TOWARDS MORE TRUSTWORTHY TRUST-BASED SYSTEMS FOR ANONYMITY AND WEB SECURITY

ZHOU PENG

Ph.D

The Hong Kong Polytechnic University

2014

The Hong Kong Polytechnic University
Department of Computing

Towards More Trustworthy Trust-based Systems for Anonymity and Web Security

Zhou Peng

A thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

January 2014

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

Signed: _____

Date: 2014-06-18

Abstract

In today's Internet, trust has been widely used to design anonymity and security enhanced systems. Some of these trust-based systems have been successfully deployed in the Internet for a long time and benefit a large population of Internet users. In particular, trust-based onion routing network is a representative example for the use of trust in protecting anonymity. As one of the most popular onion routing systems, Tor serves more than 3 millions of Internet users. It hides Internet users' identities behind a circuit of selected onion routers but runs a high risk of being compromised by attackers who employ malicious onion routers to launch correlation-like attacks. Without an effective trust model, it is very difficult for Internet users to evade attackers' routers when establishing onion circuits. As a result, recent research proposes trust-based onion routing to thwart the correlation-like attacks. Using a priori trust that users have readily assigned to routers' owners, attackers' routers are likely to be identified and excluded from users' onion circuits.

As an example to demonstrate the effectiveness and popularity of trust in protecting security, we study the public key infrastructure (PKI for short) which has been successfully deployed in the web for more than two decades. This infrastructure employs a well-known certification based trust model for website authentication. Based on this trust model, modern browsers trust a group of trust anchors (also known as root certificate authorities or CAs for short) in advance, and authenticate remote websites by checking whether the site certificate is signed by one of the pre-trusted trust anchors.

Although trust-based systems are widely used for securing anonymous communications and web services, recent studies reveal that the use of trust could incur new problems. For example, despite that trust-based onion routing successfully defeats correlation-like attacks by using a priori trust among users, the use of trust for onion routing still suffers from two challenging problems due to the inherent weakness of trust. One is the biased trust distributions among users, and the other is how to verify the correctness of trust one person assigns to the other. The biased distribution will reduce the entropy (i.e., anonymity) of the whole routing system and hence induce a new inference attack, whereas the incorrect trust could render trust-based onion routing ineffective in protecting anonymity. On the other hand, the trust-based systems

that secure web services are also vulnerable due to the inadequate use of trust. For instance, the certification based trust model can be subverted globally if a single trust anchor is compromised. The root cause is that the trust model treats every trust anchor equally and accepts site certificate issued by any one of the pre-trusted trust anchors. This serious design flaw has been exploited to successfully hijack around $300,000$ Gmail users. As a result, these problems largely limit the effectiveness of even the state-of-the-art trust-based systems.

Motivated by these challenges, the overall objective of this thesis is to make the aforementioned trusted-based systems more trustworthy. We present our research results in three parts. First, we propose the use of trust degree and global trust to address the biased trust distribution problem. Our trust degree based routing algorithm encourages users to select the onion routers that are trusted by more other users with a higher probability, hence reducing the possibility that the user's identity can be inferred by attackers. The global trust, on the other hand, is designed to guide users to discover and trust more honest routers, thus mitigating the bias of trust distributions. We also aggregate group trust from mutual friends to verify the correctness of users' trust assignments. The group trust is designed based on a key insight: the trust from a group of honest people is more likely to be correct than the trust from a single honest person. We design a novel trust graph based onion routing algorithm that offers these new trust features, and show that this algorithm is more effective than existing trust-based onion routing systems.

Second, we use an active approach to correct the flaw in the certification based trust model. Our approach is designed to maximize the protection against man-in-the-middle attacks by actively exhausting available trust anchors and exploiting Internet path diversity. Equipped with our approach, compromising a single trust anchor cannot compromise the entire trust model. Instead, subverting the entire trust model requires compromising a large number of trust anchors and hijacking nearly all the Internet paths to victim websites. Our approach consists of four distinct countermeasures, each of which has a unique tradeoff between the ease of deployment and the capability of defending against various man-in-the-middle attacks. These new countermeasures overcome the weaknesses of the existing countermeasures, which can neither defend first-time authentication nor resist man-in-the-middle attacks with two compromised trust anchors. We confirm the effectiveness of our approach using a real-world certificate data set and Internet experiments.

Third, relating to web security, we also survey the landscape of file download vulnerabilities across different domains and countries, and discover the weak protection against this vulnerability in many web systems today. Our further investigation discloses the root causes of this weak protection: existing protection systems against file download vulnerability rely on either ad hoc user input sanitization mechanisms (whose implementations are error-prone) or directory based permission control (that suffers from undesirable flexibility). Based on this observation, we propose FileGuard, a new protection system that secures file download in the script engine

layer. The basic idea is to isolate the web files from the rest of local filesystem through the embedding of dedicated ownership information into extended file attributes. Using this ownership information, a reliable and fine-grained access control can be performed to block illegal file downloads. FileGuard can mitigate the impact of erroneous implementations, because it provides a unified protection regardless of specific file download logic and achieves desirable flexibility due to per-file ownership statement. We have implemented a proof-of-concept prototype of FileGuard by modifying the source code of PHP5 script engine and have confirmed that FileGuard can provide more reliable protections.

Publications

The following published papers are the partial outputs of my PhD studies in HK PolyU.

Journal Articles

J3: **Peng Zhou**, Xiapu Luo and Rocky K. C. Chang, "Inference Attacks Against Trust-based Onion Routing: Trust Degree to the Rescue", **Computers & Security**, Elsevier, vol. 57, pages: 3522-3544, 2013.

J2: **Peng Zhou**, Xiapu Luo, Ang Chen and Rocky K. C. Chang, "SGor: Trust Graph based Onion Routing", **Computer Networks**, Elsevier, vol. 39, pages: 431-446, 2013.

J1: Xiapu Luo, Edmond W. W. Chan, **Peng Zhou** and Rocky K. C. Chang, "Robust Network Covert Communications Based on TCP and Enumerative Combinatorics", **IEEE Transactions on Dependable and Secure Computing (TDSC)**, vol. 9, pages: 890-902, 2012.

Conference Papers

C4: **Peng Zhou**, Xiapu Luo and Rocky K. C. Chang, "More Anonymity Through Trust Degree in Trust-based Onion Routing", **ICST/ACM SecureComm**, pages: 273-291, 2012.

C3: Xiapu Luo, **Peng Zhou**, Junjie Zhang, Roberto Perdisci, Wenke Lee, and Rocky K.C. Chang, "Exposing Invisible Timing-based Traffic Watermarks with BACKLIT", **ACSAC**, pages: 197-206, 2011.

C2: Xiapu Luo, **Peng Zhou**, Edmond W. W. Chan, Rocky K. C. Chang, and Wenke Lee, "A Combinatorial Approach to Network Covert Communications with Applications in Web Leaks", **IEEE/IFIP DSN**, pages: 474-485, 2011.

C1: Xiapu Luo, **Peng Zhou**, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci, "HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows", **ISOC/USENIX NDSS**, pages:381-400, 2011.

Acknowledgements

In the past four years, my research career is filled with joyful and “painful” experiences. During this period, a lot of people gave me selfless help, useful suggestions, critical comments and encouraging feedbacks when I encountered problems in my research and daily life. Without their help and support, I can hardly imagine that I can complete my thesis on time. To this end, I must express my sincere gratitude to them.

At first, I should thank my supervisor, Prof. Rocky K. C. Chang. He spends a large amount of his resources to help me grow from a fresh student to an independent researcher. He opens the door of network science to me, and teaches me how to write a high quality research paper and how to present research topics in a scientific manner. I think I will continuously benefit from his guidance in my future research. Moreover, I would like to express my thanks to my labmates, Xiapu Luo, Edmond W. W. Chan, Waiting W. T. Fok, Ricky K. P. Mok, Weichao Li, Ang Chen and Daoyuan Wu. Many of my research ideas are inspired from the discussion with them. Last, but the most important, I must appreciate my parents. Without their love and support, I cannot focus on my research for such a long time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, the Internet is becoming an essential part of human's daily life and work. It provides a very efficient and cost-effective information channel which can significantly reduce communication distance among human beings across the globe. Due to this advantage, more and more human beings participate to the Internet and rely on online services for their daily activities. For example, people can communicate with their friends and colleagues through e-mails or instant messages in a very quick manner. They can also share their status and any other information to their unfamiliar audiences by means of (micro)blogs, forums or online social networks. By using online shopping and banking services, people can purchase life necessities and almost other goods they demand without going out to traditional stores and malls. Moreover, e-health can help optimize the use of medical resources while e-government can facilitate public resource usage. Benefiting from these attractive Internet services, the population of Internet users expands rapidly. As reported in a very recent statistical survey [3], the number of Internet users exceeds 2.4 billions in June 2012 (i.e., more than 34.3% of world total population).

Along with the increasing popularity of online activities, Internet privacy and security become hot-button research topics. For example, to ensure Internet users visiting the Internet without revealing their addresses and locations, a large number of anonymous communication techniques [4–9] have been designed and deployed. Among these techniques, onion routing is perhaps the most popular one. Its prototype system, Tor [9], has been deployed in the Internet for more than ten years and already used by thousands of millions of Internet users. Moreover, to prevent users from being attacked by impersonated websites, a certification based trust model [10] has been applied on the web. This model helps Internet users certify the authenticity of Internet websites by checking whether these sites can provide domain-bounded site certificates signed by a pre-trusted authority (i.s., a trust anchor). Internet websites, on the other hand, are also threatened by web vulnerabilities. By exploiting these vulnerabilities, malicious Internet users could attack,

or even completely compromise, vulnerable sites. For this reason, the design of vulnerability-free web applications (i.s., web applications run in a trustworthy snadbox) is extremely valuable although it is not a trivial task.

To ensure Internet privacy and security, trust-based systems have attracted considerable attentions in recent studies. Although these systems have shown success experiences in the use of trust to address challenging privacy and security issues, they also incur new problems. To this end, we conduct an in-depth research to investigate state-of-the-art trust-based systems in this thesis. Our research aims to analyze the key problems of existing trust-based systems and attempts to propose solutions to solve those new problems.

1.1 Three Trust-based Systems for Anonymity and Web Services

In our research, we are mainly focusing on the study of three trust-based systems: trust-based onion routing system [11–15], certification based trust model [10, 16, 17] and file download protection system [18]. These three systems represent state-of-the-art trust-based methods for securing anonymity and web security in their respective domains. Our findings and results can also shed light to the in-depth research for other trust-based systems.

1.1.1 Trust-based Onion Routing

Onion routing [7–9] is perhaps one of the most popular low-latency anonymous communication systems running in the Internet. It wraps users' traffic using successive layers of encryption along a circuit of selected onion routers, hence hiding the initial users behind the onion circuits. However, recent research has found that existing onion routing is subject to various correlation-like attacks [19–27]. These attacks can completely deanonymize users' onion circuits in case when users unconsciously select attackers' onion routers in their circuits. For example, attackers can passively analyze traffic characteristics and patterns [19–22, 25, 26] or actively embed traffic watermarks [19, 23, 24, 27] to correlate the onion routers controlled by them. By this mean, attackers can largely reduce the anonymity protected by onion routing systems, especially when a large fraction of the network is compromised. Figure 1.1 illustrates a typical example of correlation-like attacks in onion routing network. As can be seen, if the first and the last onion router in a user's circuit are both controlled by an attacker, the user and the web server this user visits can be correlated and revealed directly.

