Durability and Damageability of MP4 Files amid Data Corruption

Matthew Barry
Dr. Riccardo Bettati
CSCE 685

Fall 2024

Abstract

Video media permeates our lives: movies and shows for entertainment, tutorials or documentaries for education, security footage, scientific monitoring, video calls for work or keeping in touch, recordings of cherished memories, etc. We capture this data in video files using various cameras and smartphones, but unfortunately, these file formats can be brittle and prone to corruption. Imagine opening an irreplaceable video only to see an error message, "Sorry, this file is damaged and cannot be played." What causes this to happen? How often does this happen? Which video files are at risk? This research answers these questions for the MP4 file format, the most commonly used format in commercial and consumer devices at the time of writing.

Contents

List of Figures

Chapter 1

Introduction

Video media pervades modern life: it captures memories of important events, keeps records for later review, tells engaging stories for entertainment, and spreads news about current events. Much of this video data has great intrinsic or sentimental value and is worth preserving. Yet, some videos succumb to data corruption over time and eventually lose their ability to be viewed. Digital video files are saved in many different formats. At the time of writing, MPEG-4 (MP4 for short) is one of the most common formats, and a preliminary survey shows it is particularly susceptible to this type of corruption and loss. This study investigates the underlying cause of this problem.

1.1 Background

Digital forensics, as the subject implies, brings the rigor of forensic investigation to digital applications. Any forensic science seeks to answer the question, "What happened here?" usually in the wake of some unknown or undesirable event such as a criminal investigation. Digital forensics frequently involves working with data assumed to contain valuable information, but that information may not be known ahead of time and must be uncovered through analysis and interpretation. Low-level sequences of bits stored in a computer may seem like random noise on initial inspection. These bits derive meaning from how they are interpreted, which, in turn, depends on the method used to encode the information they represent. The loss of this critical metadata might result in the effective loss of the data itself, akin to losing the key needed to decrypt a secure message. More fundamentally, even if the proper decoding method is known, the resulting interpretation is potentially meaningless if the data itself is corrupted.

Within the realm of digital forensics, data recovery methods provide a means of detecting and repairing data corruption or reverse engineering its interpretation. However, such *a posteriori* methods come into use only after the damage has been done. The task of hardening data against corruption or aiding in recovery *a priori* belongs to the field of data preservation. Both of these fields complement each other: improvements to data preservation tactics amplify the effectiveness of data recovery methods, and advances in data recovery improve

the perceived reliability of current data formats and storage practices.

Media information collectively refers to audio, photographic, and video information. Encoding such information into files poses a particular challenge due to its complexity. By contrast, numeric and textual information can be encoded efficiently in compact formats that are relatively easy for both humans and computers to interpret. This is due to the symbolic nature of text and numbers, where information is conveyed through discrete finite values and glyphs. Media data carries much more information and is intended to reconstruct the signals that stimulate our senses directly. Consider the nature of this data. Our senses and the mind's interpretation leave much room for the data to vary: two similar-looking videos might have subtle variations in subject placement, colors, brightness, audio, timing, etc. While there is significant room for variation, media formats represent a form of ordered information that is distinct from randomness. Note the virtual impossibility that any such meaningful patterns could emerge from truly random data (e.g., white noise and "snow"). The combination of variation and non-randomness in this information implies a large amount of entropy and, therefore, a large amount of bandwidth required to encode it.

Media data are also highly dimensional, depending on the type. Audio data consists of a sequence of channels oriented along a single time dimension. Photographic media consists of color channels within a two-dimensional pixel array and reconstructs a visual stimulus at a single point in time. Video data is particularly interesting because it is a combination of both, with two-dimensional photographic data drawn out over time and usually accompanied by parallel audio data. If left in their raw formats, such video data would occupy an inordinate amount of storage space; so audio and visual *codecs* employ various compression methods to retain the original meaning or value of the data while reducing the amount of space required to store it. Some codecs are perfectly reversible and therefore *lossless*, whereas others represent an approximation of the original data and discard the residuals—i.e., *lossy* codecs.

Codecs encode a single stream of either audio or visual information, but not both. Therefore, to represent a video as a single file unit, the file format must contain a combination of encoded audio and visual *streams* related by a common timescale. These file formats are called *containers*. At the time of writing, MPEG-4 Part 14 (MP4), Matroska (MKV), and WebM (a derivative of MKV) represent some commonly used video container formats, with MP4 being highly favored by a vast majority of available hardware and software. Each container format supports one or more codecs for storing each contained audio and video stream. In the case of MP4, the most often used audio and video codecs are AAC and H.264, respectively. The Advanced Audio Coding (AAC) codec was first standardized in MPEG-2 Part 7, and the Advanced Video Coding (H.264) codec was introduced in MPEG-4 Part 10. Focusing on these most popular formats limits the scope of research while maximizing relevance to the general public.

1.2 The MP4 Container Format

1.2.1 Specification History

The specifications governing the MP4 file format have their origin in Apple's QuickTime File Format (QTFF), which was originally published in 2001 [1]. This format was standardized and promulgated in ISO/IEC 14496-1 [2] later that year by the Moving Picture Experts Group (MPEG) [3][1]. Since then, the format has been refined and updated to give rise to the following current standards:

- ISO/IEC 14496-12:2022 – ISO base media file format [5],
- ISO/IEC 14496-14:2020 – MP4 file format [6], and
- ISO/IEC 14496-15:2022 – Carriage of network abstraction layer (NAL) unit structured video in the ISO base media file format [7].

14496-12 defines the basic structure used by the MP4 file format, and then 14496-14 extends this basic structure to include additional support specifically for certain audio and video codecs. The contents of 14496-15 are beyond the scope of this study.

It should be noted that these official standards are not freely available: each publication costs 208 CHF (232.96 USD) to purchase at the time of writing. However, the specification can be obtained in other ways. The MPEG website archive [3] provides older, withdrawn versions of each publication, while Apple maintains the current version of the QTFF specification on its website with the following notice:

> The QTFF has been used as the basis of the MPEG-4 standard and the JPEG-2000 standard, developed by the International Organization for Standardization (ISO). Although these file types have similar structures and contain many functionally identical elements, they are distinct file types. [8]

Despite the claim that the file types are distinct, the two file format specifications are similar enough from a technical perspective to develop a tool capable of reading files in either format. The structure of the MP4 file format can also be learned by inspecting the source code of open-source software that is capable of opening MP4 files, such as VLC [2], MPV [3], and ffmpeg [4], to name a few.

1.2.2 Basic Internal Structure

Internally, an MP4 file contains a composite hierarchy of data units arranged as a tree. Apple's QTFF Specification [8] refers to these as "atoms," whereas the ISO/IEC MPEG-4

[1] The MPEG group has since been closed in favor of the *Moving Picture, Audio, and Data Coding by Artificial Intelligence (MPAI)* community [4].

[2] `https://www.videolan.org/vlc/`

[3] `https://mpv.io/`

[4] `https://ffmpeg.org/`

specification refers to these as "boxes." The two definitions are practically identical, and the terms may be used interchangeably in this document. However, the latter will be preferred out of respect for the standard, even though the information presented here relies heavily on Apple's freely available specification instead of the paywall-restricted ISO/IEC specification.

Each box consists of a header and contents. The header includes a 4-byte discriminant value that helps to determine the type of data that the box contains. This discriminant, in combination with the box's position within the tree (i.e., "ancestor" boxes and their types), determines the exact field structure of a box. Apple gives the following example in their specification: "the *profile* atom inside a *movie* atom contains information about the movie, while the *profile* atom inside a *track* atom contains information about the track" (emphasis added to denote types). Given the above general description, all boxes contain at least the following two, maybe three, basic header fields in the provided order:

1. `size`: *unsigned 32-bit integer*, the number of bytes that the whole box occupies, including these fields. There are two special cases for this value:
   (a) The root box (and *only the root*) may contain a size of 0 if its contents continue until the end of the file.
   (b) A box whose size is over 2^{32} bytes in length will have a size of 1 and provide the actual size in an additional `extended size` field below.
2. `type`: *4-character string*, the box's type discriminant.
3. `extended size`: *unsigned 64-bit integer (optional)*, this field is present only if it is larger than what can fit in the `size` field. Otherwise, it is either omitted or filled with a placeholder `wide` box.[5]

Of the many types of boxes defined by the specification, the following types are of interest: `moov` and `mdat`. A movie file must contain a `moov` box because it contains metadata about the movie and how to interpret it. Such metadata includes the number of tracks in the file, the type (audio, video, subtitles, etc.), codec used, and time synchronization information for each track, references for converting time indices to data locations, etc. The data referenced by the `moov` is stored in `mdat` boxes elsewhere in the file and not in the `moov` box itself. `mdat` boxes are relatively simple: each contains a header with the `size`, a `type` of `mdat`, and optionally the `extended size`, followed directly by binary media data. The structure of a `moov` box is much more complicated, but it is sufficient to note that its presence and integrity are vital to the file's playability.

