Analysis and Mitigation of Information Leaks in Web Browsing Traffic

Master Thesis

Department of Computer Science
Saarland University

Submitted by
Goran Doychev

Supervisor:
Dr. Boris Köpf

Reviewers:
Prof. Dr. Michael Backes
Dr. Boris Köpf

Saarbrücken, 2012

Abstract

Web traffic is exposed to potential eavesdroppers, and despite the use of encryption mechanisms, it has been shown vulnerable to side-channel attacks which reveal information about its contents. Although various countermeasures against such attacks have been proposed, the lack of mathematical foundations for reasoning about information leaks in web traffic has prevented the development of provably secure countermeasures.

In this thesis, we develop a novel framework for reasoning about information leakage in web traffic. We propose flexible models of web applications, web browsing and web traffic, and develop two security models based on different notions of security: (1) an attacker's knowledge about the possible secret values contained in the web traffic, and (2) the probability of an attacker to guess the secret values. In those models, we formalize countermeasures and network protocols, defining them as compositions of several basic building blocks.

We use this framework to perform formal analysis of various aspects of information leakage in web traffic. We design principles for countermeasure composition which strengthens security, as well as investigate the utilization of structural properties of web applications to design countermeasures which provide strong security guarantees. Additionally, we leverage and extend results from quantitative information flow to design algorithms for calculating security guarantees. Furthermore, we investigate the differences in the proposed security models, as well as the role of caching to the security of web applications.

Finally, we perform two case studies empirically measuring the performance of countermeasures to a sample of web applications. We propose a methodology for dealing with computational challenges arising during practical analysis, and discuss the results of the case studies.

Acknowledgements

First of all, I am thankful to Boris Köpf for pointing me to the problem of side-channel attacks in web applications, for the numerous fruitful discussions of the challenges the problem presents, for the motivating guidance and advice. I am also grateful to Michael Backes for the helpful comments to my work, as well as for the inspiring lectures and cooperation. Furthermore, I am grateful to the IMDEA Software Institute for providing its facilities and a friendly working atmosphere, as well as for the partial funding of my work. This thesis would not have been possible without the love and understanding of my girlfriend Elissaveta who has encouraged me during my hard times in the last year, as well as the support and patience of my parents Nelly and Valentin, my sister Cveta, and all my friends.

This work is a product of my work in Madrid, Sofia, Saarbrücken, New York City, Mainz, and Bonn, mainly in 2011. I am grateful to all friends and relatives who have hosted me in this period.

Contents

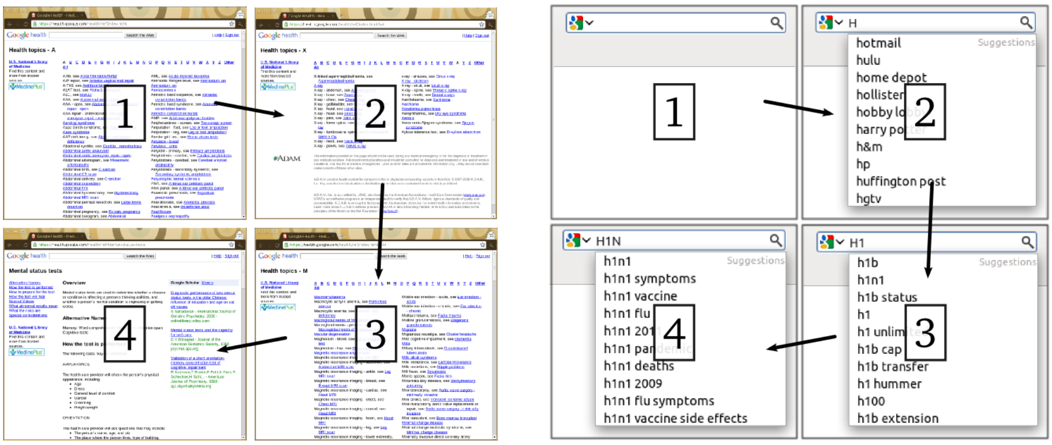
1 Introduction

The versatility of the Internet today offers its users the opportunity to pursue their various interests online, to exchange ideas with other users, and to perform everyday activities with several mouse clicks, e.g. shopping, paying bills, or even consulting a doctor. Performing all those activities online causes a huge amount of traffic to be transmitted over the Web, and this traffic often contains sensitive information about users. To limit unauthorized access to their sensitive information, security-aware Internet users can force their traffic to be encrypted by accessing web applications which offer TLS encryption, or by sending their data through an encrypted tunnel.

While state-of-the-art encryption mechanisms make data practically unreadable for unauthorized parties, they fail to provide sufficient protection from traffic analysis which inspects patterns in the *volume* of encrypted data, as demonstrated by an array of attacks [11, 51, 30, 3, 39, 16, 18, 29, 41, 10]. Various countermeasures against such attacks have been proposed [11, 51, 30, 39, 18, 41, 10, 55, 42], however those countermeasures are often targeted against specific attacks, and a sound theoretical foundation for reasoning about their effectiveness is still lacking. More specifically, the design of tools for the formal analysis of information leaks in web applications, as well as the development of general-purpose, provably secure countermeasures, has been an open problem.

1.1 Our contributions

In this thesis, we devise a novel framework for reasoning about information leaks in web applications, which we use to perform formal and empirical analysis of web applications. We concentrate on two scenarios: *web navigation*, where a user is navigating in the Web by following hyperlinks (see Figure 1.1(a)), and *auto-suggest*, where a user is typing a word into an auto-suggest input field (see Figure 1.1(b)). We assume that the communication between the user's web browser and the web server is encrypted, and that a passive attacker has physical access to encrypted Internet traffic. Whenever a user performs an action such as navigating to a webpage or typing a character in the input field, messages are exchanged between the browser and the server. The attacker then observes a burst of network packets, and despite encryption is able to distinguish the size and the direction of each transferred network packet. The attacker uses this information to uncover certain (secret) information about the encrypted data.



(a) Web-navigation (b) Auto-suggest input fields

Figure 1.1: The attack scenarios

In the proposed framework, we model a web application as a directed graph, and a user's browsing behavior as a path in this graph. Edges correspond to actions a user can take, and vertices correspond to the messages that need to be transferred in order to display the result of those actions to the user. We model observable patterns in web traffic as a function of the vertices in the graph, and the information which is to be kept secret as a function of the paths in the graph. To allow reasoning about security properties of web applications, we devise a *possibilistic security model*, which considers the secret values which to an attacker look possible generators of the observed patterns in traffic. Additionally, we extend this model to a *probabilistic security model*, which accommodates potential stochastic behavior of users and stochastic traffic patterns. In the probabilistic model, we use information theoretic definitions to measure the probability that an attacker can guess the secret values. Furthermore, we capture mechanisms that change observations, such as countermeasures and network protocols, and call them collectively *network fingerprints*. We identify several basic network fingerprints, which we use as building blocks for more complex network fingerprints, and this allows modeling of a landscape of countermeasures and network protocols.

We use the proposed framework to perform formal analysis of information leaks in web applications, where we consider both possibilistic and probabilistic security guarantees. We obtained the following results:

- First, we show that a proper composition of basic network fingerprints results in a network fingerprint which gives stronger security guarantees. In particular, this means that applying a countermeasure on top of another countermeasure can only strengthen security, and if an attacker observes traffic which is modified by a network protocol, he does not learn additional information about the secret.

- Second, we show that a property of the graph structure – the minimal out-degree of the vertices in the graph G (denoted $\delta(G)$), can be used to design a parametrized countermeasure which gives lower bounds on the possibilistic security guarantees. These lower bounds grow polynomially in the parameter k and in $\delta(G)$, and exponentially in the length of a path in the graph.

- Third, we leverage and extend existing results for quantitative information flow to simplify the calculation of probabilistic security guarantees for deterministic network fingerprints.

- Fourth, we analyze the possibilistic and probabilistic security guarantees of network fingerprints which add more ambiguity (e.g., by randomizing a deterministic countermeasure), as well as of adding edges to graphs (e.g., by adding hyperlinks between webpages). This analysis underlines the differences between possibilistic and probabilistic security guarantees, and indicates that probabilistic security captures better potential security vulnerabilities.

Additionally, we incorporate browser caching into our models, and discuss its controversial role for security: employing a cache can both be helpful or harmful to security.

Based on the proposed framework and on our formal results, we present a methodology for practical evaluation of countermeasures, and develop simulation approaches for dealing with computational challenges. We demonstrate this methodology in two case studies where we analyze the performance of a countermeasure to a sample of web applications, in terms of security and overhead.

1.2 Related work

In 1997, Wagner and Schneier [53] considered the analysis of the volume of network traffic flow as a possible vulnerability in the SSL 3.0 protocol, and stated that ignoring such traffic analysis "seems like a reasonable design decision" of the protocol's designers. Nevertheless, in the following years an array of attacks which exploit patterns in the volume of encrypted web traffic to extract user information was shown.

To the best of our knowledge, the first attacks utilizing the volume of encrypted web traffic were proposed in 1998 by Cheng and Avnur [11], and since then there have been more than ten published attacks of this nature [51, 30, 3, 39, 16, 18, 29, 41, 10]. There has been an evolution of the information those attacks are designed to extract, as well as of the settings of those attacks. While the attack presented in [11] is targeted at identifying a webpage within a website accessed through HTTPS, later attacks aim at identifying the *website* a user is visiting when browsing through an encrypted tunnel [51, 30, 3, 39, 16, 18, 29, 41], and the assumed setting of those attacks is an HTTPS connection to an anonymizing proxy [51, 30, 18], an

SSH or a VPN connection to an anonymizing proxy [3, 39, 29, 41], data contained in anonymized NetFlow records [16], onion routing or web mixes [29]. In 2010, Chen et al. [10] turn the focus of their attacks to web applications accessed through HTTPS and WPA, and those attacks aim at extracting information about user's health conditions and financial status.

In [11, 51, 30, 39, 18, 41, 10], along with presenting attacks, the authors discuss countermeasures for those attacks. Additionally, the main focus of two recent works [55, 42] is the design of countermeasures for dealing with previous attacks. An often proposed countermeasure is *padding*, i.e., adding noise to observations, as found in [11, 51, 30, 39, 18, 41, 10]. A different approach proposed by [51] and implemented by [55] changes patterns of observations corresponding to one website to look like patterns corresponding to another website, which allows sensitive traffic to be camouflaged as traffic coming from certain popular services. Note that there is confusion in the term used for this technique: while [51] call this approach *mimicking*, in [55] it is called *morphing*. [51] use the term "morphing" to describe a different approach: techniques which make patterns in observations different from ones expected by the attacker, e.g. by utilizing features of the HTTP protocol; features of the TCP and HTTP protocols were utilized in the techniques proposed by Luo et al. [42] used to build the HTTPOS system which offers browser-side protection from traffic analysis. In our work, we propose a framework which allows building previously discussed and novel countermeasures, using basic countermeasures as building blocks.

To evaluate the effectiveness of proposed techniques, [11, 51, 39, 55, 42] perform empirical evaluation, where the performance of certain attacks is compared before and after the application of the countermeasures. Additionally, [42] inspect the design of certain attacks, showing that proposed countermeasures make the studied attacks ineffective. In contrast to those works, we offer a general-purpose framework for formally reasoning about countermeasures. It does not assume the use of particular attacks, and security is measured in the amount of information leaked from the produced observations.

Besides attacks and mitigation techniques, recent work also handles the detection and quantification of information leaks in web traffic. [56] present the first language-based approach for securing web applications, using a combination of taint-based analysis and repeated sampling to estimate the information revealed through traffic patterns. [9] implement an approach which crawls web applications in order to perform black-box detection of vulnerabilities. They view an attacker as a classifier and quantify information leaks in terms of classifiability, using an entropy-based measurement, as well as a measurement based on the Fisher criterion.

Information-theoretic approaches related to the probabilistic security model in our work were used for modeling and mitigating side-channel attacks against cryptographic functions [36, 37].

Outline

The remainder of this work is organized as follows. Chapter 2 presents technical aspects of web browsing, as well as mathematical foundations needed in the rest of the work. In Chapter 3, we present a formal model of information leaks in web applications, which we extend with traffic patterns caused by network protocols and countermeasures in Chapter 4. We perform formal analysis of leakage in web traffic in Chapter 5. In Chapter 6, we extend our models to capture browser caching. Chapter 7 performs practical evaluation of countermeasures, and we conclude in Chapter 8.

2 Preliminaries

This chapter reviews the necessary background information used in the rest of this work. We introduce the technical properties of web browsing which make information leaks possible, and introduce necessary mathematical concepts.

2.1 Technical aspects of web browsing

In this section, we discuss technical properties of web browsing. We define the basic concepts, and discuss the patterns in traffic which enable attackers to perform analysis on encrypted traffic.

2.1.1 Web browsing

We define *web browsing* as the process of handling data on the Web, and the user's interface for web browsing – a *web browser* – is an application which resides on a personal computer or a mobile device. Web browsing usually entails *data retrieval*, the process of a user navigating through the Web in order to download data such as text, images, or video, and is sometimes supplemented by *data provision* – the process of uploading data, for example by publishing messages and images in web forums. The data is stored on web servers and is transported between browsers and servers through computer networks. The most commonly used protocol for web browsing is HTTP, although browsers usually support further protocols such as FTP, Telnet, SSH, or protocols for email transfer (POP, SMTP, IMAP). In the current work, we concentrate on information leaks in the data retrieval process through HTTP.

A typical web browser (such as Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Opera) contains a main window which displays the downloaded content, as well as navigation functionalities: an address bar for specifying URLs explicitly, bookmarks, buttons for going back and forward in the navigation history, and a button for reloading a webpage. A user can interact with the browser by either using its navigation functionalities, or by performing certain *user actions*: usually keyboard or mouse input. A user action either changes the content displayed to the user (e.g., when typing in text or triggering the execution of a browser script), or induces an HTTP request to the server. A web server (such as Apache, IIS, nginx, or GWX) awaits incoming HTTP requests and generates responses containing the requested resource or error messages. If the requested resource contains server scripting, the web server makes sure the supported scripts are executed.

An example of a typical user action is clicking on a hyperlink to load the webpage `index.html` hosted on the server `example.com`, which would incur the following HTTP messages to be exchanged:

(1) the HTTP request:

```
GET http://example.com/index.html HTTP/1.1
[request headers]
```

(2) the HTTP response:

```
HTTP/1.1 200 OK
[response headers]
[response payload]
```

Besides `GET`, further HTTP request methods are possible, e.g. `POST` for submitting data to be processed by the server; besides `200 OK`, further response codes are possible, e.g. the `404 Not Found` error code. In the example above, the response payload will consist of the `index.html` file. After obtaining (a part of) this file, the browser parses it, displays its (properly formatted) contents on the screen, and if the file includes additional files, such as images, JavaScript or Stylesheet files, the browser issues `GET` requests for them as well, usually in the order in which they are included in the HTML file.

Throughout this work we use the term *web application* to denote an application which has two parts – a client-side, running in the web browser, and a server-side, running in the web server, where the two sides communicate through HTTP messages. A web application can be a website containing only static webpages connected through links, or an interactive web service containing Ajax elements.

2.1.2 Attacker's view of web traffic

Any party which has physical access to web browsing traffic can perform analysis on it, and we call such analysis an *attack*, and a traffic analyst an *attacker*. In practice, physical access to our Internet traffic can be easily gained by various parties: the person sitting next to us in an Internet café, a company's sysadmin, by the Internet providers involved in the data transportation, the regulatory authorities in our country, the regulatory authorities of all other countries where our traffic passes through.

The browsing traffic observed by an attacker corresponds to a sequence of HTTP packets exchanged between a web browser and a web server. During transportation, an HTTP messages passes through the different layers of the Internet protocol stack: the TCP and IP protocols, and link layer protocols (e.g., Ethernet or PPP). Those protocols encapsulate the packet with specific headers containing information needed for transportation, most importantly the addresses of the sending and receiving parties (e.g. a TCP port number, an IP address, a MAC address). Additionally, protocols may cause packets to be segmented, i.e., divided into smaller packets on entering, and reassembled on leaving the scope of the protocol.

Whenever a user performs an action which triggers an HTTP request, an attacker can observe a burst of packets which are exchanged between the browser and the server. Figure 2.1 sketches an unencrypted Ethernet packet captured by an attacker. Given such a burst of packets, the attacker can obtain the original transmitted HTTP messages by reassembling the packets and removing the unnecessary headers.

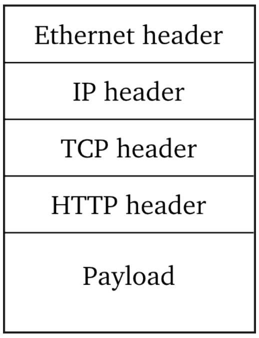


Figure 2.1: Unencrypted Ethernet packet

To prevent attackers from reading the content of the transmitted data, standard protection mechanisms relying on cryptographic primitives can be deployed. In our work, we assume that encryption is applied to all messages containing HTTP requests and responses, and in the following we discuss several state-of-the-art protection mechanisms and the traffic which an attacker observes when they are applied.

HTTPS connection to web server

A frequently used protection mechanism consists of setting up an encrypted HTTPS connection between a browser and a web server. This approach uses the TLS protocol (or its predecessor, SSL) to provide the encrypted connection, and allows web service providers to force the whole communication between browser and server to be encrypted. Figure 2.2 sketches an Ethernet packet containing an HTTPS message captured by an attacker: the whole HTTP message is encrypted, however headers corresponding to lower layers can be read in clear, as they are not encrypted. In particular, an attacker could read the source and destination IP addresses and port numbers, and from them infer which website a user is visiting. The visited website can also be inferred by inspecting unencrypted DNS messages. From TCP headers, an attacker can additionally gain the information needed to reassemble packets corresponding to the same HTTP message.

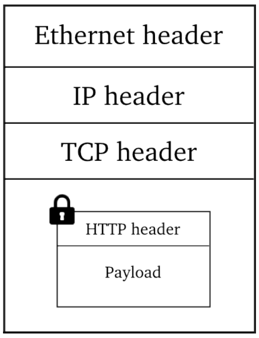


Figure 2.2: An Ethernet packet with HTTPS encryption

Encrypted connection to proxy server

The second protection mechanism we consider is tunneling the Internet traffic through an encrypted connection to a proxy server. The encrypted connection can be established using an SSH tunnel, a VPN connection, or an HTTPS connection to a web proxy. Using this mechanism, the data between the browser and the proxy server is encrypted. The resulting Ethernet packets are sketched on Figure 2.3. Here, along with the HTTP messages, TCP and IP headers are encrypted, as well as DNS messages, and thus information about the website visited by a user is concealed from the attacker. Additionally, an attacker cannot distinguish which packets are parts of which HTTP messages, and reassembly of packets is not directly possible. Furthermore, packets corresponding to background traffic may be misinterpreted as corresponding to the targeted browsing session.

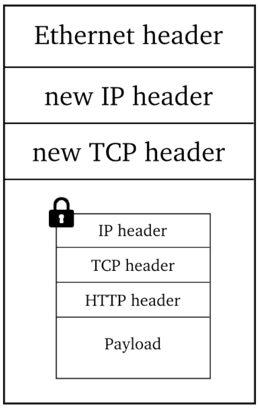


Figure 2.3: An Ethernet packet with an encrypted tunnel to a proxy server

Wireless encrypted traffic

The data between a computer and a wireless router can be encrypted using the WPA2 protocols (or, its predecessors WPA and WEP). The observations an attacker can make are similar to those from an encrypted connection to a proxy server (see above), and the corresponding Ethernet packet is sketched on Figure 2.4.

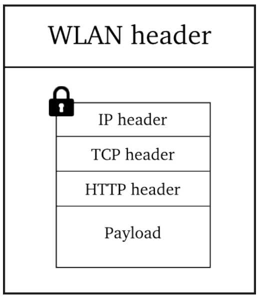


Figure 2.4: A WLAN packet with WPA2 encryption

2.1.3 Caching

In order to save bandwidth and to display certain content faster to users, instead of fetching objects from the web servers, they can be fetched from *web caches.* Those caches are usually deployed by a browser or by proxies, and if certain content is downloaded from a cache instead of from a web server, it will be missing in the traffic available to an attacker. In the following chapters we assume that caches are disabled, and we return to browser caching in Chapter 6

2.2 Mathematical background

This section introduces mathematical concepts used in the current work. We present information theoretic definitions and Markov chains, as well as introduce the used notation.

