Quantitative Information Flow of Side-Channel Leakages in Web Applications

Huang, Xujing

Quantitative Information Flow of Side-Channel Leakages in Web Applications

Xujing Huang

Thesis submitted for the degree of Doctor of Philosophy

Queen Mary, University of London

10 May 2016

Statement of Originality

I, Xujing Huang, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Details of collaboration and publications:

[66] X. Huang and P. Malacaria. Sideauto: quantitative information flow for side-channel leakage in web applications. In *Proceedings of the 12th ACM Workshop on privacy in the electronic society*, pages 285-290. ACM, 2013.

[65] X. Huang. It Leaks More Than You Think: Fingerprinting Users from Web Traffic Analysis. Acta Informatica Pragensia, pages 206-225, 2015, 4(3).

Signature: Xujing Huang
Date: 10 May 2016

Abstract

It is not a secret that communications between client sides and server sides in web applications can leak user confidential data through side-channel attacks. The lower-lever traffic features, such as packet sizes, packet lengths, timings, etc., are public to attackers. Attackers can infer a user's web activities including web browsing histories and user sensitive information by analysing web traffic generated during communications, even when the traffic is encrypted.

There has been an increasing public concern about the disclosure of user privacy through side-channel attacks in web applications. A large amount of work has been proposed to analyse and evaluate this kind of security threat in the real world.

This dissertation addresses side-channel vulnerabilities from different perspectives. First, a new approach based on verification and quantitative information flow is proposed to perform a fully automated analysis of side-channel leakages in web applications. Core to this aim is the generation of test cases without developers' manual work. Techniques are implemented into a tool, called SideAuto, which targets at the Apache Struts web applications.

Then the focus is turned to real-world web applications. A black-box methodology of automatically analysing side-channel vulnerabilities in real-world web applications is proposed. This research demonstrates that communications which are not explicitly involving user sensitive information can leak user secrets, even more seriously than a traffic explicitly transmitting user information.

Moreover, this thesis also examines side-channel leakages of user identities from Google accounts. The research demonstrates that user identities can be revealed, even when communicating with external websites included in Alexa Top 150 websites, which have no relation to Google accounts.

Contents

List of Figures

List of Tables

Acknowledgements

I am very thankful to my supervisor Pasquale Malacaria for guiding and supporting my work during the past years. When I got new ideas, he always encouraged and supported me to do.

I am also very grateful to Edmund Robinson for his patient and kindness in helping me review the thesis.

I would like to thank the anonymous reviewers of CCS, ACSAC and NDSS for their valuable comments and suggestions to improve the work described in this thesis, though they rejected the work submitted.

I am grateful to my fellow PhD student Quoc-Sang Phan for helping me entering the field of program analysis and his generosity of sharing materials with me. I am also grateful to my fellow PhD students Yihan Tao and Xinyue Wang for their friendship and many discussions about the research thought we work in different fields.

I would also like to thank my parents for their endless love, encouragement and support. They give me courages to overcome many many difficulties. And also many thanks to the sisters and brothers in London Chinese Baptist Church for their prayers, loves and encouragements.

At the last, all the praise, glory and thanks to the only God! Because of him, I learn the patient and perseverance. Because of him, I have courages and can face and overcome all the difficulties.

Chapter 1

Introduction

Communications between the clients and the servers lead to security threats of user privacy. An attacker is capable of stealing user confidential information by eavesdropping network side channels, such as packet sizes, packet lengths, and timing information. These side channels are public to attackers, and they may vary depending on user sensitive information. For example, different user inputs in a search engine can generate unique traffic sequences which reveal user sensitive inputs [39].

There has been a large amount of work on analysing side-channel vulnerabilities in web applications, which demonstrates that network traffic, even though it is encrypted, can reveal a large amount of confidential information, e.g. [40, 76, 111]. This kind of attack is also called as traffic-analysis attack.

1.1 Overview

In this thesis, we aim to provide an automated analysis for detecting side-channel vulnerabilities in web applications for developers, mainly via packet sizes and packet directions. We use white-box and black-box approaches to analyse web applications based on Struts [16] framework and real-world applications respectively.

In white-box approach, we analyse vulnerabilities in Struts-based web applications. Static analysis and symbolic execution [72] are performed to automatically generate test cases. Test cases are executed and their web traffic is collected. By analysing the web traffic, the leaks of user sensitive information can be evaluated using quantitative information flow.

However, the current white-box approach can only be used to analyse Struts web applications, and the source code of the web applications is required.

To analyse real-world web applications, we extend our analysis using a black-box approach. The approach is motivated by [37], which uses a crawler to detect side-channel vulnerabilities. We develop an automated detection system containing an automated test case generation for real-world web applications. After executing the test cases generated, guessing probabilities [82] are used to evaluate the leakages based on the observations of web traffic.

During the experiments on the Google website, we discover unexpectedly that Google user identities may be leaked largely from user accounts. To future explore, our final work particularly analyse the leakage of user identities from Google accounts, through the communications with third-party websites included in Alexa Top 150 websites [1]. This widens the communications under examination, which are not only within a single website, but also between different websites.

Notice that side-channel vulnerabilities in web applications have been commonly examined through communications explicitly transmitting user sensitive information. Little work has particularly studied communications inexplicitly transmitting sensitive information. In our research, therefore, we also examine communications which appear to transmit no user sensitive information, i.e. without explicit interactions with user sensitive information.

Overall, the main purpose of this thesis is to analyse side-channel vulnerabilities in web applications from a perspective of a developer. We aim to provide references and suggestions about the vulnerabilities in a web application for a developer, rather than giving an accurate evaluation of the leakage. According to the references, the developer can further explore and make countermeasures on the specific vulnerabilities.

1.2 Contributions

In summary, the primary contributions of this thesis are as follows:

1. This thesis proposes a system towards automated test case generation in analysing side-channel vulnerabilities in Struts web applications. It exploits techniques based on verification and static analysis to fulfil the automation of test case generation. It is the first research referring automated test case generation to the side-channel analysis in web applications. In this sense it is the first tool towards a completed automated side-channel analysis in web applications.

2. This thesis demonstrates that user sensitive information can be leaked from communications which appear to be irrelevant with the sensitive information. Moreover, it also discovers that user secrets can be inferred from communications with "third-party" websites which bear completely no relation to sensitive information.

3. This thesis shows that user identities, even Google users, are vulnerable through traffic analysis. It also examines that whether cookies and logged user accounts are the factors causing web traffic varying depending on user identities.

1.3 Thesis Outline

This thesis gives a comprehensive analysis of side-channel vulnerabilities against user privacy in web applications. The outline of this thesis is described as follows.

Chapter 2 introduces the background and the primary techniques used throughout this thesis. In summary, static analysis and symbolic execution are performed on the source code of web applications; black-box analysis is used for analysing real-world web applications; hidden Markov models motivate our analysis of packet data and quantitative information flow and probability of guessing are used to measure leaks. Furthermore, this chapter introduces a basic fingerprinting model and a threat scenario. They are the essentials of the models and the threat scenarios proposed in this thesis.

Chapter 3 proposes an automated system which advances on the automated test case generation for Struts-based web applications. This complements previous tools towards a fully automated system.

Chapter 4 extends side-channel analysis to real-world web applications using a black-box approach. It demonstrates that user privacy can be sneaked through communications which appear to have no relation to sensitive information. Moreover, an approach, motivated by the hidden Markov model, is proposed to construct a traffic pattern best matched to the web traffic observed in a communication. Then the information leakage is evaluated by the probability of guessing a secret correctly in one try [82].

Chapter 5 proposes a novel threat scenario in terms of leakage of use identities. User identities examined are leaked from 50 Google accounts to communications with third-party websites included in Alexa Top 150 websites. Moreover, four testing scenarios are developed to explore if cookies and logged user accounts can be the leaking sources of user identities.

Then **Chapter 6** presents a review of previous work related to this research, including side-channel vulnerabilities, fingerprinting users, leakage of user locations, automated test case generation, hidden Markov models, security threats of web cookies and quantitative information flow.

Finally, **Chapter 7** concludes this thesis and suggests the future work.

Chapter 2

Preliminaries

This chapter introduces the background and the primary techniques used throughout this thesis.

2.1 Quantitative Information Flow

In the context of protecting confidential information, quantitative information flow (QIF) provides a powerful approach to analyse leakages of sensitive information in secure information flow. It is used to quantify how much confidential information is leaked.

2.1.1 Information Flow and Non-interference

Information flow shows the transfer of information from a variable x to a variable y in a given program. Denning et al. [50] present a secure flow of information based on a lattice structure through a program. They partition variables into security labels, where the confidential variables are given the “high” security label H and the public variables are taken the “low” security label L. “Low” security variables are publicly observable.

Figure 2.1(a) shows the fundamental model of information flow. A program takes confidential input H, public input L and produces output O. It can be seen that confidential information H potentially flows to the observation O. Figure 2.1(b) shows an attacker model where an attacker aims to get the confidential information H from the public input L and the observation O.

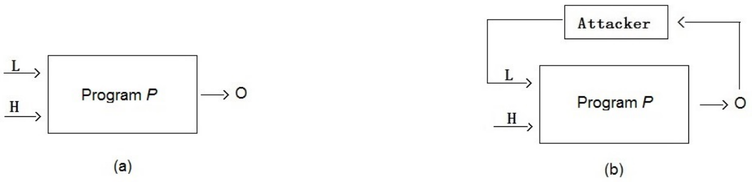


Figure 2.1: Information flow in (a) and Attacker model in (b)

A system P is deemed to be vulnerable if there is information flow from confidential information H to observable output O. The following code gives an example of a password authentication program.

```
if (H==L)
O= true;
else
O=false;
```

Figure 2.2: A password authentication program

H is the password, i.e. the confidential information, and L is the user input. O is the observable output telling if the user inputs the correct password. $H == L$ indicates the input is accepted. In this case, there is still information flow from H to L even the password input is incorrect, i.e. $H \neq L$. More precisely, with this prompting, an attacker can finally get the correct password if he is able to perform enough attempts.

To guarantee no such information-flow leaks in a system, Goguen and Meseguer [60] introduce a security policy in a general automaton framework: *non-interference*. The formal expression of non-interference is defined as:

Definition 1 (Non-interference) *Given a program P, characterised by a function f, taking public inputs L, secret inputs H and producing outputs O, the program P is non-interference guaranteed, i.e. with no information-flow leaks, if and only if: for all input values: $l_1, l_2 \in L$ and $h_1, h_2 \in H$:*

$$\forall \quad l_1 = l_2 \quad \wedge \quad f(l_1, h_1) \to o_1 \quad \wedge \quad f(l_2, h_2) \to o_2 \quad \Rightarrow \quad o_1 = o_2,$$

where $f(l_1, h_1) \to o_1$ denotes program P takes inputs of l_1 and h_1 and generates a result of o_1.

This means a program P is non-interference guaranteed if the public output does not depend on any secret inputs.

2.1.2 Partition and Equivalence Relation

First, define *partition* and *equivalence relation*, which will be used in the measurement of interference in next section 2.1.3.

Given a set, a *partition* X of a set S is a family of subsets of S, in such a way that every element $s \in S$ is contained in one and only one block of X. A block of the partition X is a set in X.

A partition is defined in a mathematical way as follows.

Definition 2 (Partition) *A partition of a set S is a family of sets X if and only if all the following conditions hold:*

1. $\emptyset \notin X$

2. $\bigcup_{A \in X} A = S$

3. if $A,\ B \in X$ *and* $A \neq B$ *then* $A \cap B = \emptyset$.

Definition 3 (Equivalence Relation) *A relation* R *on a set* S *is said to be an equivalence relation if and only if it is reflexive, symmetric and transitive. That is, for all* x, y *and* z *in* S:

1. $x \sim x$ *(Reflexive)*

2. if $x \sim y$ *then* $y \sim x$ *(Symmetric)*

3. if $x \sim y$ *and* $y \sim z$ *then* $x \sim z$ *(Transitive).*

Two elements x and y which are equivalent given an equivalence relation R are denoted by either "$x \sim y$" or "$x \sim_R y$". The equivalence class of x on equivalence relation R of a set S can be denoted as $[x] := \{x' \in S | x \sim_R x'\}$.

2.1.3 Information Theoretical Measurements

Non-interference, i.e. the absence of interference, provides a strict security policy for determining a program to be well behaved. From the example depicted in Figure 2.2, it suggests that it is hard to hold the non-interference policy. As a matter of fact, the presence of interference considered as information being leaked generally happens when the interference reaches a threshold.

Thus instead of asking *"does a program leak information"*, one prefers to ask *"how much information does the program leak"*. This kind of question transfers the assessment of information-flow leaks in programs from qualitative to quantitative analysis. It may also need to compare two programs: "which program P or P' leaks more confidential information". Information theoretical measurements provide solutions for these questions, including information theory and probability theory.

Discrete Probability Theory

Discrete probability theory is introduced in this section. It is the mathematical foundation of reasoning about probabilities of events, which solves questions like "how likely is it that an event happens". Probability theory also provides the foundation of information theory. Details of probability theory are provided in standard textbooks, such as [55].

Definition 4 (Probability Distribution) *Given a discrete random variable* $X : \Omega \to N$, *the probability distribution of* X *is a list of probabilities* $p(x)$ *associated each of its possible value* $x \in N$. *It is denoted by a function:* $p : N \to [0, 1]$ *such that*

$$\sum_{x \in N} p(x) = 1,$$

where the probability $p(x)$ *or* $p(X = x)$ *is to measure the likelihood that* X *takes the value* x.

A probability distribution is also known as a probability mass function for the discrete random variable.

Definition 5 (Discrete Random Variable) *A discrete random variable is a function $X : \Omega \rightarrow N(N \subseteq \mathbb{R})$ from the set of possible outcomes Ω to a set N with unique numerical values.* In this thesis, a discrete random variable is abbreviated as the random variable.

Definition 6 (Joint Probability Distribution) *The joint probability distribution of two random variables X and Y is defined as*

$$p(X = x, Y = y) = p(x, y) = p(y|x)p(x) = p(x|y)p(y)$$

such that

$$\sum_{x \in X} \sum_{y \in Y} p(X = x, Y = y) = 1$$

$p(x|y)$ is the conditional probability of x given the knowledge of $Y = y$.

Joint probability measures the likelihood of $X = x$ and $Y = y$ occurred together.

Definition 7 (Conditional Probability) *Given two random variables X and Y, the conditional probability of an event $X = x$ given the knowledge of another event $Y = y$ is defined as:*

$$p(X = x|Y = y) = p(x|y) = \frac{p(x, y)}{p(y)}$$

Information Theory

Information theory provides an answer for questions like "how much information is leaked in the program P" [47]. One of the main concepts of information theory is entropy, also called as Shannon entropy [108]. It quantifies the amount of information uncertainty in a random variable.

Definition 8 (Shannon Entropy) *Given a random variable $X : \Omega \rightarrow N$ with the probability distribution $P = (p(x_i))_{x_i \in N}$, Shannon entropy is defined as:*

$$H(X) = -\sum_{x_i \in N} p(x_i) \log p(x_i),$$

The logarithm is to the base 2. Shannon entropy measures the amount of information, or the uncertainty that the random variable contains.

The entropy is bounded by $H(X) \geq 0$, where an extreme case $H(X) = 0$ holds when an event x occurs with a probability $p(x) = 1$. No uncertainty of the variable exists since there is only one event with a probability of 1.

On the other hand, the entropy is maximal when the probabilities of events are uniformly distributed, i.e. every event is likely to happen evenly. In this case, $H(X) = \log |N|$, where $|N|$ is the cardinality of the space of the events.

Proposition 1 *Given a random variable $X : \Omega \rightarrow N$ with the probability distribution $P = (p(x_i))_{x_i \in N}$, the boundary of the Shannon entropy of X is:*

$$0 \leq H(X) \leq \log |N|,$$

where $|N|$ is the cardinality of the set X.

Definition 9 (Joint Entropy) *Given two random variables X and Y, the joint Shannon entropy of X and Y is defined as:*

$$H(X, Y) = - \sum_{x \in N_X} \sum_{y \in N_Y} p(X = x, Y = y) \log p(X = x, Y = y),$$

where $p(X = x, Y = y)$ is the joint probability of X and Y.

The joint entropy measures how much uncertainty there is when two random variables $(X$, $Y)$ are happening at the same time, i.e. joint.

Definition 10 (Conditional Entropy) *Given two random variables X and Y, the conditional entropy of X given the knowledge of variable Y is defined as follows:*

$$H(X|Y) = \sum_{y \in N_Y} p(y) H(X|Y = y)$$

The conditional entropy $H(X|Y)$ refers to the average uncertainty of X conditional on the value of Y, averaged over all possible values of Y. It is different from the one $H(X|Y = y)$, which is the entropy of X conditioning on a particular value y that Y takes.

The conditional entropy of X given the knowledge of Y can also be defined in terms of the chain rule as:

$$H(X|Y) = H(X, Y) - H(Y)$$

The chain rule can be proved as follows:

$$\begin{aligned}
H(X|Y) &= \sum_{y \in N_Y} p(y) H(X|Y = y) \\
&= \sum_{y \in N_Y} p(y)[- \sum_{x \in N_X} p(x|y) \log p(x|y)] \\
&= - \sum_{x \in N_X} \sum_{y \in N_Y} p(x, y) \log p(x|y) \\
&= - \sum_{x \in N_X} \sum_{y \in N_Y} p(x, y) \log \frac{p(x, y)}{p(y)} \\
&= - \sum_{x \in N_X} \sum_{y \in N_Y} p(x, y) \log p(x, y) + \sum_{x \in N_X} \sum_{y \in N_Y} p(x, y) \log p(y) \\
&= H(X, Y) + \sum_{y \in N_Y} p(y) \log p(y) \\
&= H(X, Y) - H(Y)
\end{aligned}$$

The higher correlation between X and Y, the low uncertainty $H(X|Y)$ contains. If X is a function of Y, i.e. the value of X is completely dependent on Y, then $H(X|Y) = 0$ as there is no uncertainty of X when giving the knowledge of Y. The other extreme case is that if X and Y are independent then knowing the value of Y does not change the uncertainty of X, which indicates $H(X|Y) = H(X)$.

Definition 11 (Mutual Information) *Given two random variables X and Y, the mutual information of X and Y is defined as:*

$$I(X;Y) = \sum_{x \in N_X} \sum_{y \in N_Y} p(x, y) \log(\frac{p(x, y)}{p(x)p(y)})$$

Mutual information is a measure of how much information that random variables X and Y share. It can be rewritten in terms of conditional entropy as:

$$\begin{aligned}
I(X;Y) &= \sum_x \sum_y p(x, y) \log(\frac{p(x, y)}{p(x)p(y)}) \\
&= - \sum_x \sum_y p(x, y) \log p(x) + \sum_x \sum_y p(x, y) \log p(x|y) \\
&= H(X) - H(X|Y)
\end{aligned}$$

Similarly, it turns out that mutual information can also be

$$I(X;Y) = I(Y;X) = H(Y) - H(Y|X)$$

Entropy gives an average value of information leakage, but Smith [109] proposes a different concept, the concept of *vulnerability* which is a worst-case measurement, the probability that a secret can be guessed correctly.

Definition 12 (Vulnerability) *Given a random variable $X : \Omega \to N$ with the probability distribution $P = (p(x_i))_{x_i \in N}$, the vulnerability of X, denoted by $V(X)$, is defined by*

$$V(X) = \max_{x_i \in N} P(X = x_i)$$

The vulnerability $V(X)$ is the worst-case probability of the value of X being guessed correctly in one try.

Definition 13 (Min Entropy) *Given a random variable X, the min entropy of X, denoted by $H_\infty(X)$, is given by*

$$H_\infty(X) = \log \frac{1}{V(X)}$$

where $V(X)$ is the vulnerability of X.

The min entropy, it turns out, converts the probability to an entropy measure.

Notice that if X is uniformly distributed among n values, $V(X) = \frac{1}{n}$ and $H_\infty(X) = \log n$. Thus the equality of Shannon entropy and min entropy holds on uniform distribution.

To measure the conditional entropy of X given the knowledge of Y of min entropy $H_\infty(X|Y)$, first consider the *conditional vulnerability* about the probability of guessing X correctly in one try given Y.

Definition 14 (Conditional Vulnerability) *Given random variables X and Y, the conditional vulnerability of X given the knowledge of Y is*

$$V(X|Y) = \sum_{y \in N_Y} P(Y = y)V(X|Y = y)$$

where

$$V(X|Y = y) = \max_{x \in N_X} P(X = x|Y = y)$$

Definition 15 (Conditional Min Entropy) *Given two random variables X and Y, the conditional min entropy of X given Y is defined by*

$$H_\infty(X|Y) = \log \frac{1}{V(X|Y)}$$

Expected Probability of Guessing

Probabilities can also be used to ask questions like "how likely a secret can be guessed correctly in n tries". The average probability of guessing a secret correctly in n tries is defined in [82], related to Smith's conditional vulnerability [109].

Definition 16 (Expected Probability of Guessing) *Given a random variable $X : \Omega \to N$ and the probability distribution $P = (p(x_i))_{x_i \in N}$, where the probabilities are* *ordered decreasingly i.e.* $p(x_i) \geq p(x_{i+1})$*, the probabilities of guessing the secret in n tries is defined as*

$$g_{n,P}(X) = \sum_{1 \leq i \leq m} p(x_i)$$

where $m = \min(|X|, n)$*.*

Given a partition Y of a set X and a distribution P on X the probability of guessing the secret in n tries is

$$G_{n,P}(Y) = \sum_{Y_i \in Y} g_{n,P}(Y_i)$$

This definition can also be regarded, in terms of the probabilities in each block of the partition Y, as

$$G_{n,P}(Y) = \sum_{Y_i \in Y} g_{n,P}(Y_i) = \sum_{Y_i \in Y} p(Y_i) \sum_{1 \leq j \leq m, x_j \in Y_i} \frac{p(x_j)}{p(Y_i)}$$

where $m = \min(|Y_i|, n)$. When $n = 1$, the definition becomes

$$G_{1,P}(Y) = \sum_{Y_i \in Y} p(Y_i) \frac{p(x_i)}{p(Y_i)}$$

where $p(x_i)$ is the highest probability in a block Y_i, and so $G_{1,P}(Y)$, abbreviated by $G(Y)$, is related to the conditional vulnerability. The expected probability of guessing $G(Y)$ is also regarded as *guessing probability* in this thesis.

2.1.4 Measuring Leakage

In the context of quantitative information flow, information leaked in terms of entropy is quantified by

$$information\ leaked = initial\ uncertainty - remaining\ uncertainty$$

Initial uncertainty specifies the original entropy of confidential information before observations. And remaining uncertainty means the amount of information the secret remains after observations.

Given a random variable X, the entropy $H(X)$ measures the initial uncertainty of the secret in X. The remaining uncertainty of X after observing the output O is, by definition, the conditional entropy $H(X|O)$. Accordingly, information leaked of X given the observation O can be calculated by

$$\Delta_H(X) = H(X) - H(X|O) = I(X; O)$$

Mutual information, i.e. the information shared between X and O, turns out to be the information leaked.

In terms of min entropy, information leaked can be measured by

$$\Delta_{H_\infty}(X) = H_\infty(X) - H_\infty(X|O) = I_\infty(X;O)$$

where $I_\infty(X;O)$ is the information share between X and O on min entropy.

Either Shannon entropy $H(X)$ or min entropy $H_\infty(X)$ can be used to measure information leaked. Consider

$$F = \{H,\ H_\infty\}$$

which denotes that either one measure is chosen between H and H_∞ to calculate the information uncertainty. As a result, the information leaked of X can be expressed by

$$\Delta_F(X) = F(X) - F(X|O)$$

In the case of non-interference, the confidential variable X is independent on the observation O, which means $F(X|O) = F(X)$. Therefore there is no information leaked in non-interference as $\Delta_F(X) = 0$.

2.2 Side-Channel Leakages

In the context of *secure information flow*, if the question is about how much information a program could leak from its high inputs to the low inputs, then in the context of side-channel leakage, the question becomes like how much information about the secret can be leaked from side channels in the information systems.

In cryptography, a side-channel attack is based on the information gained from physical implementations of a cryptosystem. For example, timing information, power consumption, electromagnetic leaks or even sound can provide extra sources of encrypted information, which can break the cryptosystem.

2.2.1 Side-Channel Attacks in Web Applications

With the rise of web 2.0 applications, there has been an increasingly concern about side-channel attacks in web applications. Users can benefit from the interaction and the collaboration with each other in web 2.0, however, a subset of internal information flows of applications are inevitably exposed in the network, which increase the risks of revealing user privacy. Primary side-channel attacks in web applications include timing attacks and traffic-analysis attacks.

In timing attacks, Felten and Schneider [56] first propose a cache-based timing attack. Using the response time of accessing to a web page, it can determine whether a web page has been visited beforehand. In general, web browsers use caching to save copies of recently-accessed files to reduce perceived access time when future accesses to those files stored locally occur. Exploiting this feature of caching, an attacker is capable of measuring the required access time to determine whether a file has been accessed recently if the required time is obviously lesser than the normal time required to access a file.

Moreover, side-channel attacks have also been performed by analysing web traffic. In the following, some particular examples are taken to give an overview of this security threat.

Side-channel analysis of network traffic is also deemed as ***traffic analysis*** throughout this thesis. Confidential information of web users can be revealed even in encrypted channels through the low-level features of web traffic, such as packet sizes, packet lengths, packet directions etc. This concern was first proposed by Wagner and Schneier [119] in 1996. Two years later an actual side-channel attack through web traffic was demonstrated by Cheng et al. [40].

We take two examples to give an idea of traffic analysis against browsing histories and user sensitive inputs in a text box.

Figure 2.3 gives an example of fingerprinting web pages in a website. A user can follow one of the execution paths from the homepage, each to a unique web page.

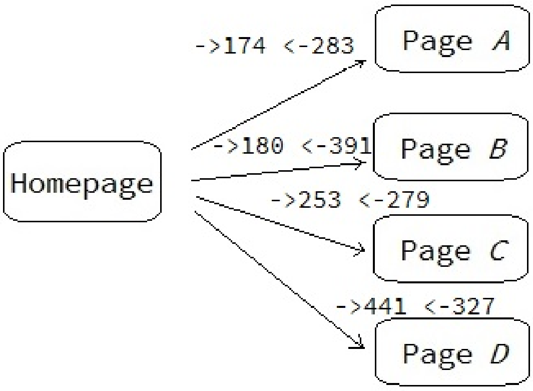


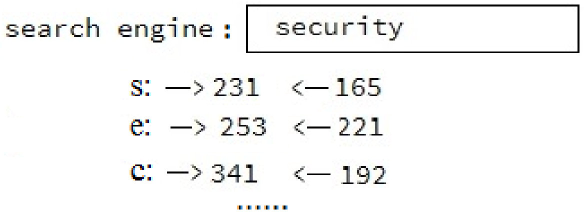
Figure 2.3: A side-channel attack against web pages

A sequence like "→ 174 ← 283" is a traffic sequence generated during a communication. Symbols "→" and "←" represent the packet directions, indicating request and response traffic respectively, whereas the numerical values specify the packet sizes. As shown in Figure 2.3, traffic patterns for each communication are distinguishable from each another. Thus an attacker can easily identify the web page a user accesses by matching the observed traffic sequence to one of the traffic patterns, each of which associates with a communication with one web page.

Image another case in this example. With the countermeasures like padding, the traffic sequences can be indistinguishable among different requests of web pages. In this case, the attacker can also infer the requested web page if the following requests generate distinguishable web traffic.

Figure 2.4 shows a scenario of inputting a search keyword in a search engine. In this example, each character input triggers a generation of web traffic, which can be observed by an eavesdropper.

To infer the first letter of a search word, an attacker matches observed traffic to a set of at most 26 possibilities (when the unique traffic is generated by each letter from



```
search engine : security

s: —> 231  <—165
e: —> 253  <—221
c: —> 341  <— 192
        ......
```

Figure 2.4: A side-channel attack against user search input

letter a to z). From the second letter, the observed traffic is matched to a smaller space of possibilities, because the number of letters, which can form a meaningful word by following previous letter, reduce drastically.

The above examples give an overview of side-channel attacks using web traffic. In summary, the public features of web traffic, such as packet sizes and directions, can leak user privacy in encrypted communications, when the web traffic varies dependently on user confidential information.

2.3 Struts Framework

Struts is a framework of Java web applications. It combines *JavaServer Pages* (JSPs), xml configuration files and Java *servlets* files to build a web application. Struts have two versions, i.e. Struts 1 and Struts 2, which are developed on top of a design of *Model-View-Controller* (MVC) [17]. Figure 2.5 gives an overview of the Struts 1 framework in MVC model, which is similar to the Struts 2.

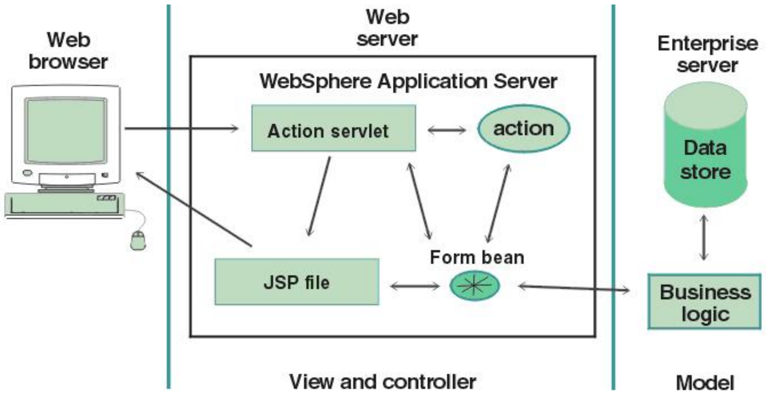


Figure 2.5: Structure of Struts 1 in MVC [19]

In the MVC model, a client sends a request to a web server. The Action servlet, which runs as a controller within the web server, receives the request and invokes a model, which is used to maintain data and can be invoked through a form bean in Struts 1. Then the servlet invokes the corresponding action to process the request depending on the data returned from the model, and communicates with the corresponding JSP page (as the viewer), which then will be rendered back to the client.

Figure 2.6 illustrates the communicating process in Struts 1, with respect to web files. A file `web.xml` is the *deployment descriptor file* in each web application to initialise

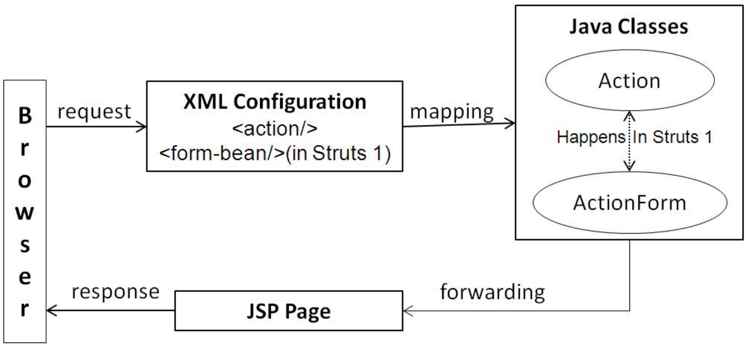


Figure 2.6: Working flow of Struts 1 web application

the Struts framework. It specifies the starting page of a web application, and how the framework should behave. A *Struts configuration file*, named as `struts-configure.xml` in Struts 1 or `struts.xml` in Struts 2, specifies the `Action` and `ActionForm`[1] classes. In Struts 1, each user action from the client side is related to an Action class and an ActionForm class. When a browser submits a request to the server, usually through a HTML form, the Action servlet associates the user action to its Action and ActionForm classes, which actually determine the flow of page transitions.

In Struts 1, user inputs in a user action are retrieved by the related ActionForm class. Then the Action class determines the returned branch based on the retrieved values, by returning an `ActionForward` string to the servlet. The `ActionForward` string tells the servlet that which JSP page should be connected. So then the servlet invokes the JSP page to respond the client. More details about Struts can be referenced in [16].

2.4 Program Analysis

Program analysis offers techniques for automatically analysing behaviours arising at a run-time when executing a computer program, in terms of a property such as safety, correctness, robustness, and liveness [89].

Program analysis can be divided into *static program analysis* or *dynamic program analysis*, depending on whether it is performed without executing the program or during the runtime. The analysis can also be performed in a combination of the two techniques.

2.4.1 Static Program Analysis

Static program analysis or static code analysis is a process that is performed without actually executing programs. In most cases the analysis is performed on the source code of programs; whereas sometimes it is performed on the object code, usually a machine code language, i.e. binaries. In general, static analysis examines all possible execution paths and variable values, in which source code and object code is analysed for quality, safety, and security.

[1] ActionForm classes exist only in Struts 1. In Struts 2 they are integrated into the Action classes.

Data Flow Analysis

Data flow analysis is a technique of static analysis, which attempts to gather information about the possible set of values used at each node in a program [89].

It is customary to think of a program as a *control flow graph* (CFG): the nodes are the elementary blocks and the edges describe how control might pass from one elementary block to another one.

A simple approach to perform data flow analysis is to set up an equation system of a set of data flow equations about inputs and outputs for each elementary block of the CFG and repeatedly calculate the outputs from the inputs at each node until the whole system stabilizes, i.e. when it reaches a *fixpoint*.

Basic principle. The basic principle of an equation system of a CFG is:

$$\begin{aligned}&\forall \text{ elementary block } eb \in CFG:\\ &\quad exit_{eb} = trans_{eb}(entry_{eb})\\ &\quad entry_{eb} = join_{p \in pred_{eb}}(exit_p)\end{aligned}$$

where $exit_{eb}$ and $entry_{eb}$ is the output and input equations of elementary block eb. The transfer function $trans_{eb}$ is the mapping from the inputs to the outputs in block eb. It executes operations on the entry state $entry_{eb}$ and yields the exit state $exit_{eb}$. The join operation $join$ combines the exit states of the predecessors of block eb, yielding the entry state of eb.

Control Flow Analysis

Control flow analysis is a static code analysis for determining the control flow of a program. It determines the receiver(s) of function calls in a program. More specifically, it determines which functions have been invoked in another function. For many imperative programming languages, the control flow of a program is directly available, e.g. the simple WHILE language. However, it is not a case for more advanced *imperative*, *high-order* and *object-oriented* languages. It is difficult to immediately point to the information what parameters a function will be called with.

For example, in a programming language with higher-order functions, the target of a function call may not be explicit: in the isolated expression (λ `(f)` `(f x)`), it is unclear to which procedure `f` may refer. To determine the possible targets, a control flow analysis must consider where this expression could be invoked, and what argument it may receive.

Therefore, control flow analysis is required, providing that: for each function application, which functions may be applied [89].

Readers interested in more details and examples of control flow analysis can be referred to the textbook [89].

2.4.2 Dynamic Program Analysis

In contrast to static program analysis without executing a program, dynamic program analysis examines a program during runtime.

While dynamic analysis cannot prove that a program satisfies a particular property, it can detect violations of properties as well as provide useful information to programmers about the behaviours of their programs [26].

Dynamic program analysis typically involves instrumenting a program to examine or record certain aspects of its run-time state. This instrumentation can be tuned to collect precisely the information needed to address a particular problem.

Dynamic program analysis is dependent on the inputs. It relates program inputs and outputs to program behaviour [99]. To be effective in dynamic analysis, sufficient inputs should be executed.

It can also detect dependencies which are not detected in static analysis. For example, dynamic analysis ensures that the program being tested is compatible with other programs.

2.4.3 White-Box and Black-Box Testing

Program testing methods are traditionally divided into white-box and black-box testing. They are divided from a point of view that a test engineer takes when designing test cases.

White-box testing examines internal structures or workings of a program at the level of the source code. A tester must have explicit knowledge of internal workings of the items being tested, to know what kinds of test cases should be created [71]. The tester creates test cases which are considered important.

Though both white-box testing and static analysis are based on the source code of a program, they are different things.

Static analysis uses techniques such as data flow and control flow analysis to minimise errors in the source code by executing every path of the source code. It does analyse the source code independently on test cases.

Relatively, white-box testing looks for problems in a different way: by examining programs dependently on test cases. Static analysis techniques such as control flow and data flow analysis can be used in the building phase of white-box testing, which analyses the source code and help testers build test cases more precisely. Though white-box testing can also uncover defects in the source code, it typically aims to examine outputs of particular test cases.

Unlike white-box testing, specific knowledge of the source code or internal structures is not required in black-box testing [94]. Black-box testing examines the functionality of a program. The tester is aware of what the program is supposed to do, i.e. knows the inputs and what the expected outcomes should be. However, the tester has no knowledge of how the program reaches the outputs.

Test cases are mainly derived from external descriptions of the program, including specifications, requirements and design parameters. The tester selects both valid and invalid inputs and determines the correct outputs.

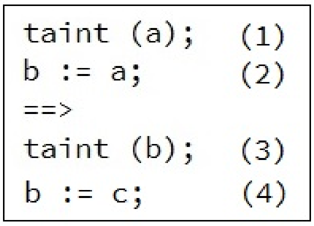
2.4.4 Taint Analysis

Taint analysis is a technique to track information flow of *tainted information* from *sources* to *sinks.* It aims to examine that any variable which can be modified by an outside user poses a potential security risk.

It runs a program and observes which computations are affected by predefined taint sources such as user input [107]. It is designed to increase security by preventing malicious users from executing dangerous commands.

Taint analysis starts with a black list of tainted sources, which are the variables potentially influenced by outside user inputs. Other variables unlisted in the black list are considered as *untainted.* A taint policy is configured to determine how taint flows while a program executes, what sorts of operations introduce new taint variables, and how to perform checks on tainted values of sinks. For example, if a tainted variable is used in an assignment which sets a second variable, the second variable is also tainted.

Taint analysis attempts to identify the tainted variables with user controllable inputs and tracks flows of them to possible vulnerable functions or commands which are known as *sinks.* And a tainted variable is *sanitized* when it is completely dependent on an untainted variable. If a tainted variable gets passed to a sink without being *sanitized*, a warning of the program with a potential dangerous tainted variable is given and the dangerous command is flagged as a vulnerability. Take the following as an example.



```
taint (a);   (1)
b := a;      (2)
==>
taint (b);   (3)
b := c;      (4)
```

Figure 2.7: Propagation of a tainted variable a

Figure 2.7 depicts the propagation of a tainted variable a. Variable a is tainted as displayed at line (1). Variable b is tainted when it is assigned by the tainted variable a at line (2). Assume that value c is untainted and it sets variable b at line (4). Thus variable b is sanitized in the end.

2.5 Symbolic Execution and Symbolic PathFinder

Symbolic execution (SE) is a program analysis technique to determine what inputs lead to the execution of each part of a program [72]. Instead of using concrete inputs to execute programs, symbolic execution analyses programs with unspecific inputs, by using symbolic values as inputs.

Symbolic execution builds a *path constraint pc*, also called as a *path condition*, for each conditional path. When the execution encounters a conditional statement, e.g. an `if` statement on a path condition c, it flows to one of two paths: either (1) branch `then` when it is *true* for condition c or (2) branch `else` when it is *false.* Thus the path constraint in terms of the `then` branch updates to $pc_1 = pc \wedge c$, while for the `else` branch $pc_2 = pc \wedge \neg c$, where pc is the path constraint of the path before arriving at the `if` statement.

Symbolic execution finds path constraints for each feasible execution path and then finally generate specific test inputs by solving the path constraints.

2.6 Hidden Markov Model

Real-world programs generally produce observable outputs which can be characterized as signals. A fundamental interest is to characterise such signals and to learn the signal source which generates the signals. A signal model is built to characterise properties of signals.

Hidden Markov Model (HMM) is such a signal model first published in a series of classic papers by Baum et al. [27, 28, 29, 30] in the late 1960s and early 1970s. It is a statistical Markov model in which the system modelled is assumed to be a Markov process with unobserved (hidden) states. From 1980s, HMM has been widespread understood and largely used in the field of speech processing, and nowadays, it has been applied in a large scale of areas, such as bioinformatics [54], and signature recognition [38]. Details of HMMs are introduced in the work by Rabiner [97]. The following refers to the notations described in [97] to denote a HMM.

Formally, a hidden Markov model can be denoted as follows:

T = length of an observation sequence

N = number of states in the model

$ST = \{ST_1, ST_2, ..., ST_N\}$ is individual states, and state at time t denoted by q_t

M = number of distinct observation symbols per state

$V = \{v_1, v_2, ..., v_M\}$ is individual symbols of observations

$C = \{c_{ij}\}$ is the state transition probability distribution, where

$$c_{ij} = P(q_{t+1} = ST_j | q_t = ST_i), \quad 1 \leq i, j \leq N$$

is the probability of transiting from ST_i at time t to ST_j at time $t + 1$.

$B = \{b_j(k)\}$ is the observation probability distribution in state ST_j, where

$$b_j(k) = P(v_k | q_t = ST_j), \quad 1 \leq j \leq N \quad 1 \leq k \leq M$$

is the probability of observing v_k under state ST_j at time t.

