Social profiling of users through information leakages

A practical security analysis

A thesis submitted in fulfillment of the partial requirements for the degree of master in Computer Science

Author:
Mariano Di Martino

Promotor:
Prof. dr. Peter Quax

Mentor:
Pieter Robyns

Universiteit Hasselt
2017-2018

Contents

Acknowledgements

First, I would like to thank my mentor Pieter Robyns and promotor Prof. dr. Peter Quax for guiding me through the difficult process of writing a thesis and academic paper. Both have put an extensive amount of time into answering any questions I had related to this thesis.
I would also like to thank Prof dr. Wim Lamotte for proofreading my paper and to provide me with suggestions for strategical improvements.
Furthermore, I would like to thank people from the Expertise Centre for Digital Media (EDM) for providing me with valuable information about related research.
And of course, family, friends and my girlfriend had a profound impact on the path that lead to this thesis.

1 Thesis outline

Network security has become an important aspect in the life of almost every individual. Recent data breaches [42, 51] have resulted in a wake-up call for businesses and consumers worldwide. Over the last decade, an overwhelming number of network attacks and methods [101, 55] have been developed. Many of these attacks have a similar goal, that is to obtain sensitive information of the end user. Such sensitive information can then be analyzed to predict the behaviour or interests of individuals [66] or even worse, to enable the use of identity theft [57]. Furthermore, the end users are still publicly sharing a lot of that personal information themselves through the wide variety of social networks available on the internet. More recently, companies that analyze such data have garnered attraction due to the legality and questionable effectiveness of these analyses [86] [16]. Regardless whether or not these social analyses are legal, they are actively being used for predicting social media influence [62], personalized advertisements, assessing financial and health conditions [34] or even in forensic science [35].
This thesis discusses and analyzes the different attacks that can be used to socially profile Internet users. First, we discuss some practical cases why social profiling is in demand and examples in which context a malicious adversary can deploy such methods. Next, we discuss several attacks that can be utilized to fingerprint the devices and websites used by an Internet user and provide some effective countermeasures to defend ourselves against those attacks. Furthermore, we propose a novel fingerprinting method to identify specific profile URLs navigated by an end user. Finally, we conduct several experiments where we analyze the impact and defenses of these attacks against existing online platforms and how they affect social profiling.

1.1 Marketing context

Marketing is one of the most prevalent aspects of selling products online. Knowing how a customer behaves and what a customer likes is the key to personalized advertisements. Nevertheless, social media plays an important role in communicating with these potential customers.
In a social media context, predicting the interest or behaviour of potential customers is usually done with consent of the end user. For instance, large social media networks gather personal data of consumers in return for their services.
However, in the recent years, the industry has given rise to a new popular technique called 'Social Wi-Fi'. This concept consist of a Wi-Fi access point (AP) that allows customers to use the internet for free if they agree to share *some* personal information (e.g. name, social media login, etc ...). Many companies have jumped on the bandwagon to provide social Wi-Fi as an easy and efficient method that can be used to identify the behaviour of customers by for instance, physically tracking them or gather statistics about the social interactions of these customers [85, 94, 93].
However, on a technological side, the features are relatively basic and are more focused on giving *meaning* to the collection of personal data than how to *gather* the personal in-

formation. According to Cisco, 58% of business users are not willing to use social Wi-Fi. From which half of them state the reason for their denial is due to not wanting to share personal information with the retailers (companies that use social Wi-fi) and due to the inconvenience of logging in and using the time to make it internet ready [98].
Nevertheless, the 'social Wi-Fi' providers do have access to the data that is sent between the consumer and the internet. Therefore, providers (or a malicious MITM) could use these streams of data without consent of the consumer (victim). Even though most of the time, the stream of data is encrypted, it is still possible to extract sensitive information without intervention of the end user by applying fingerprinting methods.
Similarly, "privacy related" software like Onavo Protect from Facebook Inc. are openly admitting the collection of network traffic for analytical purposes [7]. Recent EU leglislations regarding the possibility of opting out for browser cookies will also result into a significant drive towards other methods to extract sensitive information from the end user [102].
In order to find effective defenses against such practices, this thesis first discusses several techniques to obtain such information without breaking encryption. And only then, we can construct and compose mitigation techniques to protect the information of our end user and to mitigate the risks introduced by using 'social Wi-Fi' and the like.

1.2 Cybercrime context

In 2017, the total cost of cybercrime globally was estimated to be over $600 billion and continues to rise with an average of 24% each year [8]. According to McAfee, there are an average of 33 000 cases of phishing each day and they show that this correlates to the number of data breaches [67]. Traditional crime techniques are not sufficient in mitigating or reducing these massive costs. Therefore, novel methods have to be deployed in order to cope with the constant stream of new cybercrime attacks.
As we will discuss later in the thesis, wiretapping is such method employed by ISP providers and large government organisations to intercept the communication from and to the end user, in order to collect digital evidence in legal cases [14, 36]. Since most criminals are aware of the fact that encryption is necessary to hide their wrongdoings, fingerprinting techniques are useful in extracting sensitive information from this stream of communication data. More specifically, the utilization of social media networks has become paramount to gather and analyze public data that can aid cybercrime departments into developing their forensic cases [75]. Similarly to the marketing context, personal information that is leaked in data breaches can be used by government organisations to penetrate networks as is the case in recent legal enactments of the Netherlands [77]. It is clear that, even though the deployment of these new forensic techniques is inevitable, the privacy implications that arise are significant and thus requires defenses that protect the end users against malicious actors that might deploy similar methods.
In this thesis, we first show how forensic techniques such as the aforementioned, can be utilized to extract personal identifiable information useful in battling cybercrime. Next, we discuss the privacy considerations that become apparent when deploying these techniques

and how end users can legally protect themselves against such exposures.

2 Background

2.1 HTTP over TLS

In this section, we provide a very basic overview of the TLS 1.2 protocol [31], more specifically HTTP 1.1 over TLS.
The Transport Layer Security protocol is implemented on top of HTTP to provide integrity, encryption and authenticity of the data sent between two endpoints. In other words, it provides a channel between an authenticated webserver and webclient where the communication cannot be altered nor read by an adversary.
To establish this channel, a TLS handshake is performed where the client and server agree to, among other things, a cipher suite and version based on a *Client Hello* message and a *Server Hello* message. Next, we exchange keys or certificates to be able to encrypt and decrypt the communication between the endpoints and prove the authenticity of the server. After the TLS handshake is complete, we can encrypt the HTTP data and send it through the network using *TLS Application* messages. When it arrives at the other endpoint, we can decrypt the HTTPS data and it is then passed down to the HTTP protocol layer as usual.
A simplified version of the TLS handshake is shown in the figure below:

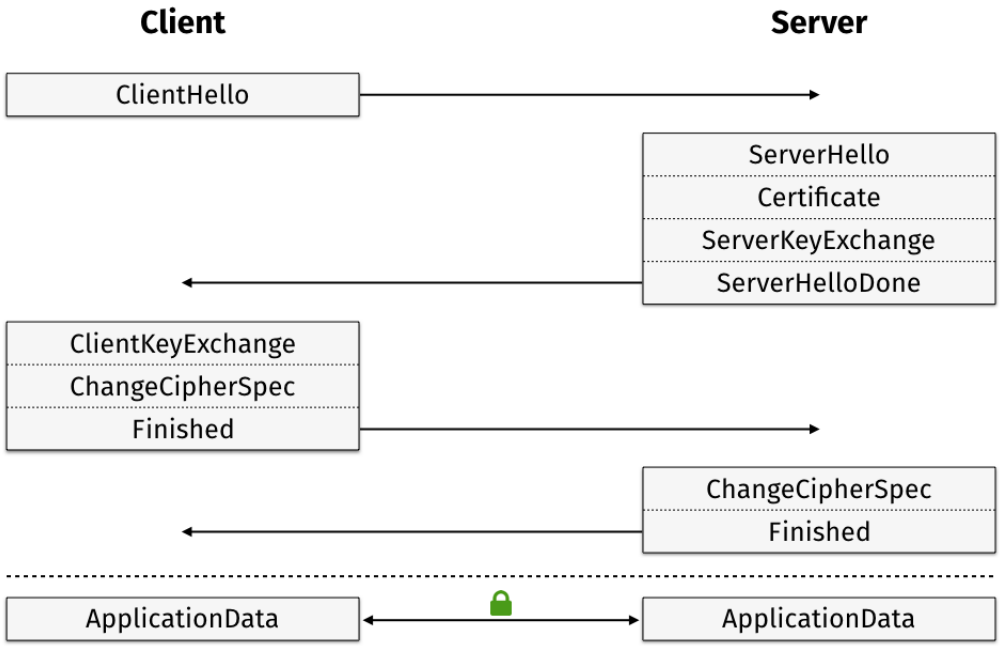

Figure 1: A basic TLS handshake (Courtesy of Tim Aubert: timtaubert.de)

The Application Data messages that contain the actual HTTP data are of course encrypted. To the contrary, the messages used in the TLS handshake are sent in plaintext and so can be read by an adversary. Alteration of these messages by an adversary are prevented by introducing a bidirectional *Finished* message at the end of the TLS handshake. This message contains a hashed form of all [1] previous TLS messages based on the agreed encryption key of both endpoints. It is therefore (in theory) not possible to modify TLS messages and *Finished* messages without knowing the shared encryption key.
The question now remains, how can 2 endpoints agree to a shared secret key on a public channel without leaking this information to other entities on this channel?
This is why the *Diffie-Hellman* algorithm is used for. In theory, the security of this algorithm is based on the fact that solving a discrete logarithm problem cannot be performed efficiently.
Since HTTP is almost always used under TCP, it is possible that a HTTP request or response in a TLS record will be split into multiple TCP segments due to the segmentation in the TCP layer (see figure 2). The complete TLS record is easily reconstructed by appending both TCP payloads.

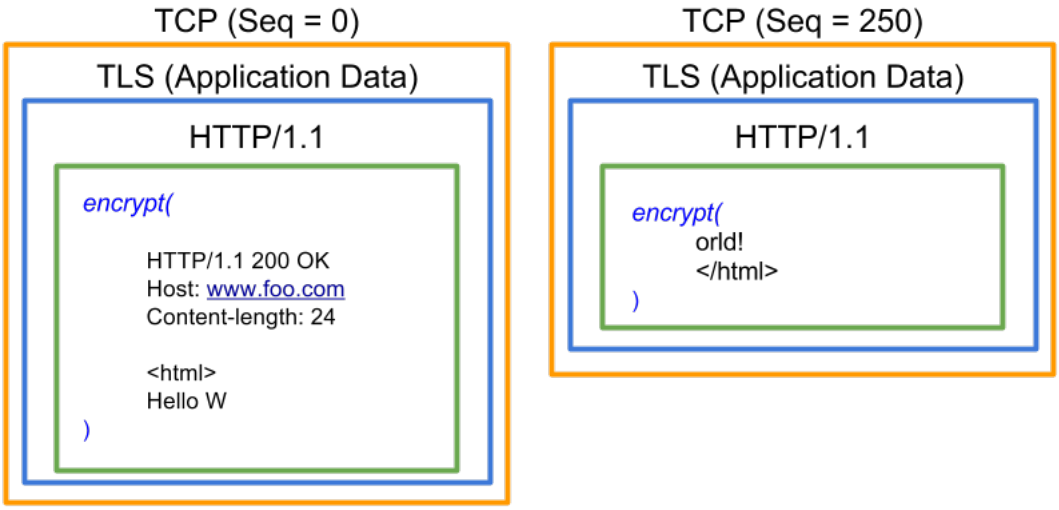

Figure 2: A conceptual example of a HTTP response split into 2 TCP segments.

2.1.1 Cipher suites

A cipher suite is a combination of algorithms that are used to encrypt a portion of data in TLS. A wide range of cipher suites are supported by TLS. In TLS <= 1.2, we can use either a *block* or *stream* cipher. In a block cipher, we split our payload (e.g. a HTTP request) into different blocks each with the same fixed size (e.g. 128 bits). Splitting the payload in

[1] With some exceptions like the *HelloRequest* message

blocks is necessary to perform block mode operations (e.g. Cipher Block Chaining) which is used in combination with a block cipher (e.g. AES). Since the size of the data is not always divisible by the block size, *padding* is introduced. Padding will add bytes to the end of the data so that the total size of the TLS payload will be a multiple of the block size (see figure 3). Then, each block individually is encrypted according to the block cipher and combined according to the block mode operation.
In a stream cipher, the data will not be split into blocks. Instead, we encrypt the data using an *infinite* keystream derived from a secret key. As a result, no padding is necessary because we can then encrypt each plaintext byte one by one.
Another important mode of operation is *Galois/Counter Mode* (GCM). This mode of operation can be used in combination with block ciphers but due to having a counter mode, it acts like a stream cipher.
We make this simplified distinction to illustrate that the data encrypted with a block cipher has padding which will transform the plaintext to an encrypted text where the size is a multiple of the block size (which is at least the same size of the plaintext). To the contrary, the encrypted data using a stream cipher will have the same length as the original plaintext data.

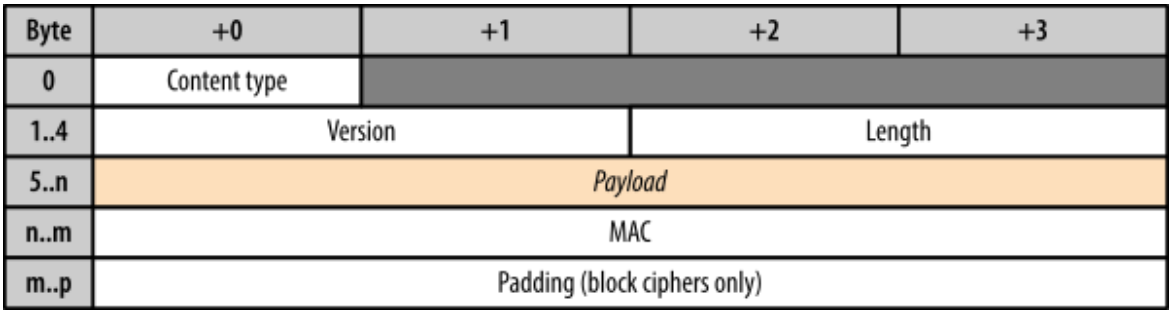

Figure 3: A TLS record for CBC ciphers [43]. The MAC value will make sure that the TLS record is not modified.

2.1.2 Practical calculation of HTTP length

As a result of the previous section, we can (to some extent) calculate the length of the original plaintext payload. With a block cipher, the length is divisible by the block size (figure 3) and with a stream cipher, the length can be calculated with byte precision. In other words, we can calculate the length/size of an original plaintext HTTP request or response by subtracting the length of the headers of the TLS record with the length of the complete TLS record [2].
In order to calculate the length, we first have to construct a stream of data that consists of TLS records (that might consist out of multiple TLS Application Data messages) in

[2]This only applies to the HTTP/1.X protocol

both directions. A TLS record sent from client to server is a HTTP request and a record sent from server to client is a HTTP response. Most modern browsers do not support or enable HTTP/1.1 pipelining by default, which means that both entities can not sent another HTTP request/response until the current one is completely received.
Thus, we can detect the beginning and end of a HTTP request and response based on the flow of data. In current literature, the point of inflection where a request or response starts is commonly referred to as a *'size marker'*, initially proposed by Panchencko et al. [81]. After we have found out the size markers, we can calculate the actual length of the HTTP payload by subtracting the total size of the complete TLS record with the size of the TLS headers (which have a fixed length).

In this ideal scenario, we have assumed that only TLS Application Data messages are being sent and that we have access to all TLS communications from the beginning of the TCP connection. The first assumption can be relaxed trivially by removing any TLS message where the content type header does not indicate being a TLS Application Data message. The length of such TLS message is given in the TLS header. The second assumption is much harder to relax.
Let's assume we have access to the exchange of TLS messages in the middle of a TLS record. In this scenario, we do not know where the current TLS records ends neither do we know the total length of the record [3]. A possible solution is to iterate over the stream of data and check for a sequence of bytes that indicates a TLS Application Data message and expected version number. For instance, 0x170303 indicates a TLS Application message with version 'TLS 1.3'. This might indicate the beginning of a new TLS Application Message with which we can then continue the calculation as discussed above. However, the issue here is that the sequence 0x170303 might also appear in the actual TLS payload data. This can be solved by iteratively checking if the expected sequence appears in the stream of data after each X bytes (where X is the length of the TLS record indicated by the header). For each iteration, the chance of detecting the beginning of a real TLS Application Data message increases exponentially. A fixed iteration depth can be set to avoid infinite loops if a TLS record is malformed.

The basics of this method are originally mentioned by Hintz [47] and further extended by Sun et al. [96]. Over the many years, a considerable amount of side-channel attacks have been developed that also use other inadvertent leakages of HTTP request/response sizes. In the bicycle attack [103], we can accurately calculate the lengths of passwords (e.g. in a login request) by knowing the size of a HTTP response and predicting the size of the HTTP headers. And the BREACH attack [55] uses the size leakage as an oracle to attack certain compression algorithms used in the HTTP protocol.
It is clear that this type of leakage is still a valid issue prevalent and useful in many HTTP attacks today.

[3] Wireshark indicates this by showing 'Continuation Data'

2.2 Man in the middle (MITM)

The term 'a man in the middle' is used to indicate that an adversary has set himself up between 2 or more endpoints that want to communicate with each other. The goal of a MITM is to intercept or modify traffic flowing over the network between those endpoints (see step 2 in figure 6). This does not necessarily mean that a MITM can read data in the clear. For instance, when using HTTP over TLS, the data that a MITM would see is still encrypted. However, as we have discussed in the previous sections, we can still extract valuable information from the TLS connection.
There are 2 types of MITM positions: active and passive. An active MITM alters the flow of data between the endpoints (for instance, DNS poisoning alters the DNS responses to redirect the victim to a malicious server). A passive MITM only observes/reads the flow of data between the endpoints. For instance in theory, a MITM cannot tamper with a TLS connection because the *Finished* messages will prevent this. If a victim would use vanilla HTTP, a MITM could possibly intercept and modify the HTTP data sent between the endpoints which is of course a security risk.
Systems that do traffic analysis (e.g. antivirus software, firewalls, etc ...) often hold MITM positions in order to check the data for malware, network attacks, etc ... . Some governments have even launched initiatives in which they hold MITM positions in various ISP networks in order to use this sensitive data for forensic analysis [36] [14]. Even though most of the intercepted network traffic is encrypted, private keys can be requested and fingerprinting techniques can be utilized to extract meaningful information. Common public tools that can hold MITM positions or make use of those positions are Burp [83] and mitmproxy [71].
Either way, it is evident that communicating over a network requires a solid mechanism that protects both endpoints for MITM attacks. Unfortunately, due to outdated concepts or technical limitations at the time, some network protocols still leak sensitive information as we will show in the next section.

