DEVELOPMENT OF A SECURE MOBILE GPS TRACKING AND MANAGEMENT SYSTEM

by

Anyi Liu

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

Dr. Jim X. Chen, Dissertation Director

Dr. Daniel Carr, Committee Member

Dr. Fei Li, Committee Member

Dr. Harry Wechsler, Committee Member

Dr. Daniel A. Menasce, Senior Associate Dean

Dr. Lloyd J. Griffiths, Dean, Volgenau School of Engineering

Date: 12/09/2011

Fall Semester 2011
George Mason University
Fairfax, VA

Development of a Secure Mobile GPS Tracking and Management System

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Anyi Liu
Master of Science
Dalian University of Technology, China, 2001
Bachelor of Engineering
Dalian University of Technology, China, 1997

Director: Dr. Jim X. Chen, Professor
Department of Computer Science

Fall Semester 2011
George Mason University
Fairfax, VA

# Dedication

I dedicate this dissertation to my parents, Risheng Liu and Lianzhi Li.
And my wife Jian Zhao.

# Acknowledgments

Upon the completion of this dissertation, I wish to express my gratitude to those people whose advice, guidance, support and understanding have enriched my studies and research here in the department of Computer Science at George Mason University. I am heartily thankful to my advisor, Professor Jim X. Chen, who introduced me to the problem of research, and guided me throughout my studies on how to approach and solve the problem. He gave his generous support and encouragement during my Ph.D. study, and has been a great role model for me in my academic career. I hope that one day I would become as good an advisor as he has been to me.

I have also benefited a lot from working with Professor Daniel Carr, Professor Fei Li, and Professor Harry Wechsler who served on my advisory committee and gave me useful advice. I would like to thank them for their time, and for all the invaluable feedback and suggestions they have given me about my research. The understanding and consideration the advisory committee so generously extended to me are much needed and deeply appreciated.

In addition to my academic committee, I would like to thank Dr. Daniel Menascé, Dr. Donna Fox, and Ms. Lisa Nolder for their invaluable and timely help on the process of my Ph.D. extension. Their help and effort deserve my deepest appreciation.

I wish to particularly thank my friends Dr. Xintong (Art) Lin and Dr. Yingjiu Li for their vision and invaluable advise during the hardest time of my Ph.D. studies. Without them, my Ph.D. studies would not have been accomplished. I also wish to thank Dr. Li Yang from the University of Tennessee at Chattanooga for providing useful comments and constant encouragement during the whole process of my Ph.D. studies. Finally, I wish to thank Dr. Don Barnes and Mrs. Karen Barnes for their great comments and tremendous patience of revising the drafts of this dissertation.

I am deeply grateful, and very fortunate, to have the love and unconditional support of a small group of people who have somehow managed to endure me for this long despite my best efforts. So to Rui He, Hui Lin, Jing Jin, Yongquan Jiang, Qi Xing, Wenzhen Yang, Bin He, Xiaoxu Wang, Hao Sun, Jin Zhang, Ning Xu, Wu Li, Di Zhang, Donghai He, Yuan Li, and my pastor Boli Zhang and Xiang Chen. I say thank you for everything. You have enriched my life more than you know.

Finally, and most importantly, I can scarcely find the words to express my thanks to my family: my parents, Risheng Liu and Lianzhi Li; my parents-in-law, Liguo Zhao and Yanfang Zhang; and my wife Jian Zhao. A thousand dissertations would not be enough to convey my love and gratitude for everything they have done and endured. This work is humbly dedicated to them.

# Table of Contents

# List of Tables

# List of Figures

# Abstract

DEVELOPMENT OF A SECURE MOBILE GPS TRACKING AND MANAGEMENT SYSTEM

Anyi Liu, PhD

George Mason University, 2011

Dissertation Director: Dr. Jim X. Chen

With increasing demand of mobile devices and cloud computing, it becomes increasingly important to develop efficient mobile application and its secured backend, such as web applications and virtualization environment. This dissertation reports a systematic study of mobile application development and the security issues of its related backend.

First, to standardize the software development of mobile application, we design an efficient mobile application that investigate the key issues of mobile application development, such as location tracking, embedded database management (EDBM), and wireless communication. Our application has been implemented and commercialized on Window Mobile smartphones.

Second, to prevent SQL injection attacks (SQLIAs), we propose a black-box input validation approach, which harnesses the effectiveness of genetic and input validation algorithms to dynamically extract users' inputs and detect malicious SQL control queries. Compared to state-of-the-art protection approaches, our method does not require any code changes on either the client, the web-server, or the back-end database. To evaluate the overhead and the detection performance of our system, we have implemented the SQLProb and tested it by using benchmark SQL attacks. Our experimental results show that we can detect all

known SQL injection attacks while maintaining very low resource utilization.

Third, to protect user's private information from being exfiltrated to outside attacker, we propose a architectural solution to detect covert channels in real-time. Our intrusion detection system, namely *Observer*, runs a secure virtual machine that mimics the malicious virtual machine so that any differences between two virtual machines can be identified in real time. Unlike most existing signature or anomaly-based covert channel detection approaches, Observer does not require any legitimate data to build a normal behavior model. To evaluate Observer, we have run covert channels and detected them in real-time. Our experimental results demonstrate that Observer can detect most covert storage channels with a high detection rate and low latency and overhead.

Lastly, to detect more advanced covert channel attacks, such as covert timing channels (CTCs), we design a novel metric that can quantitatively measure the difference between the timing patterns of normal and CTCs. The key challenge we are facing is to detect CTC online in a environment, where accurate time keeping might be affected by many dynamic conditions. Our wavelet-based metric can quantitatively measure the distance between the outbound networking flows of benign VMs and malicious VMs, which contains CTCs. In addition, this online approach reduces the whole procedure of modeling legitimate traffic while remains transparent to end-users. Our experimental result demonstrates a high detection and a low false positive rate in detecting different CTC attacks.

# Chapter 1: Introduction

Tremendous attention has been put on the advances of computer hardware and software. On one hand, small and relatively inexpensive handhold devices such as personal digital assistants (PDA), tablet computers, and mobile phones have become indispensable tools for today's highly mobile workforce. On the other hand, more and more personal and enterprise users have been focused on how to sweep up data and program from personal computer (PC) to a remote server or networked virtual environment [33]. Networked virtual environment refers to a computer networking environment that connects simulated computer environment, or virtual machine (VM), on the top of a given host hardware and software platform. Indeed, reports show that the number of virtual machines has exceeded that of physical servers since 2009 [79], and mobile internet users will eventually outnumber the desktop internet users in 2014 [94].

While the advances of both technologies improve productivity remarkably, it is critical to develop a framework that integrates mobile application, web application, and networked virtual environment together. The application should provide key functionalities specifically adapt to mobilized utilities, such as location tracking, data management, and wireless communication. In addition, the computational power and storage of the mobile application should be extendable when connect to web applications and a networked virtual environment. As this multi-level framework has been used, it also pose new risks to personal user and organizations. A shift of hacker's target from PC to mobile devices, web application, and virtual machines has made attacks more sophisticated and more dangerous than ever [56].

In light of these challenges, current research efforts have been put on developing efficient mobile application and its secured backend, which includes the hardening of web applications and virtual machines. For mobile application development, many development

environments that allow a developer to write, test, and deploy applications into the target platform environment have been developed, such as Android [5], iOS SDK [73], and Windows Mobile [130]. For web application security, different categories approaches of detecting and preventing code-injection attacks have been developed, such as input validation [19,30, 45,54,99–101], static analysis [11,42,42,68,72,132], learning-based technologies [9,47,118], and dynamic prevention [11,20,88,113]. For virtual machine security, most of the covert channel detection approaches assume that sufficient quantity of legitimate networking traffic is available to model the legitimate traffic patterns [12,13,21,39,40,71,92,104,122–124].

In this dissertation, we explore different issues of mobile application development and web and virtual machine security. Specifically, we address the key functionalities specifically adapt to mobilized utilities, such as location tracking, data management, and wireless communication. In addition, we address two challenging intrusion detection and prevention problems for web application and virtual machine security: 1) preventing code-injection attacks for web applications, and 2) detecting covert storage/timing channels in virtual machines. In Chapter 3, we study an input-validation approach to preventing SQL injection attacks. Our approach uses an input-validation proxy as an additional layer between web application and its database, and thus avoids the complication of static analysis and learning, and is highly adaptive to most existing web applications.

In Chapter 4 and Chapter 5, we investigate the intrusion system design and detective metric development. With the observation that inside secrete can be transmitted the outside attacker once a vulnerable VM has been compromised, we emphasis on the outbound traffic of virtual machines. Compared to the unnoticeable deviation between legitimate traffic, a greater deviation can be measured between a benign VM and compromised VM that contains covert channel. In the consideration that virtual machine lacks precise time keeping mechanism, our matric is robust enough to detect covert channel even with the presence of noises.

## 1.1 Contributions

Our research contributions are summarized as follows. I have designed and implemented:

1. **A new mobile application**

   Software development on mobile devices increasingly attracts more attention in recent years. Due to different hardware platforms and software developing IDEs, most mobile software development lacks a systematic methodology and scalable framework to facilitate effective functionalities. We creat a multi-level mobile application framework that supports front-end users, web interaction, and networked virtual environment that provides additional computational power and storage. On our framework, we implement a mobile application that provides functionalities of location tracking, game management, embedded database management, and blue-tooth communication. The mobile application has been deployed successfully on commercial mobile phones.

2. **A new input-validation approach to prevent code-injection attacks towards web application**

   Preventing against code-injection attacks towards web applications such as SQL-injection (SQLIAs) and Cross-site script (XSSs) is of increasing interest in recent years. However, due to the complication of distinguishing non-executable *data* and executable *code*, code-injection attacks are particularly difficult to prevent. The existing prevention schemes depend on code analysis, code instrumentation, or modeling based on a large number of training data, which are non-effective to detect most code-inject attacks. We introduce a new black-box approach to prevent most existing SQLIAs. Based on the observation that web applications use a common memory space to keep query code and user input data, we develop a new algorithm to extract user input from the application-generated query, and a new algorithm to validate the extracted user input in the context of the application-generated query's syntactic structure. Our experiment results demonstrate that our system based on our new approach has 100% detection rate in detecting most existing SQLIAs.

3. **A new architectural solution to detect covert storage channels**

   Despite extensive research, covert storage channels (CSCs) are a principal threat to information security. Most detecting techniques model legitimate network traffic. However, such approaches may not be applicable in a networked virtual environments where legitimate traffic may not be available. We design and implement a real-time covert channel detection system, *Observer*. *Observer* does not require legitimate traffic for modeling, and can be dynamically deployed in a networked virtual environment where people are interested in investigating suspicious virtual machines. In addition, *Observer* is implemented as a transparent bridge, so that it is not detectable to malicious user. The experiment results show that *Observer* can detect CSCs with a high success rate, low latency, and low overhead.

4. **A new metric to detect slow covert timing channels in virtual machines**

   Covert timing channels (CTCs) aim to transmit hidden information by perturbing the timing characters between consecutive packets in normal network communication. Most existing approaches either use signature-based approaches to detect known CTCs or anomaly-based approach by modeling normal networking traffic to detect unknown CTCs. Both of them fail to detect slow CTCs in a dynamic environment without normal traffic for training. We introduce a new metric that can quantitatively measure the distance between different outbound networking flows from virtual machines. Compared to existing approaches, our approach not only can detect most slow existing and zero-day CTCs without normal traffic data, but also demonstrate its detection with the presence of noises. Our experiment result shows that our approach is sensitive to CTCs, and is capable of detection them accurately.

## 1.2 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we present the design and implementation of a mobile application that supports golf game. In Chapter 3, we

present a proxy-based approach to prevent code inject attacks toward web applications. In Chapter 4, we present the design and implementation of an architectural solution to detect covert storage channels in a networked virtual environment. In Chapter 5, we design and implement a new metric to detect covert timing channels. We conclude the dissertation and outline possible directions for our future work in Chapter 6.

# Chapter 2: Designing and Implementing a Mobile GPS Tracking and Management System

## 2.1 Introduction

The advances of GPS [43], smartphones [105] and cloud computing [23] have opened the door for researchers and software developers to tackle novel applications at a distance and scale that were impossible to imagine just a couple of decades ago. This has led to the emergence of the field of mobile application development, which is quite different from the conventional application development. Compared to their desktop counterparts, mobile applications demonstrate unique features, such as different usage patterns, more applications associate with geographic information, limited size of screen, and short duration of activities due to power constraint and limited computational power [98]. Developing efficient and scalable mobile applications is always welcome yet challenging.

Most existing mobile application development has been focused on the stand-alone software development. Although such development can be improved through advances in hardware and software, such as graphics processing unit (GPU) [64], multi-core processor [34], and novel algorithms, the current mobile application development still lacks a systematic methodology and a scalable framework that supports front-end users, web interaction, and networked virtual environments that provide additional computational power and storage.

In this chapter, a multi-level mobile application framework is described that integrates front-end mobile application, web services, and networked virtual environment together so as to provide additional computational power and storage. In particular, some key functions are described and implemented, that have been considered as the most general functions provided by mobile applications, such as geographic position tracking, real-time graphic displaying, database management, and wireless communication.

This chapter is organized as follows. Section 2.2 briefly describes a specific example of a multi-level mobile application. Section 2.3 presents a detailed description the system overview and the major components involved. Section 2.6 concludes the chapter with a brief summary of the application and some discussion of future steps in the development process.

## 2.2 Application Example

In this section, we present an actual example of a multi-level mobile application framework, named *EarthChildren*. EarthChildren is designed to enhance the sporting experience of golfers by providing them geographic information, game managing, and game information exchange.

Figure 2.1 shows the working procedure of EarthChildren. When a golf shot is about to be taken, a golfer's mobile device, e.g., smartphone [105], receives messages from satellites, which contain the geographic information about the golfer's current position and the intended goal. The geographic information can be used for many purposes: 1) retrieving maps of the golf course that covers the position; 2) displaying the current position of the golfer on her mobile device; 3) calculating angular and distant quantities and displaying them on the golfer's mobile device (Step 1 in Figure 2.1). In addition, the distance information can also be used to recommend a particular club to the golfer based on her previous performance and expressed preferences in such situation. During the game, a group of golfers can exchange personal and game information with each other through wireless communication media, such as Wi-Fi or Bluetooth (Step 2). After the entire game, the personal and game information can be saved in the golfer's personal computer (PC) (Step 3) or uploaded onto a web application and saved in its database (Step 4) for future reference in the current of future games. Of course, golfers can also upload information to their rented virtual machine(s) in Cloud [23], which provides supplementary computational power and backup storage of the mobile device (Step 5).

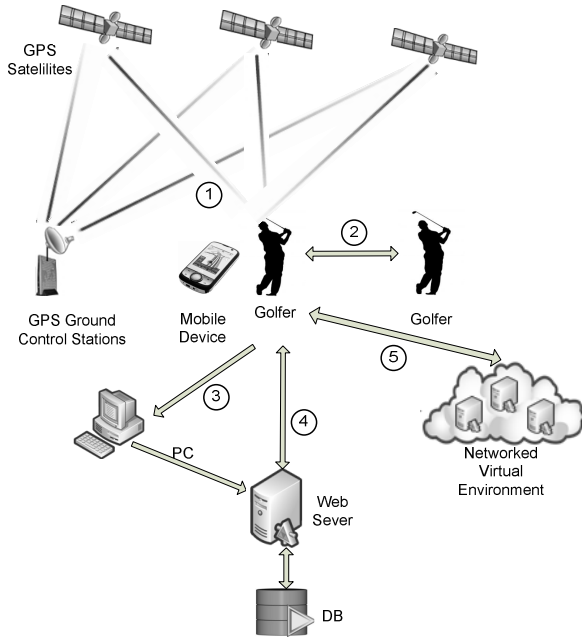



Figure 2.1: EarthChildren, the application example

## 2.3 System Design

### 2.3.1 EarthChildren System Overview

Figure 2.2 illustrates the architecture of EarthChildren, which contains five major components that can all be accessed from the main interface. 1) *Location Tracker* receives geographic location information from the satellites, and calculates the latitude and longitude of that position. 2) *Graphic Displayer* displays the current position on the mobile device's graphic user interface (GUI), as well as loads maps and icons based on the position from the embedded database (EDB). 3) *Game Manager* calculates distance and angular quantities, recommends golf clubs, manages the golfer's personal information, and saves the game records in the EDB. The game manager can also display a games which has been saved previously. 4) *Communicator* supports Bluetooth communication between golfers' mobile devices. 5)*Database Manager* bridges the communication between the above four components and the EDB. The design and implementation details of each component are described in the subsections that follow below.

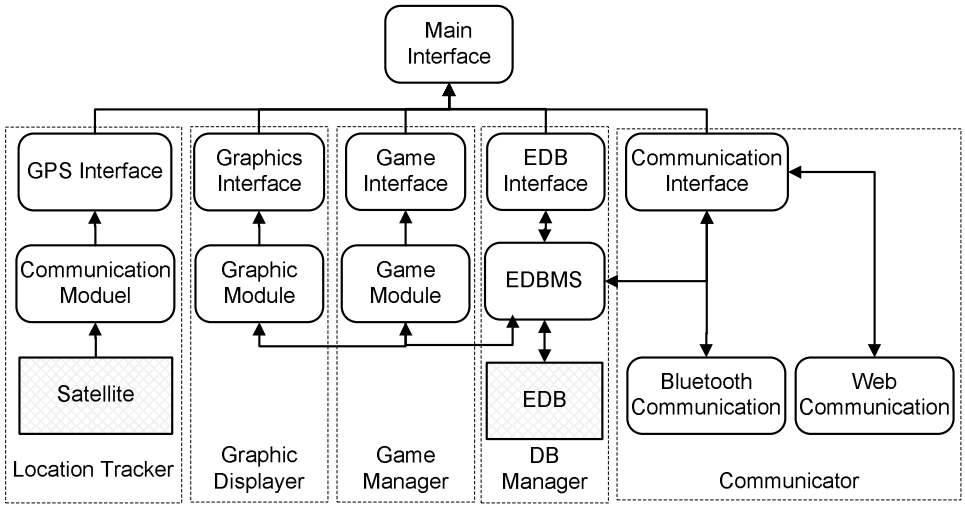



Figure 2.2: The system architecture of EarthChildren

Figure 2.3 shows the detailed software architecture of EarthChildren that is generated by Enterprise Architect [109], which contains the definition of classes and their members (variables and functions). The system was implemented on Windows Mobile [77, 130] with a size of 16MB, containing 105 source files (.cpp and .h).

### 2.3.2 Definition

In this section, we define the notations that are used in the rest of the chapter.

- We denote the a geographic point $e_i$ in the earth coordinate system as $e_i = (x, y)$, where $x$'s and $y$'s coordinates of $e_i$ are denoted as $e_i.x$ and $e_i.y$, respectively. To display $e_i$ on the screen of a mobile device, a function $F$ that converts a geographic point $e_i$ to a screen point $p_i = (x, y)$, such that $F(e_i.x) = p_i.x$ and $F(e_i.y) = p_i.y$.

- The vector from $e_i$ to $e_j$ is denoted as $V_{(i,j)} = (e_i.x - e_j.x, e_i.y - e_j.y)$. The *linear* distance between points $e_i$ and $e_j$ is the Euclidean distance between them, which is denoted as $Dist(e_i, e_j) = \sqrt{(e_i.x - e_j.x)^2 + (e_i.y - e_j.y)^2}$. Given two vectors $V_{(i,j)}$ and $V_{(i,k)}$, the angle between them is denoted as $Angle(i, j, k) = \arccos\left(\frac{V_{(i,j)} \cdot V_{(i,k)}}{\|V_{(i,j)}\|\|V_{(i,k)}\|}\right)$,

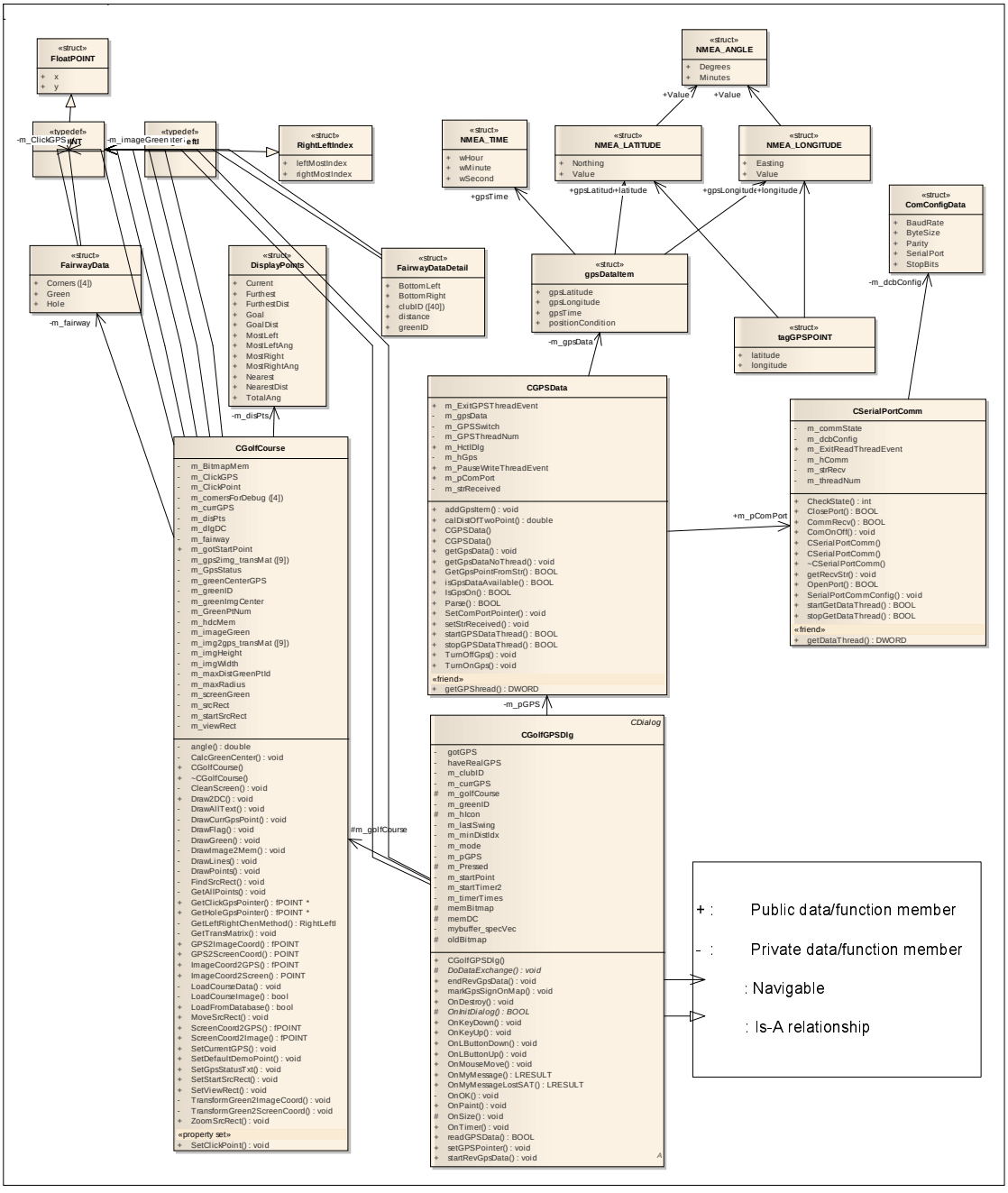
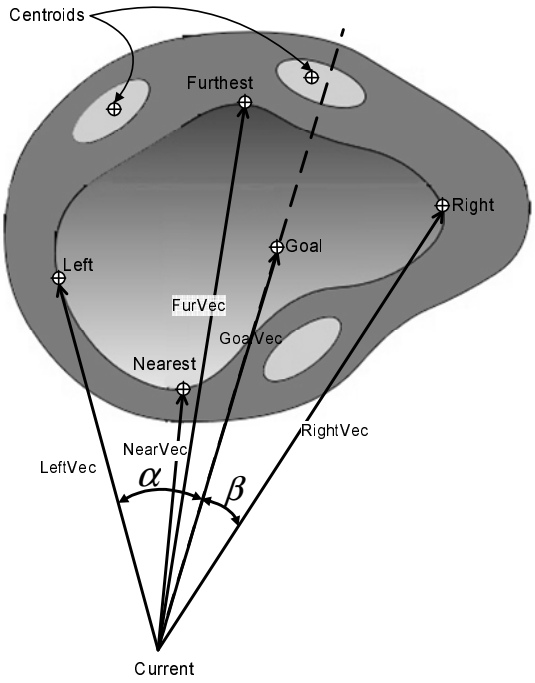



Figure 2.3: The software architecture of EarthChildren




(a) The geographic quantities that are used by EarthChildren


(b) The screen-shot of EarthChildren on a mobile device

Figure 2.4: The geographic variables and screen-shot of EarthChildren

where $V_{(i,j)} \cdot V_{(i,k)}$ is the *dot product* of these two vectors and $\|V_{(i,j)}\| = Dist(e_i, e_j)$ . We define the green of a golf course as a polygon $P_{(1,...,n)} = (e_1, ..., e_n)$. Then, the coordinates of its centroid can be calculated as $P_{(1,...,n)}.x = \frac{1}{6A}\sum_{i=1}^{n-1}(x_i+x_{i+1})(x_i y_{i+1}-x_{i+1}y_i)$ and $P_{(1,...,n)}.y = \frac{1}{6A}\sum_{i=1}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1}y_i)$, respectively [17].

- For our application, we define the following quantities that are used by EarthChildren for calculation and/or display purposes, which is illustrated in Figure 2.4(a):
  - **Current** the geographic point of the current position, which is denoted as $c$;
  - **Goal** the geographic point of the goal of the current green, which is denoted as $g$;
  - **GoalVec** the vector from point $c$ to $g$, which is denoted as $V_{(c,g)}$;

- **Furthest** the vertex on $P_{(1,\ldots,n)}$ that yields maximum $Dist(c, e_i)$ for all $1 \leq i \leq n$ and is denoted as $f$;
- **Nearest** the vertex on $P_{(1,\ldots,n)}$ that yields minimum $Dist(c, e_i)$ for all $1 \leq i \leq n$ and is denoted as $n$;
- **FurVec** the vector from point $c$ to $f$, which is denoted as $V_{(c,f)}$;
- **NearVec** the vector from point $c$ to $n$, which is denoted as $V_{(c,n)}$;
- **FurDist** the distance between point $c$ and $f$, which is denoted as $Dist(c, f)$;
- **NearDist** the distance between point $c$ and $n$, which is denoted as $Dist(c, n)$;
- **Left** the vertex on $P_{(1,\ldots,n)}$ that yields maximum anti-clock-wise angle $Angle(c, g, i)$ for all $1 \leq i \leq n$ and denoted as $l$;
- **Right** the vertex on $P_{(1,\ldots,n)}$ that yields maximum clock-wise angle $Angle(c, g, i)$ for all $1 \leq i \leq n$ and denoted as $r$;
- **LeftDist** the distance between point $c$ and $l$, which is denoted as $Dist(c, l)$;
- **RightDist** the distance between point $c$ and $r$, which is denoted as $Dist(c, r)$;
- **LeftVec** the vector from point $c$ to $l$, which is denoted as $V_{(c,l)}$;
- **RightVec** the vector from point $c$ to $r$, which is denoted as $V_{(c,r)}$;
- $\alpha$ the angle between vector $V_{(c,g)}$ and $V_{(c,l)}$, which is denoted as $Angle(c, g, l)$;
- $\beta$ the angle between vector $V_{(c,g)}$ and $V_{(c,r)}$, which is denoted as $Angle(c, g, r)$.

During the run-time, the quantities defined above displays on the screen of a mobile device as shown in Figure 2.4(b). The following subsections show the acquisition and calculation of these quantities.