2.2.1 Information theoretic definitions

Information theory is a mathematical discipline which allows measuring the *amount of information* contained in random variables. It was introduced by Claude Shannon in 1948 [47] to reason about data compression and data communication, and has found a broad usage in various fields, such as physics [32], bioinformatics [50], cryptography [5]. Information theoretic definitions are standard tools in *quantitative information flow analysis* (e.g., see [13, 35]) – a method for quantitatively reasoning about secrecy properties in a system, e.g. in a computer program. Typically, the information theoretic approach for quantitative information flow defines random variables X representing *high* (or secret) input to the system, and Y representing *low* (or publicly accessible) output, and the system's behavior is characterized by the conditional probability distribution $P[Y|X]$, i.e., the probability of the system's output given a certain input. To measure security in a system, three measures are defined, which represent (1) the *initial uncertainty* about the input X, or how much information is secret; (2) *remaining uncertainty* about the input X when the output Y is known, or how much information remains secret if one inspects the public output; (3) the *information leaked* from X to Y. For those measures, the following equality should hold:

$$\text{initial uncertainty} = \text{remaining uncertainty} + \text{information leaked},$$

and both the remaining uncertainty and the information leaked can be used to measure the security of a system. Here, a system will be considered to guarantee maximal security if there is no information leaked, and it will be considered with minimal security if there is no remaining uncertainty. Using this approach, a system designer should try to design systems which provide as high remaining uncertainty as possible.

Information theoretic definitions often used to represent the three concepts are based on Shannon entropy and min-entropy. Smith [48] has demonstrated that the probability that an attacker can guess the high value in one try after inspecting the low value is better captured by the more conservative definitions based on min-entropy[1].

In our work we adopt min-entropy-based definitions, which are presented in the following.

Let X be a random variable with a finite set of possible values $\mathcal{X}$ representing the system's high input, and Y be a random variable with a finite set of possible values $\mathcal{Y}$ representing the system's high output. To represent the initial uncertainty about X, we use the min-entropy of X, written $H_\infty(X)$, defined in the following.

Definition 1. *The* min-entropy *of a random variable X is defined as*

$$H_\infty(X) = -\log_2 \max_{x\in\mathcal{X}} P[X = x].$$

To represent the remaining uncertainty about X given Y, we use the *conditional min-entropy of X given Y*, written $H_\infty(X|Y)$, defined in the following.

Definition 2. *The* conditional min-entropy *of X given a random variable Y, is*

[1] semiformally, min-entropy can be seen as more conservative than the Shannon entropy as it is computed using the maximum probability of X, while Shannon entropy takes an expectation over the probability of X.

defined as

$$H_\infty(X|Y) = -\log \sum_{y\in\mathcal{Y}} P[Y=y] \max_{x\in\mathcal{X}} P[X=x|Y=y].$$

This definition captures the behavior of attackers which are allowed to make only one guess: such attackers would pick the value of x which maximizes the probability $P[X=x|Y=y]$. The conditional min-entropy measures the logarithm of the expected value over Y of this probability.

To represent the information leaked from X to Y, we use the mutual (min-)information $I_\infty(X;Y)$ [2], defined in the following.

Definition 3. *The* mutual (min-)information *of X and Y is defined as*

$$I_\infty(X;Y) = H_\infty(X) - H_\infty(X|Y).$$

In the following, we consider the case where a third random variable Z is given. The following definition of *conditional mutual (min-)information* of X and Y given Z can be interpreted as the reduction of the uncertainty about the input X when the output Y is inspected, if the value of Z is previously known.

Definition 4. *The conditional mutual (min-)information of X and Y given Z is defined as $I_\infty(X;Y|Z) = H_\infty(X|Z) - H_\infty(X|Y,Z)$.*

The following Lemma establishes the relationship between the different information theoretic definitions given three random variables X, Y, Z. This relationship is sketched on Figure 2.5.

Lemma 1.

$$I_\infty(X;Y,Z) = I_\infty(X;Z) + I_\infty(X;Y|Z) = I_\infty(X;Y) + I_\infty(X;Z|Y).$$

Proof. Applying the definitions of mutual (min-)information and conditional mutual (min-)information, we obtain:

$$(1)\ I_\infty(X;Y,Z) = H_\infty(X) - H_\infty(X|Y,Z);$$

$$\begin{array}{rcl}(2)\ I_\infty(X,Z) + I_\infty(X;Y|Z) & = & H_\infty(X) - H_\infty(X|Z) + H_\infty(X|Z) - H_\infty(X|Y,Z) \\ & = & H_\infty(X) - H_\infty(X|Y,Z);\end{array}$$

$$\begin{array}{rcl}(3)\ I_\infty(X,Y) + I_\infty(X;Z|Y) & = & H_\infty(X) - H_\infty(X|Y) + H_\infty(X|Y) - H_\infty(X|Y,Z) \\ & = & H_\infty(X) - H_\infty(X|Y,Z).\end{array}$$

Therefore, the equality holds. □

[2] Note that unlike in the case of Shannon-entropy, the min-entropy definition of mutual information is not symmetric, i.e., the following equality must *not* hold: $I_\infty(X;Y) = I_\infty(Y;X)$.

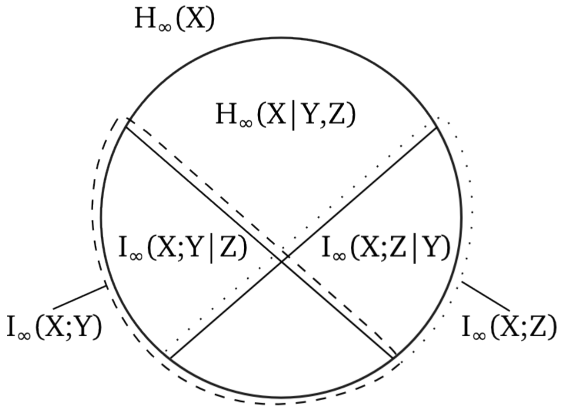


Figure 2.5: Relationship between the information theoretic measures $H_\infty(X)$, $H_\infty(X|Y,Z)$, $I_\infty(X;Y)$, $I_\infty(X;Z)$, $I_\infty(X;Y|Z)$, $I_\infty(X;Z|Y)$, for the random variables X, Y, Z

2.2.2 Markov chains

Markov chains are a powerful framework for modeling random processes. In the following, we define Markov chains, and show a well known result for Markov chains.

Definition 5. *The random variables X, Y, Z form a* Markov chain*, written $X \to Y \to Z$, if for all x, y, z, $P[Z = z|X = x, Y = y] = P[Z = z|Y = y]$ holds.*

Lemma 2. *If random variables X, Y, Z form a Markov chain $X \to Y \to Z$, they also form a Markov chain $Z \to Y \to X$.*

Proof.

$$\begin{aligned}
P[X = x|Y = y, Z = z] &= \tfrac{P[X=x,Y=y,Z=z]}{P[Y=y,Z=z]} \\
&= \tfrac{P[X=x,Y=y,Z=z]}{P[Y=y,Z=z]} \cdot \tfrac{P[X=x,Y=y]}{P[X=x,Y=y]} \\
&= P[Z = z|X = x, Y = y] \cdot \tfrac{P[X=x,Y=y]}{P[Y=y,Z=z]} \\
&\overset{(*)}{=} P[Z = z|Y = y] \cdot \tfrac{P[X=x,Y=y]}{P[Y=y,Z=z]} \\
&= \tfrac{P[Z=z,Y=y]}{P[Y=y]} \cdot \tfrac{P[X=x,Y=y]}{P[Y=y,Z=z]} \\
&= P[X = x|Y = y].
\end{aligned}$$

$(*)$ follows from $X \to Y \to Z$. The remaining steps follow from applying the definition of conditional probability. □

2.2.3 Notation

In the following, we introduce some notation used in the remainder of this work:

- $\sqsubseteq$ denotes the *subsequence relation* – for a set A and $x, y \in A^*$, $x \sqsubseteq y$ means that x can be derived from y by deleting some of the elements in y but still preserving the order of the remaining elements;

- $\sqcup$ denotes the *concatenation of sequences* – given tuples $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{y} = (y_1, \ldots, y_n)$, $\vec{x} \sqcup \vec{y} = (x_1, \ldots, x_m, y_1, \ldots, y_n)$.

- the function *set* $: A^* \to \mathcal{P}(A)$ denotes the *set representation of a tuple*, and is defined as $\mathit{set}(x_1, \ldots, x_k) = \{x_1, \ldots, x_k\}$;

- given a tuple $\vec{x} = (x_1, \ldots, x_n)$, $\pi_i(\vec{x})$ denotes the i-th element x_i, and $\Pi(\vec{x})$ denotes the set of all permutations over $(x_1, \ldots, x_n)$;

- $\mathit{paths}(G, \ell)$ denotes the set of all paths in G of length ℓ, and is defined as $\mathit{paths}(G, \ell) = \{(v_1, \ldots, v_\ell) \mid \forall i \in [\ell - 1] : (v_i, v_{i+1}) \in E\}$;

- $[n]$ denotes the set $\{1, \ldots, n\}$.

3 Modeling leaks in web applctions

In this chapter, we present a framework for reasoning about information leaks in web applications. First, we define models of web applications and corresponding traffic patterns. Second, we define a security model encompassing the specification of secret values, as well as the quantification of leaks in web applications. For this, we consider both a possibilistic, and a probabilistic view on security.

3.1 Web applications and application fingerprints

We model a web application as a directed graph

$$G = (V, E),$$

where an edge represents an action a user can make, and a vertex corresponds to the sequence of web-objects exchanged between a user and a server. We define the set of vertices as $V \subset (W \times \{\uparrow, \downarrow\})^*$, where W is the set of all possible web objects, $\uparrow$ denotes that an object is transmitted from the user to the server, and $\downarrow$ denotes that an object is transmitted from the server to the user.

Throughout our work web-traffic is assumed to be encrypted, with sizes being the only visible observations. Let the *size of a web-object* be the function

$$s : W \to \mathbb{N}.$$

We define the *applicatin fingerprint*

$$f_{app} : V \to O,$$

to model the observable part of transmitted web-objects (i.e., size and direction), where $O \subset (\mathbb{N} \times \{\uparrow, \downarrow\})^*$ is the set of observations. The application fingerprint preserves the direction of objects, and is defined as

$$f_{app}(v) = f_{app}(w_1, d_1, \ldots, w_n, d_n) = (s(w_1), d_1, \ldots, s(w_n), d_n).$$

In the following we formalize web applications, considering the two scenarios studied in the current work: (1) web-navigation, and (2) the auto-suggest scenario.

3.1.1 The web-navigation scenario

In the web-navigation scenario, a user is navigating in the Web by following hyperlinks, and we assume that every time a user selects a hyperlink, the same web-objects are exchanged. We will refer to the collection of exchanged objects induced by clicking on a link as a *webpage*[1]. In the web-navigation scenario, the web-application is modeled by the *web-graph* $G = (V, E)$, where vertices represent webpages, and an edge (u, v) indicates that the webpage u contains a link to webpage v (and thus, a possible user action is following this link). Examples of web-graphs are the Internet-graph, and the graphs corresponding to particular websites. Given a web-object w which is downloaded from the server, we model an HTTP request as the function $get : W \to W$. Thus, if a webpage incurs the downloading of web-objects $w_1, \dots, w_n$, we will have $v = [(get(w_1), \uparrow), (w_1, \downarrow), \dots, (get(w_n), \uparrow), ((w_n), \downarrow)]$. For example, we may have a webpage which incurs the downloading of the objects a.html, style.css, script.js, img.jpg, video.flv, which will correspond to a vertex

$$\begin{aligned} v = \; & [(get(\text{a.html}), \uparrow), (\text{a.html}, \downarrow), (get(\text{style.css}), \uparrow), (\text{style.css}, \downarrow), \\ & (get(\text{script.js}), \uparrow), (\text{script.js}, \downarrow), (get(\text{img.jpg}), \uparrow), (\text{img.jpg}, \downarrow), \\ & (get(\text{video.flv}), \uparrow), (\text{video.flv}, \downarrow)]. \end{aligned}$$

Accordingly, the application fingerprint here is

$$\begin{aligned} f_{app}(v) = \; & [(s(get(\text{a.html})), \uparrow), (s(\text{a.html}), \downarrow), (s(get(\text{style.css})), \uparrow), (s(\text{style.css}), \downarrow), \\ & (s(get(\text{script.js})), \uparrow), (s(\text{script.js}), \downarrow), (s(get(\text{img.jpg})), \uparrow), (s(\text{img.jpg}), \downarrow), \\ & (s(get(\text{video.flv})), \uparrow), (s(\text{video.flv}), \downarrow)]. \end{aligned}$$

3.1.2 The auto-suggest scenario

In the auto-suggest scenario, a user types some input x into an input field, and a drop-down list of suggestions appears. Let Σ be the input alphabet, let $D \subseteq \Sigma^*$ be the dictionary of possible suggestions, let $I \subseteq \Sigma^*$ be the dictionary of possible words a user can type, and let the auto-suggest functionality be implemented by the function $suggest : I \to D^*$. Typing an input x, a list of suggestions $suggest(x)$ is either empty, corresponding to the empty word ε, or contains a selection of words from D: words starting with x, words containing the substring x, words semantically related to x, or words chosen by some other criterion. We assume that initially the auto-suggest field is empty, and at this point the input x equals to ε. Let c_i be the i-th character typed in by the user. When the user types in c_i, a query containing $x = c_1 c_2 \dots c_i$ is sent to the server, and the list of suggestions is retrieved.

In this scenario, we define the web-application $G = (V, E)$ as a tree with $V = (I \times$

[1] In the literature, the term *web-request* is used for a similar concept, see [12].

$\{\uparrow\}) \times (D^* \times \{\downarrow\})$, the vertex $[(\varepsilon, \uparrow), (suggest(\varepsilon), \downarrow)]$ as root, and there exists a (u, v) edge in E if and only if $v = [(x\sigma, \uparrow), (suggest(x\sigma), \downarrow)]$, for $u = [(x, \uparrow), (suggest(x), \downarrow)]$ and $\sigma \in \Sigma$. The edges here correspond to the user action of typing in characters; thus, we implicitly forbid deleting characters or moving the cursor. Note that if in the vertices of G we consider only the first components, i.e., the resulting tree is a *prefix tree* (or *trie*, see [15]).

In the auto-suggest scenario, the application fingerprint corresponds to the size of the list of suggestions. In practice, this list is usually transmitted as a file in XML, CSV, or some other format. For a certain format, we define the *size of an auto-suggestion list* as a function

$$s_a : D^* \to \mathbb{N}.$$

Given a word $v = [(x, \uparrow), (suggest(x), \downarrow)] \in V$, we define its application fingerprint as

$$f_{app}(v) = [(s(x), \uparrow), (s_a(suggest(x)), \downarrow)].$$

3.1.3 Remarks

Let a user's *navigation path* of length ℓ be a path $(v_1, \ldots, v_\ell)$ in the graph G corresponding to visited webpages or typed-in characters. We assume that the attacker can clearly distinguish which observations correspond to which vertex, and thus knows the path length ℓ (e.g. by inspecting the timing of downloaded objects). Therefore, the attacker's view of a navigation path $(v_1, \ldots, v_\ell)$ is the corresponding sequence of observations $(o_1, \ldots, o_\ell)$, which we call an *observation path*. In the following, we use the notation $\vec{v}$ for $(v_1, \ldots, v_\ell)$, $\vec{o}$ for $(o_1, \ldots, o_\ell)$, and $[\ell]$ for the set $\{1, \ldots, \ell\}$.

In the remainder of this chapter, we develop the model of leaks in web-traffic under the assumption that we have a powerful attacker who has access to the application fingerprint of vertices, i.e., observes the sizes of web-objects. In Chapter 4, we extend our definitions to more realistic attackers who do not know the exact sizes of web objects, but can only observe sizes of network packets.

3.2 Security model

Given a web application G and an application fingerprint f_{app}, in this section we build the theory towards answering the question "How vulnerable is G to attacks?". This question should be answered with care: a too liberal approach may put up with too many privacy leaks, while a too conservative approach may prove impractical. We start by allowing the specification of the secret, which is the information a user wants to keep private. Afterwards, we present two models for quantifying information leakage in web browsing traffic: a possibilistic one and a probabilistic one.

3.2.1 Specifying the secret

If an attacker inspects a sequence of observations $(o_1,\dots,o_\ell) = (f_{app}(v_1),\dots,f_{app}(v_\ell))$, the most he can extract from those observations is the exact navigation path $(v_1,\dots,v_\ell)$. However, the whole path may not consist of sensitive information, for example only the answer to the question "Has v been visited?" may be considered secret. To capture this, we define the *secret specification* as a function *sec* which takes a user's navigation path $(v_1,\dots,v_\ell)$ and returns the secret information contained in this path. Informally, the secret specification is a question about the navigation path whose answer consists of the information one wants to protect. From the perspective of information-flow security, this corresponds to the *high* parts of a path. In the following, we define several example secret specifications.

Specifying parts of the navigation path as secret

1. Which is the exact path?

   The exact path is the maximal amount of information the attacker may extract from a sequence of observations in our model. This secret specification is defined as the identity function:

   $$sec_1(v_1,\dots,v_\ell) = (v_1,\dots,v_\ell).$$

2. Which pages has the user visited?

   The order in which pages were visited may be irrelevant to the attacker. Then we obtain the secret specification

   $$sec_2(v_1,\dots,v_\ell) = \{v_1,\dots,v_\ell\}.$$

3. Which are the last $\ell - x$ visited pages?

   The attacker may not be interested in the first x pages visited by the user, for example in a highly interlinked web portal. Then we obtain the secret specification

   $$sec_3(v_1,\dots,v_\ell) = \{v_{x+1},\dots,v_\ell\}.$$

4. Which is the i-th visited page?

   $$sec_4(v_1,\dots,v_\ell) = v_i$$

5. How many times have we visited page v?

$$sec_5(v_1, \dots, v_\ell) = |\{i \mid v_i = v\}|$$

6. Binary questions

   The attacker may also ask a binary (yes-no) question. The secret specification of such a question will be defined as follows:

$$sec_6(v_1, \dots, v_\ell) = \begin{cases} 1 & \text{if condition is fulfilled} \\ 0 & \text{otherwise.} \end{cases}$$

   For example, the question "Has page v been visited?" can be asked, which will be defined as

$$sec_6^v(v_1, \dots, v_\ell) = \begin{cases} 1 & \text{if } v \in \{v_1, \dots, v_\ell\} \\ 0 & \text{otherwise.} \end{cases}$$

   Analogously, many other binary questions can be asked, such as "Is the navigation path $(v_1, \dots, v_\ell) = (v'_1, \dots, v'_\ell)$?", "Is the i-th visited page v?", "Has v been visited before v'?", "Has a user visited a page containing more than 5 web objects?".

Note that in the auto-suggest scenario, because of the tree structure of a prefix tree, many of the questions an attacker may ask are equivalent to finding out the exact path $(v_1, \dots, v_\ell)$, or a subpath of it. For example, as each vertex $v \in V$ can be reached only following a unique path, the question "Has v been visited?" is equivalent to asking whether $(v_1, \dots, v_\ell)$ starts with this unique path.

