, Web Security, Piracy, OTT Audio Streaming

1 Introduction
OTT is an acronym for “over-the-top” and refers to the distribution of multimedia (audio, video) content over a public network. Recent trends have shown a mass adoption of smart mobile devices in the consumer market. This coupled with a higher penetration of high-speed, cheap Internet and the emergence of advanced technologies, such as 5G, 4G, developed online payment infrastructure and continual demand for content within the entertainment domain, projects the global OTT service market to grow from $81.60 billion in 2019 to $156.9 billion by 2024 exhibiting a CAGR (Compound Annual Growth Rate) of 14% [38]. The Asia Pacific region is set to record the highest growth rate during the forecast period. According to a joint report published by the Indian Music Industry (IMI) and Deloitte India [18], the audio-video OTT market in India is valued at around US$ 280 million with nearly 150 million monthly active users accessing soundtracks across various platforms.

| Service Name | Business Model | Origin | Reference |
|--------------|----------------|--------|-----------|
| Airtel Wynk  | Bundle, Ad Supported | Domestic | [32] |
| Apple Music  | Paid            | International | [4] |
| Amazon Music | Paid            | International | [2] |
| Gaana        | Ad Supported    | Domestic | [33] |
| Hungama      | Ad Supported    | Domestic | [6] |
| JioSaavn     | Bundle, Paid    | Domestic | [34] |
| Spotify      | Ad Supported    | International | [11] |
| Youtube Music| Subscription    | International | [13] |

Table 1. OTT music services currently operating in India

Revenue from digital means contributes nearly 78% to the overall recorded music industry revenue in India and 54% [27], globally. A survey of India’s audio streaming market reveals that it is primarily divided among domestic players Wynk, Gaana, JioSaavn, Hungama and global players Spotify, Amazon Music, Apple Music and more recently YouTube Music (Table 1). As per a consumer insights survey conducted by the IFPI in 2018 [28], an average internet user in India spends 21.5 hours every week listening to music, higher than the global average of 17.8 hours. It is interesting to note that despite the popularity, contemporary literature lacks security analysis of any of the domestic OTT platforms and forms the primary motivation of this work.
This easy and free access to content was thought to have solved several issues regarding unsanctioned sharing of media [30] as it provided Music-as-A-Service which was more lucrative to the consumer than content ownership [24]. However, with the consequential emergence of “stream-ripping”, piracy has increasingly kept pace. The gravity of the situation reflects in the numbers where estimates point to almost US$ 250 million lost each year in India alone while the estimated number of stream-rippers in the US have grown to an alarming 17 million [42]. The surging popularity of such platforms has also not been missed by the shadier sections of our society with more sinister agendas [14]. Couple this with the 40% - 60% of revenue that is lost to pirates, there is hence a dire need to take a critical look at the security of such content delivering platforms. A recent paper on bypassing DRM protection in online video streaming [45] is one of the many research efforts highlighting the need to have a deeper understanding on how OTT services should be deployed in practice.

In this work we systematically analyze the four leading OTT music service providers in India namely Wynk, Gaana, JioSaavn and Hungama comparing them to the best practices in the industry. To our great surprise, our research reveals that none of these platforms adopt any state-of-the-art DRM protection. Contrary to this they actually attempt a very rudimentary form of code obfuscation. As a result, we were able to not only reverse engineer their protocols but also devise mechanisms leading to automated, unsupervised and uninterrupted download of music from their servers. We develop detailed Proof-of-Concepts for the same and illustrate case-studies on each of the platforms. To put things in context, we also investigate the Spotify web-application and find it adopting very standard DRM protection making it a benchmark in the comparative study that we furnish later. Finally, we discuss possible mitigation strategies to salvage the situation. As a part of responsible disclosure, this work was attempted to be communicated to the concerned parties. With the exception of Wynk, response from others is awaited.

Our Contributions

Our contributions can be summed up as follows -

- We present a security analysis of the content protection systems in place for four of the biggest music streaming services (by subscriber base) in India.
- We highlight the lax security protocols in place in all these services by attempting to steal content in an undetectable way and provide proof of concepts to automatically acquire content by reverse engineering their content delivery protocols.
- We present a comparative study of these apps with the current state-of-the-art DRM systems.
- We present a discussion on the design choices employed by these services and make recommendations to enhance their security.

Organisation Of The Paper

The following sections contain the conclusions and results of our experiments while reverse engineering said services. We first provide a primer on Adaptive Streaming in Section[2.1] which is used by most of the OTT streaming services and which would help us elucidate the protocols involved clearly. We follow this up in Section[2.3] with a brief note on present day DRM systems. Section[3] is dedicated to describing the Widevine DRM used by Spotify to protect it’s content, to establish a benchmark for comparing the other services. This leads us into the results of the reverse engineering in Section[4] where we give reconstructions of the protocols used. Section[5] contains discussions on the flaws in current DRM systems and the design choices made by these services followed by our conclusions in Section[7].

1.1 Responsible Disclosure

All the services mentioned here were contacted prior to submission of this manuscript with reports on the vulnerabilities in their protocols and with offers to collaborate on the fix. It should be noted that none of these services have vulnerability disclosure programs and hence finding a suitable point of contact was tough. When informed of the break, Airtel Wynk was all for the idea of a collaborative fix but ended up deploying a haphazard patch without consultation and proper notice which was broken eventually using the same techniques.