### 2.3.3 Obtaining Geographic Information

EarthChildren has a built-in Global Positioning System (GPS) receiver, which receives geographic message from satellites. The principle a GPS receiver calculates its own position

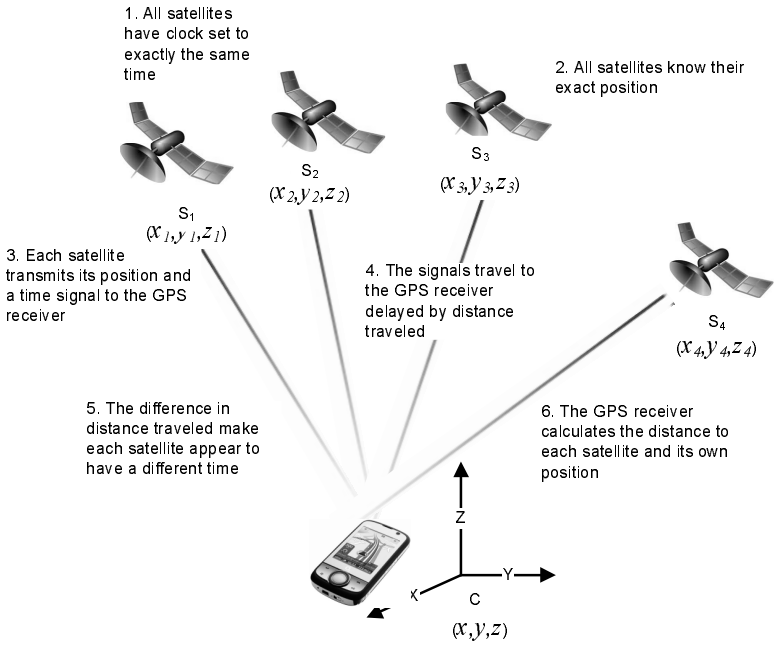



Figure 2.5: The mechanism of the GPS receiver

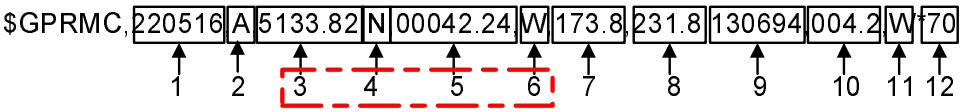
on earth based on the message it receives from satellites is shown in Figure 2.5. Assume that all the clocks of the satellites have been synchronized. Each satellite knows its own position in space and transmits the positional information $S_i = (x_i, y_i, z_i)$ and current time $t_i$ to the GPS receiver at the position $c = (x, y, z)$. Upon receiving the above information sent from satellites, the GPS receiver calculates the distance from $S_i$ to $c$ based on Equation 2.1.

$$\begin{cases} [(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2]^{\frac{1}{2}} + c(Vt_1 - Vt_0) = d_1 \\ [(x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2]^{\frac{1}{2}} + c(Vt_2 - Vt_0) = d_2 \\ [(x_3 - x)^2 + (y_3 - y)^2 + (z_3 - z)^2]^{\frac{1}{2}} + c(Vt_3 - Vt_0) = d_3 \\ [(x_4 - x)^2 + (y_4 - y)^2 + (z_4 - z)^2]^{\frac{1}{2}} + c(Vt_4 - Vt_0) = d_4 \end{cases} \tag{2.1}$$

In Equation 2.1, the speed of light is denoted as $C$ and time travels from satellite $S_i$ to the GPS receiver is denoted as $\Delta t_i (0 \leq i \leq 4)$. The distance between satellite $S_i$ and $c$ is $d_i = C \times \Delta t_i$. We also denote the clock bias error of each satellite as $VT_i$, and the clock bias error of GPS receiver as $Vt_0$. The GPS receiver uses the Equation 2.1 to solve

four unknowns $x$, $y$, $z$, and $VT_0$. It is obvious that the GPS receiver requires at least four satellites' position informatin to calculate the coordinates of the current position $c$.

Once the coordinates of $c$ have been calculated by GPS receiver, they are encoded into a standard-formatted message, namely NMEA-0183 message, and passed to a serial port for further processing. NMEA-0183 is a combined electrical and data specification for communication between marine electronic devices, such as echo sounder, sonars, anemometer, gyrocompass, and autopilot [112]. NMEA-0183 has been defined by and is controlled by the U.S. National Marine Electronics Association (NMEA), and specifies different formats of messages that are transmitting geographic information, in which Global Positioning System Fix Data (GPGGA) and Recommended Minimum Specific GPS/Transit Data (GPRMC) are the two most frequently used formats.




1 = UTC of position fix (format *hhmmss.ss*)
2 = Data status (format *x*, *A*=active, *V*=Void)
3 = Latitude (format *llll.ll*)
4 = North or South (*N* or *S*)
5 = Longitude (format *yyyyy.yy*)
6 = East or West (*E* or *W*)
7 = Speed over ground in knots (format *yyy.y*)
8 = Track angle in degrees (format *yyy.y*)
9 = UT date (format *xxxxxx*)
10 = Magnetic variation in degrees (format *yyy.y*)
11 = East or West (*E* or *W*)
12 = Checksum (format **xx*)

Figure 2.6: The fields of NMEA message in GPRMC format

Figure 2.6 and Figure 2.7 show the fields of NMEA message in GPRMC and GPGGA format, respectively. When our mobile application read a NMEA-0183 message from a serial port, it parses that message based on its format and only keeps the location-related fields, such as *latitude* (the $3^{th}$ of GPRMC or the $2^{th}$ of GPGGA), *north/south* (the $4^{th}$ of GPRMC or the $3^{th}$ of GPGGA), *longitude* (the $5^{th}$ of GPRMC or the $4^{th}$ of GPGGA), and

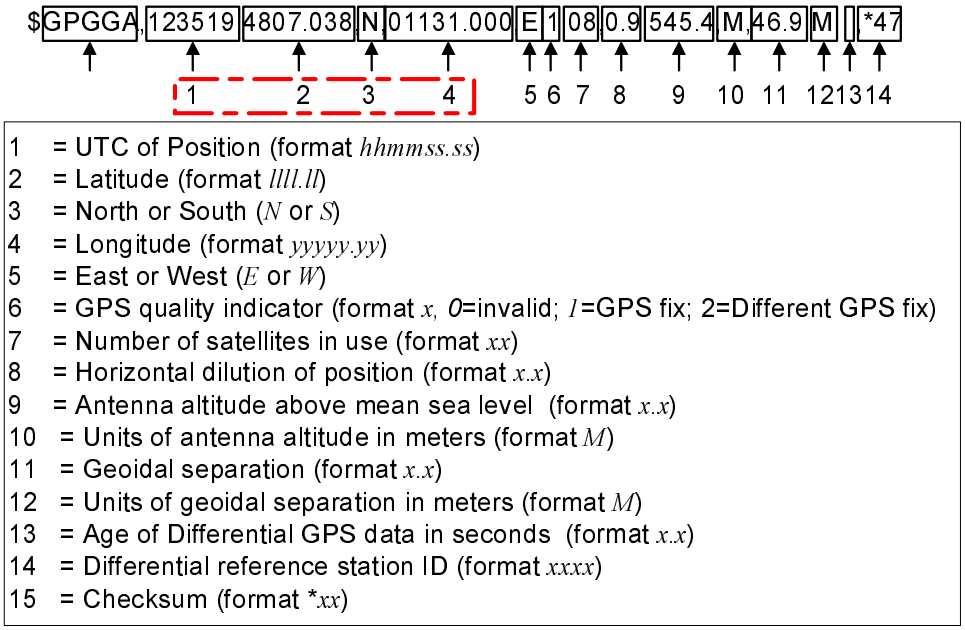



Figure 2.7: The fields of NMEA message in GPGGA format

*east/west* (the $6^{th}$ of GPRMC or the $5^{th}$ of GPGGA). The location-related fields have been highlighted by the dotted lines in Figure 2.6 and Figure 2.7.

The procedure to obtain the location-related fields is illustrated in Figure 2.8. The values of these four fields (*latitude*, *north/south*, *longitude*, *east/west*) can be used to display positions and/or to calculate distances, as discussed in the next subsection.

### 2.3.4 Displaying Points and Lines

Once a geographic point $e_i$ has been determined, the next step is to calculate distance between geographic point $A$ and point $B$, obtain the geographic quantities we defined in Section 2.3.2, and convert $e_i$ to a screen point $p_i$ on the screen of the golfer's mobile devices by using a Function $F$.

Recall that the *linear* distance between $e_i$ and $e_j$ is defined as the Euclidean distance between them. However, the *linear* distance cannot directly be used to calculate distance because this application obtains latitude and longitude on the earth, which is a sphere. To calculate the spherical distance between geographic point $A$ and point $B$ given their

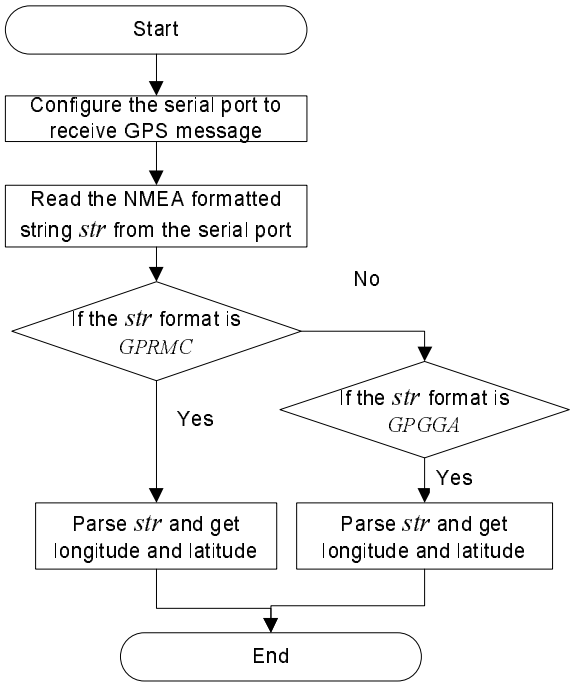



Figure 2.8: The procedure to obtain latitude and longitude

latitudes and longitudes, $p_i = (Lati_i, Long_i)$ is defined, where $Lati_i$ and $Long_i$ are the latitude and longitude of $p_i$. The latitudinal and longitudinal differences of $A$ and $B$ are given by $\Delta\phi = Lati_A - Lati_B$ and $\Delta\lambda = Long_A - Long_B$, respectively. Considering the earth as a sphere with an average radius $R = 6371.004km$ as illustrated in Figure 2.3.4, then the distance between $A$ and $B$ can be calculated from Equation 2.2 [27].

$$Dist(A, B) = 2R \cdot arcsin\sqrt{\sin^2(\frac{\Delta\phi}{2}) + \cos(Lati_A) \times \cos(Lati_B) \times \sin^2(\frac{\Delta\sigma}{2})} \qquad (2.2)$$

After that, the procedure in Table 2.1 is used to calculate the quantities that are defined in Section 2.3.2. Specifically, the procedure accepts the input of the geographic coordinates of the current green $P_{(1,\ldots,n)}$, the goal of the current green $g$, and the current position of the golfer $c$. The procedure is invoked once a new $c$ has been obtained. It runs one loop through every point of the polygon $P_{(1,\ldots,n)}$ of the green and calculates the quantities, such as the

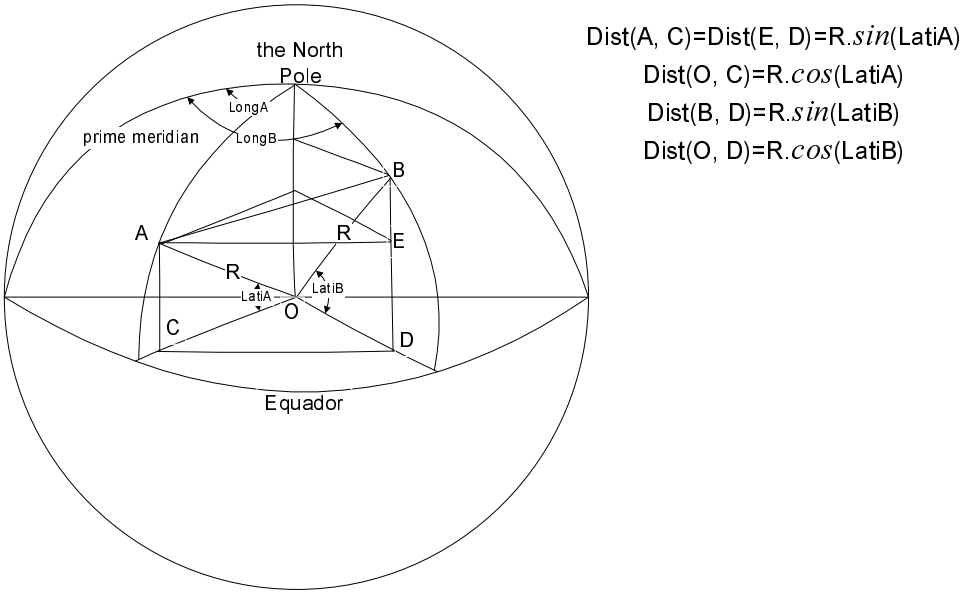



Figure 2.9: The calculation of distance between point $A$ and point $B$

coordinates ($Furthest$, $Nearest$, $Left$, and $Right$), the distances ($FurDist$, $NearDist$, $LeftDist$, $RightDist$, and $GoalDist$), and the angles ($\alpha$ and $\beta$).

Once all the necessary quantities have been determined, the final step is to convert a geographic point $e_i$ into a screen point $p_i$ of a mobile device. Remember that on the screen, two layers of images need to be displayed at the same time: the background map image and the images that display the quantities (e.g., points and lines). Therefore, $p_i$ must be drawn at the corresponding location on top of the map. To do this, our display process first converts the geographic coordinates $e_i$ into the image coordinates $i_i$, and then converts the image coordinates $i_i$ into the screen coordinates $p_i$. These two steps ensure that the quantities will be shown at the correct position on the map.

Figure 2.10 shows the two-steps of conversion. The left side of the figure illustrates the first step. The GPS coordinates of the top-left point and the bottom-right point of a golf course are denoted as $tl = (x_0, y_0)$ and $br = (x_1, y_1)$, respectively. In addition, the geographic coordinates of $e_i$ are $(x, y)$. Then, the distance from $c$ to the left boundary of the golf course is $\Delta x = x - x_0$, and the distance from $e_i$ to the top boundary of the golf course is $\Delta y = y - y_0$. Let us denote the width and the height of the background image as

Table 2.1: The procedure to calculation the geographic quantities

**Input**:The geographical coordinates of the current green $P_{(1,\ldots,n)}$, the goal $g$, and the current point $c$
**Output**:The coordinates ($Furthest$, $Nearest$, $Left$, and $Right$), the distances ($FurDist$, $NearDist$, $LeftDist$, $RightDist$, and $GoalDist$), and the angles ($\alpha$ and $\beta$)

**1.** Initialize $FurDist = 0$; $NearDist = \infty$; $LeftDist = 0$; $RightDis = 0$;
**2.** $GoalDist = Dist(c, g)$;
**3.** **for** every point $e_i$ in $P_{(1,\ldots,n)}$
  **3.1.** **if** $Dist(c, e_i) \geq FurDist$ **then**
    $FurDist = Dist(c, e_i)$; $Furthest = e_i$;
  **3.2.** **if** $Dist(c, e_i) \leq NearDist$ **then**
    $NearDist = Dist(c, e_i)$; $NearDist = e_i$;
  **3.3.** **if (i==1) then**
    $D = LeftDist = RightDist = (e_i.x - c.x) \times (g.y - c.y) - (e_i.y - c.y) \times (g.x - c.x)$;
  **3.4.** **else**
    **3.4.1.** **if** $(D < LeftDist)$ **then**
      $LeftDist = D$; $Left = e_i$;
    **3.4.2.** **if** $(D > RightDist)$ **then**
      $RightDist = D$; $Right = e_i$;
  **end**
**4.** $\alpha = Angle(c, g, l)$; $\beta = Angle(c, g, r)$;
**end**

$W$ and $H$, and thus we can get the image coordinates of $e_i$, $i_i$, from Equation 2.3.

$$i_i.x = \frac{\Delta x}{x - x_1} \times W \text{ and } i_i.y = \frac{\Delta y}{y - y_1} \times H \tag{2.3}$$

The similar calculation can be used to convert $i_i$ to $p_i$ in the second step, as illustrated in the right side of the figure. The $x$ coordinates of the left boundary and right boundary are denoted as $left.x$ and $right.x$, respectively. Similarly, the $y$ coordinates of the top boundary and bottom boundary are denoted as $top.y$ and $bottom.y$, respectively. Then, the coordinate of $p_i$ can be obtained from Equation 2.4.

$$p_i.x = \frac{i_i.x - left.x}{right.x - left.x} \times SW \text{ and } p_i.y = \frac{i_i.y - top.y}{bottom.y - top.y} \times SH \tag{2.4}$$

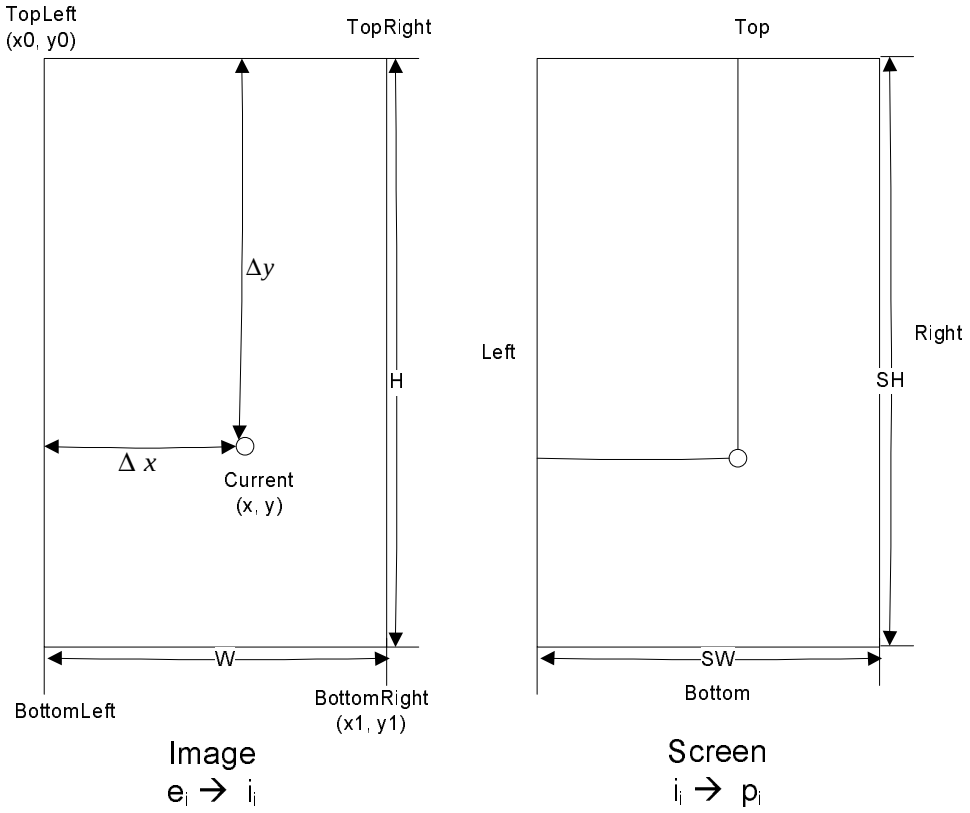



Figure 2.10: The conversion from GPS coordinates to screen coordinates

where $SW$ and $SH$ are the width and length of the screen (in unit of pixels).

After the screen points of geographic points have been calculated, the same procedure outlined in Figure 2.11 can be used to display all the graphic objects, which include points, lines, polygon, and golf course map.

## 2.4 Managing Games

The functional requirements of our mobile application are supported by the four components, which are shown in Figure 2.12. 1) Personal information management allows a user to enter/update his/her personal information, such as name, title, phone number, and email. 2) GPS configuration management allows a user to enter/update the parameters of GPS receiver. 3) Historic game management shows a saved historic game and/or replays the game. 4) Peer management shows the comparison between a golfer and other peers in the game.

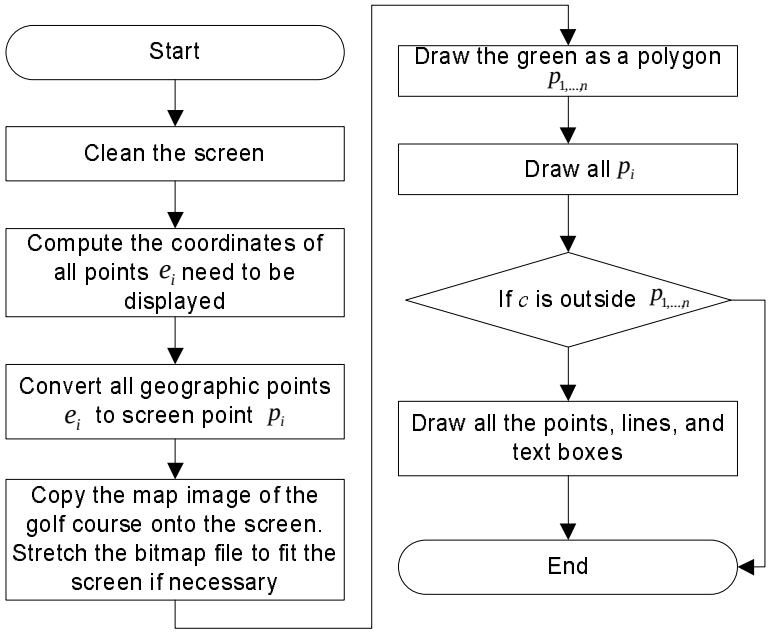



Figure 2.11: The flow-chart that draws graphic information

As described in Section 2.3.1 above, the database manager bridges most functional components and the EDB. In EarthChildren, SQLite [111] a light-weight, server-less embedded database is used to facilitate frequent data retrieval and update. SQLite is . More specifically, Figure 2.13 shows the SQLite database schema of EarthChildren, which generated by SchemaCrawler [114]. The database schema contains eight tables. Table *personal_info* keeps all the personal information. Table *gps_configuration* keeps all the parameters that are used by the GPS receiver and the serial port. Table *golf_course_gen* and Table *golf_course_spec* save the general information about the golfing venuses and all their courses, respectively. Tables *games*, *historic_scores*, *club_distance*, and *club_type* contain the information of all the saved games, the golfer's historic scores in each game, the distance of each hit, and the club type information.

The original SQLite was designed to be easy to use and to provide good performance. Confidentiality has not been considered to be a primary design goal. The confidentiality issue relateed to EarthChildren is that the EDB could be copied in a unauthorized manner

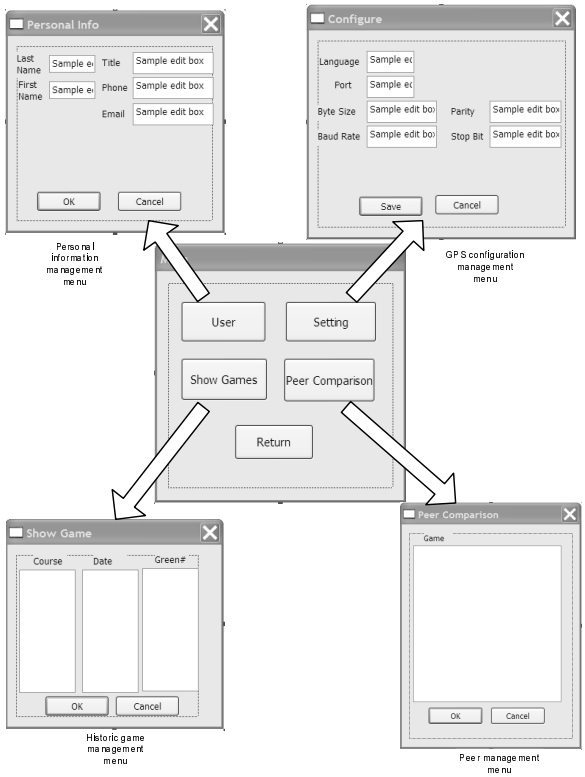



Figure 2.12: The main interface of EarthChildren

from one mobile device to another. To ensure the confidentiality, we use the SQLite Encryption Extension (SEE) [102] that leverages Advanced Encryption Standard (AES-128) encryption [10]. AES-128 is a symmetric-key encryption algorithm, which uses one *secret key* to encrypt and decrypt the data. For encryption, the AES-128 cipher takes a number of repetitive transforming rounds to convert the *plain-text* into the *cipher-text*. Each round consists of several processing steps, including `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`, using a 128-bits secrete key. For decryption, a set of reverse steps is applied to transform the cipher-text back into the original plain-text using the same secrete key. Based on [15], most known attacks that try to defeat AES are *computationally infeasible*. By using bicliques attack [15], the secrete key can be recovered by taking a computational complexity of $2^{126.1}$ for AES-128, $2^{189.7}$ for AES-192, and $2^{254.4}$ for AES-256. The current implementation of SEE uses the AES-128 for encrypting and decrypting the

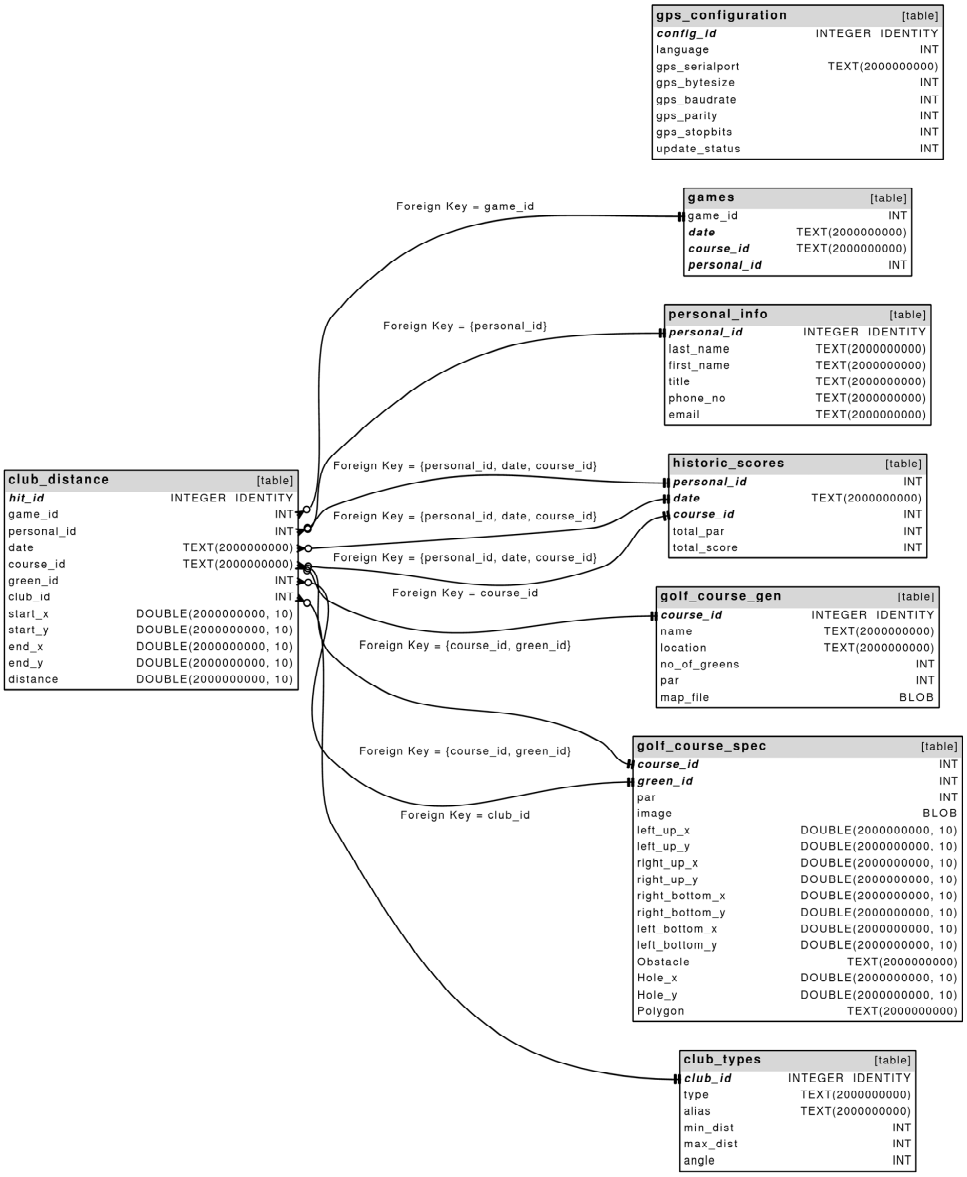



Figure 2.13: The database schema used by EarthChildren

EDB.

To uniquely assign a secrete key to a mobile device, we use the International Mobile Equipment Identity (IMEI) of that device, which is a sequence of numbers that uniquely identify that device, as the secrete key. Before the application is released, the EDB must be encrypted by its own IMEI. The process to access a encrypted EDB is as follows: Before opening the EDB, the method *GetIMEI()* is used to retrieve the IMEI of the mobile device.

Then, the IMEI is used as the secrete key to decrypt the EDB. Before all the database related operations have been finished, the application the same IMEI to encrypt the EDB.

It is worth noting that although the implementation ensures that the encrypted EDB is bundled with its IMEI, and cannot be used by simply copied to another device. However, this mechanism cannot prevent advanced attacks, which spoof the IMEI, avoid authentication (e.g., race condition attacks [91]) or exfiltrate the data through covert channels, the last of which are discussed in detail in Chapter 4 and Chapter 5.

## 2.5 Bluetooth Communication

One important feature of EarthChildren is the functionality of exchanging information among golfers. The information to be exchanged includes personal information (E-business Card) and game information (game history). Since the group of golfers is assumed to exchange information within short distance on a golf course (normally less than 100 yards), we choose Bluetooth technology [14, 51] to facilitate this functionality because of its data transmitting rate, throughput, transmission range, and suitability for such circumstances: First, most current mobile devices on the market support Bluetooth 2 Enhanced Data Rate (EDR) [110] or more advanced technologies, e.g., Version 3.0+HS and Version 4.0 [14] that provide a data transmitting rate of at least 3 Mbps and a throughput of 2.1 Mbps [14], which are sufficient for the application under discussion. Second, most current bluetooth antennas are using or can be extended to Bluetooth Power Class 1 antennas that have a communication range of 100 meters [89]. This range is far enough for our application in a golf course.

The Bluetooth communication component was implemented using Winsock API [131] provided by Windows Mobile [77]. Figure 2.5 shows the procedure of bluetooth communication, which first creates a thread that performs bluetooth device inquiry. If the device inquiry finds any bluetooth devices within the power range, the user can select the appropriate bluetooth device and establishes a connection with that device. Then, the local

device creates two threads: the *send* thread and the *receive* thread, which are responsible for sending information and receiving information, respectively.