Atoms deeper within the `moov` hierarchy are more tightly coupled with the individual pieces of the data that comprise the media information. For example, The `dinf` atom indicates where the data resides, i.e. the current file itself or in an external file, and the

[5] A `wide` box is an "empty" box containing only a header with a `size` of 8 bytes and a type of `"wide"`. Filling the extended size with a `wide` box has the effect of allowing the box to grow larger than 2^{32} bytes without needing to rewrite or reallocate it.

`stbl` atom provides a mapping from time and sample number to the offset/address where the sample's data can be found. According to the QTTF specification, "a sample is a single element in a sequence of time-ordered data," the quantum of data represented by an MP4 container.

1.3 Problem Statement

The MP4 video file format is susceptible to data corruption. The degree to which this corruption affects the encoded video varies greatly. In some cases, minor corruption may have little to no effect. Noticeable effects may consist of a benign reduction in media quality or alter the video in some way. These types of errors leave the file in a still-playable state and sometimes can be recovered to their original quality using existing video recovery methods. However, in some unfortunate cases, even small amounts of data corruption in the right (or wrong) place can render an entire MP4 video file unplayable even though the vast remainder of the file remains intact.

The goal of this report is to investigate and document how data corruption can lead to unplayable MP4 video files. This process may happen naturally over time without any human intervention, unintentionally through user error, or purposefully for some intended (possibly malicious) reason. Nonetheless, we hypothesize that a simple, common underlying mechanism exists by which MP4 videos are rendered unplayable. Namely, the destruction of critical data within the MP4 file.

Chapter 2

Literature Review

Video formats, particularly the MP4 container format, have been the subject of much research in digital forensics due to the wealth of information they contain. Their widespread publication and availability on social Internet forums and the massive variety of recording devices, especially smartphones, ease the spread of compelling narratives. For this reason, videos are also of great value to any type of forensic investigation. Feeds from security cameras, dashcams, and even mounted GoPro-style cameras capture objective evidence of how historical events unfolded from the perspective of a first-person observer.

2.1 MP4 Metadata Structure and Interpretation

From an outside perspective, MP4 video files contain a tremendous amount of data: a single video that is only a few minutes in duration can have a file size approaching hundreds of megabytes, depending on the quality. Most of this, however, is occupied by the stream data. The metadata, in contrast, is only a few kilobytes in total and can be scattered anywhere throughout the file.

The MP4 file structure is hierarchical, and its prohibitive size prevents the entire tree from being loaded into memory directly. Traversing the tree in its entirety might require multiple passes through the file, which can severely impact read performance. An optimal approach for parsing MP4 metadata is to use an event-based parser that reads the file in a single pass, noting the file offsets where various metadata of interest occur [9]. This approach uses a stack structure to record its traversal state since it encounters nested nodes in a depth-first fashion. Similar methods exist for parsing other types of hierarchical data efficiently: namely, SAX for parsing XML [10].

2.2 Data Corruption

Digital forensics often requires sifting through incomplete, fragmented, or otherwise corrupted data. Such damage to a file's data stems from various causes and can occur when

data is "in transit" between storage devices and when it is "at rest" on a single isolated storage device.

2.2.1 Corruption in Transit

Although often overlooked as such, copying data between locations within a single device is a form of data transfer. As with the scribes and historians of old, transcription errors can arise when copying data from one place to another. In a digital device, failures in the electrical hardware are likely causes of such errors. More often, however, data transfers occur between devices over a network.

The physical layer is the lowest level of Internet Protocol (IP) network infrastructure and has implementations using various wired (e.g., coaxial, Ethernet) and wireless (e.g., 802.11) technologies. At this level, network transmissions are prone to interference and noise from external sources and network hardware failures from within. These can introduce errors in the transmitted signal, thereby reducing the connection reliability. Transmission Control Protocol (TCP) offers a robust means of data transfer over these potentially unreliable connections at the expense of communication and computational overhead. On the other hand, User Datagram Protocol (UDP) provides a simple and fast way to relay data between two networked devices without the overhead. Still, these communications are susceptible to data corruption in transit.

Dropped Packets

When transmitting large amounts of data, the sending software splits the original data into multiple smaller segments or packets, which the receiver then reassembles. Some of these packets may be delayed or lost, and the data they contain will be out of order or missing entirely in the reassembly. TCP contains packet sequence numbers to enforce correct ordering and requires acknowledgment of each transmitted packet [11]. Failure to receive an acknowledgment will result in packet retransmission, and unexpected errors in sequence numbering will trigger a connection reset. In the end, the receiver can know with confidence that all packets will received and reassembled in the proper order, but this confidence comes at the cost of doubling the number of exchanges. Every transmitted data packet must have an accompanying acknowledgment returned to the original sender. This overhead prevents devices from taking advantage of the total available network bandwidth.

For live media streams, maximizing the amount of information bandwidth maximizes the perceived quality of the media, so applications generally prefer UDP for this purpose. The downside is that UDP does not detect or correct either issue, so the application layer must handle problems. Padmanaban and Ilow [12] explored ways to handle dropped network packets from H.264 video streams. Their method relies on the fact that H.264 already packetizes its stream data. In addition to sending the video packets, this approach sends interspersed parity packets from which the receiver can reconstruct dropped video packets if necessary. This manner of reconstruction requires support at both ends of the communication: the sender must encode the parity packets, and the receiver/player must know how

to take advantage of them.

Random Transmission Errors (Bit Flips)

One of the mechanisms used by TCP to ensure reliability is a computed checksum for each packet [11]. This checksum determines whether any transmission errors occurred and, in some cases, enables the receiver to correct them. In the former case, the recipient could withhold acknowledgment for a packet containing a checksum mismatch and wait for the sender to reattempt transmission. However, computing and verifying checksums require even more overhead and further reduce usable bandwidth. This overhead is yet another reason why UDP is favored for live video streams. Padmanaban's method above provides a way to use parity data to identify and correct bit flips as part of a live stream.

Premature Stream Termination (Truncation)

All forms of network transmissions are susceptible to sudden termination. This connectivity issue can arise from several causes, such as the loss of an intermediate network device, removal or severing of transmission lines, termination of the software performing the sending or receiving, etc. In any case, the receiver has only a partial copy of the intended data. The worst-case scenario for this type of error is when it occurs silently, leading the user to believe the transmission is complete when, in fact, it has failed. In the case of media files, this can occur when recording data from a live stream (e.g., security footage or a digital broadcast) that is interrupted or otherwise ends abruptly.

This type of corruption is not limited to transit over a network. For example, when copying a large file to a removable storage device, the operating system typically copies the file into an in-memory buffer. Doing so helps to keep the operating system responsive to other tasks while data is flushed from this buffer to the storage device asynchronously. As part of the device unmounting or "safe removal" process, the operating system ensures that any such cache buffers are flushed completely. However, if the storage device is removed without being safely unmounted, the operating system may not have finished writing transferred files. Files that occupy a large amount of storage (a characteristic of most media files) are particularly prone to unsafe removal. Partially written copied files can be especially troublesome if the target destination is a backup that may need to be used to recover the files later. In the event of such a recovery, a user would be troubled to find that the supposedly backed-up files are, in reality, not recoverable.

2.2.2 Corruption at Rest

Data in transit is particularly vulnerable due to the number of problems that can arise when transcribing the data. Once data is written to a (non-volatile) storage medium, we might consider the data safe from any corruption. On the contrary, storage devices themselves are prone to data degradation.

