Security Analysis of WideVine as a Digital Rights Management System.

K Markantonakis

Darren Hurley-Smith

Carlton Shepherd

Mohamed Yusuf

1. Abstract

Piracy has always been a key point of concern for content producers and content delivery networks since the development and open distribution of entertainment media. Hackers and other individuals with malicious intent have produced ample ways to conduct piracy to re-distribute such media, effectively diverting traffic away from legitimate distribution networks. In retaliation to that, content delivery networks (CDN) have produced methods and technologies to efficiently mitigate that threat. None of these methods entirely protect piracy as users still have the option of using certain screen recording software, which easily bypass protection against screen recording media. However, like any other cyber security policy the goal is not to always make a system impenetrable; sometimes, the goal is to make the system strong enough to make penetration infeasible. Similarly, screen recording can pose the following disincentives:

- Quality of recorded audio and video content suffer drastically.
- Recording must be done without any interruption on the screen. (For e.g., Browser popups).
- Recording long TV shows/ movies may be extremely time consuming.

Digital Rights Management (DRM) systems have become quite popular amongst CDN's as they seem completely fool proof, and the methods that may be used to attack them are highly sophisticated, and infeasible for most individuals who would have any reason to perform this kind of attack.

WideVine, a popular DRM solution being the subject of this document has been through some rigorous security testing and analysis and has also been exploited by David Buchanan *et al* using a well-researched Differential Fault Analysis attack.

In this work we provide an in-depth analysis of the efficacy of WideVine as a DRM solution. Most importantly we will discuss all the tests we have conducted on the WideVine system and analyse the effectiveness of its "security through obscurity" approach (3).

2. WideVine

2.1. CENC (Common Encryption)

Before describing CENC common encryption, a brief discussion of why it was implemented is paramount.

Previously, there have been two main protocols in play by CDN (Content Delivery Networks) for delivering adaptive bitrate video. MPEG-DASH and HLS (HTTP Live Streaming). Because both these protocols were supposed to be supported by all content delivery networks, they had to keep all video content packaged individually in formats supported by those protocols (MPEG-DASH supports mp4 while HLS supports m3u8) this increased the space required 2-fold. Additionally, we must consider the different encryption standards supported by the different DRM solutions (PlayReady, FairPlay), that further increased the space required on the servers and was just unnecessarily inefficient.

To tackle the file format problem came the CMAF specification (Common Media Application Format) (Apple Inc., 2018).The CMAF specification provides a method of encoding and packaging media such that all media in different formats can be fragmented and stored in one containerized fmp4 format.

To tackle the encryption problem came the CENC- or the common encryption standard developed by MPEG which specifies common encryption formats for use in any file format based on the ISO/IEC 14496-12. Now, videos and other media could be encrypted using CENC (AES-128 Counter) or the CBCS (AES-128 Cipher Block Chaining) (ISO, 2016).

With CENC common encryption methods came the need to introduce White-Box cryptography because CENC required the CDN (Content Delivery Networks) to encrypt the videos and other media only once, in which case the decryption would be done through a CDM (content decryption module). A complete and detailed examination of CDM and White-Box Cryptography is done later in this document (2.2).

2.2. Content Decryption Module

The Content Decryption Module is one of the most crucial elements of any DRM solutions as this is where all the decryption, and possibly decompression, of media content takes place. As such a CDM would also be extremely beneficial testing avenue when it comes to penetration testing the system. The CDM handles the following tasks:

- Create a license request (4.2.1).
- Decrypt the license response with the White-Boxed private-key and extract the private key that will be used to decrypt the video.
- Store the license response on the device for the duration specified by the license specification.
- Deliver the decrypted content to the player safely (without leaking any keys).