2.3 Network protocols in the clear

We assume that the reader has a basic knowledge about the Ethernet, IP and TCP protocol. In most scenarios of our discussions, we also assume that the IP and TCP fields of an Ethernet frame can be read by a MITM. In other words, we exclude the use of additional encryption services like for instance SSH tunneling or Tor, where parts of the IP and TCP data are encrypted to prevent eavesdropping.
The first important protocol that we use to extract valuable data from is the DNS protocol [72]. The Domain Name Server (DNS) is a network protocol and system that primarily translates domain names to IP addresses. It works by hierarchically or recursively requesting the IP address from other domain name servers.
For instance, a simplified example: if we visit the website 'www.uhasselt.be', a domain translation has to be done first before we can actually request the website content. This translation is performed by sending a DNS request to a domain name server (usually on

port 53). This DNS request contains the domain name 'uhasselt.be' and for the sake of simplicity, we assume the server *knows* the IP address of the domain 'uhasselt.be'. As a result, the specific IP address will be returned by the domain name server and a HTTP request can be sent to the IP address. Such a DNS request is not encrypted and so can be read by an adversary that holds a MITM position. In other words, an adversary can intercept the DNS request and know to which website the victim wants to navigate.
To the best of our knowledge, there is no evidence that shows that social Wi-Fi providers intercept DNS requests in order to gather more information about their customers. However, besides from legal repercussions, there are no technical limitations that prevents a provider from collecting DNS traffic. Furthermore, some ISPs block restricted websites (for instance, The Pirate Bay) by using DNS blockades. The DNS responses of such websites are in fact modified by the ISP with the goal of redirecting the victim to another website. Interestingly, there are reliable reports that Google is considering to add DNS over TLS in one of Android's future versions [12, 91]. This would prevent an adversary from extracting the domain name from a DNS request and it should also prevent the adversary from poisoning the DNS cache.
We deliberately left out more details about the inner workings of the protocol because they are not important to understand the DNS issues discussed in this thesis.

Another important component is the Server Name Indication field. The Server Name Indication (SNI) is a TLS extension that informs the webserver which hostname it wants to connect to. It is usually used for virtual hosting where multiple HTTPS websites can be accessed under one IP address. The same information can be extracted as with a DNS request, more specifically, the hostname that the victim wants to access. Unfortunately, all TLS extensions (including the SNI) are sent unencrypted and thus can be eavesdropped by an adversary.
Gonzalez et al.[41] present a more extensive and architecural overview of the viable side-channel leaks in HTTPS.

2.4 Sensitivity, precision and F1 score

Sensitivity and precision are measurements used in several domains of statistics. In this thesis, we use these measurements to compare and analyze the behaviour of different fingerprinting results where we try to predict which fingerprint belongs to a given traffic trace. Sensitivity (also called recall or true positive rate) can be calculated by the following formula:

$$\mathrm{TPR} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN}) \tag{1}$$

Precision (also called positive predictive value) can be calculated using the following formula:

$$\mathrm{PPV} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP}) \tag{2}$$

There are several accepted ways of defining what a *true positive*, *false positive*, *true negative* and *false negative* are. In the results of our own experiments and attacks [4], we define the following measures:

- True positive (TP): Correctly predict that the intercepted traffic belongs to fingerprint A in a certain interval of time.
- False positive (FP): Incorrectly predict the intercepted traffic to be from fingerprint A, while it is actually from fingerprint B.
- True negative (TN): N/A
- False negative (FN): Predict nothing while the intercepted traffic actually belongs to fingerprint B.

True negatives are not useful in the context of our results because it would essentially mean 'correctly predict that the intercepted traffic does not belong to any fingerprint X'. Such measure is especially utilized in closed world scenarios where they perform binary tests whether or not their algorithm correctly predicts that dummy traffic does not belong to their collection of fingerprints [52]. A clearer explanation can be found in the discussion of the IUPTIS attack in section 4.9.
Ultimately, the F1 score (or F measure) provides us with a general view on how accurate our results are. This measure can be seen as an harmonic mean of the sensitivity and precision and is calculated using the following formula:

$$F_1 = \frac{2}{\frac{1}{TPR} + \frac{1}{PPV}} \tag{3}$$

3 Profiling through fingerprinting methods

Standarized in 1996, HTTP/1.0 and HTTP/1.1 have been widely used in the World Wide Web for requesting web content. Until 2000, communication over the HTTP protocol was

[4] Existing papers regarding fingerprinting often have slightly different ways of defining positives and negatives

sent in plaintext and could be intercepted by anyone with a MITM position. Thanks to the standarization of HTTPS, half of all the websites on the internet use HTTPS. However as of October 2017, 74% of Alexa's top websites and only 61% (as of August 2017) of the websites that provide health services in the Netherlands use HTTPS [39].
Its obvious that the impact of using vanilla HTTP is far more prevalent than any attack on a HTTPS channel. Since we believe that the rise of HTTPS will eventually lead to a more encrypted internet, we will focus this section on how leakages can be utilized to construct fingerprinting methods in order to further socially profile users in a HTTPS-based context. Additionally, in some cases, we also briefly discuss the possible impact on leakages of HTTP websites that do not request or display sensitive data directly to the end user.

3.1 Webpage fingerprinting

Website or webpage fingerprinting (WFP) is a large part in the domain of traffic analysis. The goal of WFP is to identify which website/webpage a victim is visiting over a protocol that provides encryption of payload data (for instance, HTTPS or SSH). A considerable amount of research has been done into identifying websites through encrypted tunnels like VPNs or Tor [80, 81, 46, 64]. Most of that research is focused on generalised and conceptual algorithms that work for a large variation of websites or webpages. Additionally, the research is specifically focused on how to circumvent novel mitigations taken by anonymization/hidden services like Tor, which (in turn) decreases the accuracy of these attacks. However, within the scope of our thesis, we do not target users that use encrypted tunnels on top of HTTPS. Therefore, we discuss a general overview of website fingerprinting attacks but focus on simplified fingerprinting methods with high accuracy that can predict websites or webpages over HTTPS usable for social profiling.

Early work from Cheng et al. [22] and Sun et al. [96] have shown that it is feasible to fingerprint webpages over HTTPS by taking the size of web objects into account. Nonetheless, some of the assumptions provided in these works, such as one TCP connection per web object, are not valid anymore in current modern browsers [81]. Liberatore et al. [61] introduced the first use of classifiers to predict webpages over the SSL protocol. Based on this preliminary work, Miller et al.[69] and Cai et al. [20] construct Hidden Markov models to utilize the link structure of a website in combination with supplementary features such as the sizes and order of HTTPS web objects to predict the browsing path of the end user. This is certainly useful for social profiling in small to medium sized websites. Unfortunately, social media platforms (which are a large source of information for social profiling) contain millions of webpages which makes these attacks infeasible in a closed world scenario. Furthermore, the ability to fingerprint webpages over encrypted tunnels such as SSH and Tor has been researched extensively [80, 64, 81, 46].
Recent work and currently a state-of-the-art technique to fingerprint HTTPS webpages (originally developed for Tor hidden services) is k-fingerprinting developed by Hayes et al. [45]. Their work extends the approach of Kwon et al. [58] and is also suitable over HTTPS.

The classification of webpages is implemented using random forests and their experiment produces a TPR of 87% with a world size of 7000 unmonitored HTTPS webpages and 55 monitored HTTPS webpages by using the ordering, timing and size of TCP packets without the need to identify the actual web objects. However, due to the focus on Tor, it is assumed that caching is turned off which is not common for an average user browser session.

Several concepts from other fields in CS have been applied to develop new WFP attacks such as P2V from Al-Naami et al.[11]. Furthermore, the longevity of fingerprints is an important element in developing a robust attack. Shi et al.[92] fill that gap by identifying webpages that undergo frequent modifications, utilizing Haar Wavelet transformations over a VPN tunnel.

Nevertheless, defenses against WFP attacks have been designed to reduce or completely nullify the precision and sensitivity of these experiments. Dyer et al.[33] evaluates different methods to add padding to each packet in order to avoid the possibilty of selecting packet size as a main feature. Specifically designed for HTTPS, Luo et al. [65] propose a defense in the form of a client-side proxy which implements several countermeasures to make it difficult for an adversary to use features such as timing, flow and size. Their defense uses the HTTP Range header to request parts of the HTTP content multiple times instead of requesting the entire content at once. Furthermore, it injects junk data to the content in order to cover up the real traffic data. Additionally, HTTPOS modifies the MSS option in the TCP protocol to limit the size of an outgoing TCP packet.

Countermeasures such as Camouflage proposed by Panchenko et al. [81] is a method to confuse WFP attacks by randomly requesting existing dummy webpages during the request of a legitimate webpage coordinated by the end user. Such mitigation has the advantage of explicitly generating false positives and is generally easy to incorporate in existing client-side proxies.

Other defenses are much more deceptive such as "Traffic Morphing" proposed by Wright et al. [109], where they provide a theoretical approach to transform the distribution of packets of a traffic trace to another distribution in such manner that it resembles a different webpage. More recently, Cai et al. [18] have extended the approach of Dyer et al. [33] to devise a countermeasure called CS-BuFLO and subsequently the more efficient version Tamaraw [19]. This defense transforms a stream of original TCP packets to a continuous flow of fixed size packets to reduce the variance in timing and size of the original packets. Akin to the aforementioned defense, 'Walkie-Talkie' is a similar approach proposed by Wang et al. [105] where they greatly improve upon bandwidth and security by devising a method that sends packets in short bursts and is currently regarded as a state-of-the-art defense. Furthermore, an implementation of HTTP/2 with multiplexing enabled is able to defeat WFP attacks that rely on the ability to identify web objects as demonstrated by Morla [74] considering that it is then problematic to predict the size of each web object.

More recently, instead of manually selecting features to design WFP attacks, Rimmer et al. [87] have developed a process that allows an adversary to automatically select features using deep learning algorithms.

In table 1, we present a taxonomy of webpage fingerprinting attacks over HTTPS in the

context of social media platforms. We have also added our own method which is discussed extensively in section 4.

- **Features**: Main packet features used in the given method. The direction of packets is included in all methods.

- **Open world**: WFP attacks that are validated by having 2 seperate classes, one monitored and one unmonitored class of webpages [23]. The algorithm that performs the prediction can distinguish between those 2 classes. If it predicts that the webpage belongs to the monitored class, it can then also correctly predict which webpage from the monitored class it belongs to. We call this 'complete' when we can identify the webpage out of all other unknown webpages, without creating a seperate class. To the contrary, in a closed world scenario, we only have one class with all the fingerprinted webpages. The assumption in this scenario is that we *exclusively* make predictions with data that we know is coming from fingerprinted webpages/websites.

- **Caching**: This defines whether browser caching is enabled during the collection of the traffic trace. In the context of HTTPS for normal use (e.g. consumers at home), caching is generally enabled to improve performance, resulting in a different behaviour of data communication.

- **Dynamic webpages**: This defines whether the method is developed to cope with dynamic webpages that undergo frequent changes. For instance, social media or news platforms.

- **Browser**: 'Fixed' means that the webpages in the traffic trace, are all generated by a browser *known* to the attacker.

Table 1: Taxonomy of HTTPS webpage fingerprinting attacks in the context of social platforms, viable today.

Method	**Features**	**Open world**	**Caching**	**Dynamic webpages**	**Browser**
Di Martino et al.	Size	Complete	Possible	Possible	Variable
Hayes et al. [45]	Size,time	Partial	No	No	Fixed
Sun et al. [96]	Size	Partial	No	No	Fixed
Miller et al. [69]	Size	Partial	Possible	?	Fixed
Shi et al. [92]	Size,time	No	?	Possible	?
Al-Naami et al. [11]	Size	No	No	No	Fixed

To conclude this section, we refer to an extensive and critical evaluation of the various WFP attacks and their countermeasures published by Juarez et al. [52].

3.2 Smartphone fingerprinting

Smartphone fingerprinting is a field in traffic analysis that has known an exponential rise of popularity over the last few years. It is estimated that 2.5 billion people in the world have a smartphone. These devices contain a massive amount of personal information about the owner. The type of smartphone or the apps that are installed on a smartphone is therefore valuable information for a malicious adversary.
Early work of Dai et al. [27] has (arguably) shown the need for traffic analysis on smartphones for network operators. In this work, they extract fingerprints from Android apps that make use of the HTTP protocol and can then later identify these apps in a newly generated sample of network traffic. Later, Miskovic et al. [70] have extended this approach by developing a more stable and self-learning variant.
Due to the steady increase in HTTPS, Taylor et al. [99] have proposed a new method to identify apps in a stream of encrypted HTTPS traffic by extracting several statistical features and then classify those to each app. In their practical experiment, they achieve generally promising results with an accuracy of 39% and 30.4% for respectively a different device and a different app version than the one used in the training stage. Higher F1 scores are achieved when utilizing the same device and app version. Unfortunately, their method is only valid in a closed world scenario much like most of the previously discussed website fingerprinting attacks. Mongkolluksamee et al. [73] also obtain large F1 scores between 81% and 96% depending on the background traffic by training on packet size and graphlets for an experiment performed on a small scale in a closed world scenario. Other smartphone fingerprinting methods achieve similar results, albeit with different techniques and scenarios [9, 95]. Most of these techniques perform poorly on different devices and app versions, therefore limiting the use of these methods in practical situations.
On the other hand, identifying what actions a victim is performing in an app is a sensitive leakage as well. Conti et al. [25] have developed a method using supervised learning that identifies a limited set of actions on a small collection of predefined apps with an average F1 score of 90% depending on the app, assuming that the device and app version do not change. Saltaformaggio et al. [90] propose a method with a similar goal , although it relaxes the assumption that the victim is using only a single wireless network (*transient connectivity*). Fingerprinting methods that adhere to this property can be utilized in cellular networks where it is likely that the victim is constantly repositioning himself.
Given that a large number of smartphone fingerprinting methods do not perform well on changing scenarios such as having a different Android device or using different app versions, the feasibility to conduct these attacks in realistic scenarios are inadequate and should be explored in further research as shown by Taylor et al. [99].

3.3 Cross-origin fingerprinting

In section 2.1.2, we have discussed a method to identify the exact size of HTTPS requests and responses by analyzing a network traffic trace. Such method is only useful in passive traffic analysis, where we analyze what the victim is 'doing' on the internet during normal

behaviour (for instance, the IUPTIS attack discussed in section 4). A disadvantage for the victim for such a passive MITM is the difficulty to detect if an adversary is present.
In contrast, there are also active traffic analysis techniques in which we deliberately let the victim execute certain actions and then analyze the responses of those actions [100]. For instance, an adversary somehow injects Javascript code on a website and the victim's browser then executes the Javascript code when he is visiting the original website. The execution of the code will result in HTTP requests and responses sent over the network which can then be passively analyzed by holding a MITM position. Injecting Javascript code can be done in numerous ways like for instance, advertisements, setting up malicious websites, or website vulnerabilities like Cross Site Scripting (XSS) [78] and Clickjacking [79]. The actions (in this case, HTTP(S) requests) are often sent to legitimate servers (unrelated to the initially requested webpage). The HTTP responses that correspond to these initial requests might contain personal information about the victim. These HTTP requests are named 'cross-origin' requests because they are from a different origin then the original requested webpage.
A simplified example: lets assume the adversary is a social Wi-Fi provider and has setup a webpage that victims have to visit before they can use the Internet. The browser of the victim requests the malicious webpage and inside the webpage, there is Javascript code that requests the url 'myProfile.html' of a social media platform. The cross-origin request is sent over the Internet to the social media platform (SMP), then the SMP sends a response and finally, it reaches the victim's browser. If the victim is currently logged in to the SMP, then the cookies of the victim that are linked to the SMP are also sent together with the request. As a result, the SMP response (which is sent over the network) will contain the personal webpage 'myprofile.html' of the victim.
It is clear that allowing these cross-origin requests is a risky endeavour. Fortunately, cross-origin requests can be blocked by using the *Same Origin Policy* [89] or can be partially allowed by the HTTP header '*Access-Control-Allow-Origin*'. In the past, many developers have tried to circumvent these defenses using several techniques [2].
However, the Fetch API [5] allows any webpage to perform 'cross-origin' requests with limited functionality. Using the Fetch API, an adversary cannot access the HTTP response received by the SMP, but can only know *when* the response was received.
A high level overview of this concept is shown in the figure below:

[5]Introduced by Mozilla. Implemented since Firefox 39 ,Chrome 42 and Edge 14

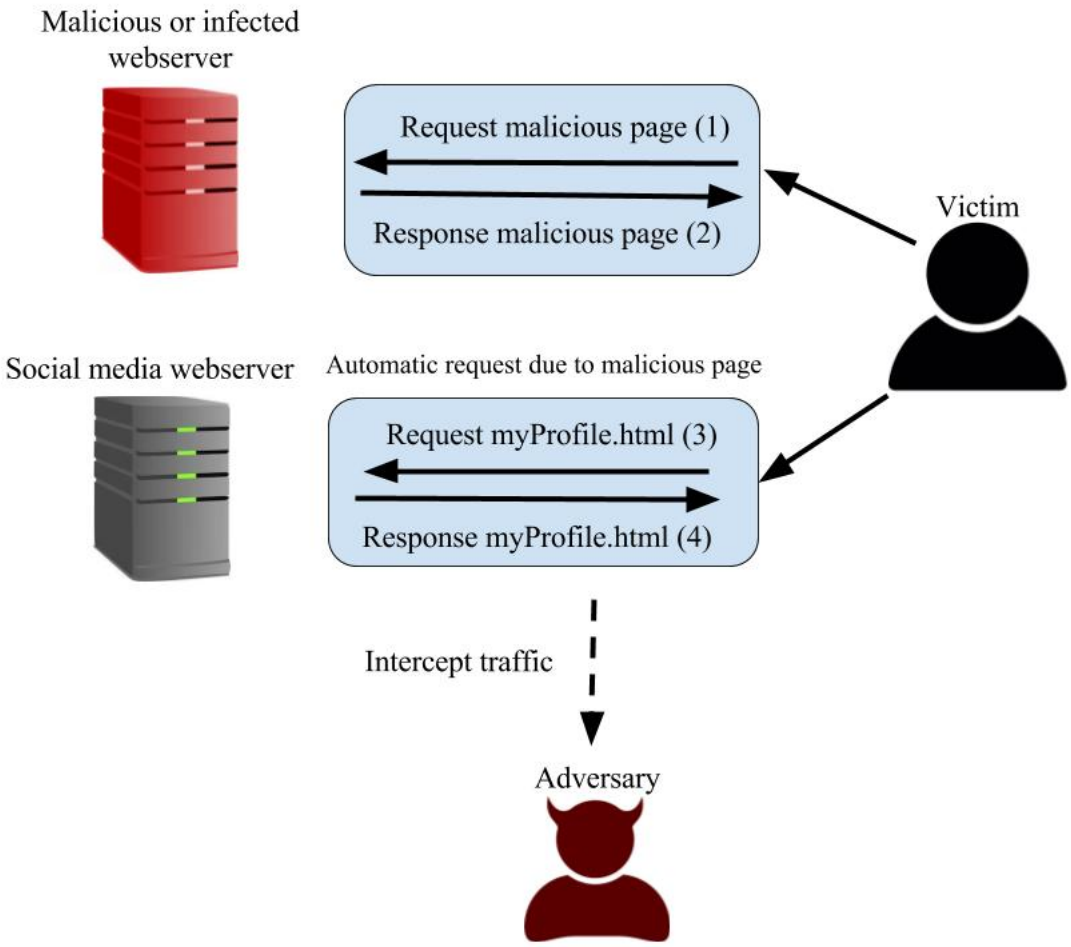

Figure 4: Fingerprinting with cross-origin requests

Van Goethem et al. [100] have proposed several methods to fingerprint a victim using such cross-origin requests in a social media context. In their *Request and Conquer* attack, they show that cross-origin requests can be identified and analyzed with a MITM position, in several protocols (among other things: HTTP2 over TLS, HTTP over TLS and WPA2). In their provided example, they (cross-origin) request 5 personal webpages from the social media platform 'Twitter' and based on the length of the corresponding responses, they predict the name of the victim that is logged in. A theoretical accuracy of 97.62% is reported for 500 000 fingerprinted Twitter profiles. Despite the high viability and interesting concept of the attack, they did not discuss the practical implications that arise when executing this attack in a realistic scenario. First of all, the requested personal webpages (HTML files) which are among other things, the 'following', 'likes' and 'followers' list are very dynamic in content and can change substantially over time, because each victim can modify parts of these pages by following or unfollowing other users. Even if the victim does not modify anything, the HTTP response can still have additional headers (for instance,

X-Response-Time) or modified HTML code that will influence the total length of the page content. As a consequence, if the adversary is not using the latest fingerprints for each user or if one of the users in the 'follower' list or 'following' list change their biography, the fingerprinted sizes do not exactly match the extracted cross-origin sizes. Furthermore, we have found out that if the victim is using Firefox Quantum and a TCP connection is already established between the targeted website (for instance,in another tab), the webpage requested by the Fetch API will go through the existing TCP connection. This complicates the problem of predicting the exact length of the HTTP response, especially when the HTTP/2 protocol is used. We only observed this behaviour in Firefox Quantum. As of December 2017, Google Chrome and Microsoft Edge do not exhibit this behaviour.
Nevertheless, a working defense mentioned by the same researchers are 'Same-Site' cookies. This type of cookies are prevented from being sent in cross-origin requests which makes the requested webpages look like if the victim is currently not logged in to the SMP. Unfortunately, as of November 2017, this functionality has not been implemented yet by Edge and Safari.

3.4 Social media profiling

Social media is without a doubt a large part of modern communications nowadays. Online business use social media to increase their sales [44] and even has a global influence on society as a whole [37]. Users often input personal information into their social media platform for the gain of social interactions or other advantages (e.g. winning prizes). However, research has shown that many social media users are unaware or at least oblivious to the fact that their personal information is used for other goals as well (e.g. predicting health conditions, forensics, ...) [40] [34].
In the case of stalking or forensics, there is a large collection of online tools that combine public available data to construct meaningful information about the behaviour or interest of a user [63] [54]. Even in job recruitments, there are controversial tools like Klout that are used to assess social influences that candidates might have [68].
For advertising, social media platforms often offer unique systems for advertisers in which they can target very specific groups of users in order to maximize their market reach.
Nevertheless, in most of the cases, the data is extracted and processed from information that is given by the users themselves, which is often preceded by agreeing to some privacy terms.
In addition to the voluntarily shared information, there have been a dozen of data breaches in which personal information of thousands or even millions of users have been leaked. Most breaches are a consequence of web vulnerabilities like for example SQL injections, local file inclusions and remote code executions [13]. Websites like 'Have I Been Pwned' collect these leaked databases and can show victims if their personal information have been leaked by any of the data breaches [49]. Such websites can be used by adversaries as well. For instance, an adversary can input any email address and check whether or not the email is leaked in one of the data breaches. Even though at first sight, the website only shows

whether or not the email has been leaked, it also provides these leaked databases free of charge for 'educational' purposes.
As a consequence of all this, social media platforms have shown that their current defenses are not sufficiently strong enough to protect the privacy of their users in a never ending battle.

3.4.1 Bug Bounty Programs

As web vulnerabilities are exploited more and more , the consequences discussed above have a serious impact on the privacy of Internet (especially social media) users. Initially invented by Netscape in 1996 [24] and popularized by web plaforms in the late 2000's, bug bounty programs (BBPs) are initiatives that allow external IT savy users to report web vulnerabilities to the appropriate vendor and in return, they get recognition or monetary rewards.
The main advantage for vendors that introduce BBPs is that they can find and fix vulnerabilities much more cost effective than using internal security engineers or researchers with the end goal of reducing data breaches. On the other side, BBP participators are happy with the recognition or substantial monetary rewards [15] and on top of that, their skillset is trained by finding such vulnerabilities. Very little research has been done into analyzing the impact on data breaches or even BBPs as a whole.
Many social media platforms have introduced individual bug bounty programs: Facebook, Twitter, Twitch, Google+, Yammer, etc ... . From 2011 to 2016, Facebook has paid more than $5 million to a total of 900 participators [3]. Even much smaller companies that solely offer recognition by using a Hall Of Fame are having a significant amount of participators (for instance, Soundcloud has listed 116 participators since 2012 [4]).
However, without a doubt, managing a BBP takes a lot of work. In response, companies like Bugcrowd[17] have introduced platforms where individual companies can manage their BBP more easily and more cost-effective. Additionally, there are companies that manage *private* BBPs in order to limit the program to a more qualified group of participators [97].
It is clear that BBPs are popular among most modern tech companies. Unfortunately, implementing such open initiatives brings implications with tracking reports, paying out rewards and researching the reported vulnerabilities [111]. For instance, of all vulnerabiliy reports that Facebook receives, only 4% are valid. With BBP managing companies like Bugcrowd, 18.5% of the reports are valid [111]. Laszka et al. show that many of the invalid reports are either a misunderstanding of the current scope of accepted vulnerabilities or the inexperience of some participators [59]. These invalid reports can be a risk when a participator cannot clearly explain the vulnerability as shown in [88], where a participator fails to disclose the vulnerability correctly and then tries to exploit the vulnerability publically to get the attention of the company. Laszka et al [59] also propose a theoretical model to help reducing the amount of invalid vulnerability reports which in turn, greatly decreases the cost of maintaining such program.
Finally, another disadvantage are malicious participators. These participators view a BBP

as a free *legal* entrance to conduct malicious activities [29]. It is clear that many current BBPs need an improved concept in order to better safeguard the security systems of the company and most importantly, protect the information of the SMP users.

4 IUPTIS: Identifying User Profiles Through Image Sequences

In section 3.1, we have discussed state-of-the-art techniques to fingerprint webpages over HTTPS. Unfortunately, in the context of social profiling, none of these attacks can be utilized to accurately predict URL's over HTTPS in a realistic scenario, such as one where the browser is unknown and caching is enabled. Many researchers also argued that (even outside social profiling), most attacks are infeasible to conduct in a real world due to the closed world scenario and the lack of background traffic [23, 80].
To close this gap in state-of-the-art techniques, we propose a novel method to identify profile webpages in the form of URL's based on analyzing a network trace intercepted by a passive MITM.

4.1 Adversary Model

In the interest of a practical and reliable WFP attack, we would like to lay out some assumptions that are made:

- The adversary has a network traffic trace from the end user during the period in which they navigated to the webpage profile. Such traffic trace can be extracted with any passive MITM attack.
- The communication between the targeted online platform and the end user is handled by the HTTP/1.0 or HTTP/1.1 protocol encapsulated in TLS records. In other words, the end user is visiting the profile webpage over the HTTPS protocol.
- Each individual *profile* page may be accessed by an individual URL where distinctive and unique images are the main source of information on the webpage of that profile. A profile is associated with a person (e.g. social network pages) or unique entity (e.g. hotel pages). The images on each webpage profile have to be large enough ($>$ 8 Kb) and usually larger than other resources (for instance, stylesheets) on the same webpage, to achieve acceptable results.
- The headers of the TCP and IP layer of the traffic trace are not encrypted and thus may be analyzed by the adversary.

Adversaries that adhere to these assumptions come in many forms. Social Wi-Fi providers and government agencies may essentially hold a MITM position and can therefore apply

these techniques for respectively personalised advertisements and surveillance. Moreover, the introduction of WiFi4EU [106] will boost the number of accessible Wi-Fi access points and in turn, increase the attack surface to perform MITM attacks. In a similar fashion, social networks such as Facebook may provide VPN tools like Onavo [82] that still have access to HTTPS payloads with the ability to correlate the data with their own collection of online profiles.

4.2 Fingerprinting Images

Profile pages often contain several images that are uploaded by the owner of the page. These images are often the largest part of the page content and are most likely unique over the whole platform. The uniqueness of these images is very convenient to select as a feature for WFP attacks. When visiting such a profile page in a browser over HTTPS, the images will be downloaded in several TCP connections. As we are using HTTP over TLS, the actual content of the images is encrypted and thus not visible. However, the complete HTTP request and response sizes are not encrypted and can therefore be calculated easily [64, 22].
Extracting the absolute raw size of each image contained in a HTTP response is not trivial due to the addition of HTTP headers, which are often dynamic in length. HTTP headers are the largest overhead in size that we have to eliminate in order to get the absolute size of each image. However, it is possible to deterministically model the appearance of these headers in each request or response. For each image contained in a HTTP response, we formulate the following equation that defines the total size of such HTTP response:

$$\text{Resp}_x = w_{\text{resp}} + p_{\text{resp}} + i_{\text{resp}} \tag{4}$$

Here, w_{resp} is the length of all HTTP headers (including the corresponding values) that are dependent on the webserver that issues the response to the browser. For instance, the header "Accept-Language" or "Server" is always added by the webserver (online platform) independent of the image that is requested or the browser that is used. Considering that we are targeting a specific webserver and leave out the presence of a *Content Distribution Network* (CDN) in the middle, the value of w_{resp} can be calculated easily.
p_{resp} is the length of all HTTP headers (including the corresponding values) that depend on the image requested. For instance, the "Content-Type" and "Content-Length" header can be different for each image requested from a given webserver and is independent of the browser that is used
i_{resp} is the length of the complete HTTP response body. In our case, this only contains the raw data of the image requested.
In a similar fashion, we also formulate an equation that defines the total size of HTTP request of a web object image:

$$\text{Req}_x = p_{\text{req}} + b_{\text{req}} \tag{5}$$

Similar to the response, the variable p_{req} is the total length of all HTTP headers that are dependent of each requested image. Examples are the GET path in the first request line

and the "Referrer" header.
b_{req} is the length of all HTTP headers (including the corresponding values) that are dependent on the browser that issues the request. For instance, the "DNT" or "User-Agent" may be different for each browser.
Since we would like to fingerprint webpage profiles based on the images that they contain, we have to determine the total size for each image. Then, based on the calculated values, we use the collection of all the images contained in a profile page to construct a fingerprint for the whole profile webpage.
We will extract the fingerprint of an image from the corresponding HTTP request and response sizes as follows:

$$\text{Img}_y = (p_{\text{req}}, \text{Resp}_x) \tag{6}$$

To construct our fingerprint database for this preprocessing stage, we develop a fingerprint for each profile webpage x with n images where the approach is similar to Lu et al.[64] ordered sequence:

$$\text{Profile}_z = \langle(\text{Img}_0, \text{Img}_1, \ldots, \text{Img}_{n-1}, \text{Img}_n)\rangle \tag{7}$$

A practical consideration that arises is the fact that the variable b_{req} is unknown and is most likely to be different for various browsers. It is therefore necessary to either figure out the browser that the end user is utilizing in order to estimate the variable [50, 21] or to set the variable b_{req} to a fixed size. The latter option will result in a trade-off with a lower precision of the WFP attack as we will show in Sect. 6. Likewise, w_{resp} may be hard to predict due to the variation of this variable within the same browser. More specifically, the usage of a CDN may introduce HTTP headers with irregular sizes, usually dependent on whether or not the requested image has been cached by the CDN. A possible solution for this concern is provided in Sect. 4.6.1 where we employ a single dimensional clustering method called 'Jenks optimization method'.

4.3 Constructing a Request/Response List

In order to build an ordered list of sizes that correspond to the HTTP requests and responses in our intercepted traffic trace, we have to utilize a practical approach to transform a raw network traffic trace to such list that we call a Request/response list (RRL).
A request/response list of length n is an ordered list that contains the timestamp of each HTTP request (T_n), the size of each HTTP request and HTTP response *associated* with a web object image:

$$\text{RRL} = \langle(R_0, R_1, \ldots, R_{n-1}, R_n)\rangle \tag{8}$$

$$R_n = \langle(T_n, \text{Req}_n, \text{Resp}_n)\rangle \tag{9}$$

When the end user navigates to a profile webpage, several TCP connections to the webserver will download the resources located on that particular webpage. These resources

consist of images, stylesheets, source files, etc.... Due to the encryption provided by HTTPS, an adversary cannot trivially identify the responses that contain images. Therefore, similar to the approach of Lu et al. [64], we use the fair assumption that images are usually larger in size than other resources and filter out all other resources that are below a fixed threshold. Depending on the targeted platform, this threshold is usually set to a value between 5Kb and 10Kb.

4.4 Building a Profile Prediction List

At this stage, we have collected the necessary image and profile fingerprints and have constructed a RRL based on the intercepted traffic trace.
Subsequently, we evaluate whether each R_x in our RRL matches one or more image fingerprints Img_y of any Profile_z where x, y and z define any element. Given R_x, Img_y and Profile_z:

$$R_x = \langle (T_x, \text{Req}_x, \text{Resp}_x) \rangle \tag{10}$$
$$\text{Profile}_z = \langle \dots, \text{Img}_y, \dots \rangle \tag{11}$$
$$\text{Img}_y = (p_{\text{req}}, \text{Resp}_y) \tag{12}$$

The matching is performed i.f.f. the following equations hold:

$$\text{Req}_x \in [(p_{\text{req}} + b_{\text{req}}) - \pi_{\text{req}}, (p_{\text{req}} + b_{\text{req}}) + \pi_{\text{req}}] \tag{13}$$
$$\text{Resp}_x \in [\text{Resp}_y - \pi_{\text{resp}}, \text{Resp}_y + \pi_{\text{resp}}] \tag{14}$$

In the equation above, we perform the matching if the request and response sizes lie within an interval defined by 2 newly introduced parameters, π_{req} and π_{resp}. Both parameters are defined as the statistical variance for respectively the request and response size and should be chosen with the intended platform and browser in mind. If the specific browser that is employed is known (for instance by extracting the User-Agent header from an unencrypted HTTP request), b_{req} can be calculated to be very accurate and thus requires a low π_{req}. Similarly, π_{resp} depends on the accuracy of b_{resp}. In the experiments discussed in Sect. 6, we show that a large π_{req} and π_{resp} are still sufficiently robust enough to achieve favorable results.

Each match will construct a P which is then appended to the *Profile Prediction List* (PPL). All elements in the PPL are chronologically ordered based on the timestamp T of the matched RRL element. The PPL is constructed as a 2D array with variable size in the second dimension. The first dimension defines each matched RRL element (ordered by the timestamp) and the second dimension defines the error images from a given profile that are assigned to each matched RRL element (equation (20)):

$$\text{PPL}[0] = \langle P_0^0, P_1^0, P_2^0, \dots \rangle \tag{15}$$
$$\text{PPL}[1] = \langle P_0^1, P_1^1, P_2^1, \dots \rangle \tag{16}$$
$$\text{PPL}[\dots] = \langle P_0^{\cdots}, P_1^{\cdots}, P_2^{\cdots}, \dots \rangle \tag{17}$$
$$\text{PPL}[n-1] = \langle P_0^{n-1}, P_1^{n-1}, P_2^{n-1}, \dots \rangle \tag{18}$$
$$\text{PPL}[n] = \langle P_0^n, P_1^n, P_2^n, \dots \rangle \tag{19}$$

We construct P_g^h as follows:

$$P_g^h = \langle \mathit{profileName}, \epsilon_{\text{req}}, \epsilon_{\text{resp}} \rangle \tag{20}$$
$$\epsilon_{\text{req}} = \text{Req}_x - (p_{\text{req}} + b_{\text{req}}) \tag{21}$$
$$\epsilon_{\text{resp}} = \text{Resp}_x - (p_{\text{resp}} + i_{\text{resp}} + w_{\text{resp}}) \tag{22}$$

In other words, we create a new object P_g identifiable by the profile name that contains the error (difference) between the request/response size and the corresponding fingerprinted image of that profile.

4.5 Finding a Valid Profile Sequence

After the creation of our PPL, we attempt to identify an uninterrupted sequence with length Φ in the PPL starting at any X such that PPL[X],PPL[$X+1$],PPL[$X+\dots$],PPL[$X+\Phi$] in the sequence have respectively **a** P^X,P^{X+1},$P^{X+\cdots}$,$P^{X+\Phi}$ such that they all have the same $\mathit{profileName}_q$:

$$P^X \in PPL[X] \tag{23}$$
$$P^{X+1} \in PPL[X+1] \tag{24}$$
$$P^{X+\cdots} \in PPL[X+\dots] \tag{25}$$
$$P^{X+\Phi} \in PPL[X+\Phi] \tag{26}$$
$$\mathit{profileName}_q \in (P^X \cap P^{X+1} \cap P^{X+\cdots} \cap P^{X+\Phi}) \tag{27}$$

In other words, we have found a valid profile sequence if Φ request and responses in a row are all matched to **a** fingerprinted image of the same Profile_z. Multiple profile sequences may obviously exist. We say that P^X,P^{X+1},$P^{X+\cdots}$ and $P^{X+\Phi}$ form a valid sequence for $\mathit{profileName}_q$.