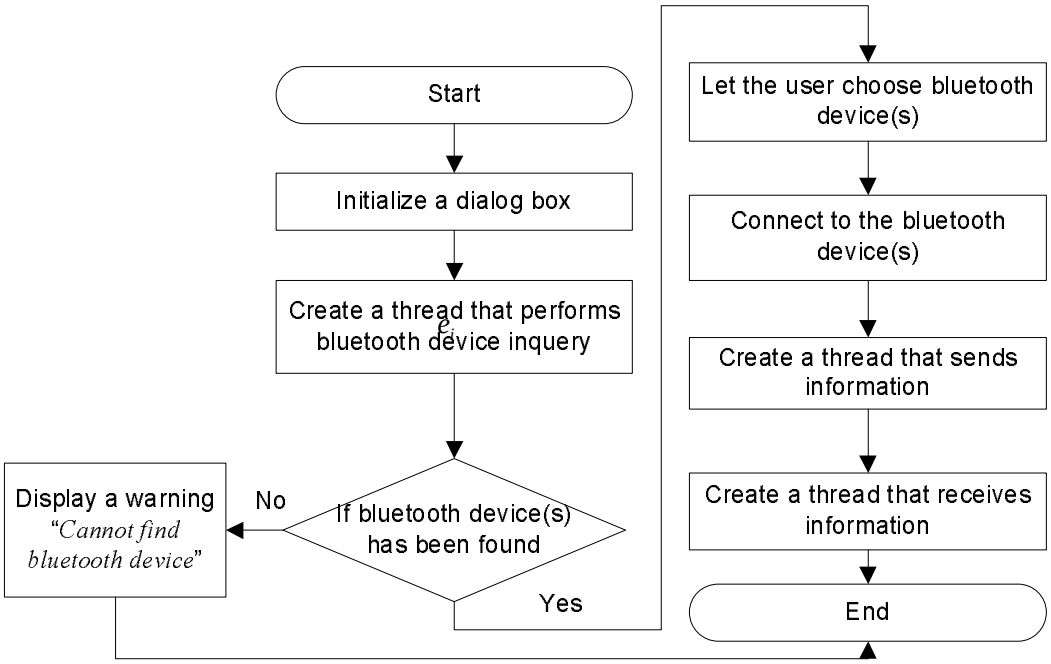



Figure 2.14: The procedure of peer-communication via bluetooth

## 2.6 Conclusion and Future Steps

In this chapter, we have presented the design and implementation of EarthChildren, a mobile application that provides many general functionalities. Our result demonstrates that our development methodology satisfies most mobile application.

# Chapter 3: Preventing SQL Injection Attacks

## 3.1 Introduction

SQL injection attacks (SQLIAs) refer to a class of attacks in which an adversary inserts specially crafted control code into the data fields of an SQL query. A successful SQLIA allows the attacker to gain control of the original query, leading to privilege escalation and extraction of unauthorized information from the database [116]. These attacks exploit inadequacies in the user input handing that are sometimes deeply embedded in the program logic [49, 132].

Earlier research has presented many techniques to defend against SQLIAs. Some research is geared towards attempting to *validate user inputs* [19, 100, 101]. Unfortunately, this strategy appears to be difficult to implement because most existing approaches have little knowledge of the syntactic structure of generated queries, hence some malicious inputs still manage to pass through [113]. Furthermore, input validation cannot offer protection against more sophisticated attacks, such as alternate encoding and stored procedure attacks [49].

Another class of *static analysis* solutions statically screens application source code to validate every user input before being integrated into a query [11,47,53,68,72,74,100,128,132]. These techniques work when application source code is available. An alternative approach uses dynamic prevention techniques [88, 113] that require minimal human interaction, but they insert extra *metadata* to delimit user inputs that may change the semantics of the original application code. Moreover, automatic preservation of *metadata* is almost impossible. Even if these approaches can effectively detect most SQLIAs, they require extra effort to distinguish user input data through the use of techniques, such as tainting or code instrumentation.

Some research [24, 75] and commercial solutions, such as using `PREPARE` statements require the programmer to define the skeleton of an SQL query in order to make the SQL structure unchangeable. These approaches, although providing a robust mechanism to prevent SQL injection attacks, require the programmer to specify the intended query at every query point, which often entails a significant amount of re-engineering.

Like most code injection attacks, SQLIAs exploit the fact that web applications use a common memory space to keep query code and the user input data, thereby injecting code as data and executing them as code [95]. Our system, SQLProb (SQL Proxy-based Blocker) extracts user input from the application-generated query, even when the user input data has been *embedded* into the query, and validates them in the context of the generated query's syntactic structure. We validate user inputs by extracting user inputs and aligning them against valid inputs by using and enhancing a genetic algorithm.

SQLProb offers several advantages: First, it is a complete black-box approach that does not require modifying application or database code, thereby avoiding the complexity of tainting, learning, or code instrumentation. Second, our input validation technique does not require metadata or learning. Third, our implementation utilizes an off-the-shelf proxy that requires minimal setup complexity. Finally, SQLProb is independent of the programming language used in the web application.

To evaluate our system, we have employed SQLProb to detect a wide range of SQL injection attacks. We show that SQLProb can prevent sophisticated attacks, such as the alternate encoding attack and the stored procedure attack. Our experimental results demonstrate remarkable effectiveness in detecting all classes of SQL attacks at a reasonable overhead.

The rest of the chapter is organized as follows. Section 3.2 illustrates the SQLIA with a simple web application example. In Section 3.3, we first define some terminologies, and then present our system overview, as well as the detailed steps of detection process. In Section 3.4, we evaluate the effectiveness of our approach. In Section 3.5, we discuss the limitation of our approach. Section 3.6 discusses related work and Section 3.7 concludes this chapter.

## 3.2 An Illustrative Example

In this section, we present an actual example of an SQLIA. Figure 3.1 depicts the login page of an online bookstore that allows users to login by providing user name and password. An SQL injection attack occurs when an attacker causes the web application to generate SQL queries that are functionally different from what the user interface programmer intended. For instance, for a database that stores user names and passwords, an attacker may attempt to gain root privileges by manipulating the user name or password string. Let's say the application contains the following code:

```
query = "SELECT * FROM accounts WHERE login='"
        + request.getParameter("login")
        + "' AND password='"
        + request.getParameter("password") + "'";
```

In this code, the web application retrieves user inputs from *login* and *password*, and concatenates these two user inputs into the query. The above code generates a query for the purpose of user authentication. However, if an attacker enters `admin` into the *login* field and `nosense' OR '1=1` into the *password* field, the query string becomes the following:

```
SELECT * FROM accounts WHERE name='admin'
AND password='nosense' OR '1=1'
```

In this example, the password field, which should have only a password string, is replaced with five sub-strings: string "`nosense`", logic control keyword "`OR`", "`1`", logical control assignment "`=`", and "`1`". Specifically, the logical control code "`OR`" connects "`password='nosense'`" and "`1=1`" to change the evaluation of the *WHERE* clause. Such a condition always evaluated to be a logic tautology; hence, an attacker can bypass the authentication process, and gain the root privilege.

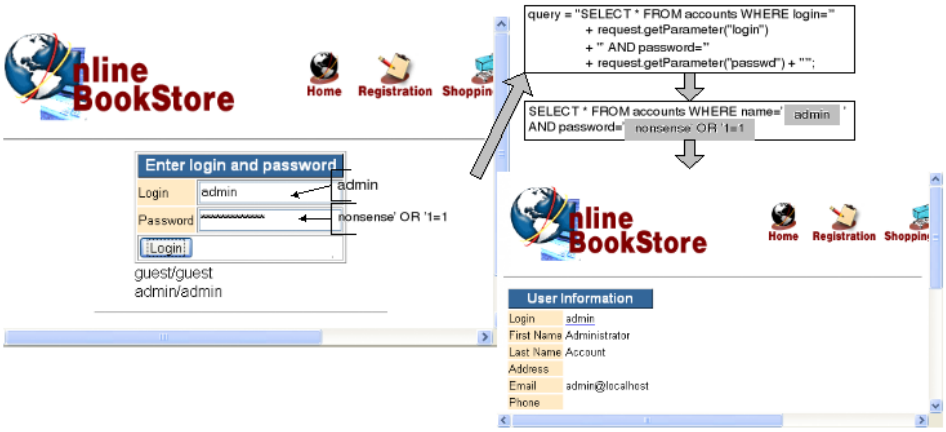



Figure 3.1: An example of SQL injection attacks

## 3.3 The SQL Proxy-based Blocker (SQLProb) System Design

### 3.3.1 SQLProb System Overview

The main system architecture of SQLProb is illustrated in Figure 3.2. SQLProb has four main components: (1) The *Query Collector* processes all possible SQL queries during the query collection phase; (2) The *User Input Extractor* implements a global pair-wise alignment algorithm to identify user input data (Section 3.3.3); (3) The *Parse Tree Generator* generates the parse tree for the incoming queries (Section 3.3.4); (4) The *User Input Validator* evaluates the user input is benign or malicious based on user input validation algorithms (Section 3.3.5). The shaded area shows the off-the-shelf proxy.

SQLProb uses two phases to detect SQLIAs: the query collection phase and the query evaluation phase. During the query collection phase, the query collector collects the queries that cover all the functionalities of the application, and stores them as the collected queries. During the query evaluation phase, when an application-generated query is captured by the proxy, the proxy forwards it to the user input extractor and the parse tree generator simultaneously. The user input extractor leverages a global alignment algorithm for the application-generated query against the collected query repository, and extracts the user

input data. Then, the user input validator validates the extracted user inputs in the parse tree, which is generated by the parse tree generator. If the user inputs are found to be benign, the generated query will be sent to database directly; otherwise, the query will be discarded as a malicious query. Here we assume that input for the query collection phase is vetted to avoid including existing SQLIAs in our training data. Such vetting can easily be done using an automated process.

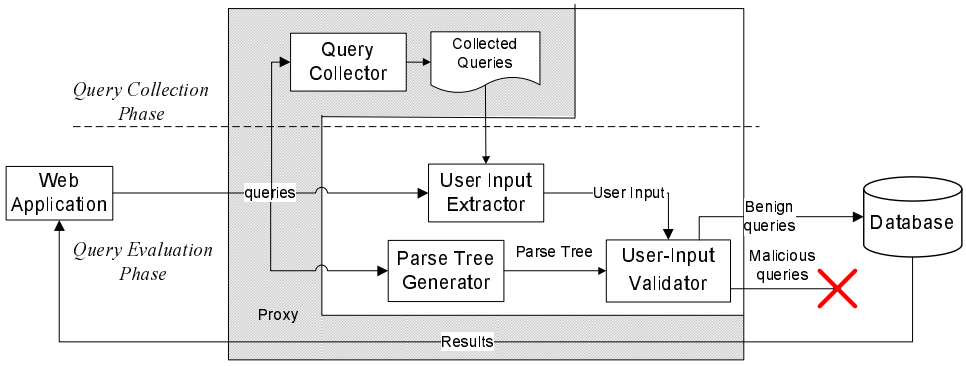



Figure 3.2: Overview of the SQLProb system architecture

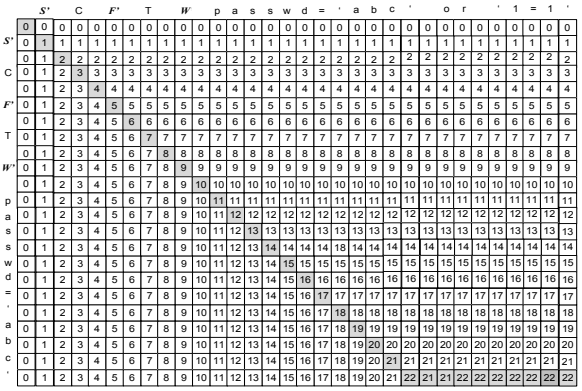
### 3.3.2 Problem Formalization

We formalize the problem of SQL injection attacks as follows:

- We denote the set of all queries generated by an application $\mathcal{A}$ as $\mathcal{Q} = \{q_i \mid 1 \leq i \leq m\}$. For a query $q_i$ that has $n(n \geq 0)$ user inputs, the set of user inputs is denoted as set $UI(q_i) = \{UI_{i,j} \mid 0 \leq j \leq n\}$. We use the term *user input data* for the *raw* user typed strings and any transformations thereof.

- The *collected queries* of $\mathcal{A}$ is the set $\mathcal{T}(\mathcal{A})$ of all SQL queries generated by $\mathcal{A}$ during the *query collection phase*.

- During the *data evaluation phase*, given a query $q_i$, the algorithm that compares the similarity for $q_i$ against a query $q_j \in \mathcal{T}(\mathcal{A})$ is given as $Sim(q_i, q_j)$. The query which $\mathring{q}_i$ gives the highest similarity value in $\mathcal{T}(\mathcal{A})$, is called *prototype query* of $q_i$.




(a) the scoring matrix

(b) The alignment result generated by Needleman-Wunsch algorithm between two SQL queries

Figure 3.3: A completest matrix processed by the Needleman-Wunsch algorithm

- After incorporating user input, the resulting query string may contains $k(k > 1)$ queries, based on SQL grammar $\mathcal{G}$. The parse tree for a single SQL query $q_i$ is denoted as $Tree(q_i)$. The parse tree of a set of queries $Q = \{q_i \mid 1 < i \leq k\}$ is denoted by $Tree(Q)$.

### 3.3.3 Separation of User Input

The assumption embedded in SQLProb is we can pre-generate the structure of all user inputs. Therefore, by having a large enough sample set, it is possible to efficiently compare any user input we receive with one that is in our sample. To efficiently perform this comparison, we use an enhanced version of the Needleman-Wunsch algorithm [28]. This algorithm was originally designed to *globally optimally aligned pairs* of DNA, RNA, or protein sequences, which attempt to align every residue in every sequence. It is most useful when the sequences in the query set are similar and of roughly equal size.

The Needleman-Wunsch algorithm [28] iteratively constructs a $(N+1) \times (M+1)$ dimensional *scoring matrix* where $N$ and $M$ are the lengths of the two sequences. The algorithm uses three steps: (1) *scoring similarity*, (2) *summing*, and (3) *back-tracking*.

Given two SQL query strings $q_1 = x_1 x_2 \ldots x_n$ and $q_2 = y_1 y_2 \ldots y_m$, we can insert gaps,

if necessary, to achieve a global maximum alignment between them. In the first two steps, our algorithm first computes the similarity score for each cell of the scoring matrix based on a predefined similarity matrix. The similarity and gap penalty can be defined as a part of the scoring matrix, or can be specified explicitly. In our work, we assign 1 for a syntactic match, 0 for a syntactic mismatch, and 5 for a gap. In the second (summing) step, the value of the maximum alignment between $q_1$ and $q_2$, or $Sim(q_1, q_2)$, is defined as the sum of terms for each aligned pair of letters $\langle x_i, y_j \rangle$ within the sequences (representing similarity as $s(x_i, y_j)$), plus terms for each gap (representing a penalty as $p$).

The cell $\mathcal{M}(i, j)$ is the score of the best alignment between the initial segment $x_1 x_2 ... x_n$ of x up to $x_i$ and the initial segment $y_1 y_2 ... y_m$ of y up to $y_j$. Initially, $\mathcal{M}(0, 0) = 0$, $\mathcal{M}(i, 0) = -ip$, $\mathcal{M}(0, j) = -jp$. Starting with the top leftmost cell $\mathcal{M}(1, 1)$ to bottom rightmost cell $\mathcal{M}(N + 1, M + 1)$, the matrix is iteratively filled by using the following equation:

$$y = MAX \begin{cases} \mathcal{M}(i-1, j-1) + s(x_i, y_j) & i \geq 1, j \geq 1 \\ \mathcal{M}(i-1, j) - p & i \geq 1 \\ \mathcal{M}(i, j-1) - p & j \geq 1 \end{cases} \tag{3.1}$$

In the third step (back-tracking), the algorithm *backtracks* from the cell with the highest score (the bottom rightmost cell) of the matrix back to the cell with the lowest score (the top leftmost cell) of the matrix, using the following three rules, in order to maximize the alignment score.

- If move *diagonal*, do nothing;
- If move *left*, insert a gap into the second sequence;
- If move *up*, insert a gap into the first sequence.

The purpose of backtracking is to move to the left, upper, or diagonal cell with the highest score. If all three cells are equal, backtracking will move to the diagonal cell. The

procedure is illustrated in Figure 3.3(a), where backtracking starts from the cell with the highest score 22 (the bottom rightmost cell). The shaded path is traced back to the top leftmost cell, which has the lowest score of 0.

We use the example in Figure 3.3(b) to demonstrate alignment between two SQL queries. The first query is in the collected query set $\mathcal{T}(\mathcal{A})$ obtained during the *query collection* phase. The second query is generated by the web application in the *data evaluation* phase. In this example, all SQL keywords are abbreviated by their initials. For example, `SELECT` is abbreviated as `S'`. `FROM` is abbreviated as `F'`. Table name(s) is abbreviated as `T'`. Column name(s) is abbreviated as `C'`. To achieve global alignment, the Needleman-Wunsch algorithm inserts "_", as a gap, into the first query after `password='`. Therefore, the alignment result for the password is "__________", which indicates that the user's input string for the second query has a length of 12. Accordingly, the sub-string at the corresponding position in the second query is `abc' OR '1=1`. Therefore, we can easily extract the user's input for the password in the second query as "abc' OR '1=1".

### 3.3.4 Complexity Analysis and Optimization

To allocate the *prototype query* $\mathring{q}_i$ of a application-generated query $q_i$, every application-generated query must be aligned with each collected query during the *query collection* phase. In the worst case, the incoming query $q_i$ must align with all $m$ collected queries in $\mathcal{T}(\mathcal{A})$. Because both time and space complexity of Needleman-Wunsch algorithm are *quadratic*, the overall time complexity to allocate the prototype query is $O(n^3)$, while the space complexity is still $O(n^2)$. Normally, $m$ is much larger than the length of most query strings, making most alignment operations superfluous. The purpose of our optimization is to reduce $m$ to $m'$ ($m' << m$), such that the overall time complexity will be reduced to approximate $O(n^2)$. To avoid unnecessary alignment operations, we cluster collected queries in the following steps: First, we cluster the collected queries based on different query types. For example, `SELECT` statements and `UPDATE` statements will be categorized into two different clusters. Secondly, within each cluster, queries carrying redundant information will

Table 3.1: The result of optimization

| Applications | Number of queries before optimization | Number of queries after optimization | % of reduction |
|---|---|---|---|
| Bookstore | 213 | 55 | 25.8% |
| Classifield | 395 | 31 | 8.1% |
| EmployDir | 218 | 33 | 15.1% |
| Events | 364 | 25 | 6.8% |
| Portal | 504 | 56 | 11.1% |

be further aggregated. Particularly, identical queries containing the same query structure, but with different user input, will be aggregated. In the aggregated representation, instead of replacing *name* and *password* by the wild-card tokens [9], we fully eliminate the user input strings. Eventually, queries with the identical query structure but different user input strings will be aggregated into the same cluster.

Table 3.1 demonstrates the number of queries, obtained during the query collection phase, before and after optimization. For different applications we obtained from `www.gotocode.com`, the optimization reduces the size of the collected query, ranging from 6% to 25%.

### 3.3.5 User Input Validation Algorithms

One of the advantages of our approach is that we can evaluate the extracted user input data in the context of the syntactic structure of the query. In order to illustrate our definition of the user input validation algorithm, we return to the two queries (benign and malicious) in Section 3.2, and show the parse trees for their *WHERE* clause in Figure 3.4 and Figure 3.5, respectively.

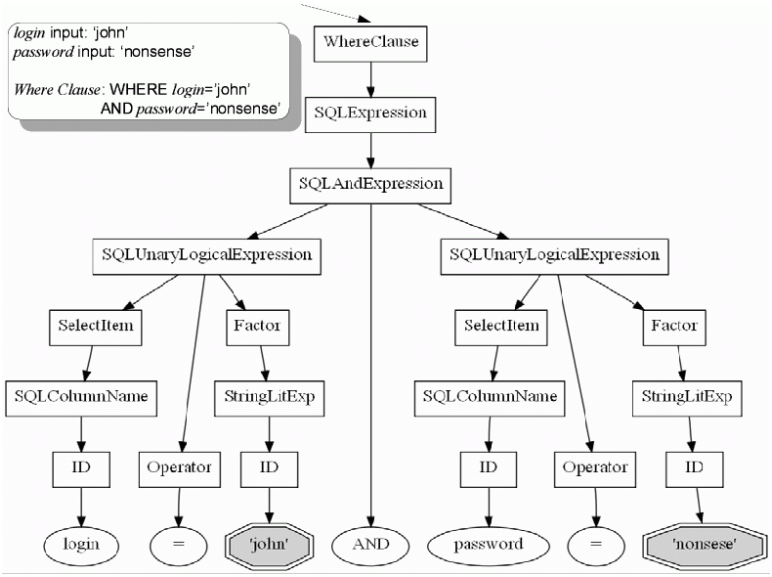



Figure 3.4: Parse tree for `WHERE` clause of a benign query

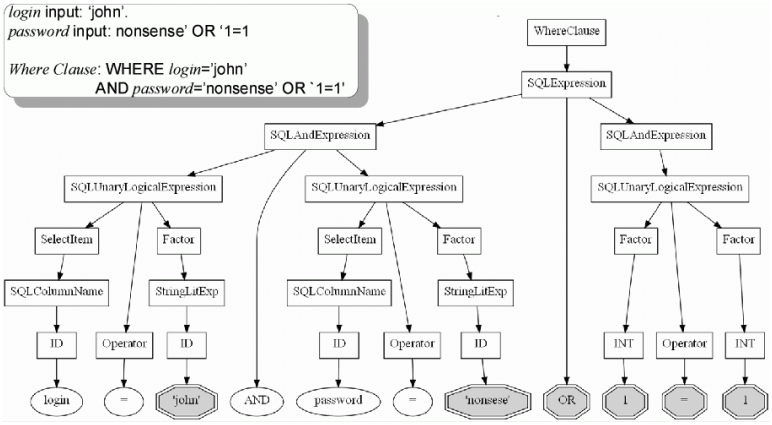



Figure 3.5: Parse tree for `WHERE` clause of a malicious query

Figure 3.4 represents the partial parse tree for the *WHERE* clause of the benign query in section 3.2, where the shaded double octagons represent the leaf nodes parse from user inputs. In the parse tree, every user input string can find a non-leaf node in the parse tree, such that its sub-tree leaf nodes comprise the entire user input string. That is, for user input leaf nodes, it is impossible to find a non-leaf node whose decedent leaf nodes contains not only the user input leaf nodes, but also other control leaf nodes. For example, consider the non-leaf node *ID* for *'john'* and *ID* for *'nonsense'*. Both are shown as shaded double

Table 3.2: The input validation algorithm

**Data**:Parse Tree $Tree(q_i)$ generated from SQL grammar $\mathcal{G}$, and the set of user input $UI(q_i)$
**Result**:*True* if the query is an SQLIA, or *False* if otherwise
**1. for** each leave node $leaf(u_j)$, which corresponding to $UI_{i,j}$ in $UI(q_i)$
  **1.1 do** *depth-first-search* upward from $leaf(u_j)$;
  **1.2** Search stops at a non-leaf node $nl\_node_j$;
  **1.3** Keep the *nl_node*;
  **end**
**2. do** *breath-first-search* downward from $NLN$ **until** reaching all $m$ leaf nodes $leaf(node)_k$;
**3. if** $\bigcup_{i=1}^{n}(leaf(u_i)) \subset \bigcup_{k=1}^{m}(leaf(node)_k)$ **then**
    Return *True*;
  **else** Return *False*;
**end**

octagon. Consequently, our algorithm is given in Table 3.2.

Figure 3.5 is an example of the application of the validation algorithm. The password field is parsed into the set $\bigcup_{i=1}^{n}(leaf(u_i))$ with five leaf nodes: `nonsense`, `OR`, `1`, `=`, and `1`. Next, we do depth-first-search from these five leaf nodes. The traversed paths intersect at a non-leaf node, *SQLExpression*. Finally, we do breath-first-search from *SQLExpression* to reach all the leaf nodes of the parse tree, which are composed as a superset of $\bigcup_{i=1}^{n}(leaf(u_i))$, implying that the input string $u_i$ is malicious.

The algorithm described above takes quadratic time, because step 2 and step 3 take time of $n \times h$, where $n$ is the number of leaf nodes parsed by $u_i$, and $h$ is the average number of layers from leaf nodes to *nl_node* in the parse tree. In addition, step 4 takes time complexity for a *breath-first-search* is $O(n^2)$. Therefore, the overall time complexity is $O(n^2)$.

## 3.4 Evaluation

To measure the overhead and performance of our approach, we implemented the *SQLProb*. The current implementation of *SQLProb* is implemented by Java and tested on a Virtual Machine with 1 GB RAM running Fedora 9 [1]. We use *MySQL 5.0.27* [78] as the back-end database server. For every SQL query initiated by a web application, we use a customized

Table 3.3: Different categories of attacks used in effectiveness evaluation

| **Type of Attacks** | **Attack Description** | **Detected?** |
|---|---|---|
| Tautology | Injecting one or more conditional statement | Yes |
| Logically Incorrect Queries | Information gathering, extract data | Yes |
| Union Queries | Return data from a different table | Yes |
| Piggy-Backed Queries | New queries piggy-back on the original | Yes |
| Stored Procedures | Invoking stored procedure | Yes |
| Inferences | Infer answers from apps' response | Yes |
| Alternate Encoding | Injecting modified control text | Yes |

*MySql Proxy* [36] to collect it, and determine if it is benign or malicious, before sending it to the database. If the query is determined to be benign, the query will be forwarded to the database; otherwise, it will be dropped immediately. To minimize the network latency, we use *wget 1.10* [129] to replay HTTP request from a different machine within the same Ethernet subnet to the machine, which runs both web application server and the prevention engine.

*JavaCC* [57] was used to automate the parse tree generation process. Specifically, we used *JJTree* [59], the pre-processor portion of JavaCC, to generate the parse trees. Figure 3.6 illustrates the screen-shot of the web application and the input of *login* and *password*. The corresponding parse tree is illustrated on the right hand side of the figure. The source codes of the vulnerable web applications are publicly available from *http://www.gotocode.com*.

### 3.4.1 Experimental Setup

We use Amnesia attack test suite [47], containing both benign and attacking string patterns. The attacking result has been extensively explored before (such as in [11,75]). Although the test suite contains 30 different attack patterns and the malicious codes have been injected

Table 3.4: Applications from the `www.gotocode.com`

| **Application Name** | **No of Requests** | **LOC** | **Failure & Syntax. Error** |
|---|---|---|---|
| Portal | 7483 | 16,453 | 2999 |
| Bookstore | 6492 | 16,959 | 1612 |
| Classifieds | 6544 | 10,949 | 1475 |
| EmplDir | 7038 | 5,658 | 1994 |
| Events | 7109 | 7,242 | 2240 |

successfully, we noticed that the set of attack patterns may not be complete. To ensure that the test suite is as complete as possible, we extended the attack pattern by including a wide category of the real-world attacking patterns[49], in order to guarantee that the malicious attacking string patterns return "*sensitive*" information. Table 3.3 illustrates a list of vulnerabilities, as well as injection attacks exploiting those vulnerabilities. Those attacks cover the most known SQLIA attacks. Furthermore, a combination of these attacks can generate more complicated new attacks. Specifically, Table 3.4 summarizes the Amnesia test suite. The second column lists the number of web requests, and the third column lists the lines of code (LOC) for each application. The fourth column reports the number of invalidate web requests detected.

### 3.4.2 Detection and Overhead

The objective of the first set of experiments was to demonstrate the effectiveness of the proposed technique to prevent SQL injection attacks. SQLProb achieved 100% detection rate with no false positive for all the attacks in the test suite.

The objective of the second set of experiments was to evaluate the performance of SQLProb. The first performance metric is the response time per web request. For each web request sent to the application, we measure the web application's original response time, the response time only with proxy, and the response time with SQLProb. All the results are shown in Figure 3.7 and have 95% confidence interval. Clearly, for different applications, the proxy only introduces modest delays, ranging from 16.7% (`Portal`) to 25.7% (`Events`).

