CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Evaluating VPN Defenses Against Video Fingerprinting: A Case Study of DAITA

Master's thesis in Computer science and engineering

Wilhelm Henriksson, Max Liman

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Evaluating VPN Defenses Against Video Fingerprinting: A Case Study of DAITA

Wilhelm Henriksson, Max Liman

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Evaluating VPN Defenses Against Video Fingerprinting: A Case Study of DAITA
Wilhelm Henriksson, Max Liman

Supervisor: Romaric Duvignau, CSE
Advisor: Odd Stranne, Mullvad. Tobias Pulls, Karlstad University
Examiner: Risat Pathan, CSE

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Evaluating VPN Defenses Against Video Fingerprinting: A Case Study of DAITA
Wilhelm Henriksson, Max Liman
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Virtual Private Networks aim to obfuscate and hide the user's internet traffic by encrypting it and rerouting it through the service providers own servers. This is no longer enough since by examining the frequency and size of the packets sent over the network, an attacker can with the help of a database of fingerprinted videos from different streaming services correctly identify which video is being streamed, despite a VPN connection being active. The VPN company Mullvad has developed a feature called DAITA for their service aiming to solve this issue through various methods.

In this paper, a type of fingerprinting attack on video data that was developed by the authors of "Endangered Privacy: Large-Scale Monitoring of Video Streaming Services" is used. The attack exploits the information leak of modern video streaming protocols which are patterns that are referred to as "bursts". These bursts are unique and can be mapped to fingerprints of specific videos.

This paper aims to examine how the DAITA feature and other defensive measures affect the given video fingerprinting attack's performance and evaluate the data derived from the tests of the attack on these different protections.

We find that the attack works against the regular use of VPN as expected, and that the padding of packets is not what breaks the attack. The attack will still work so long as the traffic is translatable into bursts and similar enough to the fingerprints. When a VPN connection with DAITA enabled has a defense active that is able to disrupt the traffic pattern enough to confuse the bursts from the fingeprints, the attack fails. However, we show that DAITA does not always break the attack.

Keywords: VPN, Fingerprinting attack, Video Fingerprinting, Cybersecurity, Network Traffic

Acknowledgements

We would like to thank our supervisor at Mullvad, Odd Stranne, for supplying us with everything we needed in order to perform our tests,

Martin Björklund, for aiding us in our understanding of the attack featured in this thesis, giving us answers to all the many questions we had regarding it,

Tobias Pulls, for giving us a good insight into the inner workings of the protection feature and methods tested and described in this thesis,

And last but definitely not least we would like to thank Romaric Duvignau, for being our supervisor, answering all our many questions, no matter their area or significance, and giving us endless feedback allowing us to finish this report as a thesis to be proud of.

Wilhelm Henriksson, Max Liman, Gothenburg, 2026-04-01

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Related Work . . . 2
1.1.1 Maybenot . . . 2
1.1.2 DAITA . . . 2
1.1.3 “Endangered Privacy” . . . 3
1.2 Goals and Challenges . . . 3
1.3 Limitations . . . 4
1.4 Thesis Structure . . . 4

2 Background **5**
2.1 Virtual Private Networks . . . 5
2.1.1 VPN Encryption . . . 6
2.1.2 WireGuard . . . 6
2.2 Fingerprinting . . . 7
2.2.1 Website Fingerprinting . . . 7
2.2.2 Robust Fingerprinting . . . 8
2.2.3 Video Fingerprinting . . . 8
2.3 Adaptive Bitrate Streaming . . . 9
2.4 DAITA and its Defenses . . . 9
2.4.1 Maybenot . . . 10
2.4.2 MTU Padding . . . 11
2.5 Packet Capture . . . 12
2.5.1 Wireshark . . . 12
2.5.2 Structure of a PCAPNG file . . . 13
2.6 The “Endangered Privacy” Attack . . . 13
2.6.1 Results of “Endangered Privacy” . . . 14
2.6.2 The Two Attack Models . . . 14
2.6.3 The Evaluation Program . . . 15

3 Methods **17**
3.1 General Approach . . . 17
3.2 Implementation . . . 17

3.2.1 Setting Up . . . 17
3.2.2 Automating the Webdriver Script . . . 18
3.2.3 Network traffic capture method . . . 18
3.2.4 Capture Environments . . . 18
3.2.5 Customizing Scripts . . . 19
3.2.6 MTU Padding . . . 19
3.2.7 Maybenot Simulator . . . 20
3.3 Evaluation . . . 21
3.3.1 Evaluation Metrics . . . 21
3.3.2 Experimentation . . . 21

4 Results **23**
4.1 Metrics . . . 23
4.2 Results of the first testing . . . 24
4.2.1 Discussion of the First Iteration . . . 25
4.3 Results of the second testing . . . 26
4.3.1 Discussion of the Second Iteration . . . 30
4.4 Bursts and bitrate . . . 32
4.5 Capturing environments effect on results . . . 32

5 Conclusion **35**

Bibliography **37**

List of Figures

2.1 Visualization of connection to an internet service without a VPN . . . 6
2.2 Visualization of connection to an internet service with Wireguard as VPN protocol . . . 7
2.3 Bursty behavior of ABR streaming [6]. . . . 9
2.4 Example input string . . . 10
2.5 Visualisation of non-padded ethernet frame . . . 12
2.6 Visualisation of padded ethernet frame . . . 12

3.1 Example machine string . . . 20
3.2 `The SimulatorArgs struct` . . . 20
3.3 `Reconstructing packets logic in pseudo` . . . 21

4.1 The attack's confidence value for an unknown video using regular HTTPS . . . 25
4.2 The attack's confidence value for an identified video using regular HTTPS . . . 25
4.3 Results of the attack on different types of protection . . . 27
4.4 Comparison of the file sizes in the second dataset . . . 28
4.5 The attack's confidence value for an unknown video using regular HTTPS in the second iteration . . . 29
4.6 The attack's confidence value for an identified video using regular HTTPS in the second iteration . . . 29
4.7 The attack's confidence value for an unknown video using DAITA in the second iteration . . . 30
4.8 The attack's confidence value for an identified video using DAITA in the second iteration . . . 30
4.9 How burstsharks output correlates to the bitrate in HTTPS, VPN and DAITA with rolling window of size 20 . . . 33

List of Tables

4.1 Performances with a threshold of 2.2 24
4.2 No defense file sizes of first dataset 24
4.3 DAITA file sizes of first dataset 24
4.4 MTU padding file sizes of first dataset 24
4.5 VPN file sizes of first dataset 24
4.6 No defense file sizes of second dataset 27
4.7 DAITA file sizes of second dataset 27
4.8 MTU padding file sizes of second dataset 27
4.9 VPN file sizes of second dataset 28
4.10 Performances with a threshold of 1.5 28
4.11 Performances with a threshold of 2.2 (same videos as in Table above) 28
4.12 Average number of bursts and burst size from burstshark output . . . 32

1

Introduction

As VPN (Virtual Private Network) usage grows more popular, so does the quality and quantity of attacks against different VPNs. A type of attack that is particularly efficient against VPNs is fingerprinting, which exploits identifiable characteristics in networking traffic and/or application behavior [1], [2], [3], [4]. OpenVPN, which is the most widely used protocol in commercial VPN services such as NordVPN, has shown to be vulnerable to fingerprinting [1]. Xue et al. showed that 85% of the traffic that the OpenVPN protocol generated was identifiable to adversaries in control of the network [1]. The anonymous network Tor has also been susceptible to fingerprinting attacks [5], suggesting that it is possible for parties that oversee and control network traffic such as ISP (Internet Service Providers) to block certain communication.

In recent years, different identification mechanisms have been used as a tool to analyze anonymous network and VPN communications [2], [6], [7]. As a result, users whose intention it is to be anonymous on the internet, are at a risk of deanonymization as private information can be extracted from the traffic analysis. There exists different approaches to enacting the traffic analysis. Both AI models trained over a multitude of recorded communication streams as well as sophisticated algorithms computed over incredibly large datasets can be used. Mullvad, a leading VPN provider, has developed DAITA (Defense Against AI-guided Traffic Analysis) as a defensive measure against fingerprinting attacks enabled by traffic analysis.

This thesis aims to study Mullvad's DAITA feature using various tools with regards to its ability to protect user's streamed video data, compare Mullvad's DAITA feature with other methods of network traffic protection in regard to different metrics as well as analysing how well these different methods of protection hold.

Our work is based largely on the work done by Martin Björklund and Romaric Duvignau in "Endangered Privacy: Large-Scale Monitoring of Video Streaming Services" [6].

1.1 Related Work

This section describes related works where an explanation is necessary for the understanding of what is covered.

1.1.1 Maybenot

Maybenot is an open-source framework developed to aid defenders against traffic analysis [8]. The framework is a continuation and improvement upon the Tor Circuit Padding framework developed by Kadianakis et al. [9]. It was designed to facilitate various systems, protocols and applications. Its main goal is to confuse adversaries engaged in traffic analysis based attacks, such as fingerprinting.

The authors of Maybenot argue that developing a framework rather than a specific application will yield more coverage and versatility for defensive strategies against traffic analysis [8]. A framework can be applied to different protocols or services to, in the short term, offer more protection.

Maybenot's main defensive mechanisms are probabilistic state machines which deploy padding and blocking strategies depending on certain events in the system [8]. In other words, Maybenot modifies network traffic with the intent of making different types of traffic indistinguishable from one another.

1.1.2 DAITA

DAITA is the main focus of this thesis. It builds upon both the Maybenot and the Wireguard frameworks in order to obfuscate the VPN encrypted data to make it harder for observers to recognize patterns in the network traffic.

DAITA does this in three different ways:

1. **Constant Packet Sizes** Since packet sizes can reveal a lot of information about the type of the network traffic, DAITA makes all the packets the same size to mitigate this issue.

2. **Random Background Traffic** DAITA generates random noise by throwing in so called dummy packets containing no meaningful data to make observers confused by what packets are real and which are not.

3. **Data Pattern Distortion** When visiting a website or doing anything that causes considerable network traffic, DAITA will send cover traffic between the user and the VPN server. This will then disturb pattern recognition, making it harder for observers to distinguish a pattern in the traffic [10].

AI, which excels at recognizing patterns, is being used as a tool to enable fingerprinting and extract information from encrypted communication. This is a significant privacy concern which is why defensive measures, such as DAITA by Mullvad, has been developed against fingerprinting attacks [11].

Currently Mullvad's DAITA is only tailored towards protecting web traffic against traffic analysis, but there is still a great need for this to be implemented within other areas such as video streaming and instant messaging [6], [12]. DAITA is also only known to be effective against website fingerprinting. It has not been studied if DAITA is also effective against other types of network traffic analysis based attacks or if other types of data than website traffic are protected from fingerprinting.

1.1.3 "Endangered Privacy"

In "Endangered Privacy: Large-Scale Monitoring of Video Streaming Services" by Björklund and Duvignau, three of the largest video streaming services in Sweden (Max, Amazon and SVT Play) are exposed to being vulnerable to fingerprinting [6].