$\Pi = \{\pi_i\}$ is the initial state distribution where

$$\pi_i = P(q_1 = ST_i), \quad 1 \leq i \leq N$$

For convenience, the above HMM can be denoted by a set of parameters

$$\lambda = \{\Pi,\ C,\ B\}$$

Traditionally, HMM is used for solving three classic problems in the following: Given an observation sequence: $O = O_1\, O_2 \cdots O_T$, and a model: $\lambda = \{\Pi,\ C,\ B\}$

- Question 1: how to efficiently compute $P(O|\lambda)$, i.e. the probability of the observation sequence, given the model?
- Question 2: how to adjust the model parameters $\lambda = \{\Pi,\ C,\ B\}$ to maximize $P(O|\lambda)$?
- Question 3: how to generate a corresponding state sequence $Q = q_1\, q_2 \cdots q_T$, which is the optimal to best explain the observations (i.e., the signal source)?

Forward-Backward algorithm [28], and *Veterbi* algorithm [58, 118] generally provide answers for these questions. Thorough explanations of the answers can be found in [97].

In Chapter 4, we use an analysis motivated by HMMs. We construct a traffic pattern from the packet data observed and use these notations to model our data. The details will be shown in Chapter 4.

2.7 Fingerprinting Model and Threat Scenario

2.7.1 Fingerprinting Model

This section describes a basic fingerprinting model of traffic analysis in web applications, which is essentially established by previous work, e.g. [24].

Formally the fingerprinting model is given by a transition system

$$TS = (G, O, A, f, S).$$

A web application is modelled as a directed graph $G = (r, N, E)$, where N is the set of nodes and $r \in N$ is the root node; each node $n \in N$ represents a web page in the web application.

An edge from one node to the next node is a transition between web pages $e = u \xrightarrow{a} v \in E$ or $e = (u,\ a,\ v)$, where $u, v \in N$. An edge e indicates that node v can be reached from node u by performing user action a. Node u is regarded as a parent node, and node v is the child of node u. Child node v is also deemed as a *forwarding node* and the corresponding web page as a *forwarding page*. An execution path is a sequence of transitions.

A user action $a \in A$ is defined as a sequence of pairs, one of $(element, value)$, where *element* is a component, e.g. a list box, on a web page and *value* is the specific value chosen in this component. For example, an action of logging into a user account consists of three pairs: one for the user name, one for the password and the other for the *submit* button with a value of *null*.

In this thesis, the terms *component* and *widget* are used interchangeably.

Assume that transitions from child nodes to parent nodes are only triggered through components such as links on web pages. This means that, in this research, the only way going back to previous pages in an execution path is using widgets on the web pages, rather than using the *"go back previous page"* buttons on the client browsers.

O is a set of observations about web traffic observed in communications. Each observation is a sequence of directional packet sizes describing request and response packets in communications. For example, a pair $(\rightarrow 528, \leftarrow 620)$ indicates a transition happening with a 528-byte packet sent from the browser and a 620-byte packet sent back from the server. In this thesis, packets sent from a client side are regarded as *request packets* while those from a server side are the *response packets*. Comparing observations between different execution paths, if they are distinguishable then the secret may be leaked.

A function $f : E^* \rightarrow O^*$ maps sequences of edges to the corresponding observations of network traffic, i.e. the execution paths to their observations.

Set S is a set of sensitive information for a secret. An item $s \in S$ is a value for the secret. It is denoted by a pair (*element*, *value*), where *element* is the component the secret is located and *value* is the specific input for the secret.

This fingerprinting model is the foundation of future models developed throughout this thesis.

2.7.2 Threat Scenario

Consider a common threat scenario used in traffic analysis in web applications. It is essentially the same threat scenario considered by other authors, see e.g. [39, 37].

A user first performs a sequence of user actions, i.e. an execution path. An passive attacker eavesdrops on the network, to extract packet sizes and packet directions of the web traffic for a communication. The attacker aims to uncover sensitive user information, e.g. web activities, from the observed traffic features.

Our work in Chapters 3, 4 and 5 analyses side-channel leakages in web applications from a perspective of a developer, instead of from an attacker's perspective. More specifically, it focuses on detecting side-channel vulnerabilities in web applications, to provide references and suggestions about the vulnerabilities in web applications to developers, rather than analysing an attacker's capability of compromising user privacy in a web application.

Further threat scenarios proposed in the following chapters are based on this general scenario.

Chapter 3

Side-Channel Vulnerabilities in Java Web Applications

3.1 Motivation and Overview

Our earliest work in this thesis proposes an automated system of analysing side-channel vulnerabilities in Struts-based web applications [16].

3.1.1 Motivation

In a previous work [39], Chen et al. show a surprising finding concerning side-channel leakage of sensitive information in high-profile, top-of-the-line web applications: an eavesdropper who eavesdrops the packet sizes and directions can infer user medications, annual family incomes, investment choices and user query words, even when the traffic is encrypted.

Afterwards, Sidebuster [125] is proposed, which is the first tool for detecting and quantifying side-channel leakage of sensitive information in web applications. It performs taint analysis on the source code to identify web components which involve sensitive information. The developer manually specifies execution paths containing those components. Execution paths then will be tested to evaluate leakage.

Moreover, Chapman and Evans [37] propose an automated black-box detection of side-channel vulnerabilities in web applications. Their approach does not need to access to the source code. However, similar to Sidebuster, it needs manual configuration of test cases of execution paths.

Motivated by such an inconvenience of manual generation of test cases in pioneer work, this work aims to achieve a completed automation of analysing side-channel vulnerabilities in web applications, i.e., containing an automated test case generation. Motivated by the approach proposed in [113], we propose a novel approach to automatically generate test cases for Struts-based web applications. The automated system aims to assist developers in detecting side-channel vulnerabilities. The main part of this chapter is published on [66].

Contributions. This chapter introduces a suite of new techniques to quantify side-channel leakage in web applications. More specifically:

1. Advance towards fully automated test case generation in the context of side-channel analysis of web applications.

2. Novel approach of automated generation of test cases in Struts-based web application. Static analysis and symbolic execution [72] are used to achieve automated test case generation.

3. Configurability to handle different probability distributions of user secret values, including uniform distribution on test cases, uniform distribution on user action and in terms of PageRank [34].

4. Implementations and evaluations on a variety of Struts-based web applications, which work well.

3.1.2 Overview

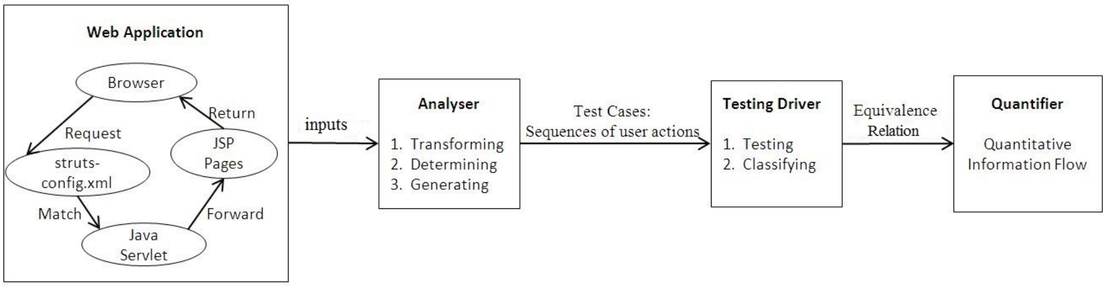


Figure 3.1: Overview of the automated system–SideAuto

Figure 3.1 gives an overview of the automated system. Given a Java web application built on Struts framework, the system analyses the structure by performing static analysis and determines path constraints by performing symbolic execution. Combining the path constraints with the structure, feasible test cases of execution paths can be derived. Each test case is then executed and we use Jpcap [10] to capture the web traffic. By building an equivalence relation on the collected traffic sequences, side-channel leaks are quantified using quantitative information flow.

Techniques are implemented into a tool called *SideAuto*, which is capable of analysing both Struts 1 and 2 web applications [16]. In this chapter, a web application upon Struts 1 framework is taken as an example to clarify how SideAuto works.

3.1.3 Model

Based on the fingerprinting model described in section 2.7, this chapter updates the transition system, defined by: $TS = (G, O, A, f, S, l)$.

A new denotation l is defined. It symbolises a loop bound. This chapter considers loops when a web page can be accessed more than once in an execution path. Hence a loop bound l is defined to limit the number of times a web page is allowed to be repeatedly visited in an execution path.

Other letters in this transition system TS are same as the one defined in section 2.7.

3.1.4 Threat Scenarios

In this chapter, we consider two threat scenarios: a passive attacker wants to (1) identify the sequence of user actions or to (2) infer the information related to a user input in a widget which accepts arbitrary values, e.g. the text box.

Now we precise the threat scenarios:

Scenario 1: Identifying user action sequence

To address this kind of threat, the values of user actions are restricted to be a predefined set of constants, e.g. a choice in a menu widget.

In this scenario, more specifically, the secret is a sequence of user actions, i.e. an execution path. SideAuto generates test cases for each execution path and determines an execution path through web traffic.

Scenario 2: Identifying information about user inputs

This scenario considers instead arbitrary values in user actions, e.g. strings in text boxes.

This scenario widens the secrets examined, including the information which relates to user inputs, but not the values themselves. For example, though an attacker cannot get a password of a user account, the disclosure of the password length can also be useful. As it will narrow down the set of passwords, leading to a higher probability of guessing the password correctly.

In this scenario, the developer needs to define specific values in testing. Though it is hardly to cover all the possible user inputs for components, such as text boxes, the developer can specify all the potential user inputs he wants to examine. More details about the completeness of test cases will be discussed later.

Now let us see how SideAuto constructs the structure graph of a web application.

3.2 Construction of a Web Application's Structure

The structure graph of a web application is determined by the web pages and the flow of web pages which depends on user inputs. Parsing the deployment descriptor file–web.xml, SideAuto starts the analysis from the starting page. Compiling each JSP page invoked to individual Java classes, the structure graph can be constructed by analysing the Java classes.

3.2.1 An Example

First consider an example about a university identity system as shown in Figure 3.2.

An execution path starts from a user input selected on page *index.jsp*, either *student* or *professor*. Observing the web traffic generated, an eavesdropper can be able to identify the user's selection if the web traffic between the two selections is different.

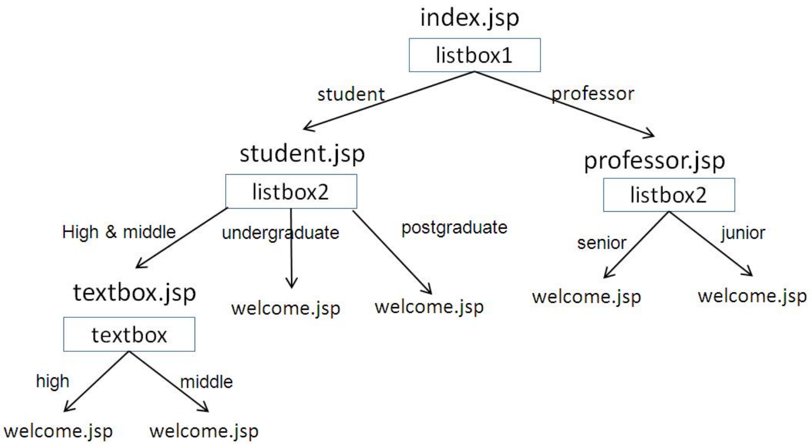


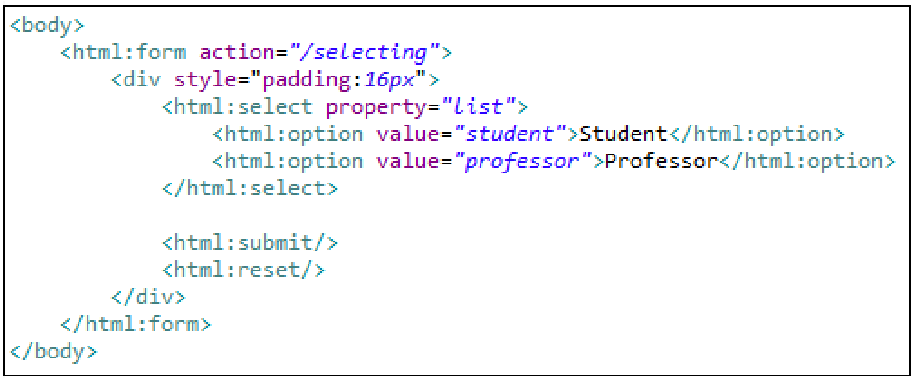
Figure 3.2: An example of a university identity system

This example will be used to clarify how SideAuto works in the following sections of this chapter.

3.2.2 Compiling JSP Pages to Java classes

As we cannot find a tool which can analyse JSP files directly, we first compile a JSP file to a Java class and then analyse the Java class to obtain the structure of the JSP page.

Jasper [8], built in SideAuto, is the JSP engine of Tomcat [2] to perform the compiling of JSP pages.



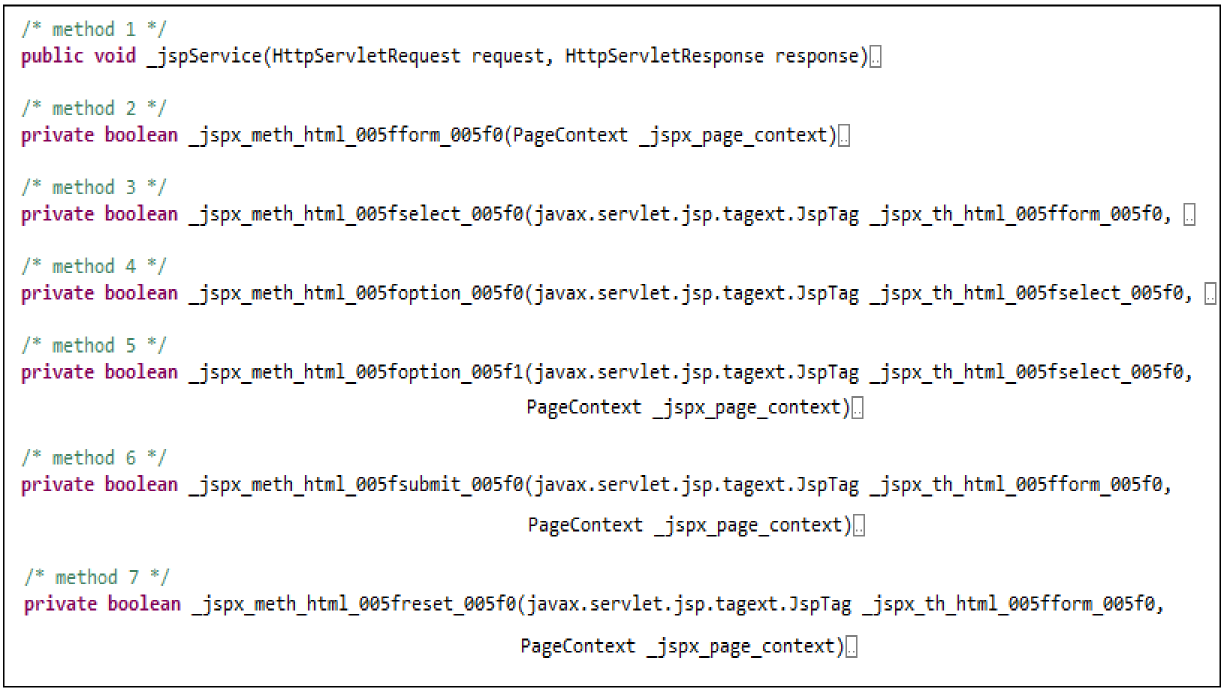
```
<body>
    <html:form action="/selecting">
        <div style="padding:16px">
            <html:select property="list">
                <html:option value="student">Student</html:option>
                <html:option value="professor">Professor</html:option>
            </html:select>

            <html:submit/>
            <html:reset/>
        </div>
    </html:form>
</body>
```

Figure 3.3: Main body of *index.jsp* in the university identity system

In a JSP page, web components are expressed using tag elements, such as <`html.form`>. Components such as buttons, text boxes, check boxes etc. are nested in a form component on a JSP page. Figure 3.3 shows a form element on page *index.jsp* in the university identity system in Figure 3.2.

When compiling a JSP page, each component is compiled to an individual method in the Java class. We call these methods as *component methods*. Figure 3.4 shows the methods generated for the form in Figure 3.3.

In Figure 3.4, method 1 is named as a `_jspService()` method, which exists in every Java class generated from a JSP page. Method 2 is a component method for the form



```
/* method 1 */
public void _jspService(HttpServletRequest request, HttpServletResponse response)

/* method 2 */
private boolean _jspx_meth_html_005fform_005f0(PageContext _jspx_page_context)

/* method 3 */
private boolean _jspx_meth_html_005fselect_005f0(javax.servlet.jsp.tagext.JspTag _jspx_th_html_005fform_005f0, 

/* method 4 */
private boolean _jspx_meth_html_005foption_005f0(javax.servlet.jsp.tagext.JspTag _jspx_th_html_005fselect_005f0, 

/* method 5 */
private boolean _jspx_meth_html_005foption_005f1(javax.servlet.jsp.tagext.JspTag _jspx_th_html_005fselect_005f0,
                                                PageContext _jspx_page_context)

/* method 6 */
private boolean _jspx_meth_html_005fsubmit_005f0(javax.servlet.jsp.tagext.JspTag _jspx_th_html_005fform_005f0,
                                                PageContext _jspx_page_context)

/* method 7 */
private boolean _jspx_meth_html_005freset_005f0(javax.servlet.jsp.tagext.JspTag _jspx_th_html_005fform_005f0,
                                               PageContext _jspx_page_context)
```

Figure 3.4: Component methods generated for the form in Figure 3.3

element <`html.form`> and method 3 for the select component <`html.select`>. Methods 4 and 5 are for components <`html.option`> nested in the select component. And methods 6 and 7 are for <`html.submit`/ > and <`html.reset`/ > respectively.

In a Java class, each component method is directly or indirectly invoked by the `_jspService()` method. Figures 3.5 and 3.6 display the invoking relations between these methods in Figure 3.4.

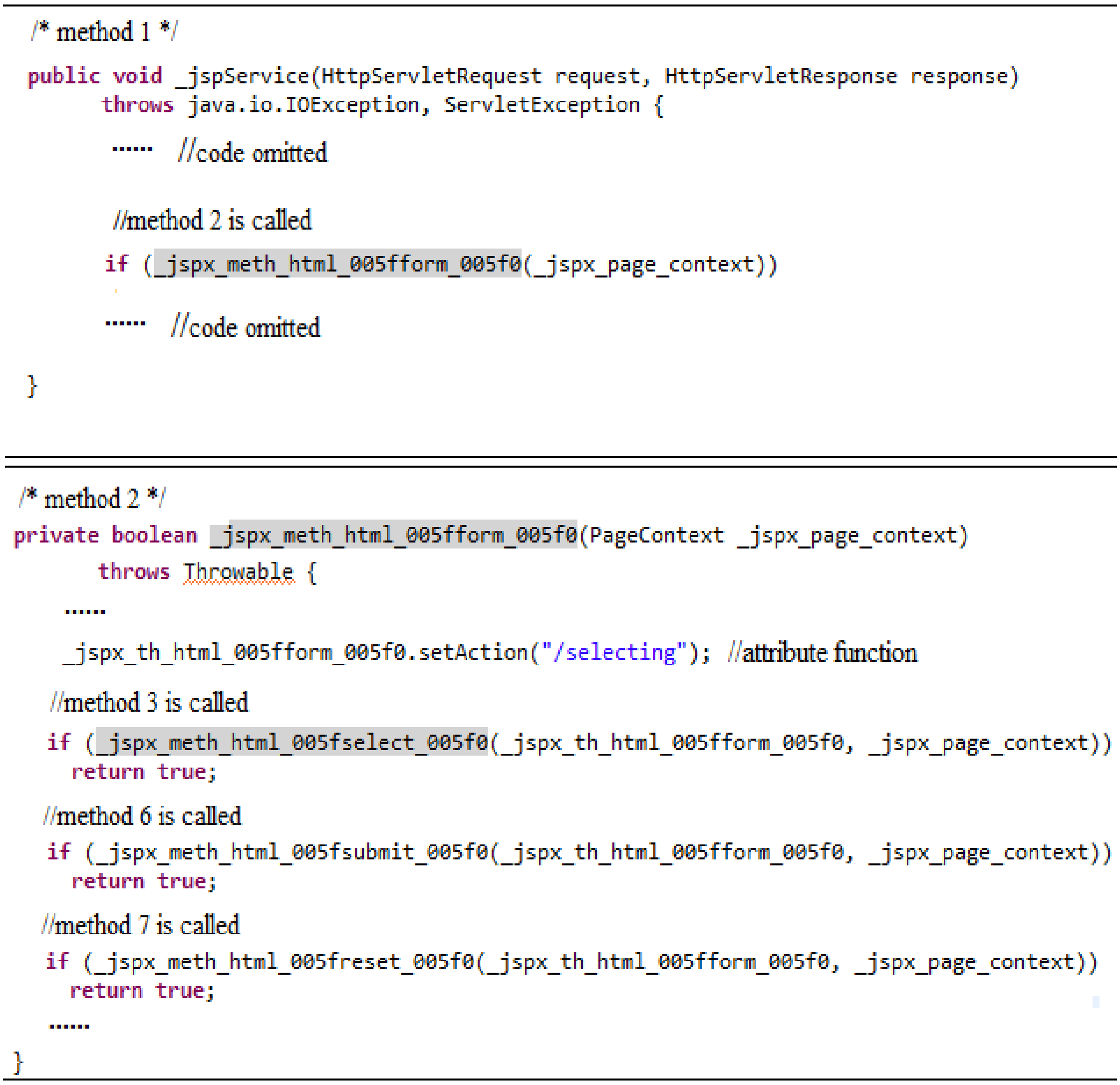
Figure 3.5 shows the invoked methods in methods 1 and 2 respectively. Method 1 invokes method 2, and method 2 invokes methods 3, 6, and 7.

Moreover, method 2 invokes a *non-component* method `setAction("/selecting")` representing the attribute `ation="/selecting"` in the form <`html.form`> shown in Figure 3.3. This kind of method is deemed as an *attribute method* or an *attribute function*.

In Figure 3.6, method 3 for component <`html.select`> invokes methods 4 and 5, each of which represents a <`html.option`> component. This means two components <`html.option`> are nested in component <`html.select`>. Function `setProperty("list")` is invoked for attribute `property="list"` of element <`html.select`>, as shown in Figure 3.3.

Therefore, component methods are directly or indirectly invoked by the `_jspService()` method. By analysing the `_jspService()` method in a Java class, SideAuto can access all the components in a JSP page and construct the structure of the page.

After compiling each JSP page to Java classes, data flow analysis is used to get the structure of a JSP page.



```
/* method 1 */
public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOException, ServletException {
        ......  //code omitted

        //method 2 is called
        if (_jspx_meth_html_005fform_005f0(_jspx_page_context))

        ......  //code omitted

}
```

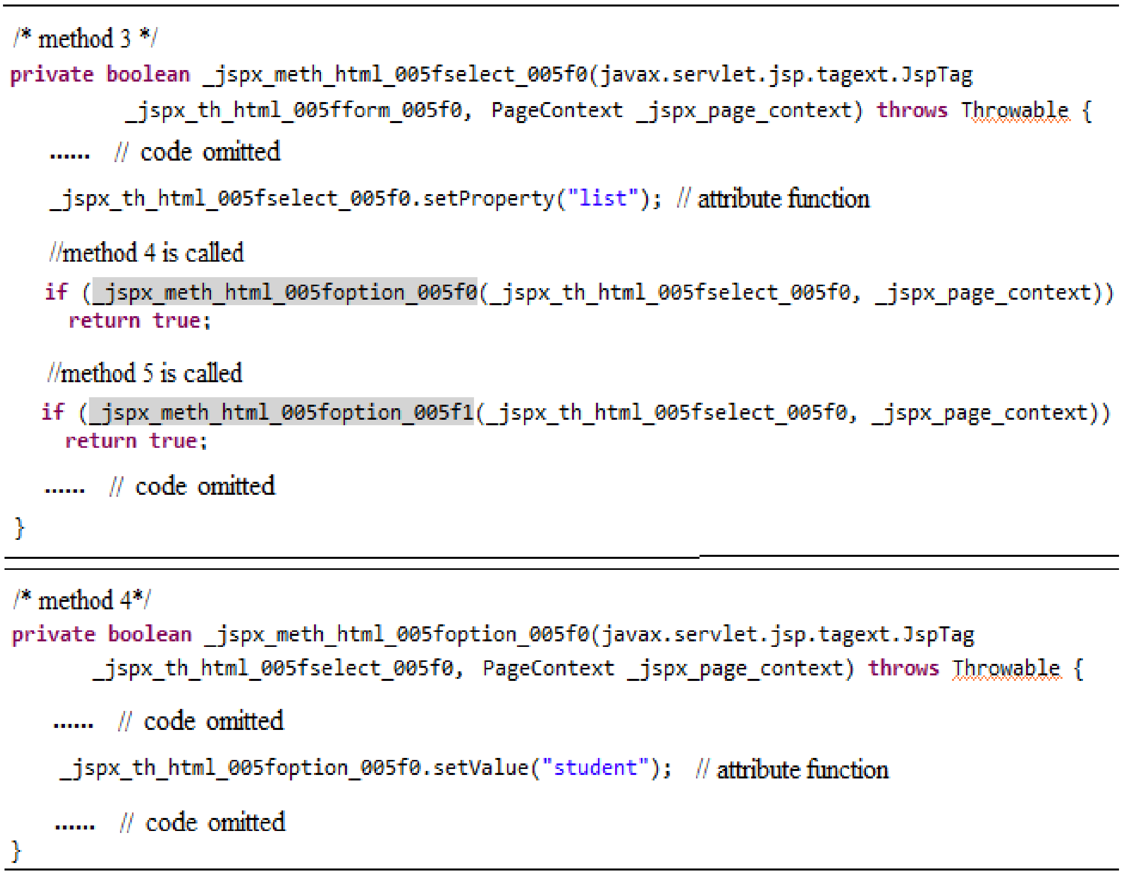
```
/* method 2 */
private boolean _jspx_meth_html_005fform_005f0(PageContext _jspx_page_context)
        throws Throwable {
    ......
    _jspx_th_html_005fform_005f0.setAction("/selecting");  //attribute function
   //method 3 is called
   if (_jspx_meth_html_005fselect_005f0(_jspx_th_html_005fform_005f0, _jspx_page_context))
     return true;
   //method 6 is called
   if (_jspx_meth_html_005fsubmit_005f0(_jspx_th_html_005fform_005f0, _jspx_page_context))
     return true;
   //method 7 is called
   if (_jspx_meth_html_005freset_005f0(_jspx_th_html_005fform_005f0, _jspx_page_context))
     return true;
   ......
}
```

Figure 3.5: Methods 1 and 2

3.2.3 Constructing the Structure of a JSP Page

SideAuto performs data flow analysis on the Java classes. Soot [115], a Java bytecode optimization tool, provides techniques for transforming a Java class into a class of *Jimple* code [116]. Jimple code is a 3-address intermediate representation designed to simplify the analysis of Java code. It provides formal statements including: (1) *AssignStmt* for assignment statements; (2) *InvokeStmt* for statements which invoke methods; (3) *IfStmt* for conditional statements; (4) *ReturnStmt* for statements of returning values and (5) *IdentityStmt* for statements which declare parameters. These five categories of statements are the main statements occurred in a Java class generated from a JSP page.

For example, part (a) in Figure 3.7 shows a Java class which contains two methods: `main()` and `simple()`. Part (b) displays the Jimple code of these two methods. Lines 6, 16 and 17 are the IdentityStmt statements which specify parameters of the methods. Lines 7, 8, 18, 19 and 20 show the AssignStmt statements, and lines 9 and 10 are the InvokeStmt statements which specify the methods invoked.



```
/* method 3 */
private boolean _jspx_meth_html_005fselect_005f0(javax.servlet.jsp.tagext.JspTag
        _jspx_th_html_005fform_005f0, PageContext _jspx_page_context) throws Throwable {
   ......  // code omitted
   _jspx_th_html_005fselect_005f0.setProperty("list"); // attribute function

   //method 4 is called
   if (_jspx_meth_html_005foption_005f0(_jspx_th_html_005fselect_005f0, _jspx_page_context))
     return true;

   //method 5 is called
   if (_jspx_meth_html_005foption_005f1(_jspx_th_html_005fselect_005f0, _jspx_page_context))
     return true;
   ......  // code omitted
}
```

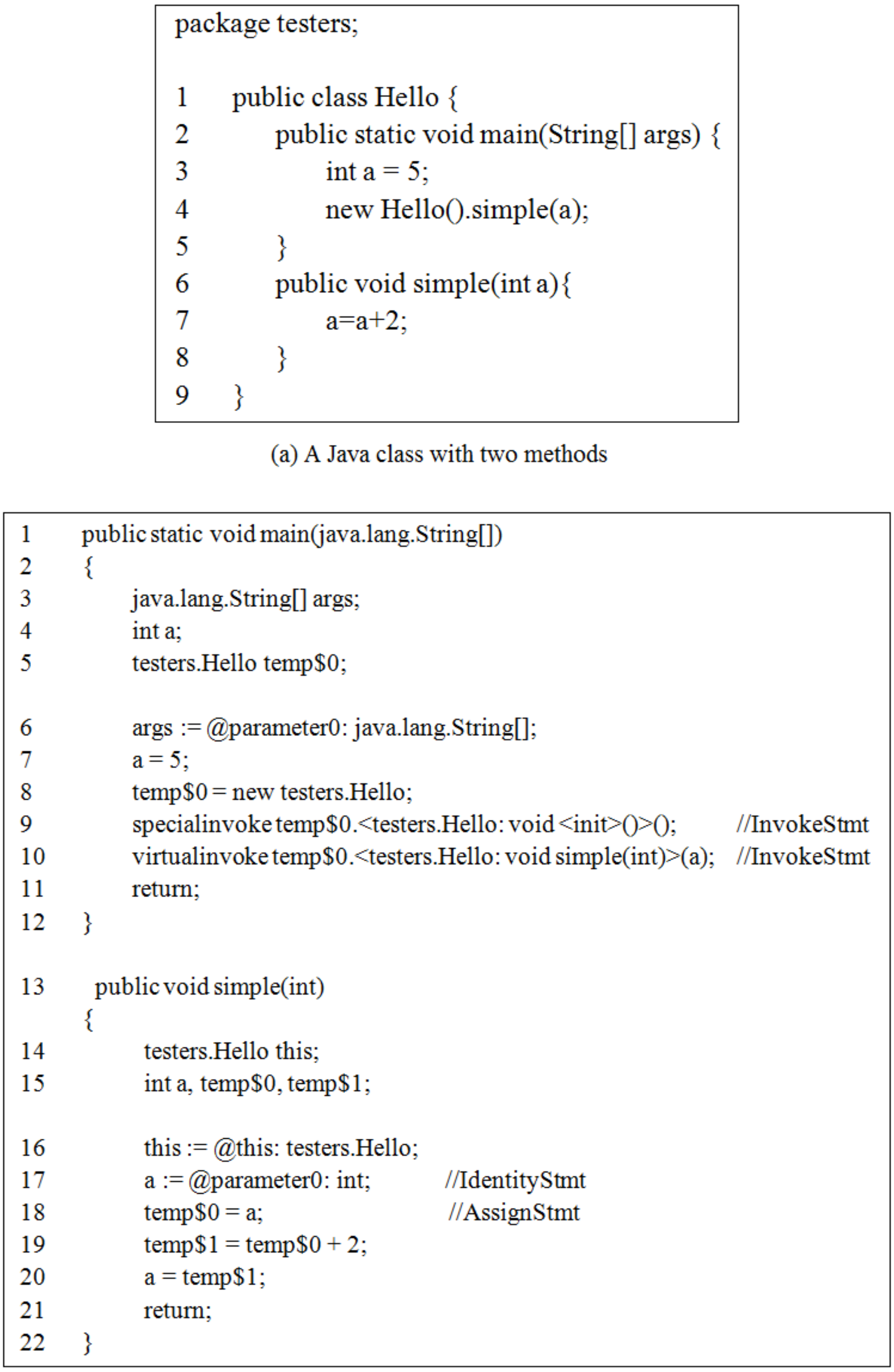
```
/* method 4*/
private boolean _jspx_meth_html_005foption_005f0(javax.servlet.jsp.tagext.JspTag
      _jspx_th_html_005fselect_005f0, PageContext _jspx_page_context) throws Throwable {
   ......  // code omitted
    _jspx_th_html_005foption_005f0.setValue("student");  // attribute function
   ......  // code omitted
}
```

Figure 3.6: Methods 3 and 4

From the _`jspService()` method in the Jimple class for the starting page, SideAuto parses each statement one by one. Each InvokeStmt statement for a component method contains a *tag phrase*, which specifies the type of the component. For example, a tag phrase *"html.FormTag"* indicates that the method is for a <`html.form`> element. Extracting the tag phrases from each InvokeStmt statement, SideAuto can obtain web components on a JSP page.

A component which holds nested components is defined as a parent component, such as a select component <`html.select`> with nested elements <`html.option`>. Methods for the nested components are invoked by those for parent components. By analysing the InvokeStmt statements in a Jimple class, the nesting relation between web components can be retrieved. When a method for component *b* is invoked by a method for component *a*, component *b* is considered as a child of component *a*. SideAuto then performs analysis on the component method for component *b*. The analysis recurs until all the methods in a Jimple class are examined.

Taint analysis. In the Jimple code, a new variable can be declared to express an invoked component. Hence taint analysis is performed for examining statements to track the flow of invoked components. The taint propagation rule is that for *IdentityStmt* or *AssignStmt* statements, if the right-hand side of a statement is tainted, then the variable



```
package testers;

public class Hello {
    public static void main(String[] args) {
        int a = 5;
        new Hello().simple(a);
    }
    public void simple(int a){
        a=a+2;
    }
}
```

(a) A Java class with two methods

```
public static void main(java.lang.String[])
{
    java.lang.String[] args;
    int a;
    testers.Hello temp$0;

    args := @parameter0: java.lang.String[];
    a = 5;
    temp$0 = new testers.Hello;
    specialinvoke temp$0.<testers.Hello: void <init>()>();       //InvokeStmt
    virtualinvoke temp$0.<testers.Hello: void simple(int)>(a);   //InvokeStmt
    return;
}

 public void simple(int)
{
     testers.Hello this;
     int a, temp$0, temp$1;

     this := @this: testers.Hello;
     a := @parameter0: int;          //IdentityStmt
     temp$0 = a;                     //AssignStmt
     temp$1 = temp$0 + 2;
     a = temp$1;
     return;
}
```

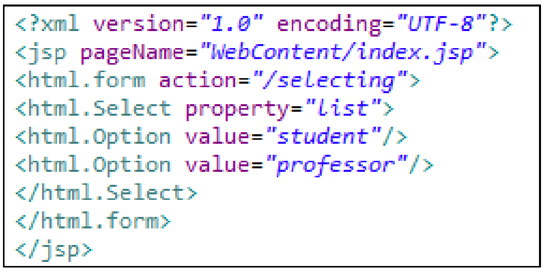
(b) Jimple code of the two methods in (a)

Figure 3.7: A Java class and the related Jimple code

on the left-hand side will be tainted. And if a variable has been tainted but then is given to an untainted value, the variable is sanitised as untainted. Therefore, taint analysis guarantees the completeness of checking all component methods and attribute methods.

Figure 3.8 shows the structure graph built for page *index.jsp* in Figure 3.3.

After analysing the structure of a single page, control flow analysis is used to retrieve the structure of a web application.

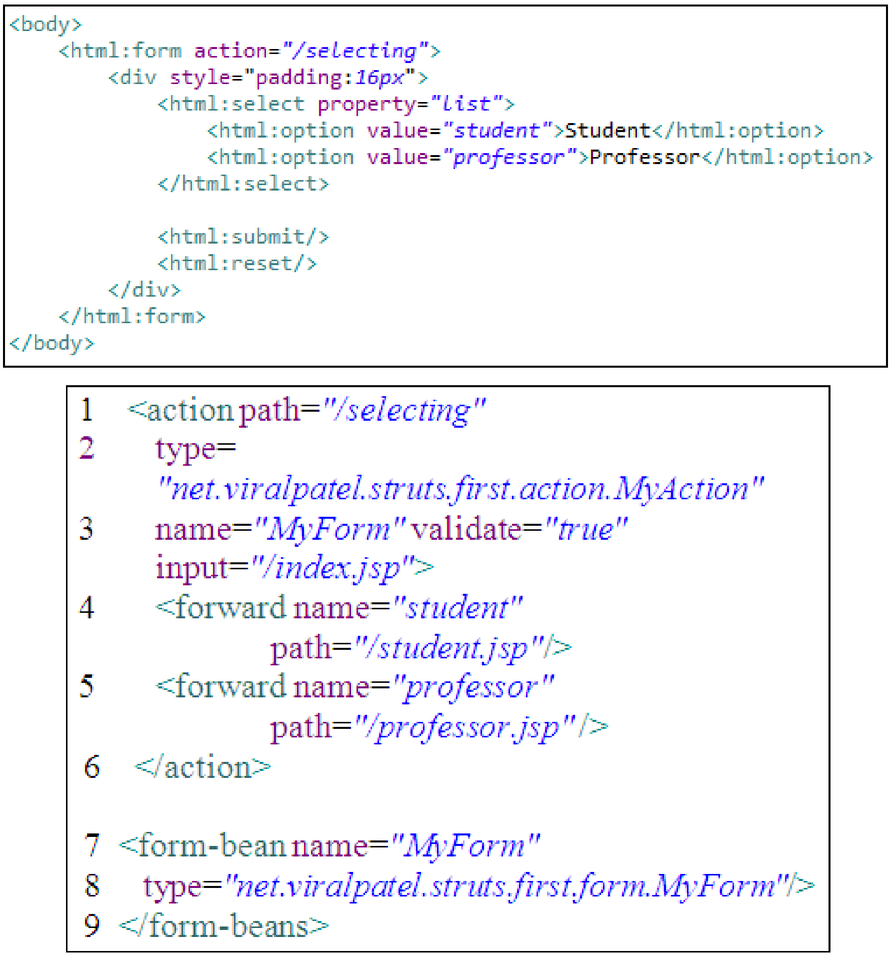


```
<?xml version="1.0" encoding="UTF-8"?>
<jsp pageName="WebContent/index.jsp">
<html.form action="/selecting">
<html.Select property="list">
<html.Option value="student"/>
<html.Option value="professor"/>
</html.Select>
</html.form>
</jsp>
```

Figure 3.8: Structure graph built for page *index.jsp* in Figure 3.3

3.2.4 Constructing the Structure of a Web Application

SideAuto simulates the role of Action servlet to identify the forwarding pages derived from this page. In Struts 1, when a browser submits a user action using a form in a format like `<html:form action = "...">` or a link like `<html:link action = "...">`, Action servlet associates the user action to an action defined in the Struts configuration file. To easily specify the action defined in the Struts configuration file, a term *Action* is used to indicate it. In an *Action*, the Action class used for analysing user actions and the ActionForward strings with related forwarding pages are specified.



```
<body>
    <html:form action="/selecting">
        <div style="padding:16px">
            <html:select property="list">
                <html:option value="student">Student</html:option>
                <html:option value="professor">Professor</html:option>
            </html:select>

            <html:submit/>
            <html:reset/>
        </div>
    </html:form>
</body>
```

```
<action path="/selecting"
   type=
   "net.viralpatel.struts.first.action.MyAction"
   name="MyForm" validate="true"
   input="/index.jsp">
   <forward name="student"
            path="/student.jsp"/>
   <forward name="professor"
            path="/professor.jsp"/>
 </action>

<form-bean name="MyForm"
  type="net.viralpatel.struts.first.form.MyForm"/>
</form-beans>
```

Figure 3.9: The user action on page *index.jsp* and its related *Action*

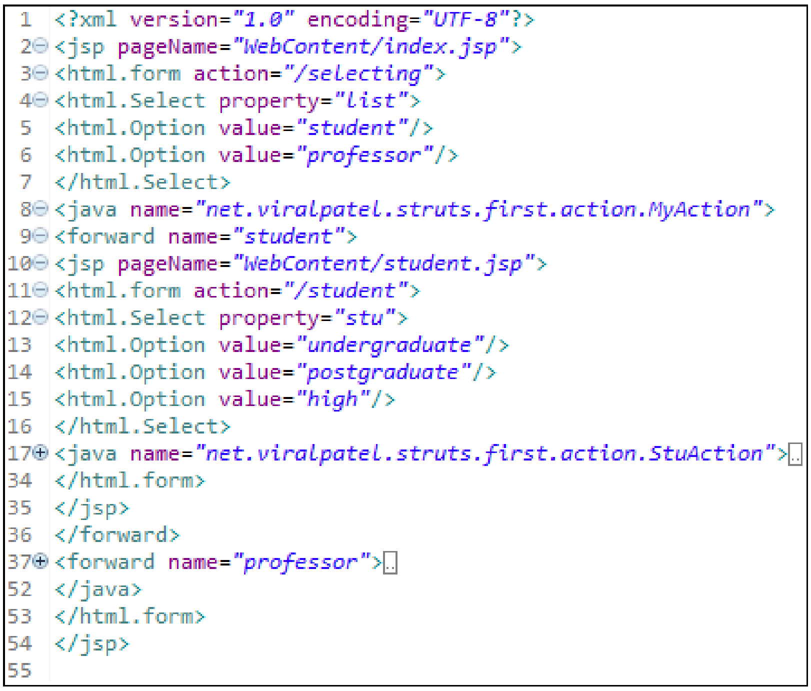
Figure 3.9 shows the related *Action* for the user action with attribute `action="/selecting"` in page *index.jsp* in Figure 3.3. To read easily, page *index.jsp* in Figure 3.3 is displayed in the upper part of Figure 3.9, and the lower part displays the *Action*.