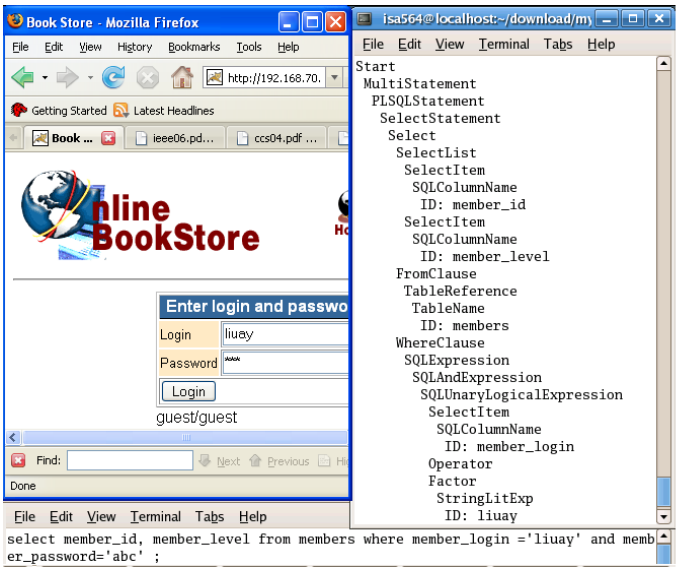



Figure 3.6: The screen-shot of SQLProb

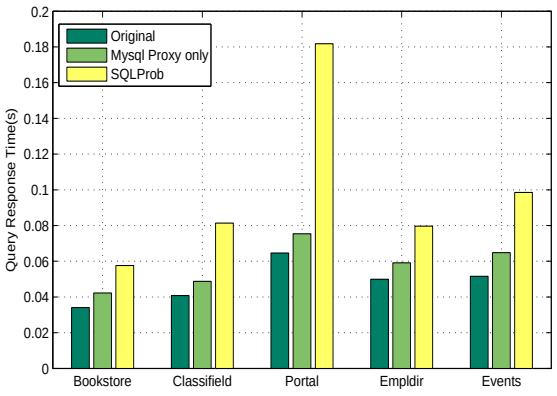



Figure 3.7: The response time

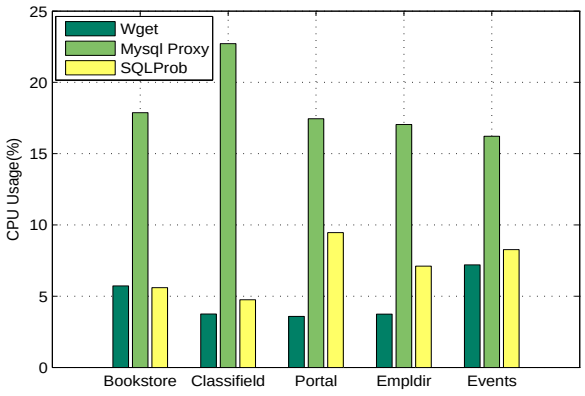



Figure 3.8: The CPU usage

For every request, the prevention engine had varying delays ranging from 59.5% (`Empldir`) to 181.3% (`Portal`). The delay of our prevention engine was mainly attributed to query alignment, parse tree generation, and user input validation procedure, specifically, most queries collected by `Portal` at training phase are very long strings. Due to the nature of the Needleman-Wunsch algorithm, the incoming query takes more time to align with the collected long queries in order to determine its prototype query. An area of future research is to develop methods for reducing the alignment time.

The second performance metric is the resource usage, such as CPU usage. Figure 3.8 displays the CPU usage over time for different web applications. The results clearly demonstrate that SQLProb utilizes much fewer computational resources than *Mysql Proxy*; i.e., 20.9% vs. 54.2%.

## 3.5 Discussion

Section 3.4.2 investigated the detection capabilities and overhead in extracting user inputs and using the input validation algorithm to analyze parse trees in order to detect SQLIAs. This section discusses the possible limitations and intended improvements of the work.

***Incomplete collection of web applications***

The accuracy of the alignment algorithm to extract user inputs is based on the fact that our alignment algorithm can guarantee to allocate the *prototype query* for an incoming query. Although we tested our methodology with a wide range of attacks, one potential limitation of our approach is that the accuracy of the extracted user inputs largely depends on the *completeness* of the collected query during *query collection* phase, a common problem with most black-box approaches.

***Possible errors*** Given an incoming query, without the knowledge of where the intended query was been invoked in the source code of web applications, it is possible for the alignment algorithm to use an inappropriate *prototype query* to do the alignment and extraction of the user input data. This situation can arise, for example, if there are two query structures in the application that was collected during the query collection phase, such as:

```
(1) "SELECT * FROM employdb WHERE name='" + nam1 + "'"
(2) "SELECT * FROM employdb WHERE name='" + nam1 + "'" +
    "OR name='" + nam2 + "'"
```

In this case, during the query evaluation phase, if we observe an incoming query as

*SELECT * FROM employdb WHERE name='john' OR name='admin'*, the alignment algorithm will choose query (2) as the *prototype query*, because it yields the maximum alignment value. However, it is possible for the input query being invoked by query(1) to be interpreted a SQL injection attack that is using `john' OR name='admin` as the injected code.

## 3.6 Related Work

SQL injection attacks have been researched in depth, resulting in a number of protection techniques that can be broadly categorized as: *input validation*, *static analysis*, *learning-based prevention*, and *dynamic prevention* approaches. This section compares SQLProb with each of these categories.

***Input Validation*** Because the root cause of SQLIAs is the intermingling of data and control code, improper input validation accounts for most security problems in database and web applications. Many input validation approaches are signature-based, resulting in incomplete input validation routines, introducing false alerts. In [100, 101], a human-developed security-policy description language(SPDL) specifies and enforces user input constraints by analyzing and transforming HTTP requests/responses to enforce the specified policy. This approach is human-based and requires that the developer know which data and pattern filter to apply to the data. PowerForms [19] and Commercial tools, such as AppShield [54] and InterDo [53] operate with a similar methodology.

The common weakness of these techniques is that they have no insight on the structure of the generated queries, and therefore, may still admit bad inputs. In addition, they ignore the fact that the original user input may be subject to manipulation and transformation, which may eventually defeat the effectiveness of this approach. The SQLProb approach is complimentary to most of the existing input validation approaches. In addition, it avoids the complicated steps of tracing the user input string manipulation throughout the application. Furthermore, compared to some products' claims to have the *proxy-alike* capability of

intercepting all incoming queries and blocking the suspicious ones [30, 45, 99], the SQLProb approach does not require specifying any signature or rule for known attacks.

Some products claim to have the *proxy-alike* capability of intercepting all incoming queries and blocking the suspicious ones, such as SANA Security's Primary Response [99], McAfee Entercept [30], and GreenSQL [45], though their efficiency and resistance to zero-day attacks may need further study. While those approaches match incoming queries against the certain behavior rules and known attack signatures [30, 45], the SQLProb approach does not require specifying any signature or rule for attacks.

***Static Analysis*** To guarantee security, [42, 68, 72, 132] perform static analysis over the entire application's source code to ensure that every piece of input is subject to an input validation check before being incorporated to a query. However, these approaches require the entire source code of the application, while our approach is a black-box based approach that requires no source code for applications and databases.

JDBC Checker [42] statically checks the type correctness of the dynamically generated SQL queries. Although JDBC Checker can detect SQL injection vulnerabilities caused by improper type checking of the user inputs, this technique is not applicable to more general forms of SQL injection attacks, because most of these attacks consist of syntactically correct and type-correct queries.

Our work is closely related to the recent work of CANDID [11], which dynamically mines programmer-intended query structure on any user input. While it is an effective approach, CANDID requires extra instrumentation to transform the web application code, usually tied to a specific programming language. Moreover, there is no guarantee that the byte code transformation process is error-free and that it will not introduce any potential vulnerabilities. Furthermore, byte code transformation is expensive and may negatively impact the availability of the web applications.

***Learning-based Prevention*** A set of learning-based approaches have been proposed to learn all the intended query structure statically [47] or dynamically [9, 118]. The effectiveness of detection largely depends on the accuracy of the learning algorithms. Compared

to this kind of approaches, our approach has at least two advantages. First, our approach neither *requires* a learning algorithm, nor is it *limited to* the number of collected queries. Second, our approach validates the user input within the syntactic structure of generated query, which more efficiently reveals the syntactic meaning of the user input.

***Dynamic Prevention*** Dynamic tainting approaches [48, 81, 133] taint the input strings and track those taints within the information flow of a program. All of them require not only the source code of the entire application, but also the collaboration of external libraries.

Many recent works [20, 88, 113], explicitly mark the user input data by using *metadata*. Although these approaches all work efficiently, it is widely believed that the metadata introduce many disadvantages, such as changing the semantics of the original program and requiring metadata preservation functions throughout the application, which is almost impossible to implement in an automatic manner [11]. In contrast, our approach is a complete black-box approach that requires no source code of web applications. SQLrand [18] leverages secret keys, while SMask [60] uses keyword mask to randomize and de-randomizes every SQL keyword through a proxy filter before passing the query to the database. As a result, the injected commands will cause a syntactic failure after passing to the proxy filter. This approach has immediate drawbacks: SQLrand requires extra effort to *rewrite* all the "plain-text" queries in the web applications as "randomized" ones. In addition, the security of the above approaches depends on the secret key, which can possibly be compromised by brute force attacks. Furthermore, this technique "decrypts" SQL instructions throughout the applications, which imposes tremendous cost overheads. By comparison, our approach does not require any secret key. Instead, it has a different purpose and uses the proxies for the alignment purpose.

## 3.7 Conclusion

We have presented SQLProb, a novel online and adaptive detection system against SQLIAs. SQLProb employs dynamic user input extraction and analysis, taking into consideration the

context of query's syntactic structure. Unlike current protection techniques, the new approach is fully modular and does not require access to the source code of the web applications or the database. In addition, it is easily deployable to existing enterprise environments and can protect multiple front-end web applications without any modifications. To measure the performance and overhead of the technique, a prototype of SQLProb has been used. The experimental results indicate that the method can achieve a high detection rate with reasonable performance overhead making the system ideal for environments where software or architecture changes are not an economically viable option.

# Chapter 4: Detecting Covert Storage Channels In a Highly Virtualized Environment

## 4.1 Introduction

The widespread deployment of firewalls and other perimeter defenses to protect information in enterprise information systems in the past decade has raised the bar for the malicious outsider attacking a networked enterprise. However, the seemly secured system can be penetrated and defeated easily by insider attackers if the attackers steal data from inside and sent it out through a secret communication channel, of which *covert channels* are a particularly virulent kind [44]. According to a report of the US Computer Security Institute (CSI) [41], the percentage of the insider attacks among the overall number of attacks reported rose to 59% in 2007. Insider attacks have overtaken viruses and worms on the list of the most prevalent forms of attacks [97].

As a common communication channel used by insiders, covert channels, refer to a class of communication channels that can be exploited by a process of transferring information in a manner that violates a system's security policy [83]. A successful covert channel leaks inside information to the outside human attackers in such a way that the unauthorized behavior undetectable.

To defend against various covert channels, researchers have presented a number of detection and prevention approaches. The detection approaches described in [12,21,22,38,58,104] mainly focused on first building clean traffic models from historic data, and then detecting various covert channels by comparing their behavior with the predicted behavior of the clean traffic models. After detection, countermeasures manipulate traffic to prevent information from leaking through covert channels [35, 62, 63, 120].

These defense approaches that explore deviation from clean historic traffic patterns have been shown to be effective against most covert channels. However, a common assumption of most of these methods is that covert channels traffic will reveal itself in a statistical pattern that is different from that of clean historic traffic. Therefore, their effectiveness depends on having a sufficient amount of clean traffic to determine such a pattern.

Unfortunately, modeling from the clean historic traffic is problematic in a networked virtual environment, such as the cloud computing environment. First, traffic generated from virtual machines (VMs) are subjected to unpredictable factors: (1) VMs may be migrated arbitrarily across the virtual networks; (2) VMs may constantly reverted to a previously snapshot; and (3) VMs may be configured to run multi-booting systems. Due to these unexpected environment changes in a virtual environment, clean historic traffic is either unavailable or unable to be represented the correct statistical patterns of a clean state in real time. Therefore, these modeling mechanisms are not generally applicable in a dynamic virtual environment.

To dealing with all these challenges, we have proposed and implemented a real time system Observer (Outbound Service Validater), which detects covert storage channels in real-time visualized environment. Observer leverages a secure VM to a) mimic the behavior of a vulnerable VM, and b) process networking traffic in two basic steps: First, it duplicates and redirects all the inbound traffic from a vulnerable VM to a secure VM. Second, it differentiates the outbound traffic of two VMs, such that any difference between two VMs can be easily identified.

Compared to existing approaches to detect covert channels, Observer offers several advantages. First, it is a real-time approach that does not require modeling or historic traffic. Second, Observer can be dynamically facilitated in a cloud infrastructure once a vulnerable VM has been identified. Third, our implementation is *transparent* to outside attackers, such that it minimizes the risk of turning an intrusion detection system into a vulnerable target.

To validate the effectiveness, we evaluated Observer with a set of three experiments. Our experimental results demonstrate that, without modeling, our system can detect many

covert channels, while inducing an average of 0.5ms latency to the inter-packet delays. The average increase of CPU usage induced by Observer is about 35% in a virtual network of 100Mbps throughput.

## 4.2 Related Works

### 4.2.1 Differential Analysis

A number of differential analysis techniques have been developed for building intrusion detection systems. In this regard, our work is closely related to Netspy [121], which compares the outgoing packets from a clean system with outgoing packets from an infected system. While Netspy generates signatures characterizing the malicious networking behaviors of spywares, our system improves over detection in the following ways: First, Netspy was only designed to detect spywares that leak victims' private information, and send it out through an HTTP response as *plain-text*. In contrast, our approach detects more stealthy attacks, such as covert channels, through which the inside information might be leaked through encrypted traffic. Second, in order to generate signatures, Netspy assumes that spywares must generate extra network traffic from the infected system. This assumption fails if the information is transmitted through a passive covert channel where no extra traffic is generated when leaking the victim's information. Third, Netspy only correlates the inbound packets with the corresponding outbound packets being triggered. Sophisticated covert channels generators, however, can evade detection by postponing the outbound packets, such that the inbound and outbound packets can never be correlated. Our system addresses the above problem directly.

Privacy Oracle [61] proposed a similar approach to discovering information leakage. By using the perturbed user input, the system can identify the fields of network traces by aligning various network traces pair-wise. Siren [16] used crafted user input, along with the legitimate user activities, to thwart the mimicry attacks. Although both Privacy Oracle and Siren use an architecture that is similar to ours, how they differentiate the anomalous

output to detect covert channels is unknown.

The recent work of Mesnier *et al.* [76] predicts the performance and resource usage of one device by the use of an mirror device. Their primary difference between their approach and ours is that it the former focuses on predicting the workload characteristics of I/O devices, whereas our technique focuses on detecting covert channels. Technically, these two problems are quite different and employ different techniques.

### 4.2.2 Attacks Design and Detection

A number of design methodologies have been developed for either building [22, 39, 65, 106] or detecting covert channels [4, 38, 58, 70, 86]. The effectiveness of these detections largely depends on the accuracy of modeling and the availability of a substantial quantity of *clean* historic traffic. Compared to this category of approaches, our approach demonstrates at least two advantages: First, our approach is an *online* approach that neither requires modeling nor large quantities of clean historic traffic. Second, our system can dynamically be deployed or migrated across the networked virtual infrastructure, such that it is applicable in a highly virtualized environment.

Many recent works apply covert channel design schemes to trace back suspicious traffic. For example, [124] took advantage of well-designed inter-packet delays, namely *watermark*, to trace suspicious VOIP traffic [122, 123]. Their recent work [92] utilized watermarked packets to trace back a bot-master. In contrast, our work only focuses on covert channels detection, rather than how to design them, although their covert channels design schemes of others can nicely complement our detection methodology.

Recent work addressed cross virtual machine covert/side channels [85, 96, 125] and their countermeasures [55]. All of them dealt with specific types of covert channels that leak information between VMs, that share the same VM monitor or hardware. In contrast, our work was not designed to detect inter-VM covert/side channels, but instead, the more aggressive covert channels that exist between inside and outside attackers.

Replayer [80] describes a system that replays the application dialog between two hosts,

and expects the recurrence of all the consequential behavior of a system after its infection. Although we face similar challenge as Replayer that modifies the networking packets to maintain a stateful dialog with off-line traffic, our approach manipulates the online traffic to keep a stateful network dialog with the secure VM. In comparison, our approach has a different purpose by manipulating the traffic for covert channels detection.

## 4.3 Threat Model

In this section, we address the threat model of covert channels in a networked virtual environment. We mainly focus on detecting unauthorized information leakage that transmits sensitive inside information to outside attackers through *covert channels*. Covert channels can be roughly categorized into two types: *covert storage channels* that manipulate the contents of a storage location (e.g., disk, memory, and packet headers) and *covert timing channels* that manipulate the timing or ordering of events (e.g., disk accesses, memory accesses and the inter-packet delays). The focus of this chapter is on the detection of covert storage channels.

Despite of the complex structure of a networked virtual environment, we wish to handle threats of covert channels as generally as possible. First, we assume that a vulnerable virtual machine (VVM) can be compromised by many exploits, such as the existing exploitations towards vulnerable services running on a VM, *zero-day attacks*, and inside subverting. Initially, we do not consider attacks that change the behavior of VMs through the virtual machine monitor (VMM) or hypervisor, a system software that allows multiple guest operating systems to run on a physical computer concurrently [117]. Second, we assume that, once a VM has been compromised, its user-space applications and the kernel-space device drivers can be fully controlled by the attacker. Since the incentive for attacker to launch covert channels is to leak insider information to outside attackers, covert channels can be detected from outbound network traffic. Figure 4.1 illustrates the proposed attacking scenario.

We established the foundation of security based on the following two assumptions. First, the VMM, which is under the control of the current virtual computing environment, is

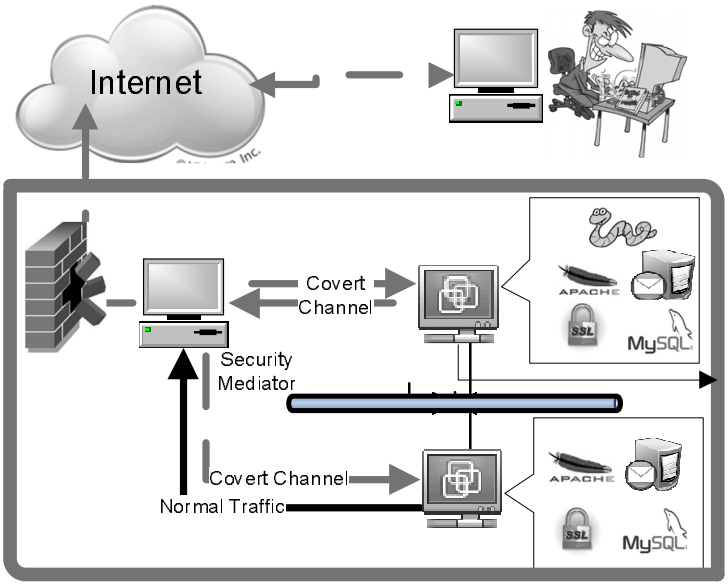



Figure 4.1: The threat of covert channels

assumed to be *correct*, *trusted*, and *unbreachable*. Second, it is assumed that there exist a certain number of secure VMs (SVMs), which are also under the control of the current virtual computing environment. The SVMs can be created either along with the vulnerable VM by simply cloning them from a clean state, or from a VM prototype, such as Amazon's Elastic Compute Cloud (EC2) service [29] or Microsoft's Azure Services [8]. The SVMs are protected by the virtual computing environment, such that outside attackers cannot compromise them. Note that, although our approach requires the synchronization of at least one SVM with each VVM, the number of SVMs is bounded by the number of vulnerable servers we are inspecting. Therefore, Observer can be applied to monitor as many vulnerable servers as necessary, once extra computational and storage resource become available. We do not require that the VMMs of the virtual environment knows the software being installed on VMs (e.g., the OS and the applications), although such information is useful for determining the integrity of VMs.

## 4.4 System Design

### 4.4.1 Definition

We address the problem of covert channel detection by isolating and validating the outbound traffic. From a formal perspective, we define our system as an extended finite state machine (EFSM) due to its advantages to represent input and output parameters, context variables, operations, and predicates, which is widely used to model communication system behaviors [87, 103].

**Definition 1.** Our CTCs detection system can be defined as an extended finite state machine (EFSM) $\mathcal{M}$, which is represented as a quintuple $\mathcal{M} = (\Sigma, S, \vec{v}, D, T)$, where:

- $\Sigma$ is an *event alphabet* of the EFSM, i.e., the networking packets in this paper. Each event $e \in \Sigma$ has its name and argument. In our system,

- $S$ is a finite set of states of EFSM, where $s_0 \in S$ is the *start state*, and $s_z \in S$ is the *final state*.

- $\vec{v}$ is a vector of finite set of *state variables*, $\vec{v} = (v_1, v_2, \ldots, v_k)$ $(1 \leq k \leq z)$.

- $D$ is a set of domain values for state variables, D $= (D_1, D_2, \ldots, D_k)$, in which $D_i$ denotes the domain of values for the variable $v_i$.

- T is a transition relation: $S \times D \times \Sigma \rightarrow (S, D)$.

The transition $t \in T$ is a tuple $\langle s_i, e, P_t, A_t, s_{i+1} \rangle$, where $s_i, s_{i+1} \in S$ are the starting and ending states of the transition $t$, respectively. Since our detection system keeps the states of at least two individual machines. $s_i$ is the composition of two states, $s_i = \mathring{s}_{i,m_1} \times \mathring{s}_{i,m_2}$, where $m_1$ and $m_2$ are two machines(or virtual machines). $P_t(e \times \vec{v})$ is a predicate on the valuation of a event $e$ and current state variables $\vec{v}$, within which $\vec{v} = \vec{v}_{i,m_1} \times \vec{v}_{i,m_2}$. $\vec{v}_{i,m_1}$ and $\vec{v}_{i,m_2}$ represent the vectors of the finite state variables on machine $m_1$ and $m_2$. The context

update function $A_t(\vec{v})$ is defined as $\vec{v} := A(\vec{v})$ that changes the current state variable vector $\vec{v}$. It changes the values of current state vector $\vec{v}$ before moving to a new state.

**Definition 2.** We further divide events in the above definition as input event $?event(\vec{x})_m$, and output event $!event(\vec{x})_m$ for machine $m$, in which $\vec{x} = (x_1, x2, \ldots x_n)$ is the vector of event parameters. The transition edges between the nodes of the EFSM directed graph can be represented as $\langle s_t, c?event(\vec{x}), P_t, A_t, q_t\rangle$ or $\langle s_t, c!event(\vec{x}), P_t, A_t, q_t\rangle$. Now suppose that $\mathring{s}_{0,m_1} = \mathring{s}_{0,m_2}$, and $e$ for all event $e \in \Sigma$, then we will have $\mathring{s}_{i,m_1} = \mathring{s}_{i,m_2} (0 \leq i \leq k)$. There exists a function that evaluates the difference between as $\Delta(\mathring{s}_{i,m_1}, c!event(\vec{x}), \mathring{s}_{i,m_1}, c!event(\vec{x}), \theta)$ $= \{Benign, Attack\}$, which returns either *Benign* or *Attack*, given a threshold $\theta$.

### 4.4.2 Observer System Overview

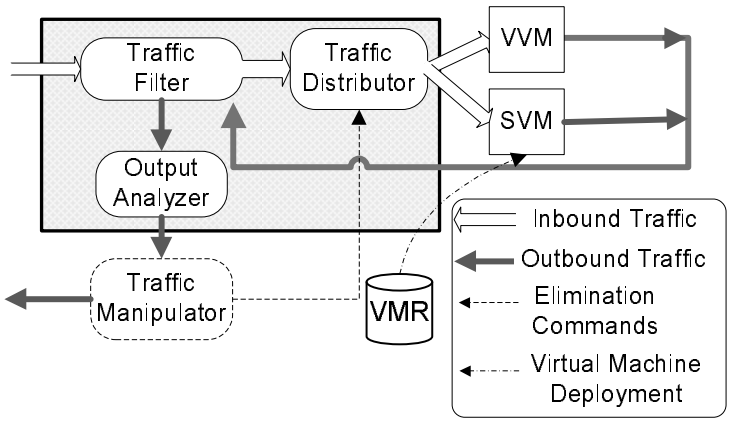



Figure 4.2: Observer system architecture

Figure 4.2 illustrates the system architecture of Observer. The shaded area highlights the components and the structure that are of special interest. Observer contains two major components: the *Security Mediator* (SM) and the *Virtual Machine Repository* (VMR). The SM consists of three sub-components: (1) The *Traffic Filter* monitors the inbound packets from the Internet, and filters the specific traffic of interest; (2) The *Traffic Distributor* identifies the networking protocol information from the intercepted packets, replicates the

packets, and sends them to both the VVM and SVM; (3) The *Output Analyzer* singles out the outbound traffic from both VMs, and detects anomalous patterns. VMR maintains a set of VMs, as well as activating, deactivating, cloning, and updating the snapshots of VMs. The shaded area highlights the components and the structure of *Observer*.

### 4.4.3 Intercepting Interesting Traffic

The function of the traffic filter is straight-forward. To protect certain services, Observer maintains a list of rules that are used to intercept relevant packets. If an incoming packet satisfies one rule, it will be subject to further processing. Whenever a new susceptible service has been launched, new rules that correspond to that service will be added to the existing list of rules. A few lines of rules are shown in Figure 4.4.3, which are similar to general purpose firewall rules. The rules specify the packet header fields, such as `Protocol`, `Src IP`, `Src Port`, `Dst IP`, `Dst Port`, and `Packet header flags`. For example, the first rule specifies that TCP packets sent from an external host (with an address of 123.123.123.123 and any port number) to an internal HTTP service (with an address of 129.174.2.123 and port number of 80) will be intercepted. The second rule drops all TCP packets that reply to any external host, whose packet header field contains an `rst` flag. The set of rules can be updated by inserting new rules or deleting obsolete rules at run-time, such that the attacker can not take advantage of penetrating the system when the traffic filter updates the list of rules.

It is worth to noting that the rules used by the traffic filter came from a priori knowledge or reported vulnerabilities. The traffic filter takes a time of $O(N)$ where $N$ is the total number of packets (measured in terms of number of bytes) that satisfies the rule-set provided that we only restrict our focus to those services most easily compromised. Since the packet filter does not buffer any processed packet, the storage requirements are bounded by the maximum size of a networking packet.

```
<?xml version="1.0" encoding="utf-8"?> <root>
    <rule>
        <Action>INTERCEPT</Action>
        <Direction>OUT</Direction>
        <Protocol>TCP</Protocol>
        <From_IP>123.123.123.123</From_IP>
        <From_Port>ANY</From_Port>
        <To_IP>129.174.2.123</To_IP>
        <To_Port>80</To_Port>
        <Flags>NA</Flags>
    </rule>
    ......
    <rule>
        <Action>DROP</Action>
        <Direction>IN</Direction>
        <Protocol>TCP</Protocol>
        <From_IP>ANY</From_IP>
        <From_Port>ANY</From_Port>
        <To_IP>129.174.2.123</To_IP>
        <To_Port>80</To_Port>
        <Flags>TCPFLAG=rst</Flags>
    </rule>
</root>
```

Figure 4.3: A snippet of configuration file used by the traffic filter

### 4.4.4 Re-directing Traffic to Virtual Machines

In general, the primary job of the traffic distributor is to forward packets, which are originally destined to the VVM, to the SVM. More specifically, the traffic distributor has jobs: First, when the traffic distributor receives a packet $e$ from the traffic filter, it constructs a new packet, namely $e'$. $e'$ keeps some packet fields from $e$, such as `Src IP`, `Src Port`; while it modifies others, such as the `Dst IP`, `Dst Port`, the sequence numbers, and checksum. The `Dst IP`, `Dst Port` are the IP address and port number of the SVM. Then, the traffic distributor dispatches $e$ and $e'$ simultaneously. Second, when the traffic distributor receives reply packets from both VMs, it only sends out the reply packets, which corresponding to the incoming packet $e$ from VVM. In such a way, outside attackers may have no knowledge of the SVM.

Figure 4.4 illustrates how does the traffic distributor works as a simplified state machine, where $M1$ and $M2$ represent the VVM and SVM, respectively. The states of the state machine are represented as ellipses, and the behavior of the traffic distributor are represented

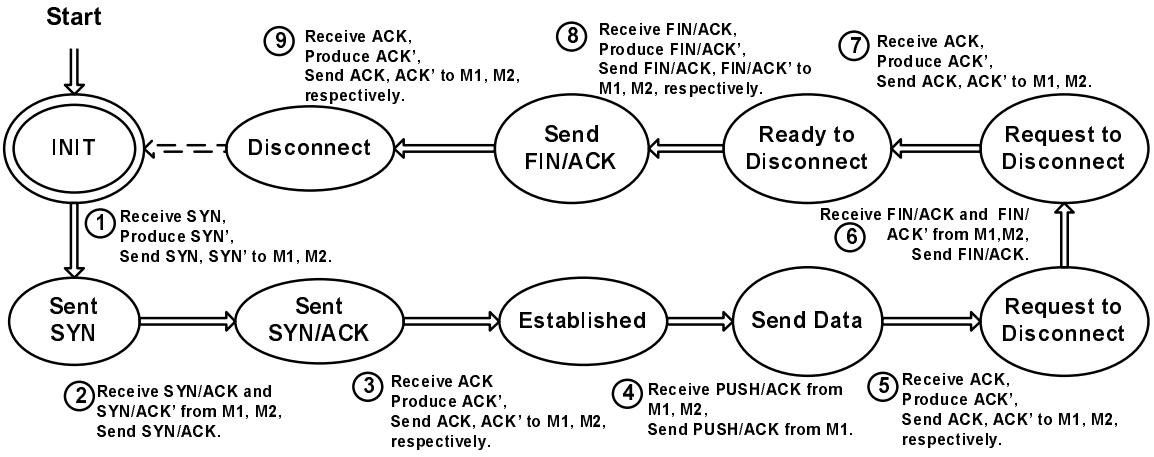



Figure 4.4: The simplified state machine of *traffic distributor*

as the edges (or transitions) between states. For example, when the traffic distributor receives a `SYN` packet, it constructs a new SYN packet, namely `SYN'`, and sends both `SYN` and `SYN'` to *M1* and *M2*, respectively. Similarly, when it receives an `ACK` and `ACK'` packets, it only sends out `ACK`. To synchronize the output of *M1* and *M2*, the traffic director keeps all previous states of communication in a *queue*, in case of packets loss or fragmentation.