To defeat correlation-like attacks, recent research proposes the use of trust for excluding attackers' routers. For example, Krishna et al. [11] restrict users to only select onion routers from their 1- and 2-hop friends in an online social network. Drac [13] and Pisces [15] perform random

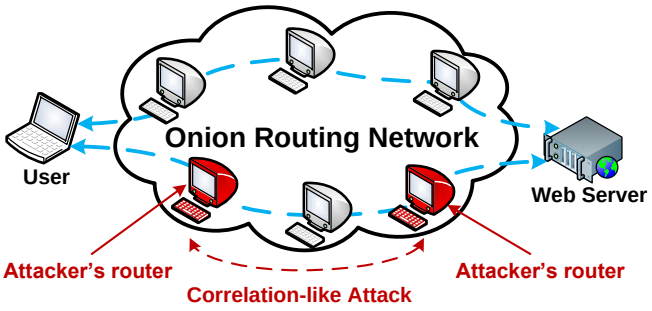


FIGURE 1.1: An example of correlation-like attacks in onion routing network.

walks through social links on top of an online social network to discover honest routers. Each directed social link represents a trust that one person assigns to another person. However, these three studies cannot present a theoretical analysis for attack models and optimal trust-based routing algorithms. To fill in this gap, Johnson et al. provide a general model for trust-based onion routing [12]. This model includes completely theoretical attack models and discusses corresponding optimized trust-based onion routing algorithms in the presence of these attack models. Moreover, Johnson et al. design an enhanced model to analyze trust-based onion routing in practical scenarios [14]. This practical model considers that different users have different distributions of local trust among onion routers, since users are usually attacked by different attackers in the wild. This practical model also studies how the accuracy of trust assignments affects the effectiveness of trust-based onion routing. These trust-based onion routing algorithms have successfully demonstrated the effectiveness of trust in protecting anonymity. They can enhance the protection of onion routing even if a significant fraction of the network is compromised.

1.1.2 Certification based Trust Model

Certification based trust model is designed to employ browser-trusted certificates for website authentication [10]. As an essential part of public key infrastructure deployed in the Internet, this trust model relies on a group of trust anchors (also known as certificate authorities or CAs for short) that are pre-installed in browsers (e.g., Chrome, Firefox or Internet Explorer), and authenticates remote websites on behalf of Internet users by means of checking whether the website certificates are issued from one of pre-installed trust anchors. If yes, the website passes authentication. Otherwise, it fails. This model exploits trust's delegatability to help Internet users thwart impersonation attacks [28] and man-in-the-middle attacks [29]. Figure 1.2 shows an example of website authentication scenario using the certification based trust model. In this example, Internet users pass the authentication of a remote web server since the server's site certificate is signed by one of trust anchors.

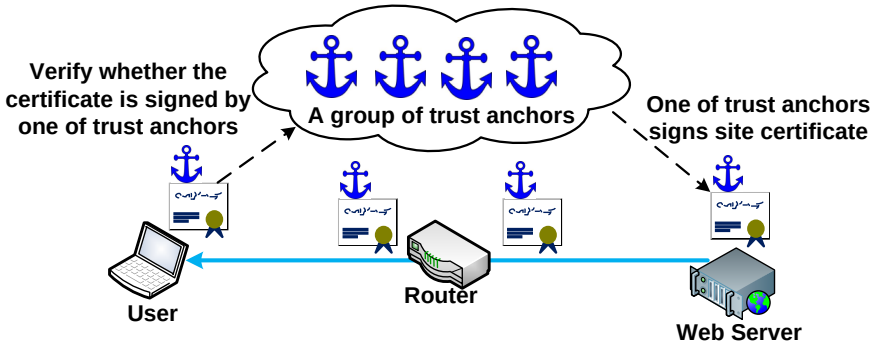


FIGURE 1.2: An example of website authentication using the certification based trust model.

1.1.3 File Download Protection

In today's Internet, web is becoming a readable and writable medium for sharing and managing files that are stored across globally distributed web servers. Along with this trend, a large number of web applications provide file management services (such as file upload, download and online editing etc.). These services usually implement a spectrum of file operation functions by means of server-side scripts, and releases these functions to the public in the form of dynamic URLs. Among these functions, file download is a very basic primitive that is responsible for publishing and delivering files to remote web users across the Internet. However, since server-side script engine inherits web server permissions to access local filesystem, it can read many sensitive local files by default. As a result, without additional protections, file download scripts can be exploited to download any sensitive files the web server has read permission to. We call these vulnerable scripts *file download vulnerabilities*.

Figure 1.3 demonstrates a typical example of File download vulnerability on the web. By exploiting this kind of vulnerabilities, malicious Internet users could illegally download sensitive files (such as source code of server-side scripts, web configuration files and critical system files) stored on online services' local filesystem. The leakage of these sensitive local files could result in disastrous consequences. For example, if web configuration files are downloaded, malicious users could launch database intrusion as a consequence. Even worse, if password files (e.g., `/etc/passwd` or `/etc/shadow` in Linux) are leaked, system intrusion becomes possible.

To prevent these dangerous file download vulnerabilities, user input sanitization mechanisms can be implemented in server-side scripts [30, 31] and directory based permission control can be configured in web server or server-side script engine. These protection methods ensure that only trustworthy file read operations can be performed by file download scripts, hence preventing sensitive files from being leaked through these scripts.

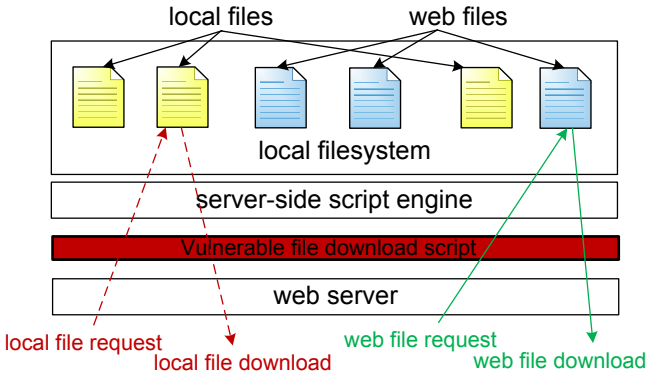


FIGURE 1.3: An example of vulnerable file download script.

1.2 Problems

Although the aforementioned three trust-based systems have successfully demonstrated the effectiveness and advantages in protecting anonymity and web services, they still face challenges due to the new problems caused by the use of trust. More importantly, the use of trust for protecting anonymity could reduce the entropy of the whole anonymity network because the trust distributions are usually biased, while existing trust-based protections for Internet security are either flawed or unreliable. We will elaborate on those new problems in the following subsections.

1.2.1 Biased Trust Distributions

The use of trust for onion routing has successfully demonstrated its effectiveness in excluding malicious routers from users' onion circuits and thus defeating correlation-like attacks. However, since normal users usually have knowledge only for a small group of onion routers and different users have knowledge for different small fractions of the network, the resulting trust distributions are usually biased. This kind of biased distributions can largely reduce the entropy of the whole onion routing network, hence leading to a new kind of attacks, called inference attacks [14]. More precisely, an attacker who knows a priori trust relationships has a high probability to guess the initial user of an onion circuit if this attacker can observe a router in targeted circuit (i.e., inference attack). Although prior research has proposed a downhill algorithm [14] to mitigate this kind of attacks, this algorithm is still based on local trust and cannot address the root cause: biased local trust distribution leading to inference attacks.

1.2.2 The Correctness of Trust

In contrast to biased trust distributions, the correctness of trust is also hardly to be guaranteed in trust-based onion routing. In particular, existing trust-based onion routing computes trust only according to users' own knowledge, but this knowledge could be inaccurate and incorrect [12, 14]. If an onion routing user wrongly trusts an attacker, trust-based onion routing has no effective ways to remove this incorrect trust assignment. Even worse, if trust-based onion routing relies on a large number of incorrect trusts for onion routing, it cannot provide better protection of anonymity than the onion routing without trust. To the best of our knowledge, no prior studies have proposed solutions to enhance the correctness of trust for onion routing. Instead, they only theoretically analyzed the impacts of the correctness and accuracy of trust on anonymity protection [14].

1.2.3 Disastrous Vulnerability in Certification based Trust Model

Although certification based trust model has been successfully deployed on the web for more than two decades, it has been found vulnerable to a disastrous flaw: the compromised of a single trust anchor can subvert the entire trust model globally. The root cause is that end users do not have the knowledge about which certificate authority is the legal issuer of which site certificate. As a result, the attackers who compromise one certificate authority (usually the weakest one) can issue faked site certificates which can be used to launch man-in-the-middle attacks. As shown in Figure 1.4, the man-in-the-middle attackers can replace the real site certificate with a faked one which is issued by a compromised trust anchor. Internet users cannot be aware of this replacement since the compromised trust anchor is also pre-trusted by browsers. This vulnerability has made severe impacts to the real world. For example, two real-world certificate authorities, Comodo [32] and DigiNotar [33], were found compromised in 2011. Attackers employed them to issue faked certificates for popular websites, and launched man-in-the-middle attacks to successfully hijack around $300,000$ Iranian's Google mail connections [34].

Patching this dangerous flaw is very challenging, because man-in-the-middle attacks with different vantage points and attacking patterns pose different levels of threats to the trust model. The state-of-the-art countermeasures do not make full use of trust the certification based trust model provides, hence probably becoming ineffective against man-in-the-middle variants. On the one hand, notary based solutions (such as Perspectives [35] and Crossbear [36]) deploy a third-party service to collect different copies of a suspicious site certificate through globally distributed hosts and alert a possible man-in-the-middle attack to end users if these copies are not the same. This countermeasure explores a portion of Internet path diversity to detect man-in-the-middle. However, the third-party service can be regarded as an additional trust anchor and the compromise of this trust anchor can also subvert the entire trust model. Even if the third-party cannot be

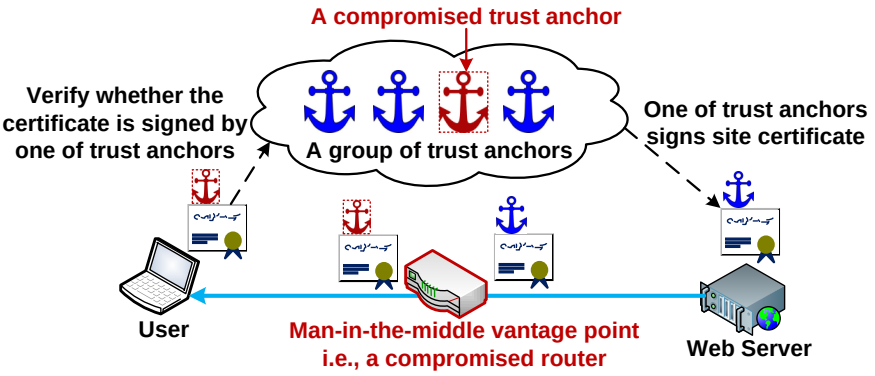


FIGURE 1.4: An example of man-in-the-middle attack against the certification based trust model.

compromised, this solution is still ineffective against the man-in-the-middle attacks whose vantage points are near to web servers [36]. On the other hand, some recent studies [37, 38] propose the use of pre-shared secrets (e.g., a password in [37] or an HTTP cookie in [38]) to enhance site authentication. Although this kind of countermeasures introduces new information from the design space outside trust model, it cannot establish trust on the first time authentication, which is a very important property the certification based trust model intends to protect.