The introduction of the parameter Φ defines a balance between browser caching and resulting precision and sensitivity. Φ is defined as the minimum streak of HTTP responses in sequence that can possibly belong to the same specific profile. When choosing this value, it is useful to look at the number of images that are exposed on each individual webpage. A large value for Φ will have a more accurate prediction but might reduce the effectiveness of the attack. For instance, if a profile webpage only has 2 images, then a Φ below 3 is

necessary to identify that particular webpage. More importantly, the parameter is also utilized to reduce the impact of browser cached images. For instance, if the end user is visiting the webpage of Profile$_z$ which has 10 images where 5 of those are already cached by the browser, we can still set Φ to a value below 6 in order to succesfully find a valid sequence.

4.6 Evaluating a Profile Sequence

For a small collection of image fingerprints, the resulting profile sequences are already a valuable prediction. However, this is insufficient for larger collections of fingerprints due to the fact that there might be multiple profiles that match to the same sequence. Therefore, we have to measure how similar each element in the profile sequence is, by calculating the standard deviation σ_{req} and σ_{resp} over all ϵ_{req} and ϵ_{resp} in the valid sequence. Afterwards, we evaluate whether or not the standard deviations are below a threshold H_{req} and H_{resp}. (both thresholds are fixed for each online platform). If both deviations are below the threshold, we say that the profile sequence is complete and we then formulate a prediction that the end user has navigated to the corresponding webpage of *profileName*. For instance, assume we have Φ profiles in our valid sequence – P_1, P_2 , $P_{...}$, $P_{\Phi-1}$ and P_{Φ}:

$$\mu_{\text{req}} = \frac{\sum_{i=1}^{\Phi} \epsilon_{\text{req}}(P_i)}{\Phi} \quad \text{and} \quad \mu_{\text{resp}} = \frac{\sum_{i=1}^{\Phi} \epsilon_{\text{resp}}(P_i)}{\Phi} \tag{28}$$

$$\sigma_{req} = \sqrt{\frac{\sum_{i=1}^{\Phi} (\epsilon_{\text{req}}(P_i) - \mu_{\text{req}})^2}{\Phi}} \tag{29}$$

$$\sigma_{resp} = \sqrt{\frac{\sum_{i=1}^{\Phi} (\epsilon_{\text{resp}}(P_i) - \mu_{\text{resp}})^2}{\Phi}} \tag{30}$$

4.6.1 Jenks optimization method

Determining the standard deviations and then comparing it to a predefined threshold is relatively robust considering that b_{req} and w_{resp} remains constant over all images of the same profile. Although, the presence of a CDN will essentially break that assumption by appending additional proprietary HTTP headers such as 'X-Cache' or 'X-Amz-Cf-Id', when the requested image was cached by a CDN server.
In the interest of distinguishing cached images from uncached images or at least reduce the effect on the standard deviations, we employ the Jenks optimization method. This optimization method (also known as '*Goodness of Variance Fit*') clusters a 1D array of numbers into several classes with minimal average deviation from each class mean.
For our IUPTIS attack, all ϵ_{resp} of each P_g^h (Eq. 20) in a valid sequence will be clustered

into 2 classes (CDN-cached and uncached images). The integration of this method happens immediately after finding a valid sequence. Following the clustering, we compute σ_{req} and σ_{resp} for each class, which makes a total of 4 standard deviations. However, if one of the calculated classes only contain 1 element, we will have to assume that the single element is a false positive and therefore, fallback to the original method of computing the standard deviations for the whole sequence.

4.7 Recap

Our IUPTIS method (Fig. 6) is composed of the following steps:

1. Intercept a network traffic trace from the end user.
2. Establish the collection of fingerprints by extracting the fingerprints of each targeted profile (Sect. 4.2).
3. Build an ordered Request/Response list (RRL) from the raw traffic trace as discussed (Sect. 4.3).
4. Construct a Profile Prediction List (PPL) by matching the elements from the RRL to one or multiple image fingerprints (Img_x).
5. Find a sequence of Φ elements in the PPL that all contain at least one image from the same $Profile_y$ (Sect. 4.5)
6. Evaluate the formed sequence by our IUPTIS algorithm which decides whether or not the sequence is classified as a valid profile prediction (Sect. 4.6).

Each attack is executed with the following tuneable parameters:

- b_{req}: Expected size of data in a request dependent on the browser.
- H_{req}, H_{resp}: The threshold of the maximum standard deviation for respectively, the requests and responses. Both parameters are fixed for each online platform.
- $useJenks$: Whether or not the Jenks optimization method is applied.
- Φ: Minimum matching sequence or streak of images.
- π_{req}, π_{resp}: Request and response bias that allows matching to an image fingerprint.

The ideal combination of parameters depends on the adversary model, such as whether he wants to allow browser caching or a high precision in trade for a lower sensitivity. Possible combinations are provided in Sect. 6.
In figure 5, we show an example of the IUPTIS method consisting of the first 5 steps. Starting from the bottom, the actual images are downloaded by the browser. This results

into a request and response, each with a specific size. Subsequently, the PPL is constructed by matching the request and responses to one or more profiles from our fingerprinting database. Then, we find an uninterrupted sequence of at least length Φ, which is 'Profile C' in our case. Finally, the sequence is evaluated to be fit for a valid profile prediction.

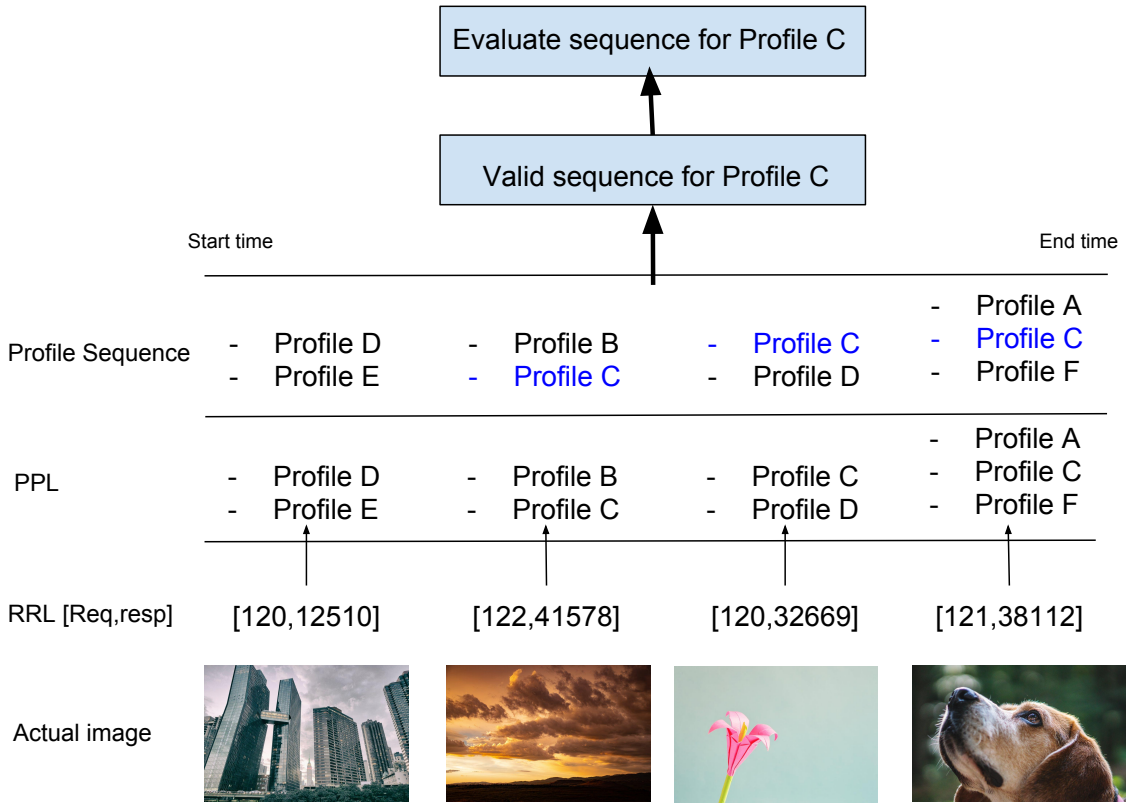

Figure 5: Visual overview of the first 5 steps of the IUPTIS method. Assume Φ is 3 and our fingerprint database consists of Profile A to Profile G.

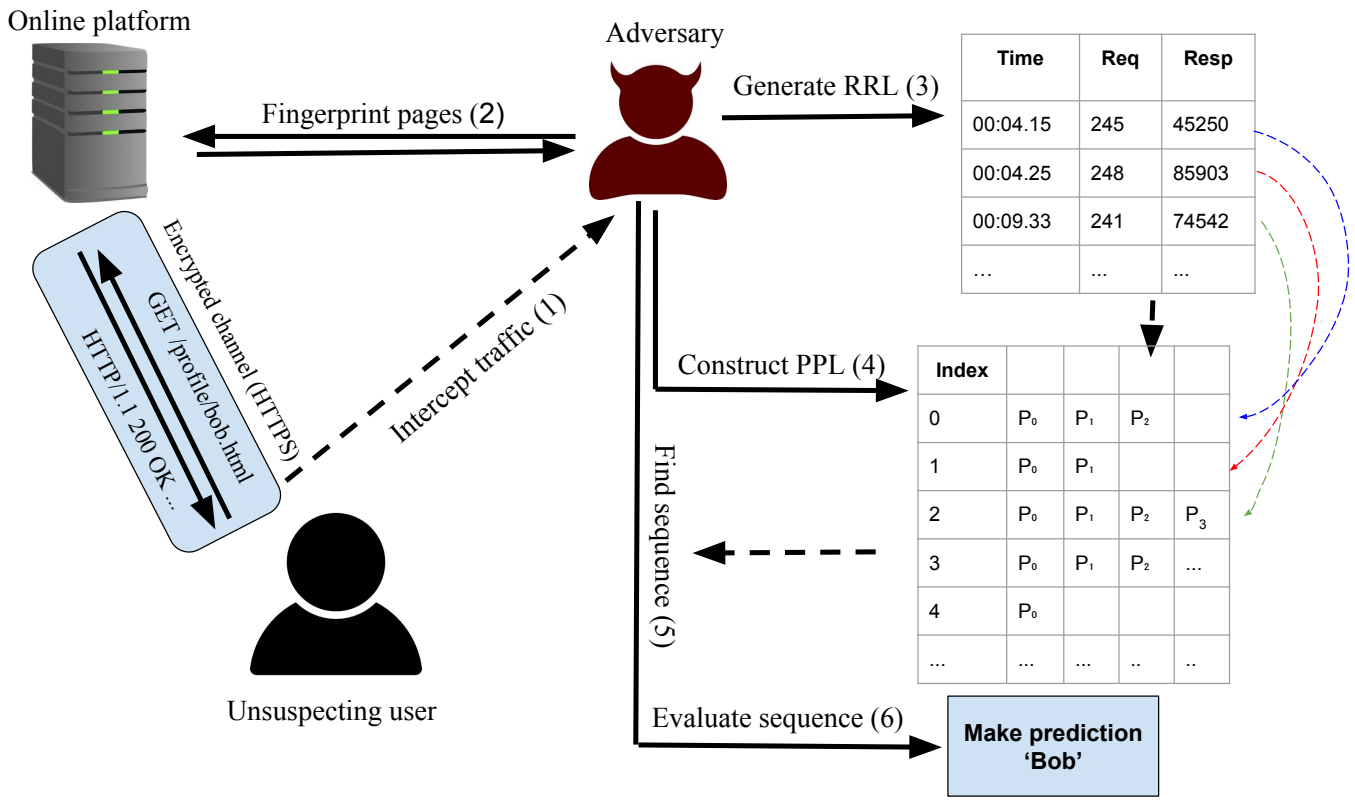

Figure 6: Visual overview of the IUPTIS method

4.8 Practical considerations

4.8.1 Fine-tuning parameters

When choosing the value Φ, it is useful to look at the number of images that are shown on each individual webpage. E.g. when using the Instagram Android app, it shows 9 images for each profile. As a result, Φ cannot be larger than 9. A high value Φ will have a more accurate prediction but might reduce the effectiveness of the algorithm. E.g. if a social media page from a specific user only has 2 images, then a Φ below 3 is necessary to identify the profile.

Individual social media pages often show individual images in the chronological order in which they are uploaded by the user. If we assume that all X individual images from a profile are cached by the browser before the start of our victims traffic trace (which means that we cannot match the HTTP response to an image) and another Y new images are uploaded by the user, then these Y new images are downloaded in sequence at the end or the start of the X cached images when navigating to the webpage of this profile. As a result, we can still find a valid sequence in our PPL if Φ is smaller or equal than Y.

So we conclude that in an ideal scenario, the user starts a fresh browser session to clear the

cache at the beginning of the trace, but it is in no circumstances necessary to perform the attack. If there are a sufficient number of images on the webpage, then browser caching will only slightly affect the effectiveness of the method as we will show in our experiment at section 6.1.
π_1 and π_2 define an upper and lower bound of variance that indicates whether or not a $Profile_x$ can be assigned to a specific R/R pair. If π would be set to $+\infty$, we would have to calculate the σ for every possible combination of fingerprinted and received request/response pair, which is of course infeasible.
Finally, H_{req} is used to filter out predictions where the corresponding HTTP requests are too different in length. H_{resp} does the same for the corresponding HTTP responses. We use the standard deviation with the logic that the length of HTTP headers when requesting images or responding with the content of images, do not differ greatly for a given webserver. Both boundaries are an important trade-off between many predictions with low accuracy and none to little predictions with high accuracy.
Despite the strict assumptions, there are a considerable number of social media platforms and general online platforms vulnerable to this attack. We have succesfully experimented with the following platforms: Pinterest, We Heart It, DeviantArt, Hotels.com, Instagram (non HTTP/2 support) and Pornhub. Our attack is also applicable on some video streaming websites due to the thumbnails images generated from videos. In light of the last platform mentioned, this can result into predicting the sexual orientation or preference of the victim.

4.8.2 Predicting HTTPS requests/responses

With the goal of performing our IUPTIS attack in mind, we discuss a variety of methods to predict the length of HTTP requests and responses to images from Content Distribution Networks (CDN), often deployed on social media platforms and online travel platforms. We will use DeviantArt and Hotels.com as an example, but our method can be used for any online platform that complies to our attacker model.

When a user navigates to a webpage, the browser will open multiple TCP connections in parallel to download the resources (e.g. images) requested by the webpage. The resources are often downloaded using HTTP requests to CDNs. An example of such resource can be observed in B.1.
In addition to the known headers like 'Host' or 'Content-Length', other headers are dependent on the browser that is used and the CDN that handles the requests. As a result, each HTTP request or response for a specific resource can be different depending on various factors. Our goal is to predict the length of such HTTP request and response in order to use this information to improve our IUPTIS attack.
First, we discuss HTTP headers that are often used to request images:

- HTTP Status: If an image has been downloaded succesfully, the status code will be 200 (HTTP/1.1 200 OK). As a result, this value is usually fixed in length.

- Host: This is the domain of the server where we request the image from. The length of this value varies depending on the targeted server. In the DeviantArt experiment, the length usually varies 2 to 3 bytes.

- Accept,Accept-Encoding: Most of the time, the 'Accept' value is set to '*/*'. The 'Accept-Encoding' value will depend on the language that the browser is set to. The abbreviation of the country is added which is always 2 bytes in length. This value is usually fixed in length.

- GET url: Most url's that request an image from CDNs have a rather static length. On some social media platforms (e.g. DeviantArt), the name of the image is added to the URL (which we know beforehand). Most hashes that are added are usually fixed in length. In our DeviantArt experiment, we observed a 3 to 6 byte difference in length for a specific image. To the contrary, in the Hotels.com experiment, search queries will add many GET parameters to the URL resulting into a very large range of possible lengths.

- Referrer: We usually know from which webpage the resource is requested. In our DeviantArt experiment, this is *'[USER].deviantart.com/gallery'*. As such, the length can be predicted easily. However, in our Hotels.com experiment, there are several GET parameters added to this URL. This makes it difficult to calculate the average length.

- Connection-type: Most modern browsers open TCP connections and download multiple images in each connection. In order to do this, this value should be set to 'keep-alive'.

- User-Agent: This value is the bottleneck of our prediction. Fortunately, fingerprinting a browser or User-Agent header is well researched [50] [76] [21] and is usually predicted by analyzing the communication between browser/device and server. Another simple solution is to extract a User-Agent header from an unencrypted connection to a webserver (which still happens very often). It is however important to note that the User-Agent is usually the same for each combination of browser and device. Once we get the User-Agent from a specific browser version/device, we can then use this value forever.

As shown above, the length of a HTTP request in our scenario can be calculated fairly accurate. Except from the GET and Referrer url, all other headers are usually dependent on the browser and not on the specific profile that is requested. On top of the headers discussed, there might be other headers that are added by specific browsers (e.g. DNT). These headers are usually fixed in length and can often be linked to the User-Agent of the request.
Next, we discuss HTTP responses that contain the image that is requested by the browser:

- Content-type: The content type is usually the same type as the file extension (which we know beforehand). E.g. images with the extension '.jpg' have content-type 'image/jpeg'.

- Content-length: This value is the total size of the image that we have requested (which is known beforehand). For our fingerprinting process, we can even send a HTTP HEAD to extract the content-length without downloading the complete image.

- Connection-type: Same explanation as in the HTTP request.

- Date: The date is always fixed in size.

- Cache-Control: This value depends on the server handling the request. In the scenarios we encountered, we discovered that the length of this value is the same for each image on a given webserver.

- Expires: If this header is added to a response (which depends on the targeted server), this is usually consistent and also has the same length as the 'Date' header.

- X-Cache: Specifically used by CDs (non-standard HTTP), this response header depends on the CDN handling the initial request. Depending on whether or not an image is cached, this header has a different value. E.g. DeviantArt uses CloudFront, which means that the header will be either 'Miss from cloudfront' or 'Hit from cloudfront'.

- Age: This header is added if the image is found in the cache of the corresponding CDN. In other words, the addition of this header and its corresponding value depends on the 'X-Cache' header. The length of this header is hard to predict. In the DeviantArt example, we combine this header with the length of the 'X-Cache' header and take the average of the total length.
  This header can be also found in responses that go through a proxy, which further complicates the prediction.

- Via: This header is often added by CDNs. Fortunately, the length of this value is rather fixed.

- X-Amz-Cf-Id: This value is usually fixed in length and is used by Amazon's Cloudfront. Similar headers can be added by other CDN's.

- E-tag: Used by web caches (or CDNs) which allow conditional requests. This value is usually fixed in length, but can also be variable for CDNs of for instance, Akamai. The average length of this header is preferred to be used.