Content Decryption Modules are often black boxed as they store very confidential data, for example the private keys to decrypt the license response. The WideVine CDM comes pre-installed with chrome and the DLL file in Windows PCs (Personal Computers) can be found in one of the two directories listed below:

- %UserProfile%\AppData\local\google\chrome\user data\WideVineCdm\
- C:\Program Files\Google\Chrome\Application\<chrome version>\WideVineCdm\

Given that such a confidential file is exposed to the end user, reverse engineering it may sound like a very trivial process, however this is not the case for the following two reasons:

The use of very heavy code obfuscation (3.3), and white box cryptography (2.7).

A brief introduction to code obfuscation is provided in this document but Sebastian Schrittwieser *et al* go into intricate details about it in their paper. (Schrittwieser)

2.3. Encrypted Media Extension

Security best practices dictate that the decryption of any media will not be done in the browser. This is because the browser in not in the trusted layer of the processor, which will allow anyone to view the source code of the browser and see how the decryption is done. Instead, the media will be sent to the CDM for decryption.

This transfer of data is where the Encrypted Media Extension is utilised. The EME (Encrypted Media Extensions) is a built-in extension that comes with Google Chrome and acts a communication medium between the CDM and the HTML5 video player. The EME is a JavaScript API (Application Programming Interface) that comes with certain functions that allows the HTML5 video player (which is also built in JavaScript) to make certain calls to the CDM.

More focus and attention should be paid to the EME as this document will explore how EME could be used to forge communication between the player and the CDM, in turn giving an attacker control over the entire system (3.4).

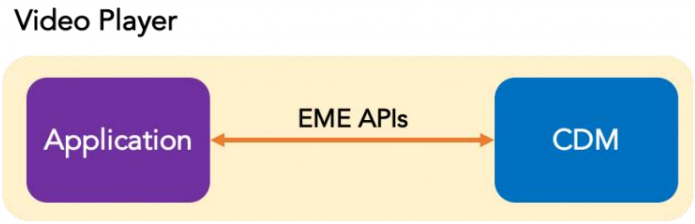

Figure 1 (Vijayanagar, EME diagram)

2.4. DASH (Dynamic Adaptive Streaming over HTTP)

Once the complete setup of authenticating the session with the CDN (Content Delivery Network) is done, comes the issue of efficiently sending playback to the user without have any lags due to limited network resources on the users end. This is where the DASH (Dynamic Adaptive Streaming over HTTP) protocol comes in. With DASH, the content is broken down into a sequence of small segments and the segments are then delivered to the end user.

The browser is responsible for identifying the network resources available to the end user and based on that it calculates a suitable specification for the video chunks, this is what makes the entire system dynamic. Given that commonly the most high-quality video content is made at 24 frames/second the end user must have enough network capabilities to download video at least 24 frames of content per second. If this is not the case, the browser suggests the CDN to deliver even smaller chunks of video, so the end user has a smooth experience without lag. This means that the frequency at which the videos will be downloaded will be much greater than usual.

In terms of having an international standard of streaming content over HTTP, MPEG-DASH is the solution. This standardization has helped to have a much more supportive environment for dynamic streaming of HTTP. Even though MPEG-DASH is not directly supported by HTML5 the MSE (Media Source Extension) in most

browsers allow for this easily. Additionally, Netflix and other CDNs (Content Delivery Networks) have their own implementation of video players that support MPEG-DASH (Stockhammer).

.

Product	Product Type	Platform	Live Streaming	DRM-Free	As of Version	Editor
Microsoft Edge	Web browser	Windows 10	Native support on Edge Legacy. Support via Media Source Extensions on Edge Chromium.	No	Supported natively on Edge Legacy's engine EdgeHTML from version 12 to 18.[53] No native support on Edge Chromium from version 79 to present.[54]	Microsoft
VLC media player	Media player	Windows, macOS, Linux, Android, iOS, Windows Phone	Yes	Yes	v3.0	VideoLAN
Dash.js	SDK	HTML5 (MSE Browsers)	Yes	No	—	Dash Industry Forum
Shaka Player	SDK	HTML5 (MSE Browsers)	Yes	No	—	Google
Rx-Player	SDK	HTML5 (MSE Browsers)	Yes	No	—	Canal+

Figure 2 (Wikipedia)

2.5. The Shaka packager

/*

Description.