The overhead required for the traffic distributor to construct a new packet $e'$ is nearly the same for each packet. The storage requirement is dictated by the total number of packets been forwarded from the traffic filter. Although Observer collects the live traffic, which can be increased without a bound, the storage requirement is still bounded by $2n$, where $n$ is the number of packets in the queue.

### 4.4.5 Detection of Anomalous Traffic Patterns

To detect covert channels, we use shape tests, which are referred to as first-order statistics, such as means, variances, and distributions [86]. To examine the shape of traffic patterns, we run two statistic tests: Chi-Square test ($\chi^2$ test) [25], and two-sample Kolmogorov-Smirnov test (KS-test) [50]. In particular, we use the $\chi^2$ test for discrete samples, and the KS-test for continuous samples. One prominent feature of the both test is that it is general enough that it *does not* depend on the actual cumulative distribution function being tested, which

is *distribution free*.

More formally, for the $\chi^2$ test, x1 and x2 are the samples collected from SVM and VVM that divided into $k$ bins and the test statistic can be defined as follows:

$$\chi^2 = \Sigma_{i=1}^{k} \frac{(K_1 R_i - K_2 S_i)^2}{R_i + S_i} \tag{4.1}$$

where the summation is over bins 1 to k, $R_i$ is the observed frequency for bin $i$ for sample 1, and $S_i$ is the observed frequency for bin $i$ for sample 2. $K_1$ and $K_2$ are scaling constants that are used to adjust for unequal sample sizes; specifically,

$$K_1 = \sqrt{\frac{\Sigma_{i=1}^{k} S_i}{\Sigma_{i=1}^{k} R_i}}, \tag{4.2}$$

and

$$K_2 = \sqrt{\frac{\Sigma_{i=1}^{k} R_i}{\Sigma_{i=1}^{k} S_i}} \tag{4.3}$$

For the default $\chi^2$ test, the *null hypothesis* that two samples come from the same distribution, against the alternative that two samples do not come from the same distribution. The result $h$ is 1 if the null hypothesis can be rejected at the 5% significance level. The result h is 0 if the null hypothesis cannot be rejected at the 5% significance level. In our test, x1 and x2 are the samples collected from SVM and VVM, respectively, so the null hypothesis is that x1 and x2 are both normal.

The two-sample Kolmogorov-Smirnov test (KS-test) has been used as an alternative to $\chi^2$ test in some previous approaches [39, 86]. The KS-test is also *distribution-free* but is restricted to continuous distributions. Specifically, KS-test measures the maximum distance between two empirical distribution functions ($S_1(x)$ and $S_2(x)$) of two samples (x1 and x2), whose measurement is determined as follows.

$$KSTEST = max|S_1(x) - S_2(x)| \tag{4.4}$$

Where $S_1(x)$ is the proportion of x1 values less than or equal to x and $S_2(x)$ is the proportion of x2 values less than or equal to x. For the default KS-test, the null hypothesis is that x1 and x2 are from the same continuous distribution. The alternative hypothesis is that they are from different continuous distributions. The result h is 1 if the test rejects the null hypothesis at the 5% significance level; or 0 if otherwise. In our test, x1 and x2 are the samples collected from SVM and VVM, respectively. Similarly, the null hypothesis is that x1 and x2 are both normal.

## 4.5 Implementation

To measure the effectiveness and performance overhead of *Observer*, we have implemented Observe on top of a `VMware ESX Server` [31]. The traffic filter and the traffic distributor were implemented as part of a *transparent bridge*, which uses a customized IP Firewall (ipfw) [37] to intercept networking packets initiated from the Internet. We used the `divert socket` [108] technology to manipulate intercept packets. The divert socket can alter raw packets before they being processed by the networking stack. The traffic filter and the traffic distributor were implemented by `C` language with approximately 1000 lines of code.

To minimize the networking latency after the packets leave the security mediator, we clone the VVM and the SVM cloned from the same virtual machine image of the same state, and configure them *one hop* away from Observer. Since the VVM and SVM must be closely synchronized, we use the *sequence number* field of TCP packets to synchronize the outbound traffic from both VMs.

The output analyzer uses `ethereal` [32] to collect the traffic and separate the timing information, and `ntop` [82] to measure the statistics of networking traffic at runtime. The output analyzer module was written in `C`, `perl`, `Dataplot`, and `MATLAB`.

## 4.6 Evaluation

In this section, we describe how Observer was evaluated. In Section 4.6.1, we describe the attacks used in our experiment. In Section 4.6.2, we analyze the effectiveness of Observer in terms of sensitivity and detection accuracy achieved. Finally, in Section 4.6.3, we discuss the performance overhead of the approach.

### 4.6.1 Attacks in the evaluation

In our evaluation, we limited our detection analysis to two types of CSCs: IP/TCP Packet Fields Channel (PFC) and Botnet Traceback Watermark Channel (BTWC). The PFC operates by modifying the `urgent` fields of TCP packets to transmit information, so that it transmits a `1` bit by increasing the value of `urgent` fields by an integer modulo $w$, and transmits a `0` bit by increasing the value of `urgent` fields by an integer modulo $\lfloor \frac{w}{2} \rfloor$.

The BTWC operates by modifying the length of the encoding packet by padding characters to achieve a specific length, in which the padded characters could be invisible (such as whitespace) or visible characters. We use the scheme similar to [92] to construct the BTWC. Specifically, to encode an $i$-bit sequence $S = s_0, ..., s_{i-1}$, we use $2i$ randomly chosen packets pairs $\langle P_{r_i}, P_{e_i} \rangle$ $(i = 0, ..., L)$, where $r_i \leq e_i$. We call $P_{r_i}$ the *reference packet* and $P_{e_i}$ the *encoding packet*. A covert bit $s_k (0 \leq k \leq i-1)$ can be encoded into the packet pair $\langle P_{r_i}, P_{e_i} \rangle$ with the following equation, where $L_r$ and $L_e$ are the values of the encoded filed in $P_{r_i}, P_{e_i}$, respectively.

$$e(L_r, L_e, L, s_k) = l_e + [(0.5 + s_k)L - (l_e - l_r)] mod 2L \tag{4.5}$$

To test the effectiveness to detect slow covert channels, we extended the original covert channel design scheme. Rather than using $2i$ packets, we use $2ai (a \geq 1)$ packets to encode $S$, where $a$ was could be a constant or a randomized number generated by a *pseudorandom number generator* (PRNG). $P_{r_i}$ and $P_{e_i}$ will be chosen from $2a$ packets. We call $a$ as the *amplifier*, where a larger value of $a$ indicates a slower covert channel. We apply the *amplifier*

Table 4.1: The detection window size for various covert channels

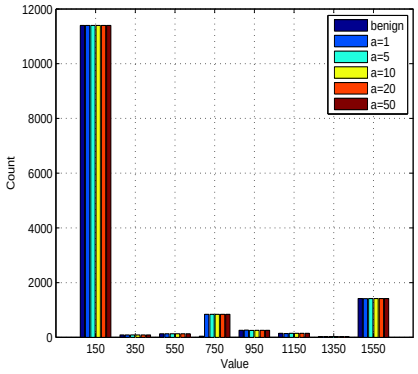
| | BTWC | PFC |
|---|---|---|
| $a = 1$ | 160 | 60 |
| $a = 5$ | 181 | 460 |
| $a = 10$ | 7150 | 390 |
| $a = 20$ | 24060 | 730 |

to all covert channels design schemes in this chapter.

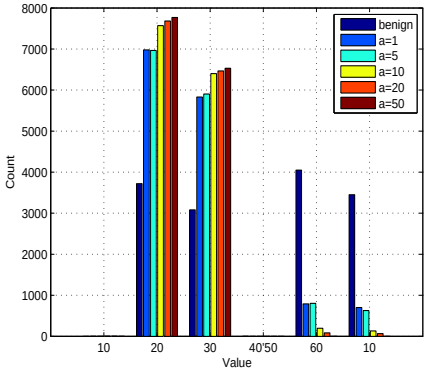
### 4.6.2 Effectiveness

#### Latency

We measure the latency of Observer in terms of the detection window size, which is defined the minimum number of packets needed to detect a covert channel once it starts to transmit information. Clearly, a larger windows size indicates a longer latency with less sensitivity and greater vulnerability to covert channels in real-time. Table 4.6.2 shows the detection window size of Observer to reach 100% true positive rate. As we stated before, a large value of $a$ indicates slow covert channels. The results obviously indicate slower covert channels require larger detecting windows. The true positive rate is the proportion of covert channels in test that are correctly been identified as covert. For example, to detect the most aggressive BTWC, which sends one bit per packet, the detection window size is 160. However, a slower BTWC, which transmits one bit every 20 packets, needs a windows size of 24060! Similarly, the detection window sizes vary from 60 to 730 for PFCs, indicating that Observer is much more sensitive to PFC than it is to BTWCs. This result can be explained by comparing the distributions of these two channels types in Figure 4.5, and the cumulative distributions of them, as illustrated and Figure 4.6. These data show that, for various value of $a$, the distributions of BTWCs more similar to each other than is the case with the distributions of PFCs. Therefore, BTWCs are more difficult to detect by Observer than are PFCs.

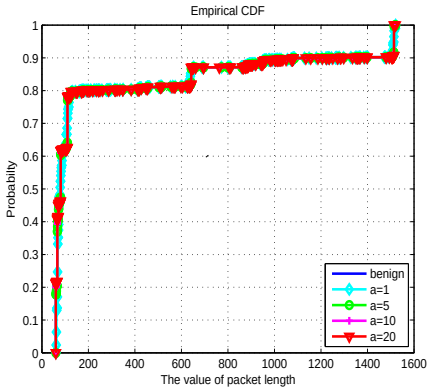



(a) The distribution of packet lengths for BTWCs.

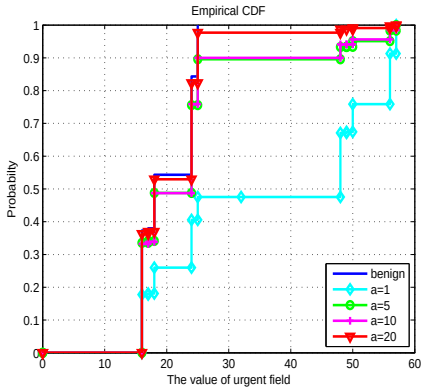



(b) The distribution of *urgent* fields values for PFCs.

Figure 4.5: The distribution of BTWCs and PFCs




(a) The cumulative distribution of packet lengths for different $a$




(b) The cumulative distribution of *urgent* field for different $a$

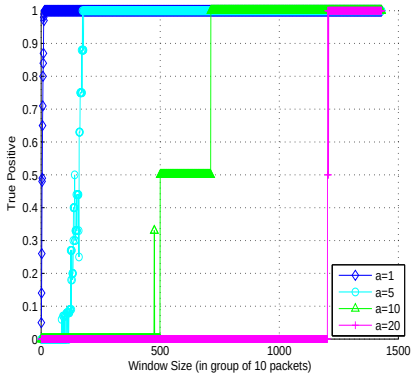
Figure 4.6: The cumulative distribution of BTWCs and PFCs

### Detection Rate

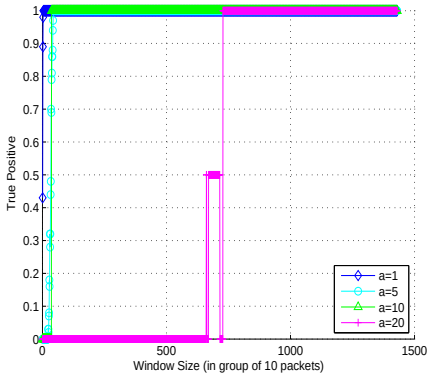
To determine the false positive rate, which is the proportion of legitimate samples in the test set that are *incorrectly* been identified as covert, Observer collects live traffic from both VVM and SVM. We run both BTWC and PFC for $a = 1$. Under these conditions, the false positive rate of Observer for both BTWC and PFC are 0. Table 4.6.2 and Table 4.6.2 show the results of the associated statistical tests. As expected, the slower covert channels, which have larger values of $a$, exhibit statistics that are closer to those of the legitimate traffic. For the analysis of false positives, we set the targeted false positive rate to be 1%.

To achieve this false positive rate, we use the 99th percentile of the legitimate samples as the *cutoff point* between samples that samples are considered benign or covert.

Figure 4.7 demonstrates that Observer reaches 100% true positive rate for both PFCs and BTWCs for various values of $a$. We also notice that the effectiveness of detection depends on having a sufficient number of collected packets. For example, for the PFC with $a = 10$, the truth positive rate fluctuates when Observer collects 5000 packets. The similar situation also happened for BTWCs. Clearly further research is needed to improve the effectiveness of detection.




(a) The true positive rate of PFCs for various amplifier $a$




(b) The true positive rate of BTWCs for various amplifier $a$

Figure 4.7: The true positive rates of two covert channel detection

Table 4.2: The test scores of PFCs

| | Legitimate Http | PFC ($a = 1$) | PFC ($a = 5$) | PFC ($a = 10$) | PFC ($a = 20$) | PFC ($a = 50$) |
|---|---|---|---|---|---|---|
| *mean* | 20.143 | 36.926 | 23.48 | 23.343 | 20.878 | 20.478 |
| *stdev* | 3.925 | 16.485 | 10.576 | 10.316 | 6.137 | 5.088 |
| $\chi^2$ test statistics | 0 | 6789.002 | 82.136 | 75.657 | 3.902 | 0.796 |
| $\chi^2$ *test*(CDF value) | 0 | 1 | 1 | 1 | 0.580575 | 0.061007 |
| $\chi^2$ *test*(1% cutoff point) | $\geq 15.087$ | $\geq 11.345$ | $\geq 15.086$ | $\geq 13.277$ | $\geq 13.277$ | $\geq 13.277$ |

Table 4.3: The test scores of BTWCs

| | Legitimate Http | BTWC ($a = 1$) | BTWC ($a = 5$) | BTWC ($a = 10$) | BTWC ($a = 20$) | BTWC ($a = 50$) |
|---|---|---|---|---|---|---|
| *mean* | 283.601 | 284.785 | 283.824 | 283.721 | 283.660 | 283.622 |
| *stdev* | 453.532 | 452.005 | 453.215 | 453.357 | 453.441 | 453.503 |
| KS-test statistics | 0 | 0.248 | 0.049 | 0.025 | 0.012 | 0.004 |
| KS-test (p value) | 1 | 0 | 2.937e-015 | 2.53e-004 | 0.212 | 0.951 |
| KS-test (1% cutoff point) | $\geq 0.1923$ | $\geq 0.1923$ | $\geq 0.1923$ | $\geq 0.19233$ | $\geq 0.1929$ | $\geq 0.1929$ |

Table 4.4: The comparison of throughput under the best and the worst case scenarios

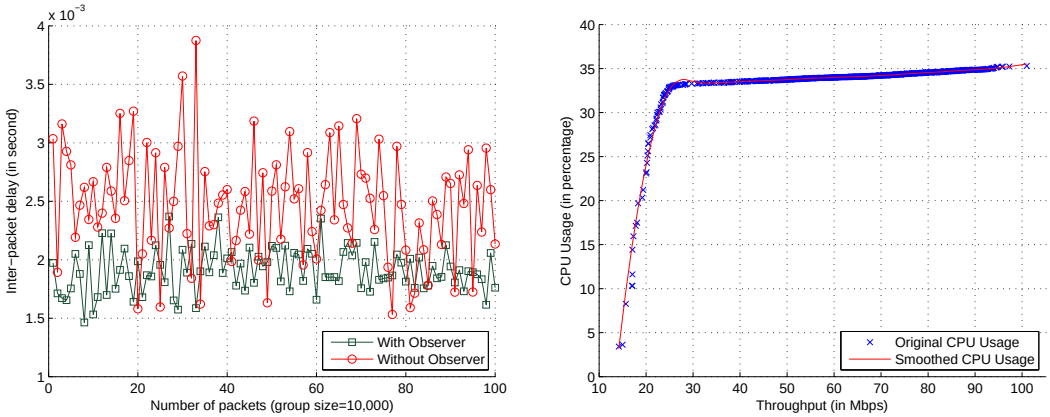
| | Best scenario | Worst scenario | Ratio |
|---|---|---|---|
| # of packets | 1,802,176,469 | 170,712,403 | 0.094 |
| # of bytes | 1,214,518,973,379 | 128,003,547,066 | 0.105 |
| # of packets/sec | 187,727 | 17,783 | 0.098 |
| # of bytes/sec | 126,512,393 | 13,333,703 | 0.110 |
| Duration | 160 min | 160 min | 1 |

### 4.6.3 Performance

In the first experiment to evaluate the performance of Observer, we examine the throughput *boundary* of Observer, under the *worst* and the *best* case scenarios. In the worst case scenario, since most inbound packets sent from outside to VVM are part of a covert channel, the majority of the packets are subjected to being intercepted by the traffic filter. However, in the best case scenario, none of the packets need to be intercepted. In both scenarios, we measure the throughput of outbound HTTP packets from the VVM for 160 minutes, such as the number of packets, the number of bytes, the number of packet per second, the number of bytes per second. The detailed statistics of both scenarios are demonstrated in Table 4.6.3. Although the throughput of Observer under the worst case scenario decreases to nearly 10% of that is under the best case scenario, the average processing time of each packet is roughly $56.2ms$ under the worst case scenario. The possible reasons of this decreasing can attribute to the serial processing of Observer. Parallel packet processing technique is needed to improve the drawback of the current implantation.

In the second experiment, we measure the average inter-packet delay (IPD) latency

introduced by Observer. For that purpose, we collect 1,000,000 packets with Observer installed and without Observer installed. The average latency added to the IPD is 0.5ms, which is much smaller than the average IPD without Observer installed (2.267ms). The IPD latency can be explained as the queuing delays that Observer adds to the inbound packets, when redirecting them to both VMs at the same time.




(a) The delay introduced by the security mediator *(SM)*.

(b) The delay introduced by the security mediator *(SM)*.

Figure 4.8: Performance overhead of Observer.

In the third experiment, we measure the resource usage, such as the CPU usage, with the respect to the throughput. The result is encouraging in that the resource usage does not increase significantly over time. Indeed, the CPU usage remains fairly steady, so that there continues to be a linear relationship between usage and throughput when the throughput is less than 100Mbps. In short, this result shows that the average system overhead scales with the networking throughput as expected. It is also worth nothing that the overhead caused by Observer has no negative performance impact on the machines being monitored, which is a significant benefit of this virtual architecture.

## 4.7 Discussion

We have so far demonstrated the effectiveness and performance of Observer to detect covert channels. In the following, we discuss the limitations and possible improvements to our

system.

***Redundancy cost*** Currently, the ability of our system to accurately detect covert channels depends on the assumption that the virtual computing environment can provide at least one secure VM for each vulnerable VM, which means that the networked virtual environment has to keep a number of secure VMs, in order to detect the same number of vulnerable VMs. Although the cost to maintain a secure VM is much less than to maintain a physical machine, this fact limits the scalability of our approach. How to dynamically allocate secure VMs and manage them effectively will be a future research topic for us.

***Synchronization between VMs*** Although our current approach mainly focuses on the services that have limited usage of user interface, and simplify the input of services as the inbound networking traffic, there still exists a large number of services that need frequent interaction with human users through command line or graphic interface. To keep a secure VM across its life cycle is difficult and laborious. In addition, since our approach is a online intrusion detection system, there might be inbound traffic that contain exploits that could compromise secure VMs at runtime. We will examine this question in our future research.

## 4.8 Conclusion

We have presented Observer, a real-time intrusion detection system against covert channels. Observer detects new covert channels in a networked virtual environment by running a secure VM to mimic the vulnerable VM, such that the difference between two VMs can be identified. Unlike most existing covert channels detection approaches, our approach does not depend on modeling historic data. Our experimental results demonstrate that we can achieve low latency and high detection rate with reasonable overhead.

# Chapter 5: Detecting Covert Timing Channels in a Networked Virtual Environment

## 5.1 Introduction

Covert timing channels (CTCs) are one of the most prevalent forms of attacks in computer security [84, 97]. They are a class of attacks that exfiltrate inside credentials, such as passwords and secret keys, by manipulating the timing or the ordering of networking events [38]. A successful CTC leaks sensitive inside information to outside human attackers without triggering any alert from a well-protected network, presenting a challenging that is well-known to the world of information security.

Most existing approaches to address this security problem either use signature-based approaches to detect known CTCs [22] or anomaly-based approaches by modeling legitimate networking traffic to detect unknown CTCs [12, 21, 38, 104]. However, these methods have at least two obvious limitations: First, the effectiveness of these approaches depend on the availability of a sufficient amount of legitimate (or attack-free) traffic to construct the signatures of attacks or to build an accurate model of legitimate traffic, thereby making it possible to detect CTCs either as a match to attack signatures or a deviation from the models of legitimate traffic. Unfortunately, in a networked virtual environment, where virtual machines are interconnected, legitimate traffic is either hard to obtain or contains noise due to the imprecise timing keeping mechanism used in virtual machines [52]. Therefore, most existing approaches fail to detect CTCs because legitimate traffic for signature construction or modeling is absent. In addition, since the virtual machines use emulated devices to obtain timing information, that timing information contains more noises than that collected from a physical networking environment. Second, most of the approaches were designed to detect those CTCs which transmit information at a high rate of speed to achieve large bandwidth.

However, an attacker who is aware of this fact can deliberately extend the duration of the CTC transmission process. In such a way, the timing patterns of CTCs traffic become almost indistinguishable from that of legitimate traffic and the attacker can exfiltrate the insider information with complete peace of mind. Although some model-based CTC design schemes [39, 71] have been proposed to model slow CTCs, there is no detection approach has been implemented to detect those CTCs.

In this chapter, we present a new wavelet-based approach to detecting CTCs in a networked virtual environment, where no historic legitimate traffic is available. We build our CTC detection approach upon our novel intrusion detection system, Observer, which measures the distance between outbound traffic generated by two virtual machines (VMs): one is the potentially infected VM and the other is a benign VM. More specifically, the metric employs the variables derived from the discrete wavelet-based multi-resolution transformation (DWMT) to measure the variability of the timing difference at all decomposition scales. To the best of our knowledge, this approach is the first online CTC detection approach that can detect CTCs by quantitatively measuring the distance between two networking flows.

To evaluate our wavelet-based metric, we conduct a series of experiments to detect different CTCs and their slow variations. Then, we determine the robustness of our approach with the presence of noise. In short, the experimental results demonstrate that our wavelet-based metric is very effective in detecting existing CTCs and their slow variations in real-time.

## 5.2 Related Works

CTCs have been the subject of a great deal of recent research. For example, Berk et al. [12] implemented a simple binary covert timing channel based on the Arimoto-Blahut algorithm, which computes the input distribution that maximizes the channel capacity [13]. Cabuk et al. developed the first IP CTC, which we refer to as *IPCTC* [21] and *TRCTC* [22], which is a more advanced traffic replay CTC. Shah et al. [104] developed a keyboard device, called

*JitterBug*, that slowly leaks user typed information over the Internet.

Giffin et al. [40] showed that although not a CTC, low-order bits of the TCP timestamp can be exploited to create a CTC due to the shared statistical properties of timestamps and packet timing. Another application is to apply CTC to trace suspicious traffic. For example, Wang et al. [124] took advantage of well-designed inter-packet delays, namely *watermark*, to trace VOIP traffic [122, 123]. Their recent work [92] utilizes watermarked networking traffic to trace back bot-master through CTC traffic. Our work, however, does not consider how to design and trace back watermarks, although the design scheme of [92] can be used to test the effectiveness of our detection metric. Gianvecchio et al. [39] and Liu et al. [71] designed model-based covert channel encoding schemes which seek to achieve undetectability and robustness at the same time.

A number of CTCs detection methods have also been developed. Peng et al. [86] showed that the Kolmogorov-Smirnov test is an effective way to detect CTCs that manipulate inter-packet delays. Cabuk et al. [22] investigated a regularity-based approach of detecting CTCs. They also developed a metric, named $\epsilon$-similarity, to measure the proportion of similar inter-packet delays. The limitation of $\epsilon$-similarity metric is that it only targets IPCTC. Therefore, it is not general enough to detect other CTCs. Berk et al. [12] employed a simple mean-max ratio test to detect binary or multi-symbol CTCs. However, the assumption of the mean-max ratio test is that the legitimate inter-packet delays follow a normal distribution, which is often not true for real world traffic. Gianvecchio et al. [38] investigated an entropy-based approach to detecting CTCs, and they achieved good results of detection. Compared to their work, our approach demonstrates the following three advantages: First, as an essential step to computing the patterns of length $m$, the corrected conditional entropy (CCE) metric of [38] spends a quadratic time complexity, while our approach takes linear timing complexity. Second, CCE metric cannot detect stealthy CTC, while our WBD metric can detect all CTCs, as well as their stealthy versions. Third, compared to previous approaches, our approach works online and is applicable in a virtualized environment, while most previous approaches requires training data and cannot work in virtual machines.

Jing et al. developed a wavelet-based approach to measure time distortion of low latency anonymous networks [58], which is closely related to our work. Their timing distortion metric is particularly designed to deal with the issues of packet transformation that are caused by anonymous network, such as flow mixing, merging, adding chaff, and packet dropping. In comparison, our approach has at least two advantages: First, our WBD metric can clearly differentiate slow versions of different CTCs, while the time distortion metric cannot. Second, our approach is particularly designed to detect CTCs, which have more stealthy timing characteristics than the timing distortion of packet transformation introduced by anonymous networks. Therefore, our approach is a different technology that addresses more subtle issues.

Askarov *et al.* [7, 134] proposed online covert timing channels prevention mechanisms that mitigate information leakage. Although their approaches were proven to be efficient to bounds the information leakage as a function of elapsed time, how to detect covert timing channels is still an open problem which has not been addressed. The online prevention mechanisms without considering which networking flow contains covert timing channel will affect throughput and the response time of a system significantly. Another problem of these online prevention mechanisms is that it affects networking performance, particularly for those streaming services which require fast response time.

## 5.3 Background

### 5.3.1 The Challenges to Detecting CTCs

After a vulnerable VM has been compromised by an attacker, an effective way to exfiltrate inside information from a well-protected network is to leak information through covert timing channels (CTCs). There are two types of CTCs: active and passive. Active CTCs refer to the covert channels that generate additional traffic along with the existing traffic to transmit information, while passive CTCs refer to covert channels that manipulate the existing traffic and do not generate additional traffic. In general, active CTCs are easier

Table 5.1: The means and standard deviations of two flows (in second)

| Statistics | Legitimate flow | Jitterbug flow |
|---|---|---|
| Mean | 0.14722 | 0.15097 |
| Standard Deviation | 0.04064 | 0.04317 |

to detect than are passive CTCs [38]. In this chapter, we only focus on the CTCs that manipulate the timing information of networking packets, say *inter-packet delays* (IPDs) of a network flow.

The primary challenge to detecting CTCs is that the statistics of covert traffic are so close to those of legitimate traffic that it is hard to differentiate CTC traffic from legitimate traffic by using standard statistical tests. Figure 5.1 illustrates a comparison of the empirical cumulative distribution (ECD) of the inter-packet delays of a legitimate networking traffic sample and a CTC traffic sample, namely *JitterBug* [104] (sample size=300). The distribution and ECD of these two samples are very close. Other statistics of these two samples, such as means and standard deviations, are also very similar as shown in Table 5.1. Because of this great similarity, detection methods only based on standard statistical tests are not accurate and robust for detecting CTCs.

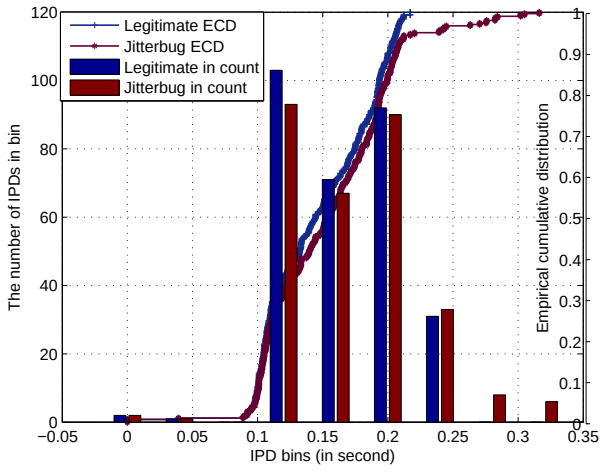



Figure 5.1: The comparison of ECD between a legitimate flow and a *JitterBug* flow

### 5.3.2 Time Drifting in Virtual Machines

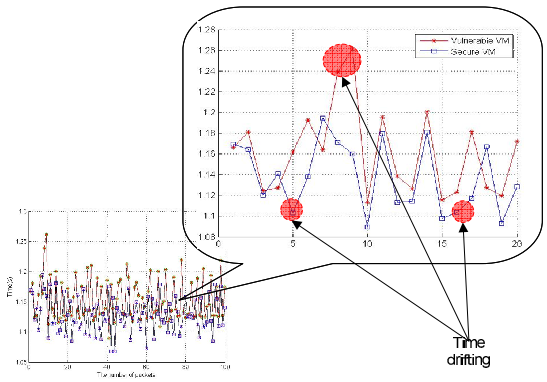
To detect CTCs in a physical networking environment, most existing approaches commonly assume that legitimate traffic contains accurate timing information that is always available. However, this assumption might not be hold in a networked virtual environment for at least two reasons.