How storage degrades over time is dependent on the storage technology in use. Here are some common storage methods and how they can fail with age:

- *Mechanical* storage devices, such as hard drive disks (HDDs), depend on finely tuned moving parts that can wear out, drift, or break due to physical shock, whereas *solid-state* storage devices are more durable in this regard.
- *Magnetic* storage, such as tapes, HDDs, and floppy disks, can weaken over time when exposed to heat or external magnetic fields (such as that of the earth) or other electromagnetic sources.
- *Electrical* storage, such as flash memory, often relies on keeping a capacitor at a specific voltage, but electrons can drain slowly as the dielectric material wears out with use and age. Flash memory is particularly vulnerable due to destructive write operations.
- *Optical* storage, such as compact disks (CDs), digital versatile disks (DVDs), and Blu-ray disks, are susceptible to warping and scratches. However, with the proper materials and care, this storage category is generally considered the most stable for single-volume, long-term digital storage. The mechanical drive or reader may wear out eventually, but the data storage medium tends to be robust. [1]

Bit Rot

Regardless of the storage technology in use, corruption at rest arises from the physical properties of the storage medium. Degradation over time eventually leads to errors during data retrieval: what was once a 0 might now be interpreted as a 1 or vice versa. The storage device is still operational, but its data might need to be refreshed to "full potency." The term *bit rot* describes this general type of operational degradation. [13]

Many modern storage technologies use error correction to improve storage density and lifetime. As bit rot begins to occur, read performance will degrade as the device must correct any misread bits. However, older and simpler storage devices may not have any forms of error detection and correction: the bits read from the device are sent verbatim to the operating system.

Sector Read Errors

Storage devices, filesystems, and operating systems all tend to work with data in equal-sized chunks called *sectors*. This term originates in rotating storage media: as the medium rotates, the read/write head sweeps out an arc or sector along the surface as it reads the requested data. If a sector contains too many bit errors for the on-device error correction to handle,

[1] Optical storage is a class of other physical storage methods that can be "etched" into a physical medium. Such strategies tend to be "write-once, read-only." By contrast, other energy-based storage methods rely on preserving some type of energy gradient. Such states have a natural tendency to return to an equilibrium. These strategies can be erased and rewritten.

the device will report a read failure, leaving a gap in the retrieved data. Any damage or imperfections in the storage medium can cause similar failures when reading the affected sectors. In the case of SCSI- and SATA-connected (magnetic) hard disk drives, vendors specify an error rate of one error per 10^{13} to 10^{16} bits read. [14] These read errors can also manifest as bit rot.

Fragmentation

While not a form of corruption *per se*, the block-based nature of filesystems can lead to fragmentation. This occurs when blocks containing a file's data cannot be stored contiguously in the storage medium. Fragmentation becomes problematic for interpreting video data when the filesystem metadata governing the location and ordering of blocks is damaged or no longer present. Such situations arise when recovering deleted files or quick-formatted storage volumes.

2.3 Data Integrity and Preservation

2.3.1 Transmission Validation

Savvy users know the potential for transmission errors or foul play when downloading large or important files from the Internet. Hence, a commonly recommended practice is to validate a file's integrity against a published checksum, hash, or cryptographic signature provided by the original file's author. A validation failure indicates a problem with the downloaded file, in which case the download should be reattempted, perhaps from an alternate mirror if multiple hosting servers are available.

2.3.2 Multiple Volume Storage: RAID

Fortunately, bit rot and sector read errors are relatively easy to prevent: RAID arrays (levels 2–6) maintain parity information that allows information to be recovered in the event of read errors or hardware failures [15]. All RAID systems have a limited redundancy threshold representing the number of disk failures that can be tolerated. RAID's error correction detects problems only when reading the data, and, as previously mentioned, errors can aggregate silently over time. If left unchecked, the data could still degrade to the point where it is no longer recoverable. Modern software RAID systems such as ZFS and BTRFS address this shortcoming by proactively and periodically "scrubbing" the data to maintain its integrity [16] [17] [18]. A scrub operation reads all stored data to check for and repair any errors, which prevents silent errors from accumulating.

2.4 Data Recovery

2.4.1 Filesystem-Based Recovery: File Carving

Regardless of the type or source of corruption, recovering usable data from a corrupted source requires considerable effort and is not always successful. Methods such as file carving [19] [20] attempt to recover a file's original content from remnants of filesystem metadata. File carving recovers deleted files from the "free space" on a storage medium. When deleting a file, the operating system only marks the file's allocated blocks as unused, but the blocks themselves (and file data therein) remain intact until they are overwritten. Large files that span multiple blocks become fragmented when the blocks used to store a file are not contiguous within the storage medium. This is often the case for video files due to their size.

Early file carving tools like Scalpel [21] did not compensate for filesystem fragmentation. Poisel and Tjoa recognized this problem and outlined a "roadmap" to develop a file carving technique for fragmented multimedia files [22]. Casey and Zoun explored tradeoffs for methods that produce longer playable fragments versus many shorter fragments and found the total duration of recovered video varies by case and method [23]. Na *et al.* [24] propose using H.264 frame headers to identify video stream data across multiple blocks and then arrange those blocks into a valid video file. Other recent tools like VidCarve [25] similarly account for fragmentation and, in this case, have been tailored specifically for recovering video files.

File carvers for video data maintain a metadata index and attempt to match available movie data. Carvers generally discard segments of movie data without a matching header because they are unplayable. , data becomes spliced into another video file because of a false positive match in the metadata index. Generally, file carving assumes the presence of all video data and metadata needed to construct a playable video, but it often struggles to assemble video file fragments in the proper order. When only partial video data or metadata is available, file carving produces unexpected results.

2.4.2 Stream-Based Recovery Methods

Noise reduction

Theoretically, the media data contained in a stream—assuming it can be properly decoded from its container—represents an ordered signal, and any corruption could be classified as introducing some random interference. Stated differently, the signal represents some function we wish to recover, and the data corruption is an undesired alteration to the output of the signal function. From this perspective, data recovery takes the form of noise reduction within the realm of signal processing. Fourier transforms and wavelet decompositions are well-known tools in this area and have been used in the recovery of degraded audio [26] [27] and image [28] [29] data.

Frame Reconstruction

Zhang and Stevenson [30] proposed an alternate video encoding method consisting of parallel streams, allowing dropped frames in one stream to be reconstructed from nearby frames in an alternate stream. Chen *et al.* [31] explored using steganography to embed recovery data directly within an H.264 stream.

Corrupted video data can originate from online sources as well. Transmission errors from live-streamed video sources require live error detection, correction, and, if that fails, recovery.

More recently, Mary P. D. *et al.* [32] and Hoang *et al.* [33] have developed tensor completion techniques for filling in missing pixels in degraded video frames. However, as mentioned previously, all of these methods operate at the stream level, and their success depends on intact and well-formed MP4 container metadata.

2.5 Available MP4 Video Datasets

Datasets like the one developed by Gloe [34] as well as VISION [35] and FloreView [36] aggregate hundreds—even thousands—of media files from dozens of devices for comparative metadata analysis and research. The latter dataset even uses the same subjects across all media captures. The EVA-7K [37] dataset includes videos modified by various video editing software and shared online through common social platforms. These datasets are intended for comparative analysis to identify a video's origin and any possible manipulation.

2.6 Honorable Mention

A survey of the current state of digital forensics, especially that which is pertinent to media data, would be incomplete without including some significant areas of interest. These topics may not be readily or obviously applicable to the research presented here, but they are nonetheless vital to understanding current problems. Perhaps someone else will find the material presented here helpful.

2.6.1 Manipulation and Deepfake Detection

In the past, video evidence was regarded as indisputable. The underlying assumption was that video would be extremely difficult or impossible to fabricate. Video manipulation methods were primitive and left noticeable artifacts in the resulting videos. Today, video editing software is more widely available and has improved, making it more challenging to identify edits visually. Moreover, with the recent introduction of "deepfakes" created by artificial intelligence, verifying a video's authenticity is crucial before trusting its content. Manipulation detection is a valuable defense against the effects of disinformation campaigns in today's Internet-connected culture.

Pixel-based Methods

Early work in pixel-based manipulation detection focused primarily on detecting visual artifacts or comparing similarities with a known original video [38]. However, as editing software has improved, these artifacts are becoming more challenging to identify. In some cases, distinguishing between an original video and a modified version might not be possible from visual indicators alone.