Specifying the identity of websites as secret

If the visited website is not known to the attacker (e.g., if the traffic is tunneled through an anonymizing proxy, see Section 2.1.2), the question "Which website is the user currently visiting?" may be asked. The secret specification here is defined as follows. Let $G = (V, E)$ be the Internet-graph, and let $G_1 = (V_1, E_1), \dots, G_n = (V_n, E_n)$ be the graphs corresponding to all websites. Then, $V_1, \dots, V_n$ is a partition of V into disjoint blocks. The secret specification in this case will be the question "To which block(s) of the partition do $v_1, \dots, v_\ell$ belong?", or formally

$$sec_I(v_1, \dots, v_\ell) = (V'_1, \dots, V'_\ell), \text{ with } v_i \in V'_i \text{ and } V'_i \in \{V_1, \dots, V_n\}, \text{ for all } i \in [\ell].$$

Complex secret specifications

The previously discussed secret specifications represent questions about the navigation behavior of a user, which can be answered by inspecting the sequence of visited vertices and the graph structure, and in the case of website identification – by additionally inspecting the partition of the Internet-graph into websites. We call such secret specifications *pure* secret specifications. An attacker will often be interested not only in asking questions about visited locations in a graph, but also in extracting some sensitive information from the answer to such a question. For example, the attacker may use the answer to the question “Is the navigation path $(v_1, \ldots, v_\ell) = (v'_1, \ldots, v'_\ell)$?” to answer the question “Does this user read articles about lactose intolerance?”. Or, the answer to such a question may be used as features for a classifier which may answer questions such as “Does this user suffer from lactose intolerance?”, “Is this user taking part in demonstrations against the government?”, or “Is this user a terrorist?”.

To handle such questions, we define *complex* secret specifications as functions or pattern recognition systems which take as input the answer to a pure secret specification, and use some background knowledge to answer the questions. This background knowledge may be knowledge about the context and contents of the visited websites, or some training data. In the remainder of this work, we will only consider pure secret specifications, and we leave precise definition of complex secret specifications to future work.

3.2.2 Possibilistic security model

The first security model we present takes a *possibilistic* view on security. Given a secret specification *sec* and an observation path $\vec{o}$, it deals with the answer to the question “Which are the possible values of $sec(\vec{v})$ corresponding to certain observations?”. We start by defining the observations corresponding to a vertex.

Definition 6. *The* possible observations *of a vertex in G is the function*

$$g : V \to \mathcal{P}(O),$$

which returns a set containing all possible observations corresponding to this vertex.

We have assumed that a vertex v has a unique application fingerprint $f_{app}(v)$, and thus, we obtain $g(v) = \{f_{app}(v)\}$. In the discussion of countermeasures in Chapter 4, the definition of g will change, allowing more than one observation per vertex.

In the following we define the core of the possibilistic security model: the *ambiguity* about the user's navigation path, which is the set containing all secret values $sec(\vec{v})$ consistent with a given observation path.

Definition 7. *Given a sequence of observations $(o_1, \ldots, o_\ell)$ corresponding to a user's navigation path, and a secret specification sec, the* ambiguity K_{sec} *about the user's navigation path is defined as:*
$K_{sec}(G, n, (o_1, \ldots, o_\ell), g) =$
$\{sec(v_1, \ldots, v_\ell) \mid (v_1, \ldots, v_\ell) \in paths(G, \ell) \wedge \forall i \in [\ell] : o_i \in g(v_i)\}.$

When an attacker analyses an observation path, the ambiguity is the set containing all answers to the question defined by the secret specification. If the attacker can answer the question deterministically, i.e., knows that the navigation path is $(v_1, \ldots, v_\ell)$, then the cardinality of the ambiguity set is 1; if the attacker has no information what the answer to the question may be, the cardinality will be maximal. We consider the following applications of ambiguity:

1. Cardinality of the ambiguity set: the number of possible answers to the question defined by the security specification can be used as a quantitative measure of information leakage. We consider the following definitions:

   (i) minimal ambiguity cardinality (MAC) over al possible observations

   $$MAC := \min_{\vec{o}} K_{sec}(G, n, \vec{o}, g)$$

   (ii) average ambiguity cardinality (AAC) over all possible observations

   $$AAC := \frac{1}{\#\text{ possible observations}} \sum_{\vec{o}} |K_{sec}(G, n, \vec{o}, g)|$$

2. Subset inclusion: let a secret specification and two systems be given, where on a certain input, the first system produces an observation path with ambiguity set A_1, and the second system produces an observation path with ambiguity set A_2. If on all inputs we have $A_1 \subseteq A_2$, then the first system can be considered to leak more information than the second one.

3.2.3 Probabilistic security model

The problem of leakage in web browsing traffic may include stochastic components, e.g., attackers may use prior knowledge about the user's behavior to assist their judgment, assuming that a user acts according to a certain probability distribution. Such assumptions can be found in research on *web usage mining* [49], where in order to predict a user's behavior, a probability distribution on the user's navigation is assumed, which can be calculated by inspecting web log data or structural properties of web graphs [20].

To capture such stochastic components, we propose a probabilistic security model, which builds upon the possibilistic model presented in the previous section. We assume that the user navigates according to a previously known probability distribution, i.e., we are given a probability distribution on the paths of length ℓ that can be taken by the user. Let the random variable X with a set of possible values $\mathcal{X} = V^\ell$ describe the taken path, and let the random variable Y with a set of possible values $\mathcal{Y} = O^\ell$ describe the observations which an attacker sees. The observations are determined according to the conditional probability distribution $P[Y|X]$, which gives the probability that an attacker will see certain observations if a user has taken a certain navigation path. In the current model, we have deterministic observations, and thus given an $x \in \mathcal{X}$, we obtain $P[Y = y|X = x] = 1$ for $y = f_{app}(x)$, and $P[Y = y|X = x] = 0$ otherwise, and thus, we have $Y = f_{app}(X)$. In Chapter 4, we will discuss randomized countermeasures which break this functional relationship of X and Y. As discussed in Section 3.2.1, the secret specification is defined as a function *sec* which takes values from V^ℓ. Transferring this concept to the probabilistic model, if attackers inspect the value of Y, they want to extract information about the value of the (secret) random variable $sec(X)$. Thus, the knowledge of the attacker about the secret value will be captured by the conditional probability $P[sec(X)|Y]$.

Given random variables X and Y, and a secret specification *sec*, we characterize security properties using information theoretic definitions (see Section 2.2.1). We define the *initial uncertainty* about the user's navigation X under secret specification *sec* as the min-entropy of $sec(X)$:

$$\text{initial uncertainty} := H_\infty(sec(X)) = -\log_2 \max_a P[sec(X) = a].$$

We define the *remaining uncertainty* about the user's navigation X under secret specification *sec* given observations Y as the conditional min-entropy of $sec(X)$ given Y:

$$\begin{aligned} \text{remaining uncertainty} := \;& H_\infty(sec(X)|Y) \\ = \;& -\log_2 \sum_{\vec{o}} P[Y = \vec{o}] \max_a P[sec(X) = a|Y = \vec{o}]. \end{aligned}$$

We define the *information leaked* about the user's navigation X under secret specification *sec* given observations Y as the mutual (min-)information of $sec(X)$ and Y:

$$\text{information leaked} := I_\infty(sec(X); Y) = H_\infty(sec(X)) - H_\infty(sec(X)|Y).$$

Markov property

The random variable X can be decomposed into ℓ random variables $X_1, \ldots, X_\ell$, such that if $X = (v_1, \ldots, v_\ell)$, then $X_1 = v_1, \ldots, X_\ell = v_\ell$. If we assume that $X_1, \ldots, X_\ell$ form a Markov chain, i.e., for each $i \in [\ell]$, $P[X_{i+1} = v_{i+1}|X_i = v_i, X_{i-1} = v_{i-1}, \ldots, X_1 = v_1] = P[X_{i+1} = v_{i+1}|X_i = v_i]$, this will allow simplifying certain computations. For example, to compute the probability distribution of X given a graph G, only probabilities on each edge should be specified, and not probabilities on the whole paths. In the auto-suggest scenario, the Markov property holds, as because of the tree structure of the corresponding graph, if we are currently visiting vertex v, there is only one path from the root to v, and thus, the probability of moving from v to another vertex v' will be the same if we consider the previous vertices on the path from the root or not. In the web-navigation scenario such an assumption is more problematic, as it would mean that every time users are visiting a webpage v, they have the same preferences about which hyperlink to follow. However, a user may decide to which webpage to navigate taking into account all previously visited webpages, and if there are several different paths from the root to a vertex, the Markov property may be violated. Nevertheless, we note that the assumption that X is a (first or higher order) Markov chain can be found in the literature about prediction of user's web navigation (e.g. see [45, 20]).

3.2.4 Discussion of security models

We have presented two notions of security: a possibilistic one based on ambiguity, and a probabilistic one based on min-entropy. Relying on ambiguity allows comparing the possible secret values corresponding to given observations, and an ambiguity set of small cardinality indicates that from an observation, an attacker can quite accurately determine the secret values. However, an ambiguity set of a higher cardinality may give a false feeling of security, as it ignores stochastic components of the system. The probabilistic notions of security allow capturing such stochastic components, such as attacker's belief about the user's behavior, and as we will show in Chapter 4, the probabilistic model allows a more natural analysis of randomized countermeasures. While we will not give a clear verdict about whether possibilistic security guarantees are sufficient to capture the vulnerability of web browsing traffic to attacks or not, our formal analysis indicates that in some cases possibilistic definitions are too permissive (see Section 5.4 and Section 5.5). Note that for purposes different from the ones pursued in the current work, possibilistic notions of security have been shown to be inferior to notions of security incorporating probabilistic assumptions [52].

4 Network fingerprints

In the previous chapter, we presented a model of leaks in web traffic, where we assumed that an attacker can observe the application fingerprint of vertices, i.e., the exact sizes of web objects. Building upon those foundations, in the current chapter we handle *network fingerprints*, which allow attackers to view sizes of network packets. The network fingerprints can be different from the application fingerprints because of changes due to network protocols, or because of applied countermeasures. We define network fingerprints formally, and identify several basic network fingerprints which we will use as building blocks for network protocols and countermeasures.

4.1 Formal definitions

Network fingerprints change how observations look like in the eyes of an eavesdropper. We assume that network fingerprints are *stateless*: they change the observations corresponding to one vertex only, and do not take into account observations of previously visited vertices (the statelessness will be loosened in the discussion of caching, see Chapter 6). We start by defining possibilistic network fingerprints which extend the possibilistic security model (see Section 3.2.2), and further we define probabilistic network fingerprints which extend the probabilistic security model (see Section 3.2.3).

4.1.1 Possibilistic network fingerprints

In accord to the possibilistic security model from Section 3.2.2, we define possibilistic network fingerprints. Given a graph $G = (V, E)$ and an application fingerprint function $f_{app} : V \to O$, a *network fingerprint* is a function

$$f_{net} : O \to \mathcal{P}(O_1),$$

which is to be interpreted as a non-deterministic function from O to O_1. Both O and O_1 are sequences of integer sizes with a direction, i.e., $O, O_1 \subset (\mathbb{N} \times \{\uparrow, \downarrow\})^*$ (see Section 3.1). Non-determinism here captures potential randomization of a network fingerprint: a statement $f_{net}(o) = A$ means that f_{net} may output any of the values in A. As we want a non-deterministic function to be total, we forbid an o with $f_{net}(o) = \emptyset$. Given a network fingerprint f_{net}, there is trivially at least one corresponding deterministic network fingerprint $f^d_{net} : O \to O_1$, such that $\forall o \in O : f^d_{net}(o) \in f_{net}(o)$.

Network fingerprint composition

Network fingerprint composition allows applying one network fingerprint to the output of another network fingerprint, and for its definition we use function composition. The definition of network fingerprint composition is straightforward for deterministic network fingerprints with matching types: given network fingerprints $f'_{net} : O \to O_1$ and $f''_{net} : O_1 \to O_2$, the composed network fingerprint $f''_{net} \circ f'_{net} : O \to O_2$ is defined, with

$$(f''_{net} \circ f'_{net})(o) = f''_{net}(f'_{net}(o)).$$

To allow further compositions, in the following we define two natural extensions of functions: a natural extension to power sets which allows composition with non-deterministic network fingerprints, and a natural extension to tuples, which allows composition with network fingerprints returning tuples.

Natural extension to power sets To allow composing a non-deterministic network fingerprint $f'_{net} : O \to \mathcal{P}(O_1)$ with a deterministic network fingerprint $f''_{net} : O_1 \to O_2$, we define the *natural extension of a network fingerprint f''_{net} to power sets* as the function

$$\hat{f}''_{net} : \mathcal{P}(O_1) \to \mathcal{P}(O_2),$$
$$\hat{f}''_{net}(\{x_1, x_2, \dots\}) = \{f''_{net}(x_1), f''_{net}(x_2), \dots\},$$

and using this definition, the composition of f'_{net} and f''_{net} can be built as $\hat{f}''_{net} \circ f'_{net}$. To allow composition of two non-deterministic network fingerprints $f'_{net} : O \to \mathcal{P}(O_1)$ and $f''_{net} : O_1 \to \mathcal{P}(O_2)$, we define the natural extension of f''_{net} to power sets differently:

$$\hat{f}''_{net} : \mathcal{P}(O_1) \to \mathcal{P}(O_2),$$
$$\hat{f}''_{net}(A) = \bigcup_{x \in A} f''_{net}(x),$$

and using this definition, the composition of f'_{net} and f''_{net} can be built as $\hat{f}''_{net} \circ f'_{net}$. For ease of notation, in some cases we will use an overloaded notation f_{net} instead of $\hat{f}_{net}$, and will write $f''_{net} \circ f'_{net}$ instead of $\hat{f}''_{net} \circ f'_{net}$.

Natural extension to tuples Next, we define the *natural extension of network fingerprints to tuples*, which can be applied in two cases. First, if we have a network fingerprint $f'_{net} : O \to O_1^k$ returning k-tuples and a second network fingerprint $f''_{net} : O_1 \to O_2$, a composition $f''_{net} \circ f'_{net}$ will not be possible because the types of f'_{net} and f''_{net} do not match. However, it is possible to combine them in a natural way, by computing $f''_{net}(o_i)$ for all $o_i \in (o_1, \dots, o_k)$. We define the natural extension of f''_{net} to k-tuples as the function

$$\bar{f}''_{net} : O_1^k \to O_2^k$$

$$\bar{f}''_{net}(o_1, \dots, o_k) = (f''_{net}(o_1), \dots, f''_{net}(o_k)),$$

and this definition makes a network fingerprint composition $\bar{f}''_{net} \circ f'_{net}$ possible. In the case of non-deterministic $f''_{net} : O_1 \to \mathcal{P}(O_2)$, we change the definition to

$$\bar{f}''_{net} : \mathcal{P}(O_1^k) \to \mathcal{P}(O_2^k),$$

$$\bar{f}''_{net}(A) = \bigcup_{x \in A} f''_{net}(x).$$

A second case where a natural extension to tuples can be applied is if we are given a vertex v containing multiple web objects, i.e., $v = (w_1, w_2, \dots)$ and want to apply a network fingerprint $f_{net} : O \to \mathcal{P}(O_1)$ to the observations of each file separately. To underline the different scenario when natural extension to tuples is applied, in this case we will write *perfile*(f_{net}) instead of $\bar{f}_{net}$.

Adapting the possibilistic security model

When the (composed) network fingerprint f_{net} is applied, in order to compute the ambiguity of an observation, we have to adapt the definition of the possible observations $g : V \to \mathcal{P}(O)$ (see Section 3.2.2). When visiting a vertex v, as a result of the applied network fingerprint, the attacker sees an element of $(f_{net} \circ f_{app})(v)$. Accordingly, we define the possible observations as

$$g = f_{net} \circ f_{app}.$$

To calculate the ambiguity $K_{sec}(\cdot)$ defined in Section 3.2.2, we apply this new definition of g.

Remark

Note that we can interpret the case where no network fingerprint is applied as using the identity function as a network fingerprint. Therefore, we can always talk about *the applied network fingerprint* f_{net}, which may be any (composed) network fingerprint or the identity function.

4.1.2 Probabilistic network fingerprints

The possibilistic network fingerprints introduced in the previous section are non-deterministic functions, and in order to analyze their effect, we need to reason about sets of *possible observations.* In practice, some observations may be possible, but not *likely*, and will be rarely observed by attackers. As discussed in Section 3.2.4, this may lead to unrealistic security guarantees. To deal with this issue, we give a probabilistic definition of network fingerprints, which will be incorporated into the probabilistic security model (see Section 3.2.3).

Definition

Given a possibilistic network fingerprint $f_{poss} : O \to \mathcal{P}(O_1)$, we define its probabilistic counterpart as the function

$$f_{net} : O \to (O_1 \to [0,1]),$$

such that for each $o \in O$, $\sum_{o' \in O_1} f_{net}(o)(o') = 1$, and $f_{net}(o)(o') > 0$ iff $o' \in f_{poss}(o)$. We call such a network fingerprint a *probabilistic network fingerprint.*

For convenience of notation, given a probabilistic network fingerprint f_{net}, we define the random variables F^{in} and F^{out} and use the notation $f_{net} = P[F^{out}|F^{in}]$, where $f_{net}(o) = P[F^{out}|F^{in} = o]$ and $f_{net}(o)(o') = P[F^{out} = o'|F^{in} = o]$. Using this notation, the conditions from above become: for each $o \in O$, $\sum_{o' \in O_1} P[F^{out} = o'|F^{in} = o] = 1$, and $P[F^{out} = o'|F^{in} = o] > 0$ iff $o' \in f_{poss}(o)$. In the case of deterministic network fingerprints, F^{out} is a function of F^{in}.

Transforming a possibilistic to a probabilistic network fingerprint can be done by explicitly defining the probability distribution $P[F^{out}|F^{in}]$. In future definitions of specific network fingerprints, we will give a possibilistic definition, and when the network fingerprint is viewed in a probabilistic scenario, unless otherwise specified, we assume an arbitrary probability distribution $P[F^{out}|F^{in}]$.