1.2.4 Unreliable File Download Protection

Defenses against file download vulnerability have been existing for a long time. As a result, some may believe this is a solved problem. However, we find that existing defenses still have a number of practical drawbacks, hence making this vulnerability still prevalent in today's web. On the one hand, sanitizing sensitive user inputs before using these inputs as file names to access local filesystem is an effective defense. However, this defense requires customized implementations for different file download logics and therefore becomes easy to introduce insufficient or erroneous protections. Moreover, since an enterprise web server often consists of legacy scripts that are deployed by different web administrators, at different times, and for different purposes, it is very difficult to assure all these scripts are not vulnerable. On the other hand, directory based permission control is another kind of mainstream defenses. Although this defense can restrict web server permission to some specified directories and forbids any file read operations outside, it is not flexible and falls victim to multiple file download scripts. For example, if the web server permission is specified to a web application's home directory, the local files such as server-side scripts and web configuration files inside this home directory cannot be protected. Even if the specified directories do not contain any local files, the files managed by one web application can also be illegitimately downloaded through other irrelevant file download scripts running in the same web server.

By sampling the landscape of file download vulnerabilities across different domains and countries, we confirm the weak protection of file download scripts in today's web. In particular, we have collected $19,137$ suspicious file download scripts by means of Google search engine, and successfully discovered $6,060$ (or 31.7% of our sample set) vulnerabilities. Among these vulnerabilities, we have found $5,807$ (or 30.3%) can download script source codes, $4,285$ (or 22.4%) can retrieve web configurations, and $1,987$ (or 10.4%) can expose critical system files. Our further experiments confirm a high probability of disastrous consequences that our discovered vulnerabilities can induce (i.e., 12.8% of our discovered vulnerabilities can result in system intrusion and 17.8% can lead to database intrusion). Our results motivate us to reconsider new defense to enhance the protection of file download scripts.

1.3 Major Contributions

In this thesis, our contributions primarily lie in an comprehensive analysis of fundamental challenges and tradeoffs in existing trust-based systems which employ trust to protect anonymity and web services. Our major contributions also cover the proposal of more trustworthy solutions to enhance these trust-based systems.

1.3.1 More Trustworthy Trust-based Onion Routing

In trust-based onion routing, biased trust distributions and incorrect trust assignments are two key limitations that hinder the effective use of trust for protecting anonymity. To mitigate these limitations, we design novel more trustworthy trust-based onion routing algorithms. These algorithms exploit new trust features from a trust graph to address biased trust distributions and incorrect trust assignments. More importantly, we make two major contributions as follows.

First, we uncover trust degree, an essential feature of routing anonymity that is effective in defeating inference attacks but has been overlooked in the design of existing trust-based onion routing. We conduct an isolated model based analysis to understand why the trust degree is effective and how it can be used to resist inference attacks. In particular, we present a model to exclusively reason about inference attacks in trust-based onion routing. This model isolates the anonymity compromised by inference attacks from other attacks (e.g., correlation-like attacks), and hence derives an exclusive design space that reveals trust degree as the key feature against inference attacks. To show the usefulness of our model, we design a new routing algorithm by taking into account of trust degree. Our algorithm can protect anonymity against inference attacks without sacrificing the capability against attackers' routers. To confirm the effectiveness of trust degree in defeating inference attacks under real-world settings, we compare trust-based

routing algorithms with and without considering trust degree through experiments based on real-world social networking datasets.

Second, we propose SGor, a trust graph based onion routing algorithm that mitigates the key limitations of trust in protecting anonymity. SGor is novel with three unique properties. First, SGor aggregates group trust from mutual friends to verify the correctness of users' trust assignments. Second, SGor employs an adaptive trust propagation algorithm to derive global trust from trust graph. The global trust removes the restriction of users' local knowledge and defeats inference attacks by guiding users to discover and trust more honest routers (i.e., reducing the bias of trust distribution). Third, SGor is designed to operate in a fully decentralized manner. This decentralized design mitigates the leakage of a priori trust relationships. We evaluate SGor with simulation-based experiments using several real-world social trust datasets. The experimental results confirm that SGor can mitigate key limitations in the use of trust for protecting anonymity but introduces only a few overheads.

1.3.2 More Trustworthy Certification based Trust Model

To protect against man-in-the-middle attacks with a single compromised trust anchor, we propose an active approach to harden the flawed certification based trust model. The idea is to collect a priori trust from more trust anchors and Internet path diversity instead of just one trust anchor, hence making the trust model more trustworthy. Our approach presents the first attempt to maximize the protection against man-in-the-middle attacks not beyond the certification based trust model. Its novelty primarily lies in three aspects. First, our approach is proposed to address a critically important problem discovered in the use of certification based trust model for today's web: the compromise of a single trust anchor can subvert the entire trust model globally. This problem poses a serious security threat and has caused a number of disastrous consequences in today's web [32–34]. Second, compared with prior countermeasures, such as notary-based solutions [35, 36] and pre-shared secrets methods [37, 38], our approach can defeat all kinds of man-in-the-middle attack variants and does not remove the capability against man-in-the-middle attacks during the first time authentication. Moreover, our approach is the first solution that can maximize the use of trust for website authentication purposes. Third, although the idea of defending against man-in-the-middle attacks through the collection of a priori trust from more trust anchors and Internet path diversity is easy to understand, rendering it practice still faces many challenges. As a result, we have designed four different countermeasures to adapt our approach to different attacking scenarios and available resources. Each countermeasure has a unique tradeoff between the difficulty of deployment and the capability against man-in-the-middle variants. For example, if a web server refuses to adopt server-side countermeasure to enhance the protection for the users who visit this server, the Internet users can only deploy client-side countermeasures to protect themselves.

1.3.3 More Trustworthy File Download Protection

Motivated by the high volume of vulnerable file download scripts we have discovered in today's web, we propose FileGuard, a more trustworthy defense that protects local filesystem against vulnerable file download scripts through the embedding of dedicated ownership information into extended file attributes. Compared with user input sanitization mechanisms, FileGuard can be instrumented in script engine layer and thus provide unified protections regardless of different file download logics. This design can significantly reduce the chance of erroneous protection implementations. Moreover, FileGuard is more flexible than directory-based permission control since its permission control runs on file basis. To show the correctness of FileGuard design, we modify the source code of PHP5 script engine to implement a proof-of-concept prototype. Our experiments using this prototype implementation successfully confirm the negligible performance overhead introduced by FileGuard.

1.4 Thesis Structure

The remainder of this thesis is organized as follows. We conduct a comprehensive literature review in Chapter 2. We discuss inference attacks against trust-based onion routing in Chapter 3. We also reveal trust degree as a rescue to defeat this kind of attacks in this section. Moreover, we design SGor, a trust graph based onion routing algorithm that can mitigate key limitations in the use of trust for protecting anonymity in Chapter 4. In Chapter 5, we elaborate on the disastrous vulnerability in certification based trust model and propose an active approach to harden this flawed trust model. In Chapter 6, we study file download vulnerabilities in today's web. Our study uncovers the root cause of this vulnerability and hence leads to more trustworthy defenses. To this end, we conclude our current research and look ahead several future research directions in Chapter 7.

Chapter 2

Literature Review

To defend against the growing number of Internet attacks, a large number trust-based systems have been proposed and widely deployed in today's Internet. For example, SybilGuard [39], SybilLimit [40] and many other similar systems [41–44] have been proposed using a priori trust from social networks to thwart Sybil attack [45], which is perhaps the most dangerous attack in peer-to-peer networks [46]. Moreover, a large number of reputation systems have been designed to ensure the reliability of distributed computing [47–49] and online services [50–52]. These systems could calculate the reputation based on performance (such as quality of services) [53–55], past behaviors [47, 56] or users' feedbacks and recommendations [57].

Although there are huge number of trust-based system studies in the literature, we review only the past works that are closely related to our research in this Chapter. In particular, we first survey existing anonymous communication systems and major threats to these systems in Section 2.1. We then elaborate on the certification based trust model, an essential part of HTTPS protocol in today's web, in Section 2.2. After those two, we discuss file download vulnerabilities in Section 2.3.

2.1 Anonymous Communications

In today's Internet, users always expect to prevent their Internet behaviors (such as who is talking to whom) and addresses (usually in the form of IP addresses) from leaking to others when they visit the Internet. For this reason, anonymous communication is becoming a very important part of Internet communications. With this trend, a number of anonymous communication techniques and systems are proposed in the past three decades, such as MIX-net [4], DC-net [5], Crowds [6] and onion routing [7–9]. Among those techniques, onion routing is the most popular one since its prototype implementation Tor [9] has been deployed in the Internet for nearly ten

years and attracted global interests from Internet users. According to Tor project's official report [58], there are more than 4 millions of users visiting Internet through Tor till November 2013.

In the following sections, we will analyze existing anonymous communication techniques in details (see Section 2.1.1). We will also demonstrate the need of trust-based methods for anonymous communications through the survey of existing attacks (see Section 2.1.2) and report state-of-the-art trust-based anonymous communication systems as consequence (see Section 2.1.3).

2.1.1 Anonymous Communication Techniques

MIX-net [4] and its practical variants [59, 60] are perhaps the first anonymous communication design in the literature. The basic idea is to buffer and mix multiple users' messages using some relays or proxies (or call *mixers*) before sending them to destinations. By this mean, third parties other than senders and receivers cannot identify the source and destination of messages. MIX-net has severe performance issue because it should wait to buffer enough messages for mixing. Although more messages that are used to mix can lead to a better anonymity of MIX-net, the network also requires a longer delay to wait more messages to mix and forward. For this reason, MIX-net are usually used for latency-tolerate anonymous communications, such as transmitting electronic mails.

DC-net [5] is another typical anonymous communication technique inspired by the dining cryptographers problem. DC-net assumes a group of cryptographers sit around in a dinner table. These cryptographers agree with each other to make an anonymous payment but eager to know whether none of them paid the menu. They achieve this using a two-stage protocol. In the first stage, every two neighbor cryptographers share a secret bit (1 or 0 with equal probability). That is, each cryptographer can have two shared secret bits on his hand. In the next stage, every cryptographer publicly announce the exclusive OR (XOR for short) result of his two shared secret bits but the cryptographer who paid the dinner just invert his result. By this way, if the XOR result of all the announced bits is 0, none of these cryptographers paid the dinner. Otherwise, some cryptographer paid it but we cannot confirm who is the payer. This protocol can anonymously transmit one bit information to the public in each round of announcement. It can be easily extended to transmit longer messages by running this protocol many times. Although DC-net looks like simple and useful, it is hard to be applied to practical scenarios because of three limitations. First, it cannot support the message transmission by more than two cryptographers simultaneously. Second, the DC-net can be subverted if one cryptographer is malicious. Third, the DC-net is not scalable because each round of transmission requires the participation of all the cryptographers.