The authors created their own tools "burstshark" and "karl". Burstshark, which is a wrapper to tshark, creates "bursts" of recorded packet captures, either in real-time or from PCAPNG (Packet Capture Next Generation) files. karl is a data extraction tool which also builds a fingerprint based on the collected data. An evaluation script with sophisticated identification logic revolving around data created by karl and querying these based on recorded bursts. More technical information about their work can be found in their paper [6].

1.2 Goals and Challenges

The aim of this thesis is to analyze techniques to tailor video fingerprinting attacks on VPN traffic, focusing primarily on Mullvad's DAITA feature which main emphasis lies on protection against website fingerprinting.

This includes things such as:

- Further examining, testing and measuring the effectiveness of Mullvad's DAITA service against video fingerprinting.
- Comparing basic and tailored attacks on key metrics such as bandwidth usage, recognition accuracy and response times to identify strengths, weaknesses and trade-offs.

Some of the challenges regarding this project are be the following:

- Rewriting the attack and data gathering programs to work for our intended purposes.
- Accurately collecting data to in a realistic way simulate a real world scenario.
- Finding documentation on DAITA as it is a relatively new concept.

The thesis aims to answer these research questions:

- Is DAITA an effective mitigation strategy against video fingerprinting attacks?
- Is it possible to optimize DAITA for defense against video fingerprinting attacks?

1.3 Limitations

In order to define a reasonable scope for this thesis, some limitations needs to be set. The thesis will compare Mullvad's DAITA feature in regards to a limited number of metrics and perform the same tests using a limited number of other methods of protection in order to achieve this reasonable scope.

Since the dataset of videos used to test the performance of the attack on is fairly large, a limit is set to only include videos from SVT, since that decreases the necessary memory size quite a bit in addition to not needing a subscription service to stream the videos since SVT is accessible for free.

The size of the dataset is set to 100 videos as well. Having a larger dataset would be beneficial but since the videos are streamed in real time limiting the dataset to 100 videos saves necessary time.

1.4 Thesis Structure

Chapter 1 covers the introduction to the thesis, other projects or papers related to this thesis, the goals and challenges of the thesis work and finally the limitations stowed upon this work.

Chapter 2 describes the necessary background information about VPNs, fingerprinting and video streaming data which enable the main attack of the thesis.

Chapter 3 covers the implementation steps as well as the different experiments conducted in this work.

Chapter 4 presents the findings by the experiments.

Chapter 5 concludes the thesis by discussing the findings and calls on further research.

2

Background

While encrypted end-to-end communication has grown in popularity and level of security, traffic analysis remains an efficient tool for adversaries [8]. It is therefore of great worth that defensive measures against traffic analysis based attacks, such as fingerprinting, are rigorously developed and continuously improved to adapt to evolving attack strategies, ensuring robust protection of user privacy and secure communication [6], [12].

There are multiple problems regarding the vulnerability to fingerprinting attacks [3], [8], [9]. For example, an attacker could shut down users' network traffic when they stream videos with certain political views even if the traffic is encrypted and the users are using VPNs [3]. Another example is an attacker setting up phishing schemes after figuring out what a user is interested in, in order to try and scam the user [13].

One major issue when implementing defensive strategies against traffic analysis is computational overhead. Secure tunnels or VPNs operating in user-space require a lot of computational power and generate overhead due to the need of copying packets from user-space to kernel-space [14]. Even more overhead occur when taking defensive measures such as padding [8], which is why DAITA was developed using the Maybenot framework on the WireGuard secure tunnel. Maybenot and WireGuard are covered in more depth in sections 2.1 and 2.2.

2.1 Virtual Private Networks

VPNs, or Virtual Private Networks, when introduced were used as a tool to access the Internet from a remote location or network. The intention of the VPN was to extend private networks like LANs (Local Area Networks) to networks in different geographic locations. However, information shared over the internet is in clear-text and anyone listening to communication can observe the contents of the communication. Therefore it was necessary to create a way for embedding packets from various different protocols and applications into IP-packets and transmitting them safely over the internet. Although technology for both encryption and encapsulation of packets in IP-packets already existed, the VPN technology combines these into a secure way of communicating on the otherwise insecure internet.

The two main parties of the VPN technology are the client and the server. The VPN

service establishes a connection with a remote server that encrypts communication between the VPN client and server, i.e. a secure tunnel [1], [15]. As such, the contents of the VPN user's communication are encrypted and appears as if sent from a completely different location.

Recently, the use of VPNs has evolved into anonymizing and securing user data by masking user's IP-addresses and hiding communication from ISPs [15]. Many users of VPNs are currently using them to communicate and use the internet anonymously. Although this was not the original intention of the development of VPN technology, VPNs play a major role in enabling or preserving privacy on the internet.

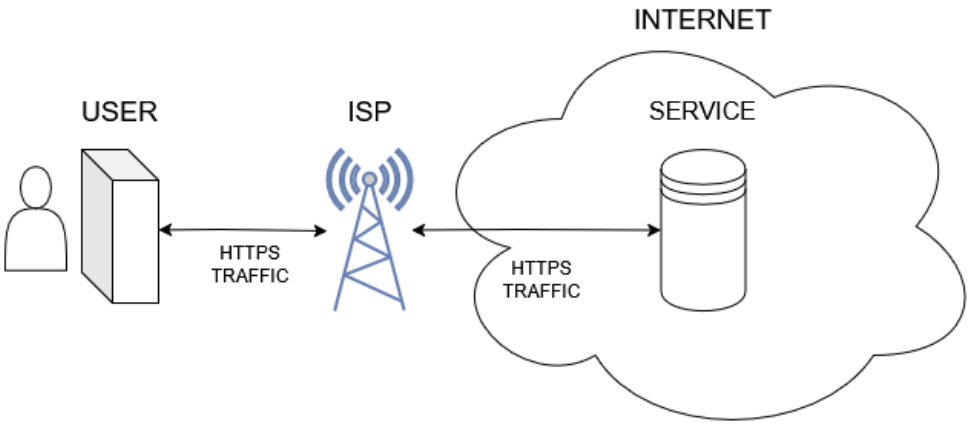


Figure 2.1: Visualization of connection to an internet service without a VPN

2.1.1 VPN Encryption

The main functionality of VPNs that makes them secure and private is the use of different encryption technologies. Different VPNs use different secure protocols and in turn different encryption techniques but the result is more or less the same. The commonalities shared between the VPN's are the two parties communicating with each other agreeing on the encryption scheme, encrypting the packets in accordance with the encryption scheme, the process of encapsulating the encrypted packets in IP packets (tunneling) and finally decrypting the packets after they have reached their destination.

2.1.2 WireGuard

WireGuard is a secure network tunnel developed by Jason A. Donenfield which acts as a kernel virtual network interface for Linux machines [14]. WireGuard functions at the network layer which makes it less computationally intensive and more user-friendly than other protocols such as OpenVPN or IPsec which operate at user-space [14]. The design choices for WireGuard, such as single round trip key exchanges, shorter pre-shared keys as well as encapsulation of packets in UDP through authenticated-encryption all contribute to WireGuard generating less overhead and better performance while still being secure [14].

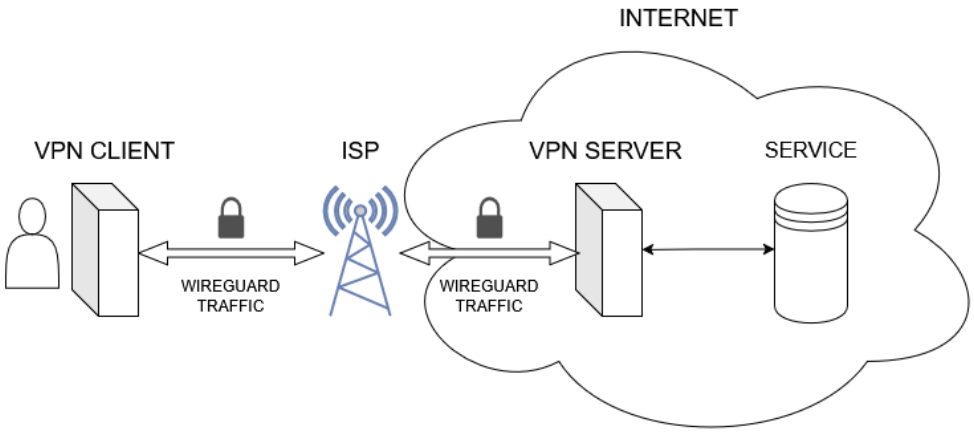


Figure 2.2: Visualization of connection to an internet service with Wireguard as VPN protocol

WireGuard defends itself against denial-of-service while also providing authentication and encryption through IP-binding cookies. The cookie functionality is an improvement upon the mechanisms of IKEv2 and DTLS [14]. The combination of all aforementioned implementations lead to a system that does not require any resource allocation for received packets.

2.2 Fingerprinting

Using a VPN to connect to internet applications only hides the IP-address of the user and the content of the traffic [1], [11], [12]. The structure of the traffic, i.e. the patterns of the packets exchanged between the user and the service, are still fully visible to ISPs (or anyone who has access to the ISP) [1], [6], [11], [12]. Different websites generate different patterns of network packets and are therefore discernible from one another. Useful information can consequently be extracted from encrypted communication [2], [7], [8], [9], [16].

Fingerprinting attacks leverage traits inherent to encrypted traffic to discern patterns within VPN communications [2], [5]. For instance, attackers can analyze packet size, timing Windows or other metadata to infer specific applications, software or websites and/or identify users behind the VPN communication.

2.2.1 Website Fingerprinting

Website fingerprinting is the type of fingerprinting attack which analyzes and exploits patterns in website traffic and communications. More specifically, it is a traffic analysis based attack which tries to identify and classify packet traces and infer which websites they belong to. This traffic-analysis-based approach is more akin to a side-channel attack as the contents of the packets are encrypted and the revealing information is instead embedded in the patterns or packet traces. Website fingerprinting has been shown as an effective way to reveal information and identify

both anonymous users and what websites they visit.

Website fingerprinting is effective against both encrypted traffic through HTTPS and anonymity networks such as Tor [17]. The result of this is a real privacy concern as both the anonymity of the user and the websites they visit are compromised.

2.2.2 Robust Fingerprinting

Shen et al. proposed "Robust Fingerprinting" as a type of website fingerprinting attack tailored specifically against fingerprinting defenses of anonymity networks such as Tor [17]. The authors developed a robust traffic representation method that generates a TAM (Traffic Aggregation Matrix). The TAM collects features of the traffic traces from Tor traffic and is used to train a CNN (Convolutional Neural Network) classifier which learns multiple traffic features revealed by various defense mechanisms.

The "robustness" of the attack comes from the "robust" traffic traces, or features, that are not easily disrupted by the defense mechanisms. These features include packet direction, time and number and they are aggregated in the TAM. The authors showed that it is possible for the TAM to capture these robust features and train a CNN on them to accurately fingerprint Tor traffic [17]. The result of Robust Fingerprinting is a fingerprinting attack which is 8.9 percent more accurate on Tor traffic with defensive measures taken than other state-of-the-art fingerprinting attacks.

