IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 2, 2020, accepted June 13, 2020, date of publication June 18, 2020, date of current version June 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3003378

VTIM: Video Title Identification Using Open Metadata

JUHYUNG SONG[1], SUYEONG LEE[2], BAEKJUN KIM[2], SOONUK SEOL[3], BEN LEE[4], AND MYUNGCHUL KIM[1,2], (Member, IEEE)

[1]Graduate School of Information Security, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea
[2]School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea
[3]School of Electrical, Electronics and Communication Engineering, Korea University of Technology and Education (KOREATECH), Cheonan 31253, South Korea
[4]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA

Corresponding author: Myungchul Kim (mck@kaist.ac.kr)

ABSTRACT Video-streaming applications are very popular these days. Existing studies of video streaming have attempted to identify video titles of users using machine learning techniques to identify specific patterns of video packets transmitted over the network. However, these studies have limitations when applied to actual environments where the network is congested or there are multiple users in the same network. This paper proposes *Video Title Identification using open Metadata* (VTIM), which identifies video titles by analyzing storyboards and Media Presentation Description (MPD) of MPEG-DASH in connection with video packets transmitted over the network. Attack was carried out using VTIM on 13,291 videos selected from actual video-streaming environment of YouTube. Our experiments show that VTIM is able to identify video titles with 100% accuracy at nearly thirty times faster than existing methods based on machine learning techniques. The paper also proposes and evaluates a countermeasure against VTIM.

INDEX TERMS Information security, network security, video surveillance.

I. INTRODUCTION

According to a forecast by Cisco, video traffic has grown on average of 33% annually and is expected to account for 82% of the Internet traffic worldwide by 2022 [1]. As of October 2018, more than 47% of worldwide video traffic stems from video-streaming applications [2], such as YouTube [3] and Netflix [4]. Currently, YouTube and Netflix use Moving Picture Experts Group-Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [5], which provides smooth user viewing experience and efficient video delivery over the network using HTML5 based web applications.

Video-streaming applications are categorized into progressive download and adaptive streaming methods depending on how videos are delivered to users [6]. The former method involves downloading an entire video file from a video server to a client in a single request, while the latter method divides a video into ''chunks'' and periodically downloads the necessary video chunks for a predetermined period. Since the former method receives an entire video from the server at the initial playback time, even portions of the video that users will end up not watching, the server will become more loaded compared to the latter method where only the video chunks required for the current video playback point are received. Therefore, the adaptive streaming approach is more widely used. These approaches include Apple's HTTP Live Streaming (HLS), Microsoft's Smooth Streaming, Adobe's HTTP Dynamic Streaming, and MPEG-DASH. Transmitting video data between a server and a client creates a unique traffic pattern that distinguishes each video according to the size and the number of video packets. If this unique traffic pattern can be identified, the video title that a user is watching can then be known. Currently, video-streaming applications apply Secure Socket Layer (SSL)/Transport Layer Security (TLS) encryption technology over the HTTPS protocol when servers and clients send and receive data [7], [8]. Despite the application of secure encryption technology, identification of video titles is still possible because encrypted packets also create unique traffic patterns. The identification of video titles is important for some reasons: (1) invasion of privacy in video streaming application [14], (2) harmful video censorship techniques [15], (3) employee-monitoring solutions, and (4) Quality of Experience (QoE) assessments in video streaming application for network operators [16]. Identifying video titles that

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

TABLE 1. Comparison of machine learning based video title identification methods.

Existing methods	The number of videos	Classifiers in machine learning	Accuracy in experiments
Dubin *et al.* [9]	2,100	Support Vector Machine (SVM) and nearest-neighbor algorithms	95%
Schuster *et al.* [10]	20	Convolutional neural networks	99.5%
Miller *et al.* [11]	Not specified	Logistic regression and hidden Markov model	90%
Xie *et al.* [12]	1,000	C4.5 decision tree, SVM and BPNN	99.9%
Pan *et al.* [13]	10	C4.5 decision tree, Random Forest, Bayes networks and AdaBoost	97.57%

users are watching can leak personal information such as a person's political and sexual orientation and entertainment preference. In addition, video title identification is important to detect and isolate harmful videos such as child pornography from the Internet. Similarly, in workplaces, employee-monitoring solutions are used to reduce employees' non-work related activities so as to improve employee work productivity. Another application is for network operators to predict QoE and to provision resources proactively.

A number of studies [9]–[13] have been conducted to identify video titles watched by clients and they use mainly machine learning methods to distinguish video traffic features that change over time. In another study, a technique called dynamic time warping was used to measure the similarity between video fingerprints or signatures calculated using video packet length [17]. However, these studies are inadequate in real-world applications because they assume that the network condition is stable such that the same traffic pattern is always reproduced for the streaming of one video. However, traffic patterns of videos will not be unique when the network is congested because the size and the number of video packets depend on the state of the network. Moreover, these studies assume that an attacker and a user are on the same network path. This assumption makes it difficult to identify the unique traffic pattern of a video when multiple users watching it at the same time due to nested video packets.

Based on the aforementioned discussion, video title identification commonly exploits video traffic patterns using machine learning methods. In contrast, this paper proposes *Video Title Identification using open Metadata* (VTIM), which exploits open metadata from the storyboard that summarizes video playback scenes located within the video webpage's source codes [18] and the Media Presentation Description (MPD) of MPEG-DASH that provides the necessary information for video playback of the client. Once a video is uploaded to a video server, metadata, such as MPDs and storyboards, are generated and do not change until the video is modified or deleted. Therefore, videos are identified by first collecting and parsing storyboards and MPDs from a real environment and then associating them with packets obtained during video downloading. Finally, we introduce a novel video title identification attack by exploiting Common Vulnerabilities and Exposures (CVE). The main contributions of this paper are as follows:

- VTIM can identify video titles using open metadata from storyboards and MPDs with respect to video traffic. This is in contrast to using machine learning techniques based on network video traffic features.
- VTIM can identify titles of videos in a real-world setting such as YouTube, which encrypts video traffic to protect users' privacy information and obfuscates video webpage's source codes to prevent illegal copying of videos.
- VTIM was implemented and demonstrated in a real-world environment with a representative video set from YouTube. Our study shows that video titles can be identified significantly faster than previous methods and with 100% accuracy.
- A countermeasure is also proposed and implemented against VTIM.

This paper is organized as follows. Section II introduces the related work. Section III describes the vulnerabilities on the storyboard and MPEG-DASH. Section IV presents our attack model and attack procedures. Section V describes the modules used in VTIM. Section VI explains our experimental environment and the results of the attack. Section VII proposes a countermeasure against the attack and Section VIII presents experimental results on the countermeasure. Finally, Section IX concludes the paper and discusses possible future work.

II. RELATED WORK

Existing methods for identifying video titles using video traffic features are based on machine learning techniques that learn the unique traffic patterns of MPEG-DASH video packets over HTTPS [9]–[13]. As shown in Table 1, we compared existing methods according to three criteria: the number of videos, classifiers in machine learning, and the accuracy of the experiments.

Dubin *et al.* [9] created a new video traffic feature called *bit-per-peak* to represent the peak value of the video packet size according to the video playback time on YouTube and learned it using machine learning algorithms such as the Support Vector Machine (SVM) [19] and nearest-neighbor [20] algorithms. Their study showed that if the network becomes congested and the packet loss rate increases by up to 6%, the accuracy of video pattern identification decreases significantly to less than 80%. To prevent a drop in accuracy, the number of streams that captured each video packet had to be increased. In order to increase the accuracy to 95% or more, the packet loss rate needed to be less than 3% and at least five streams were required for each video. Also, it took a month to collect 10,000 streams during the experiment.

Schuster *et al.* [10] calculated the byte size and the average length of all packets, including downstream and upstream packets for both downstream and upstream between the video server and the client. When the packet size and length were

learned through convolutional neural networks [21], the training time to identify 20 video titles took 94 seconds.

Miller *et al.* [11] measured the number of response packets from a web server to a client and the number of request packets from the client to the web server. The former measurement is defined as a positive integer and the latter is a negative one. They paired positive values with negative values to identify unique patterns of video through logistic regression [22] and hidden Markov models [23]. However, an average of 4.3 links were required per URL for each video's webpage to learn patterns leading to large computational overhead.

Xie *et al.* [12] attempted to analyze video scenes and unique traffic patterns when quick-scene action movies were played. Based on the two video themes of romance and action movies, video packets were grouped according to size and a traffic feature called the *byte code distribution* was created. However, their study did not include identification tests of videos in a real environment. Their method is also difficult to apply in a real-world environment because video packets must be captured every three seconds through machine learning algorithms such as SVM, Back-Propagation Neural Networks (BPNN) [24] or C4.5 Decision Tree [25].

Pan and Cheng [13] undertook related work by analyzing YouTube video traffic for QoE assessments. They collected server and client IP addresses and trained by unique traffic patterns with C4.5 Decision Tree [25], Random Forest [26], Bayes networks [27] and AdaBoost [28] to classify video packets. The experiment result shows 97.57% accuracy in classifying video packets. In order to verify the experiment result from machine learning techniques, a debugger analyzing unique traffic patterns was developed with a Man-In-The-Middle Attack (MITM) [29] tool such as Fiddler [30].

In summary, the work of [9]–[13] provides the method to predict the unique traffic patterns of video packets using machine learning techniques, and to identify the video titles without decryption of video packets. In contrast to the aforementioned related work that relied on machine learning techniques and traffic features to identify video titles, the proposed VTIM uses open metadata contained in storyboards and MPDs. Therefore, it offers the following advantages:

- VTIM can identify the title of a video not only when video resolution degrades but also in the event that several videos of the same resolution are played at the same time. This contradicts existing studies that did not take into account simultaneous video playback by multiple users and network congestion.
- VTIM can significantly reduce the time needed to collect video streams and identify the related video titles.

III. STORYBOARD AND MPEG-DASH

This section explains the video webpage's source codes and storyboard, discusses the vulnerabilities of storyboard and MPEG-DASH, and the limitations of existing methods that rely on traffic patterns in MPEG-DASH.

A. VIDEO WEBPAGE'S SOURCE CODES

In YouTube, a video webpage hides the source codes of the website using obfuscation to protect its intellectual property rights (e.g., JavaScript codes handling events such as video playback and shutdown and displaying advertisements). However, the URLs of storyboard and MPD in the video webpage's source codes are not obfuscated. This is the starting point of our work. Even when video traffic is encrypted by HTTPS, it is possible to identify video titles by collecting metadata such as the MPD and storyboard, which are not obfuscated in the video webpage's source code.

B. STORYBOARD

When a video is uploaded to YouTube, it is fragmented into several video chunks, called *segments*, where each segment represents a single image frame during video playback. A storyboard represents a sequence of images where each image tiles several image frames so that an entire video can be viewed at a glance. The storyboard supports two types of resolutions, 800 $\times$ 350 and 800 $\times$ 315 pixels (p) depending on the image size, and can be viewed in the application's playback bar. The storyboard displays the playback screen that the user wants to watch without moving the actual playback position in the user's video. This reduces the time needed for a user to move to a desired playback point. Then, YouTube videos have multiple storyboards because YouTube summarizes the videos according to the video playback time, with longer videos having more storyboards. For instance, a video with a playback time of 1 minute 50 seconds has approximately seven storyboards, and typically two resolutions of storyboards.