2 Background

This section is provided as a primer for familiarising the reader with certain technologies that are heavily referred to in this work.
2.1 Adaptive Streaming

Classical streaming protocols used a technique called progressive streaming to deliver content. In this technique, a single file sitting on the vendor's server was delivered to the client requesting it. Though this method was simple, it had some obvious inefficiencies which are demonstrated using a toy example below:

1. Consider two clients with two different displays, one having a 720p display and the other having a 4K one. With the progressive streaming protocol, both clients would be delivered the same content despite the differences in their hardware capabilities. If the content streamed was in 4K, it would not pose a problem for the second client, however for the first client it would imply that he receive 4K media which would eventually be downscaled to 720p (or not run at all, depending on the decoding hardware).

2. A problem would also arise if one of the clients had severely limited network bandwidth. This client would be unable to consume content meaningfully owing to it's unnecessarily large size.

The idea of adaptive streaming aims at solving both these issues in real time. The first problem is solved by having encodings at multiple bitrates of the same media on the content-delivery servers while the second problem is solved by providing the client with the ability to switch between encodings mid-stream depending on it's resources. This adaptiveness is facilitated by dividing the source content into chunks and indexing it. Hence if network degradation is detected by the client, the next chunk can be retrieved from a lower bitrate, thus maintaining the flow of content.

We will now describe the HTTP Live Streaming (HLS) Protocol, an adaptive streaming protocol which is predominantly used by most streaming services including, in our case, Wynk Music to serve content efficiently. An understanding of its architecture is necessary to grasp the underlying protocol. The HLS Protocol was developed by Apple Inc and released to the public in 2009. According to survey reports from 2019, HLS remains the most adopted protocol with more than 45% of broadcasters using it to provide streaming services to their clients.

2.2 HTTP Live Streaming (HLS)

As the name suggests, HLS is an adaptive streaming protocol that delivers content over HTTP/HTTPS. The HLS Architecture [26] essentially involves three components:

1. The Streaming Server
2. The Distribution Component
3. The Streaming Client

A typical configuration (Figure 2) consists of a hardware encoder that encodes Audio/Video input into MP3/H.264 and encapsulates it into an MPEG-2 transport stream. A software segmenter then divides the stream into chunks (.ts files) of equal duration and creates an index file index.m3u8 that contains links to those chunks. This process is carried out for each encoding of the A/V stream and a master index file, also called the manifest is generated and usually named master.m3u8. The manifest identifies and points to the different index files available for that particular A/V stream. This manifest is then served by the streaming server over HTTP to the client which then selects the suitable encoding based on the resources available and requests the index file of that encoding. Once the index is received, the client sequentially makes requests for the chunks, enabling playback on its device. When a network change is detected, a lower bitrate encoded index file is requested in order to retrieve the next chunk, for continuous playback.

2.3 Digital Rights Management (DRM)

DRM can be thought of as a digital lock or as protections in place to secure proper usage of proprietary technologies and copyrighted works [21]. Although enforced in many countries through licensing agreements and laws such as the Digital Millenium Copyright Act (DMCA) [40] which criminalise circumvention, their efficacy and ulterior motives have been the subject of constant debate. A discussion on these technicalities is however not relevant to this paper, where we will instead choose to focus upon the DRM techniques used primarily by OTT service providers. Most streaming services such as Netflix, Hulu, Amazon Prime require playback devices to support some form of DRM. Common choices for DRM schemes are Microsoft PlayReady [8],
Apple FairPlay[3], Adobe PrimeTime[1], Marlin[7] and Google Widevine[12]. Most of these DRM schemes at least provide browser support through Content Decryption Modules[20] (CDMs) which follow the Encrypted Media Extensions (EME) [20] specification which in turn is implemented by all major browsers today. This uniformity has resulted in an unprecedented ease of implementing basic content protection at an efficient cost. An example would be Google’s Shaka Player[10]: an open source player which can be integrated into a project with relative ease, which uses the Widevine DRM scheme and supports adaptive streaming over MPEG-DASH[48] and HLS.

Earlier Work on DRM

Breaking DRM protection has been the focus of all kinds of hacker groups ever since the popularity of commercial software grew, giving rise to the so-called pirate hubs which are still popular today. Right from spoofing KMS systems for acquiring Windows licenses to patching AAA titles deploying the Denuvo Anti Tamper[5] system, the community has been witness to some rather creative albeit illegal ways of stealing content over the years. Academic attention to the problem of breaking DRM systems [15, 16, 46] however, has proved to be rather mild. To the best of our knowledge, our work is the first such analysis of OTT Indian music streaming webapps. We did however take inspiration for the subject from the work done by Wang et. al [46] on automatically bypassing DRM systems and for a way to present our findings, we looked at Kumar et. al’s [31] work on analysing UPI apps in India.

3 Spotify: Demonstrating Widevine

Having been baffled by the results of our preliminary investigations into Wynk, we were curious to see if this trend was followed across the board by even the big players in the game and hence we decided to focus our attention on Spotify. We were quite satisfied to observe that Spotify proved resilient to the basic reversing techniques that had proved fatal for the other services in terms of content security. However we must clarify, we do not claim that Spotify is infallible, just that it would require more effort than what was put in for all the others combined. Here we present a high level overview, explaining how Spotify protects its content while streaming and also use this analysis later to highlight the deficiencies in the other protocols.

3.1 Methodology

We decided to target the Spotify Web Player as it was clearly suited for comparison with the other services. By monitoring network requests made by the web player and using a combination of static and dynamic analysis of the client-side Javascript modules, we were able to piece together the inner workings. The documented architecture[23] for Widevine was also heavily referred to in order to establish context.