The match between a user action and the related *Action* is established through the match between the attribute of the user action, i.e. `action="/selecting"`, and the attribute in the *Action*, i.e. `path="/selecting"`, as shown in Figure 3.9.

From the *Action* in Figure 3.9, the attribute *type* at line 2 specifies the Action class used to analyse the user inputs. And attribute *name* at line 3 refers to the corresponding ActionForm class, which is defined at lines 7-9. The ActionForm class is used to access the user values. Attribute `input="/index.jsp"` at line 3 indicates that this *Action* is invoked by a user action from page *index.jsp*.

Tags <forward> at lines 4-5 specify the forwarding pages, where attribute *name* of a tag represents the ActionForward string and attribute *path* indicates the forwarding page. In the user action, one of two forwarding pages is forwarded, either page *student.jsp*, or page *professor.jsp*.

Therefore, by analysing an *Action*, SideAuto obtains the forwarding pages of a given JSP page. When a forwarding page obtained has not been analysed before, SideAuto recurs to analyse the page, starting from compiling it to a Java class. The process continues until all the forwarding pages have been examined.



```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jsp pageName="WebContent/index.jsp">
3  <html.form action="/selecting">
4  <html.Select property="list">
5  <html.Option value="student"/>
6  <html.Option value="professor"/>
7  </html.Select>
8  <java name="net.viralpatel.struts.first.action.MyAction">
9  <forward name="student">
10 <jsp pageName="WebContent/student.jsp">
11 <html.form action="/student">
12 <html.Select property="stu">
13 <html.Option value="undergraduate"/>
14 <html.Option value="postgraduate"/>
15 <html.Option value="high"/>
16 </html.Select>
17 <java name="net.viralpatel.struts.first.action.StuAction">
34 </html.form>
35 </jsp>
36 </forward>
37 <forward name="professor">
52 </java>
53 </html.form>
54 </jsp>
55
```

Figure 3.10: The final structure graph of the university identity system

Figure 3.10 presents the final structure graph of the university identity system displayed in Figure 3.2. The first <jsp> element denotes the starting page of the web application. Tag <java> at line 8 specifies the Action class for the user action displayed at line 3.

Moreover, the forwarding pages from this user action are specified, using the <forward>

tags displayed at lines 9 and 37 respectively.

After generating the structure graph of a web application, next SideAuto obtains path constraints of each branch.

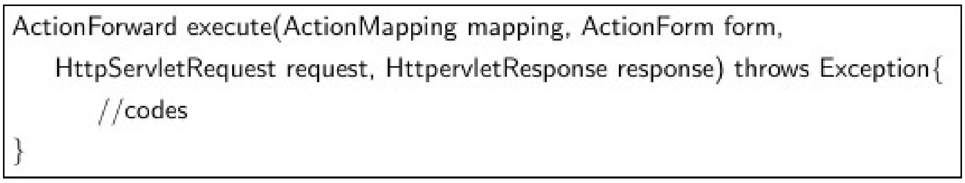
3.3 Getting Path Constraints

To get path constraints, SideAuto performs symbolic execution [72] on the methods in Action classes using the **Symbolic PathFinder** (SPF) [93]. It is a tool for performing symbolic execution and constraint solving at the bytecode level. Essentially, the SPF is built on top of the Java PathFinder (JPF) [9] model checking tool-set for Java programs. We use SPF to achieve an automatic generation of test cases.

The Action class specifies branches using ActionForward strings, conditioning on different user inputs. Hence we consider ActionForward strings as branches and user inputs represent branch conditions. To use the SPF for extracting ActionForward strings and branch conditions from Action classes, we develop a built-in *rewriting mechanism.* It rewrites Action classes so that they can be understood by the SPF.

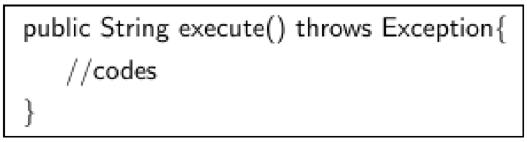
3.3.1 Rewriting Mechanism

Each Action class in Struts 1 has an *execution method* named `execute()`, which manages the user values with a method declaration like:



```
ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpervletResponse response) throws Exception{
        //codes
}
```

and that defined in Struts 2 looks like:

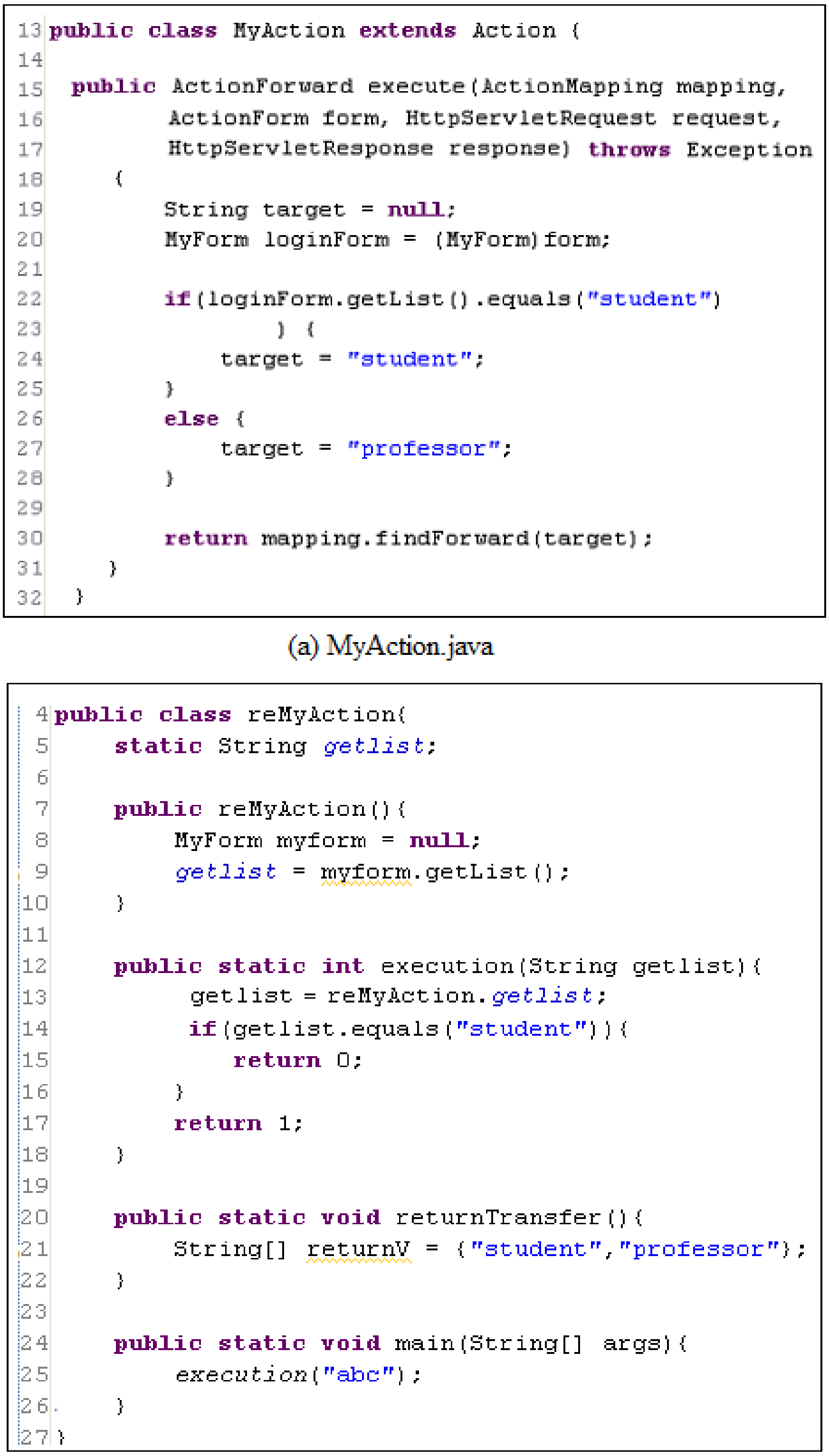


```
public String execute() throws Exception{
    //codes
}
```

SideAuto rewrites all the Action classes in a web application. Each rewritten class is named by adding a prefix *"re"* to the original class name. For example, class *A.java* is rewritten with the name *reA.java.* The execution method `execute()` in every Action class is renamed as `execution()`. Method `execution()` is built with parameter(s) which represent(s) the user inputs.

Figure 3.11(a) illustrates an Action class *MyAction.java*, which is related to the *Action* with attribute `path= "/selecting"` in Figure 3.9. And Figure 3.11(b) presents the rewritten class *reMyAction.java.*

In Figure 3.11(a), line 20 creates an object of the corresponding ActionForm class. In an ActionForm class, getter & setter methods provide the capability of accessing user inputs. Then the Action class uses the returned user inputs to determine which branch should respond.



```
public class MyAction extends Action {

 public ActionForward execute(ActionMapping mapping,
         ActionForm form, HttpServletRequest request,
         HttpServletResponse response) throws Exception
   {
        String target = null;
        MyForm loginForm = (MyForm)form;

        if(loginForm.getList().equals("student")
                ) {
            target = "student";
        }
        else {
            target = "professor";
        }

        return mapping.findForward(target);
    }
 }
```

(a) MyAction.java

```
public class reMyAction{
    static String getlist;

    public reMyAction(){
        MyForm myform = null;
        getlist = myform.getList();
    }

    public static int execution(String getlist){
         getlist = reMyAction.getlist;
         if(getlist.equals("student")){
            return 0;
        }
        return 1;
    }

    public static void returnTransfer(){
        String[] returnV = {"student","professor"};
    }

    public static void main(String[] args){
        execution("abc");
    }
}
```

(b) reMyAction.java

Figure 3.11: An Action class and its rewritten class

In Struts, each component comes with an attribute *property*. The name of a getter method involves the value of attribute *property* for a component. This suggests which component its user input is returned. For example, method `getList()` at line 22 in Figure 3.11(a) is a getter method for the user input in the component whose attribute value of *property* is *list*, i.e. <html:select property = "list">.

In this chapter, we identify a component using its attribute value of *property*. For example, the select component <html:select property = "list"> is called as component *list*.

During the rewriting, SideAuto uses Soot to convert an Action class to a class of Jimple code. Analysing the Jimple code statically, SideAuto extracts the getter methods in method `execute()`, and then transforms them into the parameters of method `execution()` in the rewritten class.

Consider the rewritten class *reMyAction.java* in Figure 3.11(b). The code in the class is meaningless. The rewritten class cannot be run as it is only used to let the SPF understand the code.

Method `execution` at line 12 contains a parameter: `String getlist`. This parameter refers to the user input returned from getter method `getList()`, as suggested from lines 5, 9 and 13. The parameter name is given to be consistent with the getter method, e.g. the name *getlist* is obtained from method `getList()`.

At the time of developing SideAuto, the SPF can only handle a numerical returned value in an examined method. Method `execute()` in an Action class returns ActionForward strings. Instead, SideAuto builds method `execution()` returning integers. A method `returnTransfer()` is constructed in the rewritten class, containing an array returnV[] which collects all the ActionForward strings in the *Action*. Then by mapping a returned integer i to the value of returnV[i], SideAuto obtains the ActionForward string returned.

The class examined by the SPF should be a *startup class*, i.e. a class containing the main method. Therefore, SideAuto creates the main method in each rewritten class, which invokes method `execution()` with random parameter values.

3.3.2 Performing Symbolic Execution

Configuration of SPF

When using the SPF, the following parameters should be configured:

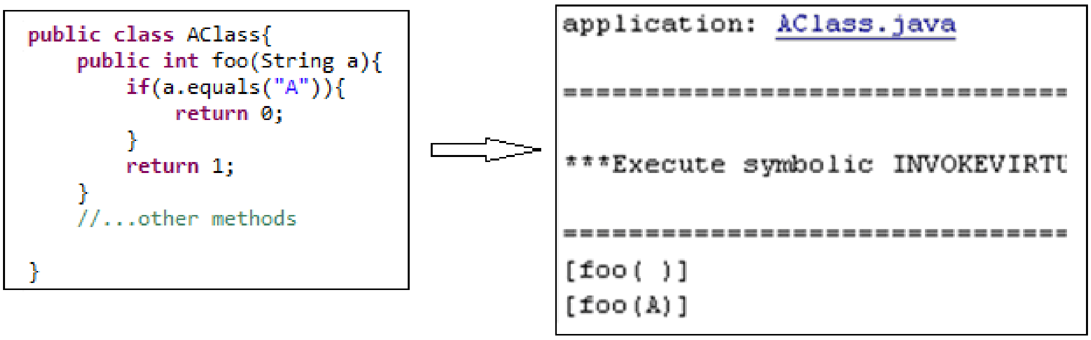
- *target*: the class that is analysed.
- *symbolic.method*: the method examined and its *symbolic parameters*, which are denoted using a symbol *sym*. If there are more than one parameter in an examined method, # is used to separate them. For example, *"sym#sym"* indicates that there are two parameters in the method.
- *listener*: the class which prints out the information in terms of branches and path constraints in a symbolic run.

By specifying the *target*, the *symbolic.method*, and the *listener* used, SPF can print out the branches with path conditions in a report.

Considering a Java class shown on the left-hand side in Figure 3.12. With the following the SPF configuration:

```
target= AClass
symbolic.method= AClass.foo(sym)
listener = gov.nasa.jpf.symbc.sequences.SymbolicSequenceListener
```

the running result of the symbolic execution on this method is shown on the right-hand side in Figure 3.12. The listener used: *gov.nasa.jpf.symbc.sequences.SymbolicSequenceListener* is predefined in the SPF.



```
public class AClass{
    public int foo(String a){
        if(a.equals("A")){
            return 0;
        }
        return 1;
    }
    //...other methods

}
```

```
application: AClass.java

=================================

***Execute symbolic INVOKEVIRTU

=================================
[foo( )]
[foo(A)]
```

Figure 3.12: Running result of the SPF for method `foo()` in *AClass.java*

This running report suggests that there are two branches in this method. One is `[foo(A)]`, conditioning on a user value A, and the other is `[foo()]`, whose condition is any value excluding value A.

However, this running result does not display the ActionForward strings returned for each branch. The path constraint, e.g. value A is specified, however, the report does not specify which parameter the path constraint is related to. More precisely, merely from a branch, e.g. `[foo(A)]`, we cannot obtain which parameter that condition A is related to.

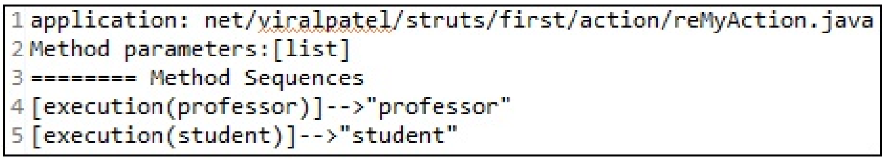
Hence we develop a new listener–*MyListener* using a class *MyListener.java*, which outputs data in terms of branch conditions (user inputs), branches (ActionForward strings) and parameters (components).

MyListener

SideAuto rewrites getter methods as parameters of method `execution()`. These parameters are recorded using an array. SideAuto also creates array returnV[] to store ActionForward strings. When performing symbolic execution, *MyListener.java* takes the array of parameters and array returnV[] as parameters. Associating the integers returned from each branch to the ActionForward strings stored in array returnV[], *MyListener* prints out the branches in terms of parameters, path constraints and returned ActionForward strings.

In SideAuto, SPF takes a rewritten Action class as the *target*, method `execution()` with the symbolic parameters as the *symbolic.method*, and *MyListener* as the *listener*.

Figure 3.13 shows the output of a symbolic execution for method `execution()` in class *reMyAction.java* in Figure 3.11(b). Line 1 shows the examined class, i.e. the target, and line 2 shows the parameter. Method sequences are the branches generated with path constraints and ActionForward strings returned. From the report, an ActionForward string *professor* is returned when a user selects value *professor* in the *list* component. Otherwise, the ActionForward string *student* is returned when the input value is *student*.



```
1 application: net/viralpatel/struts/first/action/reMyAction.java
2 Method parameters:[list]
3 ======== Method Sequences
4 [execution(professor)]-->"professor"
5 [execution(student)]-->"student"
```

Figure 3.13: The SPF running result for *reMyAction.java* in Figure 3.11(b)

Notice that the initial output at line 4 in Figure 3.13 was

$$\text{[execution()]} \rightarrow \text{“professor”}$$

which is similar to the one of [foo()] displayed Figure 3.12. This branch without path constraint is originated from the code of `"return 1"` present at line 17 in *reMyAction.java* in Figure 3.11(b). It does not provide a specific condition of the branch. The code of `"return 1"` in *reMyAction.java* comes from statement `else` at line 26 in *MyAction.java* in Figure 3.11(a), which is opposite to the branch with condition of value *student* at lines 22-25.

Originally, the listener outputs a branch with the path condition when the condition is specified in the code. For the branches, e.g. for else branch, the path conditions are not printed out.

To improve, SideAuto completes all the branches with path conditions.

Completing a Branch

When there are branches generated by the SPF without path conditions, SideAuto scans those components taken as parameters from the structure graph, to get all the possible values of them. It checks whether all combinations of these values are used as branch conditions in the symbolic execution. If not, SideAuto complements the branches without path constraints by adding the unused values as path conditions to them. There are two cases when completing the branches.

1. When the values of a parameter are a predefined set of constants, the unfinished branches will be complemented with unused constants.

   For example, assume a component takes a set of constant values $\{v_1, v_2, v_3\}$, and the branches generated by the SPF are `[method1()]` $\rightarrow$ *string*2 and `[method1(`v_1`)]` $\rightarrow$ *string*1. Then the branches will be complemented as `[method1(`v_3`)]` $\rightarrow$ *string*2, `[method1(`v_2`)]` $\rightarrow$ *string*2 and `[method1(`v_1`)]` $\rightarrow$ *string*1.

2. When the values of a parameter are arbitrary, e.g. a text box, the tester either defines a set of specific values or allows SideAuto to generate a set of random values. Then

the unfinished branches can be complemented.

Given the SPF running reports with completed branches for each Action class and the structure graph generated, SideAuto can generate test cases of execution paths.

3.4 Test Case Generation

3.4.1 Configuration for Test Case Generation

Testers must configure the following options to generate test cases:

1. Loop depth (l): the maximum number of times a web page is allowed to repeatedly appear in a path. The default value is $l = 0$.

   For example, consider there are three transitions $(A, a_1, B), (A, a_2, C)$ and (B, a_3, A) in a web application and $l = 1$. Four execution paths will be generated starting from page A, expressed by a simplified format which only displays the web pages in each transition: (1) $A \to B$; (2) $A \to C$; (3) $A \to B \to A \to B$ and (4) $A \to B \to A \to C$.

2. Option All/One: the degree of the completeness of test case generation.

   If execution paths go through identical web pages, i.e., the transitions between different execution paths only vary from user inputs, a tester can choose option *One* to generate one test case to represent all the execution paths with the identical web pages transferred. This aims to reduce the space of test cases when there are a huge number of execution paths in a web application. The first values or default values in each component are used to generate the test case in option One. By default, SideAuto uses option *All*.

To illustrate the differences between option All/One consider Figure 3.14. In option All, seven test cases are generated, one for each possible combination of values by. Choosing option One instead will result in two test cases: (1) $A \to B \to D$ and (2) $A \to C$. More details of the test cases generated under option All/One are displayed in Table 3.1. In this table, an execution path, such as $A - a - B - e - D$, indicates that page A forwards to page B through user action a and then to page D through user action e.

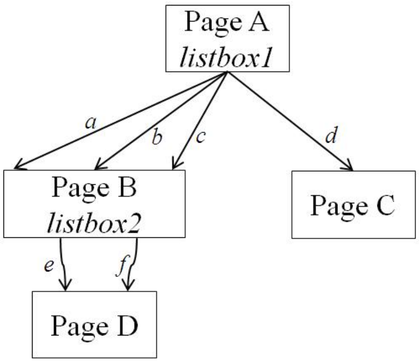


Figure 3.14: An example of execution paths

A test case in option One like 1. $A - a - B - e - D$ is regarded as a *compressed test case*, which reflects the web pages transferred among omitted execution paths. And a test case like 2. $A - d - C$ is deemed as a *non-compressed test case.*

When calculating the leakage, although omitted execution paths in option One are represented by one test case, they are still considered as individuals, instead of as a whole. Hence it is regarded that there are no leaks from the omitted execution paths.

Table 3.1: Test Cases generated from the graph in Figure 3.14 under Option All/One

Option	Test Cases
All	1. $A - a - B - e - D$
	2. $A - a - B - f - D$
	3. $A - b - B - e - D$
	4. $A - b - B - f - D$
	5. $A - c - B - e - D$
	6. $A - c - B - f - D$
	7. $A - d - C$
One	1. $A - a - B - e - D$
	2. $A - d - C$

3. Terminal nodes: by default terminal nodes are the web pages without any user actions can be performed on. Particular nodes can be specified as terminal nodes, to flexibly generate test cases according to the threat scenario examined, or to stop the test case generation earlier.

4. Component values: for a component without predefined constants like a text box, specific values used for testing should be defined. Values are specified following a format:

   $webpageName :< widgetType : attributeName = attributeValue >: value1; value2...$

   A tester can also define a *Regular Expression* [112] to generate random values. SideAuto contains a generator to generate arbitrary strings according to the regular expression specified. And if the tester does not specify the number of random values generated, by default one random string will be generated for each component.

5. Probabilities of user actions: by default all user actions in one component are equally likely, i.e. the uniform distribution on user actions. The tester can specify different probabilities of user actions. Details will be explained in section 3.5.

3.4.2 Generation of Test Cases

Given a structure graph and the branches of each Action class, SideAuto performs *depth-first search* to connect branches based on the structure graph.

Each execution path starts from the starting page (root node). When parsing to a tag `<jsp pageName="...">` in the structure graph, SideAuto searches for all the possible user actions coming from this page, through nested components including forms `<html:form action="...">`, links with attribute *action* `<html:link action="...">` and links directing to a new page `<html:link page="...">`.

With an action, each nested element <forward name="..."> contains a forwarding page. The attribute value of *name* represents an ActionForward string in this action. From the related Action class of this action, branches with branch conditions and ActionForward strings can be obtained from the SPF running report. By replacing the ActionForward strings with the forwarding pages, single transitions from a web page can be built.

SideAuto repeats this process when a forwarding page is arrived, starting from the related <jsp> tag. A completed execution path is built when a terminal node is accessed or a web page accessed is restricted by the loop limit. SideAuto stops the generation until all the test cases of execution paths are obtained.

Take an example of a test case generated for the university identity system:
WebContent/index.jsp:<html.Select property="list">(professor)→WebContent/professor.jsp:
<html.Select property = "pro">(senior)→ WebContent/welcome.jsp

This test case composes of two transitions. One is transferred from page *index.jsp* to page *professor.jsp* by selecting value *professor* in component *list*. And then the second transition is triggered from page *professor.jsp* to page *welcome.jsp* through a user selection of value *senior* in component *pro*.

Transitions can also take user actions with joint user inputs. For example an authentication includes user inputs for the *username* and the *password*, which jointly decide a transition. SideAuto is capable of generating transitions triggered by multiple user inputs.

3.4.3 Test Cases Coverage

When generating test cases, a loop bound can be used to limit the number of times a page is allowed to be accessed. And for some components, such as text boxes, specific values are required to be defined.

With these configurations, test cases are restricted to be generated. A full coverage of test cases cannot be fulfilled. However, the configurations can eliminate the generation of test cases unrelated to the secret and assist developers in examining particular test cases.

For example, a developer only wants to examine that if some particular sensitive values typed in a text box are leaked. Since the developer does not care about the leakage of other values typed in the text box, it is unnecessary to test all the possible values. So the developer can only specify and test the sensitive values that he wants to examine.

On the other hand, if the developer knows that a group of values will generate the same web traffic, he can merely test one value to get the traffic pattern for all the values in this group. In this case, the specification of values can improve the efficiency of testing.

Hence, although our system cannot guarantee a full coverage of test cases in a web application, it will generate all the test cases related to the secrets examined, based on the developer's configuration. It is the same case for test cases generated in Chapters 4 and 5.

3.5 Probability Distribution

We consider test cases with different probability distributions. In the real world, user actions do not always follow the uniform distribution. Hence, with different probability distribution, one of the analysed results may closer to the leakage in the real world. Moreover, the analysed results with multiple probability distributions can provide a more in-depth evaluation of the leakage. Details are shown in the following.

3.5.1 Predefined Probability Distributions

SideAuto predefines three probability distributions:

- Up: uniform distribution on execution paths, i.e. each test case with a same probability;
- Ua: uniform distribution on user actions in a component;
- Pr: a distribution according to the PageRank [34]

Consider the execution paths from the graph in Figure 3.14. Under option All, there are seven test cases generated in total. With the uniform distribution on test cases, i.e. in Up, the probability of each test case is 0.143 (= 1/7). On the other hand, with the uniform distribution on user actions for each component, the probabilities of user actions with values a, b, c, and d in $listbox1$ are all 0.25, and the probabilities of actions e and f in $listbox2$ are 0.5. As a result, the probabilities of each test case via path $A \rightarrow B \rightarrow D$ is 0.125, while the test case of $A \rightarrow C$ is 0.25.

In option One with distribution Up, the probability of test case 1 becomes 0.857 since it is the sum of all the compressed paths, and the probability with distribution Ua is 0.75. Details are summarised in Table 3.2.

Table 3.2: Probability Distributions of test cases in Figure 3.14 in Up and Ua

Option	Test Cases	Up	Ua
All	1. $A-a-B-e-D$	0.143	0.125
	2. $A-a-B-f-D$	0.143	0.125
	3. $A-b-B-e-D$	0.143	0.125
	4. $A-b-B-f-D$	0.143	0.125
	5. $A-c-B-e-D$	0.143	0.125
	6. $A-c-B-f-D$	0.143	0.125
	7. $A-d-C$	0.143	0.25
One	1. $A-a-B-e-D$	0.857	0.75
	2. $A-d-C$	0.143	0.25

PageRank [34] is a link analysis algorithm to measure the importance of each web page in a web application. When using the probability distribution based on PageRank, the probabilities of test cases are closer to the real-world case since it considers a user either following an anticipative action or jumping to a random page.

However, some issues exist when using PageRank. For example, PageRank results from the convergence of page transitions to a fix number. But in this work test cases generated are restricted by a loop bound. Moreover, test cases generated only follow the anticipated user actions, while PageRank takes account of the unexpected user actions.

Nevertheless, the main point here is to showcase the configuration for different probability distributions in SideAuto.

3.5.2 Configuring particular probability distributions

Moreover, a developer can configure an exclusive probability distribution of user actions using (1) exact values or (2) *probability levels*.

Still take the execution paths from the graph in Figure 3.14 as an example.

Assume that the probability of user action *(listbox1, a)* is 0.4, and the other three actions $(listbox1, b)$, $(listbox1, c)$ and $(listbox1, d)$ on Page A are 0.2 each. User actions e and f on Page B are not configured particularly, so by default the uniform distribution is applied, each one with a probability of 0.5. Therefore in option All the probabilities of test cases 1, 2 and 7 in Table 3.1 are 0.2 each and the remaining are 0.1 each.

On the other hand, the probabilities can be defined by using probability levels such as (HI, ME, LO). By default, the proportions of probability levels HI, ME and LO are established as 0.75, 0.5 and 0.25 respectively, and each user action is given a level of ME. But a developer can configure the number of probability levels and specify different proportions for them. Then the normalisation of probabilities is implemented to get the exact value for each user action.

Consider the execution paths from the graph in Figure 3.14. The probabilities of user actions $(listbox1, a)$ and $(listbox1, b)$ are configured as levels HI and LO respectively, and the other user actions in $listbox1$ stay with the default level of ME. After normalisation, the probabilities of user actions in component $listbox1$ in terms of each level become HI: 0.375, ME:0.25 and LO: 0.125.

3.6 Quantifying Leakage

After generating the test cases, SideAuto executes them to collect web traffic. This section introduces test case execution and show how to analyse web traffic collected for evaluating leakage.

HtmlUnit [7] is used to execute test cases. It automatically accesses to web pages and performs user actions using a simulated web browser. Jpcap [10], a tool for real-time network traffic capture and analysis, is used in collecting packets.

3.6.1 Collecting Web Traffic

Given a test case, the user actions in each transition are extracted and executed in order.

Consider the following test case:

WebContent/index.jsp: <html.Select property="list"> (professor) → WebContent/professor.jsp: <html.Select property="pro"> (senior) → WebContent/welcome.jsp

In the beginning, the client accesses to the homepage *index.jsp*, and selects value *professor* in component *list*. SideAuto checks the forwarding page to guarantee that it is forwarded to page `professor.jsp`. Then value *senior* is selected in component *pro* on page `professor.jsp`. Finally, SideAuto checks if the path is terminated at page `welcome.jsp`.

If a user action contains multiple user inputs, they are typed in each component one by one. When all the inputs in a user action are completed, the form is submitted automatically by SideAuto if neither a submit button is specified nor the JavaScript is used to automate the submission. During the testing, Jpcap records network traffic generated for each test case. Each test case is executed once and a sequence of web traffic is collected.

SideAuto contains a filter mechanism to improve the accuracy of traffic analysis. When capturing the traffic, the filter filters out the packets which are not from the target application by comparing the port numbers of packets with that of the communication in the web application. Moreover, duplicate packets are removed, which are identified through packet sequence numbers.

In this chapter, we assume that during a short time period, the factors which make the observations for a test case varying are invariant. For example, the network conditions during a short time period are fixed, and also the contents of web pages invariant. Therefore, it is feasible to assume that the outputs for test cases over short time periods are invariant. So in this chapter we test each test case once and collect a sequence of web traffic for each test case to analyse the leakage.

3.6.2 Quantifying Leakage

After collecting the web traffic for a test case, SideAuto extracts the sizes and directions of each packet to form an observation, i.e., a sequence directional packet sizes.

With a set of directional packet sizes for a set of test cases, then SideAuto quantifies the leakage of user privacy. Entropy metrics in terms of Shannon entropy and min entropy are used to calculate the amount of confidential information leaked.

An equivalence relation is built upon the observations for each test case, which is same as an equivalence relation built on secret values.

Given a set of test cases T for a web application G and a set of corresponding observations O where $f : T \to O$, an equivalence relation $\sim_G$ is constructed as

$$\forall\, t_i,\ t_j\ \in T,\ t_i \sim_G t_j \text{ if and only if } f(t_i) = f(t_j)$$

This indicates that two sequences of user actions are indistinguishable if and only if their observations of web traffic are identical.

As defined in Chapter 2, information leakage can be quantified using mutual information:

$$\Delta_F(S) = F(S) - F(S|O)$$

where $F(S)$ and $F(S|O)$ are the measures of entropy and conditional entropy using either

Shannon entropy $F = H$ or min entropy $F = H_\infty$.

Therefore, given a set of test cases T and an equivalence relation X on test cases$T$, the leakage of a user secret can be quantified by:

$$\Delta_F(T) = F(T) - F(T|X)$$

where $F(T)$ and $F(T|X)$ are calculated using either Shannon entropy $F = H$ or min entropy $F = H_\infty$ defined in section 2.1.3.

Shannon Entropy

Given a set of test cases T with distribution Up, i.e. the uniform distribution on test cases, the original uncertainty of a secret before observations can be calculated by Shannon entropy, as defined in Definition 8:

$$H(T) = \log(|T|), \tag{3.1}$$

where $|T|$ is the number of test cases in set T.

Given an equivalence relation X on set T after observations with distribution Up, conditional Shannon entropy defined in Definition 10 can be deduced as:

$$\begin{aligned} H(T|X) &= \sum_{X_i \in X} p(X_i) H(T|X_i) \\ &= \sum_{X_i \in X} \frac{|X_i|}{|T|} \log |X_i| \end{aligned} \tag{3.2}$$

where $X_i \in X$ is an equivalence class and $|X_i|$ is the cardinality of class X_i, i.e. the number of test cases equivalent in this class.

Equation 3.2 shows that the uncertainty of the secret after observations is related to the numbers of test cases equivalent in each equivalence class, with distribution Up. This equation is applicable to the cases in either option All or One, as the omitted execution paths in option One compressed by a single test case are also considered as individual secret values.

Therefore, the information leakage with distribution Up is calculated by:

$$\begin{aligned} I(T) &= H(T) - H(T|X) \\ &= \log |T| - \sum_{X_i \in X} \frac{|X_i|}{|T|} \log |X_i| \end{aligned} \tag{3.3}$$

Min Entropy

In terms of min entropy, the original uncertainty before observations with distribution Up defined in Definition 13 is:

$$H_\infty(T) = \log |T| \tag{3.4}$$

same as using Shannon entropy.

Given an equivalence relation X, the conditional min entropy after observations with distribution Up, defined in Definition 15, is:

$$H_\infty(T|X) = -\log \frac{N}{|T|}, \tag{3.5}$$

where $N = |X|$, which depends on the number of classes in equivalence relation X.

Proof.

$$\begin{aligned} H_\infty(T|X) &= -\log \sum_{X_i \in X} p(X_i) \max_{t_j \in X_i} p(t_j|X_i) \\ &= -\log \sum_{X_i \in X} \frac{|X_i|}{|T|} * \frac{1}{|X_i|} \\ &= -\log \sum_{X_i \in X} \frac{1}{|T|} = -\log \frac{N}{|T|} \end{aligned}$$

Therefore the information leakage with distribution Up can be measured using min entropy:

$$I_\infty(T) = H_\infty(T) - H_\infty(T|X) = \log(|T|) + \log \frac{N}{|T|} = \log(N) \tag{3.6}$$

3.7 Experiments

We evaluate SideAuto over five web applications built upon the Struts framework. They are deployed to Google App Engine (GAE) [6]. The detailed descriptions of the five web applications are shown in Table 3.3. The second column shows the number of JSP files in each web application. The third column displays the number of Java Servlet classes, i.e. the Action and ActionForm classes. The value in a bracket is the number of lines of code (Loc) for the Servlet classes in a web application.

3.7.1 Experiment Specification and Performance

Table 3.4 gives an overview of the configurations for every web application.

Application 1 is an open-source real-world web application–Struts Cookbook [15], which contains the web components mostly used in the Struts framework. We want to examine whether the path sequence, i.e. the user actions in each transition, is leaked via traffic analysis. This application was used to evaluate all the functions of SideAuto. For the terminal nodes, in addition to default terminal nodes, parent nodes whose child nodes are all default terminal nodes were also considered as terminal nodes. And a regular expression was used to configure arbitrary values. Tests 1-4 examined application 1 with different configurations.

Application 2 is a simulated online banking system [12], tested in both threat scenarios described in section 3.1.4. In scenario 1, the secret is the sequence of user actions concerning either successful access or failed access of the system. A pair of correct user ID and password was used for the successful authentication, and a pair with wrong information was used for a failed authentication.

Table 3.3: Descriptions of Web Applications

App	Number of JSP pages	Number of Java Servlet classes (Loc)	Description
1. Struts Cookbook	39	22 (1893)	An open-source large real-world web application [15], containing the most used widgets and functions in Struts web applications.
2. Online Banking System	9	6 (822)	A simple online banking system with functions of user login, password modification, money transfer, balance checker and account logout [12]. The user ID is with a fixed length, while the password length is ranging from 8 to 20.
3. Tax Claim System	1	1 (485)	A tax web application [18] similar to the one used in [125]. It is built on Struts 2 framework. A user selects her/his annual gross income, and then the application checks whether the user is eligible for a tax credit.
4. University Identity System	5	8 (235)	The web application [20] for the one shown in Figure 3.2.
5. Nation Information Checker	5	2 (175)	A web application provides a checker about the information including capital, geographical, of a nation [11].

And in scenario 2, the secret is the password length which is between 8 and 20 for a user account. We specified 13 pairs of correct ID/password, each with a unique password length. Tests 5 and 6 examined scenario 1 when communicating via HTTPS and HTTP respectively. Similarly, Tests 7 and 8 examined scenario 2 via HTTPS and HTTP respectively.

Application 3 is built as a tax claim system [18]. According to a user's annual income, the system will judge if the user satisfies to claim taxes. Here we test if the web traffic of a user execution path will leak the user's annual income.

Application 4 is the university identity system [20] displayed in Figure 3.2. By analysing the web traffic, we examine the leakage of user actions of a user path in this web application.

Application 5 provides a checker for the information of 124 countries, with regard to capitals, geographies, populations and areas [11]. The default location is the country the user's IP address belongs to. A user checks one item out of the capital, the geography, the population and the area for the country he wants to examine. We test whether the user actions, particularly the item the user selects, one of capital, geography, population and area, is leaked in this web application.

Table 3.4: Testing Specification and Performance

App	Test	No. of loop	All/One	No. of test cases	Performance (time)	
					Analysing(min)	Quantifying(min)
1	1	0	All	864	7.8	221.0
	2	0	One	84	7.8	4.0
	3	1	All	1746	7.8	365.6
	4	1	One	186	7.8	8.6
2	5	0	All	5	1.1	0.6
	6			5	1.1	0.6
	7	0	All	13	1.1	0.6
	8			13	1.1	0.6
3	9	0	All	30	0.3	1.8
4	10	0	All	6	0.5	0.73
5	11	0	One	4	0.3	0.4
	12	1		20	0.3	1.75
	13	2		84	0.3	10.45

Table 3.4 also shows the performance time, which is partitioned into the analysis time and the quantification time. The analysis time includes the time of constructing a web application's structure and generating test cases. It was performed on Windows XP system with Intel Core i5-2500K CPU @ 3.30GHz and 3.41 GB of RAM. And the quantification time includes the time of executing test cases and quantifying leakage, which were conducted on Windows XP system with Intel Core i3-2310M CPU @ 2.10GHz and 3.41 GB of RAM.

This tool is far from being optimized and ran on a fairly slow architecture. As shown, the analysing time is less than 10 minutes even when generating a large number of test cases in a large web application, e.g. nearly two-thousand test cases generated in application 1 when $l = 1$. However, the quantification time in this case lasted for several hours. In terms of other smaller web applications, the performance generally can be completed in 10 minutes.

3.7.2 Experiment Results

Table 3.5 shows the leaks in each web application in terms of Shannon entropy, and Table 3.6 in terms of min entropy. Values in each cell include the leaks with distributions Up, Ua and Pr respectively, separated by a symbol“;”. When a cell contains only a single figure, it means entropies under all distributions are identical. In column “Leakage(%)”, the values include the uncertainty leaked in entropy and in percentage respectively, where the value inside a “()” is the leakage in percentage.

Application 1

Tests 1-4 display the experimental results for application 1. Now we analyse the results in Table 3.5 in terms of Shannon entropy.

With distribution Up, around 80% of uncertainty is leaked in option All, either $l = 0$ or $l = 1$, while around 30% is leaked in option One. As seen in Table 3.4, nearly 90%

Table 3.5: Testing results in terms of Shannon entropy

App	Test	Shannon Entropy (Up; Ua; Pr)		
		Original	After observation	Leakage (%)
1	1	9.75; 6.77; 3.56	1.82; 0.58; 0.14	7.93(81.3); 6.19(91.4); 3.42(96.0)
	2	9.75; 6.77; 3.56	7.18; 1.08; 0.16	2.57(26.4); 5.69(84.0); 3.40(95.6)
	3	10.77; 7.44; 3.89	2.14; 0.87; 0.06	8.63(80.1); 6.57(88.3); 3.83(98.5)
	4	10.77; 7.44; 3.89	7.11; 1.12; 0.70	3.66(33.9); 6.32(84.9); 3.19(81.9)
2	5	2.32; 2.00; 1.65	1.8; 1.0; 0.75	0.52(22.4); 1.00(50.0); 0.90(54.8)
	6		0.0	2.32(100); 2.00(100); 1.65(100)
	7	3.70	0.0	3.70(100.0)
	8			
3	9	4.91	1.13	3.78(77.0)
4	10	2.58; 2.46; 2.34	0.33; 0.50; 0.46	2.25(87.2); 1.96(79.7); 1.88(80.5)
5	11	8.95	6.95	2.00(22.3)
	12	17.91; 14.43; 14.43	13.89; 10.43; 10.43	4.02(22.4); 4.00(27.7); 4.00(27.7)
	13	26.87; 19.49; 19.49	20.85; 13.91; 13.91	6.02(22.4); 5.58(28.6); 5.58(28.6)

Table 3.6: Testing results in terms of min entropy

App	Test	Min Entropy (Up; Ua; Pr)		
		Original	After observation	Leakage (%)
1	1	9.75; 4.46; 1.14	1.34; 0.30; 0.09	8.41(86.2); 4.16(93.4); 1.05(92.2)
	2	9.75; 4.46; 1.14	3.56; 0.39; 0.09	6.19(63.5); 4.07(91.1); 1.05(92.2)
	3	10.77; 4.46; 0.96	1.61; 0.36; 0.07	9.16(85.0); 4.10(93.2); 0.89(92.6)
	4	10.77; 4.46; 0.96	3.51; 0.41; 0.15	7.24(67.4); 4.05(90.8); 0.81(84.0)
2	5	2.32; 1.00; 0.56	1.32; 0.68; 0.43	1.00(43.1); 0.32(32.0); 0.13(23.2)
	6	2.32; 1.00; 0.56	0.0	2.32(100); 1.00(100); 0.56(100)
	7	3.70	0.0	3.70(100.0)
	8			
3	9	4.91	0.91	4.00(81.5)
4	10	2.58; 2.00; 2.13	0.26; 0.42; 0.37	2.32(89.9); 1.58(79.2); 1.76(82.4)
5	11	8.95	6.95	2.00(22.3)
	12	17.91; 9.95; 9.95	13.59; 7.94; 7.94	4.32(24.1); 2.01(20.2); 2.01(20.2)
	13	26.87; 10.53; 10.53	20.48; 8.53; 8.53	6.39(23.8); 2.00(19.0); 2.00(19.0)

of test cases generated in option All are compressed in option One. This implies that the compressed test cases taking large numbers of omitted execution paths occupy large "volumes" of single equivalence classes.