Crowds [6] is a famous anonymous communication system designed for world wide web. This system is designed based on the concept of "blending into a crowd". That is, an initial user's

web requests can be forwarded through a group of other users and ultimately reach the destination web server. By this way, the web server and third parties cannot identify who is the initial user. Compared with Mix-net, Crowds has a shorter communication latency and thus can support instant web transactions. And compared with DC-net, Crowds is scalable and can support anonymous communications carried out by multiple senders. Although Crowds shows many advantages compared with other anonymous communication systems, it still suffers from a serious issue: all the users who participate into the forwarding of initial user's request can observe the destination of this request. For this reason, Crowds cannot protect destination anonymity if some participants are malicious.

Onion routing [7, 8] and its second generation protocol, **Tor** [9] are proposed to ensure both source and destination anonymity. It can effectively prevent local attackers from knowing who is talking to whom. The core protocol of onion routing is to forward an initial user's message through a circuit of onion routers. The initial user encrypt his message using the public keys of all the routers in a layered structure, and each router can only decrypt one layer of the message using its own private key. By this mean, each router can know only its predecessor and successor, but cannot identify the source and destination of the message. Compared with Crowds, onion routing is also a low-latency anonymous communication protocol but can protect destination anonymity.

Rather than the four aforementioned anonymous communication techniques, there are many other anonymous communication systems in the literature. For example, Anonymizer [61] is a popular commercial system that can provide powerful anonymity and security protections. The freenet [62] enables users to anonymously chat on forums, share documents, browse and publish articles in the Internet without fear of being blocked or censored. I2P [63] is an anonymity network using similar algorithms like onion routing but offering better protection against traffic analysis. We do not elaborate on those systems here, because they are industrial products and nearly all the basic methods they use are covered by Mix-net, DC-net, Crowds and onion routing.

2.1.2 Attacks Against Onion Routing

With the blooming of anonymity systems, attacking techniques against these systems are also becoming hot research topics. Since different anonymity systems are designed by considering different attack models and our research focuses on trust-based onion routing, our survey here will mainly consider the attack model used by onion routing networks. In particular, Mix-net and DC-net are designed to thwart global attackers who can observe the whole anonymous communication infrastructure, while Crowds and onion routing are not. The latter two consider only the local attackers. Under local attacking model, the major attack against Crowds is a so-called predecessor attack [64]. By launching this attack, malicious participants can log their

predecessors for a long time and have a very high probability to successfully guess the initial user as the predecessor is likely the most frequent one appearing in the log.

Although predecessor attack is very effective in compromising source anonymity, it cannot attack destination anonymity which is also protected by onion routing. Instead, another kind of attacks, called correlation-like attacks, pose major threats to onion routing networks in the literature [19–27]. According to whether attackers can manipulate anonymity traffic, correlation-like attacks can be roughly grouped into two categories: passive correlation-like attack [19, 21, 22, 25, 26] and active correlation-like attack [19, 20, 23, 24, 27]. In the former attack, attackers can correlate the source and destination of an anonymity communication by passively analyzing the characteristics and patterns of anonymity traffic. In the latter one, attackers can actively embed traffic watermarks to reveal the communication anonymity. We will elaborate on the details of techniques used by various correlation-like attacks in the literature as follows.

Paul et al. [19] present a preliminary study to investigate potential attacks against onion routing systems. This study considers a comprehensive adversary model, where the compromised routers controlled by attackers may be a single one, multiple ones with a randomly fixed distribution or multiple ones with dynamic distributions from time to time. In order to correlate the source and destination of an anonymity traffic, the attackers can first employ one compromised router to mark the traffic by passively analyzing traffic features or actively delaying/corrupting this traffic. After that, attackers can use another compromised router to recognize the characteristics of this traffic on the other end. This paper gives a whole picture of security issues in onion routing systems, but does not discuss the cost of a successful attack.

Murdoch et al. [20] propose the first low-cost traffic analysis algorithm to deanonymize onion routing networks. The basic idea of this attack is to use a compromised web server to actively embed timing patterns into a traffic and utilize a compromised onion routers to measure the traffic load in other routers. If the traffic pattern created by the compromised server has been found in an onion router's traffic, attackers can directly correlate this router with the web server. Since onion routing follows a low-latency design, the timing delay compromised web server embeds into the traffic could be small (i.e., low-cost). Evans et al. [24] extend the basic idea of this attack to a more practical congestion attack. The extended attack exploits a design flaw of onion routing to maliciously amplify bandwidth by building long circuits which loop back on themselves, hence becoming practical even on heavily loaded onion routing networks.

Bauer et al. [22] describe a low-resource routing attacks against onion routing networks. This attack is not required to actively embed traffic watermarks into the anonymity traffic (i.e., low-resource). Instead, the attack can exploit the flawed circuit building algorithm to compromise anonymity. In particular, attackers can recognize whether the routers under their control are located at the first hop or last hop of onion circuits by analyzing circuit building traffic. Once attackers' routers are selected in the first and the last hops of an onion circuit, the source and

destination of this anonymity circuit can be revealed and correlated (i.s., anonymity is compromised).

Overlier et al. [21] study how to expose hidden services that are deployed with the help of onion routing networks. In this research, the authors find that a hidden server can be revealed by attackers if attackers' routers are chosen as the first hop router in the server's circuit to the rendezvous point. Attackers can impersonate as the client using the same router and passively analyze traffic characteristics to correlate the traffic across the same machine (the same machine impersonates as the first hop router in hidden server's circuit and the client who visits the hidden server). To prevent this attack, the authors also propose countermeasures. They suggest to set up some entry guard nodes polices to help hidden servers select trustworthy routers as their first hop router (the entry router). The entry guard nodes could be the routers that are behaving normally in the network for a long time. However, Alex et al. [65] find that the entry guard nodes are not sufficient to completely protect hidden services. Instead, they find practical flaws in the design and implementation of Tor. These flaws enable attackers to measure, shut down and deanonymize hidden services even with entry guards [65].

Ling et al. [23] report a new cell counter based attack against onion routing protocol. This attack is designed to embed watermarks into the variation of cell counter of the target anonymity traffic. The last hop onion router controlled by attackers can recognize this watermarks to correlate the destination and source of the traffic. This attack is highly efficient and effective, but hardly to be evaded since it exploits the application level design flaws. The same group of authors also extend the basic idea of this protocol-level attack to a more general versions and discuss them in-depth in their following papers [27, 66, 67]. Moreover, Zhu et al. [25] discuss correlation-like attacks in a high level, and Hopper et al. [26] conduct a comprehensive analysis on how much anonymity can be leaked through the traffic transmission latency in onion routing networks.

2.1.3 Trust-based Systems for Onion Routing

Although those correlation-like attacks exploit very different techniques to compromise anonymity in onion routing networks, they have a common requirement: the onion routers under their control are selected by users to establish onion circuits. As a result, identifying and excluding attackers' routers from users' onion circuits could effectively limit correlation-like attacks. However, due to the lack of effective trust models, it is very difficult for users to evade attackers' routers in existing onion routing networks. The attackers who control a large fraction of onion routers have a high probability to launch correlation-like attacks to compromise the anonymity of a large number of users. To this end, the use of trust for onion routing attracts considerable attention in recent research. With the help of trust that users have readily assigned to routers'

owners, onion routing users can effectively identify and exclude attackers' routers from their onion circuits.

Krishna et al. [11] propose the first trust-based onion routing algorithm. This algorithm integrates trust from online social networks to protect onion routing circuits. In particular, this algorithm enables onion routing users to establish onion circuits by selecting onion routers only from their first or second hop friends from online social networks, hence preventing attackers' routers. This algorithm could induce a very skewed router selection probabilities distribution and thus result in a significant reduction of the entropy (i.e., anonymity) of the entire onion routing network.

Drac [13] is another social trust based onion routing system. It allows users to select trustworthy routers through a traditional random walk [68] on top of an online social network. Although random walk can implicitly mitigate the bias of router selection probabilities distribution, it suffers from a new node degree attack [15]. The reason is that, traditional random walk will converge to a stable distribution (within a fast-mixing domain) which is proportional to each node's degree, while attackers are easy to deliberately enlarge the degree of the routers under their control. To address this problem, a new system, Pisces [15], is proposed. Pisces performs Metropolis-Hastings random walks [69] instead of traditional random walks through an online social network to discover honest routers. The Metropolis-Hastings algorithm can effectively thwart node degree attacks since the converged distribution of such random walk is irrelative to the degree of each node.

Johnson et al. [12] propose a theoretical model for reasoning about trust-based onion routing. This model discuss a complete attack model and design new onion routing algorithms based on trust to optimize the protection against correlation-like attacks. In contrast to the purely theoretical discuss, Johnson et al. also present a following paper to discuss practical scenarios [14]. This work addresses many practical challenges in the design of trust-based onion routing algorithms. For example, this work studies the accuracy and correctness of trust in trust-based onion routing design. It also discusses the biased trust distribution issues and proposes a downhill algorithm [14] to mitigate this issue as well. However, all existing trust-based onion routing systems cannot solve the trust correctness issue and biased trust distribution issue completely. To this end, we are mainly motivated to conduct an in-depth research to design more trustworthy trust-based onion routing solutions in this thesis.

2.2 Certification based Trust Model

Secure Sockets Layer (SSL for short) and its successor, Transport Layer Security (TLS for short), are perhaps the de facto Internet encryption/decryption standard [70–72], both of which

are designed using theoretically proofed cryptographic algorithms. They are widely used by Internet applications and protocols (such as HTTPs, SSH and IMPAs etc.) to ensure communication confidentiality and integrity. Moreover, SSL/TLS apply authentication using a certification based trust model which complies with a X.509 public key infrastructure (PKI) [10]. However, this trust model is missing or flawed implemented in many SSL/TLS based applications (e.g., SSH and HTTPs), hence making the authentication in these applications vulnerable to various man-in-the-middle attacks [29]. In this section, we first review SSL/TLS man-in-the-middle attacks in Section 2.2.1. We then discuss existing countermeasures against these attacks in Section 2.2.2.

2.2.1 SSL/TLS Man-in-the-middle Attacks

SSL/TLS rely on a certification based trust model to enable communication authentication. For example, if one end A will communicate with the other end B through SSL/TLS, A should trust a set of trust anchors in advance, and B need to provide a X.509 certificate to A for authentication. A authenticates B's identity by performing two verifications: one is whether the certificate's subject name or alternative name is the same as the address of B in the communication (i.e., B's IP address or domain name), and the other is whether the certificate is signed by one of trust anchors pre-trusted by A. If both verifications succeed, A regards B as legitimate. Otherwise, a man-in-the-middle attack possibly occurs.