C. VULNERABILITY ON THE STORYBOARD

The storyboards of a video are delivered from the server to the client when the video is being played or when the user is moving the playback point on the playback bar. The storyboards discussed in the present work are limited to the former type. The file sizes of storyboards do not change even when the video's resolution changes. Therefore, they can be used as a feature to identify the video title. To obtain the file sizes of the storyboards, the video webpage's source codes are parsed. Thus, the file sizes of storyboards can be obtained and used to identify the title of a video without having to de-obfuscating the video webpage's source codes. Note that with the exception of parsing the file sizes of the storyboards, obfuscation of the video webpage's source codes can also be undone by an attacker [31]. However, this requires a considerable amount of time to analyze the source codes, which is not suitable as an actual attack.

D. MPEG-DASH

As shown in Figure 1, MPEG-DASH consists of a DASH server and a DASH client. The video playback process consists of the following ten steps: (1) The user uploads the video to the DASH server through a video-streaming application.

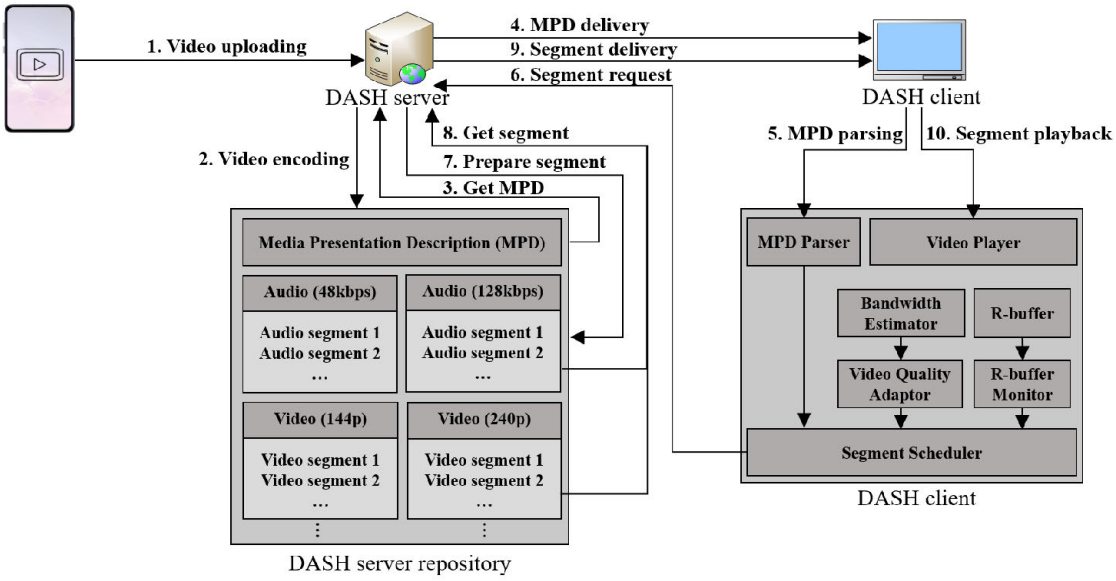

FIGURE 1. MPEG-DASH video playback process.

(2) The DASH server encodes the uploaded video into an MPD with the audio segment and video segment in the related repository. (3) When a user requests a video, the DASH server obtains the corresponding MPD from its repository. (4) The DASH server delivers the MPD to the DASH client.

(5) *MPD Parser* in the DASH client parses and interprets the MPD and informs *Segment Scheduler* of the video segment playback information. The DASH client also has *Bandwidth Estimator* that measures bandwidth and *R-buffer* [32], which is a playback buffer. The measured bandwidth is then passed to *Video Quality Adapter*, which gives *Segment Scheduler* a resolution that can be played back. *R-buffer Monitor* tracks the remaining buffer capacity and provides this information to *Segment Scheduler*. (6) *Segment Scheduler* requests to the DASH server video segments based on the analyzed information, resolution, and remaining buffer capacity.

(7) The DASH server prepares the requested video and audio segments in its repository. (8) The DASH server retrieves the video and audio segments from the repository. (9) The DASH server delivers video and audio segments to the DASH client. Finally, *Video Player* in the DASH client plays the received segments.

E. LIMITATIONS OF THE RELATED WORK ON USING TRAFFIC PATTERNS OF MPEG-DASH

The DASH server delivers video segments to the client with respect to the video playback time. In the process of delivering video segments, a unique traffic pattern is generated for each video according to the request time of the video and audio segments and the duration of the responding traffic. This traffic pattern is referred to as the ON-OFF periods, where ON refers to when video segments are received

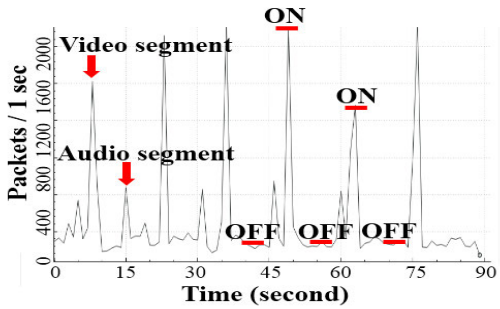

FIGURE 2. ON-OFF pattern of the received video segments.

and OFF refers to when no video segments are received [33]. Figure 2 shows a typical ON-OFF pattern when the client receives video segments from the DASH server. Earlier studies [9]–[12] identify video titles using machine learning techniques such as SVM, BPNN and C4.5 Decision Tree to such traffic patterns with a network protocol analyzer such as Wireshark [34]. However, these approaches have the following limitations:

- When network congestion occurs at the DASH client, the resolution of the video watched by the user will dynamically change [35]. When the video resolution changes, the ON-OFF pattern is distorted by a large spike due to retransmission packets as shown in Figure 3. If a spike occurs in the ON-OFF pattern even once, it cannot be used to identify the video title.
- When multiple users on the same campus network simultaneously play the same video, ON-OFF patterns overlap in terms of the packet arrival time and thus they cannot be used to identify the video title. Figure 4 and

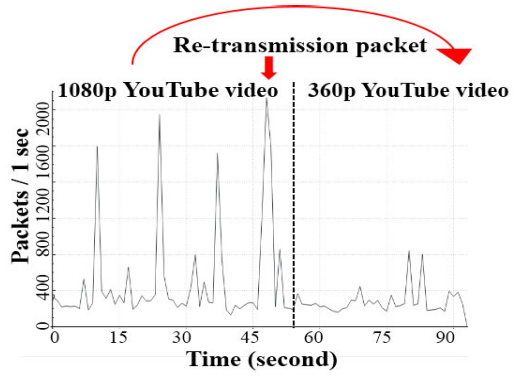

FIGURE 3. ON-OFF pattern distortion caused by retransmission of a packet.

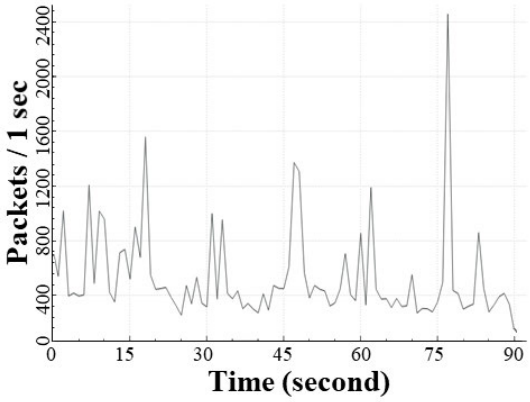

FIGURE 4. ON-OFF pattern distortion during simultaneous playback of two 360p videos.

Figure 5 shows two different cases of the same resolution video being played simultaneously: one at 360p and the other at 1080p. The only difference between Figure 4 and Figure 5 is the resolution. As can be seen, no clear ON-OFF pattern is created when the videos are played simultaneously, regardless of their resolution.

Therefore, identifying video titles in real-world situations is challenging using only video traffic features.

F. VULNERABILITY OF MPEG-DASH

The MPD in MPEG-DASH has open metadata information so that video resolution can be dynamically switched for smooth playback. As shown in Figure 6, each video has an MPD written in the XML format. The topmost level of MPD has one or more *Periods*, each containing the total duration of a section of a video. At the next level, each *Adaptation Set* tag has a MIME-type information a DASH client uses to determine what type of segment has been downloaded. In addition, these tags provide information about the language support for the video and have only one audio Adaptation Set tag and one video Adaptation Set tag in general supporting only a single language.

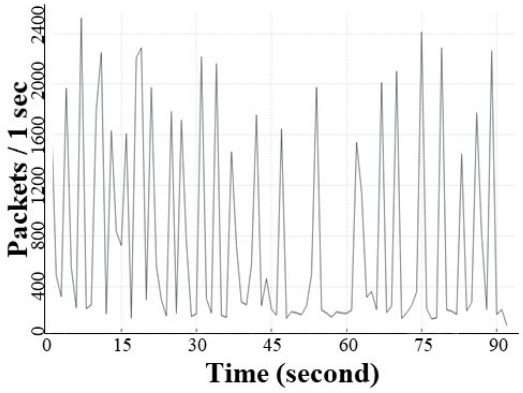

FIGURE 5. ON-OFF pattern distortion during simultaneous playback of two 1080p videos.

The *Representation* tags, which are subtags of the Adaptation Set, contain the video's height and ID and are used to distinguish supported resolutions. The number of resolutions supported is as many as the number of Representation tags, and each one has open metadata information for a single resolution. Depending on the network condition, the video player dynamically switches to the Representation tag of the selected resolution. Moreover, the resolution can be manually changed by the user.

Each Representation tag contains subtags. For example, A *SegmentList* tag inside the Representation 7 and 8 tags consist of *Initialization Segment* and *SegmentURL media*. The Initialization Segment has segment's playback information. The SegmentURL media is used to convert the segment into the playback range so that it can be played out by the video player. The SegmentURL media is composed of duration represented in time units and segment range represented in byte units. The following examples show how segment duration can be defined:

<SegmentURL media =“sq/1/dur/3.05”
= Playtime 0∼3.05 seconds
<SegmentURL media =“sq/2/dur/3.05”>
= Playtime 3.06∼6.1 seconds

This example indicates that after the first segment is played for 3.05 seconds, the second segment is also played for 3.05 seconds from the next playback point. On the other hand, the segment range can be defined as follows:

<SegmentURL media =“range/1214-26315”/>
= 26315 – 1214 + 1
= 25,102 bytes
<SegmentURL media =“range/26316-44327”/>
= 44327 – 26316 + 1
= 18,012 bytes

The above example indicates that the first segment is 25,102 bytes and the second segment plays is 18,012 bytes. Moreover, the second segment is played after the first segment is played.