Network fingerprint composition

Given network fingerprints $f'_{net} : O_0 \to (O_1 \to [0,1])$ and $f''_{net} : O_1 \to (O_2 \to [0,1])$, we define probabilistic network fingerprint composition as

$$f''_{net} \circ f'_{net} : O_0 \to (O_2 \to [0,1]),$$

$$P[F^{out}_{2\circ 1} = o_2 | F^{out}_{2\circ 1} = o_0] = \sum_{o_1 \in O_1} P[F^{out}_2 = o_2 | F^{out}_2 = o_1] \cdot P[F^{out}_1 = o_1 | F^{out}_1 = o_0].$$

The last definition allows an arbitrary long composition of network fingerprints, where a recursive application of the definition above gives

$$\begin{aligned}&(c_n \circ c_{n-1} \circ \cdots \circ c_1)(o_0)(o_n) = \\ &\sum_{o_n \in O_n} \sum_{o_{n-1} \in O_{n-1}} \cdots \sum_{o_1 \in O_1} P[C^{out}_n = o_n | C^{in}_n = o_{n-1}] \cdot P[C^{out}_{n-1} = o_{n-1} | C^{in}_{n-1} = o_{n-2}] \\ &\qquad \cdots P[C^{out}_1 = o_1 | C^{in}_1 = o_0].\end{aligned}$$

Similarly to the possibilistic case, we define the natural extension to tuples, so that the composition of network fingerprints $f'_{net} : O \to (O_1^k \to [0,1])$ and $f''_{net} : O_1 \to (O_2 \to [0,1])$ is possible, by transforming f''_{net} to $\bar{f}''_{net} : O_1^k \to (O_2^k \to [0,1])$. This means, given $P[F^{out}|F^{in}]$, we want to obtain random variables $\bar{F}^{in}$ defined over O_1^k and $\bar{F}^{out}$ defined over O_2^k such that if $\bar{F}^{in} = (x_1, \ldots, x_k)$ and $\bar{F}^{out} = (y_1, \ldots, y_k)$, and the value of y_i should only depend on x_i. For this to hold, if $\bar{F}^{in}$ is decomposed into independent random variables $\bar{F}_1^{in} = x_1, \ldots, \bar{F}_k^{in} = x_k$, and if $\bar{F}^{out}$ is decomposed into independent random variables $\bar{F}_1^{out} = y_1, \ldots, \bar{F}_k^{out} = y_k$, $\bar{F}_i^{out}$ will be only dependent on $\bar{F}_i^{in}$. Thus, we obtain

$$\begin{array}{rl} & P[\bar{F}^{out} = (y_1, \ldots, y_k)|\bar{F}^{in} = (x_1, \ldots, x_k)] \\ = & P[\bar{F}_1^{out} = y_1, \ldots, \bar{F}_k^{out} = y_k|\bar{F}_1^{in} = x_1, \ldots, \bar{F}_k^{in} = x_k] \\ = & P[\bar{F}_1^{out} = y_1|\bar{F}_1^{in} = x_1] \cdot P[\bar{F}_2^{out} = y_2|\bar{F}_2^{in} = x_2] \ldots P[\bar{F}_k^{out} = y_k|\bar{F}_k^{in} = x_k]. \end{array}$$

Adapting the probabilistic security model

We remind the reader that given a graph $G = (V, E)$ and an application fingerprint function $f_{app} : V \to O$, in the probabilistic security model defined in Section 3.2.3, two random variables were defined: X, representing the input of the system – the user's behavior, and Y, representing the output of the system – the attacker's observations, with $Y = f_{app}(X)$. Applying the (composed) network fingerprint $f_{net} : O \to (O_1 \to [0,1])$, the random variable Y will change accordingly, and will be defined as $Y = (f_{net} \circ f_{app})(X)$. Let $X_1, \ldots, X_\ell$ be the decomposition of X such that if $X = (v_1, \ldots, v_\ell)$, then $X_1 = v_1, \ldots, X_\ell = v_\ell$, and let $Y_1, \ldots, Y_\ell$ be the decomposition of Y such that if $Y = (o_1, \ldots, o_\ell)$, then $Y_1 = o_1, \ldots, Y_\ell = o_\ell$. As we consider stateless network fingerprints, i.e., the observations produced by f_{net} only depend on the current vertex being visited, we obtain $P[Y_i = o_i|X = (v_1, \ldots, v_\ell)] = P[Y_i = o_i|X_i = v_i] = P[F_i^{out} = o_i|F_i^{in} = f_{app}(v_i)]$. Accordingly, we compute $P[Y|X]$ as follows:

$$\begin{array}{rl} & P[Y = (o_1, \ldots, o_\ell)|X = (v_1, \ldots, v_\ell)] \\ = & P[Y_1 = o_1, \ldots, Y_\ell = o_\ell|X_1 = v_1, \ldots, X_\ell = v_\ell] \\ \overset{(*)}{=} & P[Y_1 = o_1|X_1 = v_1, \ldots, X_\ell = v_\ell] \cdots P[Y_\ell = o_\ell|X_1 = v_1, \ldots, X_\ell = v_\ell] \\ \overset{(**)}{=} & P[Y_1 = o_1|X_1 = v_1] \cdots P[Y_\ell = o_\ell|X_\ell = v_\ell] \\ = & P[F^{out} = o_1|F^{in} = f_{app}(v_1)] \cdots P[F^{out} = o_\ell|F^{in} = f_{app}(v_\ell)]. \end{array}$$

$(*)$ follows by the independence of $Y_i|X$; $(**)$ follows by the statelessness of f_{net}.

4.2 Countermeasures as network fingerprints

Countermeasures are modifiers of observations which aim at reducing information leaks in web applications. In our framework, countermeasures are modeled as network fingerprints. In the following, we discuss practical aspects of countermeasures.

To define countermeasures, we will use composed network fingerprints. In order to deploy a network fingerprint as a countermeasure, two conditions must hold: (1) it should be well-defined, and (2) it should be applicable in such a way that users finally receive the service they are expecting. For example, a possible countermeasure is sending the user on each request the HTTP "404 Not Found" error message. This may be a very effective countermeasure as almost no information is leaked about users' browsing behavior, however users may complain about not receiving the expected service, which makes the countermeasure not applicable. To show that countermeasures are applicable, application scenarios for them should be specified: how a countermeasure defined as a mathematical formula can be deployed on a system.

As countermeasures aim at stopping information leaks, the most natural way to deploy a countermeasure is as part of the cryptographic mechanism applied to the data: the countermeasures can be built into the encryption routine, and the decryption routine can include a mechanism to reverse the effect of the countermeasure and return the data to its original state. Similarly, this can be implemented as extensions which run on both the browser and the server. Such an approach would allow a great flexibility in developing countermeasures. Unfortunately, an adoption of countermeasures against web browsing leakage on such a large scale is a far-stretched goal as first the community should be convinced that the effect of such countermeasures will justify the cost of their deployment, followed by a long process of adoption.

In Section 4.3, along with formal definitions of network fingerprints, we discuss their applicability as countermeasures. We consider countermeasures which run only on the user's browser (*client-side* countermeasures), or only on the server (*server-side countermeasures*), but not both (see discussion in previous paragraph). We say that countermeasures have *protocol-independent applicability* if they modify the traffic before it has left its origin, and *protocol-dependent applicability*, if the effect of the countermeasure relies on certain properties of the used protocols. Thus, a countermeasure with protocol-dependent applicability may not work properly if the browsers and/or servers do not support the needed protocol features. An example countermeasure with protocol-dependent applicability is utilizing the HTTP/1.1 Range option, as proposed by Luo et al. [42]. This option allows requesting sub-ranges of objects [26], and by 2011 was supported by around 80% of the deployed web servers [42].

4.3 Definition of practical network fingerprints

In this section, we define a number of practical network fingerprints. We identify several *basic* network fingerprints which we use as building blocks for more complex network fingerprints we call *composed*. We emphasize on the deployment of those

Name	per file	protocol-independent client	protocol-dependent only client
pad	×		×
buck	×		×
split	×		×
dummy		×	
shuffle			×
wshuffle		×	
k-par		×	
k-mer	×		×
inter			×

Table 4.1: Basic network fingerprints; *per file* denotes if the network fingerprint takes as input one file, and thus application to webpages with multiple objects will be performed per file. We denote whether the applicability of the network fingerprint as a client-side countermeasure is protocol-independent, or protocol-dependent.

network fingerprints as countermeasures, and discuss their applicability on the client and on the server-side. We define network fingerprints possibilistically, and if a network fingerprint is to be used in a probabilistic scenario, we assume an arbitrary probability distribution on its output (see Section 4.1.2).

4.3.1 Basic network fingerprints

The basic network fingerprints we consider are listed in Table 4.1. In this section we elaborate on each of the basic network fingerprints, giving a formal definition, and discussing their applicability as countermeasures. As defined in Chapter 3, O denotes the set of observations with $O \subset (\mathbb{N} \times \{\uparrow, \downarrow\})^*$. In the following, some network fingerprints are defined as functions over $\mathbb{N}$; to apply such functions to elements of O, the direction is retained, and if necessary, they can be applied per file through natural extension to tuples, see Section 4.1.

Padding

Padding (which we denote as *pad*) is a widely considered countermeasure in the literature (e.g., see [51, 39, 10]). It changes observations by increasing their size. Let the set $A \subseteq \mathbb{N}$ describe the set from which values are chosen when padding is applied. Then, we define the padding network fingerprint as

$$pad : \mathbb{N} \to \mathcal{P}(\mathbb{N}),$$

$$pad(o) = \{o + p \mid p \in A\}.$$

A protocol-independent application of padding as a countermeasure is possible on the server-side. For this, to each downloaded object, p additional characters should be added, which should not change the way the browser displays the contents. Padding of web-objects can be implemented by including variable-length comments, which are possible in most languages (HTML, JavaScript), and most file formats (JPEG, FLV), or by including special characters, e.g. whitespace characters. Padding of network packets is also possible, for example by adding characters to HTTP headers. A protocol-dependent client-side approach is possible by deploying the HTTP Range option, or by causing TCP retransmissions, in order to request data that has been already downloaded (see [42]). Note that the latter methods will additionally incorporate other network fingerprints, such as *dummy* and *split* (see below).

Bucketing

Bucketing (or *buck*) is a form of deterministic padding. Given a *bucketing vector* $b \in \mathbb{N}^*$, *buck* deterministically maps a file size x to the smallest $b[i] \geq x$:

$$\begin{gathered} \mathit{buck} : \mathbb{N} \to \mathbb{N}, \\ \mathit{buck}(o) = o + p, \\ \text{where } p = \begin{cases} \min\{b[i] - o \mid b[i] \geq o\} & \text{if } \exists i : b[i] \geq o \\ 0 & \text{otherwise} \end{cases} \end{gathered}$$

For the applicability of bucketing as a countermeasure, the same considerations hold as discussed for padding. An example of bucketing – *equal-sized bucketing*, sets the bucket borders so that to each bucket, n observations from O are mapped. In Chapter 7, we perform case studies which investigate the effect of equal-sized bucketing with a varying bucket size n, on the security and the overhead of example web applications.

Dummy

The network fingerprint *dummy* adds redundant files to observations. Formally, we define *dummy* as

$$\mathit{dummy} : O \to O$$

$$\mathit{dummy}(\vec{o}) = \mathit{dummy}(o_1, \ldots, o_n) = \{(o'_1, \ldots, o'_m) \mid (o_1, \ldots, o_n) \sqsubseteq (o'_1, \ldots, o'_m)\}.$$

Protocol-independent application of *dummy* as a countermeasure is possible on the server, e.g. by including objects in an HTML file which will be downloaded but not shown to the user. On the client-side, protocol-independent application is possible by requesting certain existing files from the server; for this, the client should know in advance which files reside on the server. A protocol-dependent approach for implementing *dummy* on the client-side is using HTTP range and TCP retransmission (see [42]).

Split

The network fingerprint *split* causes splitting a file into smaller files. Formally, we define *split* as

$$split : O \to \mathcal{P}(O^+)$$

$$split(o) = \{(o_1, \ldots, o_n) \mid n \in \mathbb{N} \wedge \sum_{i=1}^{n} o_i = o\}$$

An approach for implementing *split* as a server-side countermeasure is possible on the TCP level, by segmenting the file before it is sent on the network. Although this approach uses TCP segmentation, we consider it protocol-independent, as packet segmentation is a core feature of the TCP protocol [6]. A protocol-dependent application on the client-side is possible, e.g. by setting the TCP advertising window, or by utilizing HTTP Range (see [42]).

Shuffle

In a webpage containing more than one object, the base .html file is always requested first, and the remaining web-objects (or *inline web-objects*) are usually downloaded in the order in which they are included into the .html file. In the following, we propose the *shuffle* countermeasure, which changes the order in which files are downloaded:

$$shuffle : O \to \mathcal{P}(O),$$

$$shuffle(o_1, \ldots, o_n) = \Pi(o_1, \ldots, o_n).$$

Figure 4.1 shows an example application of *shuffle* as a server-side countermeasure: the contents of the original .html file are stored as a JavaScript variable in the `source.js` file; in `main.html`, all files are preloaded by including them in an arbitrary order, after which the original HTML code is printed on screen. Note that this code will not comply with the standard HTML document type definitions (or DTDs), however our experiments show that modern browsers display sample webpages correctly. This countermeasure requires a JavaScript-enabled browser, and it will slow down the display of the contents until all files have been requested.

Weak shuffle

Using *weak shuffle* (*wshuffle*), first the base .html file is downloaded, and then the inline files are downloaded in an arbitrary order. Formally, we define this network fingerprint as

$$wshuffle : O \to \mathcal{P}(O),$$

```
<img src="img3.png" height="0" width="0" style="display:none" />
<img src="img1.png" height="0" width="0" style="display:none" />
<img src="img4.png" height="0" width="0" style="display:none" />
<script type="text/javascript" src="source.js"></script>
<img src="img2.png" height="0" width="0" style="display:none" />
<script type="text/javascript"> <!--
document.write(source);
//--></script>
<noscript>JavaScript should be enabled to view this webpage!
   </noscript>
```

main.html

```
var source =
'<head>\
<title>My image collection</title>\
</head>\
<body>\
<h1>My image collection</h1>\
<img src="img1.png" /><br />\
<img src="img2.png" /><br />\
<img src="img3.png" /><br />\
<img src="img4.png" /><br />\
</body>\
</html>';
```

source.js

Figure 4.1: A possible implementation of the *shuffle* countermeasure. The source of the original .html file is included in `source.js`, and in `main.html` all files can be preloaded in an arbitrary order.

$$\mathit{wshuffle}(o_1, \ldots, o_n) = \{(o'_1, \ldots, o'_n) \in \Pi(o_1, \ldots, o_n) \mid o'_1 = o_1\}.$$

Note that using *weak shuffle*, we can define $\mathit{shuffle}'$ as $\mathit{shuffle}' = \mathit{shuffle} \circ \mathit{wshuffle}$, and we obtain $\mathit{shuffle} = \mathit{shuffle}' = \mathit{shuffle} \circ \mathit{wshuffle}$.

Unlike *shuffle*, when *weak shuffle* is used, an attacker can be sure that the first downloaded file is the base file. It has a protocol-independent applicability on the client-side: after fetching the base .html file, instead of downloading inline files in the order of their occurrence in the base file, requests for them can be sent in an arbitrary order. On the server-side, the inline objects can be preloaded by including them as hidden files in the beginning of the body of the .html file; this does not require JavaScript, and the resulting code will be DTD-compliant. Using this countermeasure, the display of inline files can be slowed down, as those files will be requested after the whole .html file is parsed.

```html
<html>
<head>
<title>The original title</title>
</head>
<body style="margin:0px;">
<iframe src="redundant1.html" style="display:none"></iframe>
<iframe src="original.html" width="100%" height="100%" style="
   border:0px"></iframe>
<iframe src="redundant2.html" style="display:none"></iframe>
<iframe src="redundant3.html" style="display:none"></iframe>
</body>
</html>
```

main.html

Figure 4.2: A possible server-side implementation of the *k-parallelism* countermeasure. The original and the redundant files can be downloaded in anarbitrary order by including themin iframes; the original file is displayed to the user, while the remaining files are hidden.

k-parallelism

The *k-parallelism* (*k-par*) network fingerprint requests $k-1$ redundant requests every time a vertex v is visited; thus, an observer does not know which observation corresponds to v, and which corresponds to redundant vertices. Formally, k-par is defined as follows:

$$k\text{-}par : O \to \mathcal{P}(O^k),$$

$$k\text{-}par(o) = \{(o_1, \dots, o_k) | o \in (o_1, \dots, o_k)\},$$

where O^k denotes that an observation corresponds to k (simultaneously) visited vertices.

A protocol-independent application of k-par as a countermeasure is possible both on the client, and on the server-side. On the client-side, the implementation is straightforward: every time a user requests a vertex v, in parallel, HTTPS requests for additional vertices $v^{(1)}, \dots, v^{(k-1)}$ can be issued. A possible implementation of k-par on the server-side is depicted on Figure 4.2: the original webpage and the redundant webpages are all included in iframes; the original iframe is displayed to the user, while the redundant ones are hidden from the user.

An attacker observing the output of k-par may be able to directly infer the k streams of observations corresponding to the k requested vertices, e.g. by inspecting TCP headers of TLS-encrypted data. This can be addressed by applying the *interleave* network fingerprint, see below.

k-merging

Next we define the k-merging (k-mer) network fingerprint. It is defined on web-objects (and thus should be applied per file), and has a similar functionality as k-parallelism: $k-1$ redundant vertices are picked, however only one file is transferred through the network, and its size is the sum over the sizes of the k files.

Let the function $k\text{-}sum : \mathbb{N}^k \to \mathbb{N}$ be the sum of k elements of $\mathbb{N}$. We define the k-merging countermeasure as

$$k\text{-}mer : \mathbb{N} \to \mathcal{P}(\mathbb{N}),$$

$$k\text{-}mer = k\text{-}sum \circ k\text{-}par.$$

Essentially, k-merging is a form of padding, and for its application as a countermeasure, the same considerations hold as discussed for padding (see above).

Interleave

The last basic fingerprint we present is *interleave*. It takes streams corresponding to k vertices (e.g., produced by k-parallelism), and converts those streams into one big stream of observations, preserving the order in which files are downloaded. Formally, *interleave* is defined as follows:

$$O^k \to \mathcal{P}(O)$$

$$\begin{aligned} inter(\vec{o_1}, \dots, \vec{o_k}) = \{(o'_1, \dots, o'_m) \in \Pi(\bigsqcup_{j=1}^{k} \vec{o_j}) \mid \\ \forall o'_i \neq o'_1, o'_i = \pi_n(\vec{o_j})\ \exists i_2 < i : o'_{i_2} = \pi_{n-1}(\vec{o_j})\} \end{aligned}$$

To implement *inteleave* as a countermeasure, it should become impossible to tell which files belong to which stream. Protocol-independent approaches for this are not known to us; a protocol-dependent approach to achieve this is HTTP pipelining (see [42]). Interleaving of streams can be also achieved by tunneling traffic through an encrypted proxy (see Section 4.4).

4.3.2 Composed network fingerprints

The basic fingerprints defined in the previous section can be used as building blocks for more complex network fingerprints, e.g. through network fingerprint composition (see Section 4.1), or through case distinction. In this section we will discuss several example composed network fingerprints built from basic ones. The composed network fingerprints we consider are summarized in Table 4.2.

	protocol-independent	reveals
Definition	client	# files
Padding-based		
$\mathit{pad} \circ \mathit{buck}$		×
$\mathit{buck} \circ \mathit{pad}$		×
$\mathit{perfile}(\mathit{pad} \circ \mathit{buck}) =: \text{PBF}$		×
$\mathit{dummy} \circ \mathit{perfile}(\mathit{pad} \circ \mathit{buck}) =: \text{DPBF}$		
Parallelism-based		
$\mathit{inter} \circ \overline{k\text{-}\mathit{par}}$	×	
$\mathit{inter} \circ \overline{\mathit{perfile}(\mathit{split})} \circ k\text{-}\mathit{par}$	×	
$\overline{\mathit{wshuffle}} \circ k\text{-}\mathit{par}$	×	
$\mathit{inter} \circ \overline{\mathit{wshuffle}} \circ k\text{-}\mathit{par}$		
$k\text{-}\mathit{mer}\text{-}\mathit{1st} \circ \mathit{inter}' \circ \overline{\mathit{wshuffle}} \circ k\text{-}\mathit{par} =: \text{KIWK}$		
Mixed		
$\text{PBFD} \circ \text{KIWK}$		

Table 4.2: Example composed network fingerprints; *protocol-independence client* denotes if the network fingerprint has protocol-independent applicability as a client-side countermeasure, and *reveals # files* denotes that the used network fingerprint does not change the number of downloaded files

Example 1: Padding-based network fingerprints

If we are given an observation of a vertex v consisting of one web-object, i.e, $v = w$, a (randomized) padding *pad* and a bucketing *buck* can be combined by either computing $\mathit{pad} \circ \mathit{buck}$, or $\mathit{buck} \circ \mathit{pad}$. Application of any form of padding or a composition thereof is not directly possible if the target webpage consists of more than one web-object, i.e., $\vec{v} = (w_1, \dots, w_n)$, but should be applied per file. For example, in the case of $\mathit{pad} \circ \mathit{buck}$, we define the countermeasure $\mathit{perfile}(\mathit{pad} \circ \mathit{buck}) := \text{PBF}$, with

$$\mathit{perfile}(\mathit{pad} \circ \mathit{buck})(o_1, \dots, o_n) = ((\mathit{pad} \circ \mathit{buck})(o_1), \dots, (\mathit{pad} \circ \mathit{buck})(o_n)).$$

The application of any form of (composed) padding as a countermeasure may be useless if all vertices $\vec{v}$ in a web-graph consist of a unique number of web-objects. To avoid this, we can additionally apply *dummy*. Thus, we obtain the new countermeasures $\mathit{dummy} \circ \mathit{perfile}(\mathit{pad} \circ \mathit{buck}) := \text{DPBF}$, and $\mathit{perfile}(\mathit{pad} \circ \mathit{buck}) \circ \mathit{dummy} := \text{PBFD}$, which no longer deterministically leak the number of files in a webpage.