*/

2.6. The License Server

One of the key elements of most DRM solutions, specifically WideVine is the License Server. The main objective of having a license server is to maintain the confidentiality of the symmetric keys that may be used to encrypt and decrypt the media content from the content delivery networks. The License Server is responsible for authenticating License Requests, and based on those License Request packets, generate License Responses. The License Servers used by WideVine are also the key stores. The job of the key store is to map **keyids** (specific to each video/media snippet) to their respective private key.

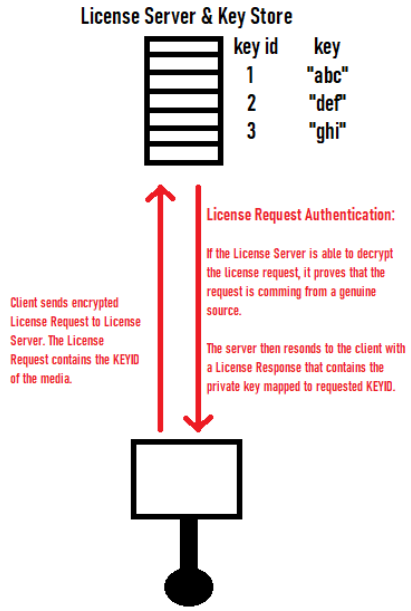

Figure 3 Simplified abstraction of the license request-response

The protocol used for the transmission of the license request and responses is simply HTTPS which means the communication between the client and the license server also happens in the browser while the CDN is being loaded.

2.7. White Box Cryptography

Implementing a DRM system may come with its own set of cryptography related questions, while some of these questions are easily answered through good old symmetric encryption algorithms, others need some more thought. The main issue with DRM systems is the fact that the end consumer of the system is also the stakeholder with the most amount of access to the system. In other words, the system in question is working with complete privileges in a potentially hostile environment- such an environment is known as a white-box environment. The AES algorithms or any other block cipher algorithms will only work flawlessly if both holders of the private keys have the same interests with regards to the security of the system. With a DRM solution like WideVine, it is not possible. At least, not without the use of white box cryptography or something similar.

"This is a common problem for software-based applications running on PC's, IPTV set-top boxes and other data consuming devices attempting to enforce DRM. By actively monitoring standard cryptographic APIs or memory dumps, hackers are then able to extract the key(s) whenever used. One example of a successful memory-based key extracting attack has enabled the BackupHDDVD

tool to copy the content of a protected DVD and remove the DRM from Windows protected media content" (SafeNet).

White-Box cryptography was first proposed by Chow *et al* in 2002 for the implementation of the AES (Advanced Encryption Standard). The proposition was later subjected to attack presented by Ech-Chatbi *et al* which led to the development of more secure white-box cryptography implementations with greater resistance against key extractions. White-box cryptography aims to transform a cryptographic primitive into a functionally equivalent software application, so it behaves like a virtually black-boxed application. In such a case a white-box attacker with full and complete access to the decryption of content, the private keys and inputs and the outputs of the system would still have no additional advantage over a black-box attacker that tests solely inputs and outputs.

In order to understand the concept and the strategy used to achieve white-box cryptography, looking at the first implementation of white-box AES is a good idea. The paper published in 2002 by Stanley Chow *et al* goes into extreme and intricate detail as to how prevention of key extraction can be achieved (Chow, 2002). Chow *et al* proposed that the rounds of encryption and decryption of the AES (128-bit at the time) must be broken up into steps and re-composed after inserting randomly generated bijections (functions where |domain|=|range|). This serves as an *internal encoding* in addition to the *external encodings* encompassing the overall cipher.

In essence, the technique proposed by Chow *et al* comprised of two major parts. The first part is where the key-instantiated block cipher is translated to a functionally equivalent array of lookup tables by mixing them with the steps of the round functions. The second part concerns the injection of randomness obtained in the first step. At the end of both steps the result is a cipher that is completely

ambiguous and random and in theory it would be absolutely infeasible to extract private keys from this.