First, in a virtual networked environment, many dynamic conditions make it almost impossible to obtain legitimate traffic at runtime [126]. For example, a VM may be arbitrarily migrated across a virtual network. A VM may be constantly reverted to a previous snapshot, which is a saved state of data and hardware configuration of a running virtual machine [107]. In addition, VMs may be configured to run multi-booting systems, accessible by different users, and/or for different purposes. Because of these dynamic conditions, legitimate traffic of a benign virtual machine can have completely different timing patterns across its life-cycle, which makes the collected legitimate traffic information inappropriate for modeling. In short, in such a case, most existing approaches fail due to insufficient *good* quality of legitimate traffic.

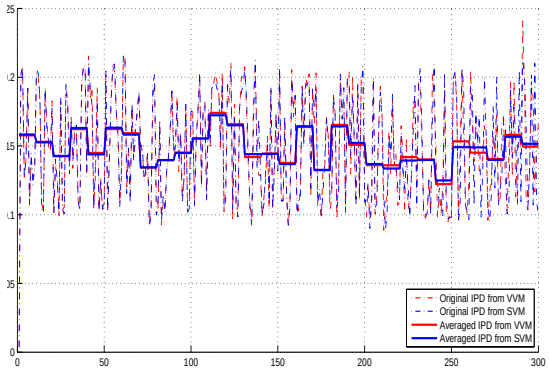
Second, unlike a physical machine that can directly access its physical CPU, a virtual machine accesses the physical CPU through emulated timer devices [52], which makes accurate time keeping almost impossible. We use the term "time drifting" to refer to the noises introduced by the emulated timer devices. Figure 5.2(a) illustrates an observation of time drifting in the inter-packet delays of outbound traffic of two identical VMs when given the same inbound traffic. However, even though time drifting exists in a networking virtual environment, the IPDs of two identical VMs are still very similar, as shown in Figure 5.2(b).

### 5.3.3 A Wavelet-Based Approach to Detecting Covert Timing Channels

In this section, we present a wavelet-based metric to quantitatively measure the distance between the inter-packet delays (IPDs) of a legitimate flow and a CTC flow. We first describe the model of our CTC detection approach. Then, we discuss how to measure the timing distance between two outbound networking flows by using the variables derived from




(a) The IPDs of two outbound networking flows from two identical VMs




(b) The correlation of IPDs between two networking flows after averaging (size=300, w=10)

Figure 5.2: The effect of *time drifting*

wavelet-based multi-resolution transformation (DWMT). Finally, we formulate the metric to measure the timing distance between two networking flows.

### The Model of the CTC Detection Approach

Given the same networking inbound flow $I$, the problem of measuring the timing distance between two outbound flows, namely $O_1$ and $O_2$, can be formulated as follows: The inbound flow $I$ contains $K > 0$ packets $\langle p_{i,1}, ..., p_{i,K-1} \rangle$. As the response of $I$, Virtual machine VM1 generates $O_1$ that contains $M > 0$ packets $\langle p_{o_1,1}, ..., p_{o_1,M-1} \rangle$ and virtual machine VM2 generates $O_2$ contains $N > 0$ packets $\langle p_{o_2,1}, ..., p_{o_2,N-1} \rangle$. Since the packets of $O_i$ were generated by VM as the response of that of $I$, we segment the packets in $I$ and $O_i$ based on their request/response relationship. Specifically, for the $j$th inbound *segment* $I^j = \langle p^j_{i,1}, ..., p^j_{i,m} \rangle$, its responding outbound *segment* in $O_i$ is defined as $O^j_i = \langle p^j_{o_i,1}, ..., p^j_{o_i,o} \rangle$. We use $t^j_{(o_i,k)}$ to represent the timestamp of the $k$th packet in the $j$th segment of $O_i$. Since the length of I is much longer than that of $O_i$, we can further aggregate $O_i$ into $w$ *aggregated segments*. In Figure 5.3, $O^{i-2}_2$ is the responding outbound segment of $I^{i-2}$ in flow $O_2$.

For $O_{1,i}$, the packets within the segment of $t_i$ is represented as $\langle p_{o_1,0}, ..., p_{o_1,m-1} \rangle (m > 0)$ . Similarly, the packets in the same segment is represented as $\langle p_{o_2,0}, ..., p_{o_2,n-1} (n > 0)$, where

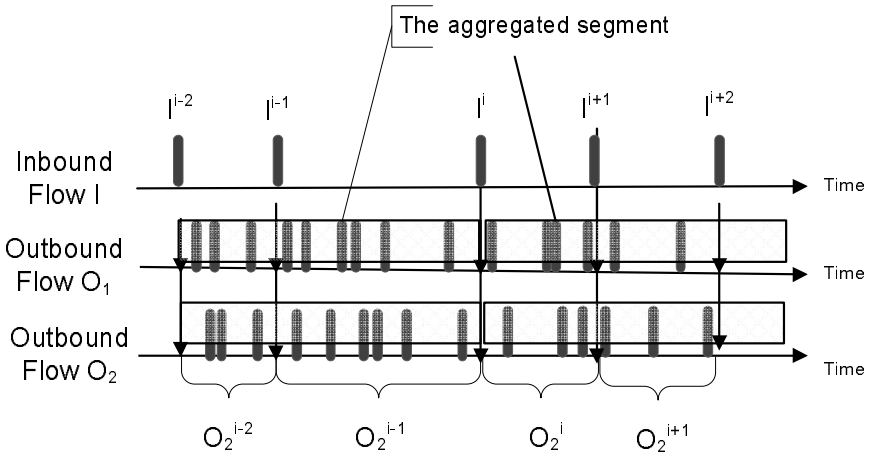



Figure 5.3: The inbound and outbound flows

$n \approx m$. The time-stamp of the $j$-th packet in $O_{1,i}$ and $O_{2,i}$ are represented as $t^j_{(O_1,i)}$ and $t^j_{(O_2,i)}$, respectively.

**Measuring Timing Distance between Flows**

The assumption behind the development of the timing distance metric is that the timing patterns of the outbound legitimate traffics are similar. Therefore, by characterizing the timing patterns of the networking flows, it is possible to efficiently measure the timing distance between a legitimate flow and a CTC flow. To effectively perform the measurement, we use discrete wavelet-based multi-resolution transformation (DWMT) [2]. The DWMT has been widely used in signal processing [3] and anomaly detection [58], which has at least three prominent features. First, DWMT provides multi-resolution analysis, which allows us to look at the sequence of data at different scales. Second, DWMT allows feature localization, i.e., it allows us to know the characteristics of the signal and "approximately" where in time they occur. Finally, DWMT supports online analysis, that is, we can compare the difference between two flows online.

The DWMT takes a sequence of data as input and transforms that sequence into a number of wavelet coefficients sequences. Specifically, the $l$ level DWMT takes a sequence

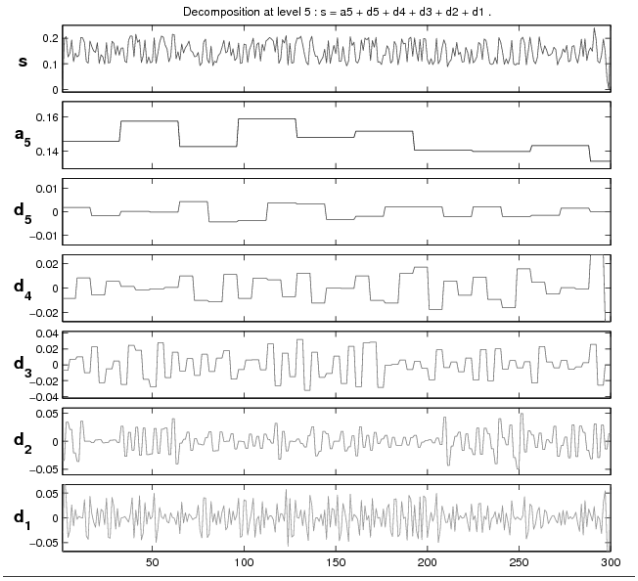



Figure 5.4: The original IPDs and its wavelet coefficients at different scales

of IPDs and transforms the sequence into 1) $l$ wavelet detailed coefficient vectors at different scales ($CD_i$, where $1 \leq i \leq l$) and 2) a low-resolution approximate vector ($CA_l$)[1]. For the $j$-th segment of $O_i$, the wavelet detailed coefficients vector at scale $l$ can presented as:

$$V(i,j,l) = \langle CD_l^j(o_i,1), ..., CD_l^j(o_i,N_j)\rangle \tag{5.1}$$

where $N_j = n_j \times 2^{-j}$ is the number of wavelet detail coefficients at scale $j$, and $c_{l,k} = c_{l-1,2k} + c_{l-1,2k+1}$ for $l \geq 0$. Figure 5.4 illustrates the transformation of a DWMT, namely Harr wavelet [46], on a IPDs sequence (sample size=300). The Harr wavelet uses five level transformation and obtains one approximate coefficient vector ($a_5$) and five detailed coefficient vectors at different scales ($d_i$ and $1 \leq i \leq 5$ ).

The design goals of our wavelet-based distance (WBD) are three-folds: First, we expect that the WBD between two legitimate flows is small. Second, we expect that the WBD between a legitimate flow and a CTC flow is detectably different. Third, the WBD should be able to differentiate regular CTC and stealthy CTC. To achieve these goals, we define

[1]In this dissertation, we use $CA_j$ and $CD_0$ exchangeable.

three derived vectors based on the coefficient vector V(i,j,l) at scale $l(l \geq 0)$: the *intra-flow vector* (intraFV), the *inter-flow vector* (interFV), and the Kullback-Leibler divergence (KLD) vector.

We define $intra(i,j,l) = \langle CD_l^j(O_i,1) - CD_l^j(O_i,0), ..., CD_l^j(O_i,N_j) - CD_l^j(O_i,N_{j-1})\rangle$, which reflects the fluctuating characteristics between adjacent coefficients within a coefficient vector at scale $j$ of one flow $O_i$. The $intraFV(j,l)$ is defined as the Euclidean distance [26] between $intra(1,j,l)$ and $intra(2,j,l)$ as shown in Equation 5.2:

$$intraFV(j,l) = dist(intra(1,j,l), intra(2,j,l)) \tag{5.2}$$

Similarly, we define the $interFV(j,l) = dist(V(1,j,l), V(2,j,l))$ as the Euclidean distance between coefficient vectors V(1,j,l) and V(2,j,l), which characterizes the deviation between two wavelet coefficients for the same segment $j$ at the same scale $j$.

The Kullback-Leibler divergence (KLD) has been used to measure the distance between two probability distributions $p_1(x)$ and $p_2(x)$ [66, 67]. From an information theory's perspective, KLD measures the expected number of extra bits required to code samples from $p_1(x)$ when using a code based on $p_2(x)$. For probability distributions $p_1(x)$ and $p_2(x)$, their KLD is defined as:

$$KLD(p_1(x), p_2(x)) = \sum_{i=0}^{|x|} p_1(x) \log \frac{p_1(x)}{p_2(x)} \tag{5.3}$$

To calculate the KLD between two wavelet coefficient vectors at scales $j$, it is necessary to obtain the probability distribution of V(i,j,l), namely $p(V(1,j,l))$. To obtain $p(V(1,j,l))$, we first covert $V(i,j,l) = \langle CD_l^j(o_i,1), ..., CD_l^j(o_i,N_j)\rangle$, which contains the numeric coefficients into a vector of symbols $\tilde{S}_i = \langle \alpha_1, ..., \alpha_l \rangle$. Then, we calculate $p(V(i,j,l))$ based on $\tilde{S}_i$.

Table 5.2: The equiprobable regions as defined under N(0,1) Gaussian distribution

| $\beta$ \ $\alpha$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|
| $\beta_1$ | -0.43 | -0.67 | -0.84 | -0.97 | -1.07 | -1.15 | -1.22 | -1.28 |
| $\beta_2$ | 0.43 | 0 | -0.25 | -0.43 | -0.57 | -0.67 | -0.76 | -0.84 |
| $\beta_3$ | - | 0.67 | 0.25 | 0 | -0.18 | -0.32 | -0.43 | -0.52 |
| $\beta_4$ | - | - | 0.84 | 0.43 | 0.18 | 0 | -0.14 | -0.25 |
| $\beta_5$ | - | - | - | 0.97 | 0.57 | 0.32 | 0.14 | 0 |
| $\beta_6$ | - | - | - | - | 1.07 | 0.67 | 0.43 | 0.25 |
| $\beta_7$ | - | - | - | - | - | 1.15 | 0.76 | 0.52 |
| $\beta_8$ | - | - | - | - | - | - | 1.22 | 0.84 |
| $\beta_9$ | - | - | - | - | - | - | - | 1.28 |

The effectiveness of converting from $V(i,j,l)$ to $\tilde{S}_i$ depends on a mapping function $F$ that maps $CD_l^j(o_i,m)$ into an alphabet $\mathcal{A} = \alpha_1, ..., \alpha_m$, where the soundness of translation from $CD_l^j(o_i,m)$ to $\alpha_m$ holds as follows:

$$F(CD_l^j(o_i,m)) = \alpha_m, \text{iff. } \beta_{j-1} \leq \alpha_m \leq \beta_j \ (1 \leq i \leq l, 1 \leq j \leq m) \tag{5.4}$$

To facilitate effective conversion, we use the data discretization scheme of SAX [69] to assign wavelet coefficients into $k$ equiprobable regions. Each continuous coefficient value that falls into a region maps to a unique symbol $\alpha_i$ $(1 \leq i \leq \kappa)$. Table 5.2 illustrates the equiprobable regions as defined under N(0,1) Gaussian distribution, in which the area between $\beta_j$ and $\beta_{j+1}(1 \leq j \leq 8)$ is $\frac{1}{\alpha}(3 \leq \alpha \leq 10)$[2]. The KLD in Equation 5.5 for V(1,j,l) and V(2,j,l) now becomes:

$$KLD(j,l) = KLD(p(V(1,j,l)), p(V(2,j,l))) = \sum_{i=0}^{k} p(\tilde{S}_1) \log \frac{p(\tilde{S}_1)}{p(\tilde{S}_2)} \tag{5.5}$$

There is a tradeoff involved in choosing the optimal size $\kappa$ of alphabet $\mathcal{A}$. That is, an large value $\kappa$ keeps more information about the distribution of the continuous data,

[2]Although the continuous wavelet coefficients may follows other distributions, it does not affect the effectiveness for calculating the equiprobable regions once the distributions has been identified.

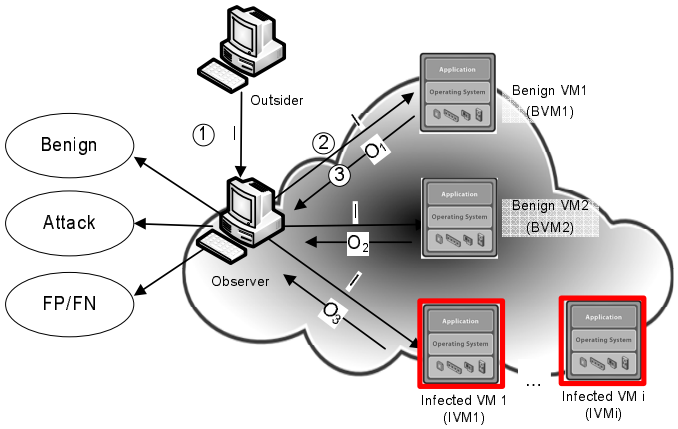



Figure 5.5: Experimental Setup.

but it might generate too many false alarms. In contrast, a small value of $\kappa$ keeps less information about the distribution, but it might be insensitive to the detection of slow or stealthy attacks. In Section 5.5.2, we will examine more closely this tradeoff and illustrate how we can determine the optimal value of $\kappa$.

Given intraFV, interFV, and KDL at scale $l$, we define the wavelet-based distance (WBD) between $O_1$ and $O_2$ for all scales as shown in Equation 5.6.

$$WBD(O_1, O_2) = \sum_{j=1}^{m} \left( \sum_{l=0}^{m} intraFV(j,l) \times \sum_{l=0}^{m} interFV(j,l) \times \sum_{l=0}^{m} KLD(j,l) \right)^2 \quad (5.6)$$

Basically, WBD summarizes the divergence of inter-flow, inter-flow, and KLD between two networking flows. A large value of WBD indicates significant difference between two flows, while a small value of WBD implies two flows are similar.

## 5.4 Implementation

### 5.4.1 Environment Setup

***Environment Construction*** Figure 5.5 shows the experimental setup. It consists of

our intrusion detection system (*Observer*) (See Chapter 4 for detail), two benign virtual machines ($BVM1$ and $BVM2$), and a series of infected virtual machines, which contains different CTCs ($IVMi(0 < i < n)$). To run experiments for a specific CTC, one infected virtual machine $IVMi$ will be chosen, and run in parallel with virtual machine Observer, $BVM1$, and $BVM2$. *Observer* intercepts all the inbound networking traffic, sending it to the $IVMi$, as well as $BVM1$ and $BVM2$. The outbound traffic from $BVM1$, $BVM2$, and $IVMi$ are immediately used for CTC detection. To ensure that $BVM1$ and $BVM2$ have the same virtual machine runtime state, $BVM1$ and $BVM2$ are cloned from the same snapshot $s_i$ of a virtual machine image.

***IDS setup*** The system architecture of Observer is illustrated in Figure 5.6 and has four main components: (1) the Traffic Filter identifies the packets that are sent to the infected VM $IVMi$; (2) the Traffic Distributor acts as a transparent bridge that forwards inbound packets to $BVM1$, $BVM2$, and $IVMi$ ; (3) the Output Analyzer uses timing distance metric to calculate the timing distance between the virtual machine pairs, $BVM1, BVM2$ and $BVM1, IVMi$; (4) the Traffic Manipulator only sends out the outbound traffic coming from $IVMi$. If the traffic of $IVMi$ has been identified as CTC traffic, the traffic manipulator also takes countermeasures to eliminate CTC in the traffic. A post-processing script compares the outbound flows from the three VMs and classifies the result into one of the following cases: (a) Observer successfully reports a CTC (*true positive*); (b) Observer reports benign flows as containing a CTC (*false positive*); (c) Observer reports a CTC as benign (*false negative*). In particular, the false positive rate is the proportion of legitimate sample in the test set that are mistakenly been identified as CTC, while the true positive rate is the percentage of CTC in the test that are correctly been identified as CTC. The false negative rate is the proportion of CTC sample in the test that are mistakenly been identified as legitimate, while the true negative rate is the percentage of CTC in the test set that are correctly been identified as legitimate.

We have implemented our CTC detection system in `C` on the top of VMware ESX Server 4.1 [31]. The *traffic filter* was implemented by `ipfw` [37], a customized transparent bridge,

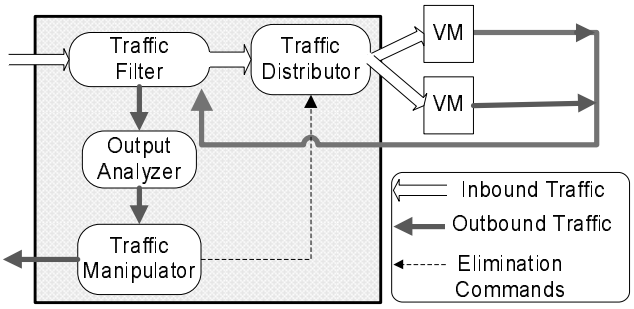



Figure 5.6: System Architecture of Observer

to intercept traffic. The *traffic distributor* was implemented by `divert socket` [108]. The *output analyzer* uses `ethereal` [32] to collect the outbound traffic, and `tethereal` [115] to separate the timing information from the outbound traffic. We also use `ntop` [82] to measure the statistics of traffic at runtime. The output analyzer was written in `C`, `perl`, `Dataplot`, and `MATLAB`.

### 5.4.2 CTC implementation

To emulate the malicious program that exfiltrates insider information. we have modified the source code of `vsftpd v2.3.4` [119] running on `Fedora Linux 9`. The modified vsftpd includes the CTC encoder by generating a timing delay before sending out the packet to the client. The CTC encoder was written in `C` and inline assembly that invokes instructions to the Read Time-Stamp Counter (RDTSC) of CPU [93]. We choose RDTSC instruction because it has excellent resolution and requires only low overhead for keeping time information [39]. A description of the procedure for generating the precise timing delay is shown in Figure 5.7. Specifically, the program first calibrates the CPU ticks and obtains the number of *nano* seconds per CPU tick. Then, based on the elapsed time, the program runs the loop for the number of *nop* instructions.

Meanwhile, we disable virtualization of the Time-Stamp Counter (TSC) within the VMs, so that the TSC within a virtual machine returns the physical machine's TSC value and writing the TSC from within the virtual machine has no effect. Each VM was configured

```
1. Initialize tick_begin=0, tick_end=0, tick_diff=0  //Initialize variables
2. tick_begin = getRDTSC()                           //Get starting time
3. Loop for n times
4. tick_end = getRDTSC()                             //Get ending time
5. tick_diff = (tick_begin-tick_end)                 //Get difference between starting and ending time
6. Get time elapsed, elaps_time, in nano second
7. nanosec_per_ticks = time_diff /elaps_time
8. noOfLoop = (10^9/nanosec_per_ticks)*w              //Calculate number of ticks for duration w (in sec)
9. Loop for noOfLoop times, each time run nop        //Generate precise delay for duration w
```

Figure 5.7: The procedure to generate precise timing delay

*one hop* away from Observer.

### 5.4.3 Detection Methods

We use four board classes of measurements to detect CTCs: statistical tests, the timing distortion metric for measuring low latency anonymous network [58], the corrected conditional entropy method [38], and our wavelet-based distance (WBD) method. For the statistical tests, we use: 1) shape tests, which describes the first-order statistics, e.g., mean, standard deviation, and empirical cumulative distribution (ECD); 2) Kolmogorov-Smirnov test (KS-Test) [50], which has been described in Chapter 4; 3) Welch's T-test (WT-Test) [127], and 4) the regularity test (RT-Test) [21].

WT-Test is used to determine whether the means of two samples are with different sample sizes and variance. The statistical $WT-Test$ is defined as Equation 5.7.

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{{s_1}^2}{N_1} + \frac{{s_2}^2}{N_2}}} \tag{5.7}$$

where $\overline{X_i}$, ${s_i}^2$, and $N_i$ are the $i$th sample mean, variance, and sample size, respectively.

For RT-Test determines whether the variance of the inter-packet delays is relatively constant. In our evaluation, we first divide the sequence of outbound IPDs into $k$ segments, and calculate the standard divination of segment $i$ that is denoted as $\sigma_i$. The regularity

score is defined in Equation 5.10. It is the standard deviation of the pairwise difference between each $\sigma_i$ and $\sigma_j$ for all segment $i < j$. The reason that we choose RT is based on the observation that the variance of the inter-packet delays changes over time for most types of network traffic. With CTCs, the code used to transmit data is a regular process and, as a result, the variance of the inter-packet delays remains relatively constant over time.

$$regularity = STDEV(\frac{|\sigma_i - \sigma_j|}{\sigma_i}, i < j, \forall i, j) \tag{5.8}$$

As stated in Chapter 4, the KS-Test measures the distance between the legitimate sample and the test sample. The small test score implies that the test sample is close to the legitimate sample. In contrast, the large test score indicates that the possible occurrence of a CTC. By contrast, the small score of the regularity test indicates the possible existence of a CTC.

Jin et al. [58] proposed an wavelet-based metric (MLAN) that quantitatively measures the practical effectiveness of anonymous networks in the presence of timing attacks. The timing attacks in networking traffic refers to transformations that mix, split, merge, add chaff, and/or drop packets introduced by anonymous networks, such as Tor [90] and Anonymizer.com [6]. Thus, this metric is applicable to determine how likely a networking flow has been distorted by a CTC. The MLAN measures

$$e_j = \frac{\sum_{p=0}^{N_j-1}[CD(X,Y,j)(p)]^2}{N_j} \tag{5.9}$$

where $CD(X,Y,j)(p)$ is the $p$th ($p = 0...N_j - 1$) wavelet detail coefficient at scale $j$ for the $j$th vector $D(X,Y,j)$. $N_j = 2^{-j}n_j$ is the number of the wavelet detail coefficient at scale $j$.

Gianvecchio et al. [38] showed that the corrected conditional entropy (CCE) is effective in detecting different types of CTCs. The CTC traffic is shown to change the entropy of a legitimate traffic. Thus, the corrected conditional entropy can be used to determine if

an obvious deviation has been detected in the change of entropy. Specifically, the CCE measures

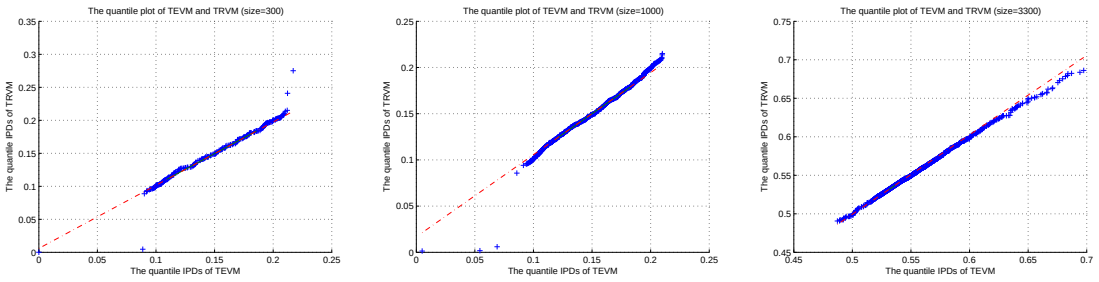
$$CCE(x_m|x1,...x_{m-1}) = CE(x_m|x1,...x_{m-1}) + perc(x_m).EN(x_1) \tag{5.10}$$

where $CE(x_m|x_1,...,x_{m-1})$ is the estimated conditional entropy for the pattern $x_1,...,x_m$. $perc(x_m)$ is the percentage of unique patterns of length $m$ and $EN(x_1)$ is the entropy with $m$ fixed at one, i.e., only the first-order entropy.

## 5.5 Evaluation

In this section, we first analyze the similarity between benign outbound traffic, which justifies the assumption of our approach (Section 5.5.1). Then, we analyze the optimal values of parameters which might affect the effectiveness of detection (Section 5.5.2). After that, we study the effectiveness of our approach for detecting different CTCs (Section 5.5.3). Finally, we evaluate the robustness of our metric to detect CTCs in the presence of noises (Section 5.5.4).

### 5.5.1 Similarity Between VMs for Legitimate Traffic




(a) sample size=300 (b) sample size=1000 (c) sample size=33000

Figure 5.8: The quantile-quantile (QQ) plot of the IPD of two legitimate flows

The objective of the first set of experiments is to justify the observation of close correlation between benign VMs, with respect to outbound traffic. Given the same inbound traffic,

we collect datasets of three different sizes of IPDs from BVM1 and BVM2. Figure 5.8 illustrates the quantile-quantile (QQ) plot of the IPDs of two legitimate outbound flows with the size 300, 1000, and 33000. The highly linear nature of all plots strongly indicates that the outbound flows from two benign VMs follow the same distribution.

Table 5.5.1 shows the test results from the datasets. For the WT-Test and the KS-Test at the 5% significance level, the value $h = 0$ means that one should accept the null hypothesis; i.e., that two flows come from the same distribution, while $h = 1$ means one should reject the null hypothesis. The p-values of the *WT-Test* and the *KS-Test* show whether two samples differ significantly, say rejecting the null hypothesis if the p-values are "small". When the sample size is small (size=100, 1000), both the *WT-Test* and the *KS-Test* accept the null hypothesis. However, when the sample size is large (size=33000), both tests *reject* the null hypothesis. This behavior results from the fact that IPDs contain more noise when there is no long-term time synchronization mechanism applied between VMs. It also indicates that the smaller detection window can detect CTCs more accurately than the larger detection window. Our average WBD result shows that all legitimate flows have values lower than 32.5, indicating that the legitimate flows are indeed very similar.

Table 5.3: The statistic values of IPDs of legitimate flows of different size

| | data size=300 | | data size=$1,000$ | | data size=$32,000$ | |
|---|---|---|---|---|---|---|
| Type of traffic | Legit1 | Legit2 | Legit1 | Legit2 | Legit1 | Legit2 |
| Mean | 0.148 | 0.148 | 0.150 | 0.150 | 0.5353 | 0.5357 |
| Stdev | 0.039 | 0.040 | 0.035 | 0.0332 | 0.0093 | 0.0091 |
| WT-Test(h) | 0 | | 0 | | 1 | |
| WT-Test(p) | 0.9111 | | 0.948 | | $1.0e-003$ | |
| WT-Test(ci) | $[-0.0067, 0.0060]$ | | $[-0.0031, 0.0029]$ | | $[-0.5370 - 0.2540]$ | |
| KS-Test(h) | 0 | | 0 | | 1 | |
| KS-Test(p) | 0.9859 | | 0.2816 | | $6.5703e-035$ | |
| KS-Test(ci) | 0.0368 | | 0.0440 | | 0.0492 | |
| WBD | 32.52464 | | 5.38012 | | 13.0492 | |

### 5.5.2 The Choice of $\kappa$ and $\omega$

One important quantity that reflects the effectiveness of our wavelet-based approach is the calculation of Kullback-Leibler divergence that measures the distributions of two discretized wavelet vectors, where the choice of the size of alphabet $\mathcal{A}$, $\kappa$, is critical. As we stated above in Section 5.3.3, the tradeoff in choosing $\kappa$ is that a small value of $\kappa$ keeps less information about the distribution of the data, thus leads to an increased number of false negatives. In contrast, a larger value of $\kappa$ retains more information about the distribution of the data and increases the detection sensitivity, thereby leading to an increased number of false positives. To determine the optimal value of $\kappa$, we run tests for $\alpha = 2$ through 10 for both legitimate and different versions of *Jitterbug* flows. All flows contain 1,000 packets. Figure 5.9 illustrates the WBD value for each case. It is clear that when the value of $\kappa$ is 9 or 10, the WBD of legitimate and malicious cases can be easily differentiated. We obtain a similar result in other CTCs detection. Therefore, we choose $\kappa = 10$ to retain the ability of measuring the deviation of malicious traffic. The WBD values calculated in this dissertation all use $\kappa = 10$, except where otherwise stated.