Example 2: Parallelism-based network fingerprints

A second example is building a network fingerprint based on $k\text{-}\mathit{par}$. Possible combinations include $\mathit{inter} \circ k\text{-}\mathit{par}$, $\mathit{inter} \circ \overline{\mathit{perfile}(\mathit{split})} \circ k\text{-}\mathit{par}$, $\overline{\mathit{wshuffle}} \circ k\text{-}\mathit{par}$,

$\mathit{inter} \circ \overline{\mathit{wshuffle}} \circ \mathit{k\text{-}par}$, where the overbar denotes a natural extension to tuples, see Section 4.1. We explain how $\mathit{inter} \circ \overline{\mathit{wshuffle}} \circ \mathit{k\text{-}par}$ works in the web-navigation scenario: $k-1$ redundant webpages are chosen, after which first one of the k base (.html) files are downloaded, followed by a sequence of all remaining files, in such an order that every downloaded inline file is preceded by the base file of the page where it was included. A special case of this network fingerprint first loads all k base files, and then loads all inline files in a random order. Formally, this network fingerprint is defined as $\mathit{inter}' \circ \overline{\mathit{wshuffle}} \circ \mathit{k\text{-}par}$, where

$$\begin{aligned}\mathit{inter}'(\vec{o}_1, \ldots, \vec{o}_k) =& \{(o'_1, \ldots, o'_m) \in \mathit{inter}(\vec{o}_1, \ldots, \vec{o}_k) \mid \\ & o'_1, \ldots, o'_k \in \Pi(\pi_1(\vec{o}_1), \ldots, \pi_k(\vec{o}_k))\}.\end{aligned}$$

Additionally, the base files can be merged through a variant of $\mathit{k\text{-}mer}$:

$$\mathit{k\text{-}mer\text{-}1st} : \mathbb{N}^m \to \mathbb{N}^{m-k+1},$$

$$\mathit{k\text{-}mer\text{-}1st}(o_1, \ldots, o_k, o_{k+1}, \ldots, o_m) = \mathit{k\text{-}sum}(o_1, \ldots, o_k) \sqcup (o_{k+1}, \ldots, o_m).$$

Then, we obtain the network fingerprint $\mathit{k\text{-}mer\text{-}1st} \circ \mathit{inter}' \circ \overline{\mathit{wshuffle}} \circ \mathit{k\text{-}par}$:=KIWK.

Example 3: Mixed network fingerprints

The example padding- and redundancy-based network fingerprints can also be combined. For example, $(k-1)$ redundant webpages can be chosen, their observations can be merged into one big observation whose first file is the merging of the k base files, then on each of the files in the new observation we can apply padding and we can add dummy packets, obtaining the new composed network fingerprint PBFD $\circ$ KIWK.

4.4 Network protocols as network fingerprints

In Chapter 3, we assumed that an attacker observes the sizes of transmitted web-objects (the application fingerprint f_{app}). In practice, as data passes through the protocol stack, the network protocols make certain changes to the transferred objects: when a message is transmitted, it is split into packets, and multiple headers are added. In the following, we show how the effect of network protocols to observations can be modeled as composed network fingerprints, built from the basic network fingerprints defined in Section 4.3.1.

Packet headers When passing through different network layers, each layer adds its own headers, often of a fixed size. A header can be interpreted as a padding:

$$\mathit{header} = \mathit{pad},$$

and if the header is fixed-sized, then *pad* will be deterministic.

Packet segmentation Packet segmentation is the process of dividing a packet into smaller chunks, where each of those chunks obtains a new header. For example, segmentation is used by the TCP protocol for congestion control. Formally, packet segmentation corresponds to

$$\mathit{segment} = \mathit{header} \circ \mathit{split},$$

where *split* is usually deterministic.

Tunneling Tunneling encapsulates traffic from one protocol into another protocol. It may add a new layer of encryption (e.g. using an SSH tunnel), and in this case none of the original packet headers will transmitted in the clear (see Section 2.1.2). As a consequence, an attacker may lose the ability to distinguish which packets correspond to which web objects. Additionally, certain background traffic which is easily filtered out in the presence of unencrypted headers, may appear as part of the downloaded contents. We model encrypted tunneling as an interleaving of multiple streams of segmented packets

$$\mathit{tunnel} = \mathit{inter} \circ \overline{\mathit{segment}} \circ k\text{-}\mathit{par}.$$

5 Formal analysis

The models presented in Chapter 3 and Chapter 4 provide tools for formal reasoning about the problem of information leakage in web applications. In the current chapter, we use those tools to formally analyze different aspects of the problem.

5.1 The effect of network fingerprint composition

We first address the question what the effect of applying a network fingerprint on the security of a web application is, and whether this effect can be strengthened by fingerprint composition. Both possibilistically and probabilistically, we show that a fingerprint f_{net} can be strengthened by composing it with a fingerprint f'_{net}, however one should be careful in which order the network fingerprints are composed.

5.1.1 Possibilistic case

The following theorem shows that in the possibilistic security model (see Section 3.2.2), a composed network fingerprint $\hat{f}'_{net} \circ f_{net}$ provides better security guarantees than the network fingerprint f_{net}, where $\hat{f}'_{net}$ is the natural extension of f'_{net} to powersets (see Section 4.1).

Theorem 1. *Let $(f_{app}(v_1), \ldots, f_{app}(v_\ell)) = (o_1, \ldots, o_\ell) \in O^\ell$ be an observation path, let sec be a secret specification, let f_{net} be a network fingerprint $O \to \mathcal{P}(O')$, and let f'_{net} be a network fingerprint $O' \to \mathcal{P}(O'')$. Given observation paths $(o'_1, \ldots, o'_\ell) \in (f_{net}(o_1) \times \cdots \times f_{net}(o_\ell))$, and $(o''_1, \ldots, o''_\ell) \in (f'_{net}(o'_1) \times \cdots \times f'_{net}(o'_\ell))$, then,*

$$K_{sec}(G, (o'_1, \ldots, o'_\ell), f_{net} \circ f_{app}) \subseteq K_{sec}(G, (o''_1, \ldots, o''_\ell), \hat{f}'_{net} \circ f_{net} \circ f_{app}).$$

Proof. Let $sec(v_1, \ldots, v_\ell) \in K_{sec}(G, (o'_1, \ldots, o'_\ell), f_{net} \circ f_{app})$. According to the definition of $K_{sec}(\cdot)$, this implies (1) $(v_1, \ldots, v_\ell) \in paths(G, \ell)$ and (2) $\forall i \in [\ell] : o'_i \in (f_{net} \circ f_{app})(v_i)$. To show that $sec(v_1, \ldots, v_\ell) \in K_{sec}(G, (o''_1, \ldots, o''_\ell), \hat{f}'_{net} \circ f_{net} \circ f_{app})$, it is left to show that (3) $\forall i \in [\ell] : o''_i \in (\hat{f}'_{net} \circ f_{net} \circ f_{app})(v_i)$, which by definition is equivalent to $o''_i \in \bigcup_{o \in f_{net} \circ f_{app}(v_i)} f'_{net}(o)$. This statement holds, because $o''_i \in f'_{net}(o'_i)$ and $o'_i \in f_{net}(o_i) = (f_{net} \circ f_{app})(v_i)$. □

Theorem 1 implies that the application of a network fingerprint cannot reveal more information about the secret values than what is known without the application of this network fingerprint. In particular this means that a network fingerprint cannot invert another network fingerprint's effect. However, as the following proposition shows, one should be careful in which order network fingerprints are composed: if one wants to strengthen a network fingerprint f_{net} by composing it with another network fingerprint, f_{net} should be put on the right-hand side of the composition.

Proposition 1. *In the setting of Theorem 1, given* $(o_1''', \ldots, o_\ell''') \in (f'_{net}(o_1) \times \cdots \times f'_{net}(o_\ell))$*, there are cases where*

$$K_{sec}(G, (o_1''', \ldots, o_\ell'''), f'_{net} \circ f_{app}) \supset K_{sec}(G, (o_1'', \ldots, o_\ell''), \hat{f}''_{net} \circ f_{net} \circ f_{app}).$$

Proof. Let $f'_{net} : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ be defined as $f'_{net}(x) = 2$ if $x \leq 2$ and $f'_{net}(x) = \{x\}$ otherwise, and let $f_{net} : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ be defined as $f_{net}(x) = x + 1$. Let $v = 2$. Then, $o''' = f'_{net}(2) = 2$, and $o'' = (f'_{net} \circ f_{net})(2) = 3$. $K_{sec}(G, (o_1''', \ldots, o_\ell'''), f'_{net} \circ f_{app}) = f''^{-1}_{net}(o''') = f''^{-1}_{net}(2) = \{1, 2\}$, and $K_{sec}(G, (o_1''', \ldots, o_\ell'''), f'_{net} \circ f_{app}) = (f'_{net} \circ f_{net})^{-1}(o'') = (f'_{net} \circ f_{net})^{-1}(3) = \{2\}$. □

Thus, by Theorem 1 and Proposition 1, in order to guarantee a strengthening of the network fingerprint f_{net} by means of composition, we should compose it with other network fingerprints f'_{net} by first applying f_{net} and then f'_{net}, and not the other way around, i.e., the resulting composed network fingerprint should be $\hat{f}'_{net} \circ f_{net}$, and not $\hat{f}_{net} \circ f'_{net}$. This also means that in some cases, while $\hat{f}'_{net} \circ f_{net}$ provides stronger security guarantees than f_{net}, using only f'_{net} would have been the better choice.

5.1.2 Probabilistic case

We now revisit the questions about the role of network fingerprint composition from the previous section using probabilistic security definitions (see Section 3.2.3). First, we show a probabilistic counterpart of Theorem 1.

Theorem 2. *Let sec be a secret specification, let* f'_{net} *be a network fingerprint* $O \to (O' \to [0, 1])$*, let* f''_{net} *be a network fingerprint* $O' \to (O'' \to [0, 1])$*, and let* Y'*,* Y''*, and* $Y^{(''\circ')}$ *be the output random variables when* f'_{net}*,* f''_{net}*, and* $f''_{net} \circ f'_{net}$ *are applied, respectively. Then,* $I_\infty(sec(X); Y') \geq I_\infty(sec(X); Y^{(''\circ')})$*.*

For the proof of Theorem 2, we introduce the *data processing inequality.* Intuitively, it states that processing of data cannot increase the amount of information contained in these data. The data processing inequality is a well-known result for Shannon-entropy [17]. In the following, we show that it also holds in the case of min-entropy[1].

[1] For an alternative proof, refer to the concurrent work by Espinoza and Smith [22]

Lemma 3 (Data Processing Inequality). *If the random variables X, Y, Z form a Markov chain $X \to Y \to Z$, then $I_\infty(X;Y) \geq I_\infty(X;Z)$.*

Proof. For this proof, we use Lemma 1 and Lemma 2 shown in Section 2.2.1. By Lemma 1, we have (1) $I_\infty(X;Z) + I_\infty(X;Y|Z) = I_\infty(X;Y) + I_\infty(X;Z|Y)$. By Lemma 2, $X \to Y \to Z$ is equivalent to $Z \to Y \to X$, and thus, for all x, y, z, we have $P[X = x|Y = y, Z = z] = P[X = x|Y = y]$, from which we obtain $H_\infty(X|Y,Z) = H_\infty(X|Y)$, and therefore (2) $I_\infty(X;Z|Y) = H_\infty(X|Y) - H_\infty(X|Y,Z) = 0$. Thus, we obtain

$$\begin{aligned} I_\infty(X;Z) &\leq I_\infty(X;Z) + I_\infty(X;Y|Z) \\ &\overset{(1)}{=} I_\infty(X;Y) + I_\infty(X;Z|Y) \\ &\overset{(2)}{=} I_\infty(X;Y) \end{aligned}$$

□

Proof of Theorem 2. By construction, the output of a network fingerprint only depends on its immediate inputs, and thus $P[Y^{(''\circ')} = \vec{o}_2|sec(X) = a, Y' = \vec{o}_1] = P[Y^{(''\circ')} = \vec{o}_2|Y' = \vec{o}_1]$. Thus, $sec(X), Y', Y^{(''\circ')}$ form a Markov chain $sec(X) \to Y' \to Y^{(''\circ')}$. According to the data processing inequality (Lemma 3), we obtain $I_\infty(sec(X);Y') \geq I_\infty(sec(X);Y^{(''\circ')})$. □

Theorem 2 shows that, as in the possibilistic case (Theorem 1), composition of network fingerprints increases the security of a web application. Next we give a probabilistic counterpart of Proposition 1, showing that using the network fingerprint $f''_{net} \circ f'_{net}$ may provide worse probabilistic security guarantees than only using f''_{net}, and thus designers of network fingerprints should be careful in which order fingerprints are composed.

Proposition 2. *In the setting of Theorem 2, there are cases where $I_\infty(sec(X);Y'') < I_\infty(sec(X);Y^{(''\circ')})$.*

Proof. The proof is an extension of the counterexample given in the proof of Proposition 1. We restrict the domain of X to be $\{1, 2\}$ and set $P[X = 1] = P[X = 2] = 0.5$. Analogously to the possibilistic version of this proof, f'_{net} is defined as $P[Y' = y|X = x] = 1$ if $y = x + 1$, and $P[Y' = y|X = x] = 0$ otherwise. In the definition of f''_{net}, we set $P[Y'' = y|X = x] = 1$ if $y = 2$, and $P[Y'' = y|X = x] = 0$ otherwise.

Applying the information theoretic definitions, in order to show that $I_\infty(sec(X);Y'') < I_\infty(sec(X);Y^{(''\circ')})$ holds, it is sufficient to show

$$\sum_y P[Y'' = y] \max_x P[X = x|Y'' = y] < \sum_y P[Y^{(''\circ')} = y] \max_x P[X = x|Y^{(''\circ')} = y].$$

Applying the Bayes law, this is equivalent to

$$\sum_{y} \max_{x} P[X = x]P[Y'' = y|X = x] < \sum_{y'} \max_{x} P[X = x]P[Y^{(''\circ')} = y'|X = x].$$

This inequality holds, as the left-hand side is equal to 1/2, and the right-hand side is equal to 1. □

5.1.3 A landscape of network fingerprints

In both Theorem 1 and Theorem 2 we show that a network fingerprint composition $f'_{net} \circ f_{net}$ can be seen as at least as strong as the network fingerprint f_{net}. We denote this notion of a "stronger network fingerprint" with the symbol $\lhd$, and given network fingerprints f_{net} and f'_{net}, we write $f_{net} \lhd f'_{net}$ iff $f'_{net} = f'_{net} \circ f_{net}$. fingerprints, as defined in Section 4.3. It shows the relationship of the defined network fingerprints according to their strength. To show *wshuffle* $\lhd$ *shuffle*, we use *shuffle* = *shuffle* $\circ$ *wshuffle* (see Section 4.3.1). The remaining relations are a direct application of Theorem 1 and Theorem 2. This figure shows that both countermeasures (see Section 4.2), and network protocols (see Section 4.4), improve the security in web applications.

5.2 Deterministic network fingerprints

In this section, we investigate the probabilistic security guarantees implied by deterministic network fingerprints (e.g., many countermeasures proposed in the literature are deterministic, see [51, 39]). A deterministic network fingerprint f_{net}, together with an application fingerprint f_{app}, form a function $f_{net} \circ f_{app} : \mathcal{X} \to \mathcal{Y}$, which induces a partition of $\mathcal{X}$ into blocks $\{B(y)|y \in \mathcal{Y}\}$, with $B(y) = (f_{net} \circ f_{app})^{-1}(y) = \{x \in \mathcal{X}|f_{net} \circ f_{app}(x) = y\}$. This allows us to leverage and extend existing methods for computing probabilistic security measures to give more direct approaches for computing the remaining uncertainty.

As defined in Section 2.2.1, the remaining uncertainty is computed as

$$H_\infty(X|Y) = -\log \sum_{y \in \mathcal{Y}} P[Y = y] \max_{x \in \mathcal{X}} P[X = x|Y = y].$$

In the following theorem, we show an alternative approach for calculating remaining uncertainty for deterministic network fingerprints.

Theorem 3. *Let f_{net} be a deterministic network fingerprint. Then, we obtain*

$$H_\infty(X|Y) = -\log \sum_{y \in \mathcal{Y}} \max_{x \in B(y)} P[X = x]$$

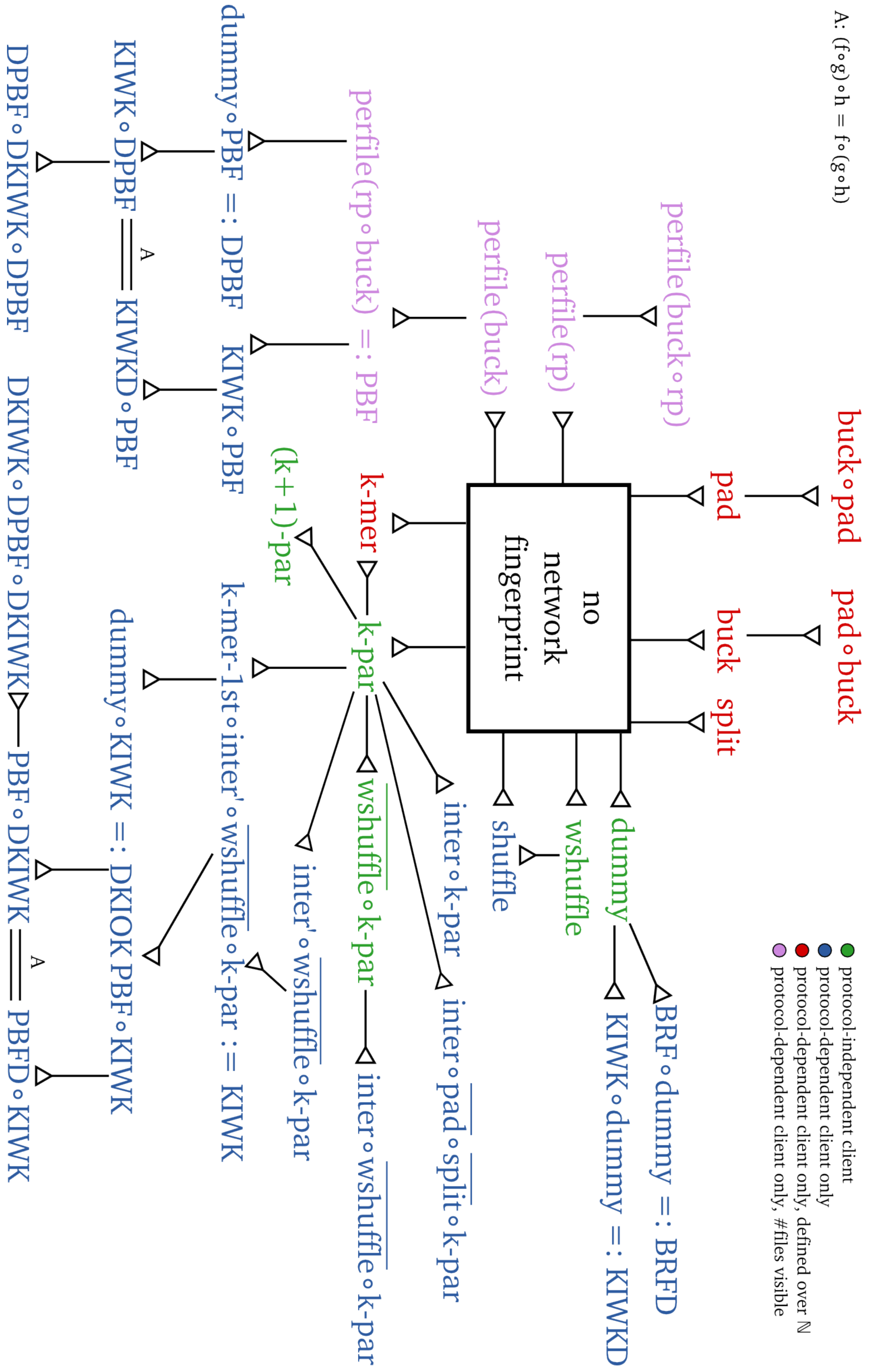


Figure 5.1: A landscape of network fingerprints. The defined network fingerprints are in an ◁-relation to each other, which denotes a “stronger” network fingerprint in the sense of Theorem 1 and Theorem 2. Different colors (see legend) denote the applicability of network fingerprints as countermeasures (see discussion in Section 4.2).