With distribution Ua or Pr, around 80%–90% of uncertainty is leaked, regardless of option All or One. This suggests that the compressed test cases have low probabilities in distributions Ua and Pr.

The low probabilities under distribution Ua imply that the components in transitions of the omitted execution paths have multiple user inputs, which cause the low probabilities of individual user actions in distribution Ua.

Notice how min-entropy leakage with distribution Up manifests the number of equivalence classes, as shown in Equation 3.6. Table 3.6 suggests that when $l = 0$, there are around 340 equivalence classes in option All in Test 1 and around 73 classes in option One in Test 2. This indicates that nearly 80% of equivalence classes in option All classify merely omitted execution paths in option One. When the loop $l = 1$, there are around 572 and 151 equivalence classes in options All and One respectively, as deduced by the results in Tests 3 and 4 respectively.

Application 2

In application 2, there are five execution paths generated in scenario 1. One test case is when a user fails to log into the system, and the other four consider successful login. After successfully logging, the user selects an option out of the four functions including the transfer, the password modification, the balance checking and the logout.

Test 5 examined the leakage of user actions in a path when the web traffic was encrypted, i.e. under HTTPS, while Test 6 examined under HTTP. The reason of testing under different protocols is to examine the effect of modern encryption in traffic analysis.

Tests 7 and 8 examined the password length via HTTPS and HTTP respectively. There are 13 execution paths generated, each for a user account with a unique password length. We discover that observations about different password lengths are completely distinguishable, even when the communications are encrypted via HTTPS in Test 7.

Therefore, comparing the results between communications via HTTPS and HTTP, it can be seen that traffic encryption via HTTPS in Test 5 is effective in protecting part of user actions, compared with the full leakage of user actions via HTTP in Test 6. However, for the information such as password length, it is fully leaked through traffic analysis, even when the communications are encrypted.

Applications 3 and 4

Application 3 is a tax claiming system, where a user claims her/his annual gross income typed in a list box. This application was built on Struts 2 framework, and it was designed in a similar way as the tax system tested in [125]. There are 30 different options available to be chosen in a list box, each with a gap of 5000, ranging from “Below 10,000”, “10,000-15,000”, ..., “145,000-150,000” to “150,000+”.

This web application is used for comparing SideAuto with Sidebuster proposed in [125]. In SideAuto, 77% of uncertainty about the user income is leaked in terms of Shannon entropy. And the value is very close to that of 73.5% in Sidebuster.

Application 4 is the university identity system whose structure is displayed in Figure 3.2. It can be seen that this web applications takes six execution paths in total.

As suggested by the min-entropy leakages in Test 10 in Table 3.6, with distribution Up, the uncertainty leaked is 2.32, which indicates that there are five equivalence classes generated based on six execution paths. By examining web traffic of each test case, it is discovered that all the user actions are leaked, excluding those in *listbox2* on page *professor.jsp*, i.e. options *senior* and *junior* cannot be identified.

Application 5

After the user checks one item, out of capital, geography, population and area for a country, on the result page there is a link of returning back to the homepage, which forms a loop. We tested this application with loop bounds of $l = 0, 1, 2$ and the leaks are shown in Tests 11, 12, 13 respectively in Tables 3.5 and 3.6.

When $l = 0$, there are 496 $(= 4 * 124)$ execution paths in total, each about an item selected out of the four options for a country. When $l = 1$, i.e. the homepage is accessed twice, there are 246512 $(= 496^2{+}496)$ execution path. And the size soars up to 122270448 $(= 496^3{+}496^2{+}496)$ when $l = 2$. To avoid path explosion when the loop bound increases, we consider that the country information is not leaked. In this case, execution paths for different countries chosen are omitted and the scenarios are considered in option One. Hence the numbers of test cases in terms of different user selections on the capital, the geography, the population and the area are 4, 20 $(= 4^2 + 4)$, and 84 $(= 4^3 + 4^2 + 4)$ when $l = 0, 1, 2$ respectively.

As shown in Table 3.6, the leaks in terms of min entropy with distribution Up suggest the number of equivalence classes. It can be deduced that the numbers of equivalence classes are 4, 16 and 84 when $l = 0, 1, 2$ respectively, which are same as the numbers of test cases. This implies that an attacker can get the item a user selects out of the capital, the geography, the population and the area through traffic analysis.

From the results in Tests 11, 12, 13 in Tables 3.5 and 3.6, it can be seen that test cases are uniformly distributed in both distributions Ua and Pr.

3.8 Discussion

SideAuto is regarded as the first tool towards a fully automated analysis of side-channel leakages in Struts-based web applications. Compared with other automated tools of analysing side-channel leakage in web applications, SideAuto mainly advances on the automated test case generation. Therefore, for a large web application, it can save a lot of time to configure test cases.

Moreover, SideAuto quantifies leaks with different probability distributions in terms of both Shannon entropy and min entropy. These provide a more comprehensive evaluation of leakage in communications, and assist developers in gaining more knowledge about the leakage.

However, the design of SideAuto is still preliminary. SideAuto is only capable of analysing Struts-based web applications, and we evaluate mainly on simulated web applications. It will be a big issue in the real world when the source code of web applications is unachievable or when the web applications are not Struts-based.

The leaks in our experiments indicate that large amount of information is leaked in web applications. However, in this chapter, we do not provide a mechanism to evaluate the precise of the analysed results. In other words, the experimental results present the leakage of user sensitive information, however, the work lacks the experimental validation of the results produced by the analyses. Without the validation, one may ask the questions like "how to prove that the web application really leaks user sensitive information as much as the result presented". Hence the future work can focus on the validation of the experimental results.

Even though this chapter did not provide the validation, the aim is still fulfilled. The purpose of this work is to provide references for developers to discover the possible security threats via traffic analysis, so that they can proceed more thorough analyses to further explore leakages.

On the other hand, one may also ask “do the real-world web applications can leak such large amount of information when the countermeasures, e.g. padding, are applied”. With countermeasures, the leakage of user privacy may be lower. However, as mentioned that the aim of our analyses is to guide the developers to give thorough analyses on the possible security threats.

To explore further the leakage of user privacy in the real world, therefore, we extend our analysis to real-world web applications, as described in the following chapters.

Chapter 4

Side-Channel Vulnerabilities in Real-World Web Applications

4.1 Motivation and Overview

4.1.1 Motivation

The work in this chapter is motivated by the development of SideAuto described in Chapter 3, but here we extend the analysis of Struts-based web applications to more general real-world web applications. Unlike SideAuto which performs white-box analysis on the source code of web applications, this chapter analyses side-channel vulnerabilities through a black-box approach which can be applied to real-world web applications.

In [37], Chapman and Evans build a black-box crawling system on top of Crawljax [3], an open-source tool designed to crawl and test web applications. The system aims to detect side-channel vulnerabilities in web applications. To use the system, a developer specifies which elements are to be interacted with. In other words, test cases of execution paths should be specified manually. In this chapter, we propose an advanced detection system containing automated test case generation.

Furthermore, transitions which appear to have no relation to user sensitive information are examined to analyse possible leakage. Large numbers of studies of side-channel leakages in web applications have already analysed communications explicitly interacting with sensitive information.

For example, Figure 4.1 outlines a scenario showing how user search inputs in an online shopping center can be revealed. In step 1, the communication explicitly transmits a search keyword typed in a search bar to the server. The web traffic generated is indistinguishable from user inputs, so an eavesdropper gains no knowledge of user search keyword from traffic analysis.

Now consider the communication illustrated in step 2 in Figure 4.1. A transition, some-transition away following the transition in step 1, is triggered when a user visits a web page regarding “Today’s deals” on the shopping site. Explicitly, this transition may transmit no information about the user’s search keyword. However, as indicated in

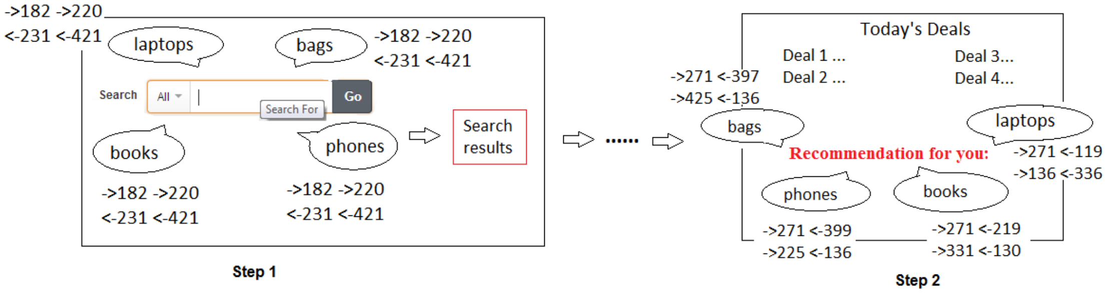


Figure 4.1: Communications explicitly and implicitly transmitting sensitive information

step 2, web traffic in this transition varies depending on search keywords, reflected in the content of "Recommendations", which can actually reveal user search inputs.

Therefore, this work pays attention to communications which implicitly interact with sensitive information.

Backes et al. [24] present a path-aware analysis, where a following transition can assist previous transition in detecting sensitive information. Consider, e.g. a scenario where an attacker observes traffic produced by typing a word in English. Assume that the first typed character is known to be either t or b, and the second character is h. From the second action h, an attacker can deduce that the first action is most likely t, i.e. the sensitive information of user character typed is disclosed. Compare this with the communications in Figure 4.1, where, similarly, sensitive information typed in step 1 can be further revealed by a following transition in step 2. However, in the scenario of [24], the second transition of action h explicitly involves sensitive information, while the transition in step 2 in Figure 4.1 implicitly transmits the confidential data.

Therefore, this chapter focuses on side-channel vulnerabilities particularly from communications not intuitively transmitting sensitive information.

4.1.2 Overview

This chapter proposes a black-box analysis to examine communications concerning both explicit and implicit interaction with sensitive information in real-world web applications. It is shown that sensitive information can really be revealed from communications appearing to have no relation to confidential information. In the test scenarios these leaks can be, sometimes, even more serious than leaks using similar analyses for transitions explicitly involving sensitive information.

Moreover, it is discovered that user fingerprints can also be constructed by traffic analysis, where the identities of users can be revealed.

Furthermore, we use a novel methodology of traffic analysis, motivated by hidden Markov model [27, 28, 29, 30], to construct a *most likely sequence*, which is a "hidden" traffic pattern best associated with a set of observations of a transition.

With the traffic patterns built, packet data are further analysed using a distance based upon Damerau-Levenshtein distance [88] with super transpositions and shifts to construct a probability distribution on observations. Then the guessing probability [82], based on Smith's vulnerability [109], is used to evaluate leakages.

Contributions. In summary, the primary contributions in this chapter include:

1. This chapter demonstrates, via a traffic analysis, that transitions not involving intuitive sensitive information can cause information disclosure, even more seriously than those directly propagating sensitive information.

2. This chapter proposes an automatic crawling system which automatically generates test cases and analyses real-world web applications.

3. This chapter introduces a novel methodology, motivated by the hidden Markov model, and uses a distance optimised based upon the Damerau-Levenshtein distance for traffic analysis. Then the leakage is evaluated by an average worst-case probability of guessing a secret.

The goal of the analysis described in this chapter is to showcase the presence of leakage from communications unexpected to leak user privacy. And the black-box approach proposed provides a framework to assist developers in automatically pinpointing vulnerable transitions in a site and devising efficient countermeasures. Though countermeasures are not concerned in this research, we believe that the results produced by the analysis could be used to mitigate potential leaks.

Next we introduce a fingerprinting model and a threat scenario used in this chapter.

4.2 Fingerprinting Model and Threat Scenario

4.2.1 Fingerprinting Model

To better clarify, this section proposes an extensive fingerprinting model upon that defined in section 2.7.

Formally the transition system is remodelled as $TS = (G, O, A, f, g, MLS, S, Sec)$.

Recall that $G = (N, E)$ denotes a web application. We now partition user actions as $A = A_h \cup A_l$, where A_h are the user actions containing sensitive data and A_l those with no sensitive data.

Transitions of interest in this chapter are partitioned into direct and indirect transitions. They are defined according to the type of user actions which trigger them. A ***direct transition***, expressed as $n_1 \xrightarrow{a \in A_h} n_2 \in E$, is triggered by a user action containing sensitive data. An example of a direct transition is the one originated by an authentication on a login page.

An ***indirect transition*** is triggered by a user action with no sensitive data input, which can be represented by: $n_1 \xrightarrow{a \in A_l} n_2 \in E$. An indirect transition, for example, is performed by clicking on a (non sensitive) link on a page.

We consider paths with more than one transition, but only "one use" of a secret. This means that the path contains only one direct transition in terms of a secret. More complex scenarios concerning one secret been involved several times, giving rise to more than one direct transition of a secret in a sequence, is left for further investigation.

However, there can be more than one indirect transition in an execution path, as any transition executed after the direct transition is regard as an indirect transition. In this chapter, the scenarios considered involve either one indirect transition or a sequence of indirect transitions in terms of one secret value. In the reminder of this chapter, without specific mention, an "*indirect transition*" refers to the indirect transition(s) in terms of a secret value, which follow(s) the direct transition in a scenario. Therefore, in some scenarios an indirect transition e contains more than one indirect transition.

In contrast, an execution path containing both direct and indirect transitions is regarded as a *combined path.*

This chapter examines communications from both direct and indirect transitions individually. It also analyses web traffic from combined paths, to get knowledge of leakage in a web application more comprehensively.

Set Sec is a set of secrets examined in web application G. And set S is a set of secret values, which is associated to a secret $sec \in Sec$. Consider the following example, sensitive information of *username* and *password* for a user account is a value of the secret, i.e. the user identity. An eavesdropper tries to identify a user who is authenticated, out of a database of users, instead of determining the specific values of *username* and *password.* The attacker succeeds in obtaining a user's identity, even though he gains no knowledge about the user name and the password of the user account.

In the basic fingerprinting model defined in section 2.7, function $f : E \rightarrow O$ maps each transition $e \in E$ to an observation $o \in O$. In this chapter, multiple observations are collected for a transition. Hence we define function $f : E \rightarrow P(O)$, which considers that a transition $e \in E$ is associated with a set of observations $O_e \in P(O)$.

Assume each transition associates to a "hidden" traffic pattern which is followed by most observations for this transition. Given a set of observations O_e for a transition e, we define a ***most likely sequence*** mls_e, which is the traffic pattern best matching the observations in set O_e.

Then a function $g : P(O) \rightarrow MLS$ maps each observation set $O_e \in P(O)$ to a most likely sequence $mls_e \in MLS$. The terms "most likely sequence" and "traffic pattern" are used interchangeably throughout this thesis.

4.2.2 Threat Scenario

To clarify consider a following scenario.

A user first performs a user action containing sensitive information on a website. This transition is considered as a direct transition. Then indirect transitions on this website following the direct transition are performed. The execution path contains both direct and indirect transitions is a combined path.

A passive attacker eavesdrops on this web traffic. The traffic sequence deriving from the user's sequence of actions is split into sequences of web traffic for each individual action. Web traffic for each transition is analysed and the aim is to detect which transition leaks sensitive information.

4.3 Test Case Generation

This section describes the black-box crawling system used for automatically generating test cases of execution paths. It is an advance towards the automated generation of test cases, which complements the crawling system proposed by Chapman and Evans [37].

The crawling system we propose builds upon a web driver–SELENIUM [14], which is exploited to crawl web components and which simulates a real-world user using the FIREFOX web browser.

First, mandatory parameters, introduced in the following, are required to be specified in a configuration file. These parameters lead the system to build execution paths with regard to sensitive information.

4.3.1 Specifications

Mandatory Parameters

In a configuration file, the following parameters are mandatory to be configured:

1. URL of the starting page. The system starts analysis from a specific starting node r in a web application G.

2. Sensitive information. For a secret $sec \in Sec$, sensitive information is configured in a format consisting of the attribute value of attribute *ID* for a component (or using attribute *name* if attribute *ID* does not exist), component type (e.g. SELECT means a list box and TEXT means a text box), and possible component values. Components, e.g. a text box, require the specific values, while not those with a predefined set of constants, e.g. a list box.

   The attribute *ID/name* is also used to symbolise the examined secret *sec*, which will be used during test case generation.

3. Maximum length of an execution path. This parameter limits the number of transitions in an execution path. In a real-world web application, an execution path can contain hundreds of transitions. To avoid the explosion of a test case, we apply a limitation on the number of transitions in an execution path.

4. Number of repeated times. Test cases are executed repeatedly, and a set of observations are collected for each test case. In this chapter, we set the number of repeated times as 5.

In addition to these *mandatory parameters* described above, two optional parameters can also be configured.

Optional Parameters

1. Extra user actions

   An *extra user action* is a supplement for an examined transition in an execution path. It is performed on the last page in an execution path. The transition triggered is appended to the end of the execution path and it is not restricted by the maximum length of a path.

When a developer wants to examine a particular indirect transition from a terminal node, which cannot be accessed by simply using a component on the web page, an extra user action is configured using a URL with regard to the web page. The URL is typed into the URL bar directly when the web page is accessed at the same time it is the last page of an execution path.

2. Connecting user actions

   On the other hand, a *connecting user action* can be used to mitigate high overheads. It is performed by interacting with component(s) on a web page.

   The crawling system accesses web components until all the secrets defined are checked. It may examine all the executable components to reach a "must-via" transition for a secret. This process can generate many redundant execution paths and produce high overheads when testing a large web application.

   If this kind of "must-via" transition is known before crawling, it can be specified using a connecting user action, to reduce the costs and improve efficiency.

   However, the configuration of connecting user actions lowers the automation of execution path generation. Hence there is a trade-off between efficiency and automation of test case generation.

Extra user actions and connecting user actions are regarded as *particular user actions* in this chapter. The main differences between them are: (1) a transition trigged by a connecting user action is restricted by the maximum length of an execution path, while not for an extra user action; and (2) a transition triggered by an extra user action is an indirect transition, while it is uncertain for a transition triggered by a connecting user action.

4.3.2 Generation of Execution Paths

Algorithm 1 shows the pseudo code of the algorithm to generate test cases as follows.

The procedure TRANSITION takes a current page u, a current path *path* and the path length *len* as parameters. It is invoked to crawl web components and generate transitions from page u.

When page u is a terminal node or the execution path *path* reaches to the length limit ML on page u (at line 2), extra user actions on page u will be executed for completing execution paths, as indicated at lines 3-4. Then the execution path will be saved if it involves at least one secret value, as expressed at lines 5-6.

Otherwise, if page u contains a secret $sec \in Sec$ required to be examined, sec is removed from set Sec to avoid repeated testing of sec on further pages, as shown at lines 9-10. Then each user action $a \in SA$, either $a \in A_S$ containing sensitive information $s \in S$ for secret sec or a particular user action $a \in PA$, is performed by invoking a function PERFORMING, as shown at line 12.

Algorithm 1 Execution Path Generation

```
Require:
    $Sec$: the set of secrets on a given application $G$
    $u$: the web page visited currently, initialised by the starting node $r$
    $S$: the set of sensitive information in terms of a secret $sec \in Sec$ on page $u$
    $A = PA \cup A_S \cup A_l$: the set of all executable user actions on page $u$. It collects
        particular user actions $PA$, i.e. extra and connecting user actions, user actions
        $A_s$ containing sensitive information $s \in S$, and the remaining user actions $A_l$.
    $SA = PA \cup A_S \subseteq A$
    $ML$: the maximum length of an execution path
    $path$: the generated execution path up to the present page $u$, initialised by node $r$
    $len$: the number of transitions in execution path $path$, initialised by $len = 0$

procedure TRANSITION($u, path, len$)
    if $u$ is a terminal node || $len + 1 > ML$ then
        for extra user action $a \in PA$ do
            PERFORMING($a, u, path, len$)
        if $path$ containing a secret value $s$ then
            Save $path$
    else
        $len \leftarrow len + 1$
        if $\exists$ $sec$ in $Sec$ then
            Remove $sec$ from $Sec$
        for $a \in SA$ do
            PERFORMING($a, u, path, len$)
        if $Sec = \emptyset$ then
            $a \leftarrow$ the first priority $a \in A_l$
            PERFORMING($a, u, path, len$)
        else
            for $a \in A_l$ do
                PERFORMING($a, u, path, len$)
                if $Sec = \emptyset$ then
                    break

procedure PERFORMING($a, u, path, len$)
    $v \leftarrow$ Performing $a$ on $u$
    if $a \in A_S$ with a secret value $s \in S$ then
        $path \leftarrow (path + a + s \rightarrow v)$
    else
        $path \leftarrow (path + a \rightarrow v)$
    TRANSITION($v, path, len$)
```

Function PERFORMING is defined at lines 21-27. Performing a user action a on page u, the function generates a transition which forwards to a new web page v (at line 22). The execution path *path* will be updated by appending the new generated transition, displayed at either line 24 or line 26. When user action $a \in A_S$ contains a secret value s, the sensitive information s will be inserted after user action a in the transition produced (at line 24). So the secret value examined in the transition is recorded. Finally, function TRANSITION is invoked at line 27 to recur the process of crawling and generating transitions for forwarding page v.

After performing all the user action $a \in SA$ on page u, if all the secrets have been accessed already, i.e. $Sec = \emptyset$ (line 13), a *priority policy* of performing a user action is taken to complete the generation of execution paths (see lines 14-15). The first priority is to submit the first encountered HTML form with the first options (or default values) in each component. Otherwise, the first encountered link is clicked.

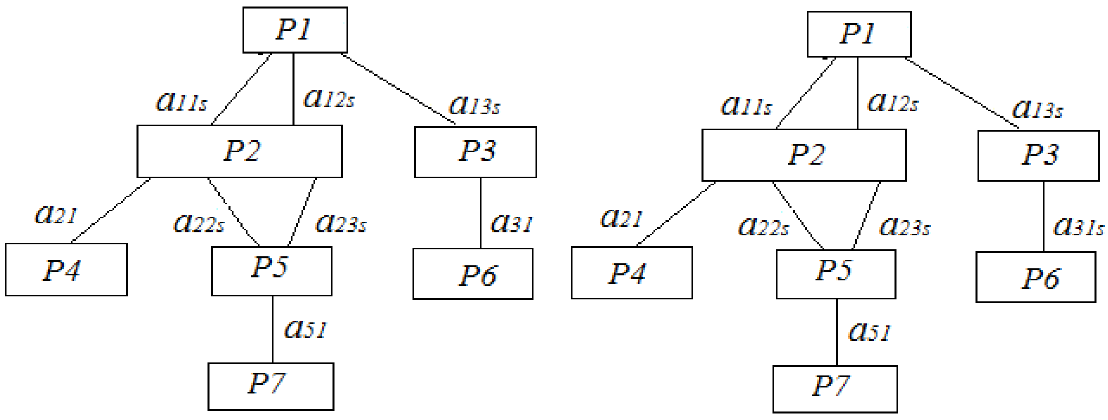
On the other hand, if there has been at least one secret untested yet, i.e. $Sec \neq \emptyset$ (line 16), a brute-force search is conducted. It attempts to access the untested secrets by performing remaining user actions $a \in A_l$ on page u (see lines 17-18). This guarantees that each untested secret can be accessed in case one has to be accessed through a user action $a \in A_l$.

During the brute-force search, when all the secrets have been accessed, i.e. $Sec = \emptyset$ (line 19), it is unnecessary to keep performing remaining user actions. Algorithm 1 stops crawling page u by executing the break command at line 20.

From Algorithm 1, it is shown that a connecting user action is efficient when there are a large space of performable user actions A_l on a web page, which can avoid huge overheads produced during brute-force search. This algorithm only generates execution paths which are related to the secrets the developers want to examine.

We now show how the algorithm works in two examples in Figure 4.2.

$P1, ..., P7$ denote the web pages. A user action with a suffix s, e.g. a_{11s}, means the user action contains the sensitive information.



(a)

(b)

Figure 4.2: Examples of test cases generation

In Figure 4.2a, a_{11s}, a_{12s} and a_{13s} are the user actions on page $P1$, which contain sensitive information for a secret. a_{21}, a_{22s} and a_{23s} are the user actions on page $P2$, where a_{22s} and a_{23s} contain sensitive information for another secret.

Using Algorithm 1, two execution paths are generated by performing user action a_{11s}, where

$$1.\ P1 \xrightarrow{a_{11s}} P2 \xrightarrow{a_{22s}} P5 \xrightarrow{a_{51}} P7 \quad \text{and} \quad 2.\ P1 \xrightarrow{a_{11s}} P2 \xrightarrow{a_{23s}} P5 \xrightarrow{a_{51}} P7$$

At this point, both the secrets on Pages $P1$ and $P2$ have been accessed, so $Sec = \emptyset$. Then an execution path is generated by performing user action a_{12s}. Since $Sec = \emptyset$, the first occurred user action a_{21} on page $P2$ is performed, according to the priority policy.

Therefore, the other two execution paths generated from the graph in Figure 4.2a are:

$$3.\ P1 \xrightarrow{a_{12s}} P2 \xrightarrow{a_{21}} P4 \quad \text{and} \quad 4.\ P1 \xrightarrow{a_{13s}} P3 \xrightarrow{a_{31}} P6$$

However, when analysing leakage of the secret on page $P1$, execution paths would be more reasonable and comparable if path 3 follows

$$3.\ P1 \xrightarrow{a_{13s}} P2 \xrightarrow{a_{22s}} P5 \xrightarrow{a_{51}} P7$$

In this case test cases 1, 2 and 3 only vary depending on the secret values in user actions a_{11s}, a_{12s} and a_{13s}.

Accordingly, we attempt to generate execution paths with minimum distances, i.e. only vary depending on secret values if possible. We develop a *consistency system* to achieve this goal.

4.3.3 Consistency System

The consistency system is designed to instruct execution paths for a same secret to follow the indirect transitions in as similar a way as possible. Indirect transitions, which are first generated on each web page for one secret, are recorded. To mention them easily, these recorded paths are named as *testing paths*.

Take the graph in Figure 4.2a as an example. In terms of the secret on Page $P1$, a testing path $P2 \xrightarrow{a_{22s}} P5 \xrightarrow{a_{51}} P7$ is generated when performing a_{11s}, which is the indirect transition of path 1. In path 2. $P1 \xrightarrow{a_{11s}} P2 \xrightarrow{a_{23s}} P5 \xrightarrow{a_{51}} P7$, indirect transition $P2 \xrightarrow{a_{23s}} P5 \xrightarrow{a_{51}} P7$ is not recorded as a testing path, as there has already been a testing path starting from page B which is related to the secret on Page $P1$. Therefore, execution path 3 by performing a_{12s} will follow testing path "$P2 \xrightarrow{a_{22s}} P5 \xrightarrow{a_{51}} P7$", when arriving on page $P2$.

Therefore, when $SA = \emptyset$, the *priority policy* becomes:

1. Following a related testing path;
2. Performing the first occurred form action with the first options or default values on each component;

3. Clicking the first occurred link.

When generating execution path 4 by performing a_{13s}, the testing path derived from execution path 1 is not applicable, as this execution path does not go through $P2$.

Now consider the web application in Figure 4.2b. When performing a_{11s}, path $P1 \xrightarrow{a_{11s}} P2 \xrightarrow{a_{21}} P4$ is accessed as there has been one secret untested yet, denoted by a_{31s} on page $P3$. The system attempts to crawl all the possible user actions until the untested secret is located. But this path will be abandoned as it does not contain any secret value, since a_{11s} has been accessed in execution paths generated previously.

When performing a_{12s}, the execution path will follow the testing path from page $P2$. The existence of the testing path from page $P2$ suggests that a brute-force search has already been performed on page $P2$ previously. Hence a brute-force search on page $P2$ will not performed this time. Therefore the use of consistency system also avoids unnecessary repeated brute-force search on one same page.

4.3.4 Test Case Construction

When generating execution paths, recall that a user action containing sensitive information is marked by the sensitive information s (line 24 in Algorithm 1). The mark aims to (1) aggregate test cases for a same secret, each for a secret value; and to (2) distinguish the direct transition from the indirect transition in an execution path.

For an execution path containing more than one secret, test cases in terms of each secret will be extracted from the whole execution path, based on the direct transitions of each secret.

For example, consider an execution path generated for the graph in Figure 4.2a: $P1 \xrightarrow{a_{11s}} P2 \xrightarrow{a_{22s}} P5 \xrightarrow{a_{51}} P7$. Then the test case in terms of the secret on page $P1$ will be: $P1 \xrightarrow{a_{11s}} P2 \xrightarrow{a_{22s}} P5 \xrightarrow{a_{51}} P7$, i.e. the whole execution path. And the test case for the secret on page $P2$ is: $P2 \xrightarrow{a_{22s}} P5 \xrightarrow{a_{51}} P7$, starting from the direct transition for the secret on page P_2.

Then test cases with regard to a same secret can be aggregated into one set.

4.4 Data Analysis

4.4.1 Web Traffic Generation

Given a set of test cases, each related to a secret value, we use a web driver–Selenium [14] to simulate user actions in every test case with the Firefox web browser. The configuration file specifies the number of times that the test cases are repeated. In every run, cookies and caches have been cleared before a new test starts. And a set of observations for each test case are collected using Jpcap [10].

Given a set of observations, request packets in each observation can be extracted according to the packet directions, to form a sequence of request packets to be analysed. Currently we do not analyse response packets. Hence each observation means a sequence of request packets.

This work analyses vulnerabilities from direct and indirect transitions separately. Hence we need to obtain the web traffic for individual direct and indirect transitions.

Split of Web Traffic

Figure 4.3 shows the process of obtaining web traffic for each individual transition. Given a set of test cases T for a secret on a page P_1, each test case T_k is a combined path containing both direct and indirect transitions. Each test case is repeatedly executed m times, so a set of observations O_k for test case T_k can be obtained. Then each observation for a combined path is split into two sequences of web traffic for direct and indirect transitions respectively.

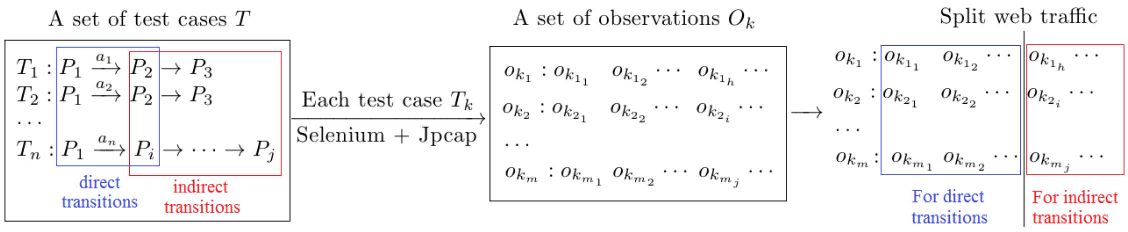

A set of test cases T
$T_1 : P_1 \xrightarrow{a_1} P_2 \rightarrow P_3$
$T_2 : P_1 \xrightarrow{a_2} P_2 \rightarrow P_3$
$\cdots$
$T_n : P_1 \xrightarrow{a_n} P_i \rightarrow \cdots \rightarrow P_j$
direct transitions
indirect transitions
Each test case T_k
Selenium + Jpcap
A set of observations O_k
$o_{k_1} : o_{k_{1_1}} \; o_{k_{1_2}} \cdots o_{k_{1_h}} \cdots$
$o_{k_2} : o_{k_{2_1}} \; o_{k_{2_2}} \cdots o_{k_{2_i}} \cdots$
$\cdots$
$o_{k_m} : o_{k_{m_1}} \; o_{k_{m_2}} \cdots o_{k_{m_j}} \cdots$
Split web traffic
$o_{k_1} : o_{k_{1_1}} \; o_{k_{1_2}} \cdots \; o_{k_{1_h}} \cdots$
$o_{k_2} : o_{k_{2_1}} \; o_{k_{2_2}} \cdots \; o_{k_{2_i}} \cdots$
$\cdots$
$o_{k_m} : o_{k_{m_1}} \; o_{k_{m_2}} \cdots \; o_{k_{m_j}} \cdots$
For direct transitions
For indirect transitions

Figure 4.3: Generating observations

When collecting web traffic for a test case with more than one transition, the following transition will not be triggered until the current one is completed, up to a time-out of, e.g., 20 seconds in this work. And a mark is appended at the end of the web traffic recorded for a direct transition, to separate the web traffic between direct and indirect transitions in a whole traffic sequence.

In real-world attacks, it is feasible for an attacker to split web traffic. For example, there is a typically large time interval between the packets generated between two transitions, relative to the time interval of the packets generated in one transition. Hence by observing the timing of packet generation, an attacker is able to separate web traffic between transitions. Moreover, stable and unique request traffic is generally produced at the beginning of a transition, which can also be used for determining the start point of web traffic from a transition.

It follows that it is reasonable to assume that a real-world attacker is able to separate web traffic in terms of individual direct and indirect transitions.

Overview of Data Analysis

After getting the observation sets for each direct/indirect transition for a secret, we start to analyse the packet data for calculating leakage.

First let us see the overall process of data analysis. Figure 4.4 gives an overview of the process. Given a set of individual direct/indirect transitions E for a secret and the related observation sets O_k for each transition E_k, we use an approach motivated by the hidden Markov model (HMM) to generate the most likely sequences mls_k, i.e. the traffic patterns for the observations in each observation set. Collecting the generated most likely sequences into a set, the leakage of the secret is evaluated using guessing probability [82].

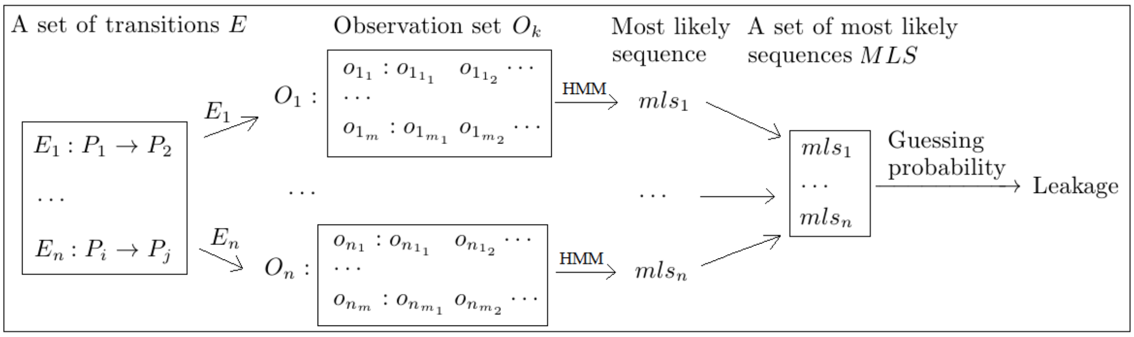

A set of transitions E
$E_1 : P_1 \to P_2$
$\cdots$
$E_n : P_i \to P_j$
E_1
E_n
Observation set O_k
O_1 :
$o_{1_1} : o_{1_{1_1}} \; o_{1_{1_2}} \cdots$
$o_{1_m} : o_{1_{m_1}} \; o_{1_{m_2}} \cdots$
O_n :
$o_{n_1} : o_{n_{1_1}} \; o_{n_{1_2}} \cdots$
$o_{n_m} : o_{n_{m_1}} \; o_{n_{m_2}} \cdots$
HMM
HMM
Most likely sequence
mls_1
mls_n
A set of most likely sequences MLS
mls_1
mls_n
Guessing probability
Leakage

Figure 4.4: Process of analysing data

Now we explain how to normalise a most likely sequence. First define the packet characteristics.

4.4.2 Packet Specification

Given a set of observations O_e, we define the following notations:

- O_e: a set of observations for a transition e;
- o_{e_i}: the i^{th} observation where $o_{e_i} \in O_e$; In a given set O_e, o_{e_i} can also be abbreviated as $o_i \in O_e$;
- $o_{e_{i_k}}$: the packet at position k in o_{e_i}. Similarly, it can be abbreviated as $o_{i_k} \in o_i$. It is presented by a pair $(value, k)$, where $value$ is the packet size and k is the position the packet is located in this observation. The use of packet location is to deal with a packet-disorder issue introduced later.

Packet Disorder

In the real world, an issue of network packet out of order often occurs. When a reliable in-order delivery of packets is required, for instance when using Transmission Control Protocol (TCP), a transmitter resends the packets which are not received properly. This may cause packets disordered, e.g. a retransmitted packet is observed after a packet where it should be observed before the packet in a proper order.

This research analyses TCP packets, where sequence numbers of packets can be extracted to recover a correct order. However, to provide an extensive mechanism which can analyse traffic even when the sequence numbers of packets are not available, this work proposes a stronger analyser with defining a *shift operation with displacement range*, which, to a great extent, mitigates possible errors caused by disordered packets.

Definition 17 (Shift with Displacement Range *d*) *Given a packet located at position k in an observation $\overrightarrow{q}$, it can be shifted up to d-position forward or backward away from the current position k. This aims to recover the original position where the packet is actually located in a proper order. After shifting the sequence becomes sequence $\overrightarrow{q'}$.*

Displacement range is used to simulate the degree of packet disorder caused by a packet transmission error. If there is a packet retransmitted and observed in an incorrect

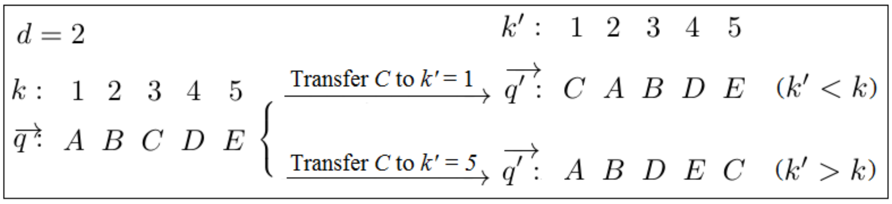

$d = 2$
k : 1 2 3 4 5
$\overrightarrow{q}$: A B C D E
Transfer C to $k' = 1$
k' : 1 2 3 4 5
$\overrightarrow{q'}$: C A B D E $(k' < k)$
Transfer C to $k' = 5$
$\overrightarrow{q'}$: A B D E C $(k' > k)$

Figure 4.5: Shift of a packet

order, it can be moved back to its correct position if the disorder happens within the displacement range.

So an appropriate setting of the displacement range is important for the precision of a most likely sequence. If the value is too small, it will miss the disordered packets outside the displacement range. But when the value is too large, there may be an incorrect judgement for those which have already been located at correct position. Thus the value can be decided according to, e.g. sliding window size. In this work, the displacement range is set to $d = 2$.

For example, Figure 4.5 shows an example of a shift operation. With displacement range $d = 2$, packet C at position 3 in sequence $\overrightarrow{q}$ can be shifted to position $k' = 1$ or $k' = 5$.

The shift of the packet at position k causes the packets at positions $[k', k-1]$ $(k' < k)$ or $[k+1, k']$ $(k' > k)$ in sequence $\overrightarrow{q}$ to be also moved to new positions of $[k'+1, k]$ or $[k, k'-1]$ in sequence $\overrightarrow{q'}$. After n times of shift, it must ensure that all the packets shifted to new positions are within displacement range from the original positions in sequence $\overrightarrow{q}$.

Indistinguishable Packets

In an ideal situation, observations for a transition are identical in a small time period. But in fact, external factors, such as performing time and session IDs, may result in a slight fluctuation of packet sizes between multiple observations. Therefore, *indistinguishable packets* are proposed for reducing the probable impacts on packet sizes among different observations from external factors.

Definition 18 (Indistinguishable Packets) *Packets from different observations are considered as indistinguishable, if their locations are within displacement range and their sizes a, b satisfy:*

$$|a - b| \leq tp \cdot \max(a, b),$$

where $tp \in [0, 0.5]$ is defined as a ***threshold of indistinguishable packets****.*

Moreover, two traffic sequences are determined to be ***indistinguishable***, if every pair of the packets located at a same position between sequences are indistinguishable. By default, the threshold of indistinguishable packets is set by $tp = 0.1$.

Next we describe how to construct the most likely sequence mls_e for transition e, i.e. $g(O_e) = mls_e$.

4.4.3 Modelling Packet Data

In a hidden Markov model (HMM), states are not observable while outputs depending on the states are observable. Similarly, we consider the observations in a set are the outputs, and the traffic pattern generated from the observations can be deemed as a sequence of hidden states. Hence we use the notations defined in a HMM to model the packet data in an observation set.