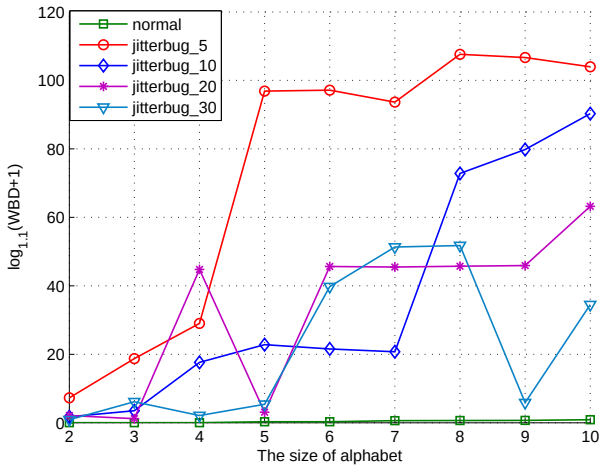



Figure 5.9: WBD plots of different networking flows for different $\kappa$

The second important quantity is the detection window size $\omega$, which is the number of IPDs to be compared between networking flows. In Section 5.5.1, our result indicates that

too large a value for $\omega$ might lead to false negatives because of noise and the absence of synchronization. In addition, a large $\omega$ also causes longer processing time of wavelet decomposition and slower response, which means that more information could be transmitted before a CTC is detected. Therefore, it is always good to have a relatively small $\omega$. However, choosing a too-small $\omega$ is also problematic, because it makes the IDS over-sensitive, leading to an increased number of false positives. Figure 5.10 shows the WBD value for window size $\omega = 100$ through 1,000 for both legitimate and different versions of *Jitterbug* flows. All flows contain 1,000 packets. It is clear that the WBD values of legitimate and malicious flows can be easily differentiated when $\omega \geq 800$. We obtain similar results with other CTCs. Therefore, the WBD values calculated in this dissertation all use $\omega = 1000$, except where otherwise stated. The issues of noises and time synchronization that could possibility lead to false negatives are discussed in Section 5.5.4.

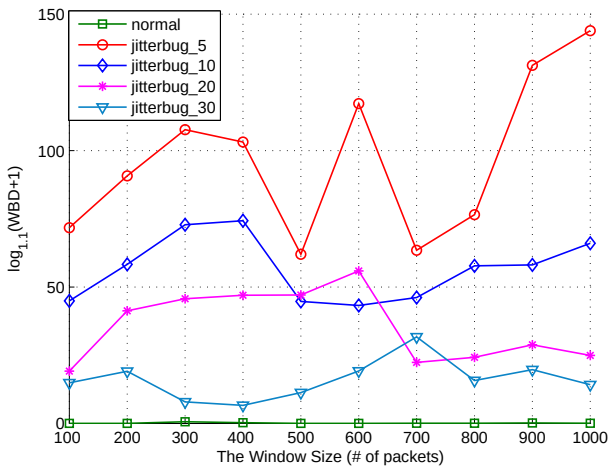



Figure 5.10: WBD plots of different networking flows for different $\omega$

### 5.5.3 Detection of CTCs

To evaluate the effectiveness, we evaluate our WBD metric with a wide range of CTCs in Table 5.4, which cover both *active* CTC and *passive* CTCs. Although some CTCs have been

Table 5.4: Covert Timing Channels Used in Evaluation

| **Name** | **Attack Description** | **Active/ Passive** | **Detectable?** |
|---|---|---|---|
| IP Channel (IPCTC) [22] | Transmitting `1`-bit or `0`-bit by choosing *send* or *not send* a packet during interval $w$ | Passive | Yes |
| Time-Replay Channel (TRCTC) [22] | Transmitting `1`-bit or `0`-bit by choosing historic legitimate IPDs as additional input from different baskets | Active | Yes |
| Botnet Traceback Watermark(BTW) [92] | Injecting modified control text | Passive | Yes |
| JitterBug [104] | Operates small delays in keystrokes to affect the original IPDs | Passive | Yes |

studied by a model-based off-line approaches [38], ours is the first online CTC detection approach that does not require any training data. Below, we describe the detection for each of the CTCs in greater detail.

### IPCTC Detection

Our first sets of experiments is designed to test the ability of our system to detect the presence of IPCTC [21]. The IPCTC encodes a 1-bit by transmitting a packet during a timing interval $w$, and encodes 0-bit by not transmitting a packet during $w$. The distribution of IPCTC inter-packet delays is determined by the timing interval $w$ and the number of 0-bits between two 1-bits. To avoid creating a pattern of inter-packet delays at multiples of a single $w$, the encoding scheme can choose different values $w$, and rotate among them [22]. For example, in Figure 5.11, in order to transmit the byte, `1011 0010`, the sender sends four packets within the duration of $8w$. The 1-bits were transmitted at the $1st$, $3rd$, $4th$, and $7th$ timing interval. The message is first encoded by the encoder, and then transmit bit by bit to the receiver. The message is rebuilt by the decoder once it receives the bit stream.

In our experiment, the CTC encoder reads the "*/etc/passwd*" file and encodes its binary into IPCTC. For IPCTC, we use four versions of encoding schemes. Given the observation that the average IPD of traffic is 0.147s , we design the first three schemes by choosing the

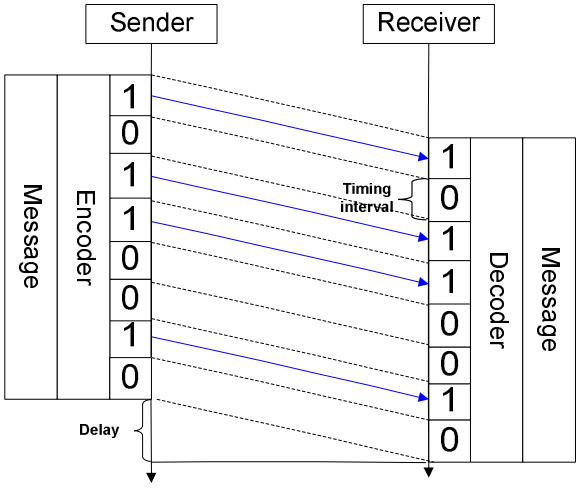



Figure 5.11: The encoding scheme of IPCTC

timing interval $w$ of $0.1s$ (IPCTC_V1), $0.12s$ (IPCTC_V2), $0.14s$ (IPCTC_V3). The fourth scheme rotates among the above $w$s after each 100 packets (IPCTC_V4) to avoid creating a regular pattern of inter-packet delays. This design makes sure that both legitimate and CTC flow have almost the same duration and send almost the same number of packets at run-time.

**Detection Analysis** In the test, we run 100 times for a duration of 50 seconds. In each flow, we collect around 400 packets. For the regularity test, we divide the sequence of inter-packet delays into 20 segments. According to [38], IPCTC is the easiest CTC to detect because its abnormality shown up in simple statistical tests. From Figure 5.12, it is obvious that the empirical cumulative distribution (ECD) of legitimate flows and IPCTC flows are quite different. In addition, the ECD of two legitimate flows, which come from two benign VMs, are very close.

Table 5.5 shows more detailed results of all tested flows. Although all the tests can detect all IPCTCs, our WBD measurement shows the better results: the WBD of legitimate flows is very small (0.6295), which is in stark contrast to the WBDs of IPCTC flows, which all all 16,0000 or higher. Although MLAN and CCE can differentiate all IPCTCs from legitimate flow, it is almost impossible to differentiate different versions of IPCTCs. For example, the

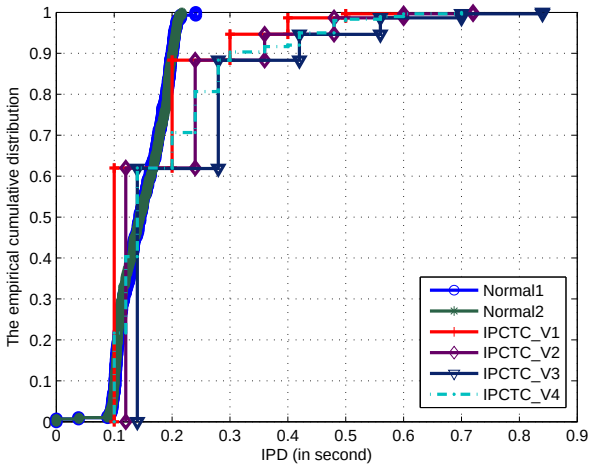



Figure 5.12: The empirical cumulative distribution of legitimate and IPCTC IPDs

CCE for different IPCTC are all close to 0.609.

Table 5.5: The test scores of IPCTC

| | **Legit** | **IPCTC_V1** | **IPCTC_V2** | **IPCTC_V3** | **IPCTC_V4** |
|---|---|---|---|---|---|
| Average # of IPDs | 0.1472 | 0.156 | 0.187 | 0.218 | 0.188 |
| Mean | 0.1472 | 0.156 | 0.187 | 0.218 | 0.188 |
| Stdev | 0.041 | 0.089 | 0.106 | 0.124 | 0.111 |
| Regularity | 0.9723 | 0.9723 | 0.9723 | 0.9723 | 0.9723 |
| T-Test | 0 | 1 | 1 | 1 | 1 |
| KS-Test | 0 | 1 | 1 | 1 | 1 |
| KS-Test(p value) | 0.96702 | 0 | 0 | 0 | 0 |
| MLAN | 0.0433 | 1.3616 | 1.5072 | 1.7034 | 1.353 |
| CCE | 1.1874 | 0.6055 | 0.6095 | 0.6099 | 0.7614 |
| WBD | 0.6295 | 15982.6817 | 37092.8784 | 76188.4579 | 50241.9389 |

Since IPCTC_V1, IPCTC_V2, and IPCTC_V3 use the same frequency to send packets, but different $w$s, the regularity of these three CTCs are identical. Therefore, it is impossible to differentiate legitimate traffic from IPCTCs using the regularity test. Interestingly, when the regularity tests combine with our WBD, it is easy to identify IPCTCs using different $w$ because different WBDs with the similar regularity indicate IPCTCs that are of the same

type, differing only in the parameter $w$ used in the encoding scheme.

### TRCTC Detection

Our second set of experiments investigates how our metric detect TRCTC [22]. Compare to IPCTC, TRCTC is a more advanced CTC that replays a set of legitimate inter-packet delays to mimic the behavior of legitimate traffic. Figure 5.13 illustrates that TRCTC collects a sample of legitimate traffic $Bin_i$ as input and replays $Bin_i$ to transmit information. $B_i$ is partitioned into two equal bins $Bin_0$ and $Bin_1$. TRCTC transmits a `0` bit by randomly replaying an inter-packet delay from bin $Bin_0$ and transmits a `1` bit by randomly replaying an inter-packet delay from bin $Bin_1$. Since $Bin_i(i = 0 and 1)$ is made up of legitimate traffic, the distribution of TRCTC traffic is approximately equal to the distribution of legitimate traffic. The encoding scheme TRCTC is first encoded by encoder, and then transmit bit by bit to the receiver. Then, the message is rebuilt by the decoder bit by bit, which is similar to that of IPCTC.

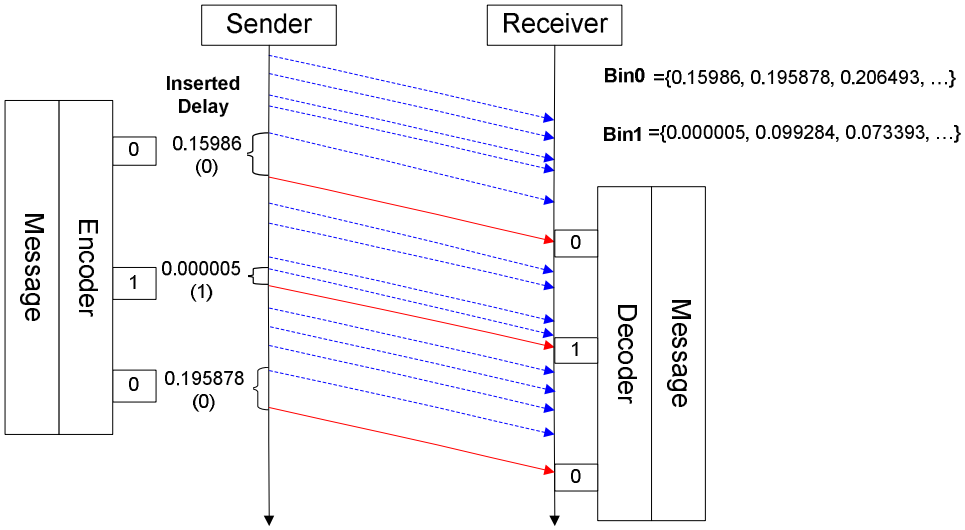



Figure 5.13: The encoding scheme of TRCTC

In our experiments, we designed four versions of TRCTCs (TRCTC_V1, TRCTC_V2,

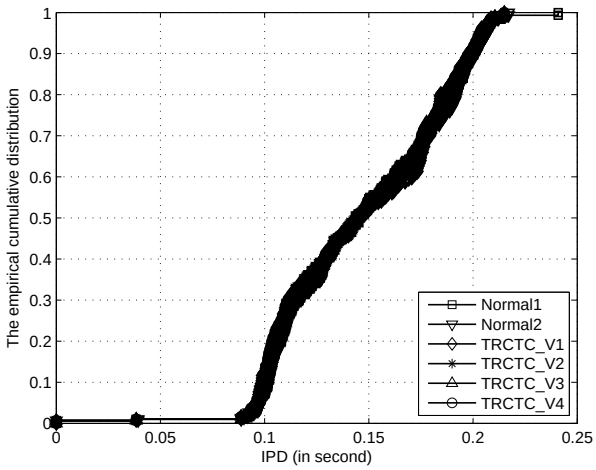



Figure 5.14: The distribution of legitimate flow and different TRCTC flows

TRCTC_V3, TRCTC_V4) by injecting one additional bogus packet into the flow after every $a$ packets ($a = 5, 10, 15, 20$). Of course, a larger value of $a$ indicates a slow attack. Figure 5.14 shows that the empirical cumulative distribution of legitimate IPDs and four versions of TRCTC IPDs are almost identical due to the encoding mechanism of TRCTC.

Table 5.6: The test scores of TRCTC

| | **Legit** | **TRCTC_V1** | **TRCTC_V2** | **TRCTC_V3** | **TRCTC_V4** |
|---|---|---|---|---|---|
| Mean | 0.1472 | 0.1465 | 0.1471 | 0.1466 | 0.1467 |
| Stdev | 0.0406 | 0.0411 | 0.0409 | 0.0410 | 0.0411 |
| Regularity (group size=20) | 0.2452 | 0.2958 | 0.1991 | 0.3009 | 0.1955 |
| T-Test | 0 | 0 | 0 | 0 | 0 |
| KS-Test | 0 | 0 | 0 | 0 | 0 |
| KS-Test(p value) | 0.9670 | 0.6451 | 0.9982 | 1 | 1 |
| MLAN | 0.0433 | 0.87 | 1.0074 | 0.8995 | 0.8561 |
| CCE | 1.1833 | 1.1589 | 1.1749 | 1.1813 | 1.1829 |
| WBD | 0.4669 | 859.5832 | 651.9172 | 455.5041 | 318.9133 |

**Detection Analysis** Table 5.6 shows the similar results that are similar to as those with IPCTC. Since TRCTCs demonstrate the distribution as legitimate traffic, the WT-Test

and the KS-Test fail to detect all TRCTCs, since they accept the null hypothesis that the IPDs of TRCTC and the legitimate traffic follow the same distribution. Although MLAN and CCE test can differentiate TRCTC from legitimate traffic, they are incapable of identifying slow TRCTCT. For both tests, the aggressive (TRCTC_V1) and the stealthy TRCTC (TRCTC_V4) yield similar results. e.g., the MLAN test for TRCTC_V1 and TRCTC_V4, and the CCE test for TRCTC_V3 and TRCTC_V4. However, the testing results shown that our WBD test can achieve detection and stealthy TRCTC identification at the same time.

**BTW Detection**

Back-track Watermark (BTW) is a passive CTC that specifically designed to track back the communication between a bot and its bot-master [92]. Specifically, to encode an $i$-bit sequence $S = s_0, ..., s_{i-1}$, we use $2i$ randomly chosen packets pairs:$\langle P_{r_i}, P_{e_i} \rangle$ $(i = 0, ..., L)$, such that $r_i \leq e_i$, in which $P_{r_i}$ is called a *reference packet* and $P_{e_i}$ is called an *encoding packet*. Let $l_e$ and $l_r$ be the inter-packet delays of the covert bit encoding and reference packets, respectively. A covert bit $s_k (0 \leq k \leq i-1)$ was encoded into the packet pair $\langle P_{r_i}, P_{e_i} \rangle$. Specifically, we use the watermark bit encoding function defined in Equation 5.11 to adjust the length of the watermark encoding packet $P_{ei}$. We use a pseudo-random number generator (PRNG) and seed $s_t$ to generate the random time $t_{ei}$ at which $P_{ei}$ will be sent out.

$$e(L_r, L_e, L, s_k) = l_e + [(0.5 + s_k)L - (l_e - l_r)] mod 2L \tag{5.11}$$

For the encoding function, given the PRNG and $s_t$ and the approximate time $t_{ei}$ at which the watermark encoding packet $P_{ei}$ should arrive. Then we use the packets in the time interval $[t_{ei} - \frac{\delta}{2}; t_{ei} + \frac{\delta}{2}]$ to decode the CTC. Specifically, we use the sum of the lengths of all the packets in the time interval $[t_{ei} - \frac{\delta}{2}; t_{ei} + \frac{\delta}{2}]$ as the length of the watermark encoding packet and apply that to the watermark bit decoding Equation 5.13.

$$d(L_r, L_e, L) = \lfloor \frac{(l_e - l_r)}{L} \rfloor mod 2 \tag{5.12}$$

iff.

$$(-0.5 + 2i)L \leq x_e - x_r \leq (-0.5 + 2i)L \tag{5.13}$$

We further extend the initial CTCs design scheme to generate slow attacks. In particular, we use $2ai(a \geq 1)$ packets to encode bit sequence $S$, where the parameter $a$ can either be a constant or a variable generated by a PRNG. $P_{r_i}$ and $P_{e_i}$ were chosen from $2a$ packets. We call $a$ the *amplifier*, which indicates how *slow* the information can be transmitted. The larger the value of $a$, the slower an attack can proceed.

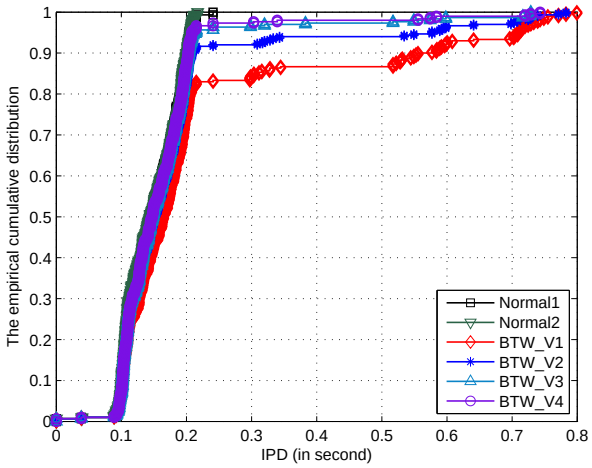



Figure 5.15: The distribution of legitimate flow and different BTW flows

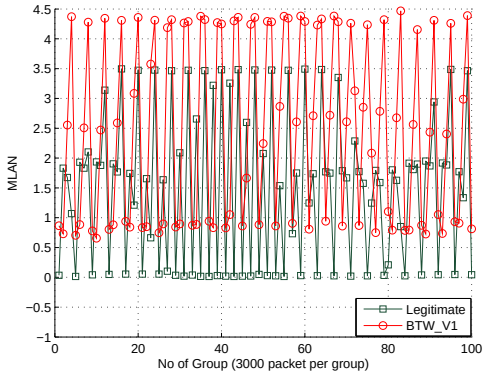
In our experiment, we construct four version of BTW by using different values of $a$. We have generated 60 seconds of live traffic from all VMs. Figure 5.15 illustrates the empirical cumulative distribution of two legitimate flows and four CTC flows. Except for the traffic for BTW_V1 and BTW_V2, all of the rest of the traffic have quite similar distributions. Compared to the legitimate traffic, the BTW traffic contains more IPDs, which are larger than 0.2s.

Table 5.7: The test scores of BTW

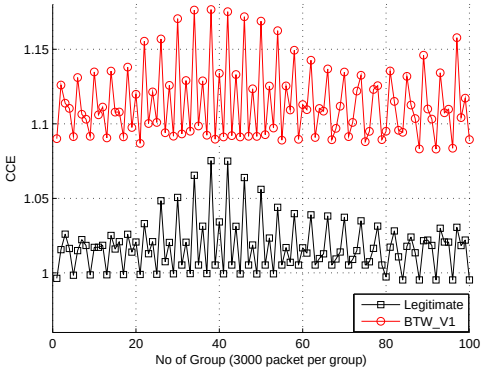
| | **Legit** | **BTW_V1 (a=5)** | **BTW_V2 (a=10)** | **BTW_V3 (a=20)** | **BTW_V4 (a=30)** |
|---|---|---|---|---|---|
| Mean | 0.1474 | 0.2207 | 0.1814 | 0.1627 | 0.1587 |
| Stdev | 0.0399 | 0.1775 | 0.1304 | 0.0918 | 0.0828 |
| Regularity (group size=20) | 0.2452 | 1.3504 | 0.7595 | 0.9206 | 0.9723 |
| T-Test | 0 | 1 | 1 | 1 | 0 |
| KS-Test | 0 | 1 | 0 | 0 | 0 |
| KS-Test (p value) | 0.9670 | 0.0004 | 0.2808 | 0.9862 | 0.9999 |
| MLAN | 0.0433 | 1.8296 | 1.2081 | 1.4790 | 1.2394 |
| CCE | 1.1848 | 1.0440 | 1.0842 | 1.10789 | 1.1312 |
| WBD | 2.6757 | 1489.8973 | 828.7632 | 250.8589 | 124.9655 |

**Detection Analysis** Our experimental result demonstrates that the mean and standard deviation of legitimate and CTC IPDs are similar, particularly for larger values of $a$. The WT-Test and KS-Test can detect aggressive BTW (BTW_V1); however, they all fail to detect BTW_V4, which is the stealthiest CTC in this set of experiments. Figure 5.16 illustrates the testing scores of a legitimate flow and a BTW flow. It is obvious that the testing score of MLAN generates a large percentage of false positive, because its testing scores of legitimate and BTW flow are not distinguishable for more cases. Table 5.5.3 demonstrates that although MLAN can reach 100% true positive rate, it generates 65% false positive rate.

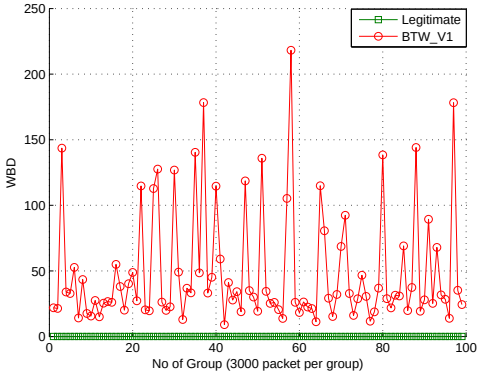
Compared to all the above tests, the CCE and WBD tests yield good results: they can not only detect all BTWs, but also reach 100% true positive with zero false positives. In particular, the WBD gives better results than CCE because it can quantify the degree of stealthiness for each BTW, in which the larger value of WBD indicates the more aggressive CTC, while CCE cannot.




(a) MLAN test (size=300, w=10)




(b) CCE test (size=300, w=10)




(c) WBD test (size=300, w=10)

Figure 5.16: The testing scores of a legitimate flow and a BTW_V1 flow

Table 5.8: BTW detection rates

| | **NT-Test** | **KS-Test** | **MLAN≥ 0.65593** | **CCE≥ 0.9953** | **WBD≥ 0.0015** |
|---|---|---|---|---|---|
| Legitimate (False Positive) | 0% | 24% | 65% | 0% | 0% |
| BTW_V1 (True Positive) | 0% | 87% | 100% | 100% | 100% |

### JitterBug Detection

JitterBug is a passive CTC [104] that manipulates the existing networking traffic. It operates by creating small delays in key-presses that affect the inter-packet delays of an application. It transmits a `1` bit by increasing an IPD to a value modulo $w$ milliseconds and

transmits a 0 bit by increasing an IPD to a value modulo d $\lfloor \frac{w}{2} \rfloor$ milliseconds. For small values of $w$, the distribution of JitterBug traffic is very similar to that of the original legitimate traffic. However, because of the small value $w$, it can also cause the CTC to be indistinguishable from the legitimate traffic containing noises.

The $w = 100$ milliseconds was chosen for our experiments. We use four versions of JitterBug, each with a different value for the amplifier: JitterBug_V1 ($a$=5), JitterBug_V2 ($a$=10), JitterBug_V3 ($a$=20), and JitterBug_V4 ($a$=30). This design makes sure that both legitimate and CTC flow have almost the same duration and send almost the same number of packets at run-time.

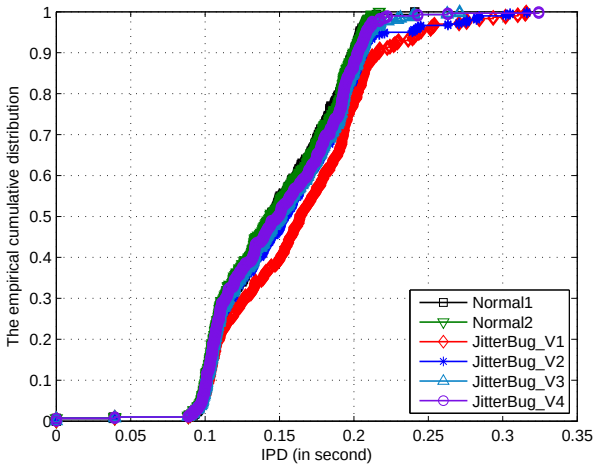



Figure 5.17: The distribution of legitimate flow and different JitterBug flows

To evaluate the detection and true/false positive rate, we collect 30,000×10 packets in real-time. Table 5.9 shows the testing scores of all tests. The mean and standard deviation of legitimate traffic (mea=0.1472, stdev=0.0406) and stealthy CTC traffic are very similar, particularly for JitterBug_V4 (mea=0.1497, stdev=0.0432). Both the WT-Test and the KS-Test fail to detect JitterBugs, except the most aggressive one (JitterBug_V1). For the regularity test, although the smaller testing scores imply CTC for aggressive JitterBug_V1 and JitterBug_V2, it fails to detect more stealthy CTC, such as JitterBugs_V3 and JitterBugs_V4. The CCE test cannot differentiate different versions of Jitterbugs because

Table 5.9: The Test Scores of JitterBug

| | **Legit** | **JitterBug _V1** | **JitterBug _V2** | **JitterBug _V3** | **JitterBug _V4** |
|---|---|---|---|---|---|
| Mean | 0.1472 | 0.1626 | 0.1549 | 0.1510 | 0.1497 |
| Stdev | 0.0406 | 0.0502 | 0.0482 | 0.0432 | 0.0432 |
| Regularity (group size=20) | 0.2452 | 0.2125 | 0.1848 | 0.2453 | 0.2969 |
| T-Test | 0 | 1 | 0 | 0 | 0 |
| KS-Test | 0 | 1 | 0 | 0 | 0 |
| KS-Test(p value) | 1-e7.5 | 1-e7.5 | 1-e7.5 | 1-e7.5 | 1-e7.5 |
| MLAN | 0.0433 | 0.638 | 0.6957 | 0.8286 | 0.07097 |
| CCE | 1.1834 | 1.20119 | 1.2015 | 1.19333 | 1.19443 |
| WBD | 7.5561E-05 | 0.0218 | 0.0341 | 0.0349 | 0.0645 |

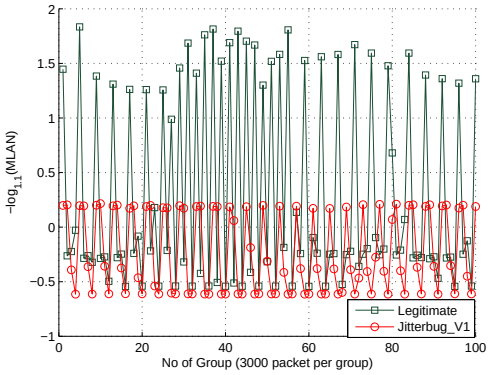
most of the test scores are between 1.18 and 1.20. The WBD can detect the most stealthy JitterBug, JitterBug_V4 because its testing score is 0.0645, which is much larger than that of the legitimate traffic.