Proof.

$$\begin{aligned}
H_\infty(X|Y) &= -\log \sum_{y\in\mathcal{Y}} P[Y=y] \max_{x\in\mathcal{X}} P[X=x|Y=y] \\
&= -\log \sum_{y\in\mathcal{Y}} \max_{x\in\mathcal{X}} P[Y=y]P[X=x|Y=y] \\
&\overset{(1)}{=} -\log \sum_{y\in\mathcal{Y}} \max_{x\in\mathcal{X}} P[X=x]P[Y=y|X=x] \\
&\overset{(2)}{=} -\log \sum_{y\in\mathcal{Y}} \max_{x\in B(y)} P[X=x].
\end{aligned}$$

(1) follows from the Bayes' law. We obtain (2) as the determinism of f_{net} implies that $P[Y=y|X=x]=1$ for $x\in B(y)$, and $P[Y=y|X=x]=0$ otherwise. □

If we are given the probabilities of the least and most likely possible values of X, the following proposition allows for simpler calculation of the upper and lower bounds of $H_\infty(X|Y)$.

Proposition 3. *Let* $x_{\max} := \arg\max_{x\in\mathcal{X}} P[X=x]$ *and* $x_{\min} := \arg\min_{x\in\mathcal{X}} P[X=x]$. *Then we obtain the following bounds:*

(i) $H_\infty(X|Y) \geq -\log|\mathcal{Y}|P[X=x_{\max}]$

(ii) $H_\infty(X|Y) \leq -\log(P[X=x_{\max}] + (|\mathcal{Y}|-1)P[X=x_{\min}])$

Proof.

$$\begin{aligned}
H_\infty(X|Y) &= -\log \sum_{y\in\mathcal{Y}} \max_{x\in B(y)} P[X=x] \\
&\geq -\log \sum_{i=1}^{|\mathcal{Y}|} P[X=x_i] \\
&\geq -\log |\mathcal{Y}| \max_{x\in\mathcal{X}} P[X=x] \\
H_\infty(X|Y) &= -\log \sum_{y\in\mathcal{Y}} \max_{x\in B(y)} P[X=x] \\
&\leq -\log(\max_{x\in\mathcal{X}} P[X=x] + \sum_{i=1}^{|\mathcal{Y}|-1} P[X=x_{n-i+1}]) \\
&\leq -\log(\max_{x\in\mathcal{X}} P[X=x] + (|\mathcal{Y}|-1)\min_{x\in\mathcal{X}} P[X=x])
\end{aligned}$$

□

If we additionally assume that X is uniformly distributed, further simplification of the calculation of security guarantees is possible. This is a known result from Smith [48], and in the following give an alternative proof using Proposition 3.

Corollary 1. *Assuming a uniform distribution on* X, *for a deterministic network fingerprint* f_{net} *we obtain:*

(i) $H_\infty(X|Y) = -\log \frac{|\mathcal{Y}|}{|\mathcal{X}|}$,

(ii) $I_\infty(X;Y) = \log|\mathcal{X}|$.

Proof. As X is distributed uniformly, we obtain that $P[X=x] = 1/|\mathcal{X}|$ for all x. Applying this to the results in Proposition 3, we obtain

$$\begin{aligned} H_\infty(X|Y) &\geq -\log|\mathcal{Y}|\max_x P[X=x] \\ &= -\log\tfrac{|\mathcal{Y}|}{|\mathcal{X}|} \end{aligned}$$

$$\begin{aligned} H_\infty(X|Y) &\leq -\log(\max_x P[X=x] + (|\mathcal{Y}|-1)\min_x P[X=x]) \\ &= -\log(\tfrac{1}{P[X=x]} + (|\mathcal{Y}|-1)\tfrac{1}{P[X=x]}) \\ &= -\log\tfrac{|\mathcal{Y}|}{|\mathcal{X}|} \end{aligned}$$

For the second statement, as X is uniformly distributed, we obtain

$$H_\infty(X) = -\log(\max_x P[X=x]) = -\log 1/|\mathcal{X}| = \log|\mathcal{X}|.$$

Thus,

$$\begin{aligned} I_\infty(X;Y) &= H_\infty(X) - H_\infty(X|Y) \\ &= \log|\mathcal{X}| - \log|\mathcal{X}| + \log|\mathcal{Y}| \\ &= \log|\mathcal{Y}| \end{aligned}$$

□

Guidelines for designing deterministic countermeasures

A general guideline for designing deterministic countermeasure is aiming at a lower number of observations $|\mathcal{Y}|$. Clearly, if $|\mathcal{Y}| = 1$, the induced security guarantees will be maximal, and if $|\mathcal{Y}| = |\mathcal{X}|$, the induced security guarantees will be minimal. If X is uniformly distributed, a smaller value of $|\mathcal{Y}|$ leads to better security (see Corrolary 1). While further formal investigation of the connection between $|\mathcal{Y}|$ and the implied security guarantees is left to future work, the experiments we performed (see Chapter 7) indicate that in a more general case, a lower $|\mathcal{Y}|$ also leads to better security.

A second guideline follows from the corollary below. It shows that, if a new countermeasure is constructed which merges the resulting blocks, then the new countermeasure gives better security guarantees.

Corollary 2. *Let f'_{net} and f''_{net} be deterministic network fingerprints, and Y' and Y'' be the corresponding output random variables. Let f'_{net} induce partition P_1, and let f''_{net} induce the partition P_2, which is obtained from P_1 by merging two blocks in P_1. Then,*

$$H_\infty(X|Y') \leq H_\infty(X|Y'').$$

Proof. Let $B_1, \ldots, B_n$ be the blocks in P_1 which are merged in P_2, i.e., $P_2 = (P_1 \setminus \{B_1, \ldots, B_n\}) \cup \{B_1 \cup \cdots \cup B_n\}$. Let g be a function such that $g(x) = a$ if

$x \in B_i$ for $i \in [\ell]$, and $g(x) = x$ otherwise. Then, $f''_{net} = g \circ f'_{net}$, and by Theorem 2, we obtain $H(X|Y') \leq H(X|Y'')$.

□

5.3 Outdegree and k-parallelism

In this section, we investigate the connection between the number of outgoing hyperlinks in webpages and the possibilistic security guarantees provided by the countermeasure k-parallelism (or k-par), defined in Section 4.3. We show a lower bound for the possibilistic security guarantees provided by k-par depending on a structural property of a web-graph – its outdegree. This allows flexible construction of countermeasures by selecting a value for the parameter k, which delivers the desired security guarantees.

Let $G = (V, E)$ be a directed graph. We define the *outdegree* of a vertex v (denoted $\delta(v)$) to be the number of outgoing edges from v, and the outdegree of G (denoted $\delta(G)$) to be the minimal outdegree of any vertex in this graph. If G is a web-graph, $\delta(G)$ indicates the minimal number of outgoing links from one page to other pages in the web-graph. When applying k-par, at each vertex on the path $(v_1, \ldots, v_\ell)$, $k-1$ additional vertices are picked and transmitted over the network. We assume that for each $i \in [\ell]$, the $k-1$ additional vertices are distinct and are not equal to v_i.

The following lemma establishes a connection between the length of a path ℓ in the graph G, the outdegree of the graph $\delta(G)$, and the number of parallel streams k.

Lemma 4. *Let $G = (V, E)$ be a connected directed graph, and let $k, \ell \in \mathbb{N}$. Let $(v_1, \ldots, v_\ell)$ be a path in G of length ℓ, and let $A_1, \ldots, A_\ell$ be subsets of V, such that for each $i \in [\ell]$, $|A_i| = k$ and $v_i \in A_i$, as depicted on Figure 5.2. If $k \geq n - \delta(G) + 1$, then the number of possible paths $(v_1, \ldots, v_\ell)$ in G such that $v_i \in A_i$, is $\geq k \cdot (k - (n - \delta(G)))^{\ell-1}$. If the outdegree of all vertices in G is exactly $\delta(G)$, then an equality holds.*

Proof. Assume that $k \geq n - \delta(G) + 1$. This proof goes by induction over the length of the path ℓ.

Let $\ell = 1$. Then, the possible paths of length 1 consist of any of the elements in A_1, therefore $|A_1| = k = k \cdot (k - (n - \delta(G)))^0$.

Assume that for an $\ell \in \mathbb{N}$, the number of possible paths is $\geq k \cdot (k - (n - \delta(G)))^{\ell-1}$. If we increase the path length to $\ell + 1$, we obtain an additional vertex $v_{\ell+1}$, and an additional set $A_{\ell+1}$ with $|A_{\ell+1}| = k$ and $v_{\ell+1} \in A_{\ell+1}$. Each vertex $u \in A_\ell$ has an outdegree of at least $\delta(G)$, and therefore there are at most $n - \delta(G)$ vertices u_1 such that $(u, u_1) \notin E$. As $|A_{\ell+1}| = k \geq n - \delta(G)$, and there are at least $k - (n - \delta(G))$ elements $w_1, \ldots, w_{k-(n-\delta(G))} \in A_{\ell+1}$, such that $(u, w) \in E$. As a result, each path of length ℓ ending with u can be extended by adding one of the

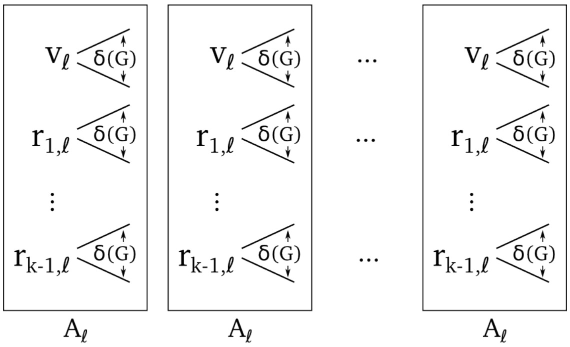


Figure 5.2: Sketch of the setting of Lemma 4 and Theorem 4. $(v_1, \ldots, v_\ell)$ is a path in a graph G, and each set A_i contains the vertex v_i and $k-1$ other vertices $r_{1,i}, \ldots, r_{k-1,i}$. Each of those vertices has an outdegree of at least $\delta(G)$.

edges $(u, w_1), \ldots, (u, w_{k-(n-\delta(G))})$. Thus, the number of paths of length $\ell + 1$ will be at least $(k - (n - \delta(G)))^{\ell-1}$ times the number of paths of length ℓ, and therefore it will be $\geq k \cdot (k - (n - \delta(G)))^{\ell}$. □

Theorem 4. *Let $G = (V, E)$ be a web-graph, let $(v_1, \ldots, v_\ell)$ be a navigation path, let the countermeasure $(k-1)$-parallelism be applied, and let the secret specification sec be the identity function. Let for each $i \in [\ell]$, $o_i \in k\text{-}par \circ f_{app}(v_i)$. If $k \geq n - \delta(G) + 1$, then*

$$|K_{sec}(G, \ell, (o_1, \ldots, o_\ell), k\text{-}par \circ f_{app})| \geq k \cdot (k - (n - \delta(G)))^{\ell}.$$

Proof. Each $o_i \in k\text{-}par \circ f_{app}(v_i)$ contains the observations to k vertices $(v_i, r_{1,i}, \ldots, r_{k-1,i}) =: A_i$. According to the definition of ambiguity, $K_{sec}(G, \ell, (o_1, \ldots, o_\ell), k\text{-}par \circ f_{app})$ is the set of possible $(w_1, \ldots, w_\ell)$ paths such that $w_i \in A_i$. After Lemma 4, $|K_{sec}(G, \ell, (o_1, \ldots, o_\ell), k\text{-}par \circ f_{app})| \geq k \cdot (k - (n - \delta(G)))^{\ell}$.

□

Theorem 4 implies that if we add more parallel streams than $n - \delta(G)$, which is the maximal number of non-adjacent vertices to any vertex in G, we can guarantee a lower bound on the cardinality of the ambiguity set, which is exponential in the length of the path ℓ.

Proposition 4. *If in the setting of Lemma 4 we have $k < n - \delta(G)$, then there exists a graph G, a path $(v_1, \ldots, v_\ell)$ in G, and k redundant streams, such that the number of possible paths $(w_1, \ldots, w_\ell)$ in G with $w_i \in A_i$ is 1.*

Proof. Let $G = (V, E)$ be a graph with $V = \{u_1, \ldots, u_{n-\delta(G)}, w_1, \ldots, w_{\delta(G)}, v\}$, let $out(u_1) = out(u_2) = \cdots = out(u_{k-1}) = \{w_1, \ldots, w_{\delta(G)}\}$, and let $out(v) = \{v, w_1, \ldots, w_{\delta(G)-1}\}$. Let the navigation path be $(v_1, \ldots, v_\ell)$ with $v_i = v$. Now, applying k-par for $k = n - \delta(G)$, at positions $1, \ldots, \ell$ we may pick the same $k-1$ redundant vertices $u_1, \ldots, u_{n-\delta(G)}$. In this case, the only possible path will be $(v_1, \ldots, v_\ell) = (v, \ldots, v)$. □

In the example given in the proof above, if we increase k by one, i.e., $k = n - \delta(G) + 1$, at each position on the path we have to add either v or w_i for some i, and according to Lemma 4, the number of possible paths grows to $k \cdot (k - (n - \delta(G)))^{\ell-1}$. In this sense, the bound given by Theorem 4 is sharp.

We note that similar analysis can be performed for other countermeasures, for example those results can be transferred to deterministic countermeasures in the spirit of Section 5.2. We will leave further possibilistic analyses based on outdegree, as well as their probabilistic counterparts, to future work.

5.4 The effect of added ambiguity

In this section, we investigate the effect of ambiguity added to the output of a network fingerprint, where added ambiguity is defined as follows.

Definition 8. *Given network fingerprints $f'_{net}, f''_{net} : O \to \mathcal{P}(O')$, we say that f''_{net} is a variant of f'_{net} with added ambiguity iff $\forall o \in O : f'_{net}(o) \subseteq f''_{net}(o)$.*

For example, a randomized network fingerprint can be interpreted as a deterministic network fingerprint with added ambiguity. In the probabilistic case, added ambiguity means that for all observations o, $P[Y' = o] > 0 \Rightarrow P[Y'' = o] > 0$. Note that f''_{net} cannot be interpreted as a network fingerprint composition of f'_{net} with a network fingerprint adding ambiguity, as the result of f''_{net} depends on the input of f'_{net}, and not on its output.

In the following, we show that using Definition 8, added ambiguity increases the security of a network fingerprint only in the possibilistic model, while in the probabilistic model it may also be harmful to security. This is an example of a case where the possibilistic definitions do not capture potential security vulnerabilities which stem from the probabilistic nature of systems.

5.4.1 Possibilistic case

The following proposition shows that if on each input a network fingerprint returns values which are possible observations of another network fingerprint on the same input, then the ambiguity set of this network fingerprint will be a subset of the ambiguity set of the second network fingerprint.

Proposition 5. *Let $f'_{net}, f''_{net} : O \to \mathcal{P}(O')$ be network fingerprints, and let observations $o'_1, \ldots, o'_\ell \in (O')^\ell$ and secret specification sec be given. If $\forall o \in O : f'_{net}(o) \subseteq f''_{net}(o)$, then $K_{sec}(G, (o'_1, \ldots, o'_\ell), f'_{net} \circ f_{app}) \subseteq K_{sec}(G, (o'_1, \ldots, o'_\ell), f''_{net} \circ f_{app})$.*

Proof. Given $sec(v_1, \ldots, v_\ell) \in K_{sec}(G, (o'_1, \ldots, o'_\ell), f'_{net} \circ f_{app})$, it follows that $\forall i \in [\ell] : o'_i \in (f'_{net} \circ f_{app})(v_i) = f'_{net}(f_{app}(v_i))$. As $\forall o \in O : f'_{net}(o) \subseteq f''_{net}(o)$, we obtain $\forall i \in [\ell] : f'_{net}(f_{app}(v_i)) \subseteq f''_{net}(f_{app}(v_i))$, and thus $o'_i \in f''_{net}(f_{app}(v_i)) = (f''_{net} \circ f_{app})(v_i)$. □

5.4.2 Probabilistic case

For the problem of added ambiguity to network fingerprints, we do not obtain a probabilistic confirmation of Proposition 5. In the following we give two simple examples for added ambiguity which decreases the probabilistic security guarantees of network fingerprints.

Let X be distributed uniformly with $P[X = x_1] = P[X = x_2] = 1/2$. As a first example, consider a network fingerprint f_{net} with a corresponding output random variable Y and observations o_1, o_2 with $P[Y = o_1|X = x_1] = P[Y = o_2|X = x_1] = P[Y = o_1|X = x_2] = P[Y = o_2|X = x_2] = 1/2$, as well as a network fingerprint f'_{net} with corresponding output random variable Y', with $P[Y = o_1|X = x_1] = P[Y = o_2|X = x_2] = 3/4$, $P[Y = o_2|X = x_1] = P[Y = o_1|X = x_2] = 1/4$. According to Definition 8, f'_{net} is a variant of f_{net} with added ambiguity. However, we obtain $H_\infty(X|Y) = 1 > 0.4 \approx H_\infty(X|Y')$, which means that f'_{net} gives worse security guarantees than f_{net}.

In our second example, the network fingerprint f'_{net} will not only change the probability distribution of f_{net}, but will allow an additional value o_3 to be observed. f_{net} will be defined as above, and f'_{net} is defined as $P[Y = o_1|X = x_1] = P[Y = o_2|X = x_1] = 1/2$, and $P[Y' = o_1|X = x_2] = P[Y' = o_2|X = x_2] = P[Y' = o_3|X = x_2] = 1/3$. Here we obtain $H_\infty(X|Y) = 1 > 0.6 \approx H_\infty(X|Y')$, which again means that adding ambiguity here makes the security guarantees worse.

5.5 The effect of added edges

A further issue we investigate is whether web-graphs with a higher density (which contain more hyperlinks between pages), leak less information than web-graphs with a lower density. Here again there is a discrepancy between the possibilistic and the probabilistic notions of security: while possibilistically adding links increases the security guarantees, probabilistically such a statement is not possible without further assumptions.

Proposition 6. *Let $G = (V, E)$ and $G' = (V, E \cup E')$ be web-graphs, and let a secret specification sec, a network fingerprint $f_{net} : O \to \mathcal{P}(O')$, and observations $o_1, \ldots, o_\ell \in (O')^\ell$ be given. Then,*

$$K_{sec}(G, (o_1, \ldots, o_\ell), f_{net} \circ f_{app}) \subseteq K_{sec}(G', (o_1, \ldots, o_\ell), f_{net} \circ f_{app}).$$

Proof. This proof goes similarly to the proof of Theorem 1. Let $sec(v_1, \dots, v_\ell) \in K_{sec}(G, (o_1, \dots, o_\ell), f_{net} \circ f_{app})$, from which follows that (1) $(v_1, \dots, v_\ell) \in paths(G, \ell)$ and (2) $\forall i \in [\ell] : o_i \in (f_{net} \circ f_{app})(v_i)$. Now it is left to show that (3) $(v_1, \dots, v_\ell) \in paths(G', \ell)$. This is trivially the case, because each edge in G is an edge in G' by definition, and thus each path in G is a path in G'. Therefore, (2) and (3) imply $sec(v_1, \dots, v_\ell) \in K_{sec}(G', (o_1, \dots, o_\ell), f_{net} \circ f_{app})$. □

Proposition 6 states that increasing the set of edges in the graph will also increase the resulting ambiguity set. However, a similar argument as in Section 5.4.2 shows that more links may lead to decrease of security. For example, more links may lead to new possible paths which have observations that can be distinguished by attackers with high probability. This is a further example for discrepancies between the possibilistic and the probabilistic models, where relying on possibilistic security may miss potential vulnerabilities. For a clear probabilistic statement on this issue, the link between more hyperlinks in a web-graph and the probabilistic notions of security should be investigated more closely, which we leave to future work.

5.6 Summary and discussion

This chapter presented formal analysis of different aspects of the problem of information leaks in web browsing traffic, based on the models presented in Chapter 3 and Chapter 4.