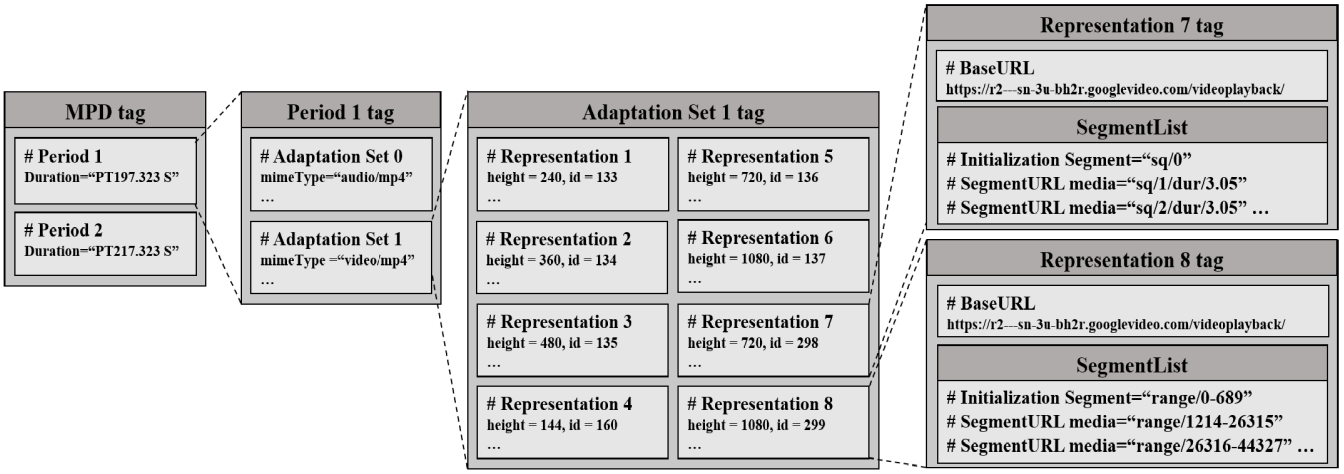

FIGURE 6. Structure of the MPD.

Currently, YouTube users can download a video up to approximately six hours with unlimited access to user rights during a single request. The segment duration and range information provided by SegmentURL media is open metadata. Therefore, an attacker can analyze the MPD to retrieve the segment sizes and then identify the video title.

IV. OVERVIEW OF THE ATTACK

A. ATTACK MODEL

The attack model consists of an attacker and multiple victims on the same campus network. Victims use Internet-connected devices such as PCs and mobile phones to watch videos using YouTube. YouTube is the most popular video streaming application in the world [36]. In addition, compared to video streaming applications (e.g., Netflix and Vimeo), it is easy to collect multiple videos for experiments and automatically encode and upload videos onto YouTube. Accordingly, there is no video that cannot be played back. This helps to reduce the time needed to collect the videos when experimenting with them. Furthermore, the attack model assumes that the attacker is authorized to set up a necessary proxy to engage in further attacks. The resolution of a video can be degraded or upgraded in real time depending on the state of the network. The attacker creates a proxy to eavesdrop on the victims' video traffic using a MITM tool. In other words, the attacker sniffs a communication channel by relaying the video traffic between the DASH server and the victims, which is done by MITM. In this situation, the following two assumptions are made about the attacker:

- The attacker and victims access the same video set from the DASH server. In other words, the list of video titles that the attacker and victims can select is identical. The attacker looks up the Video ID to identify the title of the video from the video set. For example, YouTube URLs have a Video ID that identifies the title of the video and a Playlist ID that allows users to create video groups that match their subject of interest.
- The attacker can capture and analyze video traffic of multiple victims on the same path to distinguish individual video packets from multiple users.

A YouTube video can be deleted or changed at any time. Thus, the attacker tracks the change in the video set whenever a video is added, changed, or deleted. For example, the attacker can detect that a new video was added when it finds that video packets do not correspond to any of the video segments in the video set.

B. ATTACK PROCEDURE

The attack procedure consists of the following four steps: (1) Download the storyboard and MPD, (2) Parse the MPD and extract the segment duration and range, (3) Set up a proxy on the victim to capture video traffic, and (4) Identify the video title of victims.

1) STEP 1. DOWNLOAD STORYBOARD AND MPD

The attacker searches the video webpage's source codes for the storyboard's URL and MPD's URL using keywords. The file is then downloaded into the attacker's device, and the procedure moves to *Step 2*.

However, if keywords that identify the MPD cannot be found, then the attacker uses the storyboard's URL only. As mentioned in Section III-A, the storyboard's URL in video webpage's source codes are not obfuscated and open to public thus they can still be downloaded. The attacker then obtains the storyboard's file sizes and logs this information to a file, then proceeds to *Step 4*. If the storyboard's URL is also not found, the attacker will not be able to identify the video.

2) STEP 2. PARSE MPD AND EXTRACT SEGMENT DURATION AND RANGE

In the case of an MPD which was downloaded in *Step 1*, the attacker parses its content. As mentioned in Section III-F, the MPD file contains a Representation tag for each video resolution. Therefore, the attacker obtains the segment information

of the SegmentURL media. If the segment information is represented by the segment duration, the video corresponding to the MPD in the video set should be encoded as *.m4s* format used in MPEG-DASH. Then, the segment sizes can be obtained by analyzing their file sizes from the encoded video. If the segment duration is not available, the attacker calculates the segment range as the segment size. After all segment durations and ranges have been converted to segment sizes, they are sorted in ascending order from the lowest resolution (144p video) to the highest resolution (1080p video) and saved in a file.

3) STEP 3. SET UP A PROXY ON THE VICTIM TO CAPTURE VIDEO TRAFFIC

The following attack scenario in this case can be exploited. Initially, the attacker must use Fiddler to capture video traffic encrypted by the HTTPS protocol from the victim. If the attacker attempts to use Fiddler on the victim, a proxy must have been previously set up on the victim. The specified proxy set-up procedures are as follows. The attacker sends emails which contain a notice of a campus blackout schedule to multiple victims on the campus network. In the emails, there are several document files that include malware (e.g., CVE-2019-0561 [37] and CVE-2014-2781 [38]) that create pop-up the alarm messages. Immediately after the documents are opened, alarm messages will pop-up with messages such as ''Will you allow usage of a proxy to improve the Internet speed of the campus network?'' and ''Will you refer to the embedded scripts to use the proxy?''. When a victim agrees to a proxy setting, it executes a command to disable the victim's firewall with the victim's administrator rights and then downloads ''BusyBox'' [39]. ''BusyBox'' supports a telnet service that facilitates communication with the attacker's server without the victim's awareness as well as a TFTP service that transfers files between the attacker and the victim. The attacker then exploits ''BusyBox'' to allow the victim's PC to connect to the telnet service with the pre-set attacker server's IP, account ID and PASSWORD. In addition, the attacker uses the TFTP service of ''BusyBox'' to send the victim's SSL certificate to the attacker's server. Finally, when a telnet connection is established, the LAN interface on the attacker's server is already set to the promiscuous setting such that when the victim watches the video, the victim's video packets can be passed from the attacker's server.

4) STEP 4. IDENTIFY THE VIDEO TITLE OF VICTIMS

In the case of a storyboard which was obtained in *Step 1*, the attacker collects storyboards from the video traffic of the victims and identifies the video titles by searching the file that stores the storyboard file sizes. Finally, with video titles that have identical storyboard file sizes, the attacker identifies the video title by comparing the pixel values from the storyboard by attacker and those from the storyboard that captured from the victim's video traffic. Otherwise, if the MPD was collected, the attacker converts segment duration of the MPD into segment size. Storyboard file sizes and segment sizes are compared with the size of captured segments in the video traffic watched by the victim for a specified period. The attacker counts the number of matches and identifies the victim's video title by selecting video with the highest number matches. If neither the storyboard nor the MPD is obtained, the attacker cannot obtain the file size of the storyboard and MPD, so the attacker exits without performing video title identification.

V. THE PROPOSED VTIM

The VTIM architecture in Figure 7 is devised with a detailed diagram of each module, which is required to preprocess the storyboards and MPDs and then to identify the video title. The figure shows the organization of the proposed VTIM consisting of two major modules: *Storyboard Preprocessing* and *MPD Preprocessing*. The Storyboard Preprocessing module consists of *Selector* that chooses either storyboards or MPDs, *Storyboard Downloader* that downloads storyboards, and *Writer* for sorting file sizes obtained from storyboards and MPDs. The MPD Preprocessing module includes *Selector* and *MPD Downloader* that downloads MPDs, *Analyzer* that parses MPDs, *Calculator* and *Writer* for analyzing MPDs. Finally, *Identifier* identifies video titles by comparing the victim's video packets with file sizes obtained from *Storyboard preprocessing* and *MPD preprocessing*.

A. SELECTOR

The *Selector* determines whether storyboards or MPDs are available in a video webpage's source codes. If both are available, the storyboards will be downloaded from DASH server (see Section V-B) at first. This is because storyboards are more approachable information for identifying victims' video titles compared to MPDs since storyboards are provided to public for all videos and MPDs are provided to public for some videos. Therefore, storyboards have higher priority for video title identification than MPDs. MPDs are only used if they are the only information available in the video webpage's source codes.

B. STORYBOARD DOWNLOADER AND MPD DOWNLOADER

If storyboards were selected by the *Selector*, the *Storyboard Downloader* searches the video webpage's source codes for the storyboard's URL with keywords such as *'https://i9.ytimg.com/sb/'* and *'PlayerStoryboardSpecRender'* using Selenium Framework [40]. It also collects the URLs pointing to storyboard for each video webpage, downloads two types of storyboard images (800 × 350 and 800 × 315 p). Then, the *Storyboard Downloader* delivers storyboards to the *Writer*. On the other hand, if an MPD was selected by the *Selector*, the *MPD Downloader* searches for keywords *'MPD'* and 'manifest' in the video webpage's source codes using the 'jQuery' library [41] to parse the MPD's URL and then downloads the MPD.

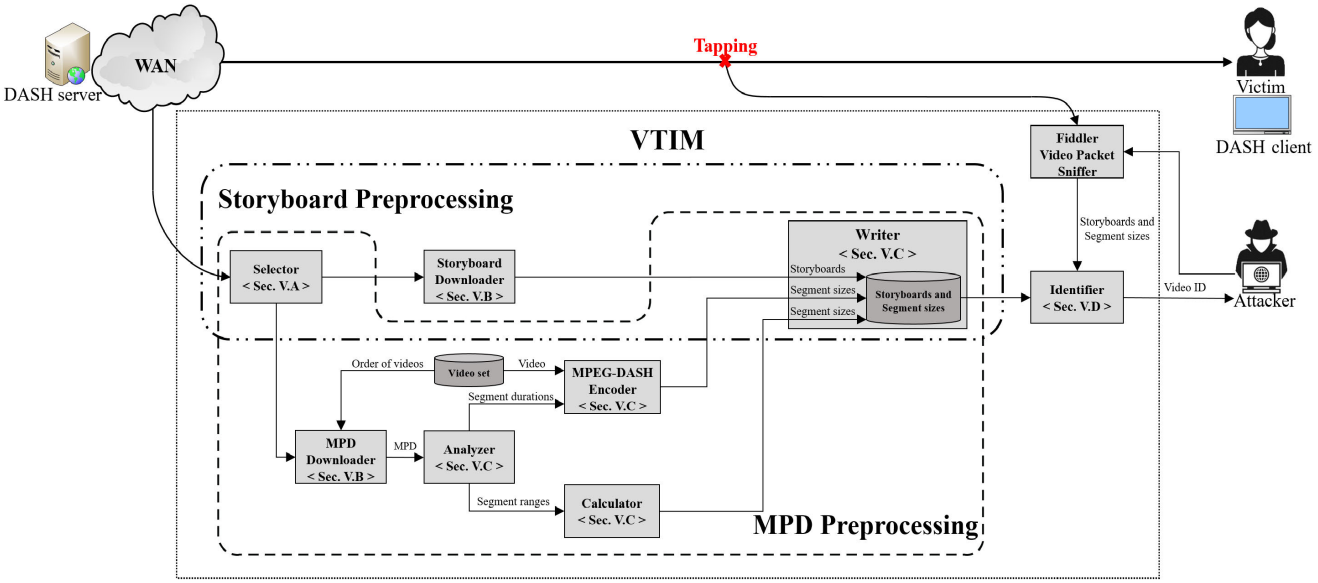

FIGURE 7. Overview of VTIM.