2.2.3 Video Fingerprinting

In contrast to website fingerprinting where packet traces regarding website traffic are analyzed, video fingerprinting analyzes the stream of video data downloaded by a client from a streaming service. In this work and in previous works [6], the video fingerprinting attack uses the fact that video streaming services induce a specific network behavior. Because of the protocols used by modern streaming services it is possible to first create fingerprints of certain videos offered by streaming services by collecting their so called manifest files. Then by analyzing the packet exchange between the client and the service one can observe what is called a "bursty" network behavior that is inherent to the streaming protocol.

In this project we will use the tshark wrapper "burstshark" developed by the authors of [6] to create the bursty representations of the video streams. After acquiring and building a dataset of the information in the manifest files and creating the bursts, it is possible to use an identification algorithm to find the video in the dataset that corresponds to the burst. This kind of attack can be described as a side-channel attack that identifies encrypted video data through analyzing the patterns in communication.

2.3 Adaptive Bitrate Streaming

Modern streaming services are facilitated by adaptive bitrate streaming protocols [6]. Earlier protocols had trouble with some crucial network traffic functionality, for example firewall and NAT (Network Address Translation) traversal, inefficient caching and failure to acclimatize to changes in the network environments. ABR (Adaptive Bitrate) protocols offer solutions to these problems through different features. These features include segmenting source videos and encoding the segments with different bitrates. Users of streaming services request the segments according to their bandwidth and the rate at which they can process buffers. When a user requests a stream, they are asked to download a manifest file of the requested video. The manifest file holds information necessary for the stream regarding the availability of encoding and the locality of the segments. CDNs (Content Delivery Networks) go hand-in-hand with ABR protocols since the manifest files are constant and are thus stored at the edge servers of the CDN.

The two main ABR protocols used by streaming services are DASH (Dynamic Adaptive Streaming over HTTP) and HTTP live streaming (HLS). DASH, developed by Moving Pictures Expert Group (MPEG), is currently the only standardized ABR protocol and is used extensively by streaming services. HLS was developed by Apple and is used exclusively on Apple devices.

ABR information leakage: After the initial manifest file request, the client fills up its empty buffer with segments from representations with lower bandwidth to begin playback the quickest. After the buffer is full the client's rate at which it requests segments depends on the time it takes to process the segments in the buffer. Thus, the client has reached the steady state of ABR streaming. Furthermore, the client calculates and predicts its bandwidth in real-time to determine from which representation's segments it should download. This alternating behavior in the network communications (bursts) between the client and the server can be observed. The bursts imply the requests and downloads of the segments. The size of the segments are determined by their content since the bitrate at which the segments are encoded are varying.

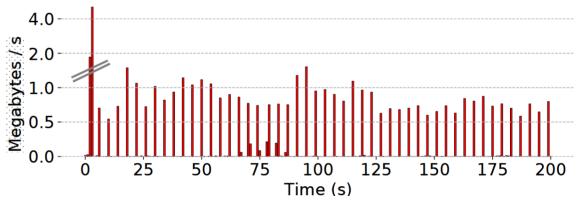


Figure 2.3: Bursty behavior of ABR streaming [6].

2.4 DAITA and its Defenses

This section covers the components of DAITA. DAITA consists of defenses implemented in Maybenot and MTU padding. When establishing a VPN connection

with DAITA applied, a defense is randomly chosen from a database of thousands of defenses.

2.4.1 Maybenot

Maybenot is a framework which enables cybersecurity developers to build applications that defend against network traffic analysis attacks [8], [18]. It is both a runtime environment that runs defenses and a framework where defenses are implemented as probabilistic state machines known as Maybenot machines. Maybenot operates both at the client and the server. The actions that the machines can take are either padding or blocking and they can also self-transition within a state or transition to other states. Padding is the addition of dummy or padding packets to the traffic which aims to disrupt patterns and obfuscate the real traffic. Blocking is an artificial delay added to the traffic that confuses packet timings and aims to disrupt timing based patterns.

Together these actions make the traffic appear uniform, noisy and statistically uncorrelated to real traffic. The actions or state transitions are induced by trigger events from the network traffic. Examples of such events are `NormalSent` and `PaddingSent` which trigger based on non-padding and padding packets sent from the perspective of the system running Maybenot. `NormalRecv` and `PaddingRecv` are events that trigger on non-padding and padding packets received.

Maybenot exists in two Rust-crates: `maybenot` and `maybenot-simulator`. `maybenot` contains all the code, the library of the maybenot framework, as it is. All of the code that runs the maybenot framework as well as the code that runs the maybenot machines is found in this crate.

The `maybenot-simulator` crate on the contrary is a library which simulates the maybenot state machines being used on communication. The main functions of the crate is `parse_trace`() and `sim`(). `parse_trace`() takes as input an undefended network trace as a string in this format: “time(ns), direction(sent/recieved)\n” where time is in nanoseconds since start of the communication and direction is in the perspective of the client. An example of an input trace in fig 2.4: The output of `parse_trace`()

```
"0, s
19714282,r
183976147,s
243699564,r
1696037773,s
2047985926,s
2055955094,r
9401039609,s
9401094589,s
9420892765,r"
```

Figure 2.4: Example input string

is a vector of events in a data structure called a `SimQueue`. It consists of each event in

the trace parsed for input into the `sim()` function. `maybenot-simulator` is mainly used for simulating or testing defenses in Maybenot.

Two examples of existing defenses implemented in Maybenot by the authors of [18] are FRONT and RegulaTor. The defenses are implemented to evaluate the performance of Maybenot on known fingerprinting attacks.

FRONT: Maybenot leverages the framework FRONT to implement the padding on the network traffic in order to obfuscate packet size information [8], [18]. FRONT samples different time values from a Rayleigh distribution before the start of the download and adds padding at those samples [19]. FRONT is initialized with a padding count for the client and the server which serves as the amount of padding packets available to each party.

In Maybenot, FRONT is implemented using a state machine [8], [18]. The machine is made up of one `START` state and a series of `PADDING` states. Its attributes are the padding count, maximum padding window and the number of `PADDING` states. The Maybenot padding is executed sequentially at each `PADDING` state and each state will do padding differently from padding done previously. This results in variation in padding and irregularities in the patterns of packet sizes, effectively making it harder for network analysis.

RegulaTor: RegulaTor is a defense mechanism for Tor. It has been observed that Tor traffic generates high concentrations or bursts of cells within short periods of time followed by low cell volume [20]. These bursts are commonly occurring during the start of downloads and intensity of traffic decreases over time. RegulaTor regularizes Tor traffic by emulating and modifying the patterns [20]. It does so through sending traffic at a controllable rate (different from the naturally occurring rate of Tor communication) regulated by a decay function. Additionally, if the cells that are waiting for transit exceed some amount, the rate is reset to that of the initial rate and starts decaying again which introduces irregularities and noise in the network patterns and creating difficulty for network analysis.

The RegulaTor defense is implemented in Maybenot as two machine designs, one for relays and one for clients.

2.4.2 MTU Padding

Padding network packets to MTU (Maximum Transmission Unit) has been a widely used defensive strategy for many years [21]. Its main purpose is to prevent adversaries from identifying patterns in packet sizes that occur in encrypted communications.

MTU padding works by appending nonsense data, often zeroes, to the payload of a packet. A normal network packet of say 100 bytes is then padded with zeroes to the MTU size of 1500 bytes. Applying MTU padding on all packets in communications leads to all packets having a constant size of 1500 bytes.

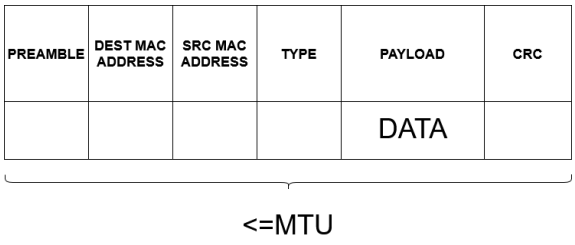


Figure 2.5: Visualisation of non-padded ethernet frame

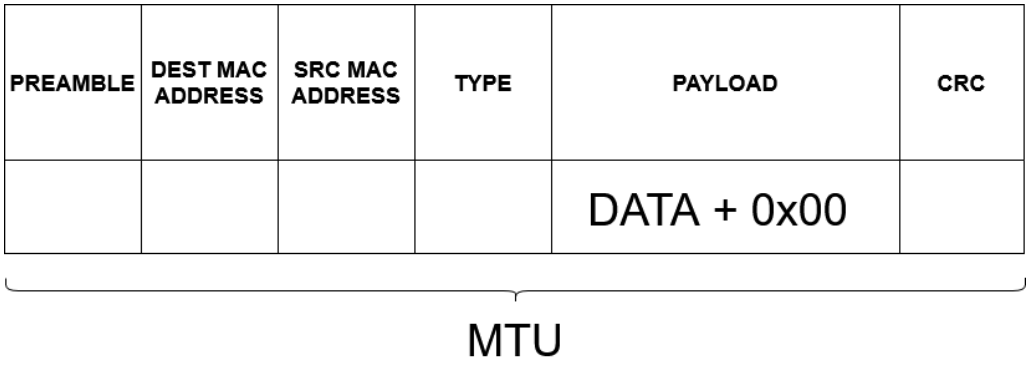


Figure 2.6: Visualisation of padded ethernet frame

2.5 Packet Capture

Packet capture or packet sniffing is a powerful tool used to analyze network traffic. It is the process of intercepting and logging network traffic, or packets, that passes through some network interface. In network traffic analysis, packet sniffing is especially powerful since information such as timing and packet size sill is very useful even when the contents of the packets are encrypted.

Wireshark is a popular tool used for packet capture [22]. In this thesis, the CLI (Command-Line Interface) version TShark is used.

2.5.1 Wireshark

Wireshark is a free and open-source packet capturing tool that intercepts and analyzes packets [22]. It has many different features that can customize the user's packet analysis. One can for example filter the packets shown based on protocol, IP-addresses (sent, received, or both), or other characteristics. These different features make Wireshark an effective tool for traffic analysis. It can be used for analyzing security, troubleshooting, verifying, debugging and learning network protocols and behaviors.

TShark: TShark is the CLI version of Wireshark [23]. It has all the packet capture functionality of Wireshark and can use or produce data in the same formats. TShark can capture packets from a live network or read packets from an already existing packet capture file. The main advantage of using TShark is the fact that it is a

CLI program which makes it possible to use TShark in other scripts or programs operated from the terminal. As such, it is possible to integrate packet capture into shell scripts or other programs, streamlining the process of traffic analysis.

2.5.2 Structure of a PCAPNG file

The files created by capturing the network traffic with either wireshark or tshark have the PCAPNG file type. The structure and content of one of these files will depend on the type of traffic captured and whether or not the traffic passes through a VPN tunnel.

In the case of the traffic being sent through a VPN tunnel, all in wireshark visible packets will have the same format where the destination address being either the clients or the VPN servers and the source address being the other with the protocol being WireGuard.