The ubiquity of artificial intelligence for handling visual information and identifying subtle patterns has led to the irony of "fighting fire with fire:" the same tactics used to make some oTxf the most compelling manipulated or fabricated videos are being used to identify them. Rana *et al.* identified 112 studies between 2018 and 2020 for deepfake detection alone [39], 86 of which used a form of deep learning or machine learning. Facial recognition accounts for most of these strategies since generated videos are not likely to reconstruct an individual subject's facial features perfectly. The success of these pixel-based approaches depends on the machine learning model used. As evidenced by the large number of studies, researchers have developed an ever-widening selection of models from which to choose.

Metadata-based Methods

The metadata embedded within video files can provide an audit trail and clues about a video file's origin. This information can be direct, such as an embedded "Edited by such-and-such software" comment or marking, but an adversary with intent to deceive would likely strip out such apparent indicators. Hence, manipulation detection methods that rely on more subtle markings are invaluable for forensic investigation. MP4's metadata structure allows for ample variation while remaining true to the specification, and different recording devices or editing software tend to leave their own "fingerprint" in the metadata of the files they produce.

Xiang *et al.* developed methods for analyzing this metadata using nearest-neighbor classification [40] and self-supervised neural networks [41]. Metadata-based manipulation detection may provide a more reliable means of manipulation and deepfake detection over pixel-based methods as video manipulation tactics continue to improve.

2.6.2 Steganography

MP4 videos can also provide an enormous cover for surreptitiously embedding additional data. Hemalatha *et al.* developed a way to embed secret audio and image data in the wavelet domain of each of the cover video's streams [42]. Their method offers increased payload capacity and a better signal-to-noise ratio than prior video steganography methods.

Steganography greatly interests the broader digital forensics field but may seem less applicable to video recovery. On the contrary, there is some relationship between these pursuits. Steganography aims to embed data without any recognizable effect on the cover data. However, any changes to the cover data invariably result in artifacts or signal degradation to some degree since a certain amount of its former bandwidth is hijacked for use by the

embedded signal. In the case of video cover data, the video recovery methods mentioned above could reverse the embedding process, returning the original video to a closer approximation of the original data. If the embedded data is also a video stream, the above video recovery methods could reduce the bandwidth needed to perform the embedding. Moreover, as previously discussed in [31], steganography can embed arbitrary recovery information in the data itself.

2.7 Summary

The existing literature demonstrates a need for further research into how intact MP4 video files might become unplayable over time. All of the identified prior research assumes that files remain consistently playable, but experience demonstrates that an MP4 file's viability is not guaranteed. Existing methods such as file carving offer some insight into how potentially fragmented video files might be reassembled, but this process itself often leads to unplayable videos. Moreover, file carving attempts to recover the file contents exactly, so a file that was already unplayable prior to carving will be unplayable even after a successful recovery. Other recovery methods operate at the codec level, repairing errors in the media stream rather than in the container itself. Fortunately, the general ways in which data can become corrupted are well-known and succinctly enumerated, but do not seem to have been applied to MP4 files.

Chapter 3

Methods

3.1 Dataset

Despite a rigorous data format specification, the set of all valid MP4 files is infinite. Hence, examining all files exhaustively is impossible. By limiting the file size or media duration, we can reduce the space of valid MP4 files to a finite but still very large set. Even limiting the scope of study to one or a handful of recording devices still produces a set that cannot easily be searched. For example, imagine trying to enumerate all possible 10-second clips that can be recorded on a single device.

We need a representative sample for experimentation, and the videos from the peer-reviewed FloreView dataset [36] satisfy this requirement. Using this dataset avoids potential bias that could result from a self-sourced dataset. It also provides an experimental standard or baseline for repeatability if anyone should choose to verify the results of this study by replicating it. FloreView provides a representative set of videos from smartphones commonly used at the time of its production, and these devices are still relevant at the time of this study.

The FloreView dataset comprises videos from 46 smartphones, numbered `D01` through `D46`. These models represent 11 unique manufacturers covering a vast majority of the global smartphone market share:

- **Apple:** iPad Air, iPhone SE, iPhone 8 Plus, iPhone X, iPhone 12, iPhone 13 mini
- **DOOGEE:** S96 Pro
- **Google:** Pixel 3a, Pixel 5
- **Huawei:** Mate 10 Lite, Mate 10 Pro, Nova 5T, P8 Lite, P9 Lite, P30 Lite
- **Lenovo:** Tab E7
- **LG:** G4c, G7 ThinQ, V50 ThinQ
- **Motorola:** Moto G, Moto G5, Moto G5S Plus, Moto G9 Plus

- **OnePlus:** 6T, 8T
- **Samsung:** (Galaxy) A12, A40, A52s, Note 8, S6, S10, S10+, S20+, S21+
- **Sony:** Xperia M2
- **Xiaomi:** Mi A2 Lite, Mi Mix 3, Redmi 5 Plus, Redmi Note 8, Redmi Note 8T, Redmi Note 9

Observe that there are only 41 unique models despite 46 devices used to construct the dataset. This is due to the repetition of the following models:

- iPhone X: `D02` and `D22`
- Xiaomi Redmi Note 8T: `D04` and `D10`
- Motorola Moto G: `D06` and `D28`
- Motorola Moto G5: `D15` and `D39`
- Google Pixel 3a: `D19` and `D23`

The dataset contains 2029 video files, all of which use the MP4 container format. Apple devices use the `.mov` filename extension, the Lenovo Tab E7 (`D08`) uses `.3gp`, and the remainder all use the typical `.mp4` extension. These videos all contain the same set of subjects for each device. Most videos use only the JPEG/H.264 encoding, while some use HEIC/H.265 (often in addition to JPEG/H.264). Again, despite differences in naming and media encoding, all share the same underlying MP4 container format, which is the focus of this study.

These files vary in size from 3.44 MB to 195.94 MB, with a median size of 54.91 MB. The full distribution of sizes is depicted in Figure 3.1.

3.2 Data Corruption Model

The previous chapter highlighted several ways in which a file might become corrupted at rest or in transit. Although these are not exhaustive, they represent a significant set of common data corruption instances that a file might encounter with regular use over time. We can generalize these and similar instances of data corruption using the following model:

1. *Bit-flipping*, which accounts for minor network transcription errors or bit rot. This is assumed to occur with uniform randomness across all data contained in a file. The severity is parameterized as the number of bits k_B that are flipped from their original values.