C. ANALYZER, CALCULATOR AND WRITER

The *Analyzer* parses MPD files collected by the *MPD Downloader* and searches for segment duration and range information within SegmentURL media, as discussed in Section III-F. In order to identify the video title, both the segment durations and ranges should be converted to segment sizes. Then, the *Analyzer* converts segment durations into segment sizes using the Bento 4 MPEG-DASH tool [42] for video transcoding, which re-encodes downloaded videos stored in the video set into video segments. On the other hand, the *Calculator* converts segment ranges into segment sizes. Afterwards, the *Writer* sorts segment sizes and storyboards according to the order of videos in the video set and writes segment sizes and storyboard file sizes to a file. The *Writer* delivers the file and storyboards to the *Identifier*.

D. IDENTIFIER

The *Identifier* distinguishes a victim's video whether storyboards or MPDs were recorded in the file. If storyboards were recorded, it will check whether the file sizes of the storyboards in the file and that of the victim's video traffic are identical. The *Identifier* identifies the titles of the videos by selecting the videos with the highest number of matches in the attacker's video set as shown Algorithm 1. The *Identifier* then checks if the pixel values of the storyboards from the identified video title match those of the storyboards from the victim's video traffic exactly. The video title can thus be found in attacker's video set when matching the pixel-values of storyboards.

If MPDs were recorded, the *Identifier* determines whether file sizes of video segments stored in the file are equal to the segment sizes of the victim's video traffic. Then, the *Identifier* identifies the video title by selecting the video with the highest number of matches in the attacker's video set as shown in Algorithm 1. Algorithm 1 has two functions to find the VideoID. The **FindVsize** function compares the victim's video segment size obtained from Fiddler with file sizes obtained from the *Writer*. The **FindVideoID** function returns the VideoID and VideoURL of the video set, which are the output of the **FindVsize** function, and stores them in a file.

VI. EXPERIMENTAL ENVIRONMENT AND RESULTS

This section describes the experimental environment and analyzes the results of VTIM.

A. EXPERIMENTAL ENVIRONMENT

1) ATTACKER AND VICTIMS

In order to carry out the attack on the same network path, both the attacker and victims are connected to the same campus network. The Internet speed of the campus network is 33.68 Mbps for downloading and 31.72 Mbps for uploading, which was measured using TestMy.net [43]. The attacker's device is a laptop with an Intel i5-8700K CPU at 3.6 GHz running the Windows 10 (64-bit) operating system.

Victims are also connected to the same campus network via wired/wireless connections. The attacker and victims use a router on the same campus network, and the video packets for victims' devices pass through Fiddler running on the attacker. Since Fiddler acts as a proxy and creates its HTTPS certificate, the attacker is able to intercept victims' video packets over HTTPS through Fiddler. Despite the advantages mentioned above, Fiddler cannot identify the video title or video ID from the video packets of the victims because the video packets do not contain the video title or video ID. Fiddler [44] captures video packets over the HTTP and HTTPS

TABLE 2. Videos collected from VTIM.

Category	Game	Movie	Cartoon	Music	News	Total
The number of videos	736 (100%)	3,590 (100%)	3,373 (100%)	543 (100%)	5,049 (100%)	13,291 (100%)
The number of videos preprocessed using file sizes of storyboards	608 (82.6%)	3,269 (91.0%)	3,062 (90.8%)	435 (80.1%)	4,398 (87.1%)	11,772 (88.6%)
The number of videos preprocessed using file sizes of MPDs	128 (17.4%)	321 (9.0%)	311 (9.2%)	108 (19.9%)	651 (12.9%)	1,519 (11.4%)
The number of storyboards collected during preprocessing	50,334					
The number of MPDs collected during preprocessing	1,519					

Algorithm 1 Algorithm for Finding Video ID

Input: File size from *Writer* (F_{size})
Segment or Storyboard size from Fiddler (V_{size})
Index of video set (*Idx*)
Counter variable matching V_{size} (*Counter*)
List of matching results (*Result*)
Output: ID of the video the victim is watching (*VideoID*)

function FindVsize(F_{size}, V_{size}, *Idx*)
  Make an empty dictionary (*Result*)
  for len(V_{size}) **do**
    for len(F_{size}) **do**
      if V_{size} *in* F_{size} **then**
        if not V_{size} in *Result.keys*() **then**
          Add a new key *Idx* matching on V_{size} with value 1
        else
          $Result[Idx] \leftarrow Result[Idx] + 1$
        end if
      end if
    end for
  end for
  Make a new list *Final* from *Result* having *Idx* as key and *Counter* as value
  $Final \leftarrow Sorted(Final)$ in descending order
  Make an empty ordered dictionary *OrderedResult*
  for data in *Final* **do**
    if not *Counter* in *OrderedResult.keys*() **then**
      Add a new key *Counter* with *Idx*
    else
      OrderedResult[*Counter*].*append*(*Idx*)
    end if
    return *OrderedResult*
  end for
end function
function FindVideoID(*OrderedResult*)
  for order in *OrderedResult.keys*() **do**
    Make an empty list *videoURL*
    for videoorder in *OrderedResult*[*order*] **do**
      videoURL.extend(*VideoID* with videoorder)
    end for
    Save *videoURL* and *VideoID* in a file
    return *VideoID*
  end for
end function
$OrderedResult \leftarrow$ **FindVsize**(F_{size},V_{size},*Idx*)
FindVideoID (*OrderedResult*)

protocols only in the application layer while Wireshark can capture video packets over HTTP in all layers. On the other hand, the attacker cannot capture video packets over HTTPS using Wireshark because it cannot act as a proxy for the MITM.

TABLE 3. The network condition of experimental environment.

The number of attacker			1
The number of victims			4
Sustainable Internet speed of victims (Mbps)	Stable	1080p video	33.68
		720p video	10.31
	Congestion	480p video	7.13
		360p video	5.32
		240p video	2.01
		144p video	0.95

In this experiment, the desktop and mobile phone of victims were used for the experiment. The specifications were as follows: a desktop with an Intel Core i5-8600k CPU at 3.6 GHz running the Windows 10 (64-bit) operating system, an iPhone XS (iOS 13.1.3) as the mobile phone and the Chrome browser (version 78.0.3904.87) for victims to watch YouTube videos. In our experiments, four victims are watching videos on a campus network, with the videos playing for exactly three seconds. The experiments were iterated 75,000 times with 11,772 videos for the storyboard cases. The experiments were then iterated 250,000 times with 1,519 videos for the MPD cases. The average playback time of the videos is approximately two minutes. In addition, the network condition of the experimental environment is described in Table 3. To experiment with network congestion, multiple HTTP request packets were sent to the victims using JMeter [45]. For the victims, it was necessary to ensure a sustainable Internet speed for video playback, even with network congestion. Note that the sustainable Internet speed in our experimental environment is the throughput Internet speed at which the victims experience resolution degradation of the video [46].

2) VIDEO COLLECTION

The video set was collected from YouTube. As shown in Table 2, the video set is composed of five themes: Game, Movie, Cartoon, Music, and News. Since we used youtube-dl [47], which is an open-source YouTube video crawling tool, the video set was collected automatically. In addition, because the number of videos downloaded depends on YouTube's algorithm, it is not in our control domain. However, given that the proposed VTIM does not rely on the category of the video, we concluded that an uneven distribution of categories does not degrade the validity or reliability of our research. Videos were categorized according to whether they can be preprocessed using file sizes of storyboards and MPDs for

the purpose of identifying their titles. The number of videos preprocessed using file sizes of storyboards is 11,772, which is 88.6% of the total number of videos, and the number of videos preprocessed using file sizes of MPDs is 1,519, which is 11.4% of the total number of videos. Furthermore, our video set is released to the public so that it can be used in additional research. The URLs of the video set are also recorded in the Appendix.

Table 2 shows that the number of storyboards collected during preprocessing of 11,772 videos is 50,334. This is because a storyboard summarizes only part of a video's playback, and thus there are multiple storyboards per video. On the other hand, the number of MPDs collected during preprocessing of 1,519 videos is 1,519. Since there is only a single MPD for a video, the number of MPDs collected is as many as the number of videos. In addition, when we download the videos from YouTube, storyboards and MPDs are downloaded as jpeg files and segments, respectively. In order to determine the file sizes which are used directly for a comparison with the victim's video, they should be preprocessed appropriately. As shown in Figure 7, the attacker processes the downloaded YouTube videos into multiple segments using MPEG-DASH Encoder [42], which is open-source software used to preprocess a video into video segments in the MPEG-DASH format. Then, the attacker preprocesses storyboards and video segments to obtain storyboard file sizes and video segment sizes, respectively, from the DASH server and stores them in files. Then, the attacker identifies video titles by comparing the storyboard file sizes and the video segment sizes obtained from VTIM with video packets of the victim.

The following subsections discuss accuracy, precision, and processing time of the proposed VTIM.

B. ACCURACY AND PRECISION

Accuracy is a measure of whether an attacker is correct when attempting to identify a victim's video title [48], and *precision* is a measure of whether the proposed VTIM is consistent when it repeatedly identifies the video title [48]. They are expressed as follows:

$$Accuracy = \frac{TruePositive + TrueNegative}{Positive + Negative}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

where *Positive* means the attack was successful, *Negative* means the attack was unsuccessful, *TruePositive* is an outcome where the proposed VTIM correctly predicts *Positive* and *FalsePositive* is an outcome where the proposed VTIM incorrectly predicts *Positive*.

C. TEST CASES FOR MEASURING ACCURACY AND PRECISION OF STORYBOARDS

To identify the video title, considerations for following two test cases are as follows: stable Internet speed and simultaneous viewing of video by multiple users. To measure the accuracy and precision of storyboards in aforementioned experimental situations, two test cases were generated as follows:

1) Identify the victim's video title based on 2 storyboard file sizes and calculate its accuracy and precision
2) Identify the title of the video when two victims concurrently play the same video at the same time and calculate its accuracy and precision

The accuracy and precision were measured in the proposed VTIM with two test cases under the condition that two victims are watching the same videos simultaneously. In the first case, the accuracy and precision were measured when the Internet speed was stable. In the second case, the attacker receives multiple video packets from the victims, and two victims are watching the same video at the same time. The two victims' video packets are identified by the attacker.

As mentioned in Section VI-A2, there were 50,334 storyboards collected from 11,772 videos. The number of storyboards is used to define the total probability space to find the title of video using Algorithm 1.