In the second case of not using any protection, all the packets coming in to and out from the client will be exposed in the PCAPNG file.

2.6 The "Endangered Privacy" Attack

The attack can be run from two models: "strong" or "weak" attacker. The strong attacker assumes the role of some party which has control of the target's network flow, for example an ISP. The weak attacker considers two scenarios: an adversary on the network path but with VPN activated and an adversary which is eavesdropping on WiFi communication. The video fingerprinting attack works by recording the packets transmitted between a VPN client and a streaming service. The attacker has beforehand created a large dataset consisting of thousands of possible videos that are offered by the streaming service as well as generated fingerprints based on the videos.

The sniffed packets are analyzed by burstshark and converted into bursts which, based on what is known about video streaming network patterns [6], correspond to some fingerprint collected. To find this corresponding fingerprint, the authors used a k-d tree algorithm to query the datasets for the fingerprint corresponding to the burst. The result of this is a prediction that the network bursts are representative of some video in the dataset. Their system produced predictions with great accuracy [6].

It is assumed that the target is not engaging actively in other online activities on their system, for example visiting other websites or using some application with internet access in the foreground. Rather, the assumption is that the target is only actively watching the video. However, other small applications and services running passively in the background are viewed as negligible network activity and should be manageable by the attack. Other protocols are not disabled either. The aim is to create a real-world scenario in which background noise in the network is present.

For both models, it is assumed that the streams are over a stable internet connection.

That is, the connection is stable enough that the target downloads segments from the same representation during around one minute of each stream. This would refer to the steady-state behavior of MPEG-DASH and HLS under stable network conditions, where the client downloads the highest possible representation. This work does not consider the scenario where streaming occurs under unstable network conditions where frequent representation switching can disturb the identification logic.

2.6.1 Results of "Endangered Privacy"

The authors of the paper on which this thesis is based achieved both interesting and alarming results with the attack. With the strong attack model and on regular HTTPS traffic 299 out of 300 videos streamed on Amazon Prime are identified and 1 unknown, 300 out of 300 videos streamed on Max are identified and 299 out of 300 videos streamed on SVT Play are identified, resulting in 898 of 900 (99.8%) videos being identified [6].

As for the weak attack model over VPN traffic, 279 out of 280 Amazon Prime videos are identified with 1 unknown, 280 out of 280 Max videos are identified and 276 out of 280 SVT Play videos are identified with 4 unknowns, yielding a total of 835 out of 840 (99.4%) videos identified. Finally for the weak attack model with VPN traffic over WiFi, Amazon Prime videos are considerably harder to identify with 165 out of 280 (58.9%) being identifed and 115 unknowns. Max and SVT Play had similar results to previous experiments with 279 (99.6%) and 275 (98.2%) identified and 1 and 5 unknowns respectively. Conclusively 99.5% of streamed videos are identified within 10 minutes of streaming with an average time of identification of within 2 minutes of starting the capture tool [6].

Additionally the authors note that the results they achieved are and can be obtained under noisier conditions with resistance to false positives but that more research is needed to validate effectiveness in other conditions as well as if the adversary is fully ignorant of the streaming service.

2.6.2 The Two Attack Models

Here we delve further into the different attack models described in the original paper.

The Strong Attack Model The strong attack model is the model which represents the adversary as a network overseer in control of network traffic such as an ISP. Although it is possible to leverage information from other layers, for example from the transport layer, this work focuses only on the IP-layer and the ABR streaming leak.

The Weak Attack Model In the paper, the weak attack model assumes two scenarios: the VPN and the WiFi eavesdropper scenario [6]. In this work, the weak model is used for engaging in the attack on VPN traffic. As stated in the paper, the weak model simulates an adversary on the network path but the target engages in communications through a VPN tunnel [6]. The WiFi eavesdropper scenario will

not be explored in this work.

2.6.3 The Evaluation Program

The main part of the fingerprinting attack is the evaluation program. The program takes these different sets of data as parameters:

- **The ground-truth data:**
  A JSON (JavaScript Object Notation) document containing identifiable information about each individual video. This is used to find corresponding videos to the bursts.
- **Representations:**
  Data pertained to each video such as actual video data, video encoding and bitrate as identification. Used for the identification algorithm.
- **The packet capture files (PCAPNGs):**
  The intercepted packets during streaming of videos. To be passed to burstshark for burst generation. Could also be evaluated for bandwidth usage.
- **Bursts**
  The network traffic bursts as generated by burstshark. Used for the identification algorithm to find corresponding videos to the bursts.

The way the program works is by first checking the parameters given to it through the call of the script. For the scope of this thesis, when the script is called to evaluate data gathered using no protection, the model parameter will be set to strong compared to when the data is protected by a VPN when the parameter will be set to weak. In both cases the method parameter will be set to network and the recompute flag will also be set.

When the model flag being set to weak means that the attacker doesn't know which service is being used and has to query all trees, and when it is set to strong it means that the attacker can query each tree individually.

As for the method flag, it is permanently set to network since the other possible parameter is WiFi which is used for when the network traffic has been captured by a separate device on the network and not directly on the client streaming the videos as in our case.

The recompute flag means that the script should create new bursts out of the data provided. This is done since there are no given bursts of data that isn't the data already provided.

When the script is run it will set relevant parameters for the later function calls.

3

Methods

This section will cover our method of creating the data for evaluation as well as the process of evaluating the vulnerability to exploitation. The attack was run in three scenarios; without VPN, with VPN and with both VPN and DAITA enabled.

3.1 General Approach

This section will cover our general approach to the project.

- **Experimentation:** a testing environment was set up that allowed for continuous watching of videos and capturing of packets. The video fingerprinting attack was tested and its effectiveness compared against different network defenses.
- **Modification:** The code base of the original attack was modified to function for the experiments of this work. Data collection, fingerprinting and packet capture automation was also modified.
- **Evaluation of implementation:** The different network defenses were evaluated against the attack.

3.2 Implementation

This section will cover the steps that are made to implement the fingerprinting attack. Modifications made to the original attack are also included.

3.2.1 Setting Up

The first step was to recreate the results from the work "Endangered privacy" that is the basis of this thesis. Simply by downloading and running the datasets and software from the "Endangered Privacy" artifact appendix, it was possible to recreate the results. However, due to hardware limitations, it was only possible to run the attack on videos from SVT. Packets were also captured from SVT videos that are not part of the given datasets, due to the SVT dataset being slightly outdated, to test if the attack is up-to-date.

3.2.2 Automating the Webdriver Script

Since the SVT datasets are outdated, a way to create up-to-date datasets, create ground-truth fingerprints and capture packets of available videos was necessary. This was solved by using the given webdriver script which will, given a URL (Uniform Resource Locator) and an amount of minutes in the form of an integer, open google chrome and play the video from the URL for the given minutes. It was too tedious to repeatedly perform this so an automatization was made. The webdriver script calls another given script which gets the URLs for all the available videos to stream on SVT. Then the webdriver script runs on a randomly selected URL from this list, runs tshark in the background for the same amount of time given the webdriver script to play the video for and at the end writes the ground truth JSON file and saves the PCAPNG file. When randomly selecting a video from the list, the script takes into consideration whether or not a video has already been watched as to not make duplicates. One issue was that when this script was first run without the use of any protection for the network traffic and then later run to gather network traffic that used some form of protection, different videos were watched. To make accurate comparisons between identification time and file sizes it is advantageous to compare the same video with different protections which is why the methodology was changed for the second iteration of experiments.

3.2.3 Network traffic capture method

The method that was chosen for gathering the data to be tested was to use the webdriver automation script to watch 100 videos while gathering all the necessary PCAPNG files and ground truth JSON files which the evaluation script needed.

To achieve this, the script was run on a laptop using the Ubuntu Linux OS which is thought to be the better option since the other available machine runs MacOS. MacOS has a different encryption standard when it comes to video streaming compared to Windows and Linux and could potentially disrupt the evaluation script, skewing the results.

While the script was running and network traffic was being captured, as few background processes were running as possible as to minimize the amount of noise packets captured that could potentially skew the results of the evaluation script. Of course there were processes running in the background regardless, however these can be seen as more reality simulating processes since the captured video streaming network traffic will not be isolated in a real world scenario.

3.2.4 Capture Environments

The environments in which the packets will be captured changed through out the process of gathering the different datasets. When capturing packets for the datasets with unprotected network traffic, some of the capturing was done at Chalmers campus while some were captured at a home network. The reason for this was simply that capturing in one environment was inefficient since it will reduce the amount of available capturing time significantly. Only capturing at Chalmers campus or

only capturing in a home network amounted to about half of the captured data per day. However, there was not an issue with the datasets containing the VPN protected network traffic and DAITA protected network traffic, since during the time of capture there was no need to be at the Chalmers campus. The traffic was allowed to be captured within a single environment allowing for consistency within the datasets, although the network traffic for the two datasets are captured on two different machines, both running Linux.

3.2.5 Customizing Scripts

The given evaluation script was not able to be run on the captured traffic "out of the box". There were changes that needed to be made in order for the script to be runnable on the newly captured datasets. When running the evaluation script on the captured network traffic it would eventually crash because the source address of the client machine sometimes changed from file to file. Due to the fact that the majority of the capturing of the network traffic is conducted on Chalmers campus, which provides us with a new IP address each day, the IP addresses changed. The reason for the crashes were that the evaluation script grouped every packet in a network traffic capture file based on the source IP of the client in order to make sure that there aren't any packets from another client or machine on the network being taken into account.

The approach first taken was to write a script that changes the source IP of all the network capture files to be the same, in order for the evaluation script to work. However, that approach would end up not working as intended. It is not clear as to why that approach did not work. It could be due to the script not changing the IP-addresses correctly or some functionality in the attack program that ignored the changes. The other solution was to remove the part of the script that divides the packets into groups based on the source IP, since the network traffic capturing is conducted on the same machine that is streaming the videos, there were no packets from other clients or machines that can be captured and can potentially disrupt the attack.

3.2.6 MTU Padding

During the project, there was an idea to mitigate this particular attack by implementing a maximum transmission unit (MTU) padding on the captured packets of the video streams. The MTU is the standard maximum size of packets transported on the network layer. Standard network communication has a maximum MTU of 1500 bytes, which also in most cases is the MTU. In the case of this experiment 1500 bytes is chosen.

MTU padding was implemented in Python using the Scapy library which is a packet manipulation tool. The Python program was written in such a way that it takes a PCAPNG file as input and returns a PCAPNG file as output, where every packet in that packet capture file that has a size smaller than the MTU is padded with zeroes to the MTU. No artificial delay is applied to the packets and as such all

the timestamps of the padded packets are the same as the raw packets. The MTU padded PCAPNG files are then used as input to the evaluation script to assess if this is a valid defense mechanism against the attack.