Given an observation set O_e of a transition e, a *position-dependent* model can be denoted as follows:

- K = number of positions, i.e. the maximum length among the numbers of packets in observations

- N_k = number of states at position k

- $ST_k = \{st_{k_1}, ..., st_{k_{N_k}}\} \in ST$ = a set of possible hidden states at position k
  Each state $st_{k_i} \in ST_k$ is unique and originated from the size of a packet observed at position k in an observation. This set may contain states related to the potential disordered packets, which are not observed but within displacement range from position k. The final confirmed state at position k is denoted by st_k.

- M_k = number of outputs at position k

- $OP_k = \{op_{k_1}, op_{k_2}, ..., op_{k_{M_k}}\} \in OP$ = a set of output at position k
  It collects packets that can be observed at position k in each observation $o_{e_i} \in O_e$, expressed by packet sizes. Similar to the states in a state set, outputs in this set may also include those probably disordered but within displacement range from position k.

- $IP = \{ip_1, ..., ip_{N_1}\}$ = initial state distribution at position 1
  ip_i denotes the initial probability of state st_{1_i}.

- $C_k = \{c_k(st_{k_i}, st_{(k+1)_j})\} \in C$ = transition probabilities between states at position k
  It is a $N_k \times N_{k+1}$ matrix, and $c_k(st_{k_i}, st_{(k+1)_j})$ denotes the probability of transferring from state $st_{k_i} \in ST_k$ at position k to state $st_{(k+1)_j} \in ST_{k+1}$ at position k+1.

- $B_k = \{b_k(op_{k_t}|st_{k_i})\} \in B$ = output probabilities at position k
  It is a $N_k \times M_k$ matrix, and $b_k(op_{k_t}|st_{k_i})$ denotes the probability of observing output $op_{k_t} \in OP_k$ in state $st_{k_i} \in ST_k$.

Similar to a HMM, this model can be characterised by a set of parameters $\mu = \{IP, C, B\}$. A most likely sequence can be regarded as a sequence of most likely states in a format as

$$mls_e = (st_1, st_2, ...)$$

Figure 4.6 shows the construction of a most likely sequence. The maximum length of observations, i.e. the maximum number of packets among observations is K. Output sets and state sets are obtained based on the packets at each position. And output probabilities and transition probabilities are calculated to get the hidden states at each position.

Details of how to consider disordered outputs within displacement range and how to obtain the most likely states at each position are described in the following section.

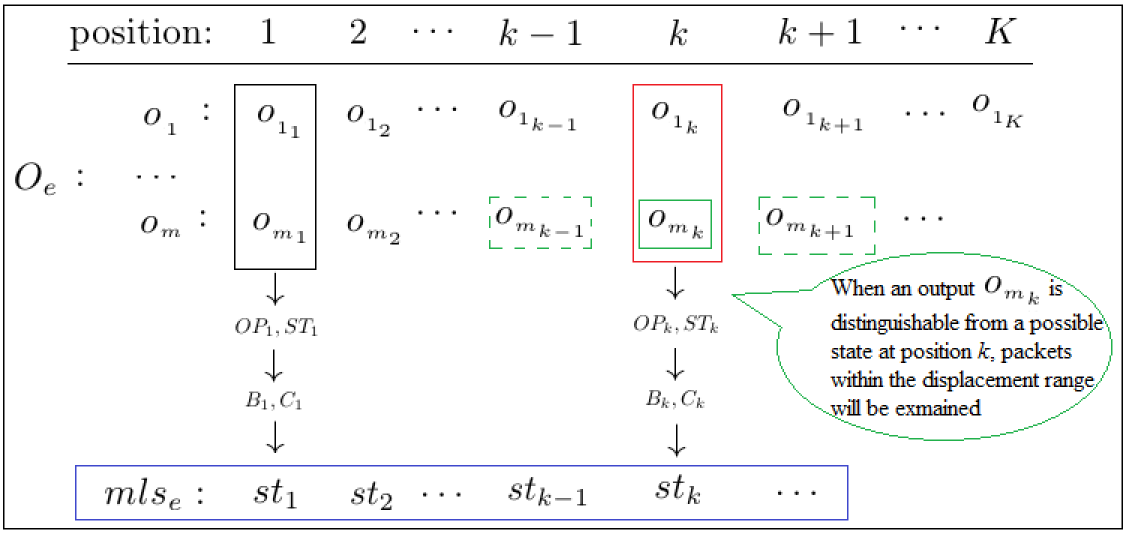

position: 1 2 ··· $k-1$ k $k+1$ ··· K
O_e :
o_1 : o_{1_1} o_{1_2} ··· $o_{1_{k-1}}$ o_{1_k} $o_{1_{k+1}}$ ··· o_{1_K}
···
o_m : o_{m_1} o_{m_2} ··· $o_{m_{k-1}}$ o_{m_k} $o_{m_{k+1}}$ ···
OP_1, ST_1
B_1, C_1
OP_k, ST_k
B_k, C_k
When an output o_{m_k} is distinguishable from a possible state at position k, packets within the displacement range will be exmained
mls_e : st_1 st_2 ··· st_{k-1} st_k ···

Figure 4.6: Constructing a most likely sequence by a HMM

4.4.4 Most Likely Sequence

In general, the generation of a most likely sequence can be related to a classic Question 3 in HMM, as described in section 2.6. It concerns how to generate a corresponding state sequence which best explains observations.

For the traditional solution in HMM, the Baum-Welch algorithm [30] is performed until the resulting probabilities converge satisfactorily, which maximises $\mu = \{IP, C, B\}$. Then the Viterbi algorithm [58] is performed to get a sequence of most likely states which has the maximum probability.

However, we do not use this approach to generate a most likely sequence. Our model is position-dependent, i.e. the states and outputs at each position can be different, and also disordered packets are considered. If the approach of HMMs is used to calculate the most likely sequence in our work, a large number of combinations of states need to be considered and then a huge overhead can be generated.

Instead of optimising $\mu = \{IP, C, B\}$, this work only aims to get the most likely sequence. Hence it is unnecessary to use the approach in HMMs to optimise $\mu = \{IP, C, B\}$ and then derive the sequence of most likely states.

Accordingly, we develop an analysis motivated by HMM, to simplify the analysis. Our proposed approach is called as the "*confirm and run*" approach, to produce a sequence of most likely states.

Specification for Generating a Most likely Sequence

To generate a most likely sequence, given a set of observations O_e, it is required to (1) determine the length of a most likely sequence, i.e. the number of hidden states; and to (2) identify indistinguishable packets in a set.

1. Calculating the length of the most likely sequence LEN_e.

   First define a notion of *indistinguishable lengths.*

   Definition 19 (Indistinguishable lengths) *Given a set of lengths of observations $L(O_e)$, i.e. the numbers of request packets in each observation, two observations are of indistinguishable lengths if the distance between their lengths is not larger than 10 percent of the longer length. It can be expressed by:*

   $$|L(o_{e_i}) - L(o_{e_j})| \leq 0.1 \times \max(L(o_{e_i}), L(o_{e_j}))$$

   If at least 50% of observations have the same length l, the length of the most likely sequence is $LEN_e = l$. Otherwise, if there at least 50% of observation lengths are mutually indistinguishable, the length with the highest probability among the indistinguishable lengths is to be LEN_e. If more than one indistinguishable length has the same highest probability, the one nearest to the average length will be LEN_e.

   Otherwise, when less than 50% of observation lengths are mutually indistinguishable, the average of the indistinguishable lengths which account for the largest proportion of observations will be LEN_e. On the other hand, if all the lengths are distinguishable, the shortest length will be the length LEN_e.

2. Re-organising the state and output sets ST and OP.

   We define indistinguishable packets in a set.

   When building a most likely sequence, indistinguishable packets are considered as the same packet. Each item in a state/output set symbolises a packet size. Indistinguishable sizes in a set will be integrated into one single size. By comparing each two items in a set, those indistinguishable are partitioned into a group. Then a representative of each group is identified, which is the size closest to the average size of the indistinguishable sizes in the group.

   For example, given a set of outputs: $\{190, 195, 200, 100\}$ and a threshold of indistinguishable packets $Tp = 0.1$, sizes 190, 195 and 200 are indistinguishable, and the representative of this group is 195. The set after reorganising becomes $\{195, 100\}$.

Next we describe how to construct the most likely sequence using the *confirm and run* approach.

Confirm and Run

In this approach, the states at each position are confirmed consecutively, starting from the first until the final in the most likely sequence.

Given a set of observations O_e, the process of building a most likely sequence is outlined as follows:

1. First, build the output/state set at position 1. Each output/state is derived from a unique size of a packet o_{i_1} in an observation $o_i \in O_e$ at position 1.

   The first state st_1 is determined with the highest probability of outputs indistinguishable from that state.

2. After confirming a state st_k at position k, if an output op_{k_i} at position k is distinguishable from st_k, the packet of the output in the observation is either a redundant packet or to be disordered if it is located within displacement range from position k. This packet is regarded as an *unconfirmed packet*. It will be confirmed when it is identified to be disordered, i.e. indistinguishable from a state at a position within displacement range from position k. An example will be used to further explained this.

3. From position 2, disordered packets will be considered. Possible states $st_{k_j} \in ST_k$ at position k are originated from either the packets observed at position k in observations or from the unconfirmed packets at previous positions within displacement range.

   For each possible state st_{k_i}, we calculate the possible highest probability of outputs indistinguishable from state st_{k_i}.

   When an output observed at position k is distinguishable from st_{k_i}, first we examine if there unconfirmed packets within displacement range in the observation are indistinguishable from st_{k_i}. If not, subsequent packets within displacement range in this observation will be checked, to find a packet indistinguishable from st_{k_i}, which is probably disordered from position k.

   A set OP'_{k_i} is used to collect these packets within displacement range identified to be indistinguishable from st_{k_i}, when the outputs observed at position k are distinguishable from st_{k_i}.

   Hence the possible highest output probability of observing outputs indistinguishable from state st_{k_i} is defined by:

   $$b_k(st_{k_i}) = \frac{|\{j|op_{k_j} = st_{k_i}\}| + |OP'_{k_i}|}{|O_e| + |OP'_{k_i}|}, \tag{4.1}$$

   where $|\{j|op_{k_j} = st_{k_i}\}|$ is the number of outputs at position k indistinguishable from state st_{k_i}. And $|OP'_{k_i}|$ is the number of packets within displacement range from position k indistinguishable from st_{k_i}.

   Figure 4.7 draws the process of identifying the state at position k. When calculating output probability $b_k(st_{k_i})$, an output o_{m_k} is distinguishable from state st_{k_i}. Packet $o_{m_{k-1}}$ is unconfirmed and it is within displacement range from position k. It will be first checked to see if it is indistinguishable from st_{k_i}. If not, then the following packets within displacement range, e.g. $o_{m_{k+1}}$, will be examined. The packets within displacement range under examination are put inside the green squares. The packet

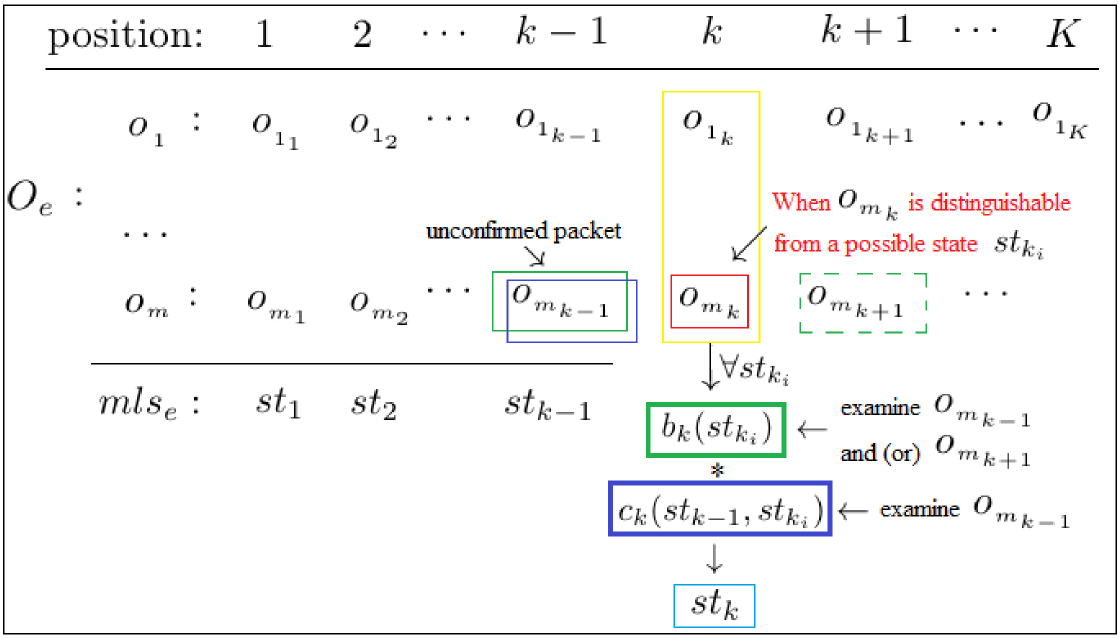

position: 1 2 ··· $k-1$ k $k+1$ ··· K
O_e :
o_1 : o_{1_1} o_{1_2} ··· $o_{1_{k-1}}$ o_{1_k} $o_{1_{k+1}}$ ··· o_{1_K}
···
o_m : o_{m_1} o_{m_2} ··· $o_{m_{k-1}}$ o_{m_k} $o_{m_{k+1}}$ ···
unconfirmed packet
When o_{m_k} is distinguishable from a possible state st_{k_i}
$\forall st_{k_i}$
mls_e : st_1 st_2 st_{k-1}
$b_k(st_{k_i})$
examine $o_{m_{k-1}}$ and (or) $o_{m_{k+1}}$
*
$c_k(st_{k-1}, st_{k_i})$
examine $o_{m_{k-1}}$
st_k

Figure 4.7: Confirming the state st_k at position k

inside a square with solid lines means it will certainly be checked, while that inside a square with dotted lines means it is probably checked.

4. Then we calculate a transition probability from the confirmed previous state st_{k-1} to a possible state st_{k_i}, i.e. $c_k(st_{k-1}, st_{k_i})$. The probability is decided by the number of indistinguishable outputs including those observed at position k and the those unconfirmed in previous positions within displacement range from position k.

   We use a set OP''_k which collects all the outputs of the unconfirmed packets within displacement range from position k, regardless they are distinguishable or indistinguishable from the states.

   Hence a transition probability is defined by:

$$c_k(st_{k-1}, st_{k_i}) = \frac{|\{j|op_{k_j} = st_{k_i}\}| + |\{j|op''_{k_j} = st_{k_i}\}|}{|O_e| + |OP''_k|}, \tag{4.2}$$

   where $|\{j|op''_{k_j} = st_{k_i}\}|$ is the number of outputs for the unconfirmed packets indistinguishable from state st_{k_i}.

   As shown in Figure 4.7, when calculating the transition probability $c_k(st_{k-1}, st_{k_i})$, we only examine the unconfirmed packets with displacement range, which are now depicted in blue squares.

5. Then a state $st_{k_i} \in ST_k$ out of all the possible states, which has the highest product $b_k(st_{k_i}) \times c_k(st_{k-1}, st_{k_i})$, will be the state st_k at position k.

6. The process of steps 2-5 is repeated until the last state in the most likely sequence is

confirmed. Finally, the most likely sequence generated is:

$$mls_e = (st_1, st_2, ..., st_{LEN_e}).$$

Next we take an example to illustrate the process of generating a most likely sequence.

4.4.5 An Example: Generating a Most Likely Sequence

Given a set of observations O_e with displacement range $d = 2$:

$$\begin{aligned} O_e = \{o_1 &= (200,1)(101,2)(50,3)(80,4)(120,5) \\ o_2 &= (200,1)(100,2)(50,3)(120,4)(200,5),(80,6) \\ o_2 &= (100,1)(200,2)(80,3)(120,4)(50,5)\} \end{aligned}$$

the length of the most likely sequence is determined as $LEN_e = 5$.

At position 1, the state set and the output sets are $ST_1 = OP_1 = \{200, 100\}$. 200 is derived from the packets $(200, 1)$ in observations o_1 and o_2, and 100 is from the packet $(100, 1)$ in observation o_3. The initial probabilities are calculated by $ip_1(st_{1_1} = 200) = \frac{2}{3}$ and $ip_2(st_{1_2} = 100) = \frac{1}{3}$. Hence $st_1 = 200$, and packet $(100, 1)$ in observation o_3 is unconfirmed.

At position 2, $ST_2 = OP_2 = \{100, 200\}$, as packet $(101, 2)$ in observation o_1 is regarded as same as packet $(100, 2)$.

The possible highest output probability for state st_{2_1} is $b_2(st_{2_1} = 100) = \frac{2+1}{3+1} = \frac{3}{4}$, where there are two outputs observed at position 2 in observations o_1 and o_2 respectively and one unconfirmed packet $(100, 1)$ in observation o_3 are indistinguishable from st_{2_1}.

And $b_2(st_{2_2} = 200) = \frac{1}{3}$, as there are not potential disordered packets, within displacement range in observations o_1 and o_2 indistinguishable from st_{2_2}.

For the transition probabilities, $c_2(st_1, st_{2_1}) = c_2(200 \rightarrow 100) = \frac{3}{4}$ as unconfirmed packet $(100, 1)$ from observation o_{e_3} is indistinguishable from state st_{2_1}. And $c_2(st_1, st_{2_2}) = c_2(200 \rightarrow 200) = \frac{1}{4}$. Hence the final state at position 2 is $st_2 = 100$. And the actual positions of packets $(100, 1)$ and $(200, 2)$ in observation o_3 are shifted.

Similarly, the final state at position 3 can be obtained as $st_3 = 50$. And the packet $(80, 3)$ in observation o_3 is unconfirmed.

At position 4, the possible states are $\{st_{4_1} = 80, st_{4_2} = 120\}$. When $st_{4_1} = 80$, the packets $(120, 4)$ observed in observations o_2 and o_3 respectively are distinguishable from the state. Then the packets within displacement range in these two observations are checked, and it is found that packet $(80, 6)$ in observation o_2 and packet $(80, 3)$ in observation o_3 are indistinguishable from st_{4_1}. Hence the possible highest output probability for state st_{4_1} is: $b_4(st_{4_1} = 80) = \frac{1+2}{3+2} = \frac{3}{5}$.

Similarly, it can be obtained that $b_4(st_{4_2} = 120) = \frac{3}{4}$ as packet $(120, 5)$ within displacement range in observation o_1 is considered.

For transition probabilities, $c_4(st_3, st_{4_1}) = c_4(st_3, st_{4_2}) = 0.5$, as unconfirmed packet (80,3) in o_3 is indistinguishable from state $st_{4_1} = 80$. As a result, $st_4 = 120$.

At position 5, although there are not any packets observed with an output of 80, unconfirmed packets $(80, 4)$ and $(80, 3)$ in observations o_1 and o_3 respectively are located within displacement range from position 5. Thus, 80 is considered as a possible state/output, and it is finally obtained that $st_5 = 80$.

According, the most likely sequence of transition e is

$$mls_e = (200, 100, 50, 120, 80)$$

After generating the most likely sequences for each set of observations, we now evaluate leakage of the secret using the guessing probability [109, 82]

4.5 Quantification of Leaks

Figure 4.8 gives an overview of the quantification of leakage. Given a set of transitions E and the observation sets, the most likely sequences for each transition e can be generated, as described in section 4.4.4.

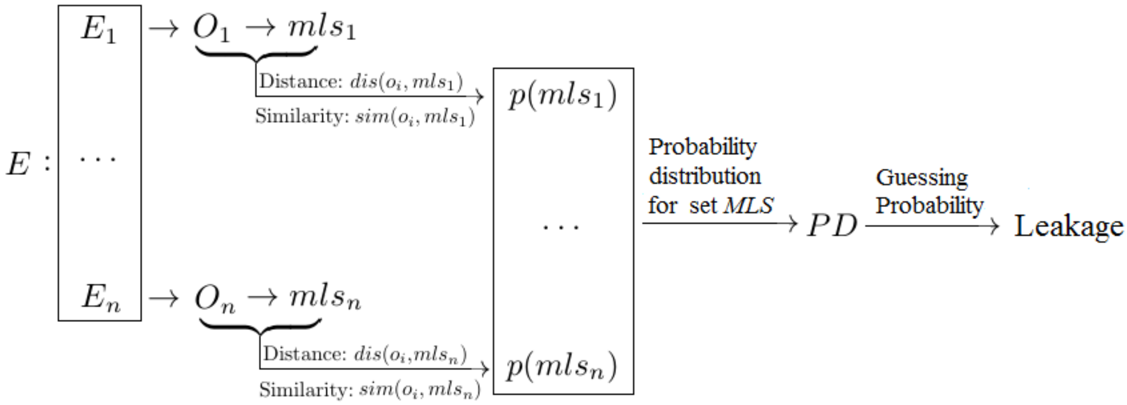

E :
E_1
. . .
E_n
$\rightarrow O_1 \rightarrow mls_1$
Distance: $dis(o_i, mls_1)$
Similarity: $sim(o_i, mls_1)$
$\rightarrow O_n \rightarrow mls_n$
Distance: $dis(o_i,mls_n)$
Similarity: $sim(o_i, mls_n)$
$p(mls_1)$
. . .
$p(mls_n)$
Probability distribution for set MLS
PD
Guessing Probability
Leakage

Figure 4.8: Quantifying leakage

For every observation set O_e, we calculate the *distance* between each observation $o_i \in O_e$ and most likely sequence mls_e, optimised from the *Damerau-Levenshtein Distance* [88]. According to the distance generated, *similarity* between each observation and the most likely sequence can be calculated. Then we calculate the probability of observations in O_e which follow the most likely sequence mls_e, i.e. $p(mls_e)$. A probability distribution PD in terms of observations for transitions in set E following the generated most likely sequences is built, and then the leakage is calculated using guessing probability [109, 82].

4.5.1 Probability of a Most Likely Sequence of a Transition

First, we evaluate the distances between each observation $o_i \in O_e$ and the most likely sequence mls_e.

Distance between Two Sequences

We define a distance based upon the *Damerau-Levenshtein Distance* [88] to calculate the distance between two sequences. It evaluates the minimum number of operations of shift, insertion, deletion, substitution, and transposition of packets, to transform a sequence to be indistinguishable from another sequence.

Definition 20 (Distance between two sequences) *A traffic sequence $\overrightarrow{a}$ can be obtained from traffic sequence $\overrightarrow{b}$ through at least n operations of shift, insertion, deletion, substitution of packets, and transposing two packets within displacement range. The distance between sequences $\overrightarrow{a}$ and $\overrightarrow{b}$ is defined as*

$$dis(\overrightarrow{b}, \overrightarrow{a}) = \min(n, |\overrightarrow{a}|), \tag{4.3}$$

where $|\overrightarrow{a}|$ is the number of packets in sequence $\overrightarrow{a}$.

In this definition, indistinguishable packets are the same packet, for example $dis(QPP', QP'P) = 0$ iff P and P' are indistinguishable.

In a traditional Damerau-Levenshtein distance, a sequence is transformed into another sequence through insertion, deletion, substitution of packets and transposition between two adjacent packets. Here we extend the transposition to packets not adjacent. The packets can be transposed as long as they are within displacement range. So a traditional Damerau-Levenshtein distance can be deemed with displacement range $d = 1$.

Figure 4.9 shows the examples of the distance between two sequences. When displacement range $d = 3$, the distance $dis(\overrightarrow{b}, \overrightarrow{a}) = 3$. And when $d = 2$, the distance $dis(\overrightarrow{b}, \overrightarrow{a}) = 4$.

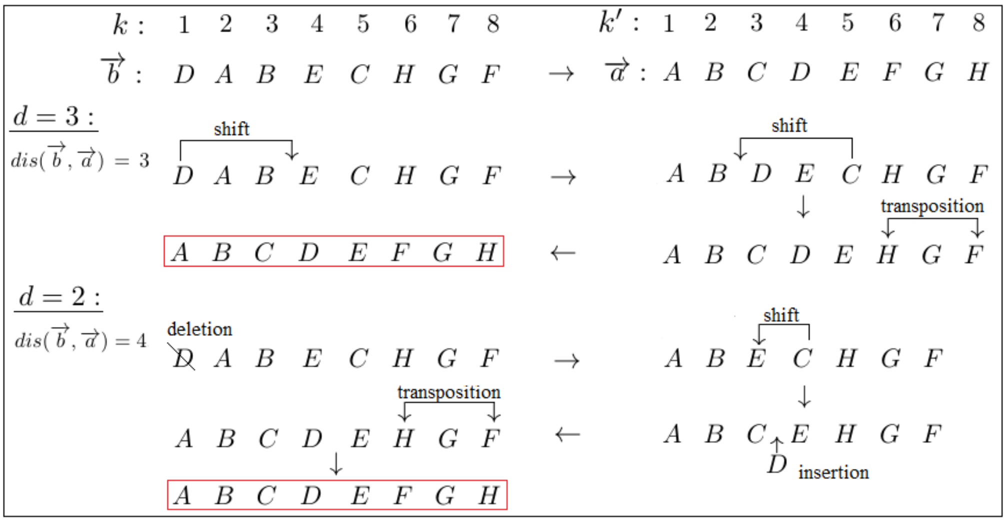


Figure 4.9: Examples of distances between two sequences

We regard the process of calculating the distance between two sequences as a *Damerau-Levenshtein process*, and sequence $\overrightarrow{a}$ is indistinguishable from sequence $\overrightarrow{b}$ after the process.

Similarity between Two Sequences

Now we define a *similarity* to measure the similarity between the two sequences.

Definition 21 (Similarity) *Given an observation $o_i \in O_e$ and a most likely sequence mls_e, similarity between o_i and mls_e is defined by*

$$sim(o_i, mls_e) = 1 - \frac{dis(o_i, mls_e)}{|mls_e|}, \tag{4.4}$$

where $|mls_e|$ is the number of packets in sequence mls_e, and $dis(o_i, mls_e)$ is the distance using Equation 4.3.

Expression $\frac{dis(o_i, mls_e)}{|mls_e|}$ transforms the distance into the percentage of packets mismatched between the two sequences, such that $0 \leq sim(o_i, mls_e) \leq 1$. The similarity examines the degree of an observation o_i following the most likely sequence mls_e.

With similarity $sim(o_i, mls_e)$, we can determine if observation o_i follows most likely sequence mls_e by defining a ***threshold of similarity***–Tm $(0 < Tm \leq 1)$.

Given the threshold of similarity Tm, observation o_i is considered conforming to most likely sequence mls_e if and only if $sim(o_i, mls_e) \geq Tm$.

Probability of a Most Likely Sequence

For a transition e, we calculate the probability of an observation following the most likely sequence depending on the similarities.

Definition 22 (Probability of a most likely sequence) *Given a set of observations O_e for a transition e and the most likely sequence mls_e, the probability of most likely sequence mls_e is defined as:*

$$p(mls_e) = \frac{M}{|O_e|} \tag{4.5}$$

where M is the number of observations in O_e following mls_e, i.e. the observation o_i satisfies $sim(o_i, mls_e) \geq Tm$.

This probability $p(mls_e)$ represents the probability of an observation from transition e following traffic pattern mls_e.

Now we define a ***threshold of most likely sequences***–Tt $(0.5 < Tt \leq 1)$, which assesses if a most likely sequence is sufficiently close to the observations. In other words, to determine if the produced most likely sequence can really be the traffic pattern of observations for the transition.

If $p(mls_e) \geq Tt$, most likely sequence mls_e is confirmed to be the traffic pattern of transition e. Otherwise, transition e is regarded as generating *random web traffic* which does not follow any traffic patterns. Due to the “one → one” mapping between a transition and a traffic pattern, mls_e is still regarded as the traffic pattern for transition e. However, the probability of mls_e is reviewed to

$$p(mls_e) = 0 \tag{4.6}$$

When $p(mls_e) \neq 1$, it is suggested that there are observations for transition e which do not conforming to the most likely sequence mls_e. In other words, *uncertain observations* may also be generated from transition e. Therefore, we define the probability of uncertain observations which do not follow mls_e.

Definition 23 (Uncertainty) *The uncertainty of observations for a transition e, i.e. the probability of generating random web traffic, is defined as:*

$$p_{uncer}(mls_e) = 1 - p(mls_e) \tag{4.7}$$

After producing the probabilities of each most likely sequence for each transition, next we normalise them to build a probability distribution in terms of the most likely sequences.

4.5.2 Probability Distribution in terms of a set of Traffic Patterns

Given a set of most likely sequences derived from the observation sets for a secret, and the probabilities of each most likely sequence, we build a probability distribution. The probability distribution represents the probabilities of observations for a set of transitions.

Given a set of most likely sequences MLS related to a set of transitions E, with the uniform distribution on set E, the probability distribution in terms of the most likely sequences MLS is denoted by $PD = P' \cup PU'$, of which

$$P' = \{\ p'(mls_e) = \frac{1}{|E|} * p(mls_e) :\ \forall\ mls_e \in MLS\ \} \tag{4.8}$$

and

$$PU' = \{pu'(mls_e) = \frac{1}{|E|} * p_{uncer}(mls_e) = \frac{1}{|E|} * (1 - p(mls_e)) :\ \forall\ mls_e \in MLS\ \} \tag{4.9}$$

where $\frac{1}{|E|}$ is probability of transition $e \in E$, as the uniform distribution on set E is considered.

Set P' collects the probabilities of observations following most likely sequences while PU' collects the uncertainties of observations which do not follow traffic patterns, such that

$$\sum_{p'(mls_e) \in P'} p'(mls_e) + \sum_{pu'(mls_e) \in PU'} pu'(mls_e) = 1$$

In this probability distribution, probability $p(mls_e)$ now is equivalent to the conditional probability of an observation following the most likely sequence $mls_e \in MLS$, given the knowledge that it is from transition e.

And probability $p'(mls_e)$ is regarded as the joint probability of an observation originated from transition $e \in E$ and following the most likely sequence $mls_e \in MLS$. And $pu'(mls_e)$ is the joint probability of an observation coming from transition e but not following traffic pattern mls_e.

In this research, we consider that an observation for a transition e_i will not follow the most likely sequence for another transition e_j, where $e_i \neq e_j$.

4.5.3 Pseudo-Partition

Now we define a ***Pseudo-Partition***, which is used to classify the most likely sequences.

Definition 24 (Pseudo-Partition) *A Pseudo-Partition of a set S is a set of subsets of S whose union is S.*

Given a set of transitions E and a set of most likely sequences MLS, where $g : f(E) \rightarrow MLS$, we build a pseudo-partition X of set MLS.

For a most likely sequence $mls_e \in MLS$ whose $p(mls_e) = 0$, it is classified in a set $X_{random} \in X$ named as the `random class`.

For a most likely sequence mls_e with $p(mls_e) \neq 0$, it is added to a `regular class` $X_i \in X$, if and only if

$$\forall \; mls_k \in X_i, \;\; dis(mls_e, mls_k) = 0$$

That is to say, every two sequences in class X_i are mutually indistinguishable. Most likely sequence mls_e in the regular class symbolises the observations following mls_e.

Moreover, if $0 < p(mls_e) < 1$, in addition to a regular class X_i, mls_e will also be added to set X_{random}. Now mls_e in the random class symbolises the uncertain observations which come from transition e but do not follow mls_e.

As can be seen, a pseudo-partition X is not a partition or equivalence relation where a most likely sequence can only be classified into one single class. However in the pseudo-partition, a most likely sequence can be classified into both a regular class and a random class, to represent regular observations following the most likely sequence and the random observations respectively.

In this sense, not only observations which follow traffic patterns, but also those inconsistent to traffic patterns are examined. This leads to a wider analysis range of observations.

For a pseudo-partition on most likely sequences, if we consider the most likely sequences as observations, the pseudo-partition can be actually regarded as a partition on the observations. In this sense, one observation is classified into one and only one block, either the random class or a regular class. And the observations in a regular class follow the indistinguishable most likely sequences.

Next, we show how the leakage of a secret can be evaluated based on a pseudo-partition.

4.5.4 Guessing Probability

As defined in Definition 16 in section 2.1.3, a guessing probability is an average worst-case probability of guessing a secret correctly in one try. The definition is proposed in [82], based on Smith's vulnerability [109].

In this work, we use the guessing probability to measure side-channel leakage, based upon a pseudo-partition.

Definition 25 (Guessing Probability) *Given a pseudo-partition X on a set of most likely sequences MLS regarding a secret sec, and a probability distribution for set MLS*

$PD = P' \cup PU'$, the guessing probability of guessing the secret sec in one try is defined by

$$Gue(sec|X) = \sum_{X_i \in X} gue(X_i), \tag{4.10}$$

where

$$gue(X_i) = \begin{cases} \max_{mls_e \in X_i} p'(mls_e) & \textit{if } X_i \neq X_{random}, \textit{ i.e. } X_i \textit{ is a regular class} \\ \\ \max_{mls_e \in X_i} pu'(mls_e) & \textit{if } X_i = X_{random}, \textit{ i.e. } X_i \textit{ is a random class} \end{cases} \tag{4.11}$$

where $\max_{mls_e \in X_i} p'(mls_e) \in P'$ is the maximum probability of a most likely sequence in a regular class X_i, and $\max_{mls_e \in X_i} pu'(mls_e) \in PU'$ is the maximum uncertainty in the random class.

In this definition, we consider that random web traffic can also infer the user secret. Imagine that a traffic sequence not matching to any most likely sequences is observed. It is more likely that this traffic comes from a transition with a higher uncertainty of observations.

Example 1 (Leaks from random web traffic) Given a set of three most likely sequences mls_1, mls_2 and mls_3 for a set with three transitions e_1, e_2 and e_3 respectively, the conditional probabilities of the most likely sequences, i.e. the probabilities before normalisation, are $p(mls_1) = 1$, $p(mls_2) = 0.9$, and $p(mls_3) = 0.8$. Consider the uniform distribution on the set of transitions. If an observation not following any most likely sequences is observed, it is more likely that this traffic comes from transition e_3, as it has the highest probability of generating random web traffic.

4.6 Experiments

We implement the methodologies proposed to conduct analyses on four real-world web applications, whose testing scenarios are shown in Table 4.1. Leakages in each web application were evaluated from both individual direct and indirect transitions.

Table 4.2 shows the results of guessing probabilities in each testing scenario. The number in column "Test" refers to the testing scenario shown in column "Test" in Table 4.1. Columns "Before" and "After" display the guessing probabilities of secrets before and after observing web traffic respectively.

Previous work has mainly analysed side-channel leakages from either individual direct transitions or combined paths. Hence we also evaluate leaks from combined paths. Column "Combined Path" in Table 4.2 shows the leaks through web traffic of combined paths.

In the experiments, each test case was executed five times repeatedly. The analysis was conducted with threshold of indistinguishable packets $Tp = 0.1$, displacement range $d = 2$, threshold of similarity $Tm = 0.7$ and threshold of most likely sequences $Tt = 0.6$.

Table 4.1: Testing Scenarios of Web Applications

Website	Test	Testing Scenario
Google	1	Direct: Logging into a user account successfully
	2	Direct: Logging into a user account failed
	3	Indirect: Accessing to the homepage with a user account logged
	4	Indirect: Accessing to the homepage with a account logged out
Facebook	5	Direct: Logging into a user account successfully
	6	Direct: Logging into a user account failed
	7	Indirect: Clicking a button of "Find Friends"
	8	Indirect: Clicking a button of "Logout"
Amazon	9	Direct: Inputting a search keyword with an anonymous user
	10	Indirect: Visiting the homepage, following Test 9
	11	Direct: Inputting a search keyword with an account logged
	12	Indirect: Visiting the homepage, following Test 11
	13	Direct: Inputting a search keyword with another account logged
	14	Indirect: Visiting the homepage, following Test 13
NHS	15	Direct: Selecting a symptom
	16	Indirect transition(s), following Test 15
	17	Direct: Inputting a postcode
	18	Indirect: transitions following Test 17
	19	Direct: Inputting a birth year
	20	Indirect: transitions following Test 19

In this section, a probability mentioned in a term like "the probability of an observation following the traffic pattern" refers to the conditional probability before normalisation, i.e. $p(mls_e)$, instead of probability $p'(mls_e)$.

Figures 4.10, 4.11, 4.13 and 4.14 plot the most likely sequences generated for each web application. In a sub-figure, each line represents a most likely sequence mls_e whose probability $p(mls_e) \neq 0$. X-axis shows the position of a most likely sequence while Y-axis represents the sizes at each position.

4.6.1 Google

Overview

The secret examined on Google website is the user identity, originated from the authentications of the five Google user accounts. A direct transition concerns an authentication of a user, by submitting the sensitive information of user name and password. There are two branches from an authentication, one of successful login and the other of failed login. Both scenarios were analysed in Tests 1 and 2 respectively, as shown in Table 4.1.

As the experiments were performed using five user accounts, the priori chance of guessing a user identity is 0.2. To clarify, the five user accounts are mentioned as *account 1*, *account 2*, ..., and *account 5* respectively in this section.

Indirect transitions after successfully authenticating were tested under two scenarios. Test 3 is for a scenario where Google homepage is accessed straightly, with a user account kept logged. And Test 4 is for the second scenario, where the homepage is accessed after

Table 4.2: Results in terms of guessing probabilities in each web application

Website	Sensitive data (Secret)	Before	Test	After	Combined Path
Google	User name & password (User identity out of 5 users)	0.2	1	0.32	0.2;0.2
			2	0.4	
			3	0.96	
			4	0.96	
Facebook	User name & password (User identity out of 5 users)	0.2	5	0.56	0.76
			6	0.28	
			7	0.58	
			8	0.4	
Amazon	Searching input (User's interest out of 50)	0.02	9	0.45	0.30
			10	0.15	
			11	0.06	0.02
			12	0.056	
			13	0.02	0.02
			14	0.032	
NHS	Symptom (User's symptom out of 11)	0.091	15	0.11	0.60
			16	0.54	
	Postcode (User's postcode out of 42)	0.024	17	0.11	0.28
			18	0.122	
	Birth year (User's birth year out of 29)	0.034	19	0.034	0.068
			20	0.068	

a user account is logged off.

On the other hand, there are not indirect transitions examined after failed authentications.

In the first round of testing on Google websites, leakages of user identities from indirect transitions in Tests 3 and 4 are unexpectedly high. To guarantee accuracy of the results, a second testing round was performed after a few weeks. Surprisingly, the results from the second round are similar to those in the first round, even the traffic patterns generated are similar among the two rounds.

The most likely sequences displayed in Figure 4.10 come from the first testing round.

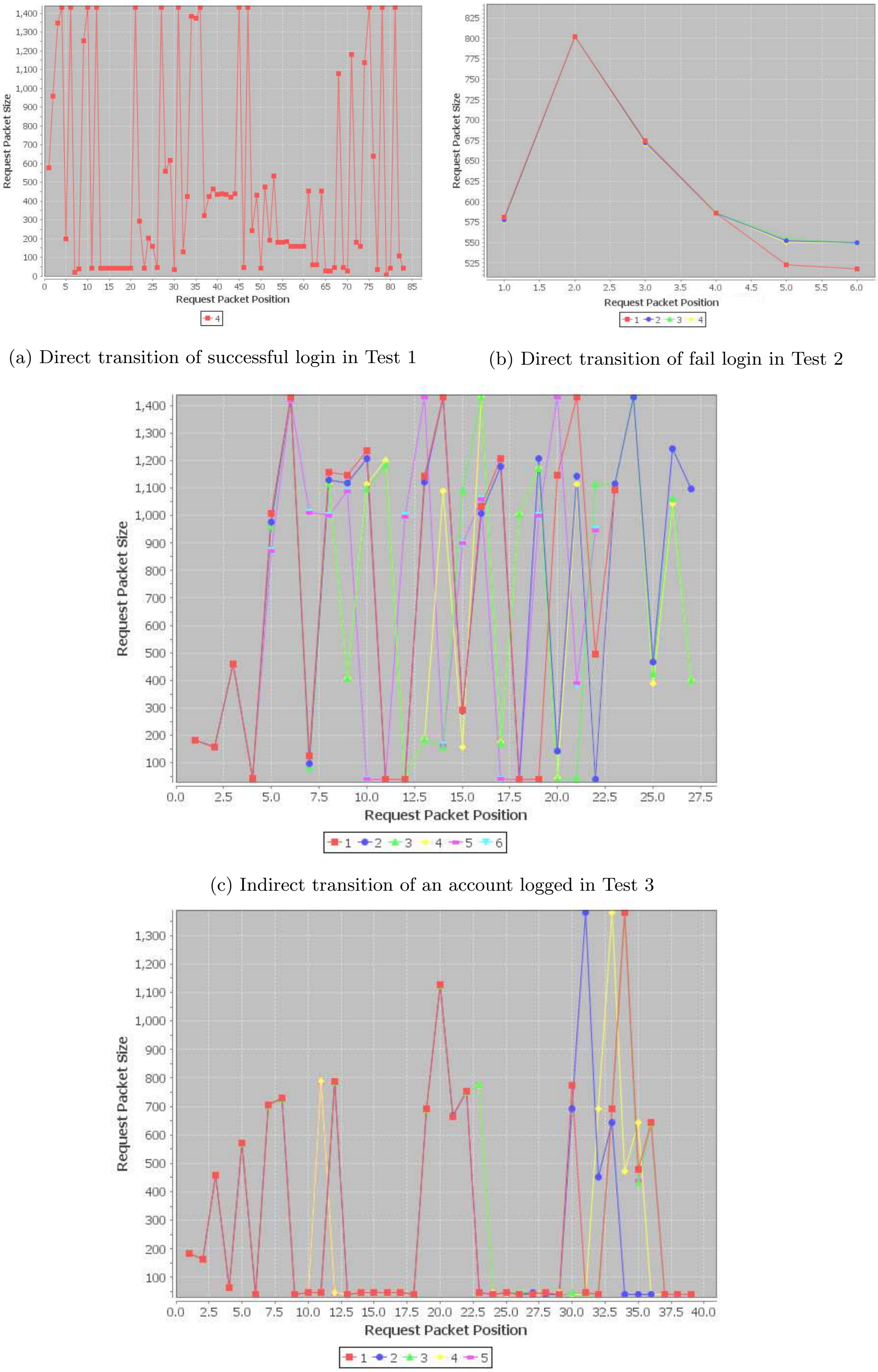
On the whole, the results show that user identities from Google accounts are leaked largely from indirect transitions, while few are leaked from direct transitions.