It is clear that most headers greatly depend on the server (often a CDN) handling the request. Despite the dependencies, we can still predict the length of such responses to a certain extent. Furthermore, it is important that we predict this length as accurate as

possible in order to reduce false positives in our IUPTIS attack.
Using this technique, it is almost impossible to perfectly predict the length of such HTTP request or response. This is why we introduce a lower and upper bound variance (π_1 and π_2) in our IUPTIS attack, to construct an interval where the length of the actual HTTP request/response is an element of that interval. The value of these bounds will mostly depend on how accurate we can predict our length. If we know the exact value of the User-Agent header (which has a big influence on the length), we can provide a small bound resulting in more accurate predictions and a better performance when running our algorithm. In cases where the User-Agent header is not known at all, we can provide a large bound with which we can still perform our attack in exchange for a lower accuracy and performance.

Lastly, we assume that each HTTP request and response is sent using a fresh browser session so that the browser cache is empty. In the context of our IUPTIS attack, it is important to know that we match images that are uploaded by users. Our attack only works for images that are downloaded the first time in a browser session because afterwards, cache headers are added which will vary the length of the HTTP requests. The HTTP response will in this case only contain the headers since the actual image data is presumed to be cached by the browser. In most SMPs, we cannot change uploaded images directly which means that the images will stay in the cache until the browser or victim decides to clear it. Despite the inaccurate prediction of cached images in this scenario, we can still predict new images that are uploaded by the user.

4.8.3 Modelling CDN cache behaviour

In the previous section, we have discussed that some response headers that are used by CDNs are unpredictable in length. Usually, the difference in length is small but in some occasions, the total response length can fluctuate strongly. As a result, the calculated standard deviations will be too high to allow adding a profile to the profile prediction list (PPL). This primarily introduces false negatives as shown in the example below:

Assume we have an array with the differences of the fingerprinted and received responses from a given profile with $\Phi = 5$, $H_{\text{resp}} = 1.5$ and $\sigma_{req} = 0$:

$$\begin{pmatrix} 4 & 19 & 20 & 5 & 4 \end{pmatrix}$$

Calculating the standard deviation gives us: 7.4 (bytes).
Such σ is often too high to be accepted as a valid profile prediction. Consequently, the valid sequence of images will be skipped and a false negative is introduced. Obviously, the σ is large due to the second and third element being very different in size. The current IUPTIS method is unaware that the second and third element are actually CDN-cached images which explains the large difference in size. Therefore, for a given sequence of images,

we have to find a method to know which elements in that sequence correspond to cached images [6]. If such method exists, we can calculate the standard deviation (σ) seperately. One for all cached images and another one for all uncached images. Both σ's have to be below boundary H_{resp} in order to be accepted in the Profile Prediction List.
In exchangefor a slight decrease in performance, we propose a solution to reduce the effect of interleaved CDN-cached and uncached images, using Jenks natural break algorithm.
The JenksNB algorithm clusters an 1D array of numbers into several classes with minimal average deviation from each class mean. This clustering method is faster than multivariate clustering methods such as k-means used in k-fingerprinting [45]. In our scenario, we can integrate this method in step 6 of our IUPTIS attack. For our attack, only 2 classes are needed (cached and uncached images).
Returning to the example, we can first use the JenksNB method to split our array into 2 classes:

$$\begin{pmatrix} 4 & 19 & 20 & 5 & 4 \end{pmatrix}$$
$$\text{Class 1: } \begin{pmatrix} 4 & 4 & 5 \end{pmatrix}$$
$$\text{Class 2: } \begin{pmatrix} 19 & 20 \end{pmatrix}$$

Calculating the σ for both classes gives us respectively, 0.47 and 0.5.
The σ's are significantly lower. A usual boundary H_{resp} will be well above both σ's, resulting into a valid profile prediction.
However, side effects arise when an unrelated image is added to the array. Consider the array below with $\Phi = 4$, $H_{\text{resp}} = 1.5$ and $\sigma_{req} = 0$:

$$\begin{pmatrix} -2 & -4 & -3 & 80 \end{pmatrix}$$
$$\text{Class 1: } \begin{pmatrix} -2 & -3 & -4 \end{pmatrix}$$
$$\text{Class 2: } \begin{pmatrix} 80 \end{pmatrix}$$

Without JenksNB and $\Phi = 4$, the profile would be not added to the PPL due to a high σ. With JenksNB, the σ's are respectively 0.66 and 0. Both σ's are within the boundary H_{resp} and thus will be added to the Profile Prediction List. Class 2 has an element with difference 80, this is not a cached image but is actually a false positive for that particular image. It is of course ambiguous to know if an image is cached or simply a false positive. If the Jenks method splits the array into a class with 1 element, the actual result will be the same as if Φ will be lowered with 1. Indeed, a sequence of 3 valid images is found while the 4th image is unrelated.
Such unwanted behaviour can be partially solved by only using JenksNB if each class has

[6] Not to be confused with browser cached images.

at least 2 elements. Otherwise, we use the original method of calculating one σ for the whole array.
The astute reader might notice that an array such as:

$$\begin{pmatrix} -2 & -4 & -3 & 80 & 81 \end{pmatrix}$$
$$\text{Class 1: } \begin{pmatrix} -2 & -3 & -4 \end{pmatrix}$$
$$\text{Class 2: } \begin{pmatrix} 80 & 81 \end{pmatrix}$$

will result into a false positive. It is however important to know that the chance of getting such array is significantly smaller than the previous array. Statistically, it is hard to get 2 false positives (80 and 81) for a particular image where they are both close to each other. We will empirically prove the effectiveness of our proposed solution in the Hotels.com experiment in section 6.2.

4.9 Comparison to State-of-the-art Techniques

Our attack differs from state-of-the-art techniques like k-fingerprinting (k-FP) [45] and the method proposed by Miller et al. [69], in the idea that we specifically target a subset of online platforms, and decouple browser caching and dynamic webpages by introducing several parameters that can be fine-tuned according to the demands of an adversary. In comparison to machine learning (ML) attacks [80, 81, 45] where we have to collect several traces from page loads, our fingerprinting stage only requires one page load for each profile. Numerous strong assumptions made in state-of-the-art methods are relaxed or completely removed in our IUPTIS attack [52]. For instance, the ability to perform our attack on different browsers and devices, without the need to collect session traces from each one individually, is an approach that is rarely proposed. Moreover as we will demonstrate in the experiments, our attack does not assume that we know the end and the beginning of a page load in a given trace, which is shown to be difficult to predict [26, 52]. Although due to such valuable properties, our attack is only applicable over TLS and thus it does not support anonymization services such as Tor. On the other hand, disadvantages of our attack can be found when applying mitigations. Due to the rather deterministic nature of our algorithm, existing defenses can be very effective to mitigate our attack as discussed in Sect. 7. Current state-of-the-art techniques are more resistant against defenses such as padding and have even defeated more advanced defenses such as HTTPOS, CS-BufLo or Tamaraw [45, 20, 53, 19]. Although, as presented in the taxonomy published by Miller et al. [69], most of these WFP methods need to fingerprint unmonitored webpages to make it feasible in an open world scenario [23, 52] and almost all WFP attacks require browser caching to be turned off.
Furthermore, the flexibility of our attack parameters requires manual preliminary work

which involves analyzing the HTTP request/responses and then tuning athese parameters in pursuance of an effective attack. In addition, we address the *base rate fallacy* [52, 104] by carefully formulating our assumptions and adversary model and focussing on precision instead of sensitivity. Subsequently, the IUPTIS technique does not explicitly measure similarities between fingerprinted profiles and thus eliminates the necessity to create a separate collection of unmonitored webpages. As a result, our attack has the valuable property that whenever we increase the world size, only the precision will be affected and the sensitivity will remain practically the same.
During the work of this thesis, a recent article has been published that shows that it is possible to predict a 'like','dislike' or 'match' of a Tinder user based on a combination of HTTPS response size and plain HTTP image resources [6]. Even though, the practical issues of such fingerprinting attack are not discussed and the recommendations proposed in the article are insufficient, the IUPTIS technique is however very suitable for this type of fingerprinting and can be applied to make the Tinder attack practical in a real world scenario.

4.10 Conclusion and Future Work

We have proposed a new webpage fingerprinting technique called 'IUPTIS' that focuses on the practicality in an open world scenario. The ability to use different browser versions and enable caching is an improvement over previous state-of-the-art techniques. Our experiments have generated favorable results with F1 scores between 90% and 98% dependent on the parameters utilized and highlights the privacy impact on various online platforms. However, our attack is only applicable on the HTTP/1.1 protocol due to the assumption that we can infer the exact response size.

Nonetheless, recent work of Wijnants et al. [107] has indicated that some implementations of the HTTP/2 protocol have a rather deterministic approach in multiplexing which might make the estimation of response sizes in such protocol still relatively accurate. Furthermore, we did show the impact of our attack on a small subset of all profiles available on a platform. However, some platforms only have a small collection of online profiles (DeviantArt has 36 million users). The impact of our attack on an even larger scale is unknown and should be explored in future work.
Moreover, our addition of the Jenks optimization method only shows improvements in parts of the experiments where the CDN heavily influences the response size. Further experiments on other online platforms should be performed to analyze how exactly incorporating this optimization method will improve the overall F1 score. Additionally, other metrics for calculating the error (ϵ) for each fingerprinted image in Sect. 4.4, such as the squared difference are not examined yet and might be able to generate superior results.

5 Information leakages in game consoles

Security vulnerabilities of desktop PCs, smartphones, IoT devices and even cars have been thoroughly analyzed and discussed in various publications. Unfortunately, very little research goes into the vulnerabilities and sensitive information that a gaming console can have [32]. Modern game consoles have become much more than only a 'toy'. The constant development of network functionalities has grown at a steady rate in the current generation of consoles. This fast growth has even given rise to research in forensics analysis in order to extract data stored on these consoles[56, 28].
It is imperative to understand that information leakages in game consoles might lead to social profiling too. In the following sections, we will briefly discuss the current state of the Playstation 4 in the context of security and social profilng.

5.1 Playstation 4 (Pro)

The Playstation 4 is a game console created by Sony Inc. [5] and released in November 2013. As of January 2017, 73.6 million units have been sold worldwide [110]. The Playstation 4 (PS4) supports games in either physical or digital format. The latter is managed by the Playstation Network service (PSN) and contains, among other things, the Playstation Store. The communication between the PS4 and the PSN is tightly integrated and is happening regularly. Internally, the PS4 operating system (called Orbis) is running on a customized FreeBSD. However, the console itself is closed source and as such, the inner workings of the system cannot be analyzed accurately. Lastly, it is important to know that the installation of Orbis updates are necessary in order to use online functionality such as the PS Store.
In the following section, we discuss the discovery of several vulnerabilities and weaknesses in the PS4 (version 5.03) that can be utilized to collect sensitive information.

5.1.1 General weaknesses

The communication between the Playstation Store servers and the PS4 is sent through the use of HTTPS. Unfortunately, a large portion of the communication between the PS4 and other parts of the PSN (for instance, update features) are sent over plain HTTP. The use of HTTP in a widespread console is outdated, risky and problematic as we will show with a few examples.
First of all, almost all thumbnail images that are loaded by Orbis are sent through HTTP (see Figure 7). Besides the fact that an adversary with an active MITM can alter these images, there is a much severe issue. The thumbnail URL's for each game are constructed by a string that identifies the game. A simplified example is shown below:

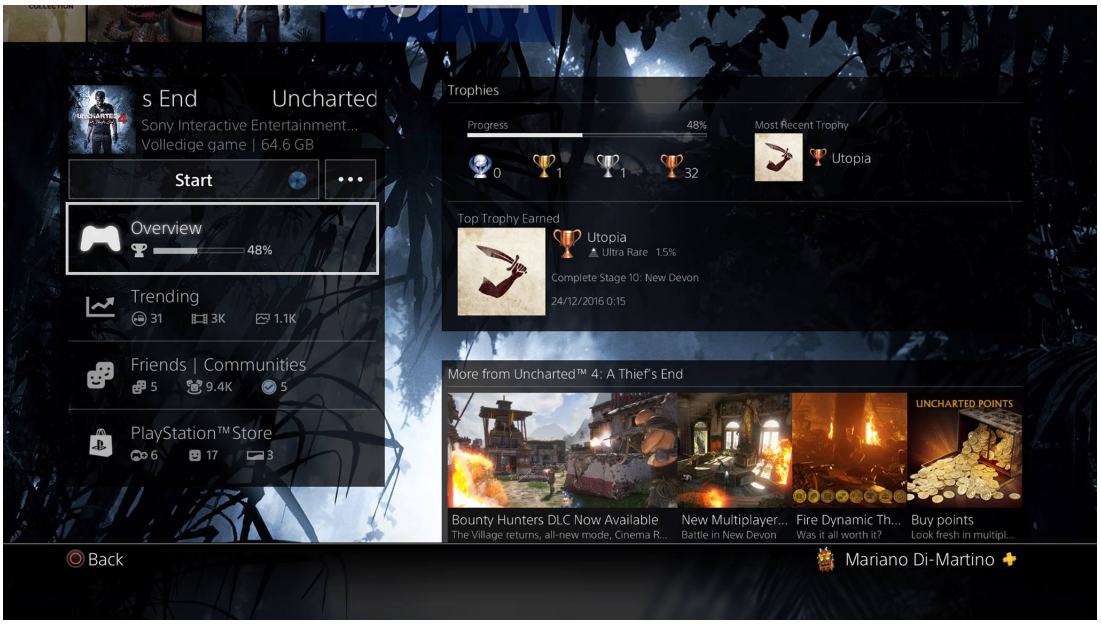

Figure 7: In the right bottom side of the screen, thumbnails from a game are downloaded in plain HTTP.

Table 2: *[CDN]* defines the hostname of the CDN server. *[GAME-ID]* identifies a unique number associated with a PS4 game. *[HASH]* defines an unknown hash. The way the URL is constructed, reveals the origin from where the URL is requested in the PS4 menu.

http://[CDN].playstation.net/cdn/EP0002/CUSA[GAME-ID]/[HASH].png(31)

The URL is also different based on where the image is requested (for instance: the PS Store or the installation menu). In other words, a passive adversary can trivially track the behavioral flow of the victim and can even discover which games the victim has installed. Furthermore, updates for Orbis OS are managed through XML files over HTTP (listing 1). Fortunately, the content of the updates (tag <*image*>) are digitally signed and therefore, cannot be altered. It is, however, possible to make the system believe that a new update is present by modifying the version numbers. If the current OS version is lower than the version number provided in the HTTP response, then the OS will try to download the image tag URL. With this behaviour, we can construct an oracle that will tell us whether the current OS version is lower or higher then a version X. By using a sequence of questions to the oracle, we can predict with 100% certainty what the current OS version is.
Modifying the release log URL of each update is also possible, making it possible for an active MITM to show any content as "release log".

Furthermore, we noticed that part of the URL in the HTTP request identifies the area in which the PS4 is bought (this is not the language in which the PS4 is set). For instance: "uk" is identified as "United Kingdom" and "jp" is identified as "Japan". Such information is of course sensitive and can be utilized to further profile the victim. Finally, other servers of Sony accessible by HTTPS still utilize a weak RC4-MD5 cipher, which is proven to be insecure for current industry standards [10].

Listing 1: PS4 OS update information in a HTTP response

```
<?xml version="1.0" ?>
<update_data_list>
  <region id="it" >
    <force_update>
      <system level0_system_version="05.010.000" level1_system_version="05.010.000" />
    </force_update>
    <system_pup label="5.01" sdk_version="05.010.000" version="05.010.000">
      <update_data update_type="full" >
       <image size="1">
       http://xx.ps4.update.playstation.net/.../PS4UPDATE.PUP?dest=jp
       </image>
      </update_data>
    </system_pup>
    <recovery_pup type="default" >
      <preinst_pup  version="default"/>
      <system_pup label="5.01" sdk_version="05.010.000" version="05.010.000"/>
      <system_ex_pup id="0" version="00.000.000.000" label="0.000.000.000"/>
      <image size="1">
      http://xx.ps4.update.playstation.net/.../PS4UPDATE.PUP?dest=us
      </image>
    </recovery_pup>
  </region>
</update_data_list>
```

To conclude, we also discovered a vulnerability that allows an adversary to read portions of memory of the PS4 Wi-Fi chip during a Wi-Fi connection setup by exploiting a buffer-overread present in the 802.11n module. The memory leakage consists of sensitive data from the previous Wi-Fi connection and parts of 802.11 frames currently in air. This vulnerability has been disclosed responsibly to Sony, but has not yet been fixed. According to the vendor 'Sony', the vulnerability did not had a significant large enough impact on security to be fixed.
During this thesis, we have also discovered a DoS vulnerability by altering HTTP responses of the game update mechanisms, which ultimately results into a hard shutdown of the system with a possible loss of data. This vulnerability has disclosed responsibly as well and has been fixed by vendor 'Sony' in one of their latest PS4 firmware updates.

In the table below, we show a summary of the current information leakages of the PS4 operating system and its corresponding network components, that can be extracted with

a MITM:

Table 3: Information leakages of PS4 network and operating system that can be extracted with a MITM attack

Description	Privacy impact	Integrity impact	Requires active MITM?	Difficulty
Modifiable images in PSN	/	High	Yes	Low
Identify games while browsing	Medium	/	No	Low
Identify game updates	Medium	/	No	Low
Identify PSN username while uploading	Low	/	No	Low
Identify PS4 version	Low	/	Yes	Low
Identify country	Low	/	No	Low
Weak RC4-MD5 cipher	High	High	No	High
Over-read PS4 Wi-Fi memory	High*	/	Yes	Low

5.1.2 Game fingerprinting

Just like identifying a website that the end user is visiting, knowing which game a user is playing has a privacy impact as well.
When a game is loaded into the PS4 drive, a plain HTTP request is sent to a server of Sony to gather more information about the game. However, if the information was requested previously in a specific timeframe, the information will then be cached and thus no plaintext communication will be visible. Subsequently, when the user starts a game, several unencrypted requests and responses are sent to each other containing various data such as analytics or update mechanisms. This unencrypted stream of communication often consists of DNS and HTTP data.
Based on this communication, we can identify which game the end user is currently starting. We compiled a list of several PS4 games that we could identify with a passive MITM solely by starting up the game:

Table 4: Fingerprintable games solely by analyzing the communication at startup

Game	Identifiable by DNS?	Identifiable by HTTP?	Additional leakages
Uncharted 4: The Lost Legacy	Yes	Yes	PSN username
Metal Gear Solid: Phantom Pain	Yes	No	PSN username
Just Cause 3	Yes	No	/
Rise Of The Tomb Raider	Yes	No	/
Crash Bandicoot: N.Sane Trilogy	Yes	No	/
Battlefield 1	Yes	Yes	analytics
Watch Dogs	Yes	No	/
Watch Dogs 2	No	No	/

Almost all of the DNS requests show the exact name or abbreviation of the game's name. Similar information is found in the HTTP requests and responses to and from the game server. In some cases, such as Uncharted 4 and MGS, the PSN username is sent in plaintext. Battlefield 1 is even sending analytics data such as the country or number of plays to the webserver over plain HTTP. To the best of our knowledge, we have not found any proof that Playstation users or researchers are currently aware of the issues discussed above.