Trust-on-first-use man-in-the-middle [35] Although SSL/TLS provide the certification based trust model for authentication purpose, not all the applications on top of SSL/TLS apply this model. For example, OpenSSH [73] is designed using SSL/TLS libraries to ensure secure remote login. However, it does not implement the certification based trust model to tackle man-in-the-middle attacks. Instead, OpenSSH Client prompts users to query whether they accept a certificate provided by an OpenSSH server during the first time communication. Users should make a decision based on their knowledge and experiences. As a result, the users who are not well educated and have not sufficient security experiences are unlikely to identify fake certificates and make incorrect decision, hence being vulnerable to man-in-the-middle attacks in the first time connections. The missing of certification based authentication is also prevalent in open source third-party libraries [74] and smart phone apps [75].

The null-prefix attacks [76] Modern browsers (e.g., Firefox, Chrome and Internet Explorer) have implemented the certification based trust model in their native code, hence being able to thwart trust-on-first-use man-in-the-middle attacks. However, their implementations contain flawed logics in processing X.509 certificates. In particular, browsers treat strings in X.509 certificates using C language which interprets the character $\backslash 0$ as the end of a string. However, certificate authorities process X.509 certificates using ASN.1 notation format [77]. This format

follows PASCAL string style and does not treat \0 as the end of a string. Instead, a string length is required when processing the string. To this end, attackers can purchase a valid certificate from a legal certificate authority (i.e., one of trust anchor) by setting the subject name of this certificate as "www.google.com\0.malicious.com". The certificate authority can sign this certificate as the domain "malicious.com" belongs to attackers. However, modern browsers will interpret this certificate as belonging to the domain "www.google.com", since they read strings by ending at \0. By this way, attackers can cheat certificate authorities to sign certificates to any domains. And then, they can use these null-prefix certificates to launch man-in-the-middle attacks and impersonate any domains.

OCSP attacks [78] The online status certificate protocol (OCSP for short) is a sub protocol of HTTPs. Its main usage is to help check against revoked or compromised certificates. However, an implementation flaw found in this protocol can make the checking ineffective, hence enabling man-in-the-middle attacks using revoked and compromised certificates. The flaw occurs because OCSP response status is not covered by signature and many browsers will ignore the authentication if the status is "tryLater (3)". As a result, attackers can use any revoked certificates to launch man-in-the-middle attacks. They need only to send "tryLater (3)" status to browsers to bypass OCSP protocol. In this scenario, no any warnings will show to users in the browsers.

SSL strip attacks [79, 80] SSL Strip is a very insidious SSL/TLS man-in-the-middle attacks. It sniffs HTTP traffic and tries to discover HTTPs links among HTTP responses. When an HTTPs link is found, SSL strip attackers use HTTP to replace the HTTPs and send the faked HTTP link to browsers. From browsers' point of view, they cannot differentiate whether the original links are HTTPs based or HTTP based. As a result, browsers could establish unencrypted connection (i.e., HTTP connection) to the SSL strip man-in-the-middle vantage point, while the SSL strip vantage point can construct HTTPs connection to remote web servers. By this mean, SSL strip attackers can monitor and modify all the traffic between browsers and remote HTTPs web servers. The authors deploy SSL strip attacks in a campus network for running a 24 hours experiment, and successfully steal more than 200 usernames and passwords from leading mail services such as gmail and yahoo. This experiment confirm the effectiveness and undetectability of this attack.

Man-in-the-middle attacks with a single compromised trust anchor [32–34] In contrast to aforementioned man-in-the-middle attacks, a more severe man-in-the-middle attack has been found along with a disastrous design flaw in the certification based trust model: a single compromised trust anchor can issue legal certificates to any domains, hence making man-in-the-middle possible to be against any domains. The root cause is that browsers cannot differentiate which trust anchor is the legal signer of which domain's certificate. Since this attack is hard to be evaded and can make disastrous consequences, it is becoming a hot topic in recent research. In

this thesis, we will concentrate our study of SSL/TLS man-in-the-middle attacks to this attack, and discuss its countermeasures in-depth.

2.2.2 Existing Countermeasures Against Man-in-the-middle Attacks

To address various SSL/TLS man-in-the-middle attacks, a lot of solutions have been proposed in the literature. Originally, the certification based trust model is an effective method that is designed to defeat man-in-the-middle attacks in the first time connection. However, this model is not implemented in all SSL/TLS based applications. OpenSSH is a typical example which does not apply this model for authentication. For this reason, a hunter system, called Perspective [35], has been designed to help those vulnerable applications defend against trust-on-first-use man-in-the-middle attacks. Perspective exploits Internet diversity to evade man-in-the-middle vantage points. The basic idea is to collect certificate samples of the same web server through a number of globally distributed certificate hunters around the world. Since globally distributed hunters are unlikely to be hijacked by man-in-the-middle vantage points at the same time, some hunters could receive the real certificate and raise a warning to indicate potential man-in-the-middle attacks.

To eliminate the null-prefix attacks and OCSP attacks, simply patching vulnerable browsers is enough. For thwarting the SSL strip attacks, a complete solution should be to strictly force all the HTTP traffic encrypted. As a response to this requirement, the Internet Engineering Task Force (IETF for short) proposes a new HTTP Strict Transport Security standard (HSTS for short) [81]. However, the population of websites that apply HSTS is extremely low in today's web. According to a recent HSTS usage statistics report [82], only less than 700 websites implement HSTS among the top 1 million sites (i.e., the usage ratio is less than 0.07%). The reason of this low ratio for HSTS usage is perhaps that HTTPs will incur high performance overhead for encryption/decryption but most of contents are not necessarily confidential (such as images and videos). As a result, to make a tradeoff between performance and protection, some solutions such as SSLLock [83] have bee proposed. SSLLock enables websites to host their secure contents in a special subdomain. This subdomain strictly follows HTTPs protocol but other domains remain the same HTTP protocol. By this way, SSLLock can well balance the performance overhead and protection capability against SSL strip attacks.

To defeat the man-in-the-middle attacks with a single compromised trust anchor, there exist three kinds of solutions in the literature. First, although Perspective [35] is not designed originally for this attack, it can thwart man-in-the-middle attacks with a single compromised trust anchor since its key idea is to evade man-in-the-middle vantage points. The following works such as CrossBear [36], NoAttackNecessary [84], HTTPS everywhere [85] and Convergence [86] extend the basic idea of Perspective with an in-depth analysis and employ global hunters

to discover the real cases of this attack in the Internet. However, as these prior systems provide detection results to end users as third-party services, the compromise of the third-party service can also subvert the entire trust model (we can consider the third-party service as an additional trust anchor). Second, Italo et al. [37] propose the use of a pre-shared password to defeat this attack, while Michael et al. [38] suggest to bind an HTTP cookie to a client-side certificate and use this pre-shared information to detect consequent man-in-the-middle attacks. Both of them employ some pre-shared secrets to address this problem, but fail to protect the first time authentication, which is an essential property the certification based trust model attempts to protect. Third, several website certificate pining protocols for SSL/TLS [87] have been proposed in the literature to tackle man-in-the-middle attacks. Typical examples are like DNS-Based authentication of named entities (DANE for short) [88], trust assertions for certificate keys [89], HTTP public key pinning extension [90] and the Monkeysphere project [91]. Although these pining methods are very effective in protecting some important domains and websites whose certificates are pre-stored in the browsers, they are not scalable to support the certificates expiration or withdraw, and even worse cannot provide a general and adaptive protection to all the Internet websites. In this thesis, our research focuses on the maximizing protection against SSL/TLS man-in-the-middle attacks by fully utilizing Internet diversity and available trust anchors. Our solutions are more trustworthy compared with prior solutions.

2.3 File Download Vulnerabilities on the Web

File download vulnerability (also known as directory or path traversal attack [92]) is one of the most harmful vulnerabilities in today's web. It could lead to disastrous consequences including but not limited to system intrusions and database intrusions. The root cause of this vulnerability is the missing or insufficient quarantine of untrusted user inputs, which is similar as many other popular web vulnerabilities such as cross-site scripting (XSS for short), SQL injection and command injection etc.. In this section, we first review file download vulnerability and its related web vulnerabilities in Section 2.3.1. We then survey existing countermeasures against file download vulnerabilities in Section 2.3.2.

2.3.1 Related Web Vulnerabilities

According to the ranking of top 10 web vulnerabilities reported by open web application security project (OWASP for short) [93] in the year of 2013, file download vulnerability belongs to both the security misconfiguration category and sensitive data exposure category. These two categories are ranked as the 5th and 6th positions, respectively. Moreover, the first and third rank web vulnerabilities are injection and cross-site scripting, both of which are due to the

similar root cause as file download vulnerability. In this sub section, we will elaborate on these three web vulnerabilities and explain their root causes.

Injection Vulnerability retains as the top 1 web vulnerability from the year of 2010 to 2013. This ranking indicates that this vulnerability is the most prevalent vulnerabilities across the web and could result in disastrous consequences. Generally, this vulnerability can be grouped into two sub categories, one is SQL injection [94] and the other is command injection [95]. Although the root cause of SQL injection and command injection is the same: unauthorized SQL or system commands cannot be sanitized by vulnerable web applications and hence being executed unwittingly, they have very different consequences. In particular, SQL injection can at most break into back end databases while command injection could compromise the whole operating systems. Due to the widespread presence and disastrous consequences, injection vulnerability is a very hot research topic [96–99] and the research for its countermeasures also attracts considerable attentions from both industrial and academical communities [100–109].

Cross-site scripting vulnerability (i.e., XSS) [110] is another prevalent vulnerabilities across the web. It is ranked as the 2nd position in 2010 OWASP report and the 3rd position in 2013 report. The root cause of XSS is similar as injection vulnerabilities: unauthorized user inputs cannot be sanitized by vulnerable web applications. However, the executor of those unauthorized contents is the front-end java script engine rather than the back-end databases and operating systems. Therefore, XSS cannot be used to attack web services directly. Instead, this vulnerability is usually exploited to steal web user credentials which are stored in browsers and bound to specific domains, hence breaking the same origin policy [111]. According to whether the injected cross-site scripts can be transmitted to the web server, XSS can be grouped into two categories: server-side XSS and client-side XSS (also known as DOM-based XSS) [112]. The server-side XSS can be further divided into stored XSS and reflected XSS, where the former one can enable attacking payload to be stored in the back-end database but the latter one cannot. To defeat server-side XSS, user input sanitization mechanisms can be implemented in either client-side scripts (e.g., java scripts) or server-side scripts (e.g., PHP, JSP or ASP.NET). In contrast to the server-side XSS, the client-side XSS allows browsers to execute the injected attacking payloads without any communications to the remote web servers. As a result, only client-side sanitization is effective in defeating the client-side XSS. In the literature, the comprehensive research on preventing both the server-side and client-side XSS has already conducted [113–118].

File download vulnerability occurs due to the use of unsanitized user inputs as file names to access local filesystem. Its consequences depend on which permission the web server runs on. For example, if a web server is configured to run on a root privilege, vulnerable web applications can enable malicious users to download any critical system files such as the `/etc/shadow` in Linux. Even if web servers are configured with a lesser privilege, they also have implicit read access to world-readable files in the system. In Linux systems, the file `/etc/passwd`

and many other files in `/etc/` directory fall into this category (a file is world-readable if the *others reference mode* of this file is set to "r" in Linux). Compared with injection vulnerability, this vulnerability targets to attacking local filesystems rather than databases and operating systems. Compared with XSS which is for stealing sensitive data in client side, file download vulnerability can be exploited to steal sensitive files (including session files) from web servers directly. However, all these three kinds of vulnerabilities share the same root cause: the missing or insufficient sanitization of user inputs. We call them input validation vulnerabilities.