3.2 Summary Of Findings

Spotify\(^1\) is currently using Widevine Level L3 to implement DRM for its content which is streamed to a modified version of the Shaka player that uses the HTML5 Media Source Extensions to interact with the CDM.\(^2\)

The CDM is a precompiled binary,\(^3\) implemented as a shared library (libwidevinedcm.so in Linux) for Google Chrome and as a plugin for Mozilla Firefox.

Coming to the retrieval, Figure (3) depicts the protocol followed. We would like to point out that we have intentionally left out the exact details in some parts of the protocol in the interests of keeping the description brief.

1. **Login** This is the first part of the protocol that a client encounters when trying to start playback. Streaming does not start until user identification and authentication is done. There are numerous options for authentication using OAuth but all of them effectively end up setting identification cookies on the client. Let us denote these cookies by C. These cookies are later used to setup and maintain a player state which is used to track playback, sync multiple devices, gather insights etc.

2. **Acquire Access Token** An access token is requested from the spotify servers using the cookies C. As in the protocol, we shall refer to this token by Bearer. Bearer is an authenticated token that has a long expiration time and is required by and subsequently used for most operations of the Spotify client.

3. **Get Resource URI** To actually retrieve the media file from the Content Distribution Network (CDN), we require an authorized URI which permits access to the content on the server. This URI is obtained from the Spotify servers by making a request and leveraging the Bearer token as authorization. If the Bearer token is valid, the server responds with a list of multiple URIs (We assume for redundancy).

4. **Retrieving the First Chunk Of Data** According to the widevine specification, the first chunk of data is used to gather licensing information for subsequent decryption of content. Having obtained the CDN URI in the previous request, the player requests the first chunk of data of a certain size by setting the Range header in the request. The server response which contains a chunk of the media file (distinguished from its header) is used to extract initial licensing information

\(^1\)For future reference, unless explicitly stated otherwise, a reference to Spotify refers to the Spotify Web Player

\(^2\)For a detailed explanation, refer to the EME documentation[20]

\(^3\)An open source CDM or OCDM can be viewed in the Chromium Project’s source, however the Chrome CDM is closed source [19]
Figure 3. Spotify’s Content Retrieval Protocol - Widevine L3 (Reconstructed)

called initdata. initdata is then passed on to the Content Decryption Module (CDM).

5. Obtaining the License The fragmented\(^4\) media chunks retrieved from the servers are encrypted using AES-128 in CTR mode. Hence, in order to initialise playback, decryption of these media needs to be performed. The information (keys, initialisation vectors) needed for decryption is included in the license. Based on the initdata, the CDM generates an encrypted license request and passes it to the player. The player then relays this request to a license URI that was obtained asynchronously. If the request and its payload is valid, the server responds with an encrypted response that is relayed by the player to the CDM. The CDM decrypts the response to obtain the license.

6. Playback Once the licensing information has been obtained, 10 second chunks are downloaded from the servers and passed to the CDM which decrypts those chunks and passes them to the Audio/Video Stack for playback.

3.3 Lessons Learnt

Spotify does a few things very well for protecting their content as it is streamed to a client’s device. For the sake of establishing standard practices, we highlight a few of them below -

1. Mandatory User Identification The login process forces a client to identify itself in order to use the services. Spotify, while providing flexibility of login options with OAuth also implements reCAPTCHA protection against bots. In addition to this, Spotify can track each user’s activity which could potentially be used to recognise malicious use.

2. Streamed Content is Encrypted To prevent stream-ripping (discussed in Section 5.5), content stored on the servers is encrypted.

3. No Hardcoded Keys The keys for decryption of the streamed content are not hardcoded in the files that the user has direct access to.

4. License Information is Invisible to the Player The license information passed between the CDM and the server is encrypted and hence is not accessible to the user.

5. Content Decryption Module The CDM is theoretically the weakest part of the protocol. However in terms of usable/practical security, since it is closed source binary, it offers a basic level of protection against direct observation of the decrypted content, but is theoretically vulnerable to black box cryptanalysis techniques and some implementation level exploits. L2 and L1 level Widevine attempt to mitigate this vulnerability by having the decryption occur in a Trusted Execution Environment(TEE)[44].

Now that a benchmark has been established, we proceed to present an analysis of the four biggest OTT music streaming services in India, in the process highlighting security gaffes where DRM is concerned.

\(^4\)See moof boxes [25] for information on format of the media fragments
4 Case Studies

This section forms the basis of the work done in this paper. We present here, a reconstruction of the streaming protocols used by the four biggest (by subscriber base) music streaming services in India, in view of formulating an exploit to steal their content. The reconstruction of these protocols involved reverse engineering the Javascript modules executing on the client browser using static and dynamic techniques such as code de-obfuscation, debugging etc., observing network packets using Burp[9] and a fair amount of intuition. In all cases, we were able to completely replicate the protocols in order to get access to the audio content using these standard techniques of reverse engineering. Some code obfuscation aside, none of these services used industry standard DRM and could be broken with minimal effort by a dedicated attacker.

Given below is a summarised analysis of Airtel Wynk, JioSaavn, Gaana and Hungama. Using this analysis we were able to write scripts to automatically steal content. In the interest of keeping the descriptions concise, we have deliberately sacrificed rigorous function definitions in favour of broad descriptions of what those functions do, while illustrating a protocol. The token/variable names mentioned are also similar to their names in the actual JS code. For a detailed description refer to Appendix A. The implementation details are furnished in Section 6.

4.1 The Curious Case Of Wynk

Airtel Wynk Music was the first service that we came across that had serious flaws in their content security mechanism. The flaws were such, that we were able to write scripts to automatically steal content. In our observation was that persisting the values for these parameters had no effect on the execution of the protocol. As a response to this request, the server replies with values for uid and token.