In this section, we only briefly examined the general weaknesses in the PS4 system and methods that are possible to fingerprint PS4 games. Further research is necessary to analyze whether games on other consoles are also vulnerable to similar fingerprinting techniques.

6 Experiments

In this section, we perform our IUPTIS attack on the social platform 'DeviantArt' and the travel booking platform 'Hotels.com' in order to gain insights into the usefulness and accuracy of a practical WFP attack. Ultimately, we believe it is important to perform experiments that closely resemble the realistic world in which an end user will visit those pages. Therefore, our experiment is simulated by randomly selecting a browser (for each webpage visit) from the list below:

- Firefox 56.0.2 (Linux)
- Google Chrome 62.0 (Linux)
- Google Chrome 61.0 (Android)

Such an approach exposes the immediate impact that our attack can have when applied in a realistic scenario. Nevertheless, in the context of our attack, there are no notable differences between different browsers except from the change in request size.

6.1 IUPTIS: Attacking DeviantArt

DeviantArt is an online art community that consists of 36 million users where artists can upload and view a substantial number of artworks [30]. We randomly compile a list of 2150 DeviantArt profile webpages [7] that have at least 5 uploaded images. Additionally, DeviantArt uses a Galois Counter Mode cipher for all their domains, which means that we can extract the exact length of a HTTP request/response.
Our traffic trace is constructed by spawning each profile webpage separately after each other until all images are loaded with a minimum delay of 3 seconds before closing the previous page and opening the next webpage. *Lazy loading* is a concept that is applied on

[7] in the form of *'https://www.deviantart.com/[USER_NAME]/gallery'*

DeviantArt which means that only the images in the current viewport will be downloaded, thus visible in the traffic trace. With this generated traffic trace containing 2150 profiles, we run our IUPTIS attack.
There are many SMPs that we could have chosen to experiment on. However, we have specifically chosen for DeviantArt because it turns out it has a lot of quirks and it raises some practical issues that implementations of this attack can have.

6.1.1 Preprocessing profiles

Our goal of this attack is to accurately predict the individual profile that a user (which we name 'victim' from now on) has visited through the use of a random web browser. The individual profile is defined to be an URL and the victim can be anyone who is surfing on DeviantArt with any device. Approximately 70% of the DeviantArt accounts are estimated to have no uploaded images. Since our attack is primarly developed for social profiling, these profiles (which are usually the fans of artists themselves and so are viewed rarely) are useless for our attack. We emperically define 'an active DeviantArt artist' as someone who has at least 5 images uploaded on their personal profile page. This makes 90% of those DeviantArt artist accounts useful for fingerprinting.
We would like to stress that all DeviantArt users are vulnerable to this attack but we can only fingerprint 90% of the profile pages that these users (victims) visit. This number can increase if we change our Φ to a lower value, in exchange for a lower accuracy.
Each individual profile page has a gallery in which users upload their artwork. This can be accessed by navigating to 'https://*[USER_NAME]*.deviantart.com/gallery', which is exactly what our URL prediction will look like. Our preliminary work starts with fingerprinting 47 000 images collected from exactly 2150 profiles. The profile pages are selected randomly and filtered to only contain 'active DeviantArt artists'.
A script is written to extract the image url's from the 2150 profiles. Next, we send a HTTP HEAD to get the headers of each image. Finally, the necessary fingerprint data is saved into the database. In this experiment, we specifically want to measure the impact of using various devices and browsers and the ability to uniquely identify a profile based on a stream of image sizes.

6.1.2 Methodology

Several parameters need to be setup for our attack to work. First, we setup the maximum σ value for HTTP requests which is defined as H_{req}. In almost all cases that we encountered, the HTTP request size is the same for every request on a given modern browser. For all experiments, a value of 0.2 is set for H_{req}.
The maximum σ value for HTTP responses (H_{resp}) is however very different. It can change substantially based on for instance, extra caching headers. For this reason, we set a value of 3.6 which we have found emperically.
Next, we setup parameter b_{req}. We seperate this from the complete Req_x so that we can

change parameter b_{req} if the browser used to issue the request is known. In this scenario, we found out that an average static length of 252 bytes works best in most cases.
Finally, we setup the request and response size variances. The request size variance (π_1) will mostly depend whether or not we know which browser the victim is using. In our experiment, we set a very small request variance (10) if we know which browser we are using and set a larger variance (300) if we use an unknown/random browser. The response size variance (π_2) mostly depends on the CDN or server handling the request. For this experiment, a value of 40 is used.
An important factor that will influence our experiment is 'lazy loading'. This technique (which is implemented on DeviantArt), allows the browser to only download images that are currently in the screen viewport of the victim. Scrolling through the window is necessary to automatically download other images that are outside the viewport. Regardless of this, our experiment does not scroll through the window but we did make sure that each profile shows at least 10 images on average, in the current viewport. It is safe to say that this assumption is fairly realistic because victims usually do not view all the images of a given profile. In these tests, the reverse DNS of the IP of every TCP connection that has the word *'deviantart'* in the domain name, was added to the network trace.
In order to run our experiment, we wrote a HTML page in Javascript that will open a popup with the gallery URL of the specific profile. Then we sleep a random amount of time (enough to at least load all the webpages in the current viewport) until we finally close the popup and start anew with the next profile in our fingerprint collection. All the traffic that is generated by the browser that runs our script is collected in one network trace. At the end of our run, the network trace is inputted into our IUPTIS algorithm and the list of profile predictions is outputted.
In figure 8, we observe that most HTTP responses are small in length (1 to 40Kb) and is generally only decreasing after 25Kb. It is therefore statistically easier to identify a profile with large images than with small images.

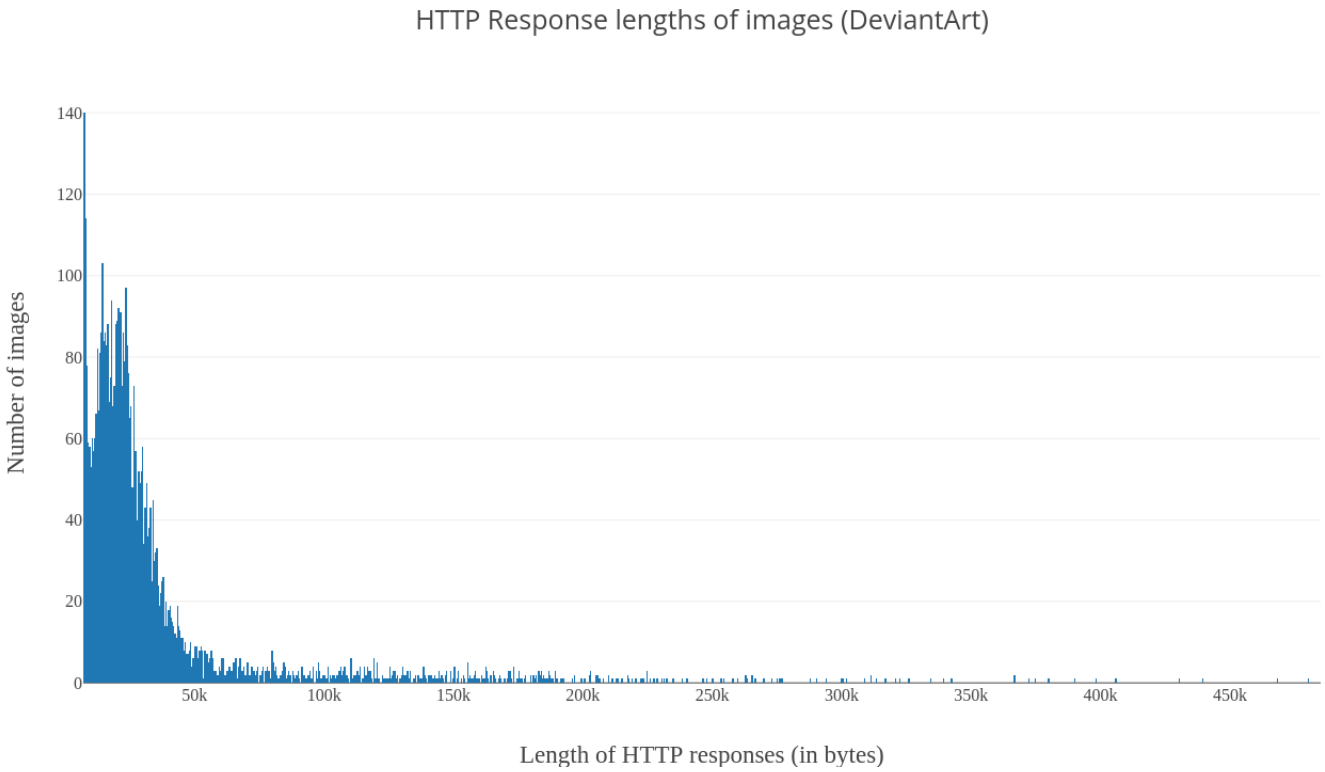

Figure 8: Graph that shows the number of images of a certain HTTP response length on DeviantArt.com. This graph contains a total of approximately 65000 images.

6.1.3 Results

Unless noted, all of our tests use the following empirically defined parameters: $\pi_{\text{req}} = 300$, $\pi_{\text{resp}} = 40$, $useJenks = \text{no}$, $b_{\text{req}} = 252$, $H_{\text{resp}} = 3.6$ and $H_{\text{req}} = 0.4$.
From running our tests, we have obtained the following results:

Table 5: Experiment on DeviantArt with variable parameters and a worldsize of 2150 profiles. Parameter '*caching*=X' indicates that we pre-cache the first X % of all the images located on the profile webpage. Sensitivity, precision and F1 score is shown in percentage.

Sequence (Φ)	**Other parameters**	**Sensitivity**	**Precision**	**F1 score**
2	π_2=10, b_{req}=X	99	98	99
2	π_2=10, b_{req}=X, $caching$=0.4	94	98	96
2	/	99	88	93
3	/	98	93	95
4	$useJenks$=yes	97	97	97
5	$useJenks$=yes	96	99	98
5	$useJenks$=yes, $caching$=0.4	87	99	92

The first test has set the parameter b_{req} which indicates that the adversary knows which browser the end user is using. Including this additional parameter has a considerable positive effect on the precision of the attack with an increase of 11% (ceteris paribus). It is also evident that a large sequence will increase the precision and decrease the sensitivity. We can attribute this due to the statistical probability that it is less likely for a profile to have the same size of several images in a row as another profile. Caching the images does influence the sensitivity since the request for those images will not lead to the image contained in a HTTP response. It is therefore possible that the number of images that are left on a particular profile do not meet the requirements to evolve into a valid sequence. Although, the precision is clearly not affected since caching does not generate any additional false positives.
We conclude that our IUPTIS attack is overall, well suited for this type of social media platform. An important factor in achieving such reliable results, is the fact that we can accurately predict the size of the HTTP response for a given image and that the HTTP request size can be essentially utilized as an extra feature for the prediction. Additionally, the length of headers seldom change which makes this attack very feasible. Such assumptions cannot be made for every platform as we will show in the next experiment.

6.2 IUPTIS: Attacking Hotels.com

6.2.1 Methodology

Hotels.com is an online travel booking platform with an average of 50 million visitors per month and currently has around 260 000 bookable properties [48].
In this experiment, we are predicting the hotel (which is the profile here) that the end user is visiting using a random browser.We chose Hotels.com for a few reasons.
First of all, the website allows us to extract images automatically without having to worry about captchas when downloading several pages in sequence. Second, Hotels.com is using Akamai as CDN, which is very different from DeviantArt's Cloudfront CDN. The difference especially lies in the length of each HTTP response. In the DeviantArt experiment, the difference in length between a cached image and a non-cached image was only 6 to 12 bytes and most profiles only had either cached or uncached images. Meanwhile, Hotels.com has a difference of 12 to 28 bytes for cached and uncached images and profiles have interleaved cached and uncached images. On top of that, they also have other headers that are dependent on various factors. Such length behaviour is hard to model. Due to this large difference, we use the Jenks natural breaks method as discussed in section 4.8.3.
Visiting a hotel through an online platform usually starts with opening the main page and providing information about the length of the stay, the date, number of rooms, etc ... .
The provided information will be sent in a GET request (in the GET url) for each search the user performs or hotel page the user opens. For each hotel page, additional GET parameters are also added and existing parameters can change due to several unpredictable factors. Therefore, we cannot uniquely fingerprint the exact size of the HTTP requests for each profile as we did in the DeviantArt experiment.

This, however, blurs the line between using a random browser and an unknown browser because as we have discussed before, using a different browser will only differ in the length of the HTTP request. Each test in this experiment is therefore performed with a large request variance (π_1).
For this experiment, we can use a high Φ because hotel pages have significantly more images compared to DeviantArt profile pages. We also execute the pre-fingerprinting stage similarly to the DeviantArt experiment with only a few minor differences.
We compile our profile list by randomly selecting 900 hotel profile webpages [8]. Similar to the DeviantArt experiment, our traffic trace is constructed by spawning each hotel webpage and then opening 75% of all the images located on the webpage. Images are not loaded automatically and thus requires the end user to click on the image in the interest of downloading the full resolution image. We argue that the average end user does not open all images on the webpage when browsing through the website. We run our experiment with the following empirically defined parameters:
$b_{\text{req}} = 250$, $\pi_{\text{resp}} = 100$, $\pi_{\text{req}} = 450$ and $H_{\text{req}} = 0.2$.

6.2.2 Results

In table 6, we show the sensitivity, precision and F1 score based on the experiment run by altering parameter H_{resp}, the sequence length and whether or not we use the Jenks method. With the exception of 'Without Jenks(H_{resp}=3.5)', a consistent F1 score between 80 - 98% is achieved. For a sequence (Φ) of 8 images, 'With Jenks (H_{resp}=6.0,Φ = 8)' yields a F1 score of 98%. Overall, sensitivity is relatively constant in almost all tests and only decreases slightly when a longer sequence is necessary as shown in Fig. 9 and Fig. 10. On the contrary, the precision starts low and increases to a very convenient 99% in some cases. However, a low sequence length is prefered to incorporate the ability for the end user to use browser caching in exchange for a lower precision. Fortunately, performing the attack without Jenks and H_{resp}=8.5 already attains a sensitivity and precision of respectively 99 and 92% for a sequence of 6 images. Furthermore, applying the Jenks method to model the CDN cache behavior does show major improvements in sensitivity over different H_{resp} values (ceteris paribus) with only a nominal decrease in precision. For instance, 'Without Jenks (H_{resp}=3.5)' has inferior sensitivity (below 55%) compared to the other tests due to the fact that some images are cached by the CDN server which makes the resulting responses very different in size. To the contrary, in the DeviantArt experiment, the CDNs employed did not have a significant impact on the response size.
In the context of our open world adversary model, we have to be careful in balancing the importance of precision and sensitivity as shown in Sect. 4.9. Therefore, we argue that 'Without Jenks (H_{resp}=8.5)' is ideal in this scenario due to the very advantageous precision (85% to 100%) and relatively high sensitivity (82% to 99%) over all possible sequence lengths. In consummation, we determine that the combination of parameters to

[8] in the form of 'https://hotels.com/ho[NUMBER]/?[GET_PARAMETERS]'

perform the attack will greatly depend on the adversary and end user model.
These results also show that the IUPTIS attack is highly suitable for predicting hotels on online travel platforms where each hotel (profile) contains many images.

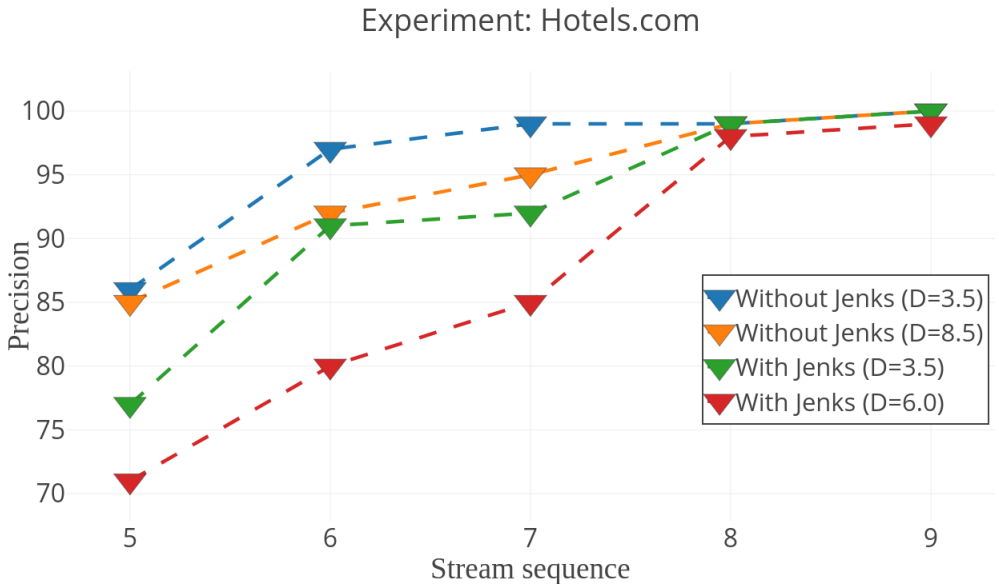

Figure 9: Hotels.com experiment consisting of various tests from table 6 with different parameters, plotting the precision on the length of a valid sequence

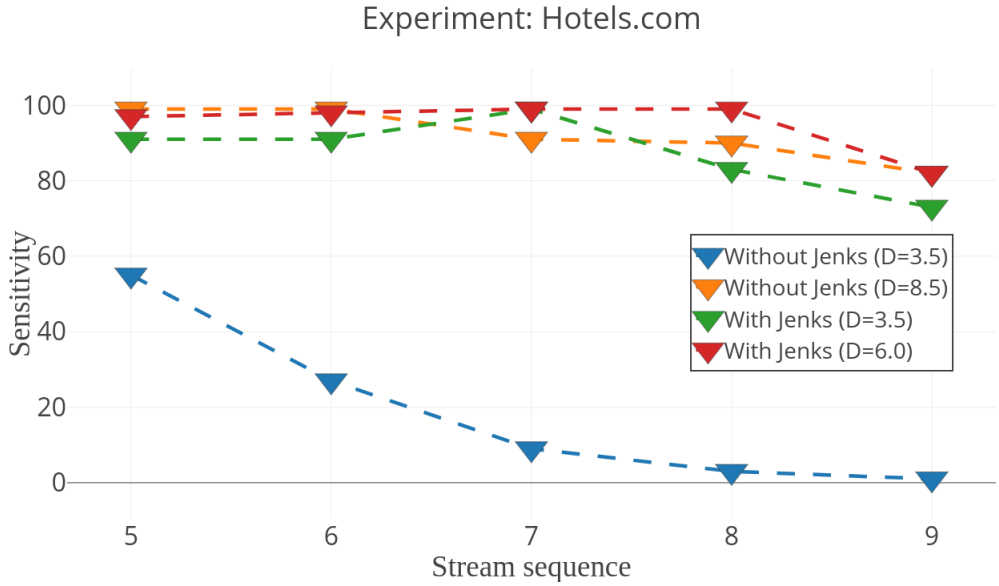

Figure 10: Hotels.com experiment consisting of various tests from table 6 with different parameters, plotting the sensitivity on the length of a valid sequence

Table 6: Experiment on Hotels.com with a worldsize of 900 hotel profiles and variable parameters. Sensitivity, Precision and F1 score are presented in percentage.