3.2.7 Maybenot Simulator

A Rust executable was implemented to run the Maybenot simulator in order to experiment how to reduce the DAITA overhead. The Rust program was built from the `maybenot-simulator` crate's example usage program. In order to run the simulator roughly as a representation of DAITA, some preparations were made. To convert the captured packets of the HTTPS traffic (undefended traffic) into a trace that is parsable for Maybenot, a small python program was written. The program takes as input a PCAPNG-file and outputs a string as shown in Figure 2.4. Two Maybenot machines, one for the client and one for the server, are selected from a large database of machines. The machines are strings which look like this:

```
"02eNp1ibEJAEAIA5Nf7B3N0v1cSESwEL0m5A6YvBqSgP7WeXfM5UoBW7ICYg=="
```

Figure 3.1: Example machine string

and are used as input into `sim_advanced()` along with the input string and a field named `SimulatorArgs`. `SimulatorArgs` is a struct which contains configurable arguments for `sim_advanced()`.

```
pub struct SimulatorArgs {

    pub network: Network,
    pub max_trace_length: usize,
    pub max_sim_iterations: usize,
    pub continue_after_all_normal_packets_processed: bool,
    pub only_client_events: bool,
    pub only_network_activity: bool,
    pub max_padding_frac_client: f64,
    pub max_blocking_frac_client: f64,
    pub max_padding_frac_server: f64,
    pub max_blocking_frac_server: f64,
    pub insecure_rng_seed: Option<u64>,
    pub client_integration: Option<Integration>,
    pub server_integration: Option<Integration>,
}
```

Figure 3.2: `The SimulatorArgs struct`

`max_padding_frac_client` and `max_padding_frac_server` decide how much padding (dummy packets) is applied to the traffic and as such are the arguments that are tweaked to experiment with the overhead.

Embedded into the rust program are some functions that produce PCAPNG files based on the output of `sim_advanced()`. This is needed since the attack is operated

on PCAPNG files. All of the packet data from the PCAPNG that the input trace is based on is saved and written to an output PCAPNG file using a Rust crate. The output vector containing `SimEvent` is iterated through and writing actions are taken conditionally. Basically, the output packet contains 1500 bytes of 'AA' characters if the `SimEvent` is a padding event or the saved packet data from the input. A rough pseudo code representation of the logic is shown in Figure 3.3.

```
if event.contains_padding():
    output_packet = [0xAA * 1500]
else:
    output_packet = input_packet
```

Figure 3.3: `Reconstructing packets logic in pseudo`

3.3 Evaluation

This section contains the methodology of the different steps taken to evaluate the effectiveness of the attack.

3.3.1 Evaluation Metrics

The attack is evaluated according to the following metrics:

- **Accuracy**: how many of the evaluated files that were identified compared to the amount unidentified or misidentified.
- **File Size**: how large are the average files with the different types of protection compared to their unprotected counterparts.
- **Time to Identify**: the average time to correctly identify a video in the dataset the attack is run on.

3.3.2 Experimentation

Regular HTTPS Traffic: The webdriver script was run beforehand to randomly select 100 videos from SVT. The script streams every video for 10 minutes and collects all of the necessary data for evaluation. The data includes the ground-truth for every video as well as the captured packets from the recorded network traffic during streaming. As the data is collected, the "bursts" are generated from the captured packets. Then, the evaluation program was run with the collected data as parameters to the program to create the prediction probabilities and collecting the results of the attack on the dataset in a text file.

VPN Traffic: After the webdriver script had collected all necessary data, the evaluation program was run with the new data as parameters. Firstly without an active VPN connection and secondly with an active connection to evaluate the performance of the VPN.

DAITA: To evaluate DAITAs mitigation, the attack was run on the 100 new videos that are collected on a machine that routes traffic through the Mullvad VPN with DAITA enabled.

MTU Padding: To evaluate the mitigation effect of the MTU padding, the python script was run on all of the 100 PCAPNG files from the regular HTTPS traffic capture. No delay was applied on the traffic. Then, the evaluation script was run on the 100 padded PCAPNG files.

Maybenot: To evaluate the mitigation effect of Maybenot and experimenting with different amounts of overhead the Maybenot simulator was run on 15 PCAPNG files. Each run was using different values of `max_padding_frac_client` and `max_padding_frac_server` which determine how many dummy/padding packets are inserted into the traffic. The values ranged from 0.05 to 0.95 with increments of 0.1. Two Maybenot machines, one for the client side and one for the server side, are arbitrarily selected for the simulation from a large list provided to us.

During the capture of these videos in the first iteration, the script was written in such a way that different videos are streamed between each of the datasets, HTTPS, VPN and DAITA.

Second Iteration: There were some changes made in the second iteration of the network traffic collection compared to the first iteration. One of the changes was that the same set of videos were captured for the datasets without any protection, with VPN protection and with DAITA protection. Regarding the capturing of the second DAITA dataset, it was run in one continuous capture which means that the same connection was held to the VPN server using DAITA. This ensured that only one defense was kept as the used DAITA defense which made a fair comparison during later analysis of the increased size for the network capture files for the different datasets.

Another change was that parallel capture was not used. In the first iteration of the data collection two laptops are simultaneously running the automated webdriver script in parallel in order to shorten the amount of time spent on the data collection. As realized before the second iteration of the data collection, parallel capture could have an impact on the PCAPNG files. All other devices connected to the local network in the capturing environment are disconnected, to ensure that there would be as little noise in the network capture files as possible.

4

Results

In this chapter the results of the different test performed are presented through different graphs and tables. The content is divided into different subsections adhering to the testing of the first iteration of the dataset, the testing of the second iteration of the dataset and maybenot.

4.1 Metrics

The performance of the attack is determined by the correctness of the prediction outputted by the evaluation program. A correct prediction is given at any time within the 10 minutes of watching each video when the confidence score exceeds the confidence threshold θ. Inversely, an inconclusive prediction is given when the score does not exceed θ within the 10 minutes of watching. Three different outputs are possible:

1. **Identified:** Minimum of one correct prediction and zero wrong predictions within the full 10 minutes watching session.
2. **Unknown:** At no point during the 10 minutes does the confidence value exceed the threshold.
3. **Misidentified:** At least one incorrect prediction within the watching session.

In the results, there are no videos that are less than 10 minutes long since the video capturing scripts that are used discard videos with length less than 10 minutes.

4.2 Results of the first testing

The following tables and graph will show results of the attack on the first dataset.

Protection	Identified	Unknown	Misidentified
HTTPS	73 (73%)	27 (27%)	0 (0%)
VPN	93 (93%)	7 (7%)	0 (0%)
DAITA	0 (0%)	100 (100%)	0 (0%)
Maybenot	0 (0%)	69 (69%)	0 (0%)
MTU	73 (73%)	27 (27%)	0 (0%)

Table 4.1: Performances with a threshold of 2.2

File	File Size (MB)
Smallest File	87.8
Average File	207.1
Largest File	472.7

Table 4.2: No defense file sizes of first dataset

File	File Size (MB)
Smallest File	204.1
Average File	415.0
Largest File	821.6

Table 4.3: DAITA file sizes of first dataset

File	File Size (MB)
Smallest File	98.3
Average File	230.22
Largest File	504.3

Table 4.4: MTU padding file sizes of first dataset

File	File Size (MB)
Smallest File	120.2
Average File	230.0
Largest File	545.4

Table 4.5: VPN file sizes of first dataset

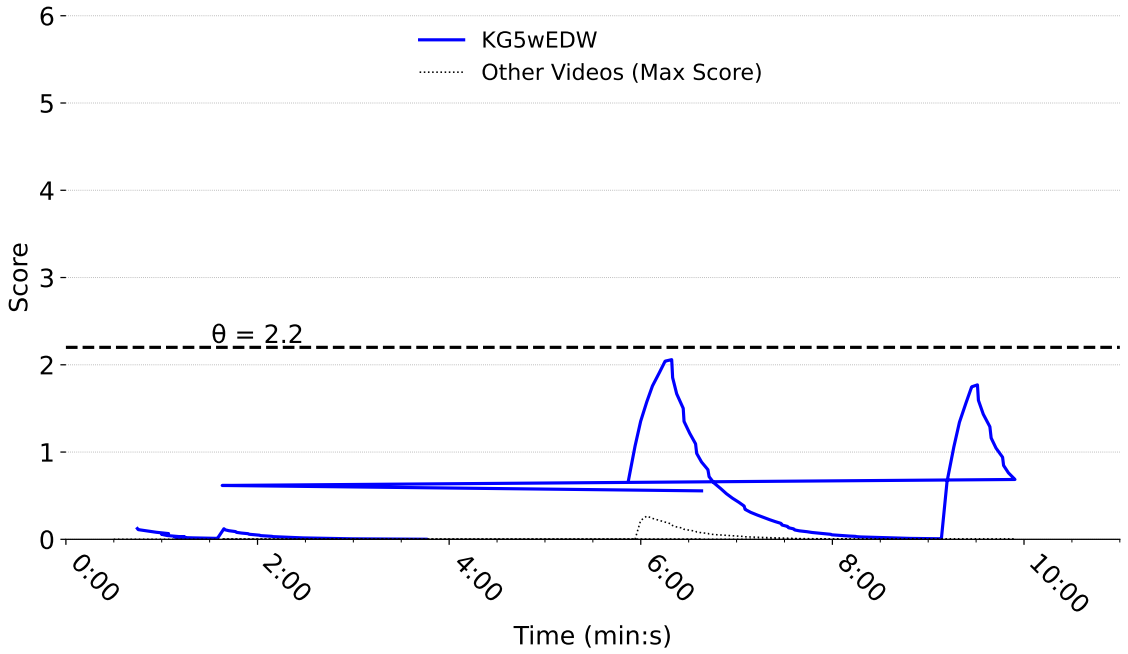


Figure 4.1: The attack's confidence value for an unknown video using regular HTTPS

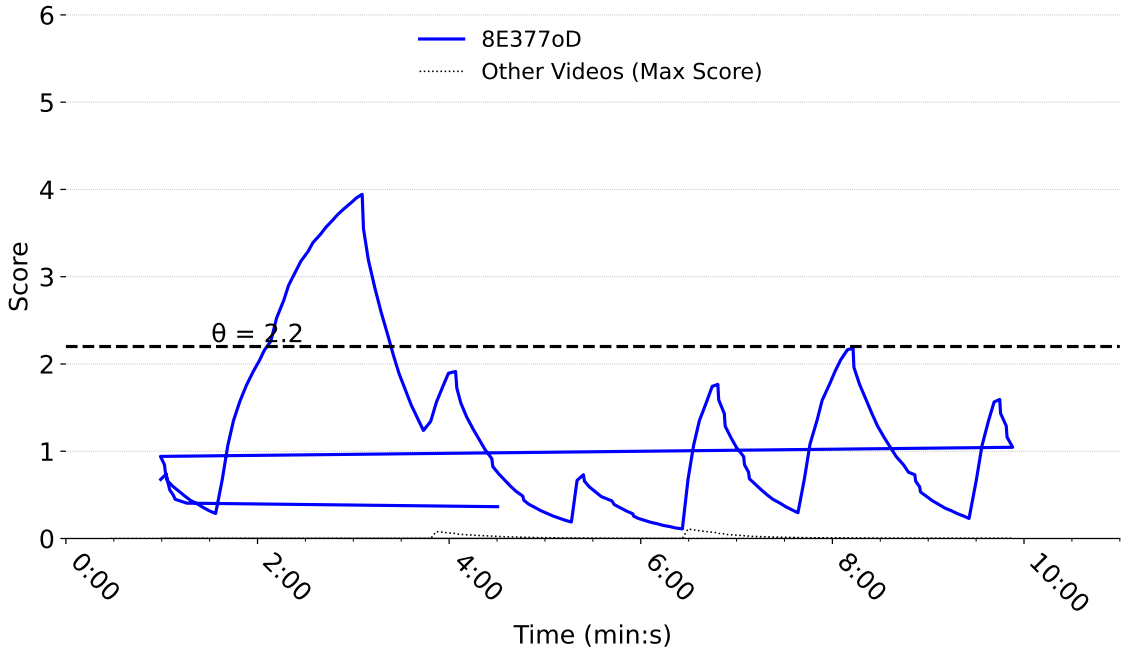


Figure 4.2: The attack's confidence value for an identified video using regular HTTPS