The Protocol in a Nutshell.

- **Client Registration** The client is identified to the server using a POST request containing the deviceId and userAgent parameters in the payload. These parameters are set by the client and appear to be random in nature. Our observation was that persisting the values for these parameters had no effect on the execution of the protocol. As a response to this request, the server returns with values for uid and token.

- **Compute Search Id For Resource** A search_id was generated using the method shown in the protocol diagram in Figure 1. It is computed based on the URL and the content of the payload.

- **Acquire Authenticated URI** A POST request is made to retrieve the authenticated URI for content retrieval from the CDN. Using token as the key, a SHA1-HMAC of a string containing the search_id is generated. The Base64 encoded value of this HMAC is assigned to a request header x-b5y-utkn after appending it to the uid. The result of this request is a URL with a set of signed cookies which we will refer to as CloudFront Cookies.

- **Acquire Manifest** On making a request to the URI obtained as a response in previous request, the server responds with the manifest file which contains URIs to the various master.m3u8 files available.

- **Acquire index.m3u8** Using the URI for the index file of highest quality available, a request is made with query parameters being set using the CloudFront Cookies obtained previously. A successful response from the server gives us the index.m3u8 file of our choice.
• **Getting Content** By making GET requests to the chunk URIs present in the index file and setting the appropriate query parameters, the client starts receiving .ts media files from the server. By appending those files in order, the complete audio file is obtained.

Following our disclosure, Wynk made certain changes to their protocol that are listed below -

1. A code obfuscation scheme was introduced that replaced function/variable names, identifiers with what were essentially array lookups. The array used for lookup was included in the source code which rendered the obfuscation useless.

2. The client registration process was redesigned and a time window was introduced using Time Based OTPs (TOTPs)[49].

However, the content retrieval part of the protocol remained the same. The introduced changes only served to complicate the process of getting the authenticated URIs for the CDN. Moreover, content was still being streamed without encryption. The revised protocol is described in Figure 5. Needless to say, we did not face any difficulties in breaking Wynk once more.

**The New Protocol in a Nutshell.**

• **Initialisation Of Identifiers** The first part of the protocol involved the generation of certain tokens, namely, BK, deviceId, pk & sk. pk and sk were values that were hard-coded into the source code while BK and deviceId were generated using the *epoch time* and a pseudorandom number.

• **spit_out(BK, deviceId)** Two requests are made to the server using the outputs of this method which basically does some intermixing of the strings deviceId and BK. The output strings are then appended with ".1.jpg" and ".2.jpg" and treated as endpoints for requests. Now, we are not entirely sure why the image extensions are used in particular, however we can confidently say that the response to those requests has no further use. That being said, if either of those requests are not made, the protocol fails subsequently.

• **check() & login()** These functions are named after the endpoints to which requests are made. A successful response to the check endpoint returns several parameters which are used to compute the values of certain headers in the request to the login endpoint. A successful response from the login endpoint contains the parameters dt, uid, token, kt among others.

• **Compute Search Id** This method did not change compared to the previous deployment of Wynk.

5When we say *without encryption*, we refer to the fact that after the decryption from the HTTPS layer and gzip unpacking has occurred, the audio content is directly playable (unencrypted)

6The changes can be observed in the Appendix

Figure 5. Revised Content Retrieval Protocol - Wynk (Reconstructed)

- **Acquire Authenticated URI** The values received in the previous step are used to set the headers for another request as follows -
  - x-bsy-uuid ← dt
  - x-bsy-utkn similar computations as before
  - x-bsy-t ← AES(kt, TOTP(dt||sk, 600, 6))
This POST request if successful returns the CloudFront Cookies with a URI. The rest of the protocol follows identically to the previous version of Wynk.

It is pretty evident from the analysis that Wynk went to greater lengths to complicate the retrieval mechanism post disclosure, however they failed to address the crux of the problem.

4.2 JioSaavn Joins The Jam

With the findings from our work on Wynk, we were inspired to look into other platforms to test if the situation found with Wynk was a general norm among established players. JioSaavn is the 2nd most popular India based music streaming service in terms of number of subscribers. It took some vigilant effort to get to the media content but once the relevant execution path was found, piecing together the protocol was found to be extremely easy and straightforward.

![Figure 6. Content Retrieval Protocol - JioSaavn (Reconstructed)](image)

The Protocol in A Nutshell.

- **Acquiring Song Info** Interestingly, the metadata related to the song is served within an HTML response. It is found within the JavaScript variable, window._INITIAL_DATA_. Parameters that are essential for fetching the media content are encrypted_medial_url and perma_url.

  - **Generate Auth Token** A GET request is made to https://www.jiosaavn.com/api.php?call=song.generateAuthToken&url=<encrypted_media_url>&bit_rate=<bit_rate> to obtain the authorised URL that is used to fetch the media from the CDNs. The relevant parameters are url which is the encrypted_media_url discussed above and bit_rate which takes the values "128", "320", "64", "32", "16". The response contains auth_url which is verified by the CDN to authorize a request.

  - **Downloading Media** A GET request is made to auth_url to finally retrieve the relevant media. This URL is sufficient for authorization.

4.3 Getting Gaana

Gaana was the third service that we looked at. Figuring out the protocol was easy as Gaana had no code obfuscation and at least for the non-logged in user, did not rely on cookies at all. Figure 7 demonstrates the working of the protocol.