In conclusion, even though white-box cryptography seems extremely intuitive and even fool proof it has been attacked numerous times and is far from what it seems. There have been multiple popular attacks that have been successful in extracting keys from white-box cryptography implementations. Differential Fault Analysis, Differential Computation Analysis, Spectre, and other side channel attacks are a few potential avenues that must be explored when attacking a white-boxed implementation. This document goes through Differential Computation Analysis in detail as one potentially successful exploit against WideVine (3.5) (Mulder, 2014) .

2.8. WideVine Security Levels

For efficient, smooth, and secure playback WideVine is designed to support 3 levels of security. The judgement of which security level to use must be done by the browser and the EME (encrypted media extension).

2.8.1. L1

The Security Level 1 is the most sophisticated security implementation provided by WideVine. At this level, the content processing, and the decryption of the media is performed within the TEE (Trusted Execution Environment) or sometimes in different chips altogether. The security level 1 is mostly used by devices that run Android or ChromeOS.

2.8.2. L2

While in L1, both content processing and decryption are done within the TEE; in L2, only the decryption is done in the TEE.

Here, the decrypted media is sent to the application domain to be processed. This is still significantly more secure compared to L3 as the cryptographic secrets are stored in the TEE.

2.8.3. L3

The security level 3 being the least secure implementation also requires less sophisticated software on the users end and hence is the most widely used. In L3 all content processing and decryption is done in the content decryption module thus appropriate measures must be taken to protect the private keys. This is done through white box cryptography and code obfuscation.

Research around breaking trusted execution environments is inconclusive for the most part, so the scope of this document is limited to the security analysis of L3 only.

2.9. Trusted Execution Environment

3. Exploitation

3.1. Publicly Available Exploit Code

3.1.1. Testing On Older Browser Versions

3.2. Reproducing Exploit on Other Browsers

3.3. Static Analysis with Ghidra

3.4. Hijacking EME calls to the CDM

*/

Downloaded media could be analysed to try and extract keyID and other data to form a license request by sending that to the CDM. The CDM could then be used to generate a license request and receive a response with a key which could then be used to decrypt the installed content. Articulate this and explain how this could be done.

*/

3.5. Differential Computation Analysis

A meticulous examination of white-box cryptography and its application in a DRM solution like WideVine has been done in this document (2.7). This section focuses on the theoretical concept behind the DCA (differential fault analysis) attack and why the DCA attack seems to be the most suitable form of attack in this scenario.

Through this document it has been made clear about the existence of a private key that is used to decrypt license responses. The key is protected through means that have been pointed out earlier. The goal of a differential computation analysis attack is to somehow extract that private key from the CDM (content decryption module). Since the idea behind the implementation of white-box cryptography is that the adversary is usually the user of the device running the software (WideVineCdm) implementation in question, with that information it can be assumed that the adversary can among other things perform static and dynamic analysis on the software implementation and inspect the memory accesses, system calls, and even alter immediate results by injecting data amidst those function calls.

Mentioned earlier in this document in section (3.3) the issues faced when statically analysing the target binary file. The use of code obfuscation made it extremely hard to properly understand the system calls that were being made and which functions were used to decrypt the license response. As a result of this, a better approach would be to conduct *dynamic analysis* on the binary file instead of

static analysis. DBA (dynamic binary analysis) is often used to improve and inspect the quality of a software implementation and in this case control the runtime state of the white-box implementation. Joppe W. Bos *et al* introduces DBI (dynamic binary instrumentation) in a detailed paper describing *differential computation analysis* (Joppe W. Bos, 2015). Dynamic binary instrumentation involves the adversary analysing the binary files by adding additional analysis code into the original code of the binary file to aid in memory debugging and profiling. Valgrind (Nicholas Nethercote) is a tool that can be used to perform such levels of binary analysis

4. Comparative Analysis Between WideVine and PlayReady

Like WideVine, PlayReady is another DRM solution proposed by Microsoft. This section of the paper goes through a detailed investigation of the differences in technologies utilized by both the competing products. Fortunately, Microsoft has a history of having particularly good and detailed documentation of their products and services and clearly PlayReady is not an exception. The documentation provided by Microsoft for PlayReady goes into intricate detail for every specific technology used by PlayReady on both the server side as well as the client side including references to other related documents and even code snippets.