2.3.2 Existing Countermeasures Against File Download Vulnerability

Defending against file download vulnerabilities have been studied for a long time and several methods have been proposed in the literature. Roughly, those defending methods can be classified into three groups: user input sanitization, web server permission control and machine learning based detection algorithms. However, all these methods still suffer from a number of practical drawbacks.

First, there exist a number of alternative sanitization mechanisms such as black-listing or white-listing file names and sensitive key words elimination. These mechanisms have very different implementation requirements. For example, to implement a black-listing method based on extension name, developers need to consider all the possible extension names sensitive files may use. While the implementation of white-listing methods requires only the knowledge of extension names for the download-enabled files. Moreover, to implement a sensitive key words elimination method, whether the web application uses relative path or absolute path to access local filesystem should be considered. These diverse requirements make the implementation of user input sanitization mechanisms error-prone. Second, the permission control for web servers is too coarse to protect many system files, especially the files that are world-readable. Although modern web servers can limit their read permission to some specific directories, the sensitive files (usually the source codes of server-side scripts) are still vulnerable. Moreover, since the directory based permission control cannot differentiate which directory is bound to which web application, an vulnerable application can be exploited to illegally download files belonging to other web applications. Third, although machine learning based detection algorithms attract broad interests in recent research [119–121], they can hardly reduce detection false positives to zero. Due to these practical drawbacks, we are motivated to design a more reliable method for defending against file download vulnerability in this thesis.

Chapter 3

Trust Degree based Onion Routing

Recently, trust-based onion routing has become a hot research topic [11–15], because it can effectively protect anonymity even if a significant fraction of the network is compromised. Onion routing protects anonymity by hiding user identities behind onion circuits. Each onion circuit is comprised of a sequence of layered onion routers. However, since existing onion routing networks such as Tor [9] do not deploy identity checking mechanisms to verify the identities of onion routers, the anonymity protected by onion routing suffers a serious threat from various correlation-like attacks [20–26, 122]. Using traffic watermarking or traffic analysis techniques, the attackers who control the first and the last routers in users' onion circuits can correlate these routers in the same circuit, and hence reveal which user visits which destination. To tackle these correlation-like attacks, trust-based onion routing [12, 14] has been proposed. By incorporating trust for routing algorithms, trust-based onion circuits can be constructed using trustworthy onion routers and therefore significantly limit correlation-like attacks.

Although trust-based onion routing is proved effective against correlation-like attacks, it induces a new kind of attacks, called inference attacks, due to biased a priori trust distributions [14]. As trust-based onion routing cannot be designed to protect anonymity by means of obscurity, the powerful attackers who have the knowledge of a priori trust relationships among users and routers are potentially existing. These attackers have a high probability to guess the initiate user of a trust-based onion circuit if they can observe onion routers in this circuit (i.e., inference attacks). As discussed in [14], the inference attack poses a major threat to trust-based onion routing because it can largely reduce the anonymity protected by trust-based onion circuits.

Defeating the inference attack in trust-based onion routing is a very challenging problem, because different users usually have preferences for different sets of onion routers according to trust. The state-of-the-art countermeasures usually make a tradeoff between the capability against inference attacks and the capability against attackers' routers (e.g., correlation-like attacks are performed based on these routers), and show their effectiveness in terms of overall

anonymity [13–15]. These countermeasures cannot differentiate the anonymity compromised by inference attacks from other attacks, hence missing opportunities to disclose key features against inference attacks in the design space. For example, a downhill algorithm is proposed to construct trust-based onion circuits using a decreasing trust threshold along these circuits [14]. Trust threshold is an indirect feature that can only mitigate inference attacks by sacrificing the capability against attackers' routers. Moreover, although onion routing through social networks [13, 15] shows better protection of the overall anonymity, the features used to thwart inference attacks cannot be isolated and analyzed independently.

To effectively protect anonymity against inference attacks, a fundamental challenge is to discover the key features in the design space. These features can offer distinct advantages for the design of effective and free-of-tradeoff countermeasures, but unfortunately have not been discovered by previous studies.

In this chapter, we address this fundamental challenge by deriving and analyzing a novel inference attack model. We first model trust-based onion routing as a probabilistic hypergraph, and then isolate the design space for inference attacks by truncating this hypergraph. Based on the analysis of this isolated design space, we uncover trust degree as the key feature against inference attacks. We design a proof of concept routing algorithm by taking advantage of our findings, and confirm the effectiveness of trust degree in protecting anonymity under real-world settings.

We list our main contributions in this chapter as follows.

1. We present a novel attack model to isolate the anonymity compromised by inference attacks from other attacks. Compared with generic models that focus on overall anonymity, our model can exclusively reason about inference attacks without being affected by irrelevant features.

2. By analyzing our model, we uncover trust degree as the key feature against inference attacks. Trust degree is very effective in preventing inference attacks, but is overlooked due to the generic model based analysis in prior research.

3. We design a proof of concept routing algorithm by taking into account of trust degree. Our routing algorithm makes no tradeoffs between the capability against inference attacks and the capability against attackers' routers. It can be embedded into existing trust-based routing algorithms [12, 14], or even existing inference attack countermeasures (e.g., the downhill algorithm [14]), to further strengthen the protection of anonymity.

4. We compare trust-based routing algorithm and the downhill algorithm with and without considering trust degree using real-world social networking datasets. The results confirm that trust degree is a very effective feature against inference attacks.

Although we concentrate on the analysis of inference attacks in this chapter, we expect the principle of isolating design space for a particular attack could be applied broadly. The isolated attack model is attractive due to its ability of extracting key features. These features are usually very effective against the particular attack, but hard to be discovered by the analysis using generic models.

Chapter Structure: The remainder of this chapter is organized as follows. We first present a novel attack model to isolate inference attacks in Section 3.1. We then reveal trust degree as the key feature against inference attacks and present a new routing algorithm by taking advantage of trust degree in Section 3.2. Finally, we confirm the effectiveness of trust degree based routing algorithm using real-world social networking datasets in Section 3.3.

For the ease of reference, we also summarize important notations used by this chapter in Table 3.1.

TABLE 3.1: Important Notations in Chapter 3.

Notation	Definition
G	A trust graph. $G = (U \cup R, U \times R)$
U, R	U is a set of onion routing users, R is a set of owners of onion routers
$t(u, r)$	A trust level an user $u \in U$ assigns to a router's owner $r \in R$
$H = (U \cup R, U \times R^{\ell})$	A probabilistic hypergraph to model onion routing network.
ℓ	The length of onion circuits that users can make in the network
$(u, C) \in U \times R^{\ell}$	An onion circuit C that is initiated by user u
C_k	The k-th router in onion circuit C
$Pr[C\|u]$	The probability that user u has to initiate the onion circuit C
$Pr[C_k\|u]$	The probability that user u uses to select the router $C_k \in R$
$O \subseteq \{k : 1 \leq k \leq \ell\}$	The set of circuit positions observed by inference attackers
$Y[u_i\|C_O]$	The success probability that attackers have to guess u_i by observing C_O

3.1 Inference Attack Model in Trust-based Onion Routing

In this section, we reason about inference attacks in trust-based onion routing. We first describe the anonymity that onion routing can protect in Section 3.1.1. We then elaborate on the trust which can be used to prevent attackers' routers but induces inference attacks in Section 3.1.2. After listing the attacking capabilities that inference attackers have in Section 3.1.3, we propose a novel attack model to isolate the anonymity compromised by inference attacks in Section 3.1.4.

3.1.1 The Anonymity

Onion routing protocols are designed to prevent an attacker from linking users and destinations that the users visit in the Internet. There are two kinds of anonymity that onion routing can protect. One is user anonymity and the other is destination anonymity. The user anonymity concerns the protection of user identities, while the destination anonymity considers the protection of which destination users visit. Since Johnson et al. [14] argued in their work that the destination anonymity can be best protected using a single hop onion circuit consisting of the most trusted router, we focus on user anonymity in our design.

3.1.2 Trust Graph

Following prior research of trust-based onion routing [11–15], we consider the trust that users have readily assigned to onion router owners in this thesis. This notion of trust contains twofold meanings [12, 14]. One is the probability that the router's owner is an attacker itself. A lower probability means a higher level of trust. The other meaning is the difficulty that an honest person's router can be compromised by attackers. A higher level of difficulty indicates a higher level of trust. Using this notion of trust, the capability of evading attackers' routers can be measured in terms of the level of trust.

By applying this notion of trust to the whole trust-based onion routing network, we model a priori trust relationships among users and onion routers as a weighted directed trust graph G. Let U be the set of users. Let R be the set of onion routers (or the owners of onion routers). We have $G = (U \cup R, U \times R)$, where $U \cup R$ is the set of vertices and $U \times R$ is the set of directed edges in G. Each edge $(u, r) \in U \times R$ represents a trust relationship from a user $u \in U$ to an onion router $r \in R$ and is associated with a weight $t(u, r)$ to indicate the level of this trust. Note that, since a person can play the role as a user and a router's owner at the same time, G is not a bipartite graph (generally $U \bigcap R \neq \varnothing$).

Since users need outside knowledge to estimate the trustworthiness of onion routers, Johnson et al. [12, 14] argue that users can only have a very coarse level of trust for onion routers. Almost existing research [11–15] designs their trust-based onion routing algorithms by considering two distinct levels of trust. We follow this setting and consider our trust model with two trust levels: $t(u, r) = 1$ means u *trusts* r while $t(u, r) = 0$ represents u *distrusts* r.

3.1.3 Threat Scenario

To perform an inference attack, attackers are required to observe at least one onion router in a user's onion circuit and have the knowledge of a priori trust distributions over the network. For this reason, we consider the attackers have the following attacking capabilities:

The capability of observing routers in onion circuits: Here, we consider attackers have two means to observe onion routers in users' onion circuits. First, we consider the destinations (e.g., web servers) that users visit through onion circuits are certainly controlled by attackers. Attackers can employ these destinations to observe the last router in users' circuits. Second, we consider attackers can deploy or compromise onion routers in the network. Although trust-based routing algorithms can be used to evade attackers' routers with a high confidence, trust-based circuits are not necessarily free of attackers' routers. Attackers can exploit their routers which are used in users' onion circuits to observe adjacent routers in the circuits. Note that, since onion routing is not designed to defend against the attackers who can monitor the whole communication infrastructure (e.g., an ISP level attacker) [8, 9], attackers are assumed to have no chance to observe routers if they cannot control destinations or onion routers in users' onion circuits.

The capability of correlating observed onion routers: Attackers can actively embed traffic watermarks or passively analyze traffic pattern using the routers and destinations under their control, and hence correlate observed onion routers in the same onion circuit [20–25, 122]. If the inference attack is performed by merely observing a single router (e.g., the last router) of onion circuits, this capability is not pre-requisite.