Direct Transitions: Authenticating User Accounts

A direct transition is an authentication with a pair of user name and password typed in on page `https://accounts.google.com/`.

Figure 4.10 presents the most likely sequences constructed under each scenario in Google website. Figures 4.10a and 4.10b are for the scenarios of successful login in Test 1 and failed login in Test 2 respectively.

As shown in Figure 4.10a, only one most likely sequence in terms of account 4 is plotted, which suggests that the direct transitions regarding other accounts generate random web traffic. The guessing probability in Test 1, as shown in Table 4.2, is derived



(a) Direct transition of successful login in Test 1

(b) Direct transition of fail login in Test 2

(c) Indirect transition of an account logged in Test 3

(d) Indirect transition of an account logged out in Test 4

Figure 4.10: Most likely sequences in Google website

from $0.2 * (0.6 + 1) = 0.32$ using Equation 4.10. Value 0.6 is the probability of the most likely sequence for account 4, and value 1 is the maximum uncertainty of observations in the random class.

However, we suppose that it is a coincidence where observations for account 4 following the traffic pattern. It is because that observations for other accounts are random and there is no ground that observations for account 4 are special from others. Moreover, the probability of the most likely sequence for account 4 is relatively small, with a value of 0.6, which also suggests that it may be a coincidence.

Therefore, we consider that direct transitions examined in Test 1 leak no user identities, even though the guessing probability after observations is higher than the prior chance.

In Test 2 concerning failed authentications, from the most likely sequences in Figure 4.10b, observations in terms of accounts 1-4 are consistent with the indistinguishable most likely sequences, while those for account 5 are random.

Similar to Test 1, we suppose that it is a coincidence where the observations for account 5 were randomly generated in this testing round.

Accordingly, it is suggested that there is no user identity leaked from direct transitions, no matter a user is authenticated successfully in Test 1 or failed in Test 2.

Indirect Transitions: Visiting Homepage

Web users often access to web pages while their Google accounts are logged in. The communications of accessing to web pages in no relation to user accounts do not invoke direct interactions with user accounts. However, it is probable that the information related to user accounts can be implicitly transmitted through, e.g. cookies.

Therefore, Test 3 examined the indirect transitions accessing to `https://www.google.com` with user accounts kept logged in. And Test 4 examined the indirect transitions after user accounts were logged out.

1. When keeping a user account logged in Test 3

Figure 4.10c depicts six most likely sequences out of five accounts, where both sequences 5 and 6 are for account 5, one with an old password before changing, while the other with a new password.

It can be seen that most likely sequences for each account are unique from one another. On the contrary, most likely sequences 5 and 6 are indistinguishable from each other, as they are related to a same account before and after password changing. It can be inferred, therefore, that observations in this scenario are dependent on user accounts, regardless whether the password of an account is changed or not.

As shown in Table 4.2, the guessing probability in Test 3 goes up to a value of 0.96, which nearly reaches to a full leak.

Therefore we conclude that observations from the indirect transitions in this scenario cause the user's identity to be easily inferred via traffic analysis.

2. When logging out a user account in Test 4

The result obtained in Test 3 is surprising and encouraged us to investigate more about the indirect transitions. People may be curious to know what happens when a user account is logged out. Can user identities still be leaked from the indirect transitions after the user accounts are logged out?

Test 4 tested the indirect transitions in as similar a way as Test 3, but after logging out user accounts.

Figure 4.10d presents that the most likely sequences for each account are unique. The guessing probability is 0.96, displayed in Table 4.2, which suggests that user identities are substantially leaked through indirect transitions, even though the user accounts are logged out.

Therefore, suggested from the high leakages in Tests 3 and 4, it is supposed that some element carrying information related to user accounts is transmitted to the indirect transitions. The element exists even after user accounts are logged out. This kind of element can be, e.g. cookies. This assumption, in fact, inspired our future work regarding the effect of cookies on observations, as described in Chapter 5.

Combined Paths

Combined paths were examined in two scenarios. The first scenario is composed of Tests 1 and 3, where a user first successfully logged into a user account and then visited the homepage. The second scenario combines Tests 1 and 4, in which the homepage was accessed after logging out the user account.

As shown in column "Combined Path" in Table 4.2, guessing probabilities are both 0.2 in two scenarios, separated by a symbol ";". It is suggested that user identities are not leak through web traffic from combined paths.

Conclusion

From the results in Google website, leaks of user identities from indirect transitions are more serious than those from direct transitions.

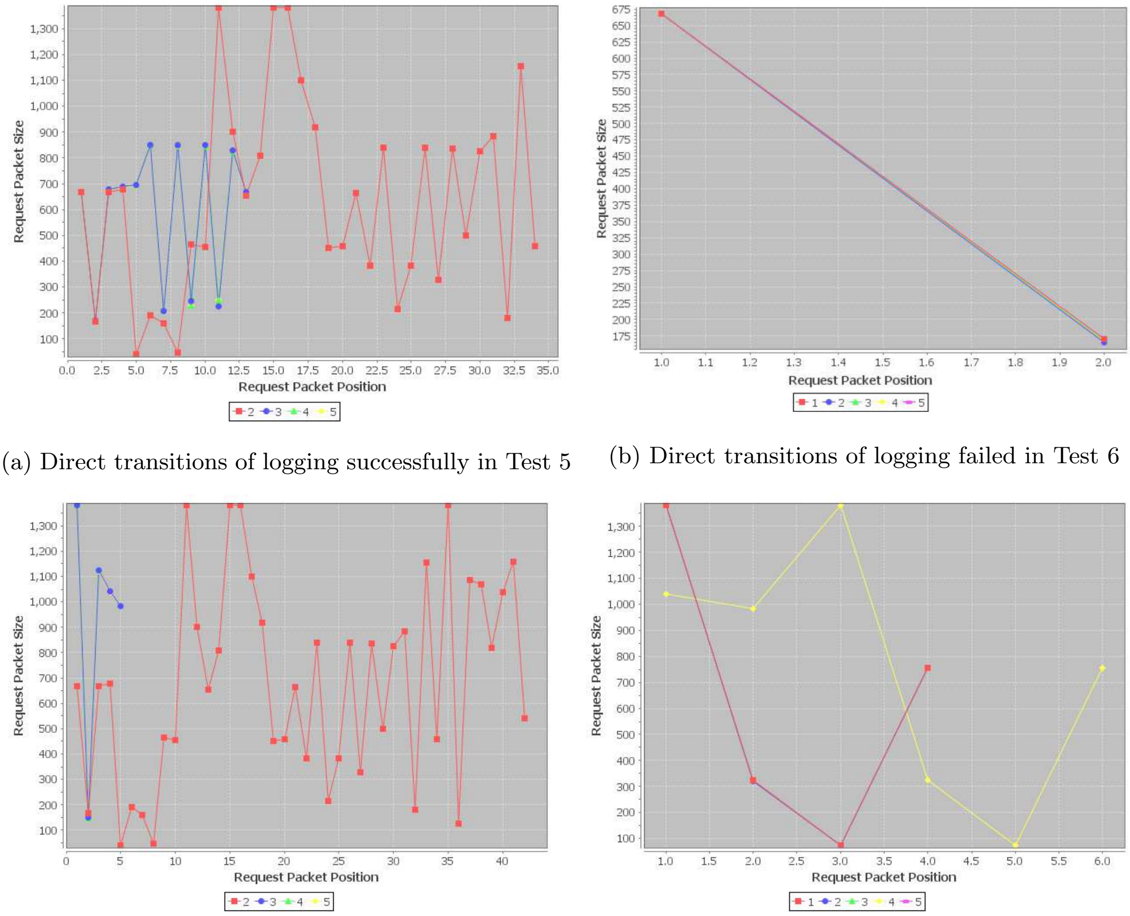
Web traffic from indirect transitions has strong capability of revealing user identities, which, however, is weakened heavily by combining with web traffic from direct transitions. It is because that the web traffic from a direct transition is largely random, which constitutes a great proportion of packet lengths in an observation from a combined path.

The experiments suggest a side-channel vulnerability in terms of user identities on Google accounts. Further investigation of the leakage of user identities from Google accounts is conducted, as described in Chapter 5.

4.6.2 Facebook

Similar to the experiments in Google website, five user accounts were used to evaluate the leakage of user identities in Facebook website. Direct transitions in terms of successful login and failed login were examined in Tests 5 and 6 respectively. Indirect transitions were triggered after successful authentications, by (1) clicking a button of "Find Friends", examined in Test 7; and then by (2) clicking a button of "Logout" in Test 8, following the indirect transition executed in Test 7.

Figure 4.11 presents the most likely sequences generated in Tests 5-8 in Facebook.



(a) Direct transitions of logging successfully in Test 5

(b) Direct transitions of logging failed in Test 6

(c) Indirect transitions by “Find Friends” in Test 7

(d) Indirect transitions by “Logout” in Test 8

Figure 4.11: Most likely sequences in Facebook website

Direct Transitions

A direct transition started from `https://www.facebook.com/` by authenticating a user.

Test 5 examined the direct transitions in terms of successful login of user accounts, where the most likely sequences are displayed in Figure 4.11a. The most likely sequences for accounts 3-5 are indistinguishable from one another, but distinguishable from that for account 2. On the other hand, the direct transitions related to account 1 generate random web traffic, suggested by the absence of the most likely sequence for account 1 in Figure 4.11a. The guessing probability shown in Table 4.2 is around 0.6 in this scenario, which has an obvious increment compared with the prior guessing probability.

Test 6 is for the direct transitions of failed authentications. Figure 4.11b indicates that the web traffic for each direct transition is generally consistent, which follows the indistinguishable traffic patterns. The guessing probability with a value of 0.28 suggests that sometimes web traffic generated during communications is unstable.

As a result, we consider that web traffic for the direct transitions in Test 1 reveals some user identities, while no user identities are leaked in Test 2.

Indirect Transitions

After logging into a Facebook account, there is a button of “Find Friends” on the web page. Test 7 examined the indirect transitions triggered by clicking this button. And a succeeding indirect transition following Test 7, triggered by a “Logout” button, was examined individually in Test 8.

Figure 4.11c plots the most likely sequences in Test 7. Like the traffic patterns generated in Test 5 in Figure 4.11a, most likely sequences for accounts 3, 4 and 5 in Test 7 are indistinguishable from one another, but distinguishable from that for account 2. Moreover, the observations for account 1 are also random in Test 7. Therefore we suppose that some factors, which depend on user information but not unique from each user, may decide the most likely sequences.

As an example of a possible factor, take the group numbers in user accounts. Assume that web traffic for user accounts with group numbers between 1 and 10 follows the indistinguishable traffic patterns. For the user accounts of more than 10 groups, observations follow a traffic pattern which is distinguishable from those regarding the group numbers between 1 and 10. On the other hand, web traffic for user accounts with no groups is randomly generated. Therefore in this example, group number is a factor which leads to the variation of web traffic between users.

Figure 4.11d plots the most likely sequences in Test 8. Excluding the unique traffic pattern for account 4, other accounts share a common most likely sequence. Comparing the traffic pattern for account 4 with that for other accounts, the first two packets in the traffic pattern for account 4 are redundant. It is possible that the first two packets are generated from noise. Or some particular features in account 4 cause the web traffic distinguishable from that for other accounts.

Combined Paths

We examined the combined paths composed of transitions in Test 5, Tests 7 and 8, where a user first logged into the user account, then she/he clicked the button of “Find Friends” and finally logged out the account using the “Logout” button.

As shown in Table 4.2, the guessing probability from combined paths is 0.76, which leak most user identities than individual transitions. By connecting the observations in different testing scenarios, the observations combined become distinguishable from users which may not be identified from individual observations.

From the results in Facebook website, some user identities can be leaked from either the direct or indirect transitions. Therefore, it is important to analyse transitions individually, so that a developer can get more details about which transition leak user privacy.

4.6.3 Amazon

For Amazon website, the secret examined is user search inputs. Before the testing, 50 random search keywords were specified as secret values.

Figure 4.12 illustrates the testing process. By requesting a search keyword, a direct

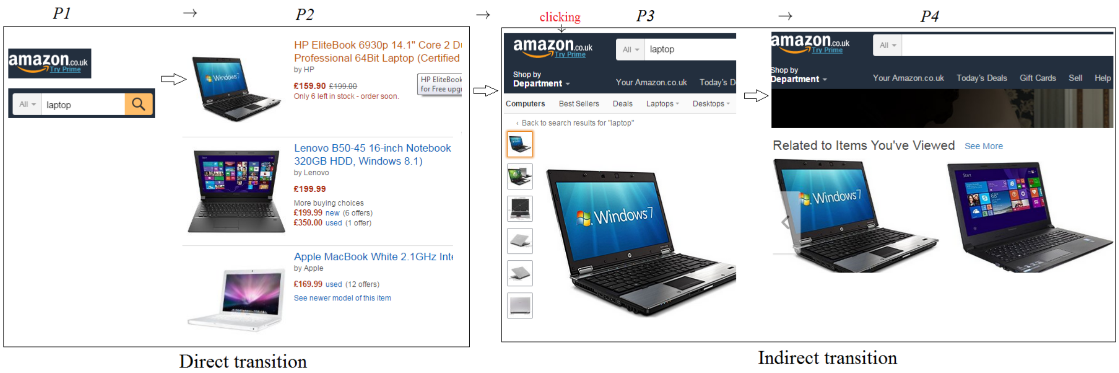


Figure 4.12: Testing process in Amazon

transition $P1 \rightarrow P2$ was performed from the homepage `https://www.amazon.co.uk` to a page of search results. A further search was conducted by clicking the first item, i.e. $P2 \rightarrow P3$. Then the indirect transition we examined is $P3 \rightarrow P4$, by clicking the Amazon logo on the page of the first item, which returned to the homepage.

Testing Scenarios

We executed test cases under different scenarios as follows:

- Scenario 1. Without logging into a user account

  Test cases were executed with an anonymous user, i.e. without logged into a user account. Test 9 examined the direct transitions in this scenario, and Test 10 examined the indirect transitions.

- Scenario 2. Logging into a user account

  In this scenario, a user account was first logged in. After a successful authentication, a direct transition was performed.

  Tests 11 and 12 examined the direct and indirect transitions in this scenario respectively.

- Scenario 3. Logging into a different user account

  Similar to scenario 2, a different user account was used in this scenario. And the direct and indirect transitions were presented in Tests 13 and 14 respectively.

  This scenario was used to double check that the leakage of user search inputs is really related to the login of a user account.

Scenario 1 was designed to investigate a general case about the leakage of user search inputs. Scenario 2 analysed particularly whether a logged user has an influence on the leakage of search inputs. And scenario 3 confirmed that the leakage is really affected when a user was authenticated.

Direct and Indirect Transitions

Displayed in Table 4.2, the result in Test 9 shows that the probability of correctly guessing the search keyword, via the direct transition, is nearly 50% when a user did not signed into an account. While the guessing probability is lower from the indirect transition, as shown in Test 10.

On the contrary, when a user account was logged in, the guessing probabilities are less than 0.1 from both direct and indirect transitions, as displayed in Tests 11-12. Similarly, when another account was logged in Tests 13-14, the guessing probabilities are also close to the prior probability before observing.

It is surprising to discover that a large amount of sensitive information is leaked with an anonymous user, while the leakage is mitigated when a user is authenticated. By examining the web traffic when a user was authenticated, the observations are largely random, which cause user inputs indistinguishable.

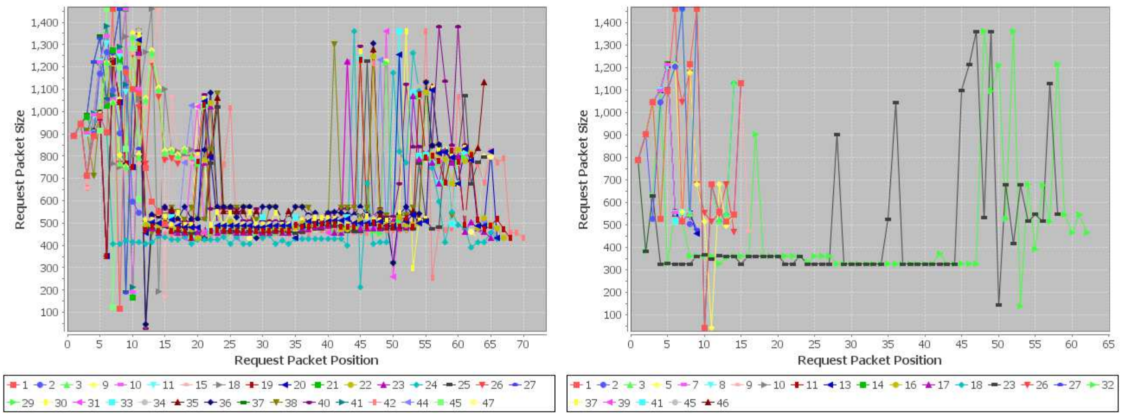
For the reason, we suppose that Amazon may provide different protection schemas on user search inputs. A stronger countermeasure is applied against traffic analysis when a user logs into her account.

Another possibility can be that random web traffic is generated because of a logged user account. More specifically, when a user account is logged in, some information about the user is also transmitted to the communications. However, the information generates random web traffic, which leads to the incapability of identifying user search inputs from web traffic.

Combined Paths

For the combined paths, the guessing probability shown in Table 4.2 is 0.3 when a user is anonymous, lower than the result from the direct transitions. While there is no leak from combined paths when a user is authenticated.

Figure 4.13 plots the most likely sequences generated in Tests 9 and 10 when a user was anonymous. On the other hand, when a user was authenticated, observations are largely random. Hence the most likely sequences generated in Scenarios 2 and 3 are not presented.



(a) Direct transitions with an anonymous user in Test 9

(b) Indirect transition with an anonymous user in Test 10

Figure 4.13: Most likely sequences in Amazon website

4.6.4 NHS Direct Symptom Checker

On NHS website, a patient uses a symptom checker[1] to get an online diagnosis advice based on a health symptom input. Sensitive information involved in an execution path includes the user's health symptom, the postcode of living address and the birth year. Leakages of the health symptom, the postcode and the birth year were analysed respectively in the experiments.

After performing several connecting user actions, a web page was accessed to select a health symptom out of a list of options. After submitting the data, consecutive transitions required more user sensitive information, including the postcode of living address and the birth year. Finally, the test case ended with a web page containing a final diagnosis advice.

Unlike the previous three web applications analysed, each with one secret examined, this website contains three secrets under test. We extracted the direct transitions and the indirect transitions in terms of the health symptom, the postcode, and the birth year respectively. And the consistency system is useful in this website, which provides an optimum consistency between indirect transitions.

Tests 15 and 16 examined the secret of user health symptom from direct and indirect transitions respectively. Likewise, tests 17-18 analysed the leaks in terms of postcodes and Tests 19-20 in terms of the birth years. Details are shown in Table 4.1. Figure 4.14 displays the most likely sequences generated in Tests 15-20.

Leakage of Health Symptoms

Test 15 examined the leakage of health symptoms from direct transitions, and Test 16 for indirect transitions.

As plotted in Figure 4.14a, in Test 15 all most likely sequences are indistinguishable from one another. Therefore, a user's health symptom is hardly disclosed from the direct transitions. On the contrary, for the indirect transitions tested in Test 16, Figure 4.14b shows that the most likely sequences are distinguishable. The details of guessing probabilities can be seen in Table 4.2.

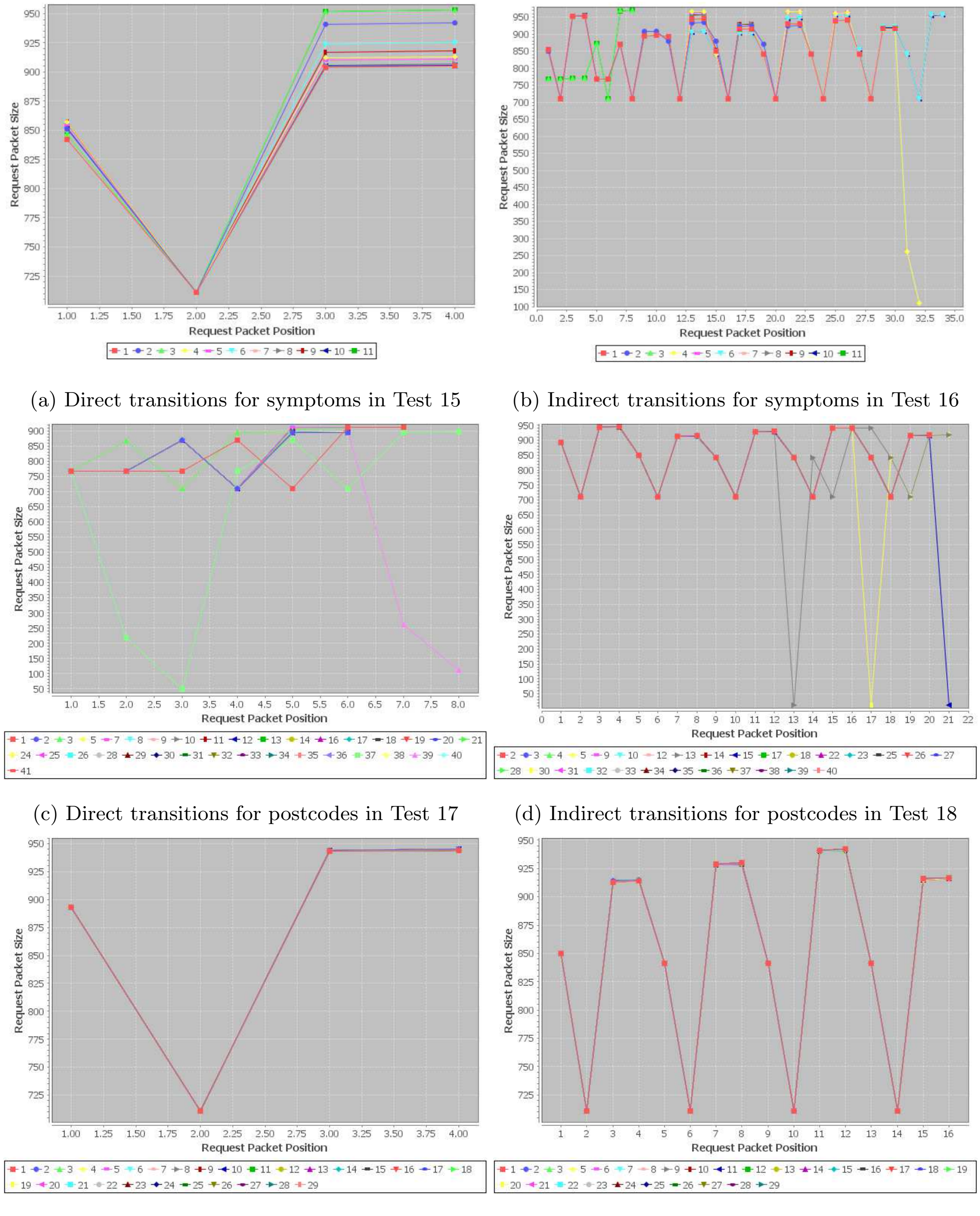
The guessing probability from combined paths is similar to that from indirect transitions, as shown in Table 4.2. Since the traffic patterns for each direct transition are indistinguishable and the web traffic for each indirect transition follows a traffic pattern, it is possible that for the secret values whose most likely sequences for the indirect transitions are classified in a same class, the most likely sequences for the combined paths are also in a same class.

Leakage of Postcodes

Figures 4.14c and 4.14d show the most likely sequences for the direct transitions and indirect transitions for the postcode respectively.

There are more traffic patterns plotted from direct transitions than from indirect

[1] https://www.nhsdirect.nhs.uk/CheckSymptoms/SATs/InitialAssessment.aspx#progress
The experiments on this symptom checker were performed in early 2014, before this checker was closed in mid 2014 and replaced by a new NHS symptom checker.



(a) Direct transitions for symptoms in Test 15

(b) Indirect transitions for symptoms in Test 16

(c) Direct transitions for postcodes in Test 17

(d) Indirect transitions for postcodes in Test 18

(e) Direct transitions for birth years in Test 19

(f) Indirect transitions for birth years in Test 20

Figure 4.14: Most likely sequences on NHS website

transitions. However, Table 4.2 shows that the indirect transitions leak more user postcodes than the direct transitions, with the guessing probabilities of 0.11 and 0.122 from direct and indirect transitions respectively. Therefore, we can suggest that there are more traffic patterns for direct transitions indistinguishable.

With regard to the combined paths, the guessing probability is more than double that for direct or indirect transitions. For the reason, it can be supposed that the observations following indistinguishable traffic patterns in one testing scenario may be combined with those distinguishable in another scenario, which result in a larger distinction between observations for the combined paths.

Furthermore, it is possible that random web traffic generated in a scenario is connected with the regular web traffic. The random traffic comprises a small proportion of packets in a combined observation, which has few impact on the combined traffic of following a traffic pattern. Therefore the combined observations leak more information than individual transitions.

Leakage of Birth Years

In terms of the scenario of user birth years, 29 values ranging from 1930 to 2013 were generated randomly.

As shown in Figures 4.14e and 4.14f, all the direct transition in Test 19 have indistinguishable traffic patterns, so do the indirect transitions in Test 20 except sequence 4.

Table 4.2 shows that the guessing probability for the direct transitions is equal to the prior chance of 0.034. Accordingly, it can be taken for granted that the direct transitions produce constant web traffic completely following the traffic pattern.

For the indirect transitions in Test 20, the fourth indirect transition generates random web traffic which does not follow the common traffic pattern. This exception can be a coincidence which is caused by, e.g., noise.

Therefore, although the guessing probabilities for the indirect transitions and the combined paths double the prior chance, we consider that there are no leaks from them.

For the reason, it is likely that (1) the server does not request the information of user birth year; or (2) the information is transmitted to communications, however, due to the same format of the birth years with four digits like 19XX or 20XX, the observations are not affected by the user inputs.

As a result, the secret about the birth year is the only one among the four websites which is not leaked by traffic analysis.

4.6.5 Discussion of the Experiments

Experiments on these four real-world websites indicate that user confidential information can be leaked, via traffic analysis, not only from direct transitions, but also from indirect transitions which implicitly communicate with sensitive information. In some cases, indirect transitions leak the largest amount of user confidential information, compared with either direct transitions or combined paths.

Moreover, from the results in the experiments, it is important to perform analysis on individual transitions. It provides more knowledge about which specific transitions are the possible side-channel vulnerabilities to leak user privacy.

The result in terms of the leakage of user birth year in NHS website seems appealing to provide a promising countermeasure against traffic analysis. If it is the case that different user inputs with a same format cause web traffic indistinguishable, then a new idea of constructing sensitive information with a same format can be a possible countermeasure. Investigation and implementation of this countermeasure are left for the future work.

Although this thesis does not implement countermeasures against traffic analysis, the analysis of vulnerabilities in individual transitions provides an extensive instruction for developers to apply effective countermeasures.

4.7 Discussion

This chapter introduces a black-box automated system for analysing side-channel vulnerabilities in real-world web applications. This system aims to identify which transitions leak user secrets, including those implicitly related to sensitive information.

Previous work has rarely concentrated on side-channel leakages from individual indirect transitions systematically. In this sense, this chapter advances towards an extensive analysis of side-channel vulnerabilities in real-world web applications.

Instead of providing an empirical study of side-channel leakages in real-world web applications, this work demonstrates a security threat that user secrets can be revealed from transitions which implicitly transmit sensitive information. The leaks, far worse than previously thought, can be even larger than those from transitions with explicit relation to sensitive information.

Therefore, it is relevant to conduct comprehensive examination on each individual transition. This is what we have done in this chapter.

On the other hand, there are some limitations in this chapter. For example, the test cases analysed in the experiments are limited to the sampling secrets we designed. And we do not make any theoretical or experimental analyses to validate the correctness of the most likely sequence derived by our approach. In the experiments, similar to Chapter 3, though our aim of the analyses is not to estimate the specific leakages by sampling, we lack the experimental validation which verify that the results produced by the analyses are correct.

Moreover, the experiments are conducted using one machine located at a single location. Questions like “do the experimental results vary from testing machines and (or) testing locations”, “is it possible that the variation of web traffic between secret values is cased by, e.g. noise or unstable network conditions, which actually does not leak user secrets”, etc. can be asked.

Hence, it is urgent for calling a more comprehensive analysis of side-channel vulnerabilities in web applications. Our next work described in the following chapter showcases a more extensive side-channel analysis of vulnerabilities in web applications.

Chapter 5

Side-Channel Leakages of User Identities

5.1 Motivation and Overview

In Chapter 4, we disclose that user identities, presented by Google accounts, can be revealed from indirect transitions via traffic analysis. Motivated by this discovery, this chapter explores the leakage of user identities in real-world web application.

Nowadays, it is important to protect user identities. Given a space of users in a company network, e.g., a web attacker wants to identify the president among the online users, in order to track the president's web activities. However, it is impractical to sneak user identities through, e.g. IP addresses, since the user may go on a business trip or uses several mobile devices to handle work. Moreover, the user can use anonymous services to cloak his actual IP address. Hence this chapter proposes a novel threat scenario of revealing user identities, through the logged user accounts on a website like Google.

To the best of our knowledge, previous work has mainly mounted side-channel attacks within individual websites, where the leakage happens through communications within a same server. Take the experiments on Facebook website described in Chapter 4 as an example, where the transitions examined are between the web pages in Facebook, at which an examined secret is located, regardless of direct or indirect transitions.

Instead, this chapter works on communications with third-party websites outside the server of which the confidential information is located.

The leakage of user identities examined is from 50 Google user accounts to websites included in the Alexa [1] Top 150 websites [1].

To answer the question asked in Chapter 4 about whether the leakages of a secret examined at different locations are consistent, this chapter analyses the leakage from five locations. The experiments aim to check if the leakages from different locations are consistent and to guarantee that the leakages evaluated are as precise as possible. Moreover, we develop four testing scenarios for investigating the leaking sources, more specifically, whether cookies and logged user accounts are the factors causing web traffic

[1] Alexa Top websites are updated by one-month Alexa traffic over the past month. The data used in this chapter was retrieved in October 2014.

to leak user identities. Although cookies is a well-known source of tracking users in web applications, there is little work studying on the effect of cookies in traffic analysis in web applications. The work of this chapter has been published on [65].

Contributions. The main contributions in this chapter are presented as follows:

1. It proposes a side-channel threat of fingerprinting users, which demonstrates that even Google users are vulnerable in traffic analysis. It shows that user identities from Google accounts can be leaked through communications between Google and third-party websites.

2. It introduces a "one → many" relation between each transition and multiple traffic patterns.

3. It analyses the effect of cookies and logged user accounts in side-channel leakages of user identities.

5.2 Threat Scenario

We propose a threat scenario of revealing user identities from Google user accounts.

A user first logs into a Google account, where the transition is regarded as a *direct transition.* Then the user accesses to a new website which is either a Google or non-Google website by typing a URL address of the website into a URL bar. This communication is regarded as an *indirect transition.*

Assume an attacker can be a Google service provider or a database administrator who profiles the traffic patterns of web traffic for each user account, given a space of Google users. We also assume that the attacker has prior knowledge concerning which website a victim is visiting. Details of how an eavesdropper fingerprints websites can be seen in many literature, see e.g. [35, 62].

The attacker aims to identify a user's identity, i.e. which user out of the space of Google users carries out the communication, by eavesdropping a traffic sequence and matching it to a traffic pattern for a user.

Consider a non-Google website as a *third-party website* or an *external website*, relative to a Google website which is regarded as an *internal website.*

Experiments in this threat scenario are implemented in a "close-world" environment from a perspective of a developer. Instead of obtaining an accurate leakage of user identities, this chapter aims to demonstrate a possible threat of leaking user identities through communications with third-party websites.

5.3 Experimental Specification

This section describes the specifications of experiments. First, we assume each user account corresponds to a unique user, and the secret examined is a user identity represented by a user account, instead of the user name and the password of the account. Each direct/indirect transition examined is for a user account.

5.3.1 Testing Process

Our experiments used 50 Google accounts as targets. Alexa Top 150 websites excluding Google websites were examined, where each is an external website.

Test Case Execution

In general, each test case consists of a direct transition and an indirect transition for an external website. A direct transition was an authentication of logging into a user account on Google's authentication page `https://accounts.google.com/ServiceLogin`. Then an indirect transition was performed by accessing to an external website from the logged account page. This kind of indirect transition is regarded as a communication *from internal to external websites.*

There is a special testing scenario where a test case contains a single transition in no relation to user accounts, without the direct transition of authenticating. Details are described in section 5.3.2 in terms of testing scenarios.

Each test case was run ten times, to build a set of ten observations in terms of each user account. Caches and cookies were cleared every time before a new test started. A round-robin test was performed on each user account for all the 150 websites.

Selenium [14] was exploited to execute the testing, and Jpcap [10] was used to record the web traffic.

If the user identity is leaked through indirect transitions with a website, the website is considered as leaking user identities or fingerprinting users. The terms of "fingerprint users" and "leak user identities" are used interchangeably.

Since there are 150 websites and 50 user accounts, 7500 test cases are required to be executed in a testing round. Moreover, if each test case is executed ten times,there are 75000 tests in total when testing on all the websites one round. Hence a large amount of time will be spent when executing one round for all the websites.

Moreover, with a preliminary analysis of some collected web traffic, we discover that not every website leaks a large amount of user identities.

Therefore, to save time for performing in-depth analysis on websites which likely leak large amounts of user identities, a coarse *cleansing operation* was first conducted to roughly refine the websites by eliminating those which may leak few user identities.

Cleansing Operation

To make results more convincing, in the beginning all 7500 test cases were tested using two laptops located in a same network. A test case was executed where a user account was first authenticated and then an external website was accessed to. It took around two months to complete the cleansing operation.

Analysing and quantifying the leaks using the methodologies described in sections 5.4 and 5.5, if the leakages for a website at both laptops are similar and the guessing probabilities are relatively high, the website probably leaks user identities stably at a network. Thus the website will be selected to be tested in depth in next stage.

As a result, 25 websites were obtained, each of which is likely to leak large amount of the secret.

The two laptops were configured, one of Windows XP system with Intel Core i3-2310M CPU @ 2.10GHz and 3.41 GB of RAM, and the other of Windows 7 system with Intel Core i5-3230M CPU @ 2.60GHz and 3.88 GB of RAM.

Next, we show how to perform the in-depth analysis.

In-depth Testing Process

In addition to the 25 external websites, two internal websites `www.google.com.br` and `www.google.it` were connected as the indirect transitions after the authentication. These transitions are considered as *from internal to internal websites.* They were performed in order to compare with those from internal to external websites.

The 50 user accounts on the 27 websites were examined in the PlanetLab testbed [13], from five virtual machines located at the Technical University of Berlin (*Berlin, Germany*), the University of Cambridge (*Cambridge, UK*), Imperial College (*London, UK*), the University of Neuchatel (*Neuchatel, Switzerland*) and the University of Stuttgart (*Stuttgart, Germany*) respectively.

We tested them at multiple locations to (1) guarantee that the leakages related to a website from different networks are consistent, in order to mitigate the error from a single testing site as far as possible; and to (2) investigate whether the leakage depends on locations, which may reveal user locations.

The Firefox web browser was installed in the five machines with versions ranging from 3.6.24 to 3.6.28 under the Fedora 14 operating system. It is the browser used by Selenium for executing the testing.

Four testing scenarios were developed for investigating leaking sources which cause web traffic to reveal user identities. They are described as follows.

5.3.2 Testing Scenarios

In the real world, a user often visits an external website with her Google account logged in. It is possible that the external web server requests information related to the logged user account. The data may be insignificant to user sensitive information, however, it may contain some unique data which leads to distinguishable web traffic between different users.

Moreover, cookies which store user login information may be sent to the network during communications with an external website.

Concerning that logged user accounts and cookies may bring information of user identities into communications, we design four testing scenarios to analyse the effects of these factors on the leakage of user identities.

- Scenario 1. General

  This is a general scenario same as the one used in the cleansing process. Communications start from a direct transition of authenticating a user account, and then immediately access to a different website, either an internal or external website.

- Scenario 2. Delete cookies

In this scenario, the cookie is cleared every time after authenticating, just before triggering a following indirect transition. Since logging into a Google account simply sets a cookie, an action of deleting cookie makes completely no information related to the user account exist.

Comparing with the result in scenario 1, this scenario can be used to analyse that whether the leakage is affected greatly by removing cookies. If the guessing probabilities are relatively high in scenario 1 but are relatively low in scenario 2, it is probable that the web traffic is affected largely by the cookies. However, simply from the statistics in scenarios 1 and 2, we are still unable to get a clear picture of which factor, out of cookies and logged user accounts, decides the web traffic. Hence, scenario 3 is developed.

- Scenario 3. Log out user accounts

  In scenario 3, the user account is logged out before communicating with a new website, but the cookie is retained. This is a way attempting to clean user trails as much as possible, but only keeping cookies as a source of user identities. It aims to see if there is an influence on web traffic when a user account is logged out.

  By comparing the results between scenarios 1, 2 and 3, a clearer picture regarding the influential factors on the leakages can be suggested, in terms of which factor, logged user accounts or cookies, accounts for a larger effect on the leakage.

- Scenario 4. Log out user accounts + Delete cookies

  In addition to logged user accounts and cookies, other external unknown-sourced factors can lead to a variation of web traffic. It can be mistakenly regarded that the variation is originated from cookies or logged user accounts.

  Therefore, how influential these unknown factors are in the variation of web traffic is estimated in this scenario, to mitigate the error caused by external unknown factors.

  This scenario attempts to clear all the user trails generated during communications. Instead of logging out user accounts and cleansing all trails, there is an alternative in a more straight way, i.e. not logging into the user accounts, which stays away from the user identities.

  This scenario should give rise to a same effect as cleansing the cookies in scenario 2, because there is not any information related to users retained. Therefore, the aim of this scenario is to provide a reference for the result in scenario 2, which confirms the accuracy of results in these scenarios. Moreover, it can also be used to check if exceptions exist when the results from scenarios 2 and 4 are different.

  In scenario 4, each test case contains a single transition, starting from Google's authentication page and forwards to an examined website straightly. A test case in terms of a website was performed 500 times and every ten traces of web traffic were aggregated into one group, to construct a space with as same the number as the user accounts analysed in other scenarios. Analysing the observations between each group, the data obtained represents the variation of leaks generated by the unknown factors. Test cases in this scenario are considered as indirect transitions.

The unknown factors evaluated in scenario 4 are regarded as *external factors* which have no relation to user identities. Relatively, logged user accounts and cookies are deemed as *internal factors* as they are related to user identities.

For a website, each testing round executed the test cases on 50 user accounts in four scenarios. The testing on 27 websites, consisting of the 25 external and the two internal websites, were performed at five locations simultaneously. One round of testing on all the 27 websites lasted over a two-week period. Multiple testing rounds were performed during a period from February to July 2015.

5.4 Data Analysis

During the testing, the raw data collected is the web traffic for a combined path containing both direct and indirect transitions. Request packets were extracted for analysis and a whole traffic sequence was split into multiple sequences of web traffic, each for an individual transition.

Given a set of observations O_e for a transition e, the traffic pattern mls_e from O_e can be constructed using the approach motivated by HMMs, same as that described in section 4.4.3 in Chapter 4.

For evaluating leakages, there are a few difference between the calculation used in this chapter and that used in Chapter 4.

Figure 5.1 gives an overview of the quantification of leakage. Given the most likely sequences generated from observation sets, we define a *confidence*–$conf(O_e, mls_e)$–for each most likely sequence mls_e. Combined with the probability of most likely sequence–$p(mls_e)$, defined in Chapter 4, a probability distribution pd on the set of most likely sequences MLS is built to quantify the leakage. The details are described in the following.