4.2.1 Discussion of the First Iteration

Regular HTTPS: As seen in Table 4.1 the attack is able to identify what videos are streamed from the traffic analysis. What is surprising however, is that there is

a significant mismatch between the original authors accuracy and ours [6]. They achieved almost 100% while we managed 73%. In a lot of cases, as seen in the confidence score Figure 4.1, which can be due to the threshold value $\theta = 2.2$ being too high for some videos. The evaluation program is confident at some points that it has identified the video however, the score does not reach the threshold value and thus outputs "unknown". The discrepancy in accuracy could also be due to the fact that we had to rewrite some data collection scripts, the capture environment being different and its variance and the modification some of the core code of this attack. However, unsurprisingly the attack is still working to some extent confirming that regular HTTPS traffic, and the different ABR streaming protocols that facilitate online streaming, are vulnerable to video fingerprinting.

DAITA: Rather expectedly, DAITA is shown in Table 4.1 to be an effective mitigation tool against this type of fingerprinting attack. As can be seen in Table 4.1, all of the videos were unknown, which is probably due to the properties of the DAITA features. Padding the packets to the same size, adding dummy packets and sending the packets at regular intervals which completely hinders the attack's mode of operations. While this is of course the desired outcome, the size of the network traffic files seen in Table 4.3 compared to the size of the unprotected files seen in Table 4.2 is over twice the amount.

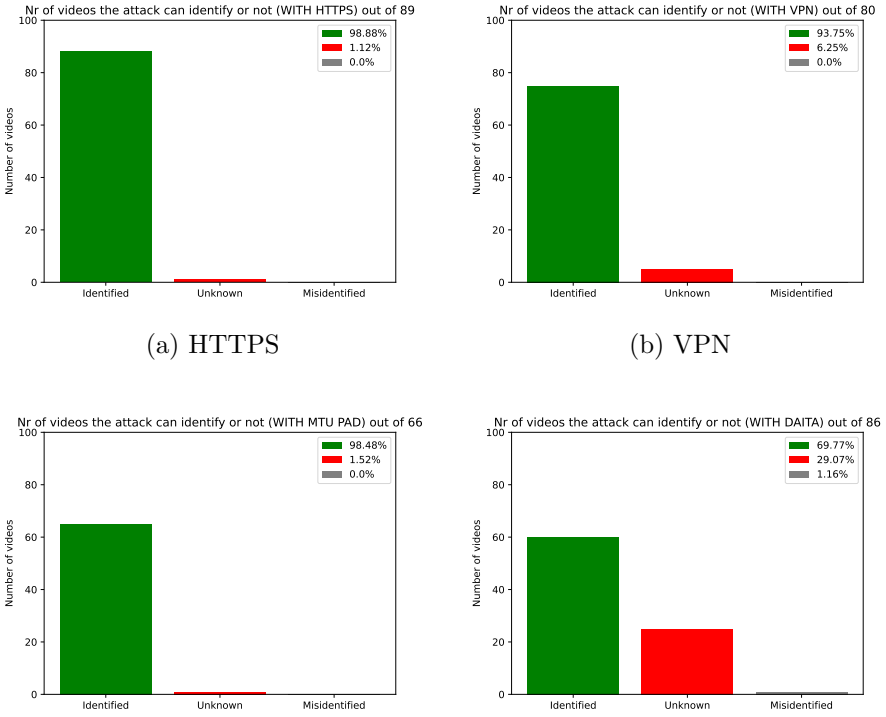
MTU Padding: As shown in Table 4.1, MTU-padding in itself is not enough to mitigate the attack. What MTU-padding does is simply increase the packet sizes to the maximum transmission size of the packet. In our case this is 1500 bytes, but the size itself is not really important, the important part is that all the packets, except the ones sent as a jumbo frame, are the same size to try and change the clarity of the bursts in the network traffic. Doing this would obfuscate the packets true sizes giving the attack a slight disadvantage in noticing bursts.

Mullvad VPN: Tests are run on network traffic only protected by Mullvads VPN, without the use of DAITA. The results in Table 4.1 show that 93 out of the 100 streamed videos are correctly identified, 7 of the videos are unknown and none are misidentified. Comparing this to the test run on traffic not using any protection, it is clear to see that adding VPN protection does not impair the attack at all. Other factors such as the streamed videos fingerprintability or network connection issues that sometimes would arise in the middle of a video being streamed during the test without VPN protection, could also have an impact. Since the usage of a VPN will increase the size of the packets, 230 MB on average seen in Table 4.5 compared to the unprotected files 207.1 MB on average seen in Table 4.2, it is unnecessary.

Maybenot: With minimal padding from the Maybenot simulator machines, the yield is still all of the videos labeled as "unknown".

4.3 Results of the second testing

These are the results of the attack on the data that was recaptured in the improved environment.



(a) HTTPS

(b) VPN

(c) MTU-padding

(d) DAITA

Figure 4.3: Results of the attack on different types of protection

File	File Size (MB)
Smallest File	122.2
Average File	217.3
Largest File	485.0

Table 4.6: No defense file sizes of second dataset

File	File Size (MB)
Smallest File	208.0
Average File	360.9
Largest File	780.4

Table 4.7: DAITA file sizes of second dataset

File	File Size (MB)
Smallest File	139.7
Average File	240.5
Largest File	521.3

Table 4.8: MTU padding file sizes of second dataset

Another issue that is shown in the above graphs are the multiple scores per unit of time. This issue could be due to the way network traffic was captured. However,

File	File Size (MB)
Smallest File	136.0
Average File	243.8
Largest File	543.8

Table 4.9: VPN file sizes of second dataset

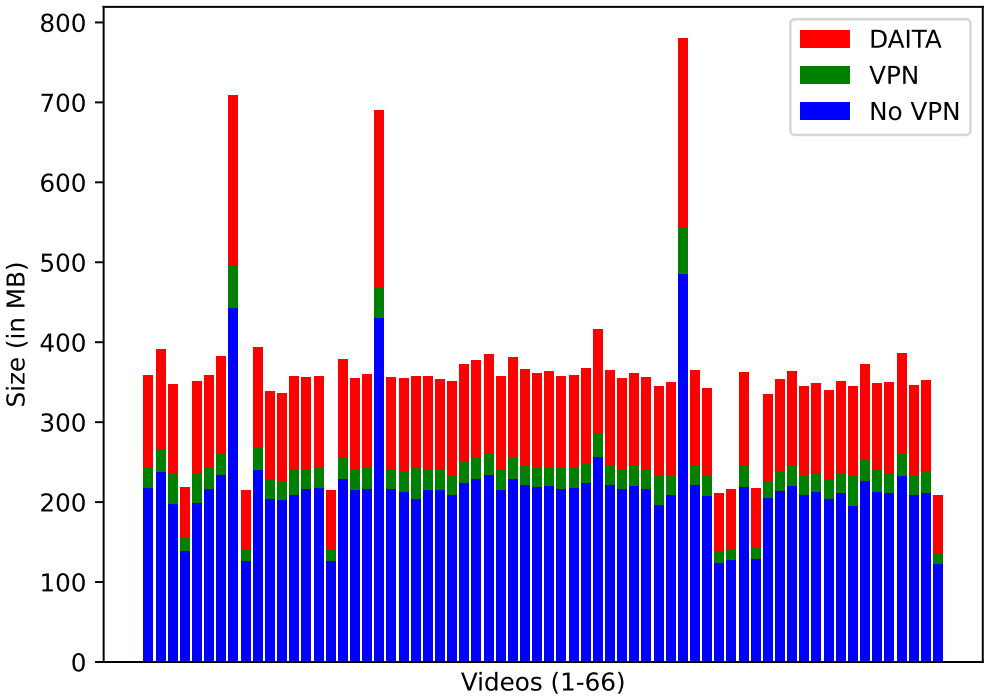


Figure 4.4: Comparison of the file sizes in the second dataset

Protection	Identified	Unknown	Misidentified	Time to identify
HTTPS	85 (100%)	0 (0%)	0 (0%)	54.2692
VPN	79 (99%)	1 (1%)	0 (0%)	179.7913
DAITA	68 (83%)	13 (16%)	1 (1%)	194.2197

Table 4.10: Performances with a threshold of 1.5

Protection	Identified	Unknown	Misidentified	Time to identify
HTTPS	84 (99%)	1 (1%)	0 (0%)	93.8076
VPN	75 (94%)	5 (6%)	0 (0%)	226.4733
DAITA	58 (71%)	23 (28%)	1 (1%)	255.4964

Table 4.11: Performances with a threshold of 2.2 (same videos as in Table above)

the majority of the videos captured had this issue, meaning that it should have no impact on wether the video gets identified or not. These parts are removed from later plots of the confidence scores but could not be removed from the first iterations plots.