![Figure 7. Content Retrieval Protocol - Gaana (Reconstructed)](image)
GET Request to https://www.hungama.com/audio-player-data/track/<song_id>

Response
[{...'mediaid': <media_id>, 'file': 'https://www.hungama.com/mdnurl/song/<media_id>?token=<token>'...}]

POST Request to http://www.hungama.com/mdnurl/song/{song_id}?token=<token>

Body:

GET Request to <media_url>

Response
Content-Type: mp3
(Served as stream (206))

Figure 8. Content Retrieval Protocol - Hungama (Reconstructed)

- **Decrypting path** The key and initialization vector is hardcoded in the JS files and we use those values to decrypt and obtain the authorized URI.
- **Acquire Manifest** A request to the authorised URI returns the manifest.m3u8 which contains the URI for index.m3u8
- **Playback** The index.m3u8 file contains URIs for all chunks of the audio. After iterating through this file and making requests for all chunks (.ts files), we can append them together to obtain the complete audio.

4.4 Hunting Hungama

Hungama is yet another popular music streaming service in India. We explored its content serving mechanism and found it to be pretty similar to JioSaavn and reverse engineered the following protocol.

The Protocol in A Nutshell.

- **Audio Player Data** A GET request is made to https://www.hungama.com/audio-player-data/track/<song_id> to fetch the metadata of the song. The song_id parameter required for this request is obtained from the WebApp URL of the song which is of the form https://www.hungama.com/song/<name>/<song_id>. The response contains the relevant values media_id and file. The file URL contains the parameter token.
- **Fetching Media Url** A POST request with an empty body is made to http://www.hungama.com/mdnurl/song/{song_id}?token=<token>. The response contains media_url which is the final URL that is used to retrieve the media. The bit rate can be chosen by setting the hcom_audio_qty parameter to one of “high”, “low”, “medium” in the Cookie header.
- **Downloading Media** A GET request is made to media_url to download the relevant media. This URL is sufficient for authorization.

5 Discussion

5.1 Comparative Analysis

Among the case studies showcased in this work, reversing the protocol for Wynk was the most challenging in terms of effort due to the intricacies and complexity of the implementation. Yet, in the end it turned out to be a matter of getting through the different layers of obfuscation which didn’t really have any theoretical security guarantees. When compared to Wynk, the other 3 services, JioSaavn, Gaana and Hungama had simpler mechanisms for content serving which were reversed quite effortlessly. The patch implemented by Wynk after our disclosure shows that they are indeed concerned about protecting their content and so it would be quite interesting to see how these platforms embrace DRM in the future instead of working on stop-gap solutions which only delay the inevitable.

We present a summary of the best/worst practices in Table 2 based on our investigations.

| Practice                | Spotify | Wynk | JioSaavn | Gaana | Hungama |
|-------------------------|---------|------|----------|-------|---------|
| Mandatory User Identification | ✓       |       |          |       |         |
| Streamed Content Encryption | ✓       | ✓    |         |       |         |
| Hardcoded Keys          | ✓       | ✓    | ✓       |       |         |
| Cookie Based Authentication | ✓      | ✓    |         |       |         |
| Timeout                 | ✓       | ✓    |         |       |         |
| Premium Content Access Restrictions | ✓ | ✓ | ✓ | ✓ | ✓ |
| Obfuscation/Minification | ✓       | ✓    | ✓       | ✓     | ✓       |

Table 2. A Comparative Analysis of Practices

5.2 Why No DRM?

What might be the possible reasons that could have led to such an oversight, we wonder. DRM protection is not exactly a new or novel concept and has been a part of the industry for a fair amount of time. This would imply that deploying these services with such primitive security measures was a
conscious decision at its worst and a rushed one at its best. A possible reason that comes to mind is the shaky compatibility of the discussed DRM schemes with current playback devices which could result in the alienation of a large chunk of the subscriber base. In a massive, competitive market such as India, this could potentially spell the demise of a competing streaming service at the hands of its rivals.

5.3 An Inherent Problem of the Analog Hole [47]

For a considerable amount of time now, there has been a raging debate on the effectiveness of DRM solutions. The aptly named analog hole is a problem inherent to the task of protecting Audio/Video Content and is something that is touted by DRM critics all the time and for good reason. Put simply, the analog problem can be illustrated as follows - Imagine an ideal system that is completely secure in terms of communication and implementation. The client receives the encrypted content from the server and the decryption process occurs securely following which, the content is displayed by the screen or played by the speakers. The analog hole problem states that human perceivable analog signals can always be recaptured and re-encoded to a digital format, thereby nullifying all previous protection. As an example, a pirate can always re-capture video content by directing the content to a video card instead of a screen and similarly tapes/mics can be used to re-capture audio content.

The proponents of DRM however argue that since all analog to digital conversions are lossy, one can never actually retrieve the original content by exploiting the analog hole. We leave it to the readers to decide which side of the fence they lean on.

5.4 Attacks On Widevine

Widevine is a DRM system developed by Google in order to protect content from misuse by the client. L3 level Widevine is supported by the latest versions of most major browsers and in our case, it is used by our benchmark - Spotify. When we chose to treat Spotify as the ideal service with content protection we did so by blindly trusting Widevine based on its popularity. We attempt to rectify our assumptions in this section.

There have been multiple reports of researchers breaking L3 level Widevine. The first such claim was made by a security researcher David Buchanan in a tweet [17] in January 2019. This was followed by a blog from Fidus Information Security [43] who claimed to have decrypted an episode of Stranger Things from Netflix which used L3 Widevine. This group claims to have used a modified Differential Fault Analysis (DFA) approach to recover the keys used for decryption by the Widevine Module. Despite making claims, neither of the security researchers published working PoCs or exploits in order to prevent misuse. Speaking of misuse, a black-hat group called The Scene also claims to have a working break [39] which they use to pirate content off of Amazon Prime and Netflix.