In Section 5.1, we showed that a network fingerprint can be strengthened by composing it with other network fingerprints. This notion of stronger network fingerprints can be interpreted as “the more, the better”; however, we show that in order to strengthen a network fingerprint f_{net} by composing it with other fingerprints, those should be applied *after* f_{net} (composition to the left), and not before f_{net}. This implies that network fingerprints such as packet segmentation, packet headers and tunneling do not decrease the security of underlying web applications; additionally, it can be used in the design and analysis of countermeasures.

In Section 5.2, we utilized properties of deterministic network fingerprints to allow simplified calculation of probabilistic security measures, leveraging and extending existing results for quantitative information flow.

In Section 5.3, we showed that a property of a web-graph's structure – a graph's outdegree, can be used for the design countermeasures which give lower bounds on the possibilistic security guarantees. Approaches which give probabilistic security guarantees in dependance of the web-graph structure are left to future work.

In Section 5.4, we analyzed the effect to security of ambiguity added to the output of a network fingerprint (e.g., randomizing a deterministic network fingerprint), and in Section 5.5, we analyse the effect of hyperlinks added to web-graphs. In both cases, we show an improvement of the provided possibilistic security guarantees, which allows intuitive statements such as “more randomness is better” and "denser graphs are better" to be made; however we show that those statements must not hold in the probabilistic security model, which indicates that probabilistic security captures better certain potential vulnerabilities. As future work, we consider probabilistic analysis to back up or refute the intuitive statements.

6 Caching

In this chapter, we incorporate caching into the models presented in Chapter 3 and Chapter 4. Furthermore, we investigate the effect of caching to the problem of information leakage in web browsing traffic. First, we show that in a system with caching enabled, our formal results from Chapter 5 still hold; second, we investigate under which circumstances caching itself increases or reduces the security of web applications.

6.1 Modeling caching

A browser cache is a temporary memory which stores downloaded web-objects, so that if an object is accessed by a user a second time, it can be fetched from the cache instead of from the web server. We assume that once a browsing session has ended, the browser cache is emptied, for example by using the privacy browsing mode offered by modern browsers.

The use of caching makes a browser request a subset of the web objects corresponding to a webpage; even if a web object is not requested, it has been downloaded at some previous point in the navigation path. We model the cache behavior by the *cached request* function *req*, which decides which web-objects of a webpage $v = (w_1, \ldots, w_n)$ to request, taking into account all web objects downloaded in the navigation path so far. Formally, given a navigation path $v_1, \ldots, v_\ell$, a cached request is a function

$$\mathit{req} : V^+ \to V,$$

such that the following conditions hold:

(i) $\forall i \in [\ell] : \mathit{req}(v_i) \sqsubseteq v_i$;

(ii) $\forall i \in [\ell] : \mathit{set}(v_i) \subseteq \mathit{set}(\mathit{req}(v_1) \sqcup \cdots \sqcup \mathit{req}(v_1, \ldots, v_i))$,

where (i) denotes that the requested web-objects are a subsequence of the actual web-objects contained in a vertex, and (ii) denotes that when the i-th vertex is visited, all its web-objects have been requested as part of the request for vertices $1, \ldots, i$.

To incorporate caching into the possibilistic security model from Section 3.2.2, we define a new function of possible fingerprints $g^c : V^+ \to \mathcal{P}(O)$ with $g^c(v_i, \ldots, v_i) = \bar{f}_{\mathit{app}} \circ \mathit{req}(v_1, \ldots, v_i)$, which unlike the g function defined in Section 4.1.1, does not

only take v_i, but also $v_1, \ldots, v_{i-1}$. We define cached ambiguity as follows:
$K^c_\phi(G, n, (o_1, \ldots, o_\ell), g^c) =$
$\{\phi(v_1, \ldots, v_\ell) \mid (v_1, \ldots, v_\ell) \in \mathit{paths}(G, \ell) \wedge \forall i \in [\ell] : o_i \in g^c(v_1, \ldots, v_i)\}$.

In the probabilistic model (see Section 3.2.3), the use of caching will affect the definition of the output random variable Y. If we decompose the input random variable X into $X_1, \ldots, X_\ell$ and Y into $Y_1, \ldots, Y_\ell$, if caching is disabled, the value of Y_i will depend only on the value of X_i, i.e., $P[Y_i = o_i | X = (v_1, \ldots, v_\ell)] = P[Y_i = o_i | X_i = v_i]$ (see Section 4.1.2); if caching is enabled, Y_i will additionally depend on the values of $X_1, \ldots, X_{i-1}$, and thus we obtain $P[Y_i = o_i | X = (v_1, \ldots, v_\ell)] = P[Y_i = o_i | X_1 = v_1, \ldots, X_i = v_i]$, which must not be equal to $P[Y_i = o_i | X_i = v_i]$.

6.2 Formal analysis revisited

In the formal analysis from Chapter 5, all presented results hold in a setting with enabled caching. In the possibilistic results in Theorem 1, Theorem 4, Proposition 5, Proposition 6, we substitute f_{app} by $f_{\mathit{app}} \circ \mathit{req}$, and for each $i \in [\ell]$, we substitute $f_{\mathit{app}}(v_i)$ by $f_{\mathit{app}} \circ \mathit{req}(v_1, \ldots, v_i)$. The proofs of Proposition 1, Proposition 2, and Proposition 4 define counterexamples, and as a system with disabled cache can be seen as a special case of a system with an enabled cache, those results can be directly used. In the probabilistic results in Theorem 2 and in Section 5.2, we only consider the random variables X and Y, and do not make assumptions about dependencies of their components $X_1, \ldots, X_\ell$ and $Y_1, \ldots, Y_\ell$; therefore, those results are still valid with caching enabled.

6.3 The effect of caching to security

Caching bears similarities to a network fingerprint (see Chapter 4), as the cached request *req* is a further function applied in order to obtain the final observations. However, there are two differences between caching and network fingerprints: (1) *req* is stateful, i.e., it takes into account previously visited vertices, while we consider only stateless network fingerprints; (2) *req* is applied directly to vertices, while network fingerprints are applied to observations, and thus, *req* is applied *before* the application and the network fingerprints. In this respect, the use of caching is similar to network fingerprint composition to the right, which according to Proposition 1 and Proposition 2 may be harmful to security. In the following, we investigate when the application of caching can be seen as harmful, and when it can be seen as helpful to security.

6.3.1 Examples

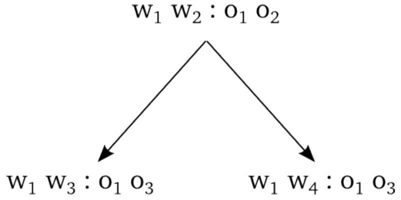
In this section we show that without further assumptions, no clear statement about the effect of caching on security can be made. We provide simple examples where caching may improve, worsen, or have no effect to security. In the examples below, we assume that if a web-object was cached at some point in the current browsing session, the browser does not request those objects from the server a second time.

Example 1: Caching neutral to security

Let $G = (V, E)$ be a web-graph without loops, and let all vertices in V contain distinct web objects, i.e., $\forall v, u \in V, v \neq u : set(v) \cap set(u) = \emptyset$. Therefore, no web objects are ever going to be loaded from the cache, and the attacker's observations with caching enabled are the same as if no caching were in place.

Example 2: Caching neutral to security

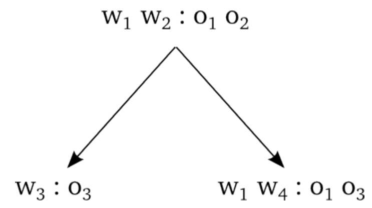
Let $G = (V, E)$ be a web-graph with vertices $v_1 = (w_1, w_2)$, $v_2 = (w_1, w_3)$, $v_3 = (w_1, w_3)$, and edges (v_1, v_2) and (v_1, v_3). Let the network fingerprint f_{net} be applied per file, producing the observations $(f_{net} \circ f_{app})(w_1) = o_1$, $(f_{net} \circ f_{app})(w_2) = o_2$, $(f_{net} \circ f_{app})(w_3) = o_3$, as depicted below:



Here, if the browsing starts at v_1, w_1 and w_2 will be cached; the second observation will be o_3, no matter whether the user has visited v_2 or v_3. If no caching is applied, the second observation will be (o_1, o_3) for both v_2 and v_3, and again an attacker would not be able to distinguish between both vertices. In this example, applying caching does not make any difference for the ability of an attacker to identify webpages.

Example 3: Caching good for security

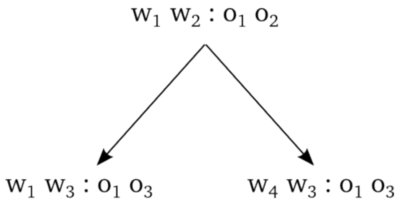
Let $G = (V, E)$ be a web-graph with vertices $v_1 = (w_1, w_2)$, $v_2 = (w_3)$, $v_3 = (w_1, w_4)$, and edges (v_1, v_2) and (v_1, v_3). Let the network fingerprint f_{net} be applied per file, producing the observations $(f_{net} \circ f_{app})(w_1) = o_1 = (f_{net} \circ f_{app})(w_4)$, $(f_{net} \circ f_{app})(w_2) = o_2$, $(f_{net} \circ f_{app})(w_3) = o_3 = (f_{net} \circ f_{app})(w_4)$, as depicted below:



In this example, if the browsing starts at v_1, again w_2 and w_3 will be cached. If caching is applied, the second observation will be o_3 for both v_2 and v_3, and if not, the second observation will be o_3 for v_2 and (o_1, o_3) for v_3. Thus, with applied caching, the attacker will have less certainty about the second webpage visited by the user, and thus in this case caching improves security.

Example 4: Caching bad for security

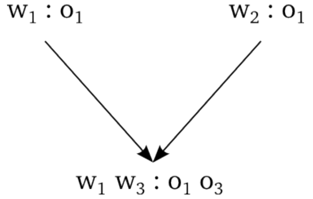
Let $G = (V, E)$ be a web-graph with vertices $v_1 = (w_1, w_2)$, $v_2 = (w_1, w_3)$, $v_3 = (w_4, w_3)$, and edges (v_1, v_2) and (v_1, v_3). Let the network fingerprint f_{net} be applied per file, producing the observations $(f_{net} \circ f_{app})(w_1) = o_1$, $(f_{net} \circ f_{app})(w_2) = o_2$, $(f_{net} \circ f_{app})(w_3) = o_3$, as depicted below:



In this example, if the browsing starts at v_1, again w_2 and w_3 will be cached. If caching is enabled, the second observation will be o_3 for v_2 and (o_1, o_3) for v_3; otherwise, the second observation will be (o_1, o_3) for both v_2 and v_3. Therefore, caching here makes it easier for an attacker to identify the second visited webpage, and can be considered harmful for security.

Example 5: Caching bad for security

Let $G = (V, E)$ be a web-graph with vertices $v_1 = (w_1)$, $v_2 = (w_2)$, $v_3 = (w_1, w_3)$, and edges (v_1, v_3) and (v_2, v_3). Let the network fingerprint f_{net} be applied per file, producing the observations $(f_{net} \circ f_{app})(w_1) = o_1 = (f_{net} \circ f_{app})(w_2)$, $(f_{net} \circ f_{app})(w_3) = o_3$, as depicted below:



Here, if the browsing starts at v_1, w_1 will be cached, and the second observation will be o_3; if the browsing starts at v_2, w_2 will be cached, and the second observation will be (o_1, o_3). Thus, by inspecting the second observation, the attacker will know which was the first visited webpage. If no caching is applied, the first observation will be o_1 for both v_1 and v_2, and the second observation will be (o_1, o_3), making it harder for the attacker to infer the first visited webpage. Thus, here caching can be considered harmful for security.

6.3.2 Caching as a privacy vulnerability

As the examples in the previous section show, caching may affect the identifiability of webpages in a positive or a negative way, or may not have any effect on the identifiability. We note that browser caching, and browser state in general, are a known source of concern in the security community, as shown in [25, 31, 14, 54], and disabling caching has been mentioned as the only reliable means for protection [31]. For example, a malicious website can find out whether certain webpages have been visited by using the knowledge that visited webpages take shorter time to load because some of their objects are cached, or by inspecting the color of visited links.

In Example 5 above, the described security vulnerability is similar in nature: the presence or absence of certain objects changes the observations in such a way that an attacker gains information about the previously opened content. The security vulnerability in Example 4 is somewhat different: the use of caching makes two webpages produce different observations, while without caching those webpages produce the same observations. As storing state information in a browser seems unavoidable, more analysis is needed to assess the severity of those problems and to find a good trade-off between bandwidth reduction from caching and the amount of leaked information.

6.3.3 Designing “good” caches

The design of a “good” cache, i.e., a cache which only exhibits a positive or a neutral effect to security, is desirable. A positive effect of caching to security can be seen in Examples 3 above. Furthermore, [29] show empirically that browser caching has a negative effect to their proposed website classification methods, which corresponds to a positive effect to security. A neutral effect of caching to security was demonstrated in Example 1 and Example 2 above. The following definition reflects the case when caching is neutral for security.

Definition 9. *For a graph $G = (V, E)$, an application fingerprint f_{app}, and a network fingerprint f_{net}, a cached request $req : V^+ \to V$ is called* fingerprint-invariant *iff for all ℓ, for all $\vec{v}_1, \vec{v}_2 \in paths(G, \ell)$, the following condition is fulfilled:*

$$(f_{net} \circ f_{app})(\vec{v}_1) = (f_{net} \circ f_{app})(\vec{v}_2) \Rightarrow (f_{net} \circ f_{app}) \circ req(\vec{v}_1) = (f_{net} \circ f_{app}) \circ req(\vec{v}_2).$$

This definition can be applied both in the possibilistic, and the probabilistic model. In the possibilistic case, it will mean that two paths with the same possible observations without caching enabled, will have the same observations with caching enabled. In the probabilistic case, it will mean that with or without caching, two paths will have the same probability distributions on their outputs.

If a cache fulfills fingerprint-invariance, an attacker will have no benefits when caching is applied. The following proposition gives a sufficient condition for fingerprint-invariance. Let the equivalence relation $\sim_f$ denote paths which have the same observations under $f_{net} \circ f_{app}$, i.e., $\vec{v}_1 \sim_f \vec{v}_2$ iff $(f_{net} \circ f_{app})(\vec{v}_1) = (f_{net} \circ f_{app})(\vec{v}_2)$.

Proposition 7. *Let $\vec{v}_1$ and $\vec{v}_2$ be paths in a graph with $\vec{v}_1 \sim_f \vec{v}_2$, and let $req : V^+ \to V$ be a cached request. If the following condition is fulfilled:*

$$\vec{v}_1 \sim_f \vec{v}_2 \Rightarrow req(\vec{v}_1) \sim_f req(\vec{v}_2),$$

then req fulfills fingerprint-invariance.

To ensure that this condition is fulfilled for certain network and application fingerprints, the operation of the cache should be specifically adapted. Fortunately, adapting which web-objects are to be cached and which not is possible both on the server side, and on the client side. On the server side, objects which cause fingerprint-invariance to be broken, can be set as *non-cacheable* by setting the `Cache-Control` or `Expires` HTTP headers, see [26]. On the client side, those elements can be deleted from the browser cache.

In our definitions, a cache decides deterministically which objects to be cached and which not. A stochastic cache, if designed properly, may reduce the probability that objects which help attackers in the identification of webpages are cached. As future work, we consider designing caches which guarantee a non-negative effect on security. As a first step, we consider devising techniques for finding objects which cause fingerprint-invariance to be broken.

7 Practical evaluation

In this chapter, we consider the performance of countermeasures in terms of effectiveness and overhead: given a countermeasure and a web-application, how effective is the countermeasure in reducing information leaks in this web-application, and what overhead does its application induce. We present a methodology for practical evaluation of countermeasures which utilizes the theory developed in the previous chapters, and develop simulation approaches for dealing with computational challenges. This methodology is demonstrated in two case studies we performed, evaluating (1) the auto-suggest scenario, and (2) the web-navigation scenario.

7.1 Choice of performance measures

The first step we take towards a practical evaluation is establishing measures for quantifying performance. In the following we present the measures used in our work, which quantify two aspects of a countermeasure's performance: security (for reasoning about the effectiveness of a countermeasure), and overhead (for reasoning about the cost of a countermeasure).

7.1.1 Security

To measure security, we use the probabilistic definitions from Section 3.2.3. We chose those definitions over the possibilistic security definitions from Section 3.2.2 because the results from Chapter 5 indicate that possibilistic definitions do not adequately capture potential security vulnerabilities. The security measures we consider here are the initial uncertainty $H_\infty(X) = -\log \max_{x \in \mathcal{X}} P[X = x]$, and the remaining uncertainty

$$H_\infty(X|Y) = -\log \sum_{y \in \mathcal{Y}} P[Y = y] \max_{x \in \mathcal{X}} P[X = x|Y = y].$$

As shown in Section 5.2, if the applied countermeasures are deterministic, we can calculate the remaining uncertainty as

$$H_\infty(X|Y) = -\log \sum_{y \in \mathcal{Y}} \max_{x \in B(y)} P[X = x],$$

where $B(\cdot)$ maps observations to blocks in the partition induced by the countermeasure. Refer to Section 5.2 for further algorithms for measuring remaining uncertainty, e.g. relying on the assumption that X is uniformly distributed.

7.1.2 Overhead

The employment of countermeasures bears both computational and communicational overhead. In the following we will concentrate on the communicational cost of countermeasures, assuming that all computationally costly calculations can be performed offline. Let the *size* of an observation be a function $size : O \to \mathbb{N}$, and let $v_1, \ldots, v_\ell$ be a navigation path which produces the observations $o_1, \ldots, o_\ell$ without a countermeasure applied, and the observations $o'_1, \ldots, o'_\ell$ if the countermeasure c is applied. We define the *overhead* of c as $ovhd(c) = \sum_i size(o'_i) - size(o_i)$. If c is probabilistic, we define the *expected overhead* of c as

$$\begin{aligned} exp\_ovhd(c) &= \sum_o \sum_{o'} P[C^{out} = o', C^{in} = o](size(o') - size(o)) \\ &= \sum_o P[C^{in} = o] \sum_{o'} P[C^{out} = o' | C^{in} = o](size(o') - size(o)). \end{aligned}$$

In the case when c is the only applied (composed) countermeasure, we obtain

$$exp\_ovhd(c) = \sum_x P[X = x] \sum_y P[Y = y | X = x](size(y) - size(f_{app}(x))).$$

If c is deterministic, we obtain

$$exp\_ovhd(c) = \sum_x P[X = x](size(c(x)) - size(f_{app}(x))).$$

7.2 Overview of evaluation approach

To evaluate the performance of a given countermeasure in terms of the measures discussed in the previous section, we consider the following approach:

(1) get a graph $G = (V, E)$ corresponding to a web application;

(2) get the application fingerprints of the vertices in G;

(3) estimate the probability of the random variable $sec(X)$, representing the secret value contained in the user's choice of a path in G;

(4) calculate the initial uncertainty $H_\infty(sec(X))$, and given a countermeasure c, calculate the conditional probability $P[Y|sec(X)]$ before and after the application of the countermeasure;

(5) calculate the remaining uncertainty $H_\infty(sec(X)|Y)$ with and without the countermeasure c applied, as well as the expected overhead $exp\_ovhd(c)$.

We demonstrate the challenges which this approach presents in two case studies we performed, evaluating equal-sized bucketing (see Section 4.3.1) on (1) the auto-suggest scenario, and (2) the web-navigation scenario. The remainder of this chapter elaborates on those case studies.

7.3 Case study: the auto-suggest scenario

The first set of experiments we performed targets the auto-suggest scenario as defined in Section 3.1.2. In this scenario, we are given a dictionary D, and on each keystroke of a user, a string v is sent to the web server, which sends back a list of suggestions.

7.3.1 Experimental setup

In our experiments, we assume that the attacker knows that the user is looking only for illness-related words, and we set D to be the 1183 words from the hyponym tree[1] of the word "illness", contained in the WordNet English lexical database [24]. We generated the prefix tree G over D, using the Python NetworkX library [28] for handling graphs.