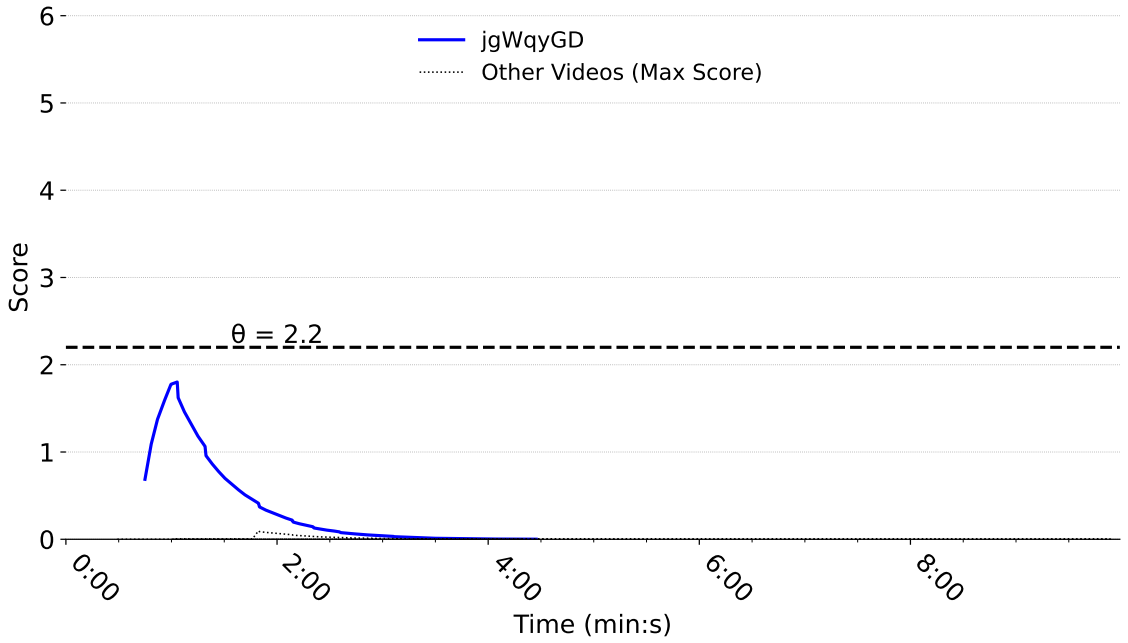


Figure 4.5: The attack's confidence value for an unknown video using regular HTTPS in the second iteration

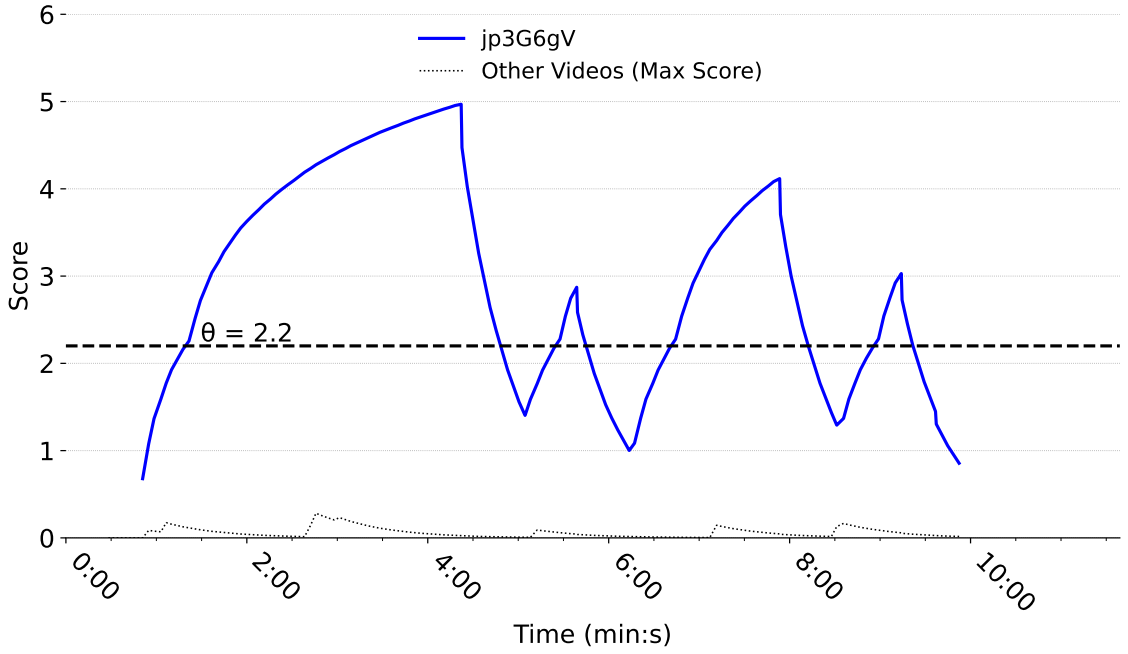


Figure 4.6: The attack's confidence value for an identified video using regular HTTPS in the second iteration

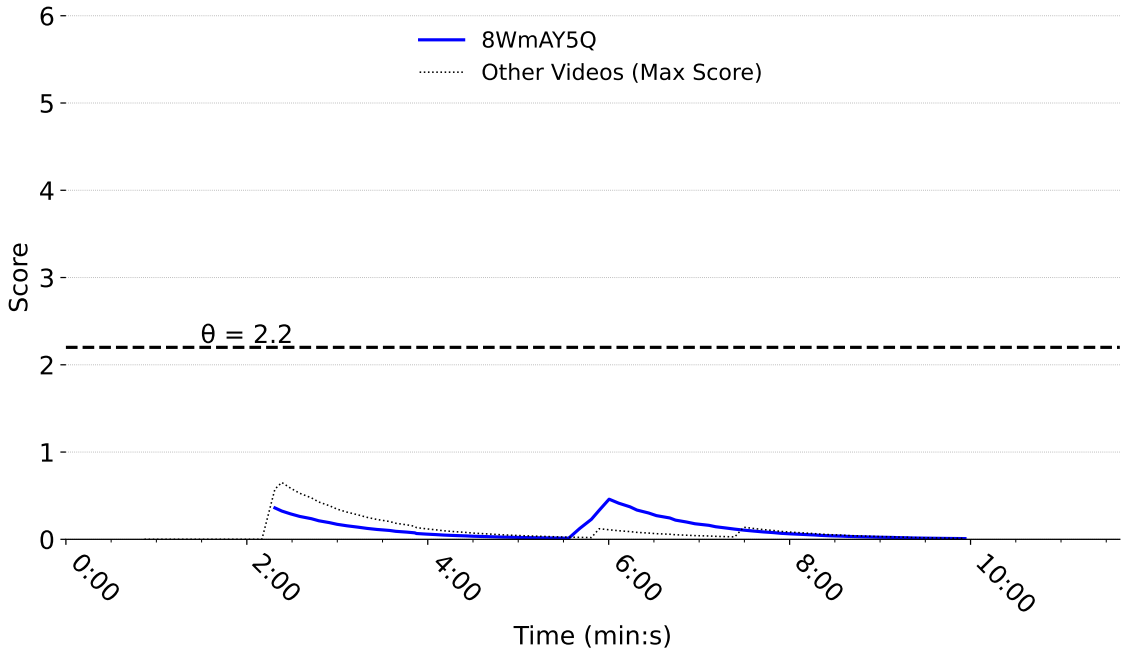


Figure 4.7: The attack's confidence value for an unknown video using DAITA in the second iteration

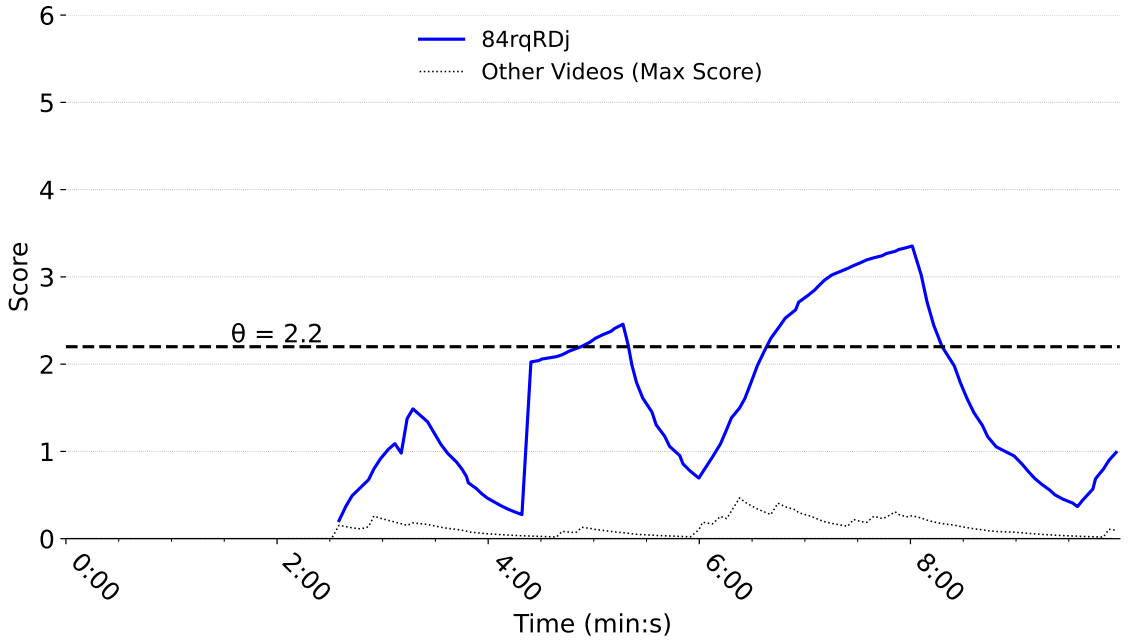


Figure 4.8: The attack's confidence value for an identified video using DAITA in the second iteration

4.3.1 Discussion of the Second Iteration

Regular HTTPS: In the second experiment, the attack is able to identify 98.88% of the videos. This is a significant increase from the first iteration. The performance of the attack in the second iteration is more similar to the one in "Endangered

Privacy" [6], which could be due to the testing environment removing all the noise. However, the environment in "Endangered Privacy" is quite noisy and still reaches identification rates of almost 100% (the attack model and environment is more close to reality where a target would be on a network with other devices and some background noise from its own device).

DAITA: The majority of the videos protected by DAITA in the second dataset are correctly identified by the attack. This is in contrast to the first dataset where none of the 100 videos are correctly identified. One explanation to this could be that DAITA changes its mode of protection by randomly selecting one among an array each time a connection to the VPN server is made. This means that in the first dataset, a mode of defense that would not be as easily broken by the attack could have been chosen while one that does not have a large impact on the attack's performance to correctly identify videos could have been chosen in the second iteration of the dataset. This also holds true for all the videos in a given dataset since in both the first and second iteration of the data collection there was only one connection made initially to the server and all the videos collected are streamed on that connection. As it is unknown what defenses Mullvad has stored in the database of defenses for DAITA and it is also unknown what mode is chosen at each connection, we can't really draw any other conclusions. One can get a general idea however of what kind of defenses that can be activated during a DAITA connection. As mentioned in Section 2.4, defenses like FRONT and RegulaTor can be implemented with Maybenot and thus one could assume that defenses similar to those are part of the DAITA defenses. In this work, other iterations of experiments with different DAITA connections were not produced. It would be interesting to see to which extent the different defenses yield different results on how effective DAITA is against this video fingerprinting attack.

MTU Padding: MTU padding has roughly the same performance as HTTPS yielding an identification rate of 98.48% which is significantly higher than the first iteration of testing. However, the result is still more or less the same as HTTPS. The change in experiment environment did not affect the attack's ability to identify the videos which confirms that only MTU padding does not disrupt the attack enough to obfuscate the traffic.

Mullvad VPN: In the second iteration, 93.75% of videos are identified, which means that the Mullvad VPN is a bit stronger at defending the data than regular HTTPS and MTU padding. However, 93.75% is still a high identification rate and not safe. On the other hand, it is surprising that the identification is slightly weaker on the second iteration since there is less noise. There's still enough reason to believe that only VPN is not enough to defend against this attack.