5.5 StreamRipping

In this section we discuss an alternate attack strategy that continues to be very popular among pirates [36]. This attack strategy, called StreamRipping exploits the un-encrypted nature of the streamed content within a browser. Essentially, once the decryption from the HTTPS layer has occurred, if the content itself is not encrypted, it is visible to the browser and hence the attacker. This content can then be simply dumped to a file for later distribution - thus ripping from the stream.

There are various tools and services which make it easy to StreamRip. An example would be the browser add-on, Audio Downloader Prime [37] for Mozilla Firefox. We found this tool to be quite effective in seamlessly capturing the content streamed from JioSaavn and Hungama where the media was being served as in response to a single request. In the case of Wynk, the chunks of the stream were detected and were retrievable as .mp3 files.

5.6 Mitigation Strategies

Following our analysis, we found most of the streaming services were guilty of malpractices (Refer Table 2). Further, our success in completely reverse engineering the protocols shows without a doubt that shallow patches will not prove to be secure in the long term. Short term strategies might include stronger obfuscation, reliance on precompiled binaries etc. but none of these techniques would stand the test of time. As of today, the best possible protection available is through DRM schemes like Widevine, Playready, Fairplay and given the ease of setting up these schemes, it might turn out to be the best possible long term strategy as well.

6 Implementation Details

After much deliberation and thought over the fact that a working PoC could potentially be misused, we decided to exclude it from this work. As our experience with Wynk taught us, unless radical changes are made to those services, their protocols can potentially always be broken and data stolen on a massive scale. Our work illustrates this very fact. Ultimately we hope that these services deploy proper DRM measures and not a workaround patch that will perpetuate this game of cat-and-mouse.

7 Conclusion

In this paper we surveyed various OTT Music streaming services, particularly those of Indian origin and analysed their attack surface. Owing to their growing popularity as the de facto standard of media consumption, our observations led us to conclude that most streaming services were
highly vulnerable to basic stream ripping and reverse engineering attacks. To present our case, we analyzed four most popular music streaming apps in India namely Wynk, Gaana, JioSaavn and Hungama and were successfully able to steal protected content without restrictions. In case of Wynk, we would like to emphasize the fact that even after the attacks had been disclosed, the patched version of their protocol was broken again using the same techniques, clearly showcasing the pitfalls of doing patchwork instead of adopting a systematic solution. A detailed comparative study is furnished to show the extent of deviation from the state-of-the-art and possible mitigation strategies are also proposed. Through our work, we hope that such platforms take cognizance of the lax security measures in place and improve upon them.

A Wynk Music Function PseudoCodes

A.1 Wynk v1

Algorithm 1: Client Side in Wynk 1.0

Input: url; /* Resource URL */
Output: [chunks]; /* Audio Chunk List */
1 begin
2 Initialise
3 Initialization
4 deviceID, userAgent ← Arbitrary strings
5 begin Authentication
6 /* Authorisation with the Wynk Servers to enable authenticated retrieval requests */
7 uid, token ← register(deviceID, userAgent)
8 search_id ← get_search_id(url)
9 C ← request_manifest(token, search_id)
10 begin Retrieval
11 [chunks] ← get_song(C)

Algorithm 2: register() Function

Input: deviceId, userAgent
Output: uid, token
Data: Request Headers: \(H\)
1 begin
2 url := "https://sapi.wynk.in/music/v3/account/login"
3 payload := \{"deviceId": "deviceId", "userAgent": "userAgent"\}
4 uid, token ← POST(url, \(H\), payload)
5 return uid, token

Algorithm 3: search_id() Function

Input: url
Output: search_id
Data: STATIC_CPMAPPING[ ]
1 begin
2 /* Each URI on Wynk has the following format - \"producerid\_string\" */
3 Extract producerid_string from url
4 id ← STATIC_CPMAPPING[producerid]
5 return (id | "_" | string)

Algorithm 4: request_manifest() Function

Input: search_id, token, uid
Output: Signed CloudFront Parameters
Data: Request Headers: \(H\)
1 begin
2 /* Generating header x-bsy-utkn */
3 prefix := "/streaming/v4/cscgw/"
4 suffix := ".html?ets=true&hlscapable=1&q=a&lang=en"
5 subdomain := "https://playback.wynk.in"
6 msg := "POST" || prefix || search_id || suffix
7 digest ← SHA1_HMAC(token, msg)
8 \(H\)\[x-bsy-utkn\] ← uid||"":"||Base64Enc(digest)
9 auth_url ← subdomain || prefix || search_id || suffix
10 C ← POST( auth_url, \(H\), payload: "{\"")
11 return C

Algorithm 5: get_song() Function

Input: C; /* Response Object Containing Authenticated URIs & Signatures */
Output: [chunks]
1 begin
2 manifest_url ← Extract manifest file URI from C
3 manifest_file ← GET(manifest_url)
4 index_uri ← Identify and extract highest quality index.m3u8 URI
5 index_file ← GET(index_uri)
6 chunks = [ ]
7 forall chunk_url in index_file do
8 tmp ← GET(chunk_url)
9 chunks.push[tmp]
10 return chunks
A.2 Wynk v2

Algorithm 6: Client Side in Wynk 2.0

Input: url; /* Resource Url */
Output: [chunks]; /* Audio Chunk List */

begin
Initialization
    BK ← gen_bk(T, R)
end

deviceid ← gen_random_id(R)

pk = Base64enc(https://sapi.wynk.in/music)

sk = 51yWn1mS

begin
Authentication
    /* Authorisation with the Wynk Servers to enable authenticated retrieval requests */
    spit_out(BK, deviceid)
end