4.1. Security Levels

In terms of security levels PlayReady is much more portable compared to WideVine when looking at its most secure design I.e., SL3000. Even though PlayReady supports a low security implementation, it is only used for testing, or on client machines that are under development (Microsoft, PlayReady Security Levels).

4.1.1. SL150

Comparable to WideVine's L3 implementation is the SL150 for PlayReady which can be used on any client implementations without any specific software or hardware requirements. At this level, any client secrets such as private keys or even decrypted media are not protected against unauthorized use. SL150 is not used commercially at all as no measures have been taken to protect the confidentiality of private keys.

4.1.2. SL2000

PlayReady implements software means to protect private keys and decrypted media. These include the use of white box cryptography and code obfuscation. This is a much less secure way of protecting that data. While WideVine's L3 uses those methods it is hence a much more suitable comparison. This level is also available on all devices.

4.1.3. SL3000

This is the most secure implementation of PlayReady and at this level it uses Trusted Execution environment (TEE) to protect private keys and decrypted media, perform decryption and process the media content. The only significant difference between WideVine's L1 and SL3000 is the portability. SL3000 is compatible with every device that meet a certain hardware requirement, while L1 only works on Android and ChromeOS devices.

4.2. License Acquisition

4.2.1. WideVine

Like mentioned earlier in this document the use of EME (Encrypted Media Extensions) as a communication medium between the DRM (Digital Rights Management) server and the CDM (Content Decryption Module); here, there is a meticulous examination of how the EME API (Application Programming Interface) is used in the license acquisition process in WideVine (WideVine, 2017), (WideVine, 2013).

The entire process starts when the client visits a CDN application protected by WideVine.

When the video is loaded by the player the player recognizes that the content is encrypted. With the content comes the INIT_DATA and some other meta data that are used later in the license acquisition process. The player makes the following EME API calls during the license acquisition process:

- Navigator.requestMediaKeySystemAccess() - This is made to find the exact path to the key system that will be used for the license acquisition.
- CreateMediaKeys() - CreateMediaKeys() call is made to initialize a MediaKeys object. This is where the keys are kept.
- CreateMediaKeySession() - This call is made to create a session between the player and the content decryption module. This function is called on the MediaKeys object.
- GenerateRequest() - Once a session object is created the next step is to make a LICENSE_REQUEST generation call to the session object. This is where the INIT_DATA comes into picture. The generateRequest() call must go with the keyID of the media content in question. The CDM is then used to create an encrypted LICENSE_REQUEST that is passed on to the DRM server through the player.

- Update() - Once the DRM server is able to validate the LICENSE_REQUEST it sends over to the player a LICENSE_RESPONSE. The player makes the update() call to the EME API to send over the LICENSE_RESPONSE to the CDM. This completes the license acquisition process.

Through this license acquisition process there are only two cases in which the private keys are being transmitted through the player. The first is when the content decryption module sends the license request to the player to forward it to the license server and the second is when license server responds with a license containing the content keys. It is important to note that in both these cases the keys are either encrypted on in byte buffer format.

Source: https://www.vdocipher.com/blog/2018/11/encrypted-media-extensions-eme/

4.2.2. PlayReady

In the case of PlayReady, when a video is loaded the DRM components on the user's device first search for a license within the license store and a license challenge (similar to a license request) is only generated if the DRM components fail to find a license within the license store.