Table 5.10: Jitterbug detection rates

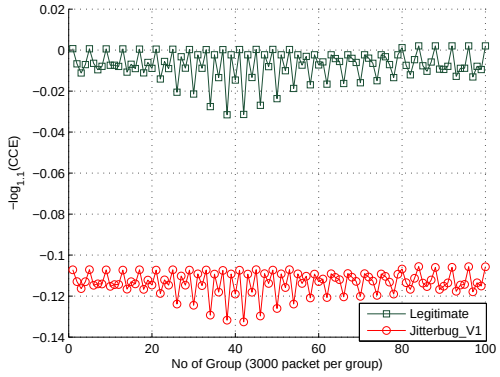
| | **NT-Test** | **KS-Test** | **MLAN≥ 0.60995** | **CCE≥ 0.9953** | **WBD≥ 0.00018** |
|---|---|---|---|---|---|
| Legitimate (False Positive) | 0 | 31.4% | 66% | 0 | 5% |
| Jitterbug_V1 (True Positive) | 62.4% | 70.1% | 100% | 100% | 100% |

For true/false positive test, Table 5.5.3 tabulates the true/false positive rates of the tests. Although the NT-Test demonstrates a zero false positive rate, its true positive rate is as low as 62.4%. Similarly, the KS-Test and the MLAN test have relative higher true positive rates (70.1% for the KS-Test and 100% for the MLAN) than that of the NT-Test, but the false positive rate of both are very high (31.4% for the KS-Test and 66% for the MLAN) as well. Our approach can achieve low false positive rate and high true positive

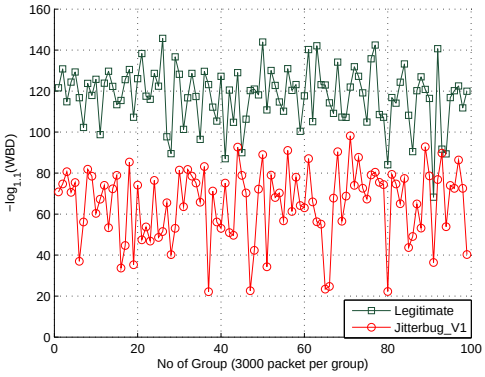
rate at the same time: when WBD $\geq 0.00018$, the false positive is 5% , while true positive is the 100%. Figure 5.18 demonstrates the testing scores of legitimate and Jitterbug_V1. In investigating the reason for the false positives, we found that most false positives happened after 138,000 packets had been received. This is most likely due to the fact of the lack of timing synchronization between the VMs in the long-term. To address this problem, one possible solution is that the virtual machine monitor should periodically synchronize the clocks in the involved VMs. We are currently investigating such techniques to reduce the false positive rate.




(a) MLAN test (size=300, w=10)




(b) CCE test (size=300, w=10)




(c) WBD test (size=300, w=10)

Figure 5.18: The testing scores of a legitimate flow and a Jitterbug_V1 flow

Table 5.11: The embedded noises used in our experiment

| | **Legit** | **Legit +Noise1** | **Legit +Noise2** | **Legit +Noise3** | **Legit +Noise4** |
|---|---|---|---|---|---|
| Noise Type | | Normal1 | Normal2 | Exponential | Uniform |
| mean | 0.1470 | 0.0147 | 0.0147 | 0.0025 | 0.0196 |
| stdev | 0.0405 | 0.0040 | 0.0406 | 0.0001 | 0.0001 |
| WBD mean | 0.1100 | 0.3796 | 1.0751 | 0.1861 | 0.7356 |
| WBD stdev | 0.0290 | 0.0522 | 0.1392 | 0.0237 | 0.0966 |

### 5.5.4 The Impact of Noise

One significant concerns about CTC detection in virtual machines is the impact of noise due to the effect of time drifting. Although our previous experiments demonstrate the effectiveness of our approach to detect CTCs with the presence of timing drifting, our next set of experiments is designed to investigate the robustness of our approach against timing drifting. In particular, we investigate the variances of WBD with the presence of noise follows different distribution. We create noises that follow four different distributions. Noise1 and Noise2 are both follow a *normal* distribution. They have the same mean but different standard deviations. Noise3 and Nose4 follow an exponential distribution and a uniform distribution, respectively. We use the same mechanism of CTC generator to embed different noises into the legitimate traffic. We collect 1000 packets in each case. Table 5.11 illustrates the WBD scores do not fluctuate greatly due to the noise. Regardless of noise, the upper bound of WBD values for all flows is 1.0751, which is very small. The legitimate flow has the smallest WBD value. Noise2 has the highest WBD value because its standard deviation (0.0406) is even larger than its mean (0.0147). The variances of WBD scores are small in the presence of noise, which means our WBD is rebuts enough with the presence of noise.

## 5.6 Conclusion

In this chapter, we have investigated a wavelet-based metric to detect CTCs in a networked virtual environment. We designed and implemented our CTC detection system on the top of Observer, which has been discussed in detail in Chapter 4. We have applied our wavelet-based approach to detect different types of CTCss with real-time experiments. Our experimental results show that our metric is capable of detecting a variety of CTCs. We have found that our approach is robust even in the presence of inaccurate time-keeping mechanisms of VMs.

# Chapter 6: Conclusion

This dissertation intends to create a secured and scalable framework that supports efficient mobile applications.

First, to overcome limitation and weakness of existing mobile applications, we have designed and implemented a mobile application product, namely Earthchildren. Regardless of specific software developing IDEs, our system addresses most important functionalities of a mobile devices, such as geographic position tracking, real-time graphic display, embedded database management, and wireless communication. Our mobile application has been deployed successfully on commercial mobile phones.

Second, we have presented SQLProb, a novel online and adaptive prevention system against SQLIAs. Unlike most existing protection approaches, our approach is fully modular and does not require access to the source code of the web applications or the database. In addition, our system is easily deployable to existing enterprise environments and can protect web applications without modifying their source code. To measure the performance and overhead of our technique, we have developed a prototype of SQLProb. Our experiment results indicate that we can achieve high detection rate with reasonable performance overhead. This prominent feature makes our system ideal for environments where software or architecture changes is not an economically viable option.

Third, to detect unauthorized information leakage from malicious virtual machines, we have presented Observer, a real-time intrusion detection system against covert channels. Observer detects new covert channels in a networked virtual environment by running a secure VM to mimic the suspect VM, such that the difference between two VMs can be identified. Unlike most existing covert channels detection approaches, our approach does not depend on modeling historic data, and can detect covert storage channels at run-time. Our experimental results demonstrate that we can achieve low latency and high detection

rate with reasonable overhead.

Fourth, to detect covert timing channel, which could exfiltrate sensitive information from a networked virtual environment, we have proposed, designed, and implemented intrusion detection system that detect covert timing channel in real-time. Our experimental results show that our wavelet-based metric can detect most current covert timing channels, without any legitimate networking traffic or model.

With the advances of mobile devices and cloud computing technology, we expect that the emerging technologies will be integrated in our proposed framework, which make future computing more easy to use and more secure.

# Bibliography

# Bibliography


[1] Fedora Linux project. `http://fedoraproject.org/`, [Accessed: July 1, 2008].

[2] Paul S. Addison. *Multiresolution signal decomposition: transforms, subbands, wavelets*. Academic Press, second edition edition, October 2000.

[3] Paul S Addison. *The illustrated wavelet transform handbook*. Taylor & Francis, first edition edition, July 2002.

[4] Dakshi Agrawal, Selçuk Baktir, Deniz Karakoyunlu, and Pankaj Rohatgi Berk Sunar. Trojan detection using IC Fingerprinting. In *Proceedings of the 28th IEEE Symposium on Security and Privacy (S&P)*, pages 296–310, Oakland, CA, May 2007. IEEE Press.

[5] Android. Android project. `http://www.android.com/`, [Accessed: October 20, 2011].

[6] Anonymizer.com. The anonymizer.com. `http://www.anonymizer.com/`, [Accessed: January 1, 2011].

[7] Aslan Askarov, Danfeng Zhang, and Andrew Myers. Predictive black-box mitigation of timing channels. In *Proceedings of the 17th ACM Computer and Communication Security Conference (CCS)*, pages 297–307. ACM Press, October 2010.

[8] Microsoft Azure. Microsoft Azure Services Platform. `http://www.microsoft.com/azure/default.mspx`, [Accessed: January 1, 2009].

[9] Davide Balzarotti, Marco Cova, Viktoria V. Felmetsger, and GiovanniVigna. Learning fingerprints for a database intrusion detection system. In *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS)*, pages 264–280. Springer-Verlag Berlin, Heidelberg, 2002.

[10] Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, and V. N. Venkatakrishnan. Announcing the Advanced Encryption Stardard (AES). National Technical Information Service (NTIS), October 2001.

[11] Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, and V. N. Venkatakrishnan. CANDID: Preventing SQL injection attacks using dynamic candidate evaluations. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 12–24. ACM Press, 2007.

[12] Vincent Berk, Annarita Giani, and George Cybenko. Covert channel detection using process query systems. In *Proceedings of FLOCON 2005*, May 2005.

[13] Richard E. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, 18(4):460–473, July 1972.

[14] Bluetooth. Bluetooth from Wikipedia. `http://en.wikipedia.org/wiki/Bluetooth`, [Accessed: January 20, 2010].

[15] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. `http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf`, Augest 2011.

[16] Kevin Borders, Xin Zhao, and Atul Prakash. Siren: Catching evasive malware (short paper). In *Proceedings of 27th IEEE Symposium on Security and Privacy (S&P)*, pages 78–85, Oakland, CA, May 2006. IEEE Press.

[17] Paul Bourke. Calculating the area and centroid of a polygon. [Accessed: September 10, 2011].

[18] Stephen W. Boyd and Angelos D. Keromytis. SQLrand: Preventing SQL injection attacks. In *Proceedings of the 2nd Applied Cryptography and Network Security Conference (ACNS)*, pages 292–302, 2004.

[19] Claus Brabrand, Anders Møller, Mikkel Ricky, and Michael I Schwartzbach. Powerforms: Declarative client-side form field validation. In *Proceedings of the 9th International World Wide Web Conference (WWW)*, pages 205–214, May 2000.

[20] Gregory T. Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti. Using parse tree validation to prevent SQL injection attacks. In *Proceedings of the International Workshop on Software Engineering and Middleware (SEM) at Joint FSE and ESEC*, pages 106–113, 2005.

[21] Serdar Cabuk. IP covert timing channels: design and detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 178–187. ACM Press, 2004.

[22] Serdar Cabuk. *Network covert channels: design, analysis, detection, and elimination*. PhD thesis, Purdue University, West Lafayette, Indiana, December 2006.

[23] Cloud Computing. Cloud computing from Wikipedia. `http://en.wikipedia.org/wiki/Cloud_computing`, [Accessed: January 20, 2010].

[24] W Cook and S Rai. Safe query objects: Statically typed objects as remotely executable queries. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 97–106. IEEE Press, 2005.

[25] Gregory W. Corder and Dale I. Foreman. *Nonparametric statistics for non-statisticians: a step-by-step approach*. Wiley Press, 1st edition edition, 2009.

[26] Elena Deza and Michel Marie Deza. *Encyclopedia of distances*, page 94. Springer, August 2009.

[27] Calculate Distance. Calculate distance, Bearing and more between latitude/longitude points. http://www.movable-type.co.uk/scripts/latlong.html, [Accessed: July 1, 2009].

[28] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[29] Amazon EC2. Amazon Elastic Compute Cloud (amazon ec2). http://aws.amazon.com/ec2/, [Accessed: October 12, 2011].

[30] McAfee entercept database. McAfee entercept database edition. http://www.anidirect.com/products/intrusionprevention/ds_entercept_databaseedition.pdf, [Accessed: July 1, 2008].

[31] VMware ESX. VMware ESXi and ESX info center. http://www.vmware.com/products/vsphere/esxi-and-esx/index.html, [Accessed: July 1, 2009].

[32] Ethereal. The Ethereal project. http://www.ethereal.com, [Accessed: January 1, 2010].

[33] Xiaopeng Fan, Jiannong Cao, and Haixia Mao. A survey of mobile cloud computing. *ZTE Communications, Special Topic: Mobile Cloud Computing and Applications*, 9(1):4–8, March 2011 2010.

[34] Sevin Fide. *Architectural optimizations in multi-core processors*. VDM Verlag, first edition edition, November 2008.

[35] Gina Fisk, Mike Fisk, Christos Papadopoulos, and Josh Neil. Eliminating steganography in internet traffic with active wardens. In *Proceedings of the 5th International Workshop on Information Hiding (IH)*, pages 18–35. Springer-Verlag Berlin, Heidelberg, 2002.

[36] MySQL Forge. MySQL Proxy project from Wikipedia. http://forge.mysql.com/wiki/MySQL_Proxy, [Accessed: July 1, 2008].

[37] FreeBSD. FreeBSD Handbook. http://lab.etfto.gov.br/material/free_bsd/handbook/handbook/firewalls.html, [Accessed: July 18, 2009].

[38] Steven Gianvecchio and Haining Wang. Detecting covert timing channels: an entropy-based approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 211–230. ACM Press, 2007.

[39] Steven Gianvecchio, Haining Wang, Duminda Wijesekera, and Sushil Jajodia. Model-based covert timing channels: automated modeling and evasion. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Springer-Verlag Berlin, Heidelberg, 2008.

[40] John Giffin, Rachel Greenstadt, Peter Litwack, and Richard Tibbetts. Covert messaging through TCP timestamps. In *Proceedings of the Workshop on Privacy Enhancing Technologies*, pages 194–208, 2002.

[41] Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn, and Robert Richardson. CSI/FBI computer crime and security survey. `http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2006.pdf`, [Accessed: January 1, 2010].

[42] Carl Gould, Zhendong Su, and Premkumar Devanbu. JDBC Checker: A static analysis tool for SQL/JDBC applications. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 697–698. IEEE Press, 2004.

[43] GPS. Global positioning system from Wikipedia. `http://en.wikipedia.org/wiki/Global_Positioning_System`, [Accessed: January 20, 2010].

[44] Gray-world. The Gray-world Team. `http://gray-world.net/projects.shtml`, [Accessed: July 1, 2009].

[45] GreenSQL. GreenSQL project. `http://www.greensql.net/`, [Accessed: July 1, 2008].

[46] Alfred Haar. Application of the discrete wavelet transform to the monitoring of tool failure in end milling using the spindle motor current. *International Journal of Advanced Manufacturing Technology*, 69(3):331–371, 1910.

[47] William G. J. Halfond and Alessandro Orso. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2006.

[48] William G. J. Halfond, Alessandro Orso, and Panagiotis Manolios. Using positive tainting and syntax-aware evaluation to counter SQL-injection attacks. In *Proceedings of 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 175–185. ACM Press, 2006.

[49] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. A classification of SQL injection attacks and countermeasures. In *Proceedings of the International Symposium on Secure Software Engineering (SEEE)*, March 2006.

[50] Myles Hollander and Douglas A. Wolfe. *Nonparametric statistical methods*. Wiley Interscience, second edition edition, 1999.

[51] Albert S. Huang and Larry Rudolph. *Bluetooth essentials for programmers*. Cambridge University Press, first edition edition, September 2007.

[52] Timekeeping in VMware. Timekeeping in Vmware Virtual Machines. `http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf`, [Accessed: January 1, 2011].

[53] Kavado Inc. InterDo version 3.0, [Accessed: July 1, 2008].

[54] Sanctum Inc. AppShield version 4.0 whitepaper. `http://www.sanctuminc.com`, [Accessed: July 1, 2008].

[55] Trent Jaeger, Reiner Sailer, and Yogesh Sreenivasan. Managing the risk of covert information flows in virtual machine systems. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 81–90. ACM Press, 2007.

[56] Wayne Jansen and Karen Scarfone. *Guidelines on cell phone and PDA security*. Addison-Wesley Professional, first edition edition, February 2005.

[57] JavaCC. JavaCC project. `https://javacc.dev.java.net/`, [Accessed: July 1, 2008].

[58] Jing Jin and Xinyuan Wang. On the effectiveness of low-latency anonymous network in the presence of timing attack. In *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 429–438. IEEE Press, 2009.

[59] JJTree. JJTree project. `https://javacc.dev.java.net/doc/JJTree.html`, [Accessed: July 1, 2008].

[60] Martin John and Christian Beyerlein. SMask: Preventing injection attacks in web applications by approximating automatic data/code separation. In *Proceedings of the ACM Symposium on Applied Computing, Seoul, Korea*, pages 284–291. ACM Press, March 2007.

[61] Jaeyeon Jung, Anmol Sheth, Ben Greenstein, David Wetherall, Gabriel Maganis, and Tadayoshi Kohno. Privacy Oracle: A system for finding application leaks with black box differential testing. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, pages 279–288. ACM Press, 2008.

[62] Myong H. Kang, Ira S. Moskowitz, and Stanley Chincheck. The pump: A decade of covert fun. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, pages 5–9. Applied Computer Security Associates, 2005.

[63] Myong H. Kang, Ira S. Moskowitz, and Daniel C. Lee. A network version of the pump. In *Proceedings of the 16th IEEE Symposium on Research in Security and Privacy (S&P)*, pages 144–154. IEEE Press, 1995.

[64] David B. Kirk and Wen mei W. Hwu. *Programming massively parallel processors: A hands-on approach*. Morgan Kaufmann, first edition edition, February 2010.

[65] Boris Kopf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 286–296. ACM Press, 2007.

[66] Solomon Kullback. *Information theory and statistics*. Dover Publications, July 1997.

[67] Solomon Kullback and Richard Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[68] Monica S. Lam, John Whaley, V. Benjamin Livshits, Michael C. Martin, Dzintars Avots, Michael Carbin, and Christopher Unkel. Context-sensitive program analysis as database queries. In *Proceedings of Principles of Database Systems (PODS)*, page 12, 2005.

[69] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery Journal*, 15(2):107–144, 2007.

[70] Yali Liu, Cherita L. Corbett, Ken Chiang, Rennie Archibald, Biswanath Mukherjee, and Dipak Ghosal. SIDD: A framework for detecting sensitive data exfiltration by an insider attack. In *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, 2009.

[71] Yali Liu, Dipak Ghosal, Frederik Armknecht, Ahmad-Reza Sadeghi, Steffen Schulz, and Stefan Katzenbeisser. Hide and seek in time: robust covert timing channels. In *Proceedings of the 14th European conference on Research in computer security (ESORICS)*, pages 120–135. Springer-Verlag Berlin, Heidelberg, 2009.

[72] V. Benjamin Livshits and Monica S. Lam. Finding security vulnerabilities in Java applications with static analysis. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, pages 271–286. USENIX Association, 2005.

[73] David Mark, Jack Nutting, and Jeff LaMarche. *Beginning iPhone 4 development: Exploring the iOS SDK*. Apress, first edition edition, January 2011.

[74] Michael Martin, Benjamin Livshits, and Monica S. Lam. Finding application errors using PQL: a program query language. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications*, pages 365–383. ACM Press, 2005.

[75] Russell A. McClure and Ingolf H. Krüger. SQL DOM: Compile time checking of dynamic SQL statements. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 88–96. IEEE Press, 2005.

[76] Michael Mesnier, Matthew Wachs, Brandon Salmon, and Gregory R. Ganger. Relative fitness models for storage. *SIGMETRICS Performance Evaluation Review*, 33(4):23–28, 2009.

[77] Windows Mobile. Windows mobile developer center. `http://msdn.microsoft.com/en-us/windowsmobile/bb264318`, [Accessed: January 20, 2010].

[78] MySQL. MySQL, the open source database. `http://www.mysql.com/`, [Accessed: July 1, 2008].

[79] Giorgio Nebuloni. Virtual machines to outnumber physical in 2009. [Accessed: October 10, 2011].

[80] James Newsome, David Brumley, Jason Franklin, and Dawn Song. Replayer: automatic protocol replay by binary analysis. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 311 – 321. ACM Press, 2006.

[81] Anh Nguyen-tuong, Salvatore Guarnieri, Doug Greene, and David Evans. Automatically hardening web applications using precise tainting. In *Proceedings of 20th IFIP International Information Security Conference (IISC)*, pages 372–382, 2005.

[82] NTOP. The NTOP project. `http://www.ntop.org/`, [Accessed: January 1, 2010].

[83] United States Department of Defense. Trusted computer system evaluation criteria. `http://csrc.nist.gov/publications/history/dod85.pdf`, [Accessed: November 1, 2010].

[84] United States Government Accountability Office. Cyberspace: United States faces challenges in addressing global cybersecurity and governance. Report to Congressional Requesters GAO-10-606, July 2010.

[85] Keisuke Okamura and Yoshihiro Oyama. Load-based covert channels between Xen virtual machines. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC)*, pages 173–180. ACM Press, 2010.

[86] Pai Peng, Peng Ning, and Douglas S. Reeves. On the secrecy of timing-based active watermarking trace-back techniques. In *Proceedings of the 27th IEEE Symposium on Security and Privacy (S&P)*, pages 335–349. IEEE Press, 2006.

[87] Alexandre Petrenko, Re Petrenko, Roland Groz, and Sergiy Boroday. Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering*, 30(1):29–42, 2004.

[88] Tadeusz Pietraszek and Chris Vanden Berghe. Defending against injection attacks through context-sensitive string evaluation. In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 124–145. Springer-Verlag Berlin, Heidelberg, 2005.

[89] Bluetooth Power. Bluetooth power classes. `http://bluetoothinsight.blogspot.com/2008/01/bluetooth-power-classes.html`, [Accessed: January 20, 2010].

[90] Tor Project. Tor, anonymity online. `http://www.torproject.org/`, [Accessed: January 1, 2011].

[91] W. Purczynski. GNU fi leutils - recursive directory removal race conition. Bugtraq-fi leutils mailing list, [Accessed: May 21, 2009].

[92] Daniel Ramsbrock, Xinyuan Wang, and Xuxian Jiang. A first step towards live botmaster traceback. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 59–77. Springer-Verlag Berlin, Heidelberg, 2008.

[93] RDTSC. Using the RDTSC instruction for performance monitoring. `http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf`, [Accessed: January 1, 2011].

[94] Morgan Stanley Research. Internet Trends. `http://www.morganstanley.com/institutional/techresearch/pdfs/Internet_Trends_041210.pdf`, [Accessed: January 20, 2011].

[95] Ryan Riley, Xuxian Jiang, and Dongyan Xu. An architectural approach to preventing code injection attacks. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks(DSN)*, pages 30–40. IEEE Press, 2007.

[96] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 199–212. ACM Press, 2008.

[97] Malek Ben Salem, Shlomo Hershkop, and Salvatore J. Stolfo. A survey of insider attack detection research. In *Insider Attack and Cyber Security: Beyond the Hacker*, pages 1–19. Springer, 2008.

[98] Ivo Salmre. *Writing mobile code: Essential software engineering for building mobile applications*. Addison-Wesley Professional, first edition edition, February 2005.

[99] Sanasecurity.com. SANA security's primary response. `http://www.sanasecurity.com/common/files/PR3.0_datasheet.pdf`, [Accessed: July 1, 2008].

[100] David Scott and Richard Sharp. Abstracting application-level web security. In *Proceedings of the 11th International World Wide Web Conference (WWW)*, pages 396–407, 2002.

[101] David Scott and Richard Sharp. Specifying and enforcing application-level web security policies. *IEEE Transactions in Knowledge and Data Engineering (TKDE)*, 15(4):771–783, 2003.

[102] SEE. SQLite Encryption Extension (SEE). `http://www.hwaci.com/sw/sqlite/see.html`, [Accessed: January 20, 2010].

[103] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: A new approach for detecting network intrusions. In *Proceedings of the 9th ACM Computer and Communication Security Conference (CCS)*, pages 265–274. ACM Press, May 2002.

[104] Gaurav Shah, Andres Molina, and Matt Blaze. Keyboards and covert channels. In *Proceedings of the 15th USENIX Security Symposium (USENIX Security)*, pages 59–75. USENIX Association, 2006.

[105] Smartphone. Smartphone from Wikipedia. `http://en.wikipedia.org/wiki/Smart_phone`, [Accessed: January 20, 2010].

[106] Ronald William Smith and George Scott Knight. Predictable design of network-based covert communication systems. In *Proceedings of the 29th IEEE Symposium on Research in Security and Privacy (S&P)*, pages 311–321. IEEE Press, 2008.

[107] Virtual Machine Snapshots. Hyper-V Virtual Machine Snapshots: FAQ. `http://technet.microsoft.com/en-us/library/dd560637(WS.10).asp`, [Accessed: January 1, 2010].

[108] Divert Sockets. Divert sockets mini-HOWTO. `http://www.faqs.org/docs/Linux-mini/Divert-Sockets-mini-HOWTO.html`, [Accessed: July 1, 2009].

[109] sparxsystems.com.au. Enterprise architect. `http://www.sparxsystems.com.au/`, [Accessed: Augest 16, 2010].

[110] Bluetooth Specification. Bluetooth specification documents. `http://en.wikipedia.org/wiki/Bluetooth`, [Accessed: January 20, 2010].

[111] SQLite. SQLite Version 3.7.8. `http://www.sqlite.org/`, [Accessed: January 20, 2010].

[112] NMEA Standard. NMEA 0183 standard, version 4.00. http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp, November 2008.

[113] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. In *Proceedings of the Symposium on Principles of Programming Languages (POPL)*, pages 372–382. ACM Press, 2006.

[114] Sualeh Fatehi. Schemacrawler version: 8.8. http://schemacrawler.sourceforge.net/, [Accessed: September 6, 2011].

[115] Tethereal. Tethereal - Dump and analyze network traffic. http://www.ethereal.com/docs/man-pages/tethereal.1.html, [Accessed: January 1, 2010].

[116] T.O.Fundation. Top ten most critical web application vulnerabilities. http://www.owasp.org/documentation/topten.html, [Accessed: July 1, 2008].

[117] Mitch Tulloch. *Understanding Microsoft Virtualization R2 Solutions (e-book)*. Microsoft Press, second edition edition, 2010.

[118] F. Valeur, D. Mutz, and G. Vigna. A learning-based approach to the detection of SQL attack. In *Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 123–140, 2005.

[119] Vsftpd. Vsftpd version 2.3.4. https://security.appspot.com/vsftpd.html, [Accessed: May 30, 2011].

[120] Mythili Vutukuru, Hari Balakrishnan, and Vern Paxson. Efficient and robust TCP stream normalization. In *Proceedings of the 29th IEEE Symposium on Security and Privacy (S&P)*. IEEE Press, May 2008.

[121] Hao Wang, Somesh Jha, and Vinod Ganapathy. NetSpy: Automatic generation of spyware signatures for NIDS. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC)*, pages 99–108, Miami Beach, FL, 2006. Applied Computer Security Associates.

[122] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. Tracking anonymous peer-to-peer VoIP calls on the Internet. In *Proceedings of the 12th ACM Conference on Computer Communications Security (CCS)*, pages 81–91. ACM Press, 2005.

[123] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. A network version of the pump. In *Proceedings of the 28th IEEE Symposium on Security and Privacy (S&P)*, pages 116–130. IEEE Press, 2007.

[124] Xinyuan Wang and Douglas S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 20–29, 2003.

[125] Zhenghong Wang and Ruby B. Lee. Covert and side channels due to processor architecture. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC)*, pages 473–482, Miami Beach, FL, 2006. Applied Computer Security Associates.

[126] Wayne Jansen and Timothy Granc. Guidelines on security and privacy in public cloud computing. http://csrc.nist.gov/publications/drafts/800-144/Draft-SP-800-144_cloud-computing.pdf, [Accessed: January 1, 2011].

[127] B. L. Welch. The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34((1-2)):28–35, 1947.

[128] Yao Wen Huang, Fang Yu, Christian Hang, Chung hung Tsai, D. T. Lee, and Sy Yen Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th conference on World Wide Web (WWW)*, pages 40–52. ACM Press, 2004.

[129] Wget. Wget project. http://ftp.gnu.org/gnu/wget/, [Accessed: July 1, 2008].

[130] Andy Wigley, Daniel Moth, and Peter Foot. *Microsoft mobile development handbook*. Microsoft Press, first edition edition, 2007.

[131] Winsock. Winsock reference. http://msdn.microsoft.com/en-us/library/ms741416.aspx, [Accessed: January 20, 2010].

[132] Yichen Xie and Alex Aiken. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the 15th USENIX Security Symposium (USENIX Security)*, pages 271–286. USENIX Association, 2006.

[133] Wei Xu, Sandeep Bhatkar, and R. Sekar. Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In *Proceedings of USENIX Security Symposium (USENIX Security)*, pages 121–136. USENIX Association, 2006.

[134] Danfeng Zhang, Aslan Askarov, and Andrew Myers. Predictive mitigation of timing channels in interactive systems. In *Proceedings of the 18th ACM Computer and Communication Security Conference (CCS)*, pages 563–574. ACM Press, October 2011.

# Curriculum Vitae

Anyi Liu has been a PhD student in the Volgenau School of Engineering at George Mason University (GMU) since 2003. His major is Information Technology. Before joining GMU, he received the BE in Computer Science in 1997 and the MS degree in Computer Science in 2001, both from Dalian University of Technology, China. His research focused on Information Security and Privacy.