Let event I_K as an event which is finding the title of video. Then, event I_K is an exclusive event which means each video title identification experiment does not affect other experiments. The total probability space can be calculated according to the Law of Total Probability from Bayes Theorem [49]. The K value corresponding to the total number of events I_K was calculated by measuring the number of storyboards. The calculated K value was 50,333, indicating that if the total number of experiments exceeds 50,333, redundancy in the experiments would occur. To consider experiments with redundant file sizes of test cases in an actual environment, the above two test cases were run 75,000 time each to capture the storyboard file sizes from the victims' video packets for three seconds. Of the 75,000 experiments in each test case, 15,000 experiments with redundant file sizes were prepared and measured. In other words, 15,000 experiments with redundant file sizes were collected from 5,000 new videos on YouTube. Note that different videos were tested in each experiment.

The result for the first test case shows that the proposed VTIM performs with 99.94% accuracy and 99.93% precision when identifying victims' video titles. The result for the second test case shows that the VTIM identifies the victims' video titles with 99.88% accuracy and 99.86% precision. In addition, when the confidence interval of 11,772 videos processed by storyboards is calculated, the storyboard file sizes are between 302 and 161,686. As the number of videos processed by storyboards increases, the probability of the redundancy of the storyboard file sizes increases. The frequency of redundant storyboard file sizes was measured using the Cumulative Distribution Function (CDF) in Figure 8.

In Figure 8, the x-axis corresponds to storyboard file size, and the y-axis corresponds to the cumulative probability of the frequency of a certain storyboard file size. The slope of the curve is gentle.

Figure 9 shows a histogram of the number of file size frequencies. In other words, there are identical file sizes

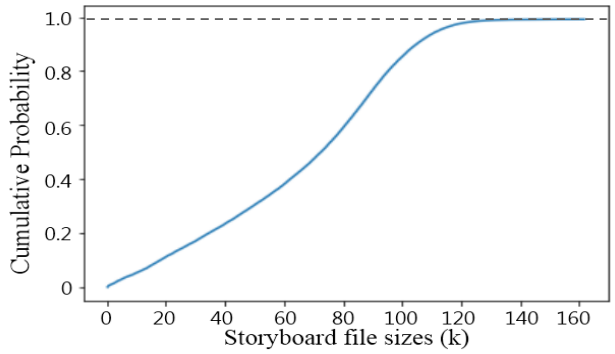

FIGURE 8. CDF from a distribution of storyboard file sizes.

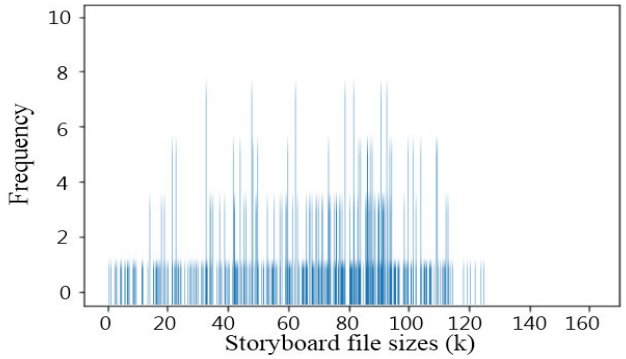

FIGURE 9. Histogram from a distribution of storyboard file sizes.

for different storyboards. Redundancy in the storyboard file sizes occurs because these file sizes are distributed between 0 and 160,000 in these test cases. However, to complement the redundancy of storyboard file sizes, the video titles are identified by both comparing the file sizes of each storyboard and analyzing the pixel values from the storyboards. Finally, the results for these test cases show that the proposed VTIM performs with 100% accuracy and precision when identifying victims' video titles. Since a set of open metadata such as storyboard file sizes and MPD segment sizes in each video is unique and the open metadata are transmitted regardless of the network situation, high accuracy and high precision can be observed.

D. STORYBOARD PROCESSING TIME

The Storyboard processing time is expressed as follows:

$$T_{Total}^{Storyboard} = T_{Preprocess}^{Storyboard} + T_{Run}^{Storyboard},$$

where $T_{Preprocess}^{Storyboard}$ refers to the total execution time for *Selector* and *Storyboard Downloader* and $T_{Run}^{Storyboard}$ is the sum of the video packet capture time and the video identification time of *Identifier*. Note that preprocessing needs to run only once for the repetitive identification of a video title. For an analysis of the relationship among $T_{Preprocess}^{Storyboard}$, $T_{Run}^{Storyboard}$, and the number of videos, Figure 10 representing Table 5 shows the average execution time as the number of videos

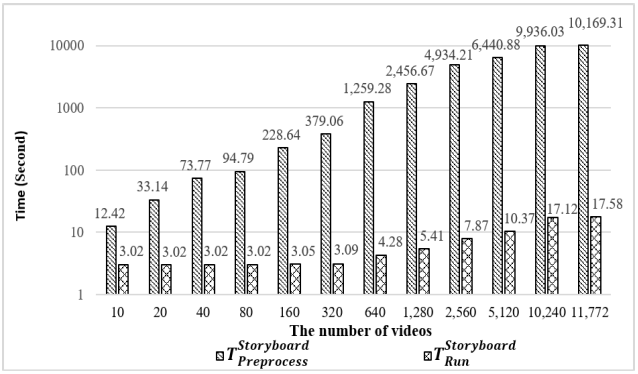

FIGURE 10. Storyboard preprocessing time and runtime.

doubles. In order to reliably collect enough segments that can be compared with preprocessed data, the video packet capture time was set to three seconds. The reason for increasing the number of videos is to check the processing time using thread parallelism in the proposed VTIM when processing multiple videos.

E. TEST CASES FOR THE ACCURACY AND PRECISION OF MPDs

To identify the video title, considerations for following three test cases are as follows: stable Internet speed, video resolution conversion, and simultaneous viewing of video by multiple users. To measure the accuracy and precision of using MPDs in aforementioned experimental situations, the following three test cases were generated:

1) Identify the victim's video title with captured video segment sizes using Algorithm 1 and calculate its accuracy and precision;
2) Identify the title of the video when resolution changes occur (e.g., 144p→240p, 240p→360p, 360p→480p, 480p→720p, 720p→1080p, and 1080p→144p) and calculate its accuracy and precision; and
3) Identify the title of the video and calculate its accuracy and precision when two victims play the same video at the same time.

In the first case, the accuracy and precision were measured when the Internet speed was stable. In the second case, a high rate of HTTP request packets (e.g., 300,000 packets/sec) was generated and sent to the victim's device to instigate network congestion, and the Internet speed is degraded. Subsequently, the accuracy and precision were measured. In the third case, the attacker receives multiple video packets from the victims, and two victims are watching the same video at the same time. The two victims' video packets are identified by the attacker. Note that the second and third cases were not considered in the existing methods [9]–[13].

In VTIM, the number of segments for each video resolution is measured during *MPD Preprocessing*. In order to measure the accuracy and precision correctly, repeated experiments should be considered. Repeated experiment means

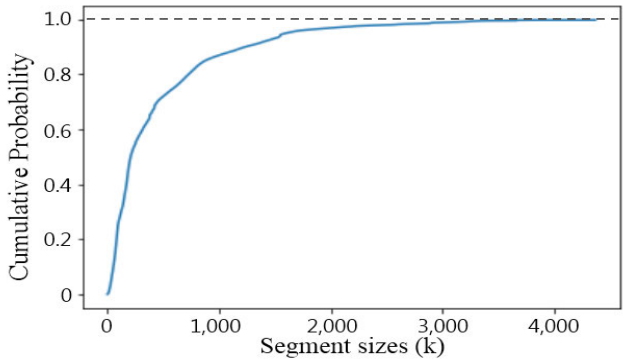

FIGURE 11. CDF from a distribution of segment sizes.

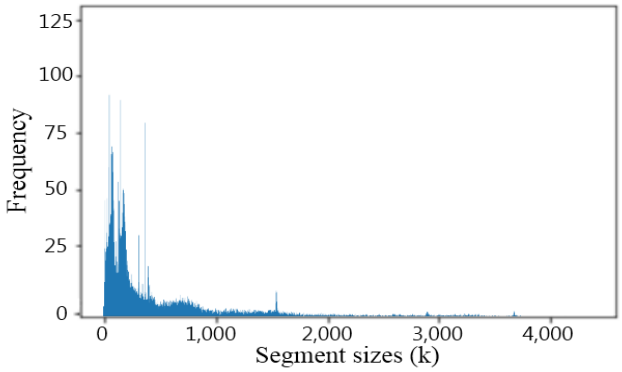

FIGURE 12. Histogram from a distribution of segment sizes.

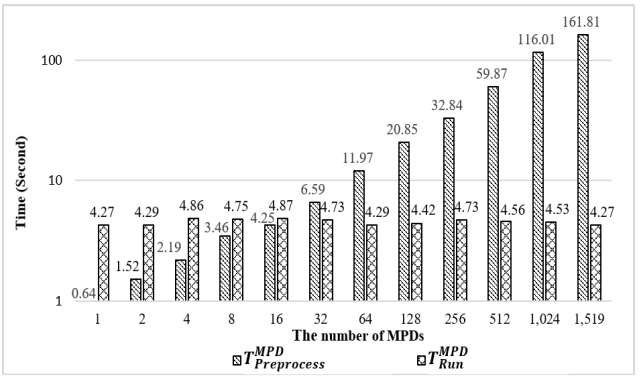

FIGURE 13. MPD preprocessing time and runtime.

that captured video segment sizes of the experiment should be equal to other experiments'. Repeated experiments can bias the result that degrades the measurement [50]. Thus, the total probability space is calculated according to Bayes' Theorem [49] by using the total number of video segments of each resolution that video supported and the non-redundant segment sizes. The calculated total probability space was 218,847, which means that if the total number of experiments exceeds 218,847 the redundancy of the experiments would occur. To consider experiments with redundant segment sizes of test cases in an actual environment, 250,000 experiments were then created for each of the three test cases. Of 250,000 experiments in each test case, 50,000 experiments with redundant segment sizes were prepared and measured. In other words, 50,000 experiments with redundant segment sizes were collected from 5,000 new videos on YouTube.

The result for the first test case shows that the proposed VTIM performs with 99.97% accuracy and 99.96% precision when identifying the victims' video titles. In addition, the result for the second test case shows that the VTIM identifies victims' video titles with 99.94% accuracy and 99.93% precision. Lastly, the result for the third test case shows that the VTIM has 99.97% accuracy and 99.96% precision when identifying victims' video titles. In addition, in the real-world, the segment sizes can overlap with each other as the number of videos processed by the MPDs increases because the number for segment sizes are between 1,136 and 4,365,324 in these test cases. Thus, the frequency of the occurrence of redundant segment sizes was analyzed as the CDF in Figure 11 and Figure 12.

In Figure 11, the x-axis corresponds to the segment size and the y-axis corresponds to the cumulative probability of the frequency of a certain segment size. Since the slope of the graph is rapid between 0 and 200,000 ranges, it can be observed that the segment sizes are distributed intensively. Figure 12 shows a histogram of the number of segment sizes frequencies. The segment sizes are concentrated between 0 and 1,000,000. These segment sizes have a resolution of less than 480p on average because, in the video set, low-quality videos such as 480p are supported more often than high-quality videos such as 1080p, and there are more low-quality videos than high-quality videos.