U := \{k, n, y, w, m, z, a, p\} → check(BK, T)
M := \{dt, uid, token, kt, ...\} → login(U, T)

C ← request_manifest(url, token, dt, sk)

begin
Retrieval
    [chunks] ← get_song(C)
end

Algorithm 7: spit_out() Function

Input: BK, deviceid

begin
    d1, d2 ← deviceid[0...36], deviceid[36...72]
In d1, d2 replace "-" → ""
    d3, d4 ← mix_it(d1, BK), mix_it(d2, BK)
end

url := "https://img.wynk.in/webassets/

GET(url||d3)||".1.jpg")
GET(url||d4)||".2.jpg")

Algorithm 8: check() Function

Input: BK
Output: k,n,y,w,m,z,a,p
Data: Request Header: \( \mathcal{H} \)

begin
    url := "https://ping.wynk.in/health/check"
\( \mathcal{H} \)[tk] ← T
\( \mathcal{H} \)[bk] ← BK[0, len(BK)/2]
p ← BK[1, len(BK)/2, len(BK)]
U ← POST(url, \( \mathcal{H} \), payload: \{"pid" : "p"\})
return U
end

Algorithm 9: login() Function

Input: k, n, y, w, m, z, a, p
Output: dt, uid, token, kt, ...
Data: Request Header: \( \mathcal{H} \)

begin
    url := "https://login.wynk.in/music/account/v1/login"
    BS := k||n||y||w||m||z||dt||p
    \( \mathcal{H} \)[x-bsy-ptot] ← T
    /* Generate x-bsy-cip from BS value */
    a ← [], b ← 0, t ← 0
end

for \( t \leq \text{len}(BS) - 1 \) do
    e = 10(BS[t]) + BS[t + 1]
    if e ≤ 55 then
        if 2 \( \not\mid \) b then
            a.push(200 + e)
        else
            a.push(100 + e)
    else
        a.push(100 + e)
    b++
\( \mathcal{H} \)[x-bsy-cip] ← concat(a)
C ← POST(url, \( \mathcal{H} \), payload: \{
return C
end

Algorithm 10: request_manifest() Function

Input: url, dt, uid, token, kt
Output: Authenticated CloudFront Resource
Parameters
Data: Request Header: \( \mathcal{H} \)"

begin
\( \mathcal{H} \)"[x-bsy-uid] ← dt
    /* Generating header x-bsy-utkn */
suffix := "/song/v4/stream?ets=true&hlscapable=
1&sq=a&lang=en&id="
search_id ← get_search_id(url)
msg := "POST"|| suffix || search_id || "{""
digest ← SHA1_HMAC(token, msg)
\( \mathcal{H} \)"[x-bsy-utkn] ← uid||":"||Base64Enc(digest)
/* Generating x-bsy-t using Time Based
OTPs and CryptoJS AES */
\( \mathcal{H} \)"[x-bsy-t] ← AES(kt, TOTP(dt||sk, 600, 6))
/* Send POST Request To Server */
X ← POST(url, \( \mathcal{H} \"), payload: \{
return X
References