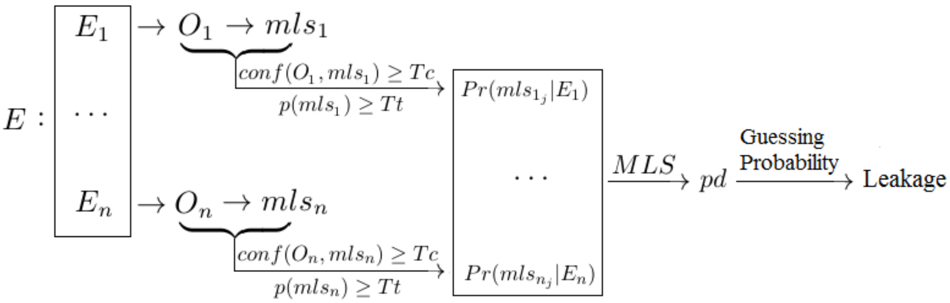


Figure 5.1: Quantification of leakage

5.4.1 Confidence of Most Likely Sequence

First we define a new notion of *confidence of most likely sequence*. It is abbreviated as *confidence*.

Definition 26 (Confidence) *Given a set of observations O_e and the most likely sequence mls_e for transition e, the confidence of most likely sequence is defined as*

$$conf(O_e, mls_e) = 1 - \min((\frac{ad + dm}{|mls_e|}), 1) \in [0, 1], \tag{5.1}$$

where ad and dm are the average distance and the dissimilarity between mls_e and observations in O_e respectively, which are defined later. $|mls_e|$ is the number of packets in sequence mls_e.

In this equation, mls_e is regarded as the *reference sequence.* $conf(O_e, mls_e)$ can be abbreviated as $conf_e$.

Confidence is used to assess the overall similarity between a most likely sequence and the observations.

Definition 27 (Average Distance) *The average distance–ad–between a traffic pattern mls_e and a set of observations O_e is defined as*

$$ad = \frac{\sum_{o_i \in O_e} dis(o_i, mls_e)}{|O_e|}, \tag{5.2}$$

where $dis(o_i, mls_e)$ is the distance between mls_e and observation o_i, defined in Definition 20, and $|O_e|$ is the cardinality of set O_e.

After calculating the distances, each observation $o_i \in O_e$ is indistinguishable from traffic pattern mls_e.

Nevertheless, there may still be dissimilarity between indistinguishable packets between observation o_i and sequence mls_e, when their packet sizes are not identical. Hence *dissimilarity* is taken into consideration, in terms of the standard deviation [32] of indistinguishable packets.

Definition 28 (Dissimilarity) *Given a set of observations O_e after Damerau-Levenshtein process and a traffic pattern mls_e,* ***dissimilarity*** *is defined by:*

$$dm = \sum_{mls_{e_k} \in mls_e} \frac{sd_k}{mls_{e_k}}, \tag{5.3}$$

where

$$sd_k = \sqrt{\frac{\sum_{o_i \in O_e} (o_{i_k} - mls_{e_k})^2}{|O_e|}} \tag{5.4}$$

is the standard deviation of indistinguishable packets at position k, and mls_{e_k} and o_{i_k} represent the packet sizes at position k in sequences mls_e and o_i respectively.

Then by substituting Equations 5.2 and 5.3 into Equation 5.1, confidence in terms of the similarity between most likely sequence mls_e and observations of set O_e can be obtained.

5.4.2 Confirmation of a Most Likely Sequence

After generating a most likely sequence mls_e for transition e, we determine if sequence mls_e is actually the traffic pattern of observations for transition e.

Here we improve the similarity between an observation $o_i \in O_e$ and traffic pattern mls_e, defined in Definition 21, based upon the confidence, where

$$sim(o_i, mls_e) = conf(\{o_i\}, mls_e) \tag{5.5}$$

Recall the definitions in section 4.5.1. If $sim(o_i, mls_e) \geq Tm$, observation o_i is considered following mls_e. Then $p(mls_e) = \frac{M}{|O_e|}$, the probability of observations following mls_e, is used to determine if mls_e can actually be the traffic pattern, i.e. $p(mls_e) \geq Tt$, where Tt is the threshold of most likely sequence.

In this chapter, we define the *threshold of confidence–Tc* to assess if a generated most likely sequence is qualified to be a traffic pattern.

Determination of the most likely sequence

Given a transition e, a set of observations O_e and the most likely sequence mls_e, if

$$conf(O_e, mls_e) \geq Tc, \text{ and } p(mls_e) \geq Tt$$

where $p(mls_e)$ and Tt are defined in section 4.5.1, mls_e is confirmed to be the most likely sequence for transition e.

Otherwise, transition e is considered generating random web traffic not conforming to any traffic patterns. We instead propose a NULL traffic pattern, so that in this case

$$mls_e = \text{NULL} \text{ and } conf(O_e, mls_e) = 1$$

Accordingly, it can be concluded that $Tc \leq conf(O_e, mls_e) \leq 1$.

Discussion of Confidence

The confidence assesses the most likely sequence mls_e from an overall angle in terms of the similarity between mls_e and the observations. In other words, it considers observations as a whole and weights an overall performance about the observations following most likely sequence.

Relatively, the probability $p(mls)$, from a perspective of individual observations, evaluates the similarities between mls_e and each observation, instead of considering the observations as a whole.

In summary, the confidence provides a stricter guarantee of confirming the most likely sequence, not only a sufficient number of observations following mls_e, but also a higher similarity among observations themselves, which ensures a lower fluctuation between observations. In this sense, the use of confidence improves the precise of traffic patterns.

A most likely sequence confirmed is regarded as a fingerprint of a user in a communication. After determining the most likely sequence for a transition, we build a relation of a "one → many" mapping between a transition and multiple traffic patterns.

5.4.3 "One → Many" Mapping for a Transition

Previous chapters consider a "one → one" mapping where each transition associates to either one observation in Chapter 3 or one traffic pattern in Chapter 4. Instead, this chapter builds a "one → many" mapping, where a transition can associate to more than one traffic pattern.

1. When $conf(O_e, mls_e) \neq 1$

The probability of observations from transition e conforming to most likely sequence mls_e is denoted by:

$$Pr(mls_e|e) = \frac{M}{|O_e|}, \tag{5.6}$$

where M is the number of observations in set O_e which follow traffic pattern mls_e, and $|O_e|$ is the cardinality of set O_e. In this case $Pr(mls_e|e)$ is same as $p(mls_e)$ defined in Chapter 4.

However, random observations not following mls_e can also be generated. They are considered following no traffic patterns. Hence we consider that they follow the NULL pattern, and the probability of observations following pattern NULL is defined as:

$$Pr(\text{NULL}|e) = 1 - Pr(mls_e|e) \tag{5.7}$$

As a whole, when $conf(O_e, mls_e) \neq 1$, an observation for transition e follows one of two most likely sequences, i.e. either mls_e or NULL. We denote the most likely sequences for transition e as mls_{e_j}, where $j = 1$ or $j = 2$. And the probability of an observation for transition e following pattern mls_{e_j} is denoted by:

$$Pr(mls_{e_j}|e) = \begin{cases} Pr(mls_e|e), & \text{if } j = 1,\ mls_{e_1} = mls_e \neq \text{NULL} \\ \\ 1 - Pr(mls_e|e), & \text{if } j = 2,\ mls_{e_2} = \text{NULL} \end{cases} \tag{5.8}$$

2. When $conf(O_e, mls_e) = 1$

Most likely sequence mls_e can be either pattern NULL or a NON-NULL pattern. In this case, all the observations follow most likely sequence mls_e.

Hence the probability of observations for transition e following pattern mls_e is:

$$Pr(mls_{e_1}|e) = Pr(mls_e|e) = 1 \tag{5.9}$$

Example 2 Given a set of observations O_e for transition e:

$$\begin{aligned} O_e = \{&(200,1)(100,2)(50,3)(80,4)(120,5);\\ &(200,1)(100,2)(50,3)(120,4)(200,5),(80,6),(100,7);\\ &(200,1)(100,2)(50,3)(120,4)(80,5)\} \end{aligned}$$

most likely sequence $mls_e = (200, 100, 50, 120, 80)$ is confirmed with thresholds of confidence, of similarity, and of most likely sequences $Tc = Tm = Tt = 0.6$ respectively. Observations o_1 and o_3 follow mls_e as $sim(o_1, mls_e) = 0.8 > Tm$ and $sim(o_3, mls_e) = 1 > Tm$. In contrast, observation o_2 is mismatched because $sim(o_2, mls_e) = 0.4$.

Hence the probability of an observation for transition e following pattern mls_e is $Pr(mls_{e_1}|e) = Pr(mls_e|e) = \frac{2}{3}$, and those mismatching is $Pr(mls_{e_2}|e) = Pr(\textsc{null}|e) = \frac{1}{3}$.

By bringing in a NULL pattern, random web traffic for a transition e is also associated to a traffic pattern. Though we propose a weak traffic pattern–NULL to represent the irregular web traffic, it is still an advance towards using multiple traffic patterns in traffic analysis. This model can improve the accuracy when evaluating leakage of user privacy, as a larger space of observations including random web traffic generated during communications are considered.

5.5 Quantification of Leaks

With a set of most likely sequences and their probabilities, this section describes how to quantify the leakage of the related secret.

5.5.1 Probability Distribution on Traffic Patterns

First we build a probability distribution on the set of most likely sequences, in terms of the probabilities of observations for a set of transitions.

Definition 29 (Probability Distribution for a set of Transitions) *Given a set of transitions E, a set of observation sets $P(O)$ where $f : E \to P(O)$, a set of most likely sequences MLS where $g : P(O) \to MLS$, and a random variable $Y : MLS \to N (N \subseteq \mathbb{R})$, the probability distribution $pd : Y \to [0, 1]$ on set MLS is defined by:*

$$pd(Y(mls_i)) = \mu(e)Pr(mls_i|e), \tag{5.10}$$

such that

$$\sum_{mls_i \in MLS} pd(Y(mls_i)) = 1$$

where $\mu(e)$ denotes the probability of a transition $e \in E$ and $mls_i \in MLS$ is a most likely sequence for transition e.

To abbreviate, the probability distribution $pd : Y \to [0, 1]$ on set MLS is expressed by $pd : MLS \to [0, 1]$ and a probability $pd(Y(mls_i))$ is abbreviated by $pd(mls_i)$. In this chapter, we suppose a uniform distribution on set E, such that $\mu(e) = \frac{1}{|E|}$.

Example 3 (Probability distribution in terms of a set of transitions) Given a set E consisting of two transitions e_1 and e_2 and a set of most likely sequences MLS, assume the probabilities of observations for transition e_1 are $Pr(mls_{e_{1_1}}|e_1) = \frac{2}{3}$, and $Pr(mls_{e_{1_2}}|e_1) = \frac{1}{3}$, and the probabilities for transition e_2 are $Pr(mls_{e_{2_1}}|e_2) = \frac{3}{5}$ and

$Pr(mls_{e_{2_2}}|e_2) = \frac{2}{5}$. With a uniform distribution on set E, the probability distribution on most likely sequences is: $pd(mls_{e_{1_1}}) = \frac{1}{2} * \frac{2}{3} = \frac{1}{3}$, $pd(mls_{e_{1_2}}) = \frac{1}{2} * \frac{1}{3} = \frac{1}{6}$, $pd(mls_{e_{2_1}}) = \frac{1}{2} * \frac{3}{5} = \frac{3}{10}$, and $pd(mls_{e_{2_2}}) = \frac{1}{2} * \frac{2}{5} = \frac{1}{5}$.

5.5.2 Quantifying Leakages

With the most likely sequences and their probabilities, now we build an equivalence relation to measure the leakage of user identities.

Given a set of transitions E, a set of most likely sequences MLS for set E, and their probabilities $Pr(mls_e|e)$, where $mls_e \in MLS$ is a most likely sequence for transition $e \in E$, an equivalence relation X on set MLS can be built as:

$$\forall mls_i,\ mls_j \in MLS,\ \ mls_i \sim mls_j$$

if and only if the similarity between them satisfies:

$$sim(mls_j, mls_i) \geq \max(Pr(mls_i|e_k), Pr(mls_j|e_t)),$$

where mls_i is a traffic pattern for transition e_k and mls_j for transition e_t, and $Pr(mls_i|e_k)$ is the probability of an observation for transition e_k following mls_i, as defined in either Equation 5.8 or 5.9.

The sequence with a shorter length is regarded as the *reference sequence*, mentioned in Equation 5.1, i.e. $|mls_i| \leq |mls_j|$. This makes the similarity smaller, which provides a "stricter" matching between two most likely sequences.

In an equivalence relation, there may be an equivalence class classifying NULL patterns, as even the irregular web traffic can leak user privacy. Given an observation not following any traffic patterns, e.g., it is more likely that this observation is originated from a transition which holds the highest probability of generating irregular web traffic among all the transitions.

Similar to Chapter 4, the guessing probability [82, 109] is used to evaluate the probability of successfully guessing user identity in one try, based on the equivalence relation on most likely sequences.

Definition 30 (Guessing probability) *Given an equivalence relation X on a set of most likely sequences MLS in terms of a secret sec and a probability distribution pd : $MLS \rightarrow [0, 1]$, the guessing probability for secret sec is defined by*

$$Gue(sec|X) = \sum_{X_i \in X} gue(X_i), \tag{5.11}$$

where

$$gue(X_i) = \max_{mls_k \in X_i} pd(mls_k) \tag{5.12}$$

is the vulnerability of equivalence class X_i, i.e. the worst-case probability the secret can be guessed in one try.

Example 4 (Guessing probability) Given a set E containing five transitions in terms of a secret *sec* and a set of their most likely sequences MLS, the probabilities for the most likely sequences are:

$Pr(mls_{e_{1_1}}|e_1) = 0.9$ and $Pr(mls_{e_{1_2}}|e_1) = 0.1$;
$Pr(mls_{e_{2_1}}|e_2) = 1$;
$Pr(mls_{e_{3_1}}|e_3) = 0.9$ and $Pr(mls_{e_{3_2}}|e_3) = 0.1$;
$Pr(mls_{e_{4_1}}|e_4) = 0.8$ and $Pr(mls_{e_{4_2}}|e_4) = 0.2$; and
$Pr(mls_{e_{5_1}}|e_5) = 0.7$ and $Pr(mls_{e_{5_2}}|e_5) = 0.3$.

Among them $mls_{e_{1_2}} = mls_{e_{3_2}} = mls_{e_{4_2}} = mls_{e_{5_2}} = \text{NULL}$.

Given an equivalence relation X on set MLS, suppose that $mls_{e_{1_1}}, mls_{e_{2_1}}$ and $mls_{e_{4_1}}$ are classified in class X_1, $mls_{e_{3_1}}$ and $mls_{e_{5_1}}$ in class X_2, and the NULL patterns are classified in the third class X_3.

With a uniform distribution on transitions E, the guessing probability in terms of secret *sec* is given, based on the equivalence relation X as:

$$Gue(sec|X) = g(X_1) + g(X_2) + g(X_3) = 1/5 * (1 + 0.9 + 0.3) = 0.44$$

Assume that an observation for transition e_i will not follow any traffic pattern for transition e_j, where $e_i \neq e_j \in E$. Hence this research does not consider a *false positive rate* caused by an observation for transition e_i mistakenly matched to a traffic pattern for transition e_j. The false positive rate can be evaluated in the future work.

5.6 Experimental Results

This section presents the experimental results of the in-depth analysis related to the 27 websites.

Given 50 Google accounts, in original the priori chance of guessing a user before observing web traffic is 0.02. The analysis used the following configurations: threshold of indistinguishable packets $Tp = 0.1$, displacement range $d = 2$, and thresholds of confidence, of similarity and of most likely sequences $Tc = Tm = Tt = 0.7$.

5.6.1 Direct Transitions

Between testing scenarios 1, 2 or 3, the direct transitions in test cases are only dependent on user accounts, regardless of the websites the indirect transitions communicate with.

By analysing the web traffic of direct transitions in different scenarios and in different testing rounds, in general, the guessing probabilities of user identities from direct transitions are all less than or around 0.1 at each of the five locations.

In the following sections, therefore, our focus is turned to the analysis of indirect transitions. Without particular explanation, the guessing probabilities mentioned are for indirect transitions.

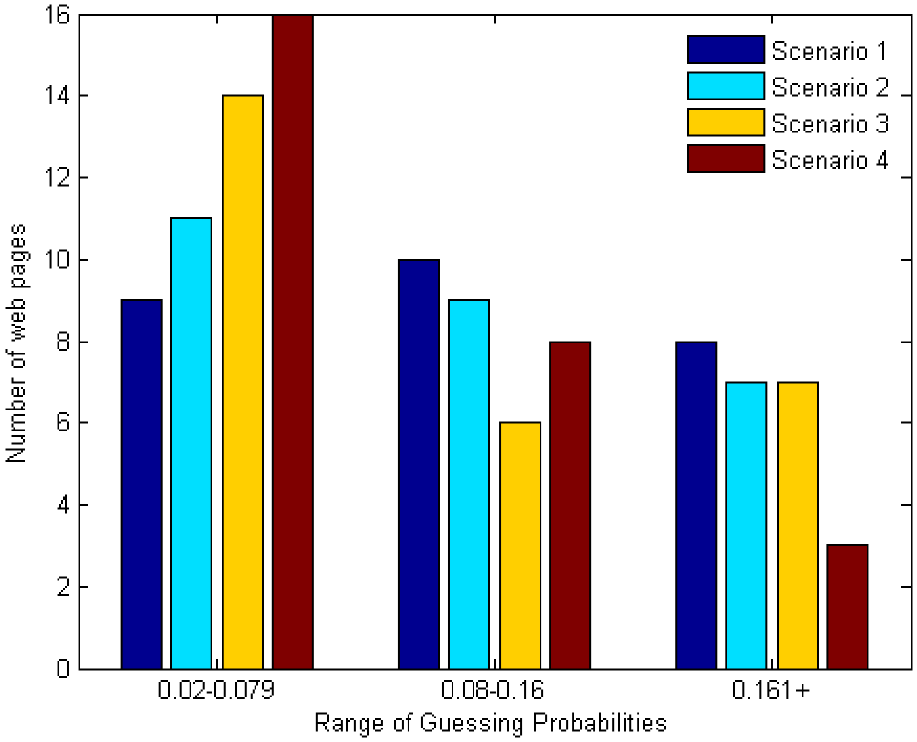


Figure 5.2: Distributions of websites in each testing scenario

5.6.2 Overview of the Website Distribution

Figure 5.2 gives an overview of the distribution of websites in each testing scenario in terms of the average guessing probabilities among five locations. X-axis displays the range of guessing probabilities, and y-axis indicates the number of websites. Each bar displays the number of websites, out of the 27, holding the guessing probabilities in a specific interval in a testing scenario. The statistics are based on the average values of two testing rounds.

Guessing probabilities are given into three intervals: "0.02-0.079", "0.08-0.16" and "0.161+". A website with a guessing probability in interval "0.02-0.079" is considered revealing few user identities. While those within "0.161+" probably leak large amounts of user identities, at least from the appearance which is eightfold increase in comparison to the prior guessing probability. However, the accurate leakage should be evaluated entirely among four scenarios.

As shown in Figure 5.2, most websites in scenario 1 are distributed in a range of guessing probabilities not less than 0.08, when the user accounts are logged in during the indirect transitions.

On the contrast, most websites in scenario 4 have a guessing probability between 0.02 and 0.079. In fact, user identities are not leaked since there is no information related to user identities in scenario 4. In an ideal environment, the web traffic collected for a test case should be consistent, and the ideal guessing probability should be 0.02, identical to the priori chance.

However, as shown in Figure 5.2, a multitude of guessing probabilities in scenario 4 are larger than 0.02, which can be supposed that there are some external elements in no relation to user identities leading to the varying web traffic. And the varying web traffic can be mistakenly considered being caused by the internal factors related to user identities.

Therefore, the guessing probability in one scenario is not sufficient to precisely evaluate the leakage of user identities. We regard a guessing probability obtained in scenario 4 as the ***reference level*** to estimate the variation of web traffic affected by external factors. By eliminating the influence from the external factors, leakage of user identities can be measured more precisely.

We believed that the guessing probabilities in scenarios 2 and 4 should be similar. However, from the distribution of websites, it can be seen that there are websites whose guessing probabilities in these two scenarios are not similar. Hence we guess that the difference is caused by, e.g., either noise or other factor, which makes the web traffic in the two scenarios inconsistent. More details are analysed later.

5.6.3 Leaking User Identities

We assume that the web traffic for an indirect transition examined in a testing scenario can fingerprint the user when it is consistent regardless of the testing locations, testing machines, testing time, etc. In other words, similar guessing probabilities will be obtained among different testing rounds and different locations.

This chapter considers, therefore, that indirect transitions for a website leak user identities if

(1) the guessing probabilities generated at different locations are consistent. The five variation trends of the guessing probabilities in four scenarios, each generated from a location, generally follow a consistent variation pattern; and

(2) the highest gap between two guessing probabilities among any two scenarios should be large enough.

Condition (1) considers the websites with consistent leakage, regardless of the external factors such as locations.

For condition (2), currently we only examine the leakage when the largest gap between guessing probabilities is originated from scenarios 1 and 4, i.e. the highest is from scenario 1 and the lowest from scenario 4. Leakages in other cases are left for future investigation.

Analysing Variation Trends of Guessing Probabilities

To ensure the variation trends of guessing probabilities from different locations are consistent, we execute test cases for each website at least two rounds at each location, except the testing sites in Berlin and Neuchatel. This is because that the two testing sites are unavailable after a round of testing for all the websites.

Next we describe the process of determining which websites for which the indirect transitions generate consistent variation trends of guessing probabilities among locations.

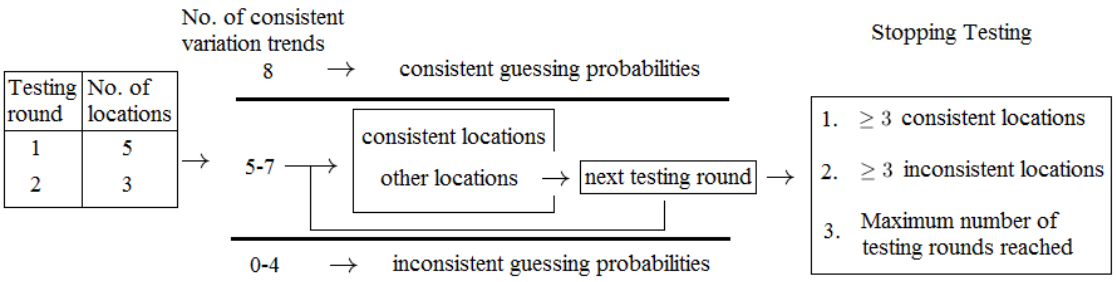

Testing round
No. of locations
1
5
2
3
No. of consistent variation trends
8
consistent guessing probabilities
5-7
consistent locations
other locations
next testing round
0-4
inconsistent guessing probabilities
Stopping Testing
1. ≥ 3 consistent locations
2. ≥ 3 inconsistent locations
3. Maximum number of testing rounds reached

Figure 5.3: Process of testing for each website

Figure 5.3 illustrates the process.

1. First two testing rounds

   In the beginning, test cases for each website were executed two rounds at each location, excluding the testing sites in Berlin and Neuchatel, where only one testing round was performed.

   After two rounds of testing, for each website, eight variation trends of guessing probabilities in four scenarios were generated, five from the five locations in the first round and the other three from the second round.

   If more than half of the variation trends, i.e. at least five trends are generally consistent with each other, the average variation trend of the average guessing probabilities among the consistent variation trends is regarded as a ***variation pattern*** of guessing probabilities for the website.

   If both of the two variation trends generated in a location are consistent with the variation pattern, the location is considered as a ***consistent location***. For the only one variation trend generated in Berlin or Neuchatel, if it follows the variation pattern, then the location is also regarded as the consistent location.

   On the other hand, for a location from which at least one variation trend is inconsistent with the variation pattern, test cases in terms of the website will be executed in the next testing round at the location.

   If all the five locations are regarded as consistent locations, i.e. all the eight variation trends are consistent, this website is considered generating consistent variation trends between locations, and the testing for this website is stopped.

   On the contrary, when at most half of the variation trends, i.e. four variation trends are consistent, it is supposed that the variation trends of guessing probabilities are unstable. Then the leakage of the user identity with regard to this website will not be analysed.

2. More testing rounds

   From the third testing round, as mentioned, test cases are only executed in the locations which are not considered as consistent locations. At the end of a testing round, for each location, if all the variation trends generated in the location, including those generated from previous testing rounds, are inconsistent with the variation pattern, this location is regarded as an ***inconsistent location***.

On the contrary, if more than half of the variation trends generated from a testing site are consistent with the variation pattern, the location is now regarded as a consistent location. Otherwise, test cases for the website will be tested in the next round at this location.

The testing at a location is repeated until (1) the location is deemed as the consistent location, i.e. more than half of the variation trends generated in this location are consistent with the variation pattern, or (2) the testing for the website is stopped.

3. Stopping testing

   At the end of each testing round, except the first two round, if at least three locations are accepted as consistent locations, the website is regarded as generating consistent guessing probabilities. Then the testing on the website is stopped.

   In contrast, if at least three locations are considered as inconsistent locations, the guessing probabilities for this website are inconsistent. Then the testing for the website is stopped and the leakage of the user identity from the indirect transitions with this website is not analysed.

   Moreover, the testing for the website is stopped when the maximum number of testing rounds is reached. In this research, test cases for a website were tested at most five rounds. This choice of loop bounds is decided from the experiments, as the variation trends tend to be consistent when the looping time arrives to five.

After examining the 27 websites, we obtained top six websites for which the guessing probabilities are consistent among locations, and the maximum gaps between guessing probabilities are large enough. Next section we analyse the variation trends for the six websites.

5.6.4 Leaking Patterns of Websites

Although test cases were only tested one round in Berlin and Neutchatel, the two locations are considered as consistent locations for most of the six websites. This can be deemed as another sign which indirectly manifests the high consistency of guessing probabilities between locations, as the variation trends at these two locations are consistent to the variation pattern in one try.

Variation Patterns of Guessing Probabilities

Table 5.1 shows the guessing probabilities from the five locations for each of the six websites. Statistics displayed in one cell include the guessing probabilities in the four scenarios, separated by “,”. Each value is the average of the guessing probabilities from the consistent variation trends in one scenario.

Table 5.1: Six websites in terms of guessing probabilities at each location

	livedoor.com	**ebay.de**	**bankofamerica.com**
Berlin	0.24, 0.1, 0.22, 0.1	0.22, 0.14, 0.22, 0.26	0.5, 0.38, 0.28, 0.36
Cambridge	0.38, 0.1, 0.26, 0.08	0.2, 0.02, 0.2, 0.04	0.58, 0.163, 0.18, 0.24
Imperial	0.34, 0.16, 0.28, 0.18	0.32, 0.18, 0.26, 0.16	0.6, 0.26, 0.26, 0.28
Neuchatel	0.34, 0.12, 0.16, 0.18	0.26, 0.04, 0.26, 0.04	0.54, 0.34, 0.34, 0.36
Stuttgart	0.38, 0.12, 0.38, 0.1	0.22, 0.07, 0.186, 0.02	0.02, 0.02, 0.02, 0.02
	dailymail.co.uk	**google.com.br**	**google.it**
Berlin	0.32, 0.06, 0.02, 0.08	0.36, 0.2, 0.24, 0.12	0.42, 0.2, 0.1, 0.08
Cambridge	0.2, 0.06, 0.02, 0.036	0.34, 0.1, 0.04, 0.06	0.38, 0.1, 0.02, 0.04
Imperial	0.2, 0.06, 0.06, 0.06	0.26, 0.14, 0.06, 0.14	0.48, 0.36, 0.02, 0.04
Neuchatel	0.2, 0.02, 0.02, 0.04	0.46, 0.32, 0.14, 0.08	0.4, 0.2, 0.04, 0.02
Stuttgart	0.3, 0.12, 0.06, 0.14	0.22, 0.12, 0.04, 0.06	0.18, 0.1, 0.04, 0.04

From Table 5.1, it is non-intuitive to observe the variation trends of guessing probabilities in four scenarios. Hence Figure 5.4 displays the guessing probabilities shown in Table 5.1 in a more intuitive way.

In Figure 5.4, a line plotted in a sub-figure straightly manifests the variation trend of guessing probabilities at one location. As can be seen, the guessing probabilities are generally highest in scenario 1 whereas lowest in scenario 4.

There may be a few vibrations between consistent variation trends. For example, in Figure 5.4 (4) in terms of website `dailymail.co.uk`, the guessing probabilities in London and Neuchatel are essentially identical under scenarios 2, 3 and 4, while in Berlin and Stuttgart, the statistics in scenario 3 are lower than those in scenarios 2 and 4. Nevertheless, relative to the gaps with guessing probabilities in scenario 1, the differences between the guessing probabilities in scenarios 2, 3 and 4 are fairly small. Thus it can still be considered that the variation trends are consistent among the five locations.

From the variation trends displayed in Figure 5.4, we produce three templates mainly followed by the variation trends for the six websites. As the websites are considered revealing user identities, a template is also deemed as a ***leaking pattern of user identities***.

For a website whose variation trends of guessing probabilities from indirect transitions follow a leaking pattern, the indirect transitions are considered as the vulnerabilities of leaking user identities.

Next we analyse the three leaking patterns.

1. $G(1) \simeq G(3) > G(2) \simeq G(4)$

In this leaking pattern, guessing probabilities in scenarios 1 and 3 are close to each other, and so are those in scenarios 2 and 4. However, the figures in scenarios 1 and 3 are much larger than those in scenarios 2 and 4. Symbol "$\simeq$" is used to express that two guessing probabilities are with similar values.

Compared with scenario 1, guessing probabilities in scenario 2 fall down sharply when

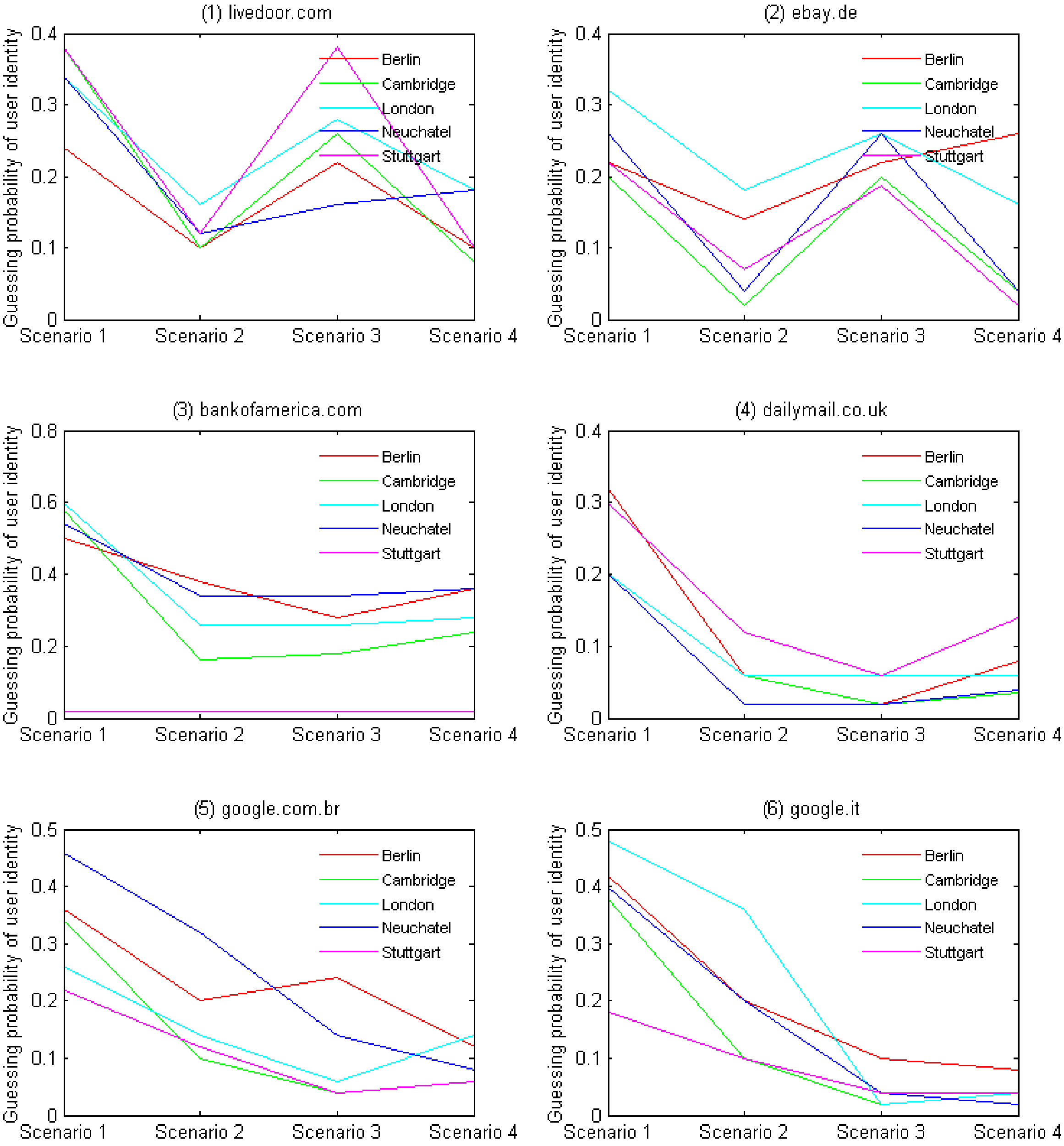


Figure 5.4: Websites with consistent variation trends of guessing probabilities

the cookies were cleared, to a same level as the values in scenario 4. This verifies what we expected as indirect transitions in both scenarios 2 and 4 should involve no information related to user accounts. On the contrary, the guessing probabilities stay at a high level as long as the cookies were retained, even when the user accounts were logged out in scenario 3.

This suggests, therefore, that cookies cause the web traffic for indirect transitions depending on users, for a website whose the variation trends of guessing probabilities follow this leaking pattern.

For websites `livedoor.com` and `ebay.de` displayed in Figures 5.4(1) and 5.4(2), their variation trends generally follow this leaking pattern, excluding that in Neuchatel for livedoor.com and that in Berlin for ebay.de.

Consider the exceptional trend in Neuchatel plotted in Figure 5.4(1). The guessing probability in scenario 3 is much lower than what we expected from the leaking pattern. On the other hand, in Figure 5.4(2), the guessing probability in Berlin in scenario 4 is much higher than the value expected to be in scenario 4.

Due to the unavailability of the testing sites in Berlin and Neuchatel from the second testing round, test cases were only examined one round at the two locations. Nevertheless, the guessing probabilities still largely follow the leaking pattern in most scenarios in one-time testing. This instead convince that the guessing probabilities from indirect transitions communicating with websites livedoor.com and ebay.de are consistent among the five locations.

When calculating the average guessing probabilities in each scenario, exceptional variation trends are not taken into account. As a result, the average guessing probabilities for website livedoor.com are 0.325, 0.12, 0.285, 0.115 in each scenario respectively, based on the statistics excluding those in Neuchatel. And the average guessing probabilities for website ebay.de are 0.25, 0.078, 0.23, 0.065 in each scenario respectively, from the data excluding those in Berlin.

In summary, this leaking pattern implies that cookies can be a leaking source of user identities which affects the web traffic in communications with external websites.

2. $G(1) > G(2) \simeq G(3) \simeq G(4)$

For websites in Figures 5.4(3) and 5.4(4), their variation trends generally follow this leaking pattern.

In this leaking pattern, guessing probabilities in scenario 1 are much higher than those in other scenarios, which are similar from each other.

Unlike in leaking pattern 1, where guessing probabilities maintain on a high level as long as the cookies are retained, in this leaking pattern the high-level guessing probabilities only come from scenario 1 when both cookies and logged user accounts were kept. This indicates that without logged user accounts, user identities are not revealed even when the cookies exist. It suggests that some information related to the logged user accounts may be propagated to the indirect transitions, but not via cookies.

Now consider the variation trends depicted in Figure 5.4(3) for bankofamerica.com. The one in Stuttgart is exceptional, where the guessing probabilities in all scenarios are 0.02, identical to the prior chance.

Examining the web traffic generated in Stuttgart, unexpectedly, it is completely identical between different user accounts, each containing simply one request packet with a same size.

Thus a question like "was the unusual web traffic caused by poor network conditions in Stuttgart when the testing was performed" may be asked. To avoid, as far as possible, the influence caused by poor network conditions, communications with website bankofamerica.com were repeatedly performed several more rounds in Stuttgart during a large time period. However, identical guessing probabilities were always obtained in every testing round. Hence the user identity is not leaked through the indirect transitions when testing in Stuttgart.

For the reason, it is supposed that web traffic generated at this testing site may be intercepted or affected by some unknown factors, e.g. the government may obscure the web traffic in communications between Google website and `bankofamerica.com`.

For website `dailymail.co.uk` shown in Figure 5.4(4), as previously mentioned, although there is a fluctuation between variation trends at different locations, however, they are still regarded as consistent with this leaking pattern.

In summary, this leaking pattern suggests that logged user accounts may be a factor which leaks user identities in communications with external websites, however, the information is not transmitted through cookies.

3. $G(1) > G(2) > G(3) \simeq G(4)$

This leaking pattern is summarised from the variation trends for internal websites `google.com.br` and `google.it` in Figures 5.4(5) and 5.4(6) respectively.

The guessing probabilities gradually decline over the first three scenarios and then level off in scenario 4, close to the value in scenario 3.

Compared with the guessing probabilities for the external websites shown in Figure 5.4(1-4), in this case when communicating with internal websites, guessing probabilities in scenario 2 are higher than those in scenario 4, which do not conform to the expectation where the guessing probabilities in scenarios 2 and 4 are similar.

Therefore, a suspicion is proposed when communications happened between Google internal websites. Google wanted to get the information about user accounts in communications. Assume that during the direct transition when a user logged into her user account, a temporary session, containing the information related to the user account and the IP address of the user, was stored in the server.

When an indirect transition was performed in scenario 3, Google detected that the cookie is available. Hence it did not request any information related to the user account as the cookie has existed and it would be used in future communications. As regards in scenario 4, no information related to the user account can be requested, therefore the guessing probabilities in scenarios 3 and 4 are close.

However, when the cookie was deleted in scenario 2, Google attempted to get the information about the user account from a different channel.

It can request for the temporary session from the direct transition saved in the server, through the match between the user's IP address in the indirect transition and that stored in the temporary session. This process can lead to the web traffic for indirect transitions varying depending on users.

It is supposed, therefore, that even when the cookies are deleted, communications between Google internal websites can still implicitly transmit the information related to user accounts. This may cause a side-channel leakage of user identities.

Now consider the variation trends for website `google.com.br` in Figure 5.4(5). Some inconsistencies exist between the variation trends in different locations. For example, in Berlin, the guessing probability in scenario 3 is higher than that in scenario 2. And in London the data in scenario 4 is higher than that in scenario 3. Nevertheless, the variation trends still generally follow this leaking pattern.

For the variation trends illustrated in Figure 5.4(6), they are much more consistent with this leaking pattern, though the largest gap between the guessing probabilities in London is fairly high, compared with those in other locations. On the other hand, the maximum gap in Stuttgart is a little bit smaller.

As for the reason that the guessing probabilities for `google.it` are more stable than those for `google.com.br`, we suppose that the communications with `google.it` happened within the Europe, while the communications with `google.com.br` were the global transmission to Brazil, which may cause the web traffic with a higher unsteady.

Therefore, it can be supposed that user identities are leaked through communications with internal websites, even when the cookies are deleted.

Summary

As shown in Figure 5.4, the guessing probabilities are mostly consistent between locations within different testing rounds. This implies that the indirect transitions communicating with external websites can really leak user identities via traffic analysis.

Furthermore, examinations under the four testing scenarios demonstrate that cookies, cleared in scenario 2, and logged user accounts, logged out in scenario 3, can really be the leaking sources providing web traffic capability of fingerprinting users.

As shown in Figure 5.4, the average increment of guessing probabilities, i.e. the maximum gap between scenarios 1 and 4, is around 0.25 when communicating with an external website. And it is around 0.3 when connecting to an internal website.

Comparing with direct transitions. Excluding website `bankofamerica.com`, the guessing probabilities on the low level for other websites in Figure 5.4, e.g. those in scenarios 2 and 4 in Figure 5.4(1), to a great extent, are around or less than 0.1. As mentioned previously, the guessing probabilities from direct transitions are generally less than or around 0.1, which are similar to the guessing probabilities on the low level from indirect transitions. This can also suggest that few amounts of user identities from direct transitions are leaked.

Moreover, we also analysed the vulnerabilities from communications without consistent variation trends.

5.6.5 Leaking User Locations

When analysing the web traffic which leaks no user identities, it is by accident discovered that there is a website whose observations completely rely on the locations. More specifically, the observations from one location always terminate with the same packet sequence, which is unique from those in other locations.

In the communications with website `craigslist.org`, one of the 25 external websites, the web traffic for indirect transitions is constant and indistinguishable from users, so that no user identities are leaked. However, the observations generated from Berlin end with a packet with size 412 in any scenarios, and the observations in Cambridge, London, Neuchatel, and Stuttgart always terminate with packet sizes 421, 415, 408 and 418 respectively.

To confirm, several more rounds of testing on this website were conducted during a large period of time. As a result, the observations always end with the same packet sizes, which are unique from locations.