Parameters	**Sensitivity**	**Precision**	**F1 score**
Without Jenks (H_{resp}=3.5,$\Phi = 5$)	55	86	67
Without Jenks (H_{resp}=3.5,$\Phi = 6$)	27	97	43
Without Jenks (H_{resp}=3.5,$\Phi = 7$)	9	99	16
Without Jenks (H_{resp}=3.5,$\Phi = 8$)	3	99	6
Without Jenks (H_{resp}=3.5,$\Phi = 9$)	1	100	2
Without Jenks (H_{resp}=8.5,$\Phi = 5$)	99	85	92
Without Jenks (H_{resp}=8.5,$\Phi = 6$)	99	92	96
Without Jenks (H_{resp}=8.5,$\Phi = 7$)	91	95	83
Without Jenks (H_{resp}=8.5,$\Phi = 8$)	90	99	94
Without Jenks (H_{resp}=8.5,$\Phi = 9$)	82	100	90
With Jenks (H_{resp}=3.5,$\Phi = 5$)	91	77	83
With Jenks (H_{resp}=3.5,$\Phi = 6$)	91	91	91
With Jenks (H_{resp}=3.5,$\Phi = 7$)	99	92	96
With Jenks (H_{resp}=3.5,$\Phi = 8$)	83	99	90
With Jenks (H_{resp}=3.5,$\Phi = 9$)	73	100	85
With Jenks (H_{resp}=6.0,$\Phi = 5$)	97	71	82
With Jenks (H_{resp}=6.0,$\Phi = 6$)	99	80	88
With Jenks (H_{resp}=6.0,$\Phi = 7$)	99	85	89
With Jenks (H_{resp}=6.0,$\Phi = 8$)	99	98	98
With Jenks (H_{resp}=6.0,$\Phi = 9$)	82	99	89

7 Defenses against IUPTIS

Research on website fingerprinting has been mostly focused on fingerprinting for a wide variety of protocols and finding countermeasures against such attacks in practice [53]. Especially performance, effectiveness and viability are measured and analyzed.
In recent years, some countries (e.g. China) have started to actively detect and block access to privacy anonymization tools like Tor [1] in order to apply censorship. Researchers have looked into the process of mitigating these censorship authorities [60] [108]. However, as we will discuss shortly, multiple tools and defenses against fingerprinting attacks exist and we believe it is only a matter of time before these defenses are detected and blocked as well. In other words, we need a defense that protects the end user against information leakages but can also be hidden.
In the context of social profiling, such a 'non-identifiability' requirement for fingerprinting defenses is very important. As discussed in section 1.1, 'Social Wi-Fi' companies provide their services as in return, they can extract personal information from the user. If a fingerprinting (FP) defense is applied, detection of such defenses can have consequences for the

end-user. For instance, the service provider can deny access to their services if a mitigation used by the end-user is detected.
As a result, we discuss which existing countermeasures are sufficient enough to mitigate our IUPTIS attack based on 'effectiveness', 'performance', 'viability' and 'non-identifiability'.

7.1 Padding

A basic defense mechanism that we consider is 'linear padding'. With linear padding applied, we make sure that the length of every HTTP request and response is divisible by a specific number of bytes. Padding is still common on some social media platforms as it is applied to block ciphers used in HTTP over TLS, for instance AES. Even if the adversary knows that padding is applied, HTTP requests and responses will look larger than they actually are, resulting into a large standard deviation σ. In this case, we cannot use a H_{req} and H_{resp} lower than $padding/2$ because our σ will likely overshoot both boundaries. So in order to compensate for this, we increased both parameter H_{req} and H_{resp} to 16 and 64 for respectively 32 byte blocks and 128 byte blocks of padding applied.
With that in mind, we ran all tests on existing configurations from our original DeviantArt experiment (section 16.1), with linear padding applied ($\pi_{\text{req}} = 300$, $\pi_{\text{resp}} = 40$, $useJenks$ = no, $b_{\text{req}} = X$, $H_{\text{resp}} = 3.6$ and $H_{\text{req}} = 0.4$) [9]:

Streak (Φ)	**Padding**	**Sensitivity (%)**	**Precision (%)**	**F1 score (%)**
3	0	98	93	95
4	0	97	97	97
3	32	96	63	76
4	32	93	90	91
3	128	92	1	65
4	128	89	2	64

The results are still very acceptable for a small amount of padding, but the precision suffers substantially when applying padding to blocks of 128 bytes. Nevertheless, it is clear that a linear padding of 128 bytes (or more) is effective in generating many false positives. However, the sensitivity is obviously not affected which means that the adversary can still narrow down the number of possible profiles that the victim has visited if we would utilize a closed world scenario. It is also important to note that we modified our parameters of the attack with the prior knowledge that padding will be applied. This assumption is fair if the padding is part of the protocol standard (for instance: block ciphers). Although, if variable padding is applied to the HTTP data itself, an adversary might have difficulties guessing which parameters he should use in order to succesfully perform the attack.
We conclude that padding does not completely mitigate our attack, but it certainly makes

[9] In all of our tests, we randomly select 100 profiles out of 2150 fingerprinted profiles as we did in the DeviantArt experiment

the preliminary work of the adversary much more cumbersome. Furthermore, such mitigation is often criticized for their performance intensive nature due to the extra bandwidth that is needed [53, 20]. Other padding schemes proposed by Dyer et al. [33] like Mice-Elephant or Exponential padding have the same effect.

7.2 Camouflage

Camouflage is a client-side defense (originally meant for Tor networks) which tries to cover up traffic by requesting a random webpage (profile) whenever the victim is trying to visit another unrelated webpage [81]. Applied to our attack, this would mean that we request a random profile page Y when the victim requests a profile page X. Both profiles will be loaded simultaneously by opening multiple TCP connections to the target server. Since our attack expects a sequence of HTTP responses that belong to the same profile, such defense will interleave the responses and requests which makes it statistically much harder to find such sequence. It also injects false predictions as the victim is not the one requesting the random page. Advantages of this defense are that it is easy to implement in a browser add-on and it is hard to detect if applied. In the DeviantArt experiment, an advantageous side effect is that dummy profile (webpage) can be requested directly by the browser itself and so there is no need to know the different sizes of each image on a given profile (only the profile URL is required). Consequently, exactly the same HTTP request and response will be transmitted as if the victim has visited the dummy webpage by himself through the same browser. However, additional data about the profiles is necessary in websites where user interaction is required (Hotels.com experiment) in order to download the fingerprinted images on a given webpage. We performed some tests where where apply the Camouflage concept to our attack (see table 7):

Streak (Φ)	**Number of dummies**	**Sensitivity (%)**	**Precision (%)**	**F1 score (%)**
2	0	99	88	93
2	1	98	51	73
2	2	82	39	53
5	0	96	99	97
5	1	85	56	68
5	2	68	47	55

Table 7: Camouflage defense applied to the IUPTIS attack performed on DeviantArt with the following parameters: $H_{\text{resp}} = 3.6$, $H_{\text{req}} = 0.5$, $useJenks = no$, $\pi_1 = 450$,$\pi_2 = 100$ and b_{req} is not set (unknown browser).

As expected, Camouflage introduces many false positives due to the extra dummy requests. If we compare the results to the experiment without defenses applied (section 6.1), we notice that the sensitivity has decreased substantially. This is the result of the interleaved

HTTP requests and responses which makes a correct sequence less likely to occur. The same explanation can be given for the fact that the precision with 2 dummy requests is higher for a streak of 5 compared to a streak of 2 (ceteris paribus). Due to the order in which the responses appear, IUPTIS effectively sees this as noise.
Whether or not Camouflage mitigates the IUPTIS attack depends on the context in which the attack is performed. Does the adversary wants to know exactly which profile the victim visited or is he satisfied with several possible predictions? Generally, we can say that an adversary already has enough sensitive information to use when several predictions are given to him. It will also depend on whether or not the fingerprints that the adversary has, is a subset of the possible dummy requests. As a result, we conclude that Camouflage does not sufficiently protect the end user against an IUPTIS attack.

7.3 Overview

There are 2 type of groups that we can protect from our attack (in order of importance):

1. The users that are visiting the fingerprinted profiles. It is their network trace that is used (client-side).

2. The users that own the fingerprinted profiles.

Despite the high accuracy of our attack, the current defenses that exist to mitigate such attack are highly effective as we will show in a moment.
The first basic countermeasure that we discuss, is **linear padding**. In our experiment in section 7.1, we showed that a padding of 128 bytes is highly effective in nullifying the precision of an IUPTIS attack. However, it is still not sufficient to completely mitigate the sensitivity.

Popular privacy applications like **Tor** [84] will introduce a large amount of padding and optional randomized pipelining [38].
Randomized pipelining renders our current attack useless since our length method only works when we can uniquely identify the length of HTTP requests and responses. Furthermore, such large amount of padding will disorient our algorithm and will, depending on the parameters used, generate a lot of false predictions or no predictions at all. In the social media context scenarios in which we focus, we do not expect an average user to use Tor.

HTTPOS (acts like a client-side proxy) is primarily an application layer defense which implements several countermeasures to make it hard for an adversary to calculate the real HTTP content lengths [65]. It uses the HTTP Range header (if supported by the server) to request parts of the content multiple times instead of requesting the complete content at once. Furthermore, it injects junk data to the HTTP content in order to cover up the real traffic data. The HTTP Range header is often only supported for multimedia resources like images. Fortunately, this is sufficient in our scenario. Using this header is a clever and effective way to change the size of HTTP responses.

Finally, it also tries to enable HTTP/1.1 pipelining (if the browser supports is). As the parameters of this defense can be fine-tuned, a large amount of variable 'fake' padding (injecting dummy data into the request) can be introduced. As discussed before, padding is an effective yet performance degrading countermeasure against IUPTIS. Additionally, HTTPOS modifes the MSS option in the TCP protocol in order to limit the size of an outgoing TCP packet. This modification has a severe impact on the performance for small values set to MSS and can be detected when modified consistently.

CS-BuFLO is a client and server-side defense which tries to send the TCP data in a continuous flow with fixed-size packets [18]. The difference of timing does not affect our attack. There are 2 padding schemes of which both add a maximum of X bytes of padding where X is the original data size. Again, such large quantity of padding defeats our attack entirely. Unfortunately, the concept of this countermeasure requires an implementation on both client (browser) and server.

As discussed in section 7.2, **Camouflage** does not completely mitigate our IUPTIS attack. It has similar results as linear padding, but it does however holds a strong "non-identifiability" requirement. An adversary cannot easily the detect the presence of such a defense which makes it very viable in certain scenarios (for instance: social Wi-Fi providers).

Finally, server implementations of **HTTP/2** would open up the possibility of using pipelining and multiplexing. Morla [74] has showed that if server implementations of HTTP/2 extensively use pipelining and multiplexing, the lengths of HTTP requests and responses are hard to predict, rendering our IUPTIS attack ineffective. It is however up to the server to implement, as most client browsers already support HTTP/2.

We conclude that most countermeasures are succesful in mitigating our attack in exchange for performance and bandwidth. However, in practice, many of such countermeasures are hard to implement since they require server-side changes or the use of additional proxies. Lastly, all defenses with the exception of Camouflage and HTTP/2 are identifiable to a certain extent. Unfortunately, HTTP/2 requires an upgrade at the server-side and so is out of reach for the client-side victims.
If identifiability and the sensitivity measurement is of no concern, then adding linear padding of at least 256 (preferrably more) bytes is sufficient to mitigate our IUPTIS attack. The grid below shows an overview of the considered defenses against IUPTIS. Viability and performance is an objective score (confirmed by facts) from 0 to 5. Camouflage(X) indicates that we load X additional dummy profiles for each requested profile page:

Table 8: Possible defenses against IUPTIS based on various factors

Defense	Mitigates?	Non-identifiable?	Bandwidth overhead	Viability	Performance
Linear padding (32 bytes)	✗	✗/✓	2.0%	5	5
Linear padding (128 bytes)	✗/✓	✗	8.1%	5	5
HTTPOS	✓	✗	?	3	4
CS-BuFLO	✓	✗	50%	1	3
Camouflage(1)	✗	✓	100%	4	4
Camouflage(2)	✗/✓	✓	200%	4	4
Tor	✓	✓/✗	?	4	1
HTTP/2	✓	✓	-X%	4	+5

8 Conclusion and Future Work

In this thesis, we have discussed and analyzed several fingerprinting techniques and modified them to be utilized in practical real-life scenarios such as marketing or cybercrime departments. We also applied the basics of these techniques to the Playstation 4 game console which to the best of our knowledge, has never been done before. Furthermore, the discoveries of weaknesses in the newest generation of PS4 system has shown that basic encryption such as HTTPS, is still not widespread. On top of that, a vulnerability in the PS4 update mechanism and another one in the Wi-Fi chip has been reported where the former one has been fixed by the vendor. As is clear throughout the text, most of the work that has been put into this thesis is involved in discussing, analyzing and experimenting with the IUPTIS attack. A short version of the section 'IUPTIS' has been submitted as a paper to the international conference ESORICS and has yet to be accepted. However, the IUPTIS attack is only applicable in HTTP/1.1. Further research about the viability of this attack in the HTTP/2 protocol and over other more general web platforms will be done in the forseeable future.

The main research questions of our thesis about social profiling are also answered. We showed that social profiling is rising in popularity and that various side-channel methods exist that can extract sensitive information from encrypted communication. Existing defenses against those attacks have been presented and have been carefully analyzed to determine whether they are practical in a realistic scenario. Even though, some of these methods are significantly complex to perform, the privacy implications are not to be neglected and should therefore be an important issue in further legal enactments.

Personally, I believe that the experience from researching the techniques and methods in this thesis has aided me in further developing desirable skills for vulnerability reward programs and has ultimately helped me into deciding that an academic path is the most interesting path to follow.

A Buffer-overread vulnerability

During the research of this thesis, we have discovered a buffer-overread vulnerability in the network component of the PS4. It is possible to over-read 128 bytes from sensitive PS4 memory during the WEP handshake in an AP association between the authenticator (attacker) and supplicant (PS4). All OS versions before 5.3 are vulnerable.
We have responsibly disclosed this vulnerability to the vendor, but is currently classified as 'informative'. According to the vendor, this vulnerability has no significant impact on security and will therefore not be patched. Timeline:

Sept 22, 2017	Initial disclosure of vulnerability to vendor 'Sony'.
Sept 26, 2017	Vendor confirms that report has been received and is currently investigating.
Oct 4, 2017	Vendor responds that vulnerability has been patched and thus case is closed.
Oct 4, 2017	Vulnerability is still exploitable. Vendor notified.
Oct 5, 2017	Vendor requires more information about vulnerability.
Oct 9, 2017	More detailled information is sent to vendor, including proof of concept code (Python).
Oct 11, 2017	Vendor does not understand vulnerability and requests additional information.
Oct 11, 2017	Clarification about scope of vulnerability is sent to vendor.
Nov 2, 2017	Current status of vulnerability requested to vendor.
Nov 6, 2017	Vendor is still investigating issue.
Apr 28, 2018	Vendor classifies vulnerability as 'informative'.

B Practical example of IUPTIS

B.1 Segmentation of HTTP request

An example of a DeviantArt HTTP request for an image of a profile:

```
1  GET /SbDJcoTRsjXcyTT0/fit-in/700x350/this_is_the_name_of_the_artwork.png HTTP/1.1
2  Host: t00.deviantart.net
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/57.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Referer: https://myprofile.deviantart.com/gallery/
8  Connection: keep-alive
9  Pragma: no-cache
10 Cache-Control: no-cache
```

p_{req} consists of the following header *values*:

- https://myprofile.deviantart.com/gallery/
- /SbDJcoTRsjXcyTT0/fit-in/700x350/this_is_the_name_of_the_artwork.png

The total length of p_{req} is therefore 109 bytes (no newlines).
b_{req} is a fixed value which depends on the webbrowser. In case of 'DeviantArt', b_{req} consists of:

- All fixed header *names* (Host,Accept,Referer, ...)
- gzip, deflate, br
- keep-alive
- no-cache
- */*
- Mozilla/5.0 (X11; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/57.0

References

[1] How the great firewall of china is blocking tor. In *Presented as part of the 2nd USENIX Workshop on Free and Open Communications on the Internet* (Bellevue, WA, 2012), USENIX.

[2] Exploiting cors misconfigurations for bitcoins and bounties. PortSwigger, October 2016. `http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html`.

[3] Facebook bug bounty: $5 million paid in 5 years. Joey Tyson, October 2016. `https://www.facebook.com/notes/facebook-bug-bounty/facebook-bug-bounty-5-million-paid-in-5-years/1419385021409053/`.

[4] Reporting a security vulnerability. Soundcloud, December 2017. `https://help.soundcloud.com/hc/en-us/articles/115003561228-Reporting-a-security-vulnerability`.

[5] Sony, 2017. `https://www.sony.com`.

[6] Are you on tinder? someone may be watching your swipe. Checkmarx, January 2018. `https://info.checkmarx.com/hubfs/Tinder_Research.pdf`.

[7] Don't trust the vpn facebook wants you to use. Wired, February 2018. `https://www.wired.com/story/facebook-onavo-protect-vpn-privacy/`.