F. MPD PROCESSING TIME

The *MPD processing time* is expressed as follows:

$$T^{MPD}_{Total} = T^{MPD}_{Preprocess} + T^{MPD}_{Run},$$

where $T^{MPD}_{Preproces}$ is the sum of the execution time for *Selector*, *MPD Downloader*, *Analyzer*, *Calculator* and *Writer*, while T^{MPD}_{Run} is the sum of execution time for video packet capturing and identification. The preprocessing runs only once to identify video titles. For an analysis of the relationship among $T^{MPD}_{Preprocess}$, T^{MPD}_{Run}, and the number of MPDs, Figure 13 representing Table 6 shows their average execution time as the number of MPDs doubles. Again, the reason for increasing the number of MPDs is to check the concurrent processing performance of the proposed VTIM. Note that $T^{MPD}_{Preprocess}$ for VTIM, which is corresponding to the training time for machine learning techniques of paper [10], is approximately thirty times faster than the outcomes in the aforementioned study.

G. ACCURACY COMPARISON OF VTIM WITH EXISTING METHODS

VTIM was also assessed experimentally with the open video sets in earlier papers [10] and [13]. The results of these experiments are presented in Table 4, indicating that the accuracy of VTIM exceeds those of existing methods.

TABLE 4. Accuracy comparison of VTIM with existing methods.

	The number of videos	Accuracy	Accuracy of VTIM (storyboard)	Accuracy of VTIM (MPD)
Schuster *et al.* [10]	20	99.5%	100%	100%
Pan *et al.* [13]	10	97.57%	100%	100%

VII. COUNTERMEASURE

This section presents techniques that can be used to mitigates the proposed VTIM.

A. ANYONE CAN DOWNLOAD THE STORYBOARD AND MPD

Even if the video webpage's source codes have been obfuscated, the keywords *'https://i9.ytimg.com/sb/'* and *'PlayerStoryboardSpecRender'* pointing to storyboards are open. To prevent downloading of storyboards, YouTube service providers must obfuscate the storyboard's URLs in the video webpage's source codes or restrict user access by blocking storyboard download requests.

For MPDs, video segment downloads via BaseURL are already blocked except for the user that is watching the video. However, an attacker can request and download an MPD from the DASH server using MPD's URL because there is no user access restriction for downloading MPDs. So, it is also necessary to restrict user access so that an attacker cannot download MPDs.

B. LEAKAGE OF STORYBOARD FILE SIZES

The storyboard file sizes are open between the DASH server and the client. To prevent an attacker from knowing the storyboard file sizes, it must either be padded with random lengths or be encrypted. When padding or encryption is performed, the file sizes change and an attacker cannot identify the title of the video.

C. LEAKAGE OF SEGMENT NUMBER AND SEGMENT INFORMATION OF MPDs

The number of segments and the segment information referring to segment duration and range are leaked through the representation tag of an MPD. Therefore, it is necessary to insert a pseudo segment into an MPD so that the number and segment information of a video are not known, meaning that only the video player that plays the video can interpret the segment information. Therefore, segment durations and ranges must also be encrypted based on a specific numerical value of the video segment using a cipher key previously transmitted between the DASH client and the DASH server.

D. UPLOADING OF FRAGMENTED VIDEO CAUSES INFORMATION LEAKAGE

A video is encoded and fragmented during the process of uploading to a DASH server. When the video is downloaded, an attacker can decode the internally fragmented video to separate the fragments into actual video chunks and obtain their size. Therefore, to prevent the actual video segment sizes from being obtained by the proposed VTIM, YouTube needs to be modified to download the non-fragmented video.

VIII. EVALUATION ON THE COUNTERMEASURE

This section presents the evaluation of the encryption of storyboards and MPDs and the restriction of user access discussed in Section VII.

A. ENCRYPTION AND DECRYPTION OF STORYBOARD AND MPD

TLS is used for encrypted communication between the DASH server and the client on YouTube. TLS performs two crucial functions during sending and receiving encrypted video packets, which are *key exchange* and *video packet encryption and decryption*. The key exchange algorithm used on YouTube is based on Elliptic Curve Diffie-Hellman (ECDH) [51], which is an asymmetric encryption algorithm. When ECDH is used as the key exchange algorithm on YouTube, the primary elliptical curves are *secp256r1* curve [52] and *Curve25519* [53]. YouTube selects one of the two elliptical curves to create a cipher key. Using the cipher key, YouTube also applies Advanced Encryption Standard (AES)-128 [54] and AES-256 [54], which are symmetric encryption algorithms, to encrypt and decrypt video packets. Compared to AES-256, AES-128 has smaller cipher key and less number of rounds. Therefore, the execution time of encryption and decryption with AES-128 are relatively fast. The encryption algorithm used on YouTube was modified so that the DASH server and the client exchange cipher keys using ECDH, and the DASH server applies either AES-128 or AES-256 encryption to the exchanged cipher keys to transmit the storyboard and MPD to the DASH client.

Based on the analysis of communication during the playing of the videos, encryption and decryption processes for the storyboards and the MPDs are added to prevent leakages of the storyboards and MPDs. Figure 14 shows the encryption and decryption process of storyboards and MPDs in TLS consisting of the following five steps: (1) The DASH client generates its private key and computes the public key for an ECDH, which is the cipher key that will be used in AES-128 and AES-256. (2) The DASH client requests the storyboard and MPD with its public key to the DASH server. (3) The DASH server generates its private key and calculates its public key. Then, the DASH server calculates a shared secret and encrypts the storyboard and MPD using AES-128 or AES-256. (4) The DASH server sends the encrypted storyboard and MPD with its public key to the DASH client. (5) The DASH client calculates the shared secret using the public key of the DASH server, and uses it for AES-128 and AES-256 to decrypt the storyboard and MPD.

Since the shared secret calculated by the DASH client and the DASH server is not mutually exchanged, an attacker who does not obtain it through the MITM cannot access the open metadata of the video titles. In addition, when the storyboards and MPDs are encrypted by the DASH server, an attacker will not be able to access their open metadata to identify

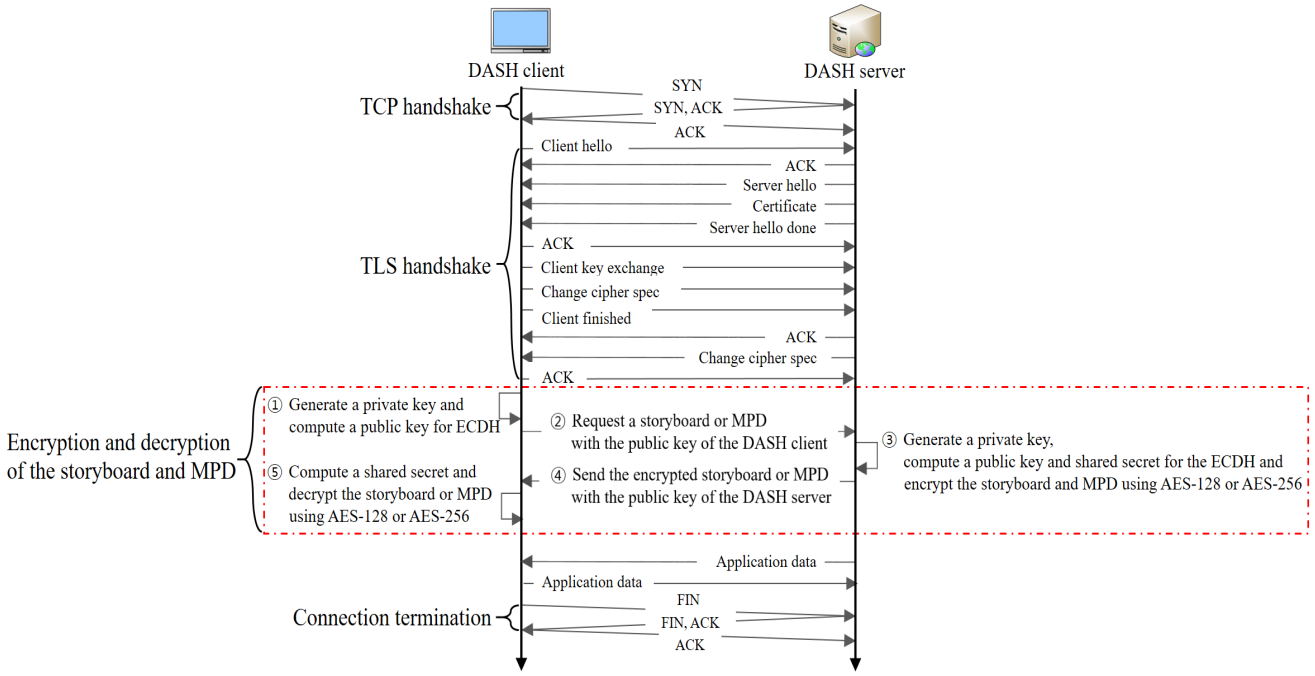

FIGURE 14. Encryption and decryption of the storyboard and MPD.

video titles by analyzing their file sizes. This is because padding is applied during the encryption process of AES-128 or AES-256 to protect the open metadata. In the case of an MPD, the MPD itself can be encrypted or open metadata of video in MPD can be encrypted with AES-128 or AES-256; therefore, it cannot be used by an attacker identify video titles. The open metadata of the MPD can be encrypted as follows:

Before encryption:

<SegmentURL media ="range/1214-26315"/>
<SegmentURL media ="sq/1/dur/3.05">

After AES-128 or AES-256 encryption:

<SegmentURL media ="range/enz9Ldda + . . . "/>
<SegmentURL media ="sq/1/dur/enMmda . . .">

As mentioned in Section III-F, 'range/' of SegmentURL media in MPD indicates the segment range, and 'dur/' indicates the segment duration. These metadata on segment range and duration are open. Therefore, the open metadata of an MPD must be parsed and encrypted using AES-128 and AES-256, or the MPD file itself can be encrypted. In order to defend against the proposed VTIM, our experiments are performed as follows:

- Measure the accuracy of VTIM when the storyboard and MPD are encrypted.
- Measure the processing time for ECDH used for the cipher key exchange.
- Measure the processing time to encrypt and decrypt the storyboard and MPD.

The reason for measuring the processing time to encrypt and decrypt a storyboard and MPD is to determine the delay time of the DASH server when the proposed method is applied. In this experiment, the DASH server is equipped with Intel Xeon E5-2630 v4 @ 2.20 GHz CPU, 64GB of main memory, which runs Ubuntu 16.04.6 LTS with Linux 4.15.0 (64-bit).

B. MEASURE THE ACCURACY OF VTIM WHEN THE STORYBOARD AND MPD ARE ENCRYPTED

VTIM cannot preprocess encrypted storyboard and MPD. Therefore, the accuracy becomes zero for both of them. The attacker can bypass the encryption of the storyboard and MPD by obtaining the cipher key and performing a brute-force attack. To defend against the former method, ECDH is used to exchange the cipher key between the DASH server and the client for AES-128 or AES-256 encryption of storyboard and MPD. If asymmetric ECDH is applied to the countermeasure, the attacker cannot obtain the cipher key [55]. The countermeasure for the latter is to use proven safety AES-128 or AES-256 to prevent the attacker from performing brute-force attacks. This is because brute-force attack against AES-128 and AES-256 are not practical due to the long processing time [56].

C. MEASURE THE PROCESSING TIME FOR ECDH USED FOR THE CIPHER KEY EXCHANGE

When the storyboard and MPD are encrypted, the DASH server and the client exchange a cipher key over ECDH to

TABLE 5. Processing time by the VTIM module in the storyboard test case.