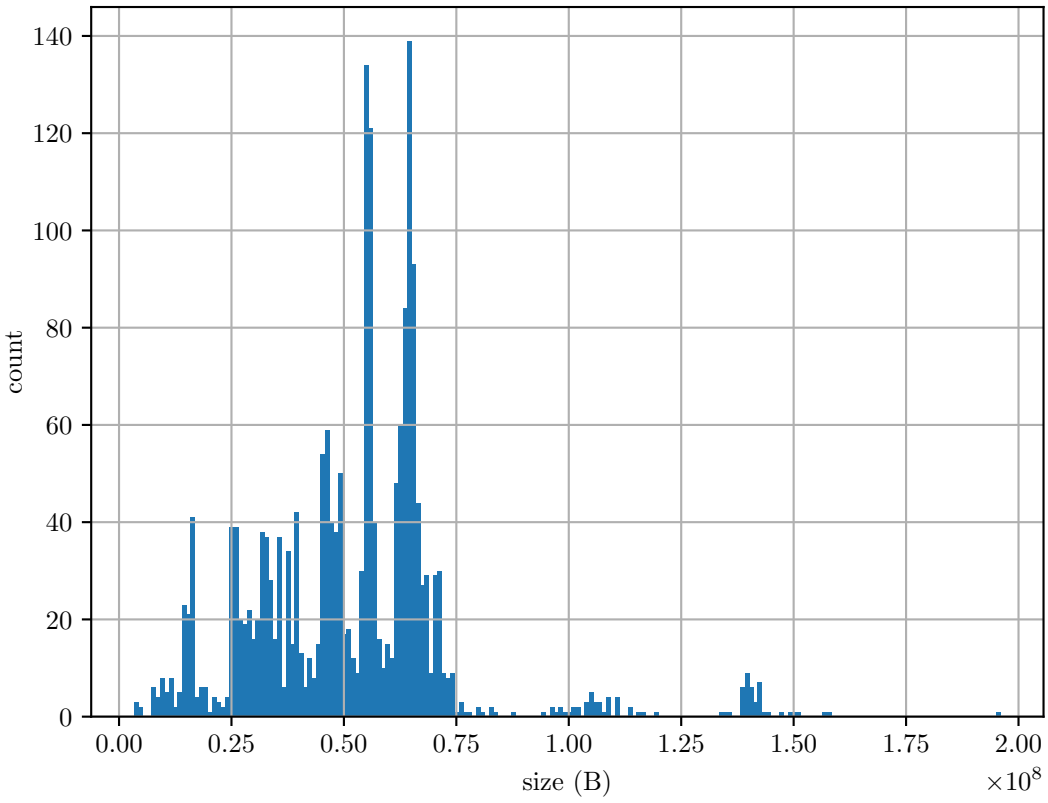

Figure 3.1: Distribution of FloreView file sizes using 200 bins

2. *Segment/sector blanking*, which represents data loss from dropped packets or sector read errors. For this type of corruption, we model file data as being split across an ordered sequence of contiguous segments, all of which are of size b bytes except for the last segment. Given a file of size n bytes, it is unlikely to be the case that b divides n, so the last segment will be partially occupied with only $n - b\left\lfloor \frac{n}{b} \right\rfloor$ bytes of data. Similar to bit-flipping, this type of corruption is assumed to occur with uniform randomness across all segments in a file, and the severity is parameterized by the number of segments k_S with missing data.

3. *Truncation*, which primarily occurs from an interrupted file transfer but could also result from partial recovery due to fragmentation or file carving. The severity of this type of corruption is parameterized as the number of bytes k_T missing from the end of the file.

3.3 Experimental Design

We can induce the effects of our model in a controlled manner by intentionally modifying the binary data comprising sample MP4 video files. To accomplish this, we implement the model as a "mangler" utility program that subjects an original input MP4 file to one of the parameterized binary modifications.

Let D be a set representing preselected corruption severity levels defined later. The following implementations and experimental conditions outline how we can determine the effects of these binary modifications on the input MP4 file's playability:

1. *Bit-flipping:* Seeking to a random byte offset within the file and flipping a random bit within that byte. This process will be repeated for the various severity levels in D.

2. *Segment blanking:* We use a segment size of $b = 4$ KB, which is a typical sector size for filesystems on hard drives. For a random byte offset o within the file, seek to the starting byte offset $\left\lfloor \frac{o}{b} \right\rfloor$ of the containing segment and set b bytes to all 0's. Let n be the file size in bytes. If the selected segment is the last in the file, only $\min(b, n - o)$ bytes will be set to 0's so that the file's size will not change as a result of this induced corruption. This process will be repeated for the various severity levels in D.

3. *Truncation:* Clipping the end of a file by reducing the file's size by the various severity levels in D.

The "mangler" utility implemented for modifying (and inspecting) MP4 video files as outlined above was developed in Rust and exists as a public GitHub repository[1]

The severity of corruption experienced by an MP4 file depends on the size of the file. In particular, flipping 100 bits in a 3 MB file is much more "damaging" than flipping 100 bits in

[1]Repository URL: `https://github.com/komputerwiz/mp4-mangler`. The version of the code used for this experiment was commit hash `a22e622c27e07aa282f47fe0c8a7233c4f46a95f`

a 200 MB file. Therefore, the severity levels in D are defined as the following geometrically distributed *percentages* of the file's size from 0.00000001% to 10%:

$$D = \left\{10^{-8}, 10^{-7}, \ldots, 10^{0}, 10^{1}\right\} \times 100\%$$

We run trials against all video files in the dataset grouped by the binary modification performed (bit-flipping, segment blanking, truncation) and severity level in D. This will produce some proportion of unplayable files in each group. We predict this proportion will increase with higher severity according to a sigmoid regression curve representing the likelihood that a corrupted file will be unplayable. Conversely, the likelihood that a file "survives" data corruption decreases from near certainty at little or no corruption to impossibility as the entire file becomes corrupted. The error margins on this curve will be given by the binomial proportion confidence interval formula (Equation 3.1) with a sample size of $n = 2029$ and double-sided Type I error tolerance of $\alpha = 0.05$ (95% CI). That is, we choose an interval such that there is only a 5% probability that the actual proportion lies outside of the experimentally measured proportion.

$$p \approx \hat{p} \pm \frac{z_\alpha}{\sqrt{n}} \sqrt{\hat{p}\,(1 - \hat{p})} \tag{3.1}$$

To accelerate experimentation, trials will be executed simultaneously across multiple machines using GNU `parallel` [43].

Chapter 4

Results

4.1 Preliminary Investigation

Before running any full-scale experimentation, we performed a small-scale exploratory experiment to investigate what happens to MP4 files when exposed to varying types of data corruption. We used the "L1S1C4" video from the following devices: "D14," "D17," "D34," and "D43." Rather than using a percentage of the file size, these small-scale experiments used raw counts for each severity level

Table 4.1 shows the playability results for each of the selected files. Cases that retained full playability are indicated by the term "full," cases that were partially playable indicate what parts of the file were affected (i.e., "no audio" or "no video"), and videos that could not be opened for playback are indicated as "unplayable." These latter videos are of particular interest for this experiment.

Induced Corruption	**D14**	**D17**	**D34**	**D43**
Bit-flipping (1×)	full	full	full	full
Bit-flipping (10×)	full	full	full	full
Bit-flipping (100×)	full	full	full	full
Blanking ($1 \times 4\,\text{KiB}$)	full	full	full	full
Blanking ($2 \times 4\,\text{KiB}$)	full	full	full	full
Blanking ($3 \times 4\,\text{KiB}$)	full	no video	full	full
Truncation (2 KB)	full	no audio	no video	full
Truncation (200 KB)	unplayable	unplayable	unplayable	full
Truncation (500 KB)	unplayable	unplayable	unplayable	full

Table 4.1: Effects of induced data corruption methods on file playability

4.1.1 Bit-flipping

All videos retained full playability when subjected to modest amounts of bit-flipping. The only side effects of this induced corruption were increased artifacts and stream distortions. The severity of these distortions increased with the number of flipped bits. Bit flips are rare occurrences: 100 flipped bits in a single file are unlikely to occur spontaneously, and even if this were to happen, it would not likely go unnoticed.

4.1.2 Segment Blanking

All but a single video retained full playability when subjected to blanking. Removing three 4 KiB data segments rendered D17 partially unplayable: playback of the file's video stream failed. Again, sector read errors are somewhat rare, but they tend to come in "bunches" as hardware begins to fail or many packets are dropped.

4.1.3 Truncation

The results of this induced corruption are more interesting. D43 retained full playability across all tested severities. D14 remained fully playable at the lowest severity, but higher severities rendered it unplayable. D17 and D34 both retained only partial playability at the lowest severity, and higher severities rendered both videos unplayable. Interestingly, D43 was the only video to remain playable across all severities.

4.1.4 Summary

The results of this preliminary investigation show that MP4 files are relatively resilient against bit-flipping and segment blanking, whereas truncation seems to have the most profound impact. After deeper investigation, the reason for this becomes clear: Some MP4 files place their metadata (i.e., the `moov` box/atom) at the end of the file, where it is vulnerable to being affected by truncation. This metadata is vital for playability and comprises a very small portion of the overall file size. For the videos in the FloreView dataset, the `moov` box/atom accounts for a median of only 0.03% of the file size. A more detailed distribution of this ratio is depicted in Figure 4.1.

The positioning of this metadata, combined with its small size give us the following additional predictions of what the full-scale experimental results should yield:

1. Uniform random corruption, as implemented in bit-flipping and segment blanking, is less likely to affect playability even for larger corruption levels in D.

2. Files that place their metadata at the end of the file will be rendered unplayable even at small corruption levels in D. Videos in the FloreView dataset from Apple, Samsung, and Google have this structure. One device-specific exception is the Xiaomi Redmi Note 9.