[8] Accenture. Cost of cyber crime study.

[9] Alan, H. F., and Kaur, J. Can android applications be identified using only tcp/ip headers of their launch time traffic? In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (New York, NY, USA, 2016), WiSec '16, ACM, pp. 61–66.

[10] AlFardan, N. J., Bernstein, D. J., Paterson, K. G., Poettering, B., and Schuldt, J. C. N. On the security of rc4 in tls. In *Proceedings of the 22Nd USENIX Conference on Security* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association, pp. 305–320.

[11] Alnaami, K., Ayoade, G., Siddiqui, A., Ruozzi, N., Khan, L., and Thuraisingham, B. P2v: Effective website fingerprinting using vector space representations. In *Computational Intelligence, 2015 IEEE Symposium Series on* (2015), IEEE, pp. 59–66.

[12] Android. Android open source project - dns-dev, 2017. `https://android-review.googlesource.com/q/topic:dns-dev-opt+(status:open+OR+status:merged)`.

[13] Armerding, T. The 16 biggest data breaches of the 21st century. CSO, October 2017. `https://www.csoonline.com/article/2130877/data-breach/the-16-biggest-data-breaches-of-the-21st-century.html`.

[14] Boundless Informant: the NSA's secret tool to track global surveillance data. The Guardian, June 2013. `https://www.theguardian.com/world/2013/jun/08/nsa-boundless-informant-global-datamining`.

[15] Bounty, F. B. Facebook bug bounty, 2014. `https://www.facebook.com/BugBounty/posts/778897822124446`.

[16] Brandwatch. Brandwatch Peer Index. `https://www.brandwatch.com/p/peerindex-and-brandwatch`.

[17] Bugcrowd. A radical cybersecurity advantage, 2017. `https://www.bugcrowd.com/`.

[18] Cai, X., Nithyanand, R., and Johnson, R. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2014), WPES '14, ACM, pp. 121–130. `http://doi.acm.org/10.1145/2665943.2665949`.

[19] Cai, X., Nithyanand, R., Wang, T., Johnson, R., and Goldberg, I. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2014), CCS '14, ACM, pp. 227–238.

[20] Cai, X., Zhang, X. C., Joshi, B., and Johnson, R. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 605–616. http://doi.acm.org/10.1145/2382196.2382260.

[21] Cao, Y., Li, S., and Wijmans, E. (cross-)browser fingerprinting via os and hardware level features. In *NDSS* (2017). https://doi.org/10.14722/ndss.2017.23152.

[22] Cheng, H., , Cheng, H., and Avnur, R. Traffic Analysis of SSL Encrypted Web Browsing, 1998.

[23] Cherubin, G. Bayes, not Naïve: Security Bounds on Website Fingerprinting Defenses. *PoPETs* (2017), 215–231. https://doi.org/10.1515/popets-2017-0046.

[24] Communications, N. Netscape announces "netscape bugs bounty" with release ofnetscape navigator 2.0 beta, 1997. https://web.archive.org/web/19970501041756/http://www101.netscape.com/newsref/pr/newsrelease48.html.

[25] Conti, M., Mancini, L. V., Spolaor, R., and Verde, N. V. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2015), CODASPY '15, ACM, pp. 297–304.

[26] Coull, S. E., Collins, M. P., Wright, C. V., Monrose, F., and Reiter, M. K. On Web Browsing Privacy in Anonymized NetFlows. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (Berkeley, CA, USA, 2007), SS'07, USENIX Association, pp. 23:1–23:14. http://dl.acm.org/citation.cfm?id=1362903.1362926.

[27] Dai, S., Tongaonkar, A., Wang, X., Nucci, A., and Song, D. Networkprofiler: Towards automatic fingerprinting of android apps. 809–817.

[28] Davies, M., Read, H., Xynos, K., and Sutherland, I. Forensic analysis of a sony playstation 4: A first look. *Digital Investigation 12* (2015), S81 – S89. DFRWS 2015 Europe.

[29] DevCore. How i hacked facebook, and found someone's backdoor script. https://devco.re/blog/2016/04/21/how-I-hacked-facebook-and-found-someones-backdoor-script-eng-ver/.

[30] DeviantArt. Accessed on 12-21-2017.

[31] Dierks, T. The transport layer security (tls) protocol version 1.2. STD 5246, August 2008. https://www.ietf.org/rfc/rfc5246.txt.

[32] Donohue, B. Gaming console hacks. Kaspersky, January 2014. https://www.kaspersky.com/blog/gaming-console-hacks/3552/.

[33] Dyer, K. P., Coull, S. E., Ristenpart, T., and Shrimpton, T. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2012), SP '12, IEEE Computer Society, pp. 332–346. http://dx.doi.org/10.1109/SP.2012.28.

[34] Economist, T. Very personal finance, June 2012. http://www.economist.com/node/21556263.

[35] Ejeta, T. G., and Kim, H. J. Website Fingerprinting Attack on Psiphon and Its Forensic Analysis. In *Digital Forensics and Watermarking* (Cham, 2017), C. Kraetzer, Y.-Q. Shi, J. Dittmann, and H. J. Kim, Eds., Springer International Publishing, pp. 42–51. https://doi.org/10.1007/978-3-319-64185-0_4.

[36] Espiner, T. Home Office presses ahead with web interception. ZDNet, January 2010. https://www.zdnet.com/article/home-office-presses-ahead-with-web-interception/.

[37] Fernández, A. Clinical report: The impact of social media on children, adolescents and families. *Archivos de Pediatría del Uruguay 82*, 1 (2011), 31–32.

[38] Foundation, O. S. Experimental defense for website traffic fingerprinting. The Tor Project, 2011. https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting.

[39] Foundation, O. S. Meerderheid zorgwebsites heeft geen veilige https-verbinding (dutch). O'Reilly Media, August 2017. https://openstate.eu/nl/2017/08/meerderheid-zorg-websites-heeft-geen-veilige-https-verbinding/.

[40] Gan, D., and Jenkins, L. R. Social networking privacywhos stalking you? *Future Internet 7*, 1 (2015), 67–93.

[41] Gonzalez, R., Soriente, C., and Laoutaris, N. User profiling in the time of https. In *Proceedings of the 2016 Internet Measurement Conference* (2016), ACM, pp. 373–379.

[42] Goodin, D. Failure to patch two-month-old bug led to massive equifax breach. Ars Technica, September 2017. https://arstechnica.com/information-technology/2017/09/massive-equifax-breach-caused-by-failure-to-patch-two-month-old-bug/.

[43] Grigorik, I. Transport layer security (tls) networking 101, chapter 4. O'Reilly Media. https://hpbn.co/transport-layer-security-tls/.

[44] Hajli, N. A study of the impact of social media on consumers. In *International Journal of Market Research* (03 2014), vol. 56.

[45] Hayes, J., and Danezis, G. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, 2016), USENIX Association, pp. 1187–1203. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes.

[46] Herrmann, D., Wendolsky, R., and Federrath, H. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial NaïVe-bayes Classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security* (New York, NY, USA, 2009), CCSW '09, ACM, pp. 31–42. http://doi.acm.org/10.1145/1655008.1655013.

[47] Hintz, A. Fingerprinting Websites Using Traffic Analysis. In *Proceedings of the 2Nd International Conference on Privacy Enhancing Technologies* (Berlin, Heidelberg, 2003), PET'02, Springer-Verlag, pp. 171–178. http://dl.acm.org/citation.cfm?id=1765299.1765312.

[48] Hotels.com. Accessed on 12-21-2017.

[49] Hunt, T. Have i been pwned? https://haveibeenpwned.com/.

[50] Husák, M., Čermák, M., Jirsík, T., and Čeleda, P. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *EURASIP Journal on Information Security* (Feb 2016). https://doi.org/10.1186/s13635-016-0030-7.

[51] Information is Beautiful. World's biggest data breaches. http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/.

[52] Juarez, M., Afroz, S., Acar, G., Diaz, C., and Greenstadt, R. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2014), CCS '14, ACM, pp. 263–274. http://doi.acm.org/10.1145/2660267.2660368.

[53] Juarez, M., Imani, M., Perry, M., Diaz, C., and Wright, M. Toward an Efficient Website Fingerprinting Defense. In *Computer Security – ESORICS 2016* (Cham, 2016), I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds., Springer International Publishing, pp. 27–46. https://doi.org/10.1007/978-3-319-45744-4_2.

[54] Kakavas, I. Creepy. https://www.geocreepy.com/.

[55] Karakostas, D., and Zindros, D. Practical new developments on breach. https://www.blackhat.com/docs/asia-16/materials/asia-16-Karakostas-Practical-New-Developments-In-The-BREACH-Attack-wp.pdf.

[56] Khanji, S., Jabir, R., Iqbal, F., and Marrington, A. Forensic analysis of xbox one and playstation 4 gaming consoles. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)* (Dec 2016), pp. 1–6.

[57] Kumar, M. India probes report on breach of national identity database. https://www.reuters.com/article/us-india-economy-biometric/india-probes-report-on-breach-of-national-identity-database-idUSKBN1ET1IX.

[58] Kwon, A., AlSabah, M., Lazar, D., Dacier, M., and Devadas, S. Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services. In *24th USENIX Security Symposium (USENIX Security 15)* (Washington, D.C., 2015), USENIX Association, pp. 287–302.

[59] Laszka, A., Zhao, M., and Grossklags, J. *Banishing Misaligned Incentives for Validating Reports in Bug-Bounty Platforms*. Springer International Publishing, Cham, 2016, pp. 161–178.

[60] Lee, L., Fifield, D., Malkin, N., Iyer, G., Egelman, S., and Wagner, D. Tor's usability for censorship circumvention. Master's thesis, EECS Department, University of California, Berkeley, May 2016. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-58.html.

[61] Liberatore, M., and Levine, B. N. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2006), CCS '06, ACM, pp. 255–263. http://doi.acm.org/10.1145/1180405.1180437.

[62] Liu, L., Preoiuc-Pietro, D., Riahi, Z., Moghaddam, M. E., and Ungar, L. Analyzing Personality through Social Media Profile Picture Choice. In *ICWSM* (2016). https://sites.sas.upenn.edu/sites/default/files/danielpr/files/persimages16icwsm.pdf.

[63] Ltd, D. P. Twitonomy. https://www.twitonomy.com/.

[64] Lu, L., Chang, E.-C., and Chan, M. C. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Proceedings of the 15th European Conference on Research in Computer Security* (Berlin, Heidelberg, 2010), ESORICS'10, Springer-Verlag, pp. 199–214.

[65] Luo, X., Zhou, P., Chan, E. W. W., Lee, W., Chang, R. K. C., and Perdisci, R. HTTPOS: Sealing information leaks with browser-side obfuscation of

encrypted flows. In *In Proc. Network and Distributed Systems Symposium (NDSS). The Internet Society* (2011). 10.1.1.300.1748.

[66] Markovikj, D., Gievska, S., Kosinski, M., and Stillwell, D. Mining Facebook Data for Predictive Personality Modeling. https://www.aaai.org/ocs/index.php/ICWSM/ICWSM13/paper/view/6179.

[67] McAfee. Economic impact of cybercrime no slowing down.

[68] Meister, J. Will your klout score get you hired? the role of social media in recruiting. Forbes, May 2012. https://www.forbes.com/sites/jeannemeister/2012/05/07/will-your-klout-score-get-you-hired-the-role-of-social-media-in-recruiting.

[69] Miller, B., Huang, L., Joseph, A. D., and Tygar, J. D. I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis. In *Privacy Enhancing Technologies* (Cham, 2014), E. De Cristofaro and S. J. Murdoch, Eds., Springer International Publishing, pp. 143–163. https://doi.org/10.1007/978-3-319-08506-7_8.

[70] Miskovic, S., Lee, G. M., Liao, Y., and Baldi, M. Appprint: Automatic fingerprinting of mobile applications in network traffic. In *Passive and Active Measurement* (2015), Springer International Publishing, pp. 57–69.

[71] Mitmproxy. Mitmproxy, 2018. https://mitmproxy.org/.

[72] Mockapetris, P. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987. http://www.rfc-editor.org/rfc/rfc1034.txt.

[73] Mongkolluksamee, S., Visoottiviseth, V., and Fukuda, K. Combining communication patterns & traffic patterns to enhance mobile traffic identification performance. 247–254.

[74] Morla, R. Effect of Pipelining and Multiplexing in Estimating HTTP/2.0 Web Object Sizes. *ArXiv e-prints* (7 2017).

[75] Mulazzani, M., Huber, M., and Weippl, E. Data visualization for social network forensics. In *Advances in Digital Forensics VIII* (Berlin, Heidelberg, 2012), G. Peterson and S. Shenoi, Eds., Springer Berlin Heidelberg, pp. 115–126.

[76] Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., and Weippl, E. Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)* (2013). http://www.ieee-security.org/TC/W2SP/2013/papers/s2p1.pdf.

[77] Overheid.nl. Wet op de inlichtingen- en veiligheidsdiensten 2017 (dutch), 2017. http://wetten.overheid.nl/BWBR0039896/2018-05-01.

[78] OWASP. Cross-site scripting (xss), 2016. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[79] OWASP. Clickjacking, 2017. https://www.owasp.org/index.php/Clickjacking.

[80] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., and Wehrle, K. Website Fingerprinting at Internet Scale. In *NDSS* (2016). https://doi.org/10.14722/ndss.2016.23477.

[81] Panchenko, A., Niessen, L., Zinnen, A., and Engel, T. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2011), WPES '11, ACM, pp. 103–114. http://doi.acm.org/10.1145/2046556.2046570.

[82] Perez, S. Facebook starts pushing its data tracking onavo vpn within its main mobile app, 2018. https://techcrunch.com/2018/02/12/facebook-starts-pushing-its-data-tracking-onavo-vpn-within-its-main-mobile-app/.

[83] Portswigger. Burp suite, 2018. https://portswigger.net/burp.

[84] Project, T. T. Tor. https://www.torproject.org.

[85] Pyka wifi. https://www.pyka-wifi.com.

[86] Rao, A., Spasojevic, N., Li, Z., and Dsouza, T. Klout Score: Measuring Influence Across Multiple Social Networks. *Conference: 2015 IEEE International Conference on Big Data (Big Data)* (2015), 2282–2289. https://doi.org/10.1109/BigData.2015.7364017.

[87] Rimmer, V., Preuveneers, D., Juarez, M., Van Goethem, T., and Joosen, W. Automated Feature Extraction for Website Fingerprinting through Deep Learning, 08 2017. (to appear).

[88] RT. Hacker posts facebook bug report on zuckerbergs wall, August 2013. https://www.rt.com/news/facebook-post-exploit-hacker-zuckerberg-621/.

[89] Ruderman, J. Same-origin policy, 2017. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.

[90] Saltaformaggio, B., Choi, H., Johnson, K., Kwon, Y., Zhang, Q., Zhang, X., Xu, D., and Qian, J. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)* (Austin, TX, 2016), USENIX Association. https://www.usenix.org/conference/woot16/workshop-program/presentation/saltaformaggio.

[91] Shah, S. Android is getting a feature that encrypts website name requests, 2017. https://www.engadget.com/2017/10/23/google-android-dns-tls/.

[92] Shi, Y., and Biswas, S. Website fingerprinting using traffic analysis of dynamic webpages. In *Global Communications Conference (GLOBECOM), 2014 IEEE* (2014), IEEE, pp. 557–563.

[93] So wifi. https://www.socialwifi.com.

[94] Social wifi. https://www.sowifi.com.

[95] Stöber, T., Frank, M., Schmitt, J., and Martinovic, I. Who do you sync you are?: Smartphone fingerprinting via application behaviour. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks* (New York, NY, USA, 2013), WiSec '13, ACM, pp. 7–12.

[96] Sun, Q., Simon, D. R., Wang, Y.-M., Russell, W., Padmanabhan, V. N., and Qiu, L. Statistical Identification of Encrypted Web Browsing Traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2002), SP '02, IEEE Computer Society, pp. 19–. https://doi.org/10.1109/SECPRI.2002.1004359.

[97] Synack. The hacker-powered security platform, 2017. https://www.synack.com/.

[98] Taylor, S. What do mobile business users want from wi-fi? Insights from Cisco IBSG Research, 2012. https://www.cisco.com/c/dam/en_us/about/ac79/docs/sp/What_Do_Mobile_Business_Users_Want_from_Wi-Fi.pdf.

[99] Taylor, V. F., Spolaor, R., Conti, M., and Martinovic, I. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security 13*, 1 (Jan 2018), 63–78.

[100] Van Goethem, T., Vanhoef, M., Piessens, F., and Joosen, W. Request and conquer: Exposing cross-origin resource size. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, 2016), USENIX Association, pp. 447–462. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/goethem.

[101] Vanhoef, M., and Van Goethem, T. Heist: Http encrypted information can be stolen through tcp-windows.

[102] Vegh, L. Cookies consent under the gdpr. EU GDPR Compliant, February 2018. https://eugdprcompliant.com/cookies-consent-gdpr/.

[103] Vranken, G. Https bicycle attack, December 2015. https://guidovranken.com/2015/12/30/https-bicycle-attack/.

[104] Wang, T. Website Fingerprinting: Attacks and Defenses (Doctoral dissertation), 2015. University of Waterloo, Canada.

[105] Wang, T., and Goldberg, I. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *26th USENIX Security Symposium (USENIX Security 17)* (Vancouver, BC, 2017), USENIX Association, pp. 1375–1390.

[106] WiFi4EU. Free wi-fi for europeans, 9 2016. https://ec.europa.eu/digital-single-market/en/policies/wifi4eu-free-wi-fi-europeans.

[107] Wijnants, M., Marx, R., Quax, P., and Lamotte, W. HTTP/2 Prioritization and its Impact on Web Performance. In *The Web Conference* (2018), WWW 2018. (to appear).

[108] Winter, P., Pulls, T., and Fuss, J. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2013), WPES '13, ACM, pp. 213–224. http://doi.acm.org/10.1145/2517840.2517856.

[109] Wright, C. V., Coull, S. E., and Monrose, F. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *In Proceedings of the 16th Network and Distributed Security Symposium* (2009), IEEE, pp. 237–250.

[110] Yin-Poole, W. Ps4 hits 70m sold. Eurogamer, December 2017. https://www.eurogamer.net/articles/2017-12-07-ps4-hits-70m-sold.

[111] Zhao, M., Laszka, A., and Grossklags, J. Devising effective policies for bug-bounty platforms and security vulnerability discovery. *Journal of Information Policy 7* (2017), 372–418. http://www.jstor.org/stable/10.5325/jinfopoli.7.2017.0372.