This discovery is exciting, but at the same time questions like "will the observations still leak user locations when the number of testing locations increase largely", "is the location really the factor leading to the uniqueness of ending packet sizes in observations", etc. can be asked and wait for further investigation.

We suppose that there are some elements unique from testing locations which were transmitted in communications and affected the ending packet sizes of web traffic. This kind of element can be, e.g. network prefixes, which are unique from user locations. However, the experiments were carried out at different machines with unique MAC addresses. It is also possible that the uniqueness of ending packet sizes is caused by the transmission of MAC addresses. If so, then the information actually leaked is the user identity, instead of the user location. And the user identity is leaked through the MAC address, instead of the Google account.

Inspired by this finding, we examined the web traffic for the six websites in Figure 5.4, in terms of the leakage of user identities. However, the observations are independent on the five testing locations.

Therefore, our experiments so far suggest that web traffic can compromise user privacy of either user identities or user locations, yet it is incapable of revealing both of them simultaneously.

Moreover, this thesis also discusses the cases in terms of websites leaking no user privacy.

5.6.6 Leaking No Information

Figure 5.5 presents two websites in terms of the variation trends of guessing probabilities among five locations, as two typical examples of leaking no information. The data plotted was generated in the first testing round.

Case 1

In Figure 5.5(1) for website `imgur.com`, the variation trends are inconsistent between locations. Likewise, the variation trends are also inconsistent in the second testing round, which are not plotted here. Moreover, the web traffic cannot identify user locations.

Therefore, the web traffic with the guessing probabilities shown in Figure 5.5(1) can leak neither user identities nor user locations.

However, it is uncertain that what kind of information causes the inconsistent guessing probabilities. Is it because of noise, or a combination of multiple user secrets transmitted to the network? And the combination of multiple user secrets causes random observations which can actually leak, e.g., both user identities and locations? These questions can

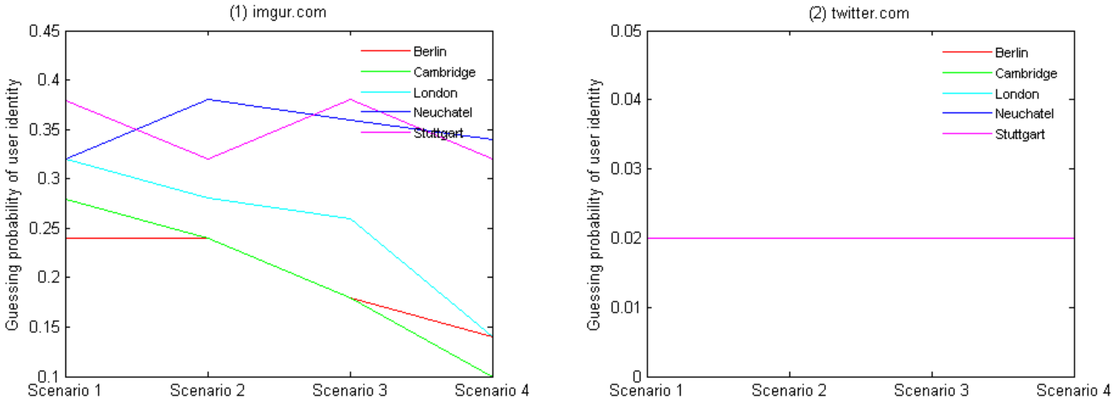


Figure 5.5: Websites without leakage

be asked and be examined in the future work, to explore the link between the irregular variation trends of guessing probabilities and the user privacy.

Case 2

Another typical case of leaking no information is when the maximum gap between the guessing probabilities is very small, regardless whether the variation trends are consistent or not.

Figure 5.5(2), in terms of website `twitter.com`, illustrates an extreme example of this case, where the guessing probabilities in each scenario are all 0.02 at each location in all the testing rounds. Moreover, we discover that the web traffic generated is totally identical in any situations, which is completely in no relation to user accounts or user locations.

Therefore, website `twitter.com` provides a perfect example of leaking no user information through the web traffic for indirect transitions. It presents the practicability of web traffic completely indistinguishable from user information, which makes a traffic analysis incapable of compromising user privacy.

Certainly, we can simply suppose that `twitter.com` did not request any information related to the users from Google website. However, it is still possible that during communications between Google and Twitter, the information related to user identities was transmitted. However, a state of the art mitigation against traffic analysis may be applied on either the web traffic or the user information, which caused the observations identical.

Hence the questions like "why does Twitter leak no user information", "does it find a solution to mitigate side-channel leakages", etc. are still attractive, which can be investigated in the future work.

On the other hand, through the communications within Twitter website, it is possible that user privacy can be leaked. For example, unique characteristics can be constructed from the posted messages, which are transmitted and which cause the web traffic depending on users during the communications within a social networking website. This kind of topic has been studied in the context of user privacy in social networking websites, e.g.

[121]. However, it is not the topic this thesis examines.

5.7 Discussion

This chapter starts an in-depth analysis of the side-channel leakage of user identities. It demonstrates a potential leaking threat which reveals user identities from Google accounts to external websites in the real world. Moreover, user location is also identifiable by traffic analysis in our experiments.

This chapter also analyses the effects of cookies and logged Google accounts on the web traffic varying depending on user identities. It is not a surprise to see that cookies leak user privacy, as it has been a well-known mean of web tracking. However, to our best knowledge, this research is the first study on cookies in side-channel leakages of user identities via traffic analysis.

On the other hand, one may come up with an argument like "is the increment of guessing probability of around 0.3 large enough to indicate that the user identity is leaked". Moreover, questions like "is the sample space too small compared with millions of Google accounts", "does the guessing probability reduce rapidly when the sample space expands" etc. are still open.

Opposite to the previous work which quantifies leakages by simulating real attacks, the experimental results in this chapter are obtained simply using the trained data. We lack the experimental validation of the leakages obtained by the analyses. In the future work, real-world attacks need to be mounted to measure the accuracy of guesses and compare with the vulnerability produced by the analyses.

Furthermore, when assessing whether a maximum gap between the guessing probabilities is large enough, we do not establish a threshold of the existence of leakage. Instead we subjectively determine whether the gap is large enough to indicate the leakage of user identities.

Even though these limitations exist, the purpose of this research is achieved. The purpose of our research is not to quantify the precise leakages in real-world web applications. Instead a primary aim of this chapter is to illustrate a new traffic-analysis threat concerning leaking user identities. The security threat is based upon user accounts on a website like Google, where a user identity can be leaked from communications with external websites.

Moreover, this chapter gives a NULL pattern for observations which do not follow the NON-NULL patterns. Although the NULL pattern is weak to present traffic features, it is a beginning in the multiple mapping between transitions and observations, which is more common in real-world communications. In this sense it is an advance towards a more precise evaluation of traffic-analysis vulnerabilities in real-world web applications.

On the whole, this chapter demonstrates a new security threat in side-channel attacks in web applications. Instead of closing a case, this research, as believed, opens a new case towards the in-depth investigation of side-channel vulnerabilities in communications with external websites.

Chapter 6

Related Work

In this chapter, we review previous work including the following contexts:

6.1 Side-Channel Vulnerabilities

6.1.1 Overview

There is a long standing and large literature on side-channel vulnerabilities. It can date back on WWII, where Bell Labs uncovered by accident that the encryption machine emitted a spike each time the machine stepped. And these spikes could unbelievably divulge the plain text of a message being enciphered by the machine. The early history of how this phenomenon was discovered is introduced in [59].

From then on, side-channel vulnerabilities on encrypted communications have been examined in various domains, including encrypted Voice over IP (VoIP) conversations [123, 122], multimedia data streaming [105], keyboard acoustic emanations [23, 126] and cryptographic systems [73, 61].

For example, in encrypted VoIP conversations, techniques such as variable bit rate (VBR) are used to encode audio for saving bandwidth. Wright et al. [123] demonstrate that the lengths of encrypted compressed VoIP packets can be exploited to identify the language spoken in an encrypted conversation. Moreover, phrases spoken within an encrypted call can also be revealed by encrypted compressed packet lengths [122].

Keyboard acoustic emanations provide a possibility of revealing typed characters on keyboard-like input devices, differentiating the sounds emanated by different keys [23, 126]. Apart from acoustic emanations, keystrokes can also be revealed by timing attacks. Song et al. [110] infer key sequences through the time differences between each two keys pressed, as an individual IP packet is sent to the remote machine immediately after every key is pressed.

Timing attacks have also been exploited on cryptographic implementations. For example, [73] demonstrates how to obtain secret keys by measuring the amount of time required for operating cryptographic computations in, e.g. Diffie-Hellman [51], RSA

[101], and DSS [57] systems.

Especially, cache based side-channel attacks have been mounted on cryptographic implementations, e.g. [31, 21]. They break cryptosystems through exploring cache states of cache access mechanisms. For example, in time-driven cache attacks the total time needed of performing certain computations can be obtained to infer the number of cache hits and misses during encryptions [31].

Another important category of side-channel vulnerabilities is traffic analysis in web applications, which have arisen increasing public concerns over the past two decades. An adversary's intention is to infer web users' online activities, such as the websites they have visited.

6.1.2 Side-Channel Leakages in Web Applications

In 1996, Wagner and Schneier [119] first supposed the leakage of visited web pages via traffic analysis in a personal communication. More specifically, they propose that the examination of ciphertext lengths may reveal the URL of a website from the GET requests under a SSL encrypted transport.

Leaking User Browsing History

The first actual experiment was implemented in 1998 by Cheng and Avnur [40]. They discover that a user's browsing pages can be revealed by observing the highly unique HTML file sizes from the encrypted traffic.

From then on, a large amount of work has studied the attacks, via traffic features, against user privacy, such as user browsing pages [40, 111], browsing websites [35, 49, 62, 64, 76] and sensitive inputs [39, 37, 125].

The general approach of traffic analysis is first to train the web traffic. By extracting packet features, e.g. packet sizes, directions and sequence numbers, traffic patterns of communications can be built in terms of each secret value. Then given a traffic sequence observed in a communication, a user's secret can be inferred or at least the space of possible values can be narrowed down by comparing the observation with the traffic patterns.

In addition, timing attacks have also been mounted to compromise user privacy, e.g. [33, 56]. Felten and Schneider [56] first propose a cache-based timing attack against the response time of accessing a static web page, to determine if the web page has been visited before. [33] exposes user privacy information, e.g. the validity of a user name at a site, by measuring the response time a website takes to assemble dynamic web pages. Ling et al. [77] propose an attack of inferring the website a user visited based on the network delay. The attacker can differentiate websites by measuring the sample means and the sample variances of the round-trip time (RTT) between a victim user and the websites.

Side-channel attacks in web applications have been implemented through encrypted communications by using, e.g. SSL/TLS [40, 111, 39, 64]. They have also been mounted with anonymity networks, e.g. anonymized NetFlows data [46], onion routing [91, 120], and proxies [64, 111, 49, 76]. Anonymity networks claim to provide anonymous surfing, but previous work demonstrates that they are still fail to defeat traffic analysis against, e.g. websites [64, 35] and a client's identity [70, 84].

Leaking User Sensitive Inputs and Detecting Tools

In 2010, the work by Chen et al. [39] turned focus of side-channel attacks from inferring web pages/sites visited to obtaining the user sensitive data on a website. They investigate real-world websites and find out that user health conditions, financial statuses, search inputs, etc. can be largely leaked through traffic analysis, based on the packet sizes and directions.

Then the first automated tool–Sidebuster, for detecting and quantifying side-channel leakages of user sensitive information, is proposed by Zhang et al. [125]. Sidebuster implements taint analysis on the source code of Struts-based web applications. It aims to obtain web widgets containing sensitive data. Then a tester manually configures execution paths containing those tainted widgets. By testing the execution paths and analysing the web traffic collected, they can identify the execution paths through the distinguishable web traffic.

Instead of depending on the source code, Chapman and Evans [37] propose a black-box analysis for detecting side-channel vulnerabilities in real-world web applications. However, their approach still takes a large amount of manual work in configuring test cases.

Motivated by the drawback of manual generation of test cases, we propose SideAuto [66]. It is a tool which aims to resolve the manual generation of test cases. Static analysis and symbolic execution are exploited to construct the structure of a web application and then build the test cases automatically. Therefore, SideAuto complements and extends Sidebuster towards a complete side-channel analysis.

Countermeasures

Mitigation systems for timing side channels have been proposed, e.g. [33, 87, 106]. Bortz et al. [33] conclude that fixing total response time for thwarting timing attacks is insufficient. Chunked encoding may leak information through inter-chunk timings even though the total response time is constant. Therefore they propose a countermeasure which fixes the inter-chunk timings.

A large scale of countermeasures have also been proposed for preventing traffic analysis, using e.g. padding [111, 124, 78], dummy [79], etc. Panchenko et al. [91] provide a *Camouglage* countermeasure, which obfuscates web traffic by randomly loading several web pages with the actual requested page simultaneously. Wright et al. [124] optimally morph one class of web traffic to look like another class of traffic.

HTTPOS [79], a client-side system, applies a suite of traffic transformation techniques on a browser to obfuscate encrypted web traffic. The defences are carried out through modifying four fundamental traffic features including packet sizes, timing of packets, web object sizes, and flow sizes.

In 2012, however, Dyer et al. [52] demonstrated that nine known countermeasures, including the popular padding-based countermeasures, are still vulnerable to prevent web site/page fingerprinting.

Formalise Traffic Analysis

In theory, some work has provided formal frameworks in terms of side-channel attacks. For example, Backes et al. [24] introduce a framework for the derivation of formal guarantees against side-channel traffic. They model the countermeasures which are constructed by the composition of basic techniques, such as padding, dummy, and split. And their model can be extended to consider advanced features like caching and timing behaviours. Moreover, they propose a framework of *path-aware countermeasures*, which require that the induced partitions have a property which ensures behavioural equivalence of states.

6.2 Fingerprinting Users

The ubiquity of 802.11 wireless networks can disclose user identities by transmitting the unique MAC addresses. Techniques, e.g. pseudonyms [68] have been proposed to mask MAC addresses. However, [92] demonstrates that implicit identifiers of 802.11 traffic, e.g. network destinations, broadcast packet sizes, etc. can be used to track users even when the unique identifiers of MAC addresses have been removed. Moreover, Verde et al. [117] propose an approach of fingerprinting users hidden behind NAT by exploiting the NetFlows in an wireless network.

In anonymity networks which particularly hiding identities of endpoints, practical traffic analysis have been proposed for compromising user identities, e.g. [36, 84]. In [36], an attacker exploits traffic fluctuations from characterising bandwidth variations to compromise the anonymity of Tor users. [84] mounts a side-channel attack by using throughput information to identify the guard relay of a Tor user and to determine whether two connections belonging to a same user. Though it is not really about identifying user identities, the attack serves as a stepping stone for completely de-anonymising users.

Additionally, there has been an increasing public concern about the disclosure of online user identities with the widespread usage of social media [121, 80]. Wondracek et al. [121] introduce a de-anonymising attack triggered by an evil website a user visits, which examines the group memberships of a user on social networking sites, i.e. the groups in a social network to which the user belongs. The information of group memberships is obtained by mounting a web browser history stealing attack. Then the attacker uses the group memberships to identify users.

Traffic analysis has also been used to recognise users in social networks. Pironti et al. [95] report a specific side-channel attack based on the packet sizes of the profile pictures downloaded when users log into their accounts on a social networking site. This attack is under an assumption that each user has a profile picture configured.

Device fingerprinting has also been concerned in terms of breaking user anonymity. For example, [53] shows that trackers can recognise users without the needs of cookies, through fingerprinting the properties of browsers and their plug-ins. In [90], Nikiforakis et al. show how popular commercial browser-fingerprinting approaches work on the Internet and how fragile the browser ecosystem is to defend against the device fingerprinting in terms of user identities.

In this thesis, however, a novel traffic-analysis scenario of fingerprinting users is proposed. It is, as believed, a more common scenario of inferring user identities. User identities, based upon user accounts on a website like Google, can be recognised via traffic analysis in communications with third-party websites.

6.3 Leakage of User Locations

Location-based services (LBS) are a general class of services which use location information to provide useful information for users. LBS are accessible with mobile devices to locate a user or an object's position, e.g. a GPS service and to find the nearest Chinese restaurant using a smartphone.

While the number of LBS and the advanced techniques have mushroomed, the abuse of location information can cause monetary losses or even endanger human lives [83]. For example, [5] reports a case where a GPS device is used in stalking a victim's location. This leads to a numerous amount of work on detecting location-aware vulnerabilities in wireless sensor networks, e.g. [83], and providing LBS without leaking user locations, e.g. [85].

A traditional way for web servers to identify a user location is to use the IP address [96]. With the wide-spread use of anonymous networks, however, users can hide their real IP addresses to web servers.

Recently, Jia et al. [67] demonstrate a new timing attack against user locations without direct access to GPS or IP addresses. Their approach is based on the browser caches from location-oriented websites. A location-oriented website, e.g. Google, redirects a user to the website in the location the user lives in. A malicious website a user visits can detect the user's location by measuring the loading time of web objects located at different locations. If the difference between the time of loading an object twice in a location using the victim's browser is less than a threshold, it indicates that the user has visited this site in the specific location previously.

On the other hand, this thesis demonstrates a likelihood of revealing user locations via traffic analysis based on packet sizes and directions. It discovers that different packet features can be generated when a website is accessed from different locations. Although we do not particularly examine the vulnerabilities for user locations, this discovery directs a new channel of leaking user locations through traffic analysis.

6.4 Path Generation

It is not a new topic in terms of automated generation of execution paths in the context of testing web applications. Techniques including UML e.g. [100], static analysis e.g. [113] and dynamic testing e.g. [22, 100] can be used to generate test cases.

For example, Rica and Tonella [100] propose an analysis based on the UML models of web applications. It downloads the web pages of a target website by sending requests using the user inputs provided by the testers. Then the UML model of the website can be built to generate test cases.

Artzi et al. [22] present a technique based on dynamic test generation, using combined concrete, symbolic execution and constraint solving. Symbolic execution is first performed to generate path constraints. Then the recorded path constraints are used to create new inputs that exercise different further control flow. This process is automatically and iteratively performed.

In 2011, Tkachuk and Rajan [113] proposed an automated driver generation for analysing JSP-based web applications. Their approach constructs a Page Transition Graph (PTG) for a web application, by encoding the application's pages, events, event-handlers and user data to the PTG. Then a PTG-based driver is implemented using Java PathFinder to produce test sequences based on the PTG.

Inspired by [113], we propose SideAuto which achieves an automated analysis of side-channel vulnerabilities containing test case generation for web applications. SideAuto uses a novel static analysis and symbolic execution to generate test cases. It is the first tool towards the fully automated side-channel analysis in Struts web applications, which integrates the test case generation into the detection of side-channel vulnerabilities. Furthermore, we also propose a black-box approach of generating test cases for real-world web applications.

6.5 Hidden Markov Models

Hidden Markov Model (HMM) was first published in a series of classic papers by Baum and his colleges [27, 28, 29, 30] in the late 1960s and early 1970s, to model a system with unobserved states in mathematical structure.

One of the earliest practical work based on the HMM was [25], which models a continuous speech-recognition system in the mid of 1970s. From 1980s, HMMs have been widespread understood and largely used in speech processing [69, 75, 98]. [97] is an excellent tutorial covering the basic techniques of HMM with regard to speech recognition developed during the period of 1980s.

Nowadays, HMMs have been successfully applied to various fields including bioinformatics [54], signature recognition [38], computer and network security [41], etc.

In network communication channels, HMMs have been used in modelling at the packet level. For example, [104] exploits the HMM to infer network states by modelling the packet flow in terms of packet loss and packet delay. Dainotti et al. [48] design a HMM for modelling Internet traffic sources from SMTP, HTTP, a network game and an instant messaging application, based on the inter-packet time and packet sizes.

In the context of side-channel attacks in web applications, HMMs have been used to model the transfer between user web activities, e.g. visited web sites/pages [40, 35, 49]. Danezis [49] uses a HMM to model the most plausible path of accessed web pages on a site, which provides the best explanation of the observed sequence of resource sizes. In the model, hidden states are the resources on a site and the outputs are the sizes of traffic data. In [35], a HMM is used to model the link structure for a website and the probable paths that a user will follow when visiting the website. Then the packet traces observed each time the user transitions from one page to another can be classified. The attacker can then compute the probability that an observed trace of packets was generated, to infer which website a user was visiting.

In our research, an approach motivated by the HMM is used to model a set of observations, to construct a traffic pattern from the observation set.

6.6 Threats of Cookies

Cookies are used to remember user inputs, user preferences, user browsing behaviours, etc. they offer powerful personalisations for improving user experience on the Internet. However, there have been security issues arisen from cookies. For example, in the recent years, cookies are abused for marketing purposes of tracking users without user permission, e.g. see [114].

On the other hand, to the best of our knowledge, there has been a little study on cookies in traffic analysis for web applications. Rizzo and Duong [102] demonstrate a side-channel vulnerability related to cookies. They propose *CRIME*, a side-channel attack through compression lengths of cookies, to unveil the information stored in cookies.

This thesis, from a different angle, demonstrates that cookies can be a factor which affects traffic patterns and results in side-channel vulnerabilities.

6.7 Quantitative Information Flow

Quantitative information flow (QIF) is first proposed by Denning [103], which determines how much information flows from high level to low level. It measures the amount of information flow caused by interference between variables using information theoretic quantities.

Smith [109] considers a new foundation based on the concept of vulnerability, which applies Renyi's min-entropy rather than Shannon entropy to measure uncertainty.

Clark et al. [42, 43, 44, 45] develop a system of syntax-directed analysis rules to give the analysis of information flow in a simple deterministic while language. However, the bounds for loops are over pessimistic, if any leakage is possibly leaked in a loop then all the security information are leaked via the loop according to their rules. Then Malacaria [81] gives a more precise quantitative analysis of loop constructs by using the partition property of entropy. Heusser and Malacaria [63] analyse leakages in real-work software of the LINUX kernel. They combine model checking with QIF to quantify the leakage.

In side-channel attacks, QIF has been exploited in cryptographic implementations, e.g. [74], and web applications, e.g. [24, 125]. Kopf and Basin [74] provide a model of adaptive side-channel attacks against cryptographic algorithms. QIF is then used to measure leakages in cryptographic systems.

In 2010, Mu [86] gave an overview of the quantitative information flow in information security.

For quantifying side-channel leakages in web applications, Sidebuster [125] measures leakages in Struts web applications, based on Shannon entropy and conditional entropy. Backes et al. [24] provide a framework of formal guarantees against traffic analysis in web applications. And they measure the leakages using QIF.

In our research, SideAuto uses Shannon entropy and min entropy to evaluate leakages in Struts web applications. And then in the study on real-world web applications in Chapters 4 and 5, the guessing probability is exploited to measure the leaks.

Chapter 7

Conclusion and Future work

7.1 Conclusion

7.1.1 Overview

This research conducts an extensive study of analysing side-channel vulnerabilities in web applications through traffic analysis. Test case generation is brought in to advance on a fully automated side-channel analysis in web applications. This research aims to assist developers in detecting side-channel vulnerabilities in their web applications.

We examine side-channel vulnerabilities of user privacy include user web activities and user identities. It also coarsely involves user locations. A wider range of communications are analysed, including those implicitly transmitting sensitive formation and those interacting with external websites.

Moreover, the leaking factors which cause side-channel vulnerabilities through traffic analysis are also analysed. We discover that cookies and logged user accounts can be the sources which lead to varying web traffic depending on user identities.

More specifically, we summarise the thesis from the practical and the theoretical sides as follows.

7.1.2 From the Practical Side

On the practical side, this thesis proposes the implementations for analysing Struts-based Java applications and real-world web applications.

Our earliest work proposes a framework for automating the analysis of side-channel vulnerabilities in Struts-based web applications. A main advance of this work is the automated generation of test cases for web applications. We use a novel approach to achieve the automation of test case generation, by combining static analysis with symbolic execution. Test cases are generated in terms of examined secrets and then the leakages are evaluated.

The techniques are implemented into a tool–SideAuto. SideAuto is then evaluated over six real-world or simulated web applications. Our study shows that the system works well and effectively on these web applications with acceptable overheads. The details are present in Chapter 3.

Next our focus is turned to real-world web applications, which are not limited to the Struts framework.

A black-box analysis is proposed to detect which transitions are vulnerable to leak user privacy. An automated algorithm is developed to generate test cases automatically. Individual transitions are examined, including those explicitly and implicitly involving sensitive information. The analysis is applied to four real-world web applications.

The experimental results show that transitions which appear to have no relation to user sensitive information can, actually, reveal more user secrets than those in explicit relation to user sensitive information. Moreover, the experiments on Google website demonstrate that the user identities can be largely leaked from Google accounts, through transitions implicitly interacting with sensitive information. The details are described in Chapter 4.

Inspired by the surprising result of large leaks of user identities from Google accounts, we then conduct an in-depth study of the leakage of user identities on Google user accounts.

We examine the leaks of user identities from fifty Google user accounts through communications with Alexa Top 150 websites. Experimental results show that user identities can be revealed through communications between Google website and external websites. Moreover, it is shown that user locations may also be leaked through traffic analysis.

Furthermore, four testing scenarios are designed to explore the leaking sources. We discover that cookies and logged user accounts can be the factors which cause web traffic to leak user identities. More details are presented in Chapter 5.

7.1.3 From the Theoretical Side

On the theoretical side, we evaluate leakages using quantitative information flow. Shannon entropy, min entropy and conditional entropy are used to quantify leaks in terms of the uncertainty of user privacy in Struts-based web applications in Chapter 3. And the guessing probability is also used to evaluate the probability of correctly guessing a secret in one try in Chapters 4 and 5.

When constructing a most likely sequence, we develop an approach motivated by hidden Markov models to find the solution which best explains the web traffic collected. Then the distance between an observation and the traffic pattern is measured based on an optimised Damerau-Levenshtein distance with super transpositions and shifts. We then calculate the probabilities of observations from transitions to build a probability distribution in terms of the most likely sequences.

Moreover, we propose an advanced fingerprinting model of "one → many" mapping where a single transition associates to at least one traffic patterns. Compared with a "one → one" mapping, this mapping is closer to the real-world cases, as a transition may generate varying web traffic. Hence the "one → many" mapping considers a bigger range of web traffic analysed. It opens a new research direction in analysing web traffic.

Above all, this thesis provides an extensive automated analysis of traffic-analysis vulnerabilities in web applications from new perspectives. Therefore it is believed that this research opens new cases in analysing side-channel vulnerabilities in real-world web applications, which make future work more exciting.

7.1.4 Limitations

In this research, we evaluate traffic-analysis vulnerabilities in web applications. However, we do not perform experimental validations to justify the correctness of the results obtained by the analyses. In Chapters 4 and 5, for example, we do not validate that the most likely sequences derived are correct. More specifically, we generated the most likely sequences using a novel method. However, we do not validate that these most likely sequences are same as those generated using a standard approach. This means that we lack the guarantee that the most likely sequences generated are correct.

On the other hand, for the leakages evaluated in the experiments, we also lack validations of the results produced by the analyses presented. More precisely, we do not validate that the accuracy of the leakages obtained. We do not perform real attacks on the web applications, to justify that the leakages of user privacy are really as much as the data produced by the analyses.

Without validations, one may ask questions such as "how can you say that the results of the analyses are accurate", "how can you say that the web applications really leak user privacy", etc. Although this research aims to discover possible side-channel vulnerabilities, instead of estimating precise leakages, the errors of results may lead the developers to a wrong way, where some important vulnerabilities are missed but some insignificant vulnerabilities are investigated. Therefore, future work needs to overcome these limitations.

7.2 Future Work

As mentioned, an immediate work can be to validate the results of the analyses produced, both the most likely sequences and the leakages. For the most likely sequences, the hidden Markov model can be really used to produce the traffic patterns. By comparing the traffic patterns derived by the hidden Markov model with those generated using our approach, we can validate the correctness of the most likely sequences produced.

For the experimental results of leakages, real-world attacks in web applications can be mounted to calculate the accuracy of guesses based on observations of web traffic. Comparing the accuracy of guesses with the leakages obtained by the analysis, the experimental results can be validated.

Some more future directions derived from this research are suggested in the following.

One is to further investigate the leaks of user locations in traffic-analysis attacks. It can determine if a user location is actually leaked via the observations of web traffic, and then check what sources lead to the varying web traffic.

In terms of communications with third-party websites, this research analysed leakages when the maximum distance of guessing probabilities coming from scenarios 1 and 4. However, it is possible that the user privacy can be revealed from communications when the maximum gap comes from other two scenarios. Hence one future work can be extended to analyse leakages from communications with the maximum distances of guessing probabilities from any two scenarios.

Furthermore, the future work can build a "one → many" mapping with multiple specific traffic patterns for each transition, instead of using a weak NULL pattern which is used in our work. With the multiple specific traffic patterns, a larger space of observations likely to be occurred in a communication are taken into consideration. This will improve the accuracy in traffic mapping, which can reduce the false positive rate of mistakenly matching an observation and traffic patterns.

Bibliography

[1] Alexa top 500 global sites. http://www.alexa.com/topsites.

[2] Apache tomcat. http://web.archive.org/web/20080207010024/.

[3] Crawljax. http://crawljax.com.

[4] Database. http://phpweby.com/software/ip2country/, retrieved in January, 2013.

[5] Foxs news. man accused of stalking ex-girlfriend with gps. http://www.foxnews.com/story/0,2933,131487,00.html.

[6] Google app engine. https://appengine.google.com/.

[7] Htmlunit - welcome to htmlunit. http://hhtmlunit.sourceforge.net/.

[8] Jasper. http://tjws.sourceforge.net/jasper.html.

[9] Java pathfinder. http://babelfish.arc.nasa.gov/trac/jpf.

[10] Jpcap-a network packet capture library for applications written in java. http://jpcap.sourceforge.net/.

[11] Nation information checker. https://test-student.appspot.com/.

[12] Online banking system. https://hxj-bank.appspot.com/.

[13] Planetlab. https://www.planet-lab.eu/.

[14] Selenium webdriver. http://docs.seleniumhq.org/projects/webdriver/.

[15] Struts cookbook. https://hxj-cookbook.appspot.com/.

[16] Struts framework. http://struts.apache.org/.

[17] Struts mvc model. http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.etools.struts.doc

[18] Tax claim system. https://hxj-tax.appspot.com/.

[19] This figure is retrieved from *Struts framework and the model-view-controller design pattern*. http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.1.1/com.ibm.etools.struts.doc/topics/cstrdoc001.html.

[20] University identity system. https://test-student.appspot.com/index1.jsp.

[21] O. Acıiçmez and Ç. K. Koç. Trace-driven cache attacks on aes (short paper). In *Information and Communications Security*, pages 112–121. Springer, 2006.

[22] S. Artzi, A. Kieżun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst. Finding bugs in web applications using dynamic test generation and explicit-state model checking. *Software Engineering, IEEE Transactions on*, 36(4):474–494, 2010.

[23] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *Procedings of the IEEE Symposium on Security and Privacy*, pages 3–11. IEEE, 2004.

[24] M. Backes, G. Doychev, and B. Kopf. Preventing side-channel leaks in web traffic: A formal approach. In *Proceddings of the 20th Network and Distributed Systems Security Symposium*, 2013.

[25] J. Baker. The dragon system–an overview. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23(1):24–29, Feb 1975.

[26] T. Ball. The concept of dynamic analysis. In *Software EngineeringESEC/FSE99*, pages 216–234. Springer, 1999.

[27] L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[28] L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bull. Amer. Math. Soc*, 73(3):360–363, 1967.

[29] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, pages 1554–1563, 1966.

[30] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, pages 164–171, 1970.

[31] D. J. Bernstein. Cache-timing attacks on aes, 2005.

[32] J. M. Bland and D. G. Altman. Statistics notes: measurement error. *Bmj*, 313(7059):744, 1996.

[33] A. Bortz and D. Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th international conference on World Wide Web*, pages 621–628. ACM, 2007.

[34] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, Apr. 1998.

[35] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and Communications Security*, pages 605–616. ACM, 2012.

[36] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *Computer Security–ESORICS 2010*, pages 249–267. Springer, 2010.

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

[38] M.-Y. Chen, A. Kundu, and J. Zhou. Off-line handwritten word recognition using a hidden markov model type stochastic network. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(5):481–496, 1994.

[39] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Procedings of the IEEE Symposium on Security and Privacy*, pages 191–206, 2010.

[40] H. Cheng and R. Avnur. Traffic analysis of ssl encrypted web browsing. *URL citeseer. ist. psu. edu/656522. html*, 1998.

[41] S.-B. Cho and S.-J. Han. Two sophisticated techniques to improve hmm-based intrusion detection systems. In *Recent Advances in Intrusion Detection*, pages 207–219. Springer, 2003.

[42] D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science*, 59(3):238–251, 2002.

[43] D. Clark, S. Hunt, and P. Malacaria. Quantified interference for a while language. *Electronic Notes in Theoretical Computer Science*, 112:149–166, 2005.

[44] D. Clark, S. Hunt, and P. Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation*, 15(2):181–199, 2005.

[45] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[46] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, M. K. Reiter, et al. On web browsing privacy in anonymized netflows. In *Proceedings of the 16th USENIX Security Symposium*, pages 339–352, 2007.

[47] T. M. Cover and J. A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[48] A. Dainotti, A. Pescapé, P. S. Rossi, F. Palmieri, and G. Ventre. Internet traffic modeling by means of hidden markov models. *Computer Networks*, 52(14):2645–2662, 2008.

[49] G. Danezis. Traffic analysis of the http protocol over tls. *Unpublished draft*, 2010.

[50] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, July 1977.

[51] W. Diffie and M. E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[52] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 332–346. IEEE Computer Society, 2012.

[53] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[54] S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14(9):755–763, 1998.

[55] W. Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.

[56] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 25–32. ACM, 2000.

[57] P. FIPS. 186-2. digital signature standard (dss). *National Institute of Standards and Technology (NIST)*, 2000.

[58] G. D. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[59] J. Friedman. Tempest: A signal problem. *Cryptologic Spectrum*, 2007.

[60] J. A. Goguen and J. Meseguer. *Security policies and security models.* IEEE, 1982.

[61] D. Gullasch, E. Bangerter, and S. Krenn. Cache games–bringing access-based cache attacks on aes to practice. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 490–505. IEEE, 2011.

[62] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42. ACM, 2009.

[63] J. Heusser and P. Malacaria. Quantifying information leaks in software. In *ACSAC*, pages 261–269, 2010.

[64] A. Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pages 171–178. Springer, 2003.

[65] X. Huang. It leaks more than you think: Fingerprinting users from web traffic analysis. *Acta Informatica Pragensia*, 2015(3):206–225, 2015.

[66] X. Huang and P. Malacaria. Sideauto: quantitative information flow for side-channel leakage in web applications. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 285–290. ACM, 2013.

[67] Y. Jia, X. Dong, Z. Liang, and P. Saxena. I know where you've been: Geo-inference attacks via the browser cache. *Internet Computing, IEEE*, 19(1):44–53, 2015.

[68] T. Jiang, H. J. Wang, and Y.-C. Hu. Preserving location privacy in wireless lans. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 246–257. ACM, 2007.

[69] B.-H. F. Juang. On the hidden markov model and dynamic time warping for speech recognitiona unified view. *AT&T Bell Laboratories Technical Journal*, 63(7):1213–1243, 1984.

[70] D. Kedogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In *Information Hiding*, pages 53–69. Springer, 2003.

[71] M. E. Khan et al. Different approaches to white box testing technique for finding errors. *International Journal of Software Engineering and Its Applications*, 5(3):1–14, 2011.

[72] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 17(7):385–394, 1976.

[73] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology-CRYPTO'96*, pages 104–113. Springer, 1996.

[74] B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 286–296. ACM, 2007.

[75] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell System Technical Journal, The*, 62(4):1035–1074, 1983.

[76] M. Liberatore and B. N. Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 255–263. ACM, 2006.

[77] Z. Ling, J. Luo, Y. Zhang, M. Yang, X. Fu, and W. Yu. A novel network delay based side-channel attack: Modeling and defense. In *INFOCOM, 2012 Proceedings IEEE*, pages 2390–2398. IEEE, 2012.

[78] W. M. Liu, L. Wang, K. Ren, P. Cheng, and M. Debbabi. k-indistinguishable traffic padding in web applications. In *Privacy Enhancing Technologies*, pages 79–99. Springer Berlin Heidelberg, 2012.

[79] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *In Proc. Network and Distributed Systems Symposium (NDSS)*, 2011.

[80] M. Madden. Privacy management on social media sites. *Pew Internet Report*, pages 1–20, 2012.

[81] P. Malacaria. Assessing security threats of looping constructs. In *ACM SIGPLAN Notices*, volume 42, pages 225–235. ACM, 2007.

[82] P. Malacaria. Algebraic foundations for information theoretical, probabilistic and guessability measures of information flow. *arXiv preprint arXiv:1101.3453*, 2011.

[83] K. Mehta, D. Liu, and M. Wright. Protecting location privacy in sensor networks against a global eavesdropper. *Mobile Computing, IEEE Transactions on*, 11(2):320–336, 2012.

[84] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 215–226. ACM, 2011.

[85] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: query processing for location services without compromising privacy. In *Proceedings of the 32nd international conference on Very large data bases*, pages 763–774. VLDB Endowment, 2006.

[86] C. Mu. Quantitative information flow for security: A survey technical report: Tr-08-06 (updated on 2010).

[87] Y. Nagami, D. Miyamoto, H. Hazeyama, and Y. Kadobayashi. An independent evaluation of web timing attack and its countermeasure. In *Third International Conference an Availability, Reliability and Security*, pages 1319–1324. IEEEComputerSociety, 2008.

[88] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[89] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis*. Springer, 1999.

[90] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting.

In *Security and privacy (SP), 2013 IEEE symposium on*, pages 541–555. IEEE, 2013.

[91] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

[92] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 99–110. ACM, 2007.

[93] C. S. Păsăreanu and N. Rungta. Symbolic pathfinder: symbolic execution of java bytecode. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 179–180. ACM, 2010.

[94] R. Patton. *Software testing.* Sams Pub., 2006.

[95] A. Pironti, P.-Y. Strub, and K. Bhargavan. Identifying Website Users by TLS Traffic Analysis: New Attacks and Effective Countermeasures. Research Report RR-8067, INRIA, Sept. 2012.

[96] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. Ip geolocation databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011.

[97] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[98] L. R. Rabiner and B.-H. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[99] T. Reps, T. Ball, M. Das, and J. Larus. *The use of program profiling for software maintenance with applications to the year 2000 problem.* Springer, 1997.

[100] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd international conference on Software engineering*, pages 25–34. IEEE Computer Society, 2001.

[101] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[102] J. Rizzo and T. Duong. The crime attack, 2012. Presented at ekoparty' 12. http://goo.gl/mlw1X1.

[103] D. E. Robling Denning. *Cryptography and data security.* Addison-Wesley Longman Publishing Co., Inc., 1982.

[104] K. Salamatian and S. Vaton. Hidden markov modeling for network communication channels. In *ACM SIGMETRICS Performance Evaluation Review*, volume 29, pages 92–101. ACM, 2001.

[105] T. S. Saponas, J. Lester, C. Hartung, S. Agarwal, and T. Kohno. Devices that tell on you: Privacy trends in consumer ubiquitous computing. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 5:1–5:16. USENIX Association, 2007.

[106] S. Schinzel. An efficient mitigation method for timing side channels on the web. In *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2011.

[107] E. J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 317–331. IEEE, 2010.

[108] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[109] G. Smith. On the foundations of quantitative information flow. In *Foundations of Software Science and Computational Structures*, pages 288–302, 2009.

[110] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, volume 2001, 2001.

[111] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 19–30. IEEE, 2002.

[112] K. Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[113] O. Tkachuk and S. Rajan. Automated driver generation for analysis of web applications. In *Proceedings of the Fundamental Approaches to Software Engineering*, pages 326–340. Springer Berlin Heidelberg, 2011.

[114] E. Toch, Y. Wang, and L. F. Cranor. Personalization and privacy: a survey of privacy risks and remedies in personalization-based systems. *User Modeling and User-Adapted Interaction*, 22(1-2):203–220, 2012.

[115] R. Valle-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan. Soot: A java bytecode optimization framework. In *CASCON First Decade High Impact Papers*, pages 214–224. IBM Press, March 2010.

[116] R. Vallee-Rai and L. J. Hendren. Jimple: Simplifying java bytecode for analyses and transformations. Technical report, 1998.

[117] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi. No nat'd user left behind: Fingerprinting users behind nat from netflow records alone. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 218–227. IEEE, 2014.

[118] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

[119] D. Wagner and B. Schneier. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, 1996.

[120] T. Wang and I. Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 201–212. ACM, 2013.

[121] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to deanonymize social network users. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 223–238. IEEE, 2010.

[122] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Security and Privacy (SP), 2008 IEEE Symposium on*, pages 35–49. IEEE, 2008.

[123] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob? In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 4:1–4:12, 2007.

[124] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceddings of the Network and Distributed Systems Security Symposium*, pages 237–250, 2009.

[125] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen. Sidebuster: automated detection and quantification of side-channel leaks in web application development. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 595–606. ACM, 2010.

[126] L. Zhuang, F. Zhou, and J. Tygar. Keyboard acoustic emanations revisited. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 373–382. ACM, 2005.