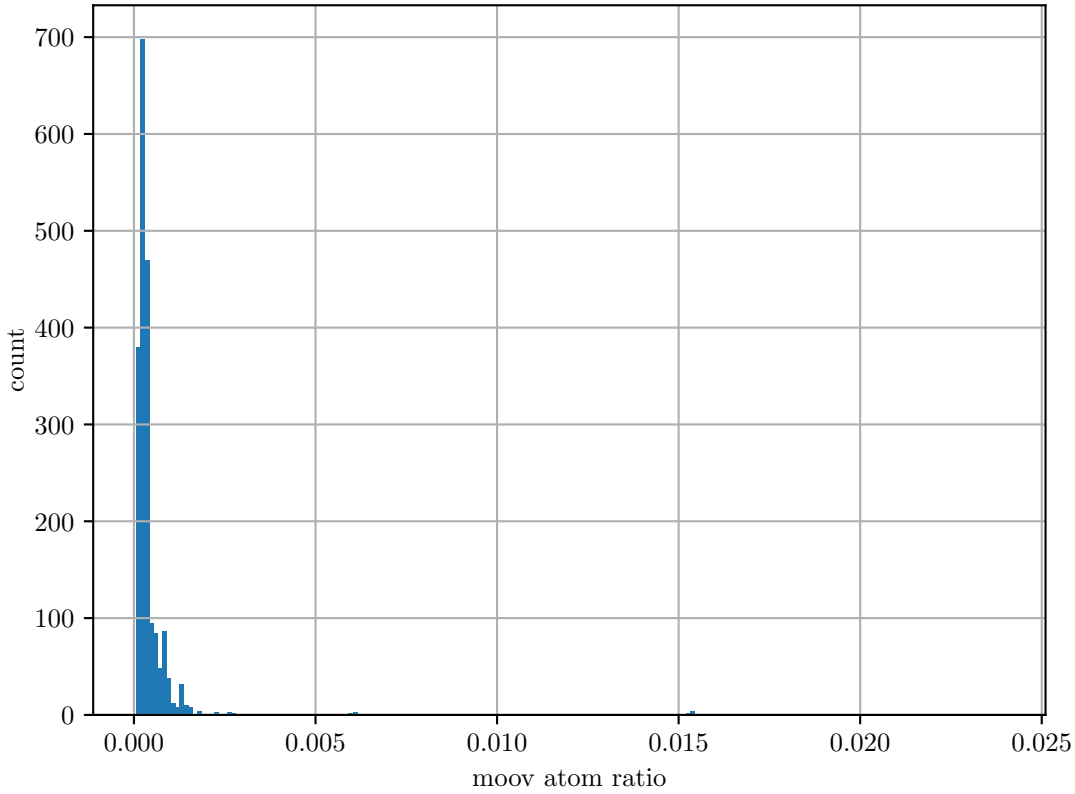

Figure 4.1: Histogram of the ratio between metadata size and file size using 200 bins

3. By contrast, Files with their metadata at the beginning of the file are not vulnerable to truncation and will remain playable even for high levels of corruption. Videos produced by all remaining device manufacturers, including all other Xiaomi models, have this structure.

4. Given 2 and 3, we expect to see the truncation "sigmoid" graph to rise sharply for low corruption levels and remain steady at 0.47, which is approximately the proportion of files with their metadata at the end.

4.2 Full-Scale Experimental Results

4.2.1 Bit-flipping

Figure 4.2 shows how the proportion of unplayable files (interpreted as the likelihood of unplayability) started low for low corruption levels and increased with higher levels, as predicted. However, the highest level of corruption saw a significant deviation from the expected upward trend.

It was later realized that the MP4 mangler would select bits randomly *with replacement*. So, for these experiments, bits could be flipped and later corrected. Another possibility is that the method used to evaluate file playability could have interpreted slow response times as a positive indication that the file was playable: later trials saw a significant decrease in computer performance as filesystem I/O caching hit a memory limit.

The likelihood of unplayability also trended much higher than originally anticipated: At the 1% corruption level, over 85% of files were rendered unplayable. This could result from box size bits being flipped: If a single box size value is affected, the offsets for all subsequent boxes are lost.

4.2.2 Segment Blanking

Figure 4.3 shows the likelihood of unplayability due to segment blanking. This graph behaved more or less exactly as predicted: the fitted sigmoid graph even lines up almost perfectly within the error bars.

4.2.3 Truncation

Figure 4.4 shows the likelihood of unplayability due to truncation. The graph leveled off just below 0.45, as was predicted, but the dip in likelihood centered around the 0.00001% corruption level is very unexpected. One would expect that trimming more data would cause more damage rather than allow more playability. This could result from having multiple tracks within a video file: There may be a point in the metadata between tracks where truncation at that position can leave a file playable.

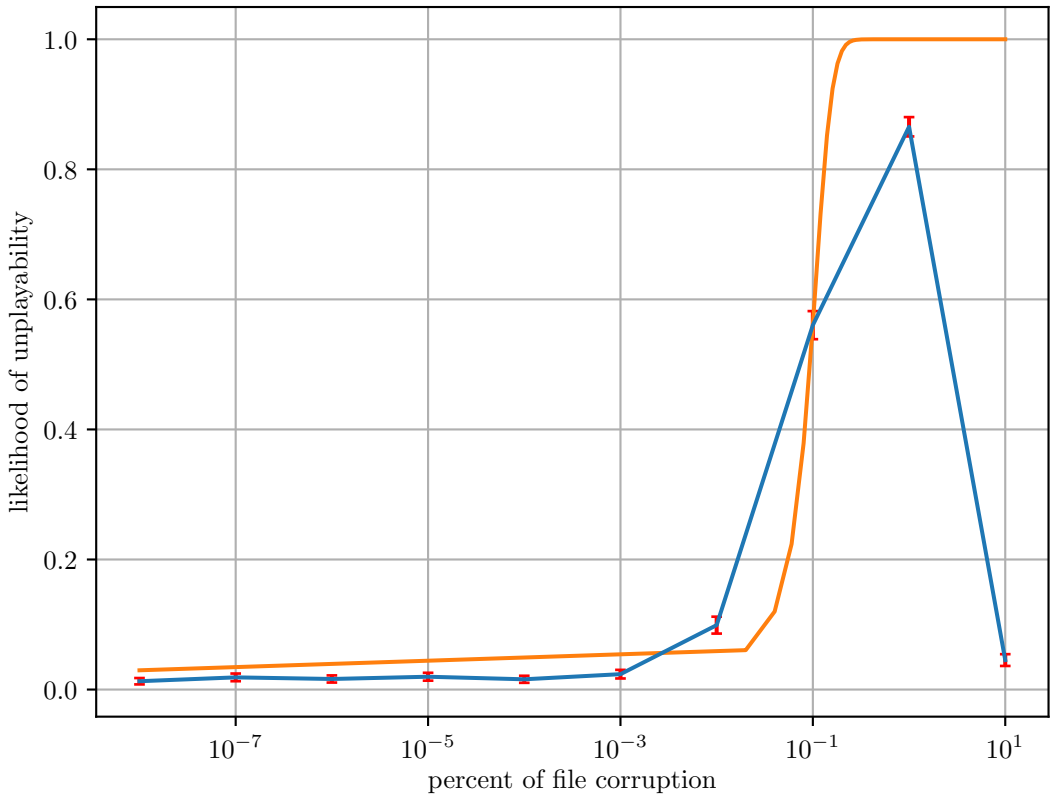

Figure 4.2: Measured proportion of unplayable files for bit-flipping with fitted sigmoid curve