The number of videos		10	20	40	80	160	320	640	1,280	2,560	5,120	10,240	11,772
The number of storyboards		61	108	203	402	787	1,603	3,462	6,697	11,852	20,403	44,404	50,334
$T^{Storyboard}_{Preprocess}$ (Second)	*Selector*	0.22	0.46	1.03	2.08	4.07	8.02	12.57	23.67	40.71	66.71	73.63	146.23
	Storyboard Downloader	12.20	32.58	72.74	92.71	224.57	371.04	1,246.71	2,433.00	4,893.50	6,374.17	9,862.40	10,023.08
	Total	12.42	33.14	73.77	94.79	228.64	379.06	1,259.28	2,456.67	4,934.21	6,440.88	9,936.03	10,169.31
$T^{Storyboard}_{Run}$ (Second)	Video packet capture time	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
	Video identification time	0.02	0.02	0.02	0.02	0.05	0.09	1.28	2.41	4.87	7.37	14.12	14.58
	Total	3.02	3.02	3.02	3.02	3.05	3.09	4.28	5.41	7.87	10.37	17.12	17.58
$T^{Storyboard}_{Total}$ (Second)		15.44	36.16	76.79	97.81	231.69	382.15	1,263.56	2,462.08	4,942.08	6,451.25	9,953.15	10,186.89

TABLE 6. Processing time by the VTIM module in the MPD test case.

The number of MPD		1	2	4	8	16	32	64	128	256	512	1,024	1,519
The number of videos		4	9	19	39	79	159	319	654	1,359	2,748	5,438	7,942
$T^{MPD}_{Preprocess}$ (Second)	*Selector*	0.04	0.12	0.11	0.11	0.32	0.89	1.79	4.89	5.79	9.89	20.08	25.89
	MPD Downloader	0.20	0.68	0.93	1.71	1.57	2.04	3.99	4.97	7.53	14.10	27.40	36.60
	Analyzer	0.03	0.05	0.11	0.41	0.90	1.64	3.43	6.53	11.53	20.30	38.17	53.32
	Calculator	0.04	0.05	0.12	0.30	0.53	1.03	1.76	3.34	6.63	13.82	27.88	42.57
	Writer	0.33	0.62	0.92	0.93	0.93	0.98	1.00	1.12	1.36	1.77	2.49	3.44
	Total	0.64	1.52	2.19	3.46	4.25	6.59	11.97	20.85	32.84	59.87	116.01	161.81
T^{MPD}_{Run} (Second)	Video packet capture time	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
	Video identification time	1.27	1.29	1.86	1.75	1.87	1.73	1.29	1.42	1.73	1.56	1.53	1.27
	Total	4.27	4.29	4.86	4.75	4.87	4.73	4.29	4.42	4.73	4.56	4.53	4.27
T^{MPD}_{Total} (Second)		4.91	5.81	7.05	8.21	9.12	11.32	16.26	25.27	37.57	64.43	120.54	166.08

TABLE 7. Processing time to generate cipher keys.

The number of videos			1	2	4	8	16	32	64	128	256	512	1,024
Processing time (Milisecond)	Selecting private keys and computing public keys	Curve25519	52.2	61.7	169.3	381.6	558.4	719.6	1,298.6	26,152.0	30,107.5	34,305.3	70,328.8
		secp256r1	53.6	65.9	176.4	392.7	561.0	867.9	1,299.6	26,083.1	31,126.1	35,589.2	74,538.6
	Computing shared secrets	Curve25519	2.9	3.3	6.1	9.7	16.6	30.3	47.0	106.3	175.2	292.1	512.0
		secp256r1	2.6	3.4	6.0	10.5	17.4	30.0	27.5	114.6	191.6	309.1	529.6

TABLE 8. Processing time to encrypt and decrypt the storyboard.

The number of videos			1	2	4	8	16	32	64	128	256	512	1,024
The number of storyboards			7	10	21	46	91	162	319	635	1,282	2,700	5,445
Encrypting and decrypting the storyboard itself (Milisecond)	AES-128	Encryption	5.1	6.6	12.8	26.4	44.5	75.1	136.2	228.1	421.4	882.1	1,880.0
		Decryption	4.8	6.0	12.2	25.7	44.0	73.6	137.9	243.0	442.9	1,028.6	1,987.6
		Total	9.9	12.6	25.0	52.1	88.5	148.7	274.1	471.1	864.3	1,910.7	3,867.6
	AES-256	Encryption	10.8	7.3	14.0	28.2	64.4	87.2	150.7	292.4	578.7	986.5	2,078.5
		Decryption	5.0	6.5	13.6	27.3	49.7	81.0	144.0	252.0	494.0	1,041.0	2,208.4
		Total	15.8	13.8	27.6	55.5	114.1	168.2	294.7	544.4	1,072.7	2,027.5	4,286.9

protect open metadata against ciphertext stealing [57]. The ECDH uses *secp256r1* curve and *Curve25519* to generate a key suitable for AES-128 and AES-256. As Table 7, the processing time required is measured for the DASH server and the client to generate private keys and compute public keys and shared secrets. Note that the processing time is proportional to the number of videos. Since the processing time of the two elliptic curves are relatively similar, both elliptic curves are appropriate for encrypting storyboards and MPDs.

D. MEASURE THE PROCESSING TIME TO ENCRYPT AND DECRYPT THE STORYBOARD AND MPD

The processing time for encryption and decryption was measured for storyboard and MPD. The encryption and decryption algorithms for storyboard and MPD use AES-128 and AES-256 by applying a cipher key generated from ECDH.

As Table 8, the processing time is measured to encrypt and decrypt storyboards as the number of videos doubled. The table also shows that AES-128 has faster encryption and decryption speed than AES-256. The faster encryption and decryption time reduces the response time of the DASH server resulting in better service for users watching videos.

As Table 9, the processing time is measured for encrypting and decrypting MPDs. This processing time consists of two parts: (1) parsing, encrypting, and decrypting open metadata of the MPD and (2) encrypting and decrypting the MPD itself. In the first part, an encryption area is determined by parsing the MPD. The encryption area is limited to segment duration and segment range in the MPD. This is because other metadata such as video encoding format should be preserved as plaintext because they can be referenced by the user before decrypting. The first part is slower than the second part because of the time required for parsing, encrypting and decrypting open metadata in MPD. The reason for the rapid increase in the processing time from 128 videos in Table 9 is that there are many high-quality movie and game videos with long playback time. In general, video with long playback time has large file size. Then, the relationship between file sizes of MPDs and processing time of encrypting and decrypting MPD should also be analyzed.

In summary, either *secp256r1* curve or *Curve25519* is suitable as the key exchange algorithm, and AES-128 is sufficient

TABLE 9. Processing time to encrypt and decrypt the MPD.

The number of videos			1	2	4	8	16	32	64	128	256	512	1,024
The number of MPDs			1	2	4	8	16	32	64	128	256	512	1,024
Parsing, encrypting and decrypting the open metadata of the MPD (Milisecond)	AES-128	Encryption	28.8	35	100.9	217.5	322.7	426.2	734.3	16,583	19,212.3	21,719.7	44,661.8
		Decryption	23.4	26.7	68.4	164.2	235.7	293.5	564.3	9,569.0	10,895.2	12,585.7	25,667.1
		Total	52.2	61.7	169.3	381.7	558.4	719.7	1,298.6	26,152.0	30,107.5	34,305.4	70,328.9
	AES-256	Encryption	29.8	35.9	102.1	221.3	323.1	488.0	726.7	16,643.2	19,350.9	21,840.1	46,655.5
		Decryption	23.8	29.9	74.3	171.4	237.8	379.96	572.9	9,440.0	11,775.2	13,749.1	27,883.1
		Total	53.6	65.8	176.4	392.7	560.9	867.96	1,299.6	26,083.2	31,126.1	35,589.2	74,538.6
Encrypting and decrypting the MPD itself (Milisecond)	AES-128	Encryption	1.6	1.9	3.4	5.2	8.9	15.9	23.2	54.3	87.8	147.8	252.5
		Decryption	1.3	1.4	2.7	4.6	7.7	14.3	23.9	52.0	87.4	144.3	259.5
		Total	2.9	3.3	6.1	9.8	16.6	30.2	47.1	106.3	175.2	292.1	512.0
	AES-256	Encryption	1.6	1.9	3.3	5.6	9.2	14.7	24.8	57.4	96.4	151.8	261.1
		Decryption	1.0	1.4	2.8	4.9	8.2	15.3	22.7	57.2	95.1	157.2	268.5
		Total	2.6	3.3	6.1	10.5	17.4	30.0	47.5	114.6	191.5	309.0	529.6

for encrypting and decrypting open metadata of storyboard and MPD.

IX. CONCLUSIONS AND FUTURE WORKS

This paper proposed VTIM as a method to identify the video titles of multiple victims on a network using the open metadata of the storyboard and MPD of MPEG-DASH. VTIM was implemented and tested via experiments using a video set collected from an actual environment. Our evaluation results show that VTIM has greater accuracy in shorter processing times compared to existing methods, even under real network conditions. To the best of our knowledge, this is the first work that exploits the vulnerability of open metadata such as storyboards and MPDs to identify video titles under real network conditions, i.e., network congestion.

As future work, VTIM will be expanded to a variety of video streaming applications such as Netflix and Vimeo as well as to experiment with more videos. In addition, an attack method that can identify video titles even when storyboards and MPDs are encrypted will be investigated. In addition, experiments with video title identification are planned for employee-monitoring solutions, QoE assessments in video streaming applications for network operators, and off-path attacks in which the attacker and victims are on different networks.

APPENDIX

In this appendix, we open URLs of video sets used in the experiments of VTIM. The video URLs are organized as follows; 1) video URLs used for preprocessing (including 13,291 videos) and 2) video URL used for testing (including 5,000 videos).

1) video URLs used for preprocessing (including 13,291 videos)
   - https://www.youtube.com/playlist?list=PLyFnTKgb8Q2WPqqF_WNoFw3wYVUPsc81i
   - https://www.youtube.com/playlist?list=PLyFnTKgb8Q2VeuDUz3pP-TnLhrgRikGra
   - https://www.youtube.com/playlist?list=PLyFnTKgb8Q2XNK2Nwk5yiIag2TjITmxTI
   - https://www.youtube.com/playlist?list=PLYHgBO4XGofkGlkHyo6sMxdWTlwEox6YO
2) video URL used for testing (including 5,000 videos)
   - https://www.youtube.com/playlist?list=PLYHgBO4XGofmUs5Qr1H_uBUJeSv3nt7ux

REFERENCES

[1] Cisco. (2019). *Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html#_Toc532256803

[2] Sandvine. (2018). *The Global Internet Phenomena Report*. [Online]. Available: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf

[3] YouTube. (2019). *YouTube Live Streaming API Document: Delivering Live YouTube Content Via DASH*. [Online]. Available: https://developers.google.com/youtube/v3/live/guides/encoding-with-dash

[4] S. Lederer, "Why YouTube Netflix use MPEG-DASH HTML5," BITMOVIN, San Francisco, CA, USA, Tech. Rep., 2015.

[5] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the Internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, Apr. 2011.