[1] [n.d.]. Adobe Primetime. https://www.adobe.com/in/marketing/primetime.html
[2] [n.d.]. Amazon Music. https://music.amazon.in/home
[3] [n.d.]. Apple Fairplay Streaming. https://developer.apple.com/streaming/fps/
[4] [n.d.]. Apple Music. https://www.apple.com/in/music/
[5] [n.d.]. Denuvo By Irdeto. https://irdeto.com/denuvo/
[6] [n.d.]. Hungama. https://www.hungama.com/
[7] [n.d.]. Marlin DRM. https://www.intertrust.com/products/drm-system/marlin-drm/
[8] [n.d.]. Microsoft PlayReady. https://www.microsoft.com/playready/
[9] [n.d.]. PortSwigger Burp Proxy. https://portswigger.net/support/using-burp-proxy
[10] [n.d.]. Shaka Player. https://github.com/google/shaka-player
[11] [n.d.]. Spotify. https://www.spotify.com/in/
[12] [n.d.]. Shaka Player. https://github.com/google/shaka-player
[13] [n.d.]. Widevine. https://www.widevine.com/
[14] [n.d.]. YouTube Music. https://music.youtube.com/
[15] Alex Biryukov, Gaetan Leurent, and Arnab Roy. 2013. Cryptanalysis of the "Kindle" Cipher. In Selected Areas in Cryptography, Lars R. Knudsen and Huapeng Wu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 86–103.
[16] Bruce E. Boyden. 2011. Is DRM working?: how could we tell?. In Proceedings of the 11th ACM Workshop on Digital Rights Management, Chicago, Illinois, USA, October 21, 2011, Yan Chen, Stefan Katzenbeisser, and Ahmad-Reza Sadeghi (Eds.). ACM, 1–2. https://doi.org/10.1145/2046631.2046633
[17] David Buchanan. 2019. Retrieved October 4, 2020 from https://twitter.com/David3141593/status/1080606827384131590
[18] Deloitte India. [n.d.]. Audio OTT Economy in India. https://www2.deloitte.com/in/en/pages/technology-media-and-telecommunications/articles/OTT.html
[19] Cory Doctorow. 2019. After Years of Insisting that DRM in HTML Wouldn’t Block Open Source Implementations, Google says It Won’t Support Open Source Implementations. Retrieved October 4, 2020 from https://boingboing.net/2019/04/03/i-hate-being-right-2.html
[20] David Dorwin, Jerry Smith, Mark Watson, and Adrian Bateman. 2017. Encrypted Media Extensions. Technical Report.
[21] EC-Council. 2013. Computer Forensics: Investigating Network Intrusions and Cybercrime. Cengage Learning.
[22] Financial Express. 2019. Audio OTT contributes 70% to music industry revenue: Deloitte-IML. https://www.financialexpress.com/industry/deloitte-iml-audio-ott-contributes-70-pct-to-music-industry-revenue-1695436/
[23] Google. 2017. Widevine DRM Architecture Overview. Retrieved October 4, 2020 from https://www.whymathatica.com/wp-content/uploads/2018/08/Widevine_DRM_Architecture_Overview.pdf
[24] Hennie van Kuieren. [n.d.]. The Preference for Music as a Service as Compared to Downloaded to Own. https://www.inholland.nl/media/10674/masterthesis-hennie-van-kuieren-the-preference-for-music-as-a-service-a.pdf
[25] Apple Inc. 2017. HTTP Live Streaming. Technical Report. https://tools.ietf.org/html/draft-pantos-http-live-streaming-23#section-3.3
[26] Apple Inc. 2020. Understanding the HTTP Live Streaming Architecture. Retrieved October 4, 2020 from https://developer.apple.com/documentation/http_live_streaming/
[27] International Federation of the Phonographic Industry. 2018. IFPI Global Music Report 2018. https://www.ifpi.org/downloads/GMR2018.pdf
[28] International Federation of the Phonographic Industry. 2018. IFPI Music Consumer Insight Report. https://www.ifpi.org/downloads/Music-Consumer-Insight-Report-2018.pdf
[29] B. Kaliski. 1998. PKCS #7: Cryptographic Message Syntax. Technical Report. https://tools.ietf.org/html/rfc2315
[30] Kate Swanson. 2013. A Case Study on Spotify: Exploring Perceptions of the Music Streaming Service. MIEEA Journal 13, 1 (2013).
[31] Renuka Kumar, SreeShishore, Hao Lu, and Atul Prakash. 2020. Security Analysis of Unified Payments Interface and Payment Apps in India. In 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 1499–1516. https://www.usenix.org/conference/usenixsecurity20/presentation/kumar
[32] Airtel Digital Ltd. 2020. Wynk Music - Homepage. Retrieved October 4, 2020 from https://wynk.in/music
[33] Gamma Gaana Ltd. 2020. Gaana - Homepage. Retrieved October 4, 2020 from https://gaana.com/
[34] Saavn Media Pvt Ltd. 2020. JoSaavn - Homepage. Retrieved October 4, 2020 from https://www.jiosaavn.com/
[35] Vaiavni Mahendra. 2019. Hybrid OTT and music streaming platforms get more users than 100% subscription-based platforms. Retrieved October 4, 2020 from https://brandequity.economictimes.indiatimes.com/news/media/hybrid-ott-and-music-streaming-platforms-get-more-users-than-100-subscription-based-platforms/0939360
[36] Colin Mann. 2020. Study: Stream-ripping piracy on the rise. Retrieved September 17, 2020 from https://advanced-television.com/2020/09/17/study-major-increase-in-stream-ripping-piracy/
[37] mark. 2019. Audio Downloader Prime. Retrieved September 21, 2019 from https://www.jiosaavn.com/
[38] Markets and Markets. 2020. Over-The-Top Services Market by Type (Online Gaming, Music Streaming, VoD and Communication), Monetization Model (Subscription-based, Advertising-based, and Transaction-based), Streaming Device, Vertical, and Region - Global Forecast to 2024. TC2445 (2020). https://www.marketsandmarkets.com/Market-Reports/over-the-top-ott-market-41276741.html
[39] Andy Maxwell. 2019. The Scene: Pirates Ripping Content From Amazon & Netflix. Retrieved October 4, 2020 from https://torrentfreak.com/the-scene-pirates-ripping-content-from-amazon-netflix-190707/
[40] U.S. Government Publishing Office. [n.d.]. Public Law 105–104–Digital Millennium Copyright Act. Retrieved October 4, 2020 from http://www.gpo.gov/fdsys/pkg/PLAW-105publ104/html/PLAW-105publ104.htm
[41] Traci Ruether. 2019. 2019 Video Streaming Latency Report. Retrieved October 4, 2020 from https://www.wowza.com/blog/2019-video-streaming-latency-report
[42] Russ Crupnick. [n.d.]. Thanks to Stream-Ripping, Music Piracy Still a Scourge. https://www.musicwatchinc.com/blog/thanks-to-stream-ripping-music-piracy-scar Still-a-scourge/
[43] Fidus Information Security. 2019. A Primer on Widevine and How It Can Be Abused to Download Encrypted Movies/Shows. Retrieved October 4, 2020 from https://fidusinfosec.com/breaking-contents-protection-on-streaming-websites/
[44] Robert Triggs. 2019. Widevine Digital Rights Management Explained. Retrieved October 4, 2020 from https://www.androidauthority.com/widevine-explained-829353/
[45] Ruoyu Wang, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2013. Steal This Movie: Automatically Bypassing DRM Protection in Streaming Media Services. In USENIX Security Symposium. USENIX Association, 687–702.
[46] Ruoyu Wang, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2013. Steal This Movie: Automatically Bypassing DRM Protection in Streaming Media Services. In 22nd USENIX Security Symposium (USENIX Security 13). USENIX Association, Washington, D.C., 687–702. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/wang_ruoyu