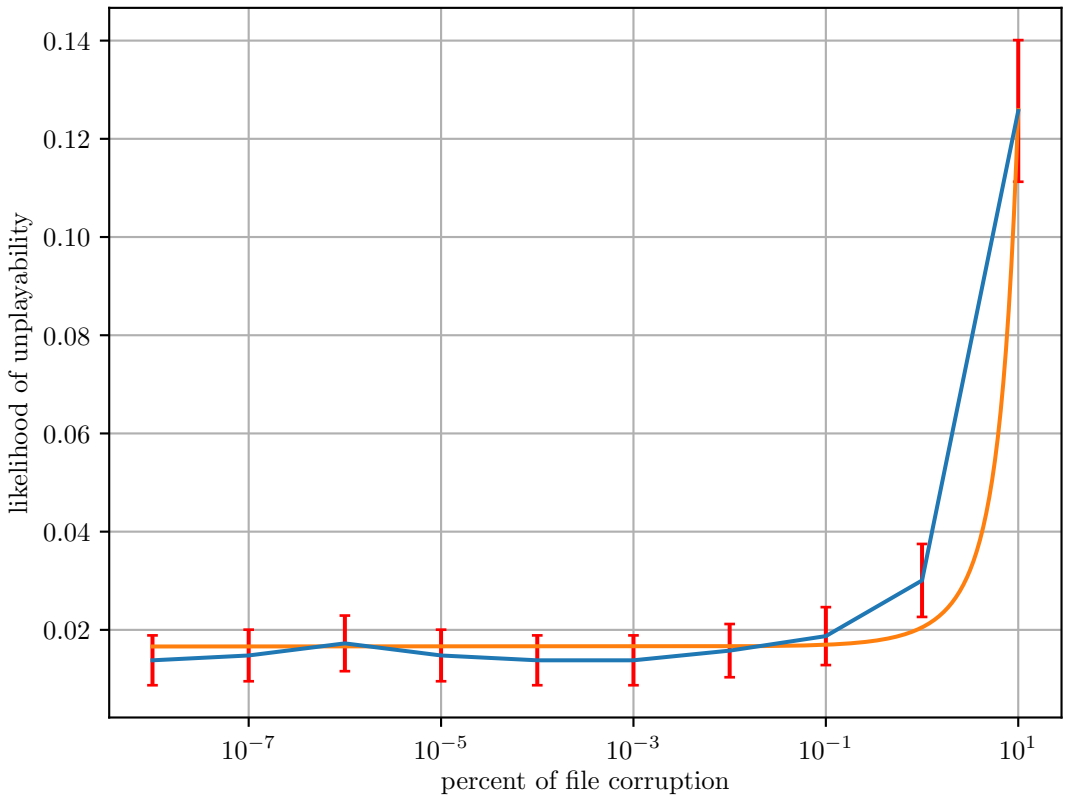

Figure 4.3: Measured proportion of unplayable files for segment blanking with fitted sigmoid curve

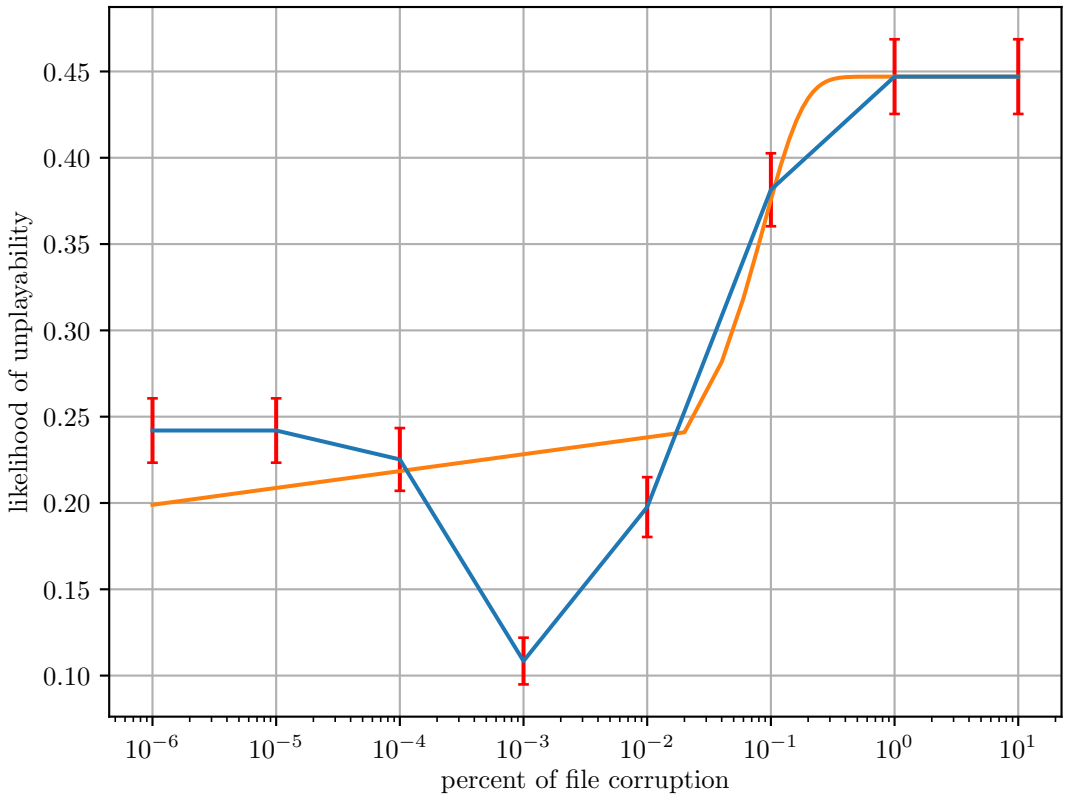

Figure 4.4: Measured proportion of unplayable files for truncation with fitted sigmoid curve

Chapter 5

Conclusion

Of all the methods in the corruption model, truncation had the most profound effect at low levels and is the most likely cause of spontaneously unplayable MP4 files. This is because the metadata required for playback is often placed at the end of an MP4 file. Relocating this metadata to the beginning of the file can successfully preserve a file's playability even when subjected to significant amounts of truncation. This modification in file structure also solves another known problem when playing MP4 files stored on remote systems: Loading the metadata ahead of the media data allows media players to begin playback immediately while the file transfer continues; hence, this is known as the "faststart" problem.

The interesting trends in survivability for both bit-flipping and truncation merit further investigation. Bit-flipping was expected to have the weakest effect on playability but ended up having the strongest for a singular peak before returning to a small likelihood. This could also be due to an error in the experiment's implementation or execution. Truncation also had a mysterious dip in its graph where truncation of the "correct" amount might leave a file playable when any amount higher or lower might break this playability.

Bibliography

[1] Apple Computer, Inc. "Classic Version of the QuickTime File Format Specification." (Mar. 1, 2001), [Online]. Available: https://developer.apple.com/standards/classic-quicktime/ (visited on 03/28/2023).

[2] "Information technology – Coding of audio-visual objects – Part 1: Systems," International Organization for Standardization, Geneva, CH, Standard, Oct. 2001. [Online]. Available: https://www.iso.org/standard/34903.html (visited on 04/14/2023).

[3] Moving Picture, Audio, and Data Coding by Artificial Intelligence (MPAI) Community. "MPEG: The Moving Picture Experts Group [archive]." (Mar. 31, 2023), [Online]. Available: https://mpeg.chiariglione.org (visited on 04/14/2023).

[4] Moving Picture, Audio, and Data Coding by Artificial Intelligence (MPAI) Community. "MPAI.community." (2021), [Online]. Available: https://mpai.community (visited on 04/14/2023).

[5] "Information technology – Coding of audio-visual objects – Part 12: ISO base media file format," International Organization for Standardization, Geneva, CH, Standard, Jan. 2022. [Online]. Available: https://www.iso.org/standard/83102.html (visited on 04/14/2023).

[6] "Information technology – Coding of audio-visual objects – Part 14: MP4 file format," International Organization for Standardization, Geneva, CH, Standard, Jan. 2020. [Online]. Available: https://www.iso.org/standard/79110.html (visited on 04/14/2023).

[7] "Information technology – Coding of audio-visual objects – Part 15: Carriage of network abstraction layer (NAL) unit structured video in the ISO base media file format," International Organization for Standardization, Geneva, CH, Standard, Oct. 2022. [Online]. Available: https://www.iso.org/standard/83102.html (visited on 04/14/2023).

[8] Apple Inc. "QuickTime File Format Specification." (Sep. 13, 2016), [Online]. Available: https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html (visited on 03/28/2023).

[9] L. Zhao and L. Guan, "An optimized method and implementation for parsing MP4 metadata," in *2010 IEEE International Conference on Progress in Informatics and Computing*, vol. 2, 2010, pp. 984–987. DOI: 10.1109/PIC.2010.5687866. (visited on 03/07/2024).

[10] L. Fegaras, "The Joy of SAX," in *XIME-P*, Citeseer, 2004, pp. 61–66.

[11] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1974. DOI: 10.1109/TCOM.1974.1092259. (visited on 08/04/2024).

[12] M. K. Padmanaban and J. Ilow, "Evaluating packet erasure recovery techniques for video streaming," in *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2015, pp. 1115–1119. DOI: 10.1109/CCECE.2015.7129430. (visited on 03/16/2024).