[6] T. Stockhammer, "Dynamic adaptive streaming over HTTP–: Standards and design principles," in *Proc. ACM Conf. Multimedia Syst.*, 2011, pp. 133–144.

[7] Google. (2019). *Google Transparency Report HTTPS Encryption Web*. [Online]. Available: https://transparencyreport.google.com/https/overview

[8] R. Stewart, S. Long, D. Gallatin, A. Gutarin, and E. Livengood, "Protecting Netflix viewing privacy at scale," NETFLIX, Los Gatos, CA, USA, Tech. Rep., 2016.

[9] R. Dubin, A. Dvir, O. Pele, and O. Hadar, "I know what you saw last minute—Encrypted HTTP adaptive video streaming title classification," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 12, pp. 3039–3049, Dec. 2017.

[10] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *Proc. 26th USENIX Secur. Symp. (USENIX Security)*, 2017, pp. 1357–1374.

[11] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, "I know why you went to the clinic: Risks and realization of HTTPS traffic analysis," in *Privacy Enhancing Technologies*. Cham, Switzerland: Springer, 2014.

[12] Y. Xie, H. Deng, L. Peng, and Z. Chen, "Accurate identification of Internet video traffic using byte code distribution features," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2018, pp. 46–58.

[13] W. Pan and G. Cheng, "QoE assessment of encrypted YouTube adaptive streaming for energy saving in smart cities," *IEEE Access*, vol. 6, pp. 25142–25156, 2018.

[14] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in Web applications: A reality today, a challenge tomorrow," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 191–206.

[15] G. Aceto and A. Pescapé, "Internet censorship detection: A survey," *Comput. Netw.*, vol. 83, pp. 381–421, Jun. 2015.

[16] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time QoE detection for encrypted YouTube traffic," in *Proc. 10th ACM Multimedia Syst. Conf.*, Jun. 2019, pp. 48–59.

[17] J. Gu, J. Wang, Z. Yu, and K. Shen, "Walls have ears: Traffic-based side-channel attack in video streaming," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 1538–1546.

[18] D. Pope, G. Urgun, and A. Wheat, "Visual seeking for iPlayer," BBC, London, U.K., Tech. Rep., 2019.

[19] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine Learning*. Norwell, MA, USA: Kluwer, 1995.

[20] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.

[21] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1725–1732.

[22] M. Scott, *Applied Logistic Regression Analysis*. Newbury Park, CA, USA: Sage, 2001.

[23] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Ann. Math. Statist.*, vol. 41, no. 1, pp. 164–171, 1970.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[25] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993.

[26] J. Li, S. Zhang, Y. Xuan, and Y. Sun, "Identifying skype traffic by random forest," in *Proc. Int. Conf. Wireless Commun., Netw. Mobile Comput.*, Sep. 2007, pp. 2841–2844.

[27] S. Mahadevan and R. Rebba, "Validation of reliability computational models using Bayes networks," *Rel. Eng. Syst. Saf.*, vol. 87, no. 2, pp. 223–232, 2005.

[28] R. E. Schapire, "Explaining AdaBoost," in *Empirical Inference*. Berlin, Germany: Springer, 2013.

[29] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the HTTPS protocol," in *IEEE Secur. Privacy*, vol. 7, no. 1, pp. 78–81, Jan./Feb. 2009.

[30] Progress Software Corporation. (2019). *Telerik Fiddler: The Free Web Debugging Proxy for Any Browser, System or Platform*. [Online]. Available: https://www.telerik.com/fiddler

[31] P. Skolka, C.-A. Staicu, and M. Pradel, "Anything to hide? Studying minified and obfuscated code in the Web," in *Proc. World Wide Web Conf. (WWW)*, 2019, pp. 1735–1746.

[32] A. Mondal, S. Sengupta, B. R. Reddy, M. J. V. Koundinya, C. Govindarajan, P. De, N. Ganguly, and S. Chakraborty, "Candid with YouTube: Adaptive streaming behavior and implications on data consumption," in *Proc. Netw. Oper. Syst. Support Digit. Audio Video*, 2017, pp. 19–24.

[33] T. Kupka, P. Halvorsen, and C. Griwodz, "Performance of on-off traffic stemming from live adaptive segmented HTTP video streaming," in *Proc. 37th Annu. IEEE Conf. Local Comput. Netw.*, Oct. 2012, pp. 401–409.

[34] The Wireshark Foundation. (2019). *Wireshark: The Network Protocol Analyzer*. [Online]. Available: https://www.wireshark.org/

[35] T. De Pessemier, L. Martens, and W. Joseph, "Dynamic optimization of the quality of experience during mobile video watching," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast.*, Jun. 2015, pp. 1–6.

[36] A. Dogtiev. (2019). *Business of Apps: YouTube Revenue and Usage Statistics*. [Online]. Available: http://www.businessofapps.com/data/youtube-statistics/#1

[37] The MITRE Corporation. (2019). *CVE: Common Vulnerabilities Exposures*. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0561

[38] The MITRE Corporation. (2014). *CVE: Common Vulnerabilities Exposures*. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2781

[39] N. Wells, "BusyBox: A Swiss army knife for linux," *Linux J.*, vol. 2000, no. 78es, p. 10, 2000.

[40] Apache. (2019). *Selenium: Web Browser Automation*. [Online]. Available: https://www.seleniumhq.org/

[41] A. K. Boduch, J. Chaffer, and K. Swedberg, *Learning jQuery 3*, 5th ed. Birmingham, U.K.: Packt Publishing, 2017.

[42] Axiomatic Systems. (2019). *Bento 4 MPEG-DASH Toolset*. [Online]. Available: https://www.bento4.com/developers/dash/

[43] Terms & Privacy. (2019). *TestMy.net: Web-Based Internet Speed Test Tools*. [Online]. Available: https://testmy.net/

[44] S. Simpkins, "Tools: Network packet tools and page performance," in *Troubleshooting SharePoint: The Complete Guide to Tools, Best Practices, PowerShell One-Liners, and Scripts*. Berkeley, CA, USA: Apress, 2017.

[45] The Apache software Foundation. (2019). *JMeter: Network Stress Testing Tool*. [Online]. Available: https://jmeter.apache.org/

[46] Google. (2019). *System Requirements of YouTube*. [Online]. Available: https://support.google.com/youtube/answer/78358?hl=en

[47] YouTube-DL Developers. (2020). *YouTube-DL: Video Downloader*. [Online]. Available: http://ytdl-org.github.io/youtube-dl/about.html

[48] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data* (Data-Centric Systems and Applications). New York, NY, USA: Springer-Verlag, 2006.

[49] T. Leonard and J. S. J. Hsu, *Bayesian Methods: An Analysis for Statisticians and Interdisciplinary Researchers*. Cambridge, U.K.: Cambridge Univ. Press, 2001.

[50] G. H. Golub and U. V. Matt, "Generalized cross-validation for large-scale problems," *J. Comput. Graph. Statist.*, vol. 6, no. 1, pp. 1–34, 1997.

[51] Certicom Research. (2000). *SEC1: Elliptic Curve Cryptography*. [Online]. Available: https://www.secg.org/SEC1-Ver-1.0.pdf

[52] Certicom Research. (2010). *SEC2: Recommended Elliptic Curve Domain Parameters*. [Online]. Available: https://www.secg.org/sec2-v2.pdf

[53] D. J. Bernstein, "Curve25519: New Diffie–Hellman Speed Records," in *Proc. Public Key Cryptogr.*, 2006, pp. 207–228.

[54] *Announcing the Advanced Encryption Standard (AES)*, Federal Information Processing Standards (NIST FIPS)-197, 2001.

[55] Y. Yusfrizal, A. Meizar, H. Kurniawan, and F. Agustin, "Key management using combination of Diffie–Hellman key exchange with AES encryption," in *Proc. IEEE Int. Conf. Cyber IT Service Manage.*, Aug. 2018, pp. 1–6.

[56] A. A. Hasib and A. A. M. M. Haque, "A comparative study of the performance and security issues of AES and RSA cryptography," in *Proc. Int. Conf. Converg. Hybrid Inf. Technol.*, 2008, pp. 505–510.

[57] P. Rogaway, M. Wooding, and H. Zhang, "The security of ciphertext stealing," in *Proc. Fast Softw. Encryption*, 2012, pp. 180–195.

JUHYUNG SONG received the B.S. degree from the Korea University of Technology and Education (KOREATECH), in 2018, and the M.S. degree from the Graduate School of Information Security, Korea Advanced Institute of Science and Technology (KAIST), in 2020. His research interests include side-channel analysis of multimedia services and applications.

SUYEONG LEE received the B.S. degree in computer science and engineering from Sogang University, in 2019. He is currently pursuing the M.S. degree with the Korea Advanced Institute of Science and Technology (KAIST). His research interests include the Internet, mobile computing, and network security.

BAEKJUN KIM received the B.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2019, where he is currently pursuing the M.S. degree. His research interests include cryptography and network security.

SOONUK SEOL received the B.S. degree from the Korea University of Technology and Education (KOREATECH), in 1998, and the M.S. and Ph.D. degrees in information and communication engineering from KAIST, in 2000 and 2004, respectively. From 2004 to 2012, he worked as a Senior Researcher with KT. He is currently an Associate Professor with the School of Electrical, Electronics, and Communication Engineering, KOREATECH. His research interests include the Internet of Things (IoT), wireless networking, QoS, and software testing.

BEN LEE received the B.E. degree in electrical engineering from the Department of Electrical Engineering, State University of New York (SUNY), Stony Brook, in 1984, and the Ph.D. degree in computer engineering from the Department Electrical and Computer Engineering, Pennsylvania State University, in 1991. His research interests include multimedia streaming, wireless networks, embedded systems, computer architecture, multithreading and thread-level speculation, and parallel and distributed systems. He has been on the program committees and organizing committee for numerous international conferences, including the 2005-2012 IEEE Workshop on Pervasive Wireless Networking (PWN) and the IEEE International Conference on Pervasive Computing and Communications (PerCom). He is currently on the Steering Committee for CCNC. He received the Loyd Carter Award for Outstanding and Inspirational Teaching, in 1994, the Alumni Professor Award for Outstanding Contribution to the College and the University from the OSU College of Engineering, in 2005, and the HKN Innovation Teaching Award from Eta Kappa Nu, School of Electrical Engineering and Computer Science, in 2008. He was a TPC Chair and a General Chair for the 15th and 17th Annual IEEE Consumer Communications & Networking Conference (CCNC 2018 & 2020).

MYUNGCHUL KIM (Member, IEEE) received the B.A. degree in electronics engineering from Ajou University, in 1982, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 1984, and the Ph.D. degree in computer science from The University of British Columbia, Vancouver, BC, Canada, in 1993. From 1984 to 1997, he was the Managing Director of the Korea Telecom Research and Development Group, where he was in charge of research and development of protocol and QoS testing on ATM/B-ISDN, IN, PCS, and Internet. He is currently a Professor with the Faculty of the School of Computing, KAIST. He has published over 150 conference proceedings, book chapters, and journal articles in the areas of computer networks, wireless mobile networks, protocol engineering, and network security. His research interests include the Internet, protocol engineering, mobile computing, and information security. He has served as a member of program committees for numerous numbers of conferences. He has also served as the Chair for the IWTCS'97 and the FORTE'01.