To estimate the probabilities that a user types a certain string, we performed Google queries for all 1183 words in D and calculated the relative frequencies of the words on the leaves. We then traversed the tree towards the root and set the probability of a vertex to be the a sum of the probabilities of its children, and the probabilities of the edges outgoing from a vertex v proportional to the probabilities of v's children. We calculate the probability of a user's choice of a path $(v_1, \ldots, v_\ell)$ as $P[X = (v_1, \ldots, v_\ell)] = P[X_\ell = v_\ell | X_{\ell-1} = v_{\ell-1}] \cdot P[X_{\ell-1} = v_{\ell-1} | X_{\ell-2} = v_{\ell-2}] \cdots P[X_2 = v_2 | X_1 = v_1]$ (see the discussion of the Markov property in Section 3.2.3).

For the application fingerprint $f_{app}(v)$, we take the size of the auto-suggest list given by Google's auto-complete service on query v. To instantiate those values, we issued 11678 queries for all vertices in G. In our setup, we specify that the entire typed-in string is sensitive information by setting the secret specification sec to be the identity function, i.e. $sec = sec_1$ (see Section 3.2.1).

[1] A *hyponym tree* of a term has this term on its root, and the children of a vertex in this tree are words in an "is-a" relationship to the vertex.

7.3.2 Results

We evaluate the performance of the countermeasure c = "equal-sized bucketing" (i.e., in each bucket, the same number of observations are mapped, see Section 4.3.1). We computed the initial uncertainty $H_\infty(sec(X))$, the remaining uncertainty $H_\infty(sec(X)|Y)$ with and without the countermeasure c applied, as well as the relative expected overhead $exp\_ovhd(c)$, for a varying path length $\ell = 2, \ldots, 8$. In all cases, the initial uncertainty was between 5.38 and 5.55. The remaining uncertainty when the countermeasure c was not applied was 0, meaning that the web application is highly vulnerable to attacks: an attacker can infer the secret input word with probability 1.

Figure 7.1 depicts the results when applying the countermeasure c, showing the expected relative overhead (i.e., expected overhead relative to the original size of the application fingerprints) needed to achieve certain levels of remaining uncertainty for different path lengths. With an expected relative overhead of between 1.3 and 3.9, the remaining uncertainty reaches the initial uncertainty, making the system maximally secure (i.e., the attacker does not learn any new information from observing the traffic), and a remaining uncertainty of more than 3.5 bits can be achieved with an expected relative overhead of 0.56–1.06. Furthermore, the results show that for shorter paths, good security can be achieved with a smaller overhead, e.g., for paths of length up to 4, maximal security is reached with an expected relative overhead of up to 1.7.

7.4 Case study: the web-navigation scenario

We additionally evaluate the web-navigation scenario (see Section 3.1.1). This evaluation proved to be more challenging because of the high complexity of web-graphs. We discuss a straightforward approach and its limitations in Section 7.4.1, and present an alternative simulation approach in Section 7.4.2, which we used in the experiments described in Section 7.4.3 and Section 7.4.4.

7.4.1 Straightforward approach and its limitations

A straightforward approach we consider consists of gaining the structure of the web-graphs corresponding to popular websites, and measuring the effects of countermeasures on those graphs. In a preliminary study, we devised a web crawler by adapting the source code of `wget 1.12` [27] to fit our needs. Our initial results showed that gaining the structure of web graphs is impractical for an evaluation study consisting of popular websites because of the large number of possible navigation paths, which makes computation of performance measures infeasible. We present several possible approaches for handling this problem:

(i) limit the evaluation to websites which induce small webgraphs;

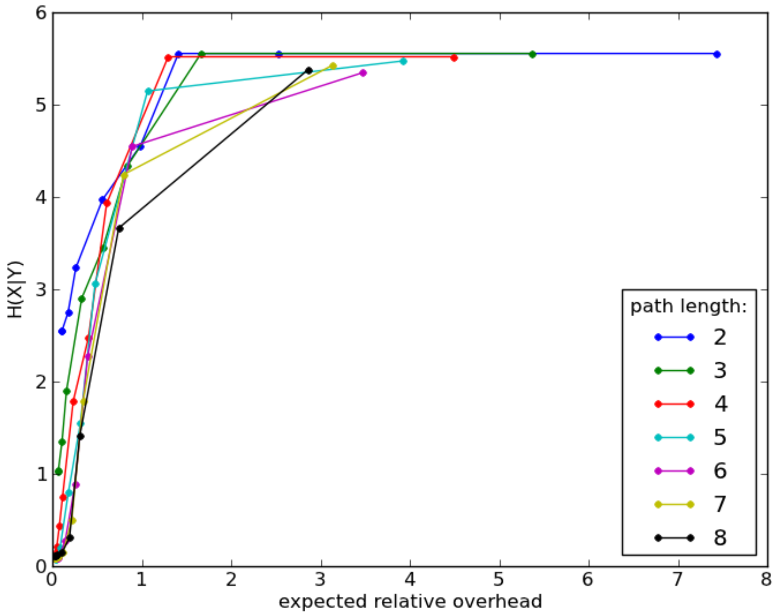


Figure 7.1: The auto-complete scenario: the conditional min entropy versus the expected relative overhead caused by equal-sized bucketing with varying number of buckets. Curves of different color correspond to a different path length. The dots correspond to number of buckets in the range $1, 2, 5, 10, 20, 40, 60, 80, 100$, where the numbers decrease to the right.

(ii) crawl websites partially, e.g. limit the crawling to a certain depth or the visited paths to a certain number;

(iii) use a crawl-specification, which limits the crawling to a pre-defined subset of paths (see [9]);

(iv) use a randomized approach for generating graphs which have the structural properties of websites, but have a controlled size.

A disadvantage of partial crawling (Approach (ii)) is that it may result in a mutilated graph structure, e.g., obtaining a high number of “leaf” vertices, which do not occur in the original web-graph. Approach (iv) has the disadvantage that the obtained results may hold for the simulated graphs, but not for real instances. Nevertheless, for our experiments we consider the latter approach, because the flexible choice of parameters it offers allows the generation of a high number of small web-graphs, and the specification of desired graph properties, such as graph density. We elaborate on this approach in the following section.

7.4.2 Simulation approach

In the simulation approach we propose, we generate random graphs which "look like" real web-graphs, and perform our experiments on them. In the following, we review structural properties which are known to be fulfilled by web-graphs. Furthermore, we describe the approach we use for generating web-graphs. For a survey of the advances in understanding real-world graphs, as well as in generation of realistic graphs, we refer the reader to Chakrabarti and Faloutsos [7].

Properties of web-graphs

There is a large body of research targeted at capturing the structural graph properties fulfilled by websites and the Web in general. In the following, we present several important results from this area.

Empirical studies have found that both in- and out-degrees of vertices in the Internet graph are distributed according to a *power law* [34, 1]; degrees of vertices in a graph follow a power-law if the probability that a vertex has a degree i is proportional to $1/i^{\alpha}$ for some $\alpha > 1$. Power laws have been also found in other aspects of the Web, for example in the physical Internet topology [23], and in the structure of online social networks [38]. The web has also been found to have a "bowtie" structure, with a large strongly connected component in the middle [4]. Dill et al. [19] have observed a self-similarity in the web: properties of the web such as "bowtie" structure and power laws hold for both the web at large, and for cohesive sub-regions of it. Additionally, web-graphs were found to have a small diameter [2].

Generating web-graphs

A classical approach for randomized generation of graphs was proposed by Erdős and Rényi [21]: given a set of vertices V, the generation algorithm decides with equal probability for each pair of vertices whether it is included in the set of edges E or not. As such graphs do not ensure properties of web-graphs such as power laws, specific generators for realistic web-graphs have been proposed, e.g. [44, 43, 8].

In our experiments, we use R-MAT (which stands for Recursive Matrix) [8], which generates directed graphs with numbers of vertices and edges provided by the user. R-MAT starts with an empty adjacency matrix, which is subdivided into four equal-sized blocks. With predefined probabilities a, b, c, d with $a + b + c + d = 1$, recursively one of the blocks is chosen, and is further subdivided into four equal-sized blocks. The recursion ends when a 1×1 block is chosen. This 1×1 matrix corresponds to two vertices, and an edge between those vertices is set, where duplicate edges are not allowed.

R-MAT provides an approach for calculating an expected outdegree of the vertices v in a generated graph:

$$E[\delta(v)] = \frac{1}{2^n} \sum_{k=0}^{m} k \binom{m}{k} \sum_{i=0}^{n} \binom{n}{i} [\alpha^{n-i}(1-\alpha)^i]^k [1 - \alpha^{n-i}(1-\alpha)^i]^{m-k},$$

where 2^n is the number of vertices, m is the number of edges, and $\alpha = a + b$. In our experiments, we utilize this formula to generate graphs with specific expected outdegrees.

7.4.3 Experimental setup

To generate web-graphs, we adapt `pywebgraph 2.72` [46], a Python implementation of R-MAT, and for the parameters a, b, c, d we use the conjecture by [8] that $a : b = a : c = 75 : 25$. In our experiments, we generated graphs with 100 vertices and expected outdegrees $5, 10, 15, 20, 30, 40, 50$, which allows comparing the performance of countermeasures for graphs with different densities. We note that the average internal outdegree of web-graphs has been used for classifying websites into functional categories (e.g. academic, blog, shop, etc.) [40]. To compute the probability of a path taken by a user, we assume that the user performs a random walk in the web-graph starting at a fixed vertex, and we set a uniform distribution on outgoing vertices. Having a fixed root vertex corresponds to an attacker who knows the entry point of the user.

To generate the application fingerprints, we devise an approach based on the result of Kamps and Koolen [33] showing that in Wikipedia, there is a strong correlation between outdegree of a webpage and document length. Note that Kemps and Koolen do not find such a high correlation in the web in general; nevertheless, we consider this a reasonable first approximation for our experiments. We assume that a webpage contains only one file, and calculate the size of a webpage v according to its outdegree $\delta(v)$ as:

$$\mathit{size}(v) = p \cdot \delta(v) + q + \mathit{rand},$$

where p and q are parameters, and rand is random noise. For our analysis, we specify as secret the set of visited vertices (i.e. we ignore their ordering), and express this by setting sec to be the set representation of a path sec_2 (see Section 3.2.1).

7.4.4 Results

In the above presented setup, we tested the effectiveness of the different variants of the "equal-sized bucketing" countermeasure c (see Section 4.3.1), for number of buckets between 1 and 20. We fixed the path length to $\ell = 5$, and varied the density of the generated graphs by setting the expected outdegree between 5 and 50. The initial uncertainty was between 8.2 and 15.6, with higher values for denser graphs.

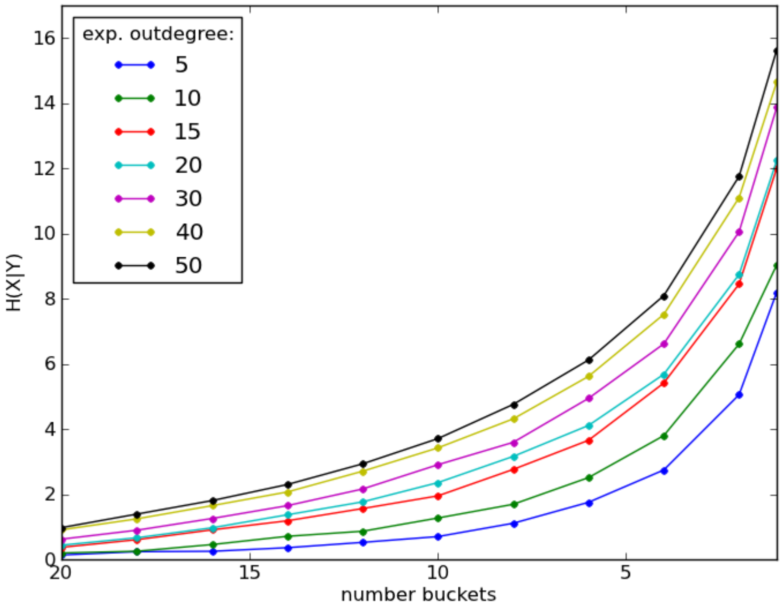


Figure 7.2: The web-navigation scenaro: the conditional min entropy versus the number of buckets of equal-sized bucketing with varying number of buckets. The different curves correspond to web-graphs with a different density (measured in expected outdegree).

In all cases, when no countermeasure was applied, the remaining uncertainty was 0, and thus the attacker learns the entire secret information.

Figure 7.1 depicts the results when applying the countermeasure c. In all cases, less buckets guarantee a higher remaining uncertainty. With 5 buckets, the remaining uncertainty is between 2.2 and 6.9, and with 2 buckets, the remaining uncertainty is between 5.1 and 11.8, where the higher values correspond to denser graphs. We see that for graphs of different densities the effect of the applied countermeasure is similar, however, the denser a graph is, the bigger the remaining uncertainty with any number of buckets.

8 Conclusion

In this work, we have presented a formal framework that enables reasoning about information leaks in web browsing traffic. To quantify the vulnerability of web applications to information leaks, we have used information-theoretic entropy and ambiguity sets, which bridges the problem of information leaks in web traffic to the growing body of research in quantitative information flow. By means of our framework, we have performed a formal analysis of the design of countermeasures, and have investigated the role of network protocols and caching to security. Furthermore, we have applied our definitions in an empirical evaluation of the effectiveness and overhead of countermeasures in a sample of web applications.

The proposed framework can facilitate the development of robust solutions against information leaks in web applications, and we elaborate on two possible lines of future work. First, the presented methodology can be used for the design of countermeasures providing formal security guarantees at a reasonable cost. A probabilistic extension of our analysis from Section 5.3 can be utilized for the design of strong countermeasures according to the graph structure of the web application. Good trade-offs between security and overhead can be established empirically (see Chapter 7), and parameters for countermeasures delivering optimal trade-offs can be obtained algorithmically, along the lines of [37].

Second, our framework can be used as a semantic basis for the development of language-based approaches for web applications (e.g., see [56]). This would enable the development of web applications which by construction provide quantitative guarantees against information leaks.

Bibliography

[1] R. Albert, H. Jeong, and A. L. Barabasi. The diameter of the world wide web. *Nature*, 401:130–131, 1999.

[2] Reka Zsuzsanna Albert. *Statistical mechanics of complex networks*. PhD thesis, Notre Dame, IN, USA, 2001.

[3] George Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Proc. Privacy Enhancing Technologies Workshop (PET)*, pages 1–11, May 2005.

[4] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Comput. Netw.*, 33:309–320, June 2000.

[5] Christian Cachin. Entropy Measures and Unconditional Security in Cryptography. Dissertation No. 12187. ETH Zürich, 1997.

[6] Vinton G. Cerf and Robert E. Khan. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22:637–648, 1974.

[7] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1):2, 2006.

[8] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *SIAM International Conference on Data Mining*, 2004.

[9] Peter Chapman and David Evans. Automated black-box detection of side-channel vulnerabilities in web applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 263–274. ACM, 2011.

[10] Shuo Chen, Rui Wang, Xiaofeng Wang, and Kehuan Zhang. K.: Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *IEEE Security and Privacy Symposium*, 2010.

[11] Heyning Cheng, Heyning Cheng, and Ron Avnur. Traffic analysis of ssl encrypted web browsing, 1998.

[12] Hyoung-Kee Choi and John O. Limb. A behavioral model of web traffic. In *ICNP*, pages 327–334, 1999.

[13] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *J. Log. and Comput.*, 15:181–199, April 2005.

[14] Andrew Clover. Css visited pages disclosure, 2002.

[15] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[16] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, and M. K. Reiter. On web browsing privacy in anonymized netflows. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 23:1–23:14, Berkeley, CA, USA, 2007. USENIX Association.

[17] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, second edition, 2006.

[18] George Danezis. Traffic analysis of the http protocol over tls, 2007.

[19] Stephen Dill, Ravi Kumar, Kevin S. Mccurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. *ACM Trans. Internet Technol.*, 2:205–223, August 2002.

[20] Magdalini Eirinaki, Michalis Vazirgiannis, and Dimitris Kapogiannis. Web path recommendations based on page ranking and markov models. In *In WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management, 2–9*, pages 2–9. ACM Press, 2005.

[21] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[22] Barbara Espinoza and Geoffrey Smith. Min-entropy leakage of channels in cascade. In *FAST 2011 – 8th International Workshop on Formal Aspects of Security and Trust, to appear*, 2011.

[23] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 251–262, New York, NY, USA, 1999. ACM.

[24] Christiane Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.

[25] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 25–32, New York, NY, USA, 2000. ACM.

[26] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[27] Free Software Foundation. Gnu wget.

[28] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[29] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial na&#239;ve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 31–42, New York, NY, USA, 2009. ACM.

[30] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Workshop on Privacy Enhancing Technologies*, 2002.

[31] Collin Jackson and Dan Boneh. Protecting browser state from web privacy attacks. In *In Proceedings of the International World Wide Web Conference*, pages 737–744, 2006.

[32] Edwin T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4):620–630, May 1957.

[33] Jaap Kamps and Marijn Koolen. Is wikipedia link structure different? In *WSDM*, pages 232–241, 2009.

[34] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: measurements, models, and methods. In *International Conference on Combinatorics and Computing*, 1999.

[35] Boris Köpf and David Basin. An Information-Theoretic Model for Adaptive Side-Channel Attacks. In *Proc. 14th ACM Conference on Computer and Communications Security (CCS '07)*, pages 286–296, New York, NY, USA, 2007. ACM.

[36] Boris Köpf and David Basin. Automatically Deriving Information-theoretic Bounds for Adaptive Side-channel Attacks. *Journal of Computer Security*, 19(1):1–31, 2011.

[37] Boris Köpf and Markus Dürmuth. A Provably Secure and Efficient Countermeasure against Timing Attacks. In *Proc. 22nd IEEE Computer Security Foundations Symposium (CSF '09)*, pages 324–335. IEEE, 2009.

[38] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 611–617, New York, NY, USA, 2006. ACM.

[39] Marc Liberatore and Brian Neil Levine. Inferring the Source of Encrypted HTTP Connections. In *Proc. ACM conference on Computer and Communications Security (CCS)*, pages 255–263, October 2006.

[40] Christoph Lindemann and Lars Littig. *Classification of Web Sites at Supergenre Level*, volume 42 of *Text, Speech and Language Technology*, pages 211–235. Springer, Genres on the Web edition, 2011.

[41] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Proc. 15th European Symposium on Research in Computer Security (ESORICS '10)*, LNCS. Springer, 2010.

[42] Xiapu Luo, P. Zhou, E. W. W. Chan, Wenke Lee, and R. K. C. Chang. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, 2011.

[43] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. *SIGCOMM Comput. Commun. Rev.*, 30:18–28, April 2000.

[44] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64(2):026118, 2001.

[45] Ramesh R. Sarukkai. Link prediction and path analysis using markov chains. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 377–386, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[46] Saket Sathe. pywebgraph: Python web graph generator.

[47] Claude E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27:379–423, July 1948.

[48] Geoffrey Smith. On the foundations of quantitative information flow. In *FOSSACS*, pages 288–302, 2009.

[49] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: discovery and applications of usage patterns from web data. *SIGKDD Explor. Newsl.*, 1(2):12–23, 2000.

[50] Bonnie J. Strait and T. Gregory Dewey. The shannon information entropy of protein sequences. *Biophysical journal*, 71(1):148–155, July 1996.

[51] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*. Society Press, 2002.

[52] Dennis Volpano and Geoffrey Smith. Probabilistic noninterference in a concurrent language. *Journal of Computer Security*, 7:34–43, 1998.

[53] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol, 1997.

[54] Zachary Weinberg, Eric Y. Chen, Pavithra Ramesh Jayaraman, and Collin Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *IEEE Symposium on Security and Privacy*, pages 147–161, 2011.

[55] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.

[56] Kehuan Zhang, Zhou Li, Rui Wang, XiaoFeng Wang, and Shuo Chen. Sidebuster: automated detection and quantification of side-channel leaks in web application development. In *CCS 2010*, pages 595–606. ACM, 2010.