[13] S. Gibson, *SpinRite: "What's under the hood"*, 1993. [Online]. Available: https://www.grc.com/files/technote.pdf (visited on 02/07/2023).

[14] J. Gray and C. van Ingen, *Empirical measurements of disk failure rates and error rates*, Dec. 2005. arXiv: cs/0701166 [cs.DB]. (visited on 10/28/2024).

[15] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: high-performance, reliable secondary storage," *ACM Comput. Surv.*, vol. 26, no. 2, pp. 145–185, Jun. 1994, ISSN: 0360-0300. DOI: 10.1145/176979.176981. (visited on 08/16/2024).

[16] J. Bonwick and B. Moore, "ZFS, the last word in file systems," 2008. [Online]. Available: https://www.snia.org/sites/default/orig/sdc_archives/2008_presentations/monday/JeffBonwick-BillMoore_ZFS.pdf (visited on 08/16/2024).

[17] Oracle, "Checking ZFS File System Integrity," in *Oracle Solaris ZFS Administration Guide*. 2006. [Online]. Available: https://docs.oracle.com/cd/E18752_01/html/819-5461/gbbwa.html (visited on 05/05/2023).

[18] O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The Linux B-Tree Filesystem," *ACM Trans. Storage*, vol. 9, no. 3, Aug. 2013, ISSN: 1553-3077. DOI: 10.1145/2501620.2501623.

[19] A. Pal and N. Memon, "The evolution of file carving," *IEEE Signal Processing Magazine*, vol. 26, no. 2, pp. 59–71, 2009. DOI: 10.1109/MSP.2008.931081. (visited on 03/17/2024).

[20] R. Poisel and S. Tjoa, "A comprehensive literature review of file carving," in *2013 International Conference on Availability, Reliability and Security*, 2013, pp. 475–484. DOI: 10.1109/ARES.2013.62. (visited on 03/17/2024).

[21] G. G. Richard III and V. Roussev, "Scalpel: A frugal, high performance file carver.," in *DFRWS*, 2005.

[22] R. Poisel and S. Tjoa, "Roadmap to approaches for carving of fragmented multimedia files," in *2011 Sixth International Conference on Availability, Reliability and Security*, 2011, pp. 752–757. DOI: 10.1109/ARES.2011.118. (visited on 03/22/2024).

[23] E. Casey and R. Zoun, "Design tradeoffs for developing fragmented video carving tools," *Digital Investigation*, vol. 11, S30–S39, 2014, Fourteenth Annual DFRWS Conference, ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.05.010. (visited on 08/21/2024).

[24] G.-H. Na, K.-S. Shim, K.-W. Moon, S. G. Kong, E.-S. Kim, and J. Lee, "Frame-based recovery of corrupted video files using video codec specifications," *IEEE Transactions on Image Processing*, vol. 23, no. 2, pp. 517–526, 2014. DOI: 10.1109/TIP.2013.2285625. (visited on 02/07/2023).

[25] K. Alghafli and T. Martin, "Identification and recovery of video fragments for forensics file carving," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016, pp. 267–272. DOI: 10.1109/ICITST.2016.7856710. (visited on 03/17/2024).

[26] S. Godsill and P. Rayner, "Frequency-based interpolation of sampled signals with applications in audio restoration," in *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, 1993, 209–212 vol.1. DOI: 10.1109/ICASSP.1993.319092. (visited on 02/09/2023).

[27] L. Simon, J. C. Valiere, and C. Depollier, "New contribution on noise reduction using wavelet techniques: Application to the restoration of old recordings," in *Audio Engineering Society Convention 94*, Mar. 1993. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=6675 (visited on 02/09/2023).

[28] J. B. Weaver, Y. Xu, D. M. Healy Jr., and L. D. Cromwell, "Filtering noise from images with wavelet transforms," *Magnetic Resonance in Medicine*, vol. 21, no. 2, pp. 288–295, 1991. DOI: https://doi.org/10.1002/mrm.1910210213. (visited on 02/09/2023).

[29] A. M. John, K. Khanna, R. R. Prasad, and L. G. Pillai, "A review on application of fourier transform in image restoration," in *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2020, pp. 389–397. DOI: 10.1109/I-SMAC49090.2020.9243510. (visited on 02/09/2023).

[30] G. Zhang and R. Stevenson, "Efficient error recovery for multiple description video coding," in *2004 International Conference on Image Processing, 2004. ICIP '04.*, vol. 2, 2004, 829–832 Vol.2. DOI: 10.1109/ICIP.2004.1419427. (visited on 03/16/2024).

[31] H.-b. Chen, Y.-n. Dong, and H. Shi, "Data hiding based error recovery for h.264 video streaming over wireless networks," in *2011 Visual Communications and Image Processing (VCIP)*, 2011, pp. 1–4. DOI: 10.1109/VCIP.2011.6115931. (visited on 03/16/2024).

[32] G. M. P. D., B. Madathil, and S. N. George, "Entropy-based reweighted tensor completion technique for video recovery," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 2, pp. 415–426, 2020. DOI: 10.1109/TCSVT.2019.2892848. (visited on 03/16/2024).

[33] P. M. Hoang, H. D. Tuan, T. T. Son, and H. V. Poor, "Qualitative HD Image and Video Recovery via High-Order Tensor Augmentation and Completion," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 3, pp. 688–701, 2021. DOI: 10.1109/JSTSP.2020.3042063. (visited on 03/16/2024).

[34] T. Gloe, A. Fischer, and M. Kirchner, "Forensic analysis of video file formats," in *Digital Investigation*, Proceedings of the First Annual DFRWS Europe, vol. 11, 2014, S68–S76. DOI: 10.1016/j.diin.2014.03.009. (visited on 03/16/2024).

[35] D. Shullani, M. Fontani, M. Iuliani, O. A. Shaya, and P. Alessandro, "VISION: a video and image dataset for source identification," *EURASIP Journal on Information Security*, vol. 2017, no. 1, p. 15, Oct. 2017, ISSN: 2510-523X. DOI: 10.1186/s13635-017-0067-2. (visited on 03/12/2024).

[36] D. Baracchi, D. Shullani, M. Iuliani, and A. Piva, "FloreView: An Image and Video Dataset for Forensic Analysis," *IEEE Access*, vol. 11, pp. 109 267–109 282, 2023. DOI: 10.1109/ACCESS.2023.3321991. (visited on 03/12/2024).

[37] P. Yang, D. Baracchi, M. Iuliani, *et al.*, "Efficient Video Integrity Analysis Through Container Characterization," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 947–954, 2020. DOI: 10.1109/JSTSP.2020.3008088. (visited on 03/08/2024).

[38] S. Milani, M. Fontani, P. Bestagini, *et al.*, "An overview on video forensics," *APSIPA Transactions on Signal and Information Processing*, vol. 1, e2, 2012. DOI: 10.1017/ATSIP.2012.2. (visited on 03/16/2024).

[39] M. S. Rana, M. N. Nobi, B. Murali, and A. H. Sung, "Deepfake detection: A systematic literature review," *IEEE Access*, vol. 10, pp. 25 494–25 513, 2022. DOI: 10.1109/ACCESS.2022.3154404. (visited on 08/04/2024).

[40] Z. Xiang, J. Horváth, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, "Forensic Analysis of Video Files Using Metadata," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 1042–1051. DOI: 10.1109/CVPRW53098.2021.00115. (visited on 03/05/2024).

[41] Z. Xiang, A. K. Singh Yadav, P. Bestagini, S. Tubaro, and E. J. Delp, "MTN: Forensic Analysis of MP4 Video Files Using Graph Neural Networks," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2023, pp. 963–972. DOI: 10.1109/CVPRW59228.2023.00103. (visited on 03/05/2024).

[42] S. Hemalatha, U. D. Acharya, and Shamathmika, "MP4 video steganography in wavelet domain," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1229–1235. DOI: 10.1109/ICACCI.2017.8126010. (visited on 03/05/2024).

[43] O. Tange, *GNU Parallel 20241022 ('Sinwar Nasrallah')*, GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them., Oct. 2024. DOI: 10.5281/zenodo.13957646. (visited on 11/01/2024).