The capability of locating the observed routers: Attackers can identify which positions the observed routers are located in onion circuits [14]. For example, since the real-world onion routing system Tor is hard coded to construct three-router onion circuits [9], attackers can locate the positions of the routers under their control easily. In particular, as the last router can be observed by the destination, attackers can confirm that a router stays at the last hop if this router can be observed by the destination. If attackers can observe the last router through another router under their control, this "another" router is certainly located at the second hop. Otherwise it locates at the first hop. Moreover, if onion circuits contain more than three routers, attackers can also estimate the positions of their routers by analyzing the control traffic when establishing onion circuits [23, 122]. Since attackers use the routers under their control to observe adjacent routers, if the hops of attackers' routers can be identified, the hops of observed routers can be located as well.

The capability of accurately estimating a priori trust distributions: As discussed in [14], highly capable attackers usually have the capability of collecting the knowledge of a priori trust distributions over the network. They can make accurate estimation to reveal trust relationships among

users and onion routers using outside knowledge. For example, if a user is a member of an organization, this user is more likely to trust the routers deployed by this organization. If both users and routers' owners are members of social networks, attackers can profile the trust relationships by crawling online social networks [123, 124]. Moreover, since trust-based onion routing algorithms are always set up by default in softwares and shared in the public, attackers who have the knowledge of a priori trust relationships can also accurately estimate users' trust-based router selection probabilities [12, 14].

3.1.4 Inference Attack Model

We present a model for exclusively reasoning about inference attacks in the context of trust-based onion routing. Unlike generic models that consider overall anonymity, our model targets on isolating the anonymity compromised by inference attacks and therefore results in an exclusive design space for the analysis of inference attacks.

3.1.4.1 Model Requirements

To derive the exclusive design space, our model should satisfy the following requirements:

1. Making inferences to guess the initiate user of onion circuits should be the only method that can be used to compromise anonymity in our model.
2. The design space inside of our model should be sufficient for interpreting inference attacks.
3. The design space outside of our model should be preserved.

The first requirement is used to exclude the anonymity compromised by other attacks. For example, if both the first and last routers of an onion circuit are controlled by attackers, the initiate user can be deanonymized immediately. This case should not be considered in our model.

The second requirement expects to protect the design space of inference attacks from being affected by other attacks, even if these attacks are pre-requisites of inference attacks. For example, although attackers can benefit inference attacks by means of controlling more routers and hence observing more routers in users' onion circuits, how attackers compromise onion routers and correlate these routers of the same circuit are out of scope of our model. As a result, given an onion circuit, which hops are observed and which are not can be simply assumed as a known constant in our model.

The last requirement is for eliminating the impacts on the design space outside of our model. With this requirement, our model should prevent the analysis of inference attacks from inducing side effects to the capability against other attacks, hence preserving the design space outside of our model.

3.1.4.2 Model Design

We propose a probabilistic model to meet the three aforementioned requirements, and hence isolate the design space of inference attacks in trust-based onion routing. Our model is built on top of the trust graph presented in Section 3.1.2.

Definition 3.1. [Onion Routing Overview] We model a trust-based onion routing network as a probabilistic hypergraph $H = (U \cup R, U \times R^{\ell})$ [1]. ℓ is the length of onion circuits that users can make in the network. Each edge $(u, C) \in U \times R^{\ell}$ represents an onion circuit C that is initiated by user u through trust-based onion routing. $C \in R^{\ell}$ consists of ℓ routers and $C_k, 1 \leq k \leq \ell$ is the k-th router in C. A weight $Pr[C|u]$ is associated with the edge (u, C) to represent the probability that user u has to initiate the onion circuit C.

As users initiate trust-based onion circuits by selecting routers for each hop independently, $Pr[C|u]$ can be calculated as $Pr[C|u] = \prod_{k=1}^{\ell} Pr[C_k|u]$. Where $Pr[C_k|u]$ represents the probability that user u uses to select the router $C_k \in R$ in the k-th hop of his onion circuit. Trust-based onion routing algorithms determine $Pr[C_k|u]$ according to the trust level $t(u, C_k)$ and the position k in the onion circuit. It can be seen, the hypergraph H models trust-based onion routing by capturing a priori trust-based onion circuits distributions over the whole network.

Although the hypergraph H can show the picture of the whole trust-based onion routing, it cannot well interpret the network exposed to inference attackers. Generally, inference attackers usually have the chance to only observe a partial onion circuits. As a result, we introduce a sub hypergraph $H(O) \subseteq H$ to capture the network from the view of inference attacks.

Definition 3.2. [Inference Attack Overview] We cut a sub hypergraph $H(O) = (U \cup R, U \times R^{|O|})$ to represent the part of trust-based onion routing that is exposed to inference attacks, where $O \subseteq \{k : 1 \leq k \leq \ell\}$ is the set of circuit positions observed by inference attackers and $|O| \leq \ell$ is the size of set O. Each edge $(u, C_O) \in U \times R^{|O|}$ represents a sub sequence of routers in u's onion circuit C (i.e., $C_O = \{C_k, k \in O\} \sqsubseteq C$). These routers can be observed by inference attackers. The weight $Pr[C_O|u]$ associated with each edge (u, C_O) indicates the probability that user u has to construct an onion circuit with the sub sequence of routers C_O.

[1] The operator $\times$ is a cartesian product operator and R^{ℓ} stands for the cartesian product of ℓ Rs.

In trust-based onion routing, $Pr[C_O|u]$ can be calculated as $Pr[C_O|u] = \prod_{k \in O} Pr[C_k|u]$, because user u can select the router C_k in the k-th hop of his onion circuit according to trust $t(u, C_k)$ independently.

The set O is resulted from how attackers control onion routers in users' circuits, but makes impacts on the effectiveness of inference attacks. It could affect the independence of inference attack analysis. The hypergraph $H(O)$ apparently excludes this side effect because it considers the set O as a deterministic constant input. As a result, $H(O)$ is required by our model to support the second model requirement listed in Section 3.1.4.1,

However, $H(O)$ is not sufficient to preserve the capability against other attacks. If we optimize $Pr[C_O|u]$ based on the hypergraph $H(O)$, it is possible to transfer selection probabilities from the routers with higher trust level to the routers with lower trust level. This side effect could induce the attacks that are prevented by trust (e.g., correlation-like attacks). To overcome this challenge, we further narrow down the hypergraph $H(O)$ by considering only the onion circuits in which the routers for each hop are equally trusted by a user. As discussed in Section 3.1.2, the routers that a user equally trusts have the same possibility of being controlled by this user's attackers [12, 14].

Without loss of generality, we consider a user $u_i \in U$ who visits the Internet using trust-based onion circuits. Let $R_e \subseteq R$ be a set of routers that u_i equally trusts, such as $\forall r \in R_e$, $t(u_i, r) = t_e$. Hence, R can be divided into several disjoint R_es with different t_es. That is, $R = \{R_1, R_2, \ldots, R_\nu\} = \{R_e | 1 \leq e \leq \nu\}$ and $\forall R_e, R_{e'} \subset R, R_e \bigcap R_{e'} = \varnothing$. Where ν is the number of distinct trust levels and we can simply consider $t_1 > t_2 > \cdots > t_\nu$. Although we set $\nu = 2$ for our experiments according to the discussion in Section 3.1.2, we use ν to describe our model in general.

To restrict our model to the routers that u_i equally trusts and hence support the third model requirement in Section 3.1.4.1, we introduce the notion of hypergraph $H(O, R_e^{|O|})$ as follows.

Definition 3.3. [Exclusive Design Space] We decompose the hypergraph $H(O)$ into several sub hypergraphs $H(O, R_e^{|O|}) = (U \cup R, U \times R_e^{|O|})$, where $R_e^{|O|} \in \{R_1, R_2, \ldots, R_\nu\}^{|O|}$. In this hypergraph, given a position $k \in O$, all the routers in this position should be from the set R_e (i.e., $\exists k \in O, \forall C_k \in H(O, R_e^{|O|}), t(u_i, C_k) = t_e$).

By using O and $R_e^{|O|}$, the hypergraph $H(O, R_e^{|O|})$ can isolate an exclusive design space for the independent analysis of inference attacks. Inference attack countermeasures that are designed based on $H(O, R_e^{|O|})$ does not necessarily sacrifice the capability against attackers' routers. We note that $H(O, R_e^{|O|})$ is from the view of a particular user u_i and different $u_i \in U$ could result in different $H(O, R_e^{|O|})$ for analysis.

Now, we turn to the interpretation of inference attacks in the hypergraph $H(O, R_e^{|O|})$. According to the first model requirement from Section 3.1.4.1, making inferences to guess the initiate user of onion circuits is the only way to deanonymize users in our model. We then give a function to describe this kind of deanonymization in our model.

Definition 3.4. [Isolated Anonymity] Let $Y[u_i|C_O]$ be the success probability that inference attackers have to guess the initiate user u_i by observing a sub sequence of routers C_O in u_i's onion circuit. $Y[u_i|C_O]$ can be calculated as:

$$Y[u_i|C_O] = \frac{Pr[C_O|u_i]}{\sum\limits_{u \in U} Pr[C_O|u]}. \tag{3.1}$$

With Definition 3.4, we have two implicit assumptions: First, inference attackers have the knowledge of the isolated hypergraph $H(O, R_e^{|O|})$. This hypergraph contains sufficient a priori trust information for the calculation of $Y[u_i|C_O]$. Second, we calculate $Y[u_i|C_O]$ without taking unobserved hops (i.e., $k \notin O$) into consideration, although these unobserved hops can benefit inference attacks with some additional information (i.e., inference attackers have the knowledge that the routers used in these unobserved hops are not controlled by them). We exclude the guessing based on these unobserved hops, because the distribution of attackers' routers in the network can affect this guessing and hence prevent the independent analysis of inference attacks in our model. That is, the second model requirement listed in Section 3.1.4.1 cannot be satisfied if these unobserved hops are taken into account.

To sum up, our isolated attack model can fulfill all the three model requirements that we have presented in Section 3.1.4.1 using the four model definitions. In particular, the first requirement is addressed by the use of definition 3.4, because this definition considers inference attacks as the only method that can be used to compromise anonymity. The second requirement is fulfilled using the definition 3.2 and definition 3.4. The definition 3.2 isolates the portion of onion routing network that is exposed to inference attackers and thus prevent the influences from other attacking techniques. The definition 3.4 expresses how to interpret inference attacks. Both of these two definitions cooperate to isolate a sufficient design space for interpreting inference attacks. The third requirement is addressed by the definition 3.3. This definition can isolate an exclusive design space for the independent analysis of inference attacks, hence preventing the impacts to the design space of other attacks.

3.2 Trust Degree to the Rescue

In this section, we study why trust degree is effective and how it can be used to resist inference attacks. We first analyze our inference attack model and discover trust degree as the key feature

against inference attacks in Section 3.2.1. We then design a new routing algorithm by taking into account of trust degree to resist inference attacks in Section 3.2.2. We also discuss several limitations of our proposed routing algorithm in this section.