The licensing of content from the License Server starts as the content producer uses a **license key seed** and a **keyid** and based on those values generates an encryption key. Once the encryption key is generated, it is used to encrypt the DRM content which is later sent to the client. The user's media player then queries the CDM (depending on the level of security implementation. In most cases this would be at the TEE (Trusted Execution Environments)). As mentioned earlier, after this stage the CDM would then query the license store for

existing licenses to play the content. A license is only requested if the CDM fails to find an existing valid license. The license request contains the following data:

- Content Header
- Device Information (Device Id)
- Content Information (Key Id)

The License Server now uses the **keyid** and the **key seed** and generates the same content key that was generated by the CDN earlier. The content key is then encrypted and added to the license with other license information such as the validity period of the license, the rights of the license and the right restrictions and right modifiers (also known as the conditions of the license). The license is then signed by the private signing key and sent over to the end user.

Once the license response is received by the user, it is stored in the license store and can be used to decrypt media whenever needed. The decryption of the license response is done using a pre-existing key that may be stored in the CDM or other DRM components based on the security level of the system using Whitebox cryptography.

In terms of transmission methods used by play ready in order to send and receive license packets; they are mostly carried out through the HTTPS protocol over the internet or on a closed network. The transaction can also be carried asynchronously which would mean that the license requests and responses are posted to separate locations which are then accessed by the respected parties. The protocol used for the structured delivery of the packets is SOAP (Simple Object Access Protocol) similar to the predecessor XML-RPC (Microsoft, PlayReady License Acquisition), (Microsoft, PlayReady License And Policies).

References

1. *Apple Inc. (2018). Common Media Application Format. Los Altos, California: Apple Inc. Retrieved from https://developer.apple.com/documentation/http_live_streaming/about_the_common_media_application_format_with_http_live_streaming*
2. *Chow, S. (2002). The First Proposed Implementation of White Box Cryptography. Retrieved from https://link.springer.com/content/pdf/10.1007%2F3-540-36492-7_17.pdf*
3. *ISO. (2016). ISO/IEC 23001-7:2016. Retrieved from https://www.iso.org/standard/68042.html*
4. *Microsoft. (n.d.). PlayReady License Acquisition. Retrieved from https://docs.microsoft.com/en-us/playready/overview/license-acquisition*
5. *Microsoft. (n.d.). PlayReady License And Policies. Retrieved from https://docs.microsoft.com/en-us/playready/overview/license-and-policies*
6. *Microsoft. (n.d.). PlayReady Security Levels. Retrieved from https://docs.microsoft.com/en-us/playready/overview/security-level*

7. *Mulder, Y. D. (2014, February). Analysis of White-Box AES Implementations. Arenberg, Germany. Retrieved from https://www.esat.kuleuven.be/cosic/publications/thesis-235.pdf#page=80&zoom=100,76,542*
8. *SafeNet. (n.d.). Understanding White Box Cryptography. Retrieved from https://www3.thalesgroup.com/email/2012/srm/whitebox/public/pdf/WP_Whitebox_Cryptography__A4__web.pdf*
9. *Schrittwieser, S. (n.d.). Application And Implementation of Code Obfuscation. Retrieved from https://publications.sba-research.org/publications/Code_Obfuscation_CameraReady.pdf*
10. *Stockhammer, T. (n.d.). Retrieved from https://dl.acm.org/doi/pdf/10.1145/1943552.1943572*
11. *Vijayanagar, K. R. (n.d.). CENC common encryption standard. Retrieved from https://ottverse.com/eme-cenc-cdm-aes-keys-drm-digital-rights-management/*
12. *Vijayanagar, K. R. (n.d.). EME diagram. Retrieved from https://ottverse.com/eme-cenc-cdm-aes-keys-drm-digital-rights-management/*
13. *WideVine. (2013). WV Modular DRM Security Integration. Google.*
14. *WideVine. (2017). WideVine DRM Architecture Overview. Google.*
15. *Wikipedia. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP*

Extra notes to add:

Some important packets from netflix:

{"version":"2.1","trackIds":{"nextEpisode":200257858,"episodeSelector":20025 7859},"video":{"title":"The Great Gatsby","synopsis":"Fascinated by the

mysterious and affluent Jay Gatsby, his neighbor Nick Carraway bears witness to the man's obsessive love and spiral into tragedy.","rating":"12","artwork":[{"w":1280,"h":720,"url":"https://occ-0-179-300.1.nflxso.net/dnm/api/v6/7e0PTVDdJ65eumyzagWiJKiw6MU/AAAABUZpCx6WgOGZprX9VvvTuh2sMlkc5LN7ZSOj1qb7qKevl0sUEGS223vqZJMub_VnduLfK4npj41jHX_GTPvGfqqeZwdp.webp?r=823"},{"w":1280,"h":720,"url":"https://occ-0-179-300.1.nflxso.net/dnm/api/v6/7e0PTVDdJ65eumyzagWiJKiw6MU/AAAABUZpCx6WgOGZprX9VvvTuh2sMlkc5LN7ZSOj1qb7qKevl0sUEGS223vqZJMub_VnduLfK4npj41jHX_GTPvGfqqeZwdp.webp?r=823"}],"boxart":[{"w":426,"h":607,"url":"https://occ-0-179-300.1.nflxso.net/dnm/api/v6/Da_vleYcahiCE7JMYt8LJRyoenc/AAAABUFBiy13kpzpdYWHoCAunrsprdo-ssr-Ir6JJBfrSlNMdxh09xDR5YiBXJAviGtlPqcd0jmNmZPmB4SXlrBq9OxohybW.webp?r=ae6"},{"w":284,"h":405,"url":"https://occ-0-179-300.1.nflxso.net/dnm/api/v6/Da_vleYcahiCE7JMYt8LJRyoenc/AAAABWb_0N6_nfM1G6FBChNsVuhI-ooyK69H20zoCC9eDDkpVzWmVMrnRIU4QlfcqHKcGc77t-6PD6x8ru3RbUW9jTQDPH7m.webp?r=ae6"}],"storyart":[{"w":1280,"h":720,"url":"https://occ-0-179-300.1.nflxso.net/dnm/api/v6/E8vDc_W8CLv7-yMQu8KMEC7Rrr8/AAAABa5f5xOvLDhVq15HMmSMBdtPVJIwdeFPY1eq9PfVla_YDQKqUAJ2BwfbAswKAPyNupJJVRE3ZV3tV-DWkPieSdLtqCro.webp?r=ed7"},{"w":1920,"h":1080,"url":"https://occ-0-179-300.1.nflxso.net/dnm/api/v6/E8vDc_W8CLv7-yMQu8KMEC7Rrr8/AAAABW54izDe-S9PkHfkdQUpz0lZAHL6I0YvvP3nJ1WQj8CwnKUOy4LwhpWoMY4J-hcgqLtwKx2OpNg2I8bHxzEHptTdek1n.webp?r=ed7"}],"type":"movie","id":70244437,"userRating":{"matchScore":94,"tooNewForMatchScore":false,"type":"thumb","userRating":0},"skipMarkers":{"credit":{"start":0,"end":0},"recap":{"start":0,"end":0},"content":[]},"start":1626303600000,"end":1657839600000,"year":2013,"requiresAdultVerification":false,"requiresPin":false,"requiresPreReleasePin":false,"creditsOffset":7997,"runtime":8521,"displayRuntime":8521,"autoplayable":true,"bookmark":{"watchedDate":1627834530343,"offset":75},"hd":false,"stills":[{"w":1920,"h":1080,"url":"https://occ-0-179-300.1.nflxso.net/dnm/api/v6/9pS1daC2n6UGc3dUogvWIPMR_OU/AAAABbKV3SY76VcqjbwU8b22v2f97ykf8PZoSwTAW_weUNEkLZyBlC7-

2KaFUc3_ZRVpVQSA1LZgWVVNkTXIbCP4_yHq7PU69oCEz8vZR2zoy6mqQOGoCA.webp?r=997"}],"hiddenEpisodeNumbers":false,"merchedVideoId":null,"cinematch":{"type":"predicted","value":"3.0"}}}