Maybenot: For all values that are tested between 0.05 and 0.95, the videos are always unknown similarly to the results of the DAITA testing which led us to testing with minimal padding and see if the blocking action is the main reason why the attack is mitigated. With minimal padding the attack is still mitigated.

Dataset size differences: The second dataset is smaller than the first dataset. The initial idea was to gather 100 videos for the experiment however this proved to

Protection	Average number of bursts	Average burst size
HTTPS	171.2	1252726.7
VPN	169.7	1343626.0
DAITA	217.1	1573741.7

Table 4.12: Average number of bursts and burst size from burstshark output

be a bit more tedious than expected. The video streaming and network traffic collection in the second iteration is done with the same method as in the first iteration. However, some of the randomly chosen videos are sometimes unable to be played due to SVT prompting a login by the user which caused the script to skip that specific video. This is not the case in all the different datasets, just some of them and for different videos in each. Another reason for the smaller dataset is that some of the videos captured in the first dataset are later unlisted when it comes time to collect the DAITA dataset for example, due to some of the videos being news broadcasts which only stay available for a few days to a week.

4.4 Bursts and bitrate

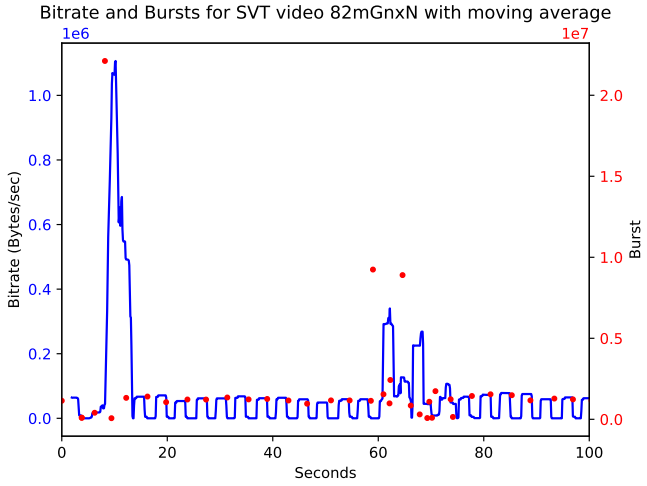
The way that bursts are correlated to bitrate can be shown in these plots below shown as a moving average with a window size set to 20.

As can be seen in Figure 4.9, DAITA worsens the distinction between the bitrate bursts, shown in the plot as a smaller difference in height between the peaks and the valleys compared to the plots depicting HTTPS and VPN.

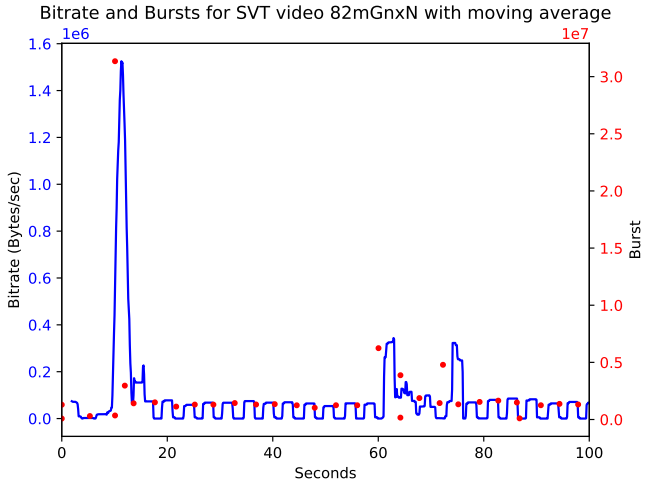
4.5 Capturing environments effect on results

As shown in Table 4.1, 27 out of the 100 unprotected videos are classed as unknown, which can be attributed to the fact that the device collecting the network traffic was connected to a router on to which other devices was simultaneously connected. The added noise could have had an impact the results.

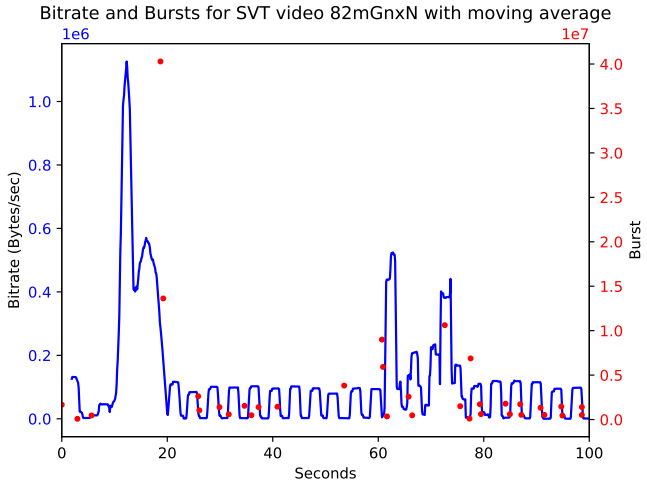
In some cases the issue is that the threshold value was set too high. The threshold value set as the standard was 2.2, but 1.5 is another value given as a recommendation. As shown in Table 4.10, after running the same evaluation with a threshold score set to 1.5, there is an increase in identified videos compared to the iteration of the attack with a threshold value set to 2.2.



(a) HTTPS



(b) VPN



(c) DAITA

Figure 4.9: How burstsharks output correlates to the bitrate in HTTPS, VPN and DAITA with rolling window of size 20

5

Conclusion

We have presented how the video fingerprinting attack presented in "Endangered Privacy" [6] affects different levels of commercial internet security. In the first iteration of experiments, HTTPS and VPN traffic was expectedly not able to defend against the attack. MTU padding, on its own, showed to not be enough of a defensive solution. DAITA managed to completely defend against the attack. The same can be said for each of the methods tested in the second iteration except for DAITA. We showed that DAITA is vulnerable to this type of video fingerprinting attack.

The difference in DAITA's performance between the two iterations can be explained by multiple reasons. The fact that we had to implement our own data collection and tweak the existing code of multiple programs can be a reason as to why results differ from the original work. How defenses are randomly chosen from a database of thousands of defenses can also impact the robustness of the defense against video fingerprinting.

We have also improved the work of the authors of [6] such that it is possible to easily recreate the experiments conducted by the authors.

Bibliography

[1] D. Xue et al., "OpenVPN is open to VPN fingerprinting," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 483–500, ISBN: 978-1-939133-31-1. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/xue-diwen.

[2] S. Bhat, D. Lu, A. Kwon, and S. Devadas, "Var-cnn and dynaflow: Improved attacks and defenses for website fingerprinting," *CoRR*, vol. abs/1802.10215, 2018. arXiv: 1802.10215. [Online]. Available: http://arxiv.org/abs/1802.10215.

[3] M. Wu et al., "How the great firewall of china detects and blocks fully encrypted traffic," in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 2653–2670, ISBN: 978-1-939133-37-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/wu-mingshi.

[4] Alice, Bob, Carol, J. Beznazwy, and A. Houmansadr, "How china detects and blocks shadowsocks," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 111–124, ISBN: 9781450381383. DOI: 10.1145/3419394.3423644. [Online]. Available: https://doi.org/10.1145/3419394.3423644.

[5] K. Abe and S. Goto, "Fingerprinting attack on tor anonymity using deep learning," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:114424634.

[6] R. D. Martin Björklund, *Endangered privacy: Large-scale monitoring of video streaming services. usenix 2025 (to appear).*

[7] S. C. B. Sikha Bagui and J. Sheehan, "Comparison of machine-learning algorithms for classification of vpn network traffic flow using time-related features," *Journal of Cyber Security Technology*, vol. 1, no. 2, pp. 108–126, 2017. DOI: 10.1080/23742917.2017.1321891. eprint: https://doi.org/10.1080/23742917.2017.1321891. [Online]. Available: https://doi.org/10.1080/23742917.2017.1321891.

[8] T. Pulls and E. Witwer, *Maybenot: A framework for traffic analysis defenses*, 2024. arXiv: 2304.09510 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2304.09510.

[9] G. Kadianakis, T. Polyzos, M. Perry, and K. Chatzikokolakis, *Tor circuit fingerprinting defenses using adaptive padding*, 2022. arXiv: 2103.03831 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2103.03831.

[10] *Introducing defense against ai-guided traffic analysis (daita)*, https://mullvad.net/en/blog/introducing-defense-against-ai-guided-traffic-analysis-daita.

[11] *Daita: Defense against ai-guided traffic analysis*, https://mullvad.net/en/vpn/daita.

[12] M. Björklund et al., "I see what you're watching on your streaming service: Fast identification of dash encrypted network traces," in *2023 IEEE 20th Consumer Communications Networking Conference (CCNC)*, 2023, pp. 1116–1122. DOI: 10.1109/CCNC51644.2023.10060390.

[13] *What is a phishing attack?* https://www.cloudflare.com/learning/access-management/phishing-attack/.

[14] J. A. Donenfield, *Wireguard: Next generation kernel network tunnel*, 2017. [Online]. Available: https://www.wireguard.com/papers/wireguard.pdf.

[15] H. Abbas et al., "Security assessment and evaluation of vpns: A comprehensive survey," *ACM Comput. Surv.*, vol. 55, no. 13s, Jul. 2023, ISSN: 0360-0300. DOI: 10.1145/3579162. [Online]. Available: https://doi.org/10.1145/3579162.

[16] E. Borges, *Cybersecurity fingerprinting: Concept, insights and strategies*, https://www.recordedfuture.com/threat-intelligence-101/vulnerability-management-threat-hunting/fingerprinting-in-cybersecurity?utm_source=chatgpt.com.

[17] M. Shen, K. Ji, Z. Gao, Q. Li, L. Zhu, and K. Xu, "Subverting website fingerprinting defenses with robust traffic representation," in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 607–624, ISBN: 978-1-939133-37-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/shen-meng.

[18] T. Pulls and E. Witwer, "Maybenot: A framework for traffic analysis defenses," in *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*, ser. WPES '23, Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 75–89, ISBN: 9798400702358. DOI: 10.1145/3603216.3624953. [Online]. Available: https://doi.org/10.1145/3603216.3624953.

[19] J. Gong and T. Wang, "Zero-delay lightweight defenses against website fingerprinting," in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, Aug. 2020, pp. 717–734, ISBN: 978-1-939133-17-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/gong.

[20] J. K. Holland and N. Hopper, *Regulator: A straightforward website fingerprinting defense*, 2021. arXiv: 2012.06609 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2012.06609.

[21] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06, Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 255–263, ISBN: 1595935185. DOI: 10.1145/1180405.1180437. [Online]. Available: https://doi.org/10.1145/1180405.1180437.

[22] R. Soepeno, *Wireshark: An effective tool for network analysis*, Sep. 2023. DOI: 10.13140/RG.2.2.34444.69769.

[23] M. Tsoukalos, "Using tshark to watch and inspect network traffic," *Linux J.*, vol. 2015, no. 254, Jun. 2015, ISSN: 1075-3583.