3.2.1 Model Analysis

In our inference attack model, a user u_i's anonymity can be measured in terms of the probability $Y[u_i|C_O]$. A lower $Y[u_i|C_O]$ indicates inference attackers have more difficulties to deanonymize u_i by making inferences based on the observation of C_O. To make the $Y[u_i|C_O]$ prone to analysis, we separate variables from constants[2] in the Eq. (3.1) as below.

$$Y[u_i|C_O] = \frac{Pr[C_O|u_i]}{Pr[C_O|u_i] + \sum\limits_{u \in U \setminus u_i} Pr[C_O|u]}. \tag{3.2}$$

It can be seen from Eq. (3.2), the calculation of $Y[u_i|C_O]$ is apparently determined by $\sum_{u \in U \setminus u_i} Pr[C_O|u]$ which is the only constant regardless of the probability that u_i uses to select C_O (i.e., $Pr[C_O|u_i]$). A larger $\sum_{u \in U \setminus u_i} Pr[C_O|u]$ leads to a lower $Y[u_i|C_O]$. As a result, given two sequences of onion routers C'_O and C_O, u_i's anonymity can be better protected by using C'_O rather than C_O in the context of inference attacks if $\sum_{u \in U \setminus u_i} Pr[C'_O|u]$ is larger than $\sum_{u \in U \setminus u_i} Pr[C_O|u]$.

Given a user, we define the trust degree of a set of routers as the sum of trust or trust-based selection probabilities from other users to this set of routers. Therefore, the $\sum_{u \in U \setminus u_i} Pr[C_O|u]$ in Equation 3.2 is the (trust) degree of the sequence of onion routers C_O in the hypergraph $H(O, R_e^{|O|})$. These probabilities are determined by trust in trust-based onion routing. According to the analysis of Eq. (3.2), we find that the trust degree $\sum_{u \in U \setminus u_i} Pr[C_O|u]$ is very effective in defeating inference attacks. We measure the effectiveness in terms of $Y[u_i|C_O]$, the success probability that inference attackers have to guess the initiate user. A smaller $Y[u_i|C_O]$ indicates a better anonymity protection. It can be seen in Eq. (3.2), users who select a sequence of onion routers with a higher trust degree can have a better protection of their anonymity in the face of inference attacks, because a larger $\sum_{u \in U \setminus u_i} Pr[C_O|u]$ apparently results in a smaller $Y[u_i|C_O]$.

Figure 3.1 presents an example to demonstrate the effectiveness of trust degree in resisting inference attacks. In this example, Bob and Ken are two volunteers who deploy onion routers. Alice, as a user of the trust-based onion routing, trusts Bob and Ken equally. Pete is an attacker who knows a priori trust relationships among users and onion routers. If Pete observes Bob's router in Alice's onion circuit, he can deanonymize Alice immediately, because Alice is the only

[2]From u_i's point of view, the variables are the parameters under u_i's control, while the constants are the parameters out of u_i's control.

one using Bob's router (i.e., Bob is trusted only by Alice). However, Pete cannot deanonymize Alice by observing Ken's router immediately, because many other users can also select Ken's router (i.e., Ken is also trusted by many other users).

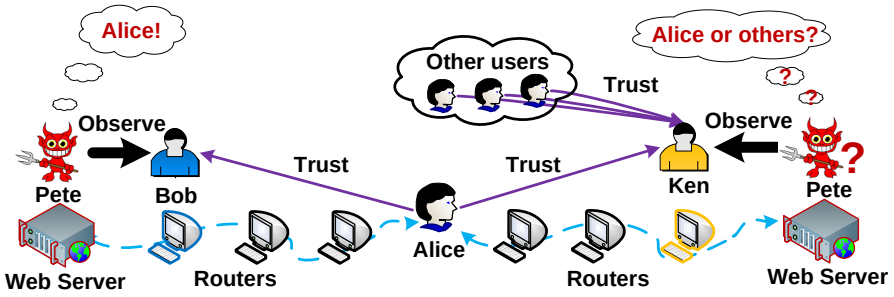


FIGURE 3.1: An example to show the effectiveness of trust degree in defeating inference attacks.

In Figure 3.1, Alice can be interpreted as u_i in the Equation 3.2, and other users are $u \in U \setminus u_i$. Since there is only one position being exposed to inference attacks, we have $|O| = 1$. The routers deployed by Ken and Bob consist of $R_e^{|O|}$. Alice (i.e., u_i) should determine his probability for selecting Ken's router or Bob's router (i.e., $Pr[C_O|u_i], C_O \in R_e^{|O|}$).

Although previous studies have not discovered trust degree as the key feature against inference attacks, they have proposed routing algorithms to resist inference attacks by implicitly and indirectly exploiting trust degree. For example, the downhill algorithm [14] can enlarge trust degree by decreasing trust threshold, while social network based onion routing [13, 15] increase trust degree through random walks on top of social networks. However, since these algorithms are not designed based on an isolated design space, they usually reduce the capability against attackers' routers. Here, we conduct a very different research. Our contributions primarily lie in the disclosure of why trust degree is effective and how it can be used to defeat inference attacks without sacrificing the capability against attackers' routers.

3.2.2 Routing Algorithm with Trust Degree

To prove the effectiveness of trust degree in resisting inference attacks, we design a new routing algorithm by considering trust degree in this section. We investigate a user u_i in the context of a population of other users whose router selection probabilities are known in advance (i.e., $\forall u \in U \setminus u_i, Pr[C_O|u]$ are constants). Our algorithm targets on minimizing the expectation of $Y[u_i|C_O]$ by optimizing the distribution of $Pr[C_O|u_i]$ in the hypergraph $H(O, R_e^{|O|})$. Although there are different functions can be used to measure $Y[u_i|C_O]$ when optimizing the distribution of $Pr[C_O|u_i]$, we choose the expectation of $Y[u_i|C_O]$ as our measure because previous studies used it to show the effectiveness of their trust-based onion routing algorithms [12, 14].

Let $E(Y)$ be the expectation of $Y[u_i|C_O]$. It can be calculated as:

$$E(Y) = \sum_{C_O \in R_e^{|O|}} Pr[C_O|u_i] \cdot Y[u_i|C_O]. \tag{3.3}$$

We call $E(Y)$ an "expectation" although it could not satisfy $\sum_{C_O \in R_e^{|O|}} Pr[C_O|u_i] = 1$. In the hypergraph $H(O, R_e^{|O|})$, we usually have $\sum_{C_O \in R_e^{|O|}} Pr[C_O|u_i] \leq 1$.

By targeting to minimize $E(Y)$, the design of the routing algorithm that takes into account of trust degree can be formalized as an optimization problem as follows.

$$\min\ E(Y),\ s.t.\ \forall k \in O, \sum_{C_k \in R_e} Pr[C_k|u_i] = \theta_{ke}. \tag{3.4}$$

Where, $\theta_{ke} \leq 1$ is the sum of probabilities that u_i uses to select routers from R_e for the k-th position of his onion circuit. Apparently, for $\forall k \in O$, $\sum_{e=1}^{\nu} \theta_{ke} = 1$. ν is the number of distinct trust levels (see Section 3.1.4.2). In the isolated design space $H(O, R_e^{|O|})$, θ_{ke} is determined by trust-based routing algorithms and should be kept as an unchanged value when solving the optimization problem described in Eq. (3.4). Since routers belonging to R_e are equally trusted by u_i, keeping θ_{ke} unchanged can preserve u_i's capability against the routers controlled by attackers. As a result, by solving the optimization problem described in Eq. (3.4) subject to θ_{ke}, we can design routing algorithm for u_i to resist inference attacks without sacrificing the capability against attackers' routers.

3.2.2.1 Optimization for Single Hop Router Selection

To solve the optimization problem listed in Eq. (3.4), we first consider the simplest scenario in which only one position of u_i's onion circuit can be observed by inference attackers (i.e., $|O| = 1$). It could be the case that attackers control the destination (e.g., a web server) and observe the last router (i.e., the router in the last position ℓ in Figure 3.2).

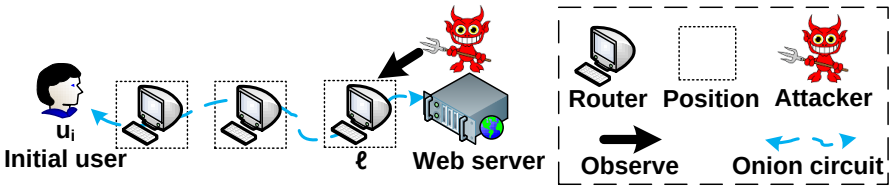


FIGURE 3.2: Attackers only observe the last hop in u_i's circuit.

In this setting, the optimization problem can be simplified to the following equation:

$$\min E(Y[u_i|C_\ell]),\ s.t. \sum_{C_\ell \in R_e} Pr[C_\ell|u_i] = \theta_{\ell e}. \tag{3.5}$$

Theorem 4.1 gives the solution of the simplified optimization problem described in Eq. (3.5). The proof of this theorem can be found at Appendix A.1.

Theorem 3.5. *Subject to $\theta_{\ell e}$, the expectation $E(Y[u_i|C_\ell])$ that inference attackers have to deanonymize the user u_i by observing the last hop (i.e., the position ℓ) of the circuit can be minimized to:*

$$\min E(Y[u_i|C_\ell]) = \frac{\theta_{\ell e}^2}{\theta_{\ell e} + \sum\limits_{C_\ell \in R_e} \sum\limits_{u \in U \setminus u_i} Pr[C_\ell|u]}.$$

The corresponding optimal distribution of $Pr[C_\ell|u_i]$ is in proportion to the trust degree of C_ℓ.

Gievn $\sum_{C_\ell \in R_e} Pr[C_\ell|u_i] = \theta_{\ell e}$, for $\forall C_\ell \in R_e$:

$$Pr[C_\ell|u_i] \propto \sum_{u \in U \setminus u_i} Pr[C_\ell|u].$$

Theorem 4.1 describes the optimal router selection probability distribution over the set R_e for the last position of onion circuit (i.e., the position ℓ). By applying Theorem 4.1 to all the $\{R_1, R_2, \cdots, R_\nu\}$ respectively, u_i can have an optimal router selection distribution over the whole router set R.

Steps for the use of the optimal single hop router selection algorithm: By using Theorem 4.1, users can apply the optimal single hop router selection strategy to existing trust-based routing algorithms with four steps:

Step 1: The user u_i can first divide the set of routers R into several mutually disjoint sub sets R_es according to different trust level t_es. Each R_e contains routers with the same trust level t_e from u_i's point of view (i.e., for each $r \in R_e$, $t(u_i, r) = t_e$).

Step 2: The user u_i determines $\theta_{\ell e}$ for each R_e according to trust-based algorithms.

Step 3: For each R_e, u_i finds the optimal distribution of $Pr[C_\ell|u_i]$ using Theorem 4.1.

Step 4: After having the optimal distribution of $Pr[C_\ell|u_i]$ for all the $R_e \subseteq R$, u_i can obtain the optimal distribution of $Pr[C_\ell|u_i]$ for R by concatenating the optimal distribution of $Pr[C_\ell|u_i]$ for all the $R_e \subseteq R$.

Using these steps, u_i can minimize the expectation of being deanonymized by the inference attackers who can observe the last hop of u_i's onion circuit (i.e., the position ℓ in the circuit), but does not sacrifice the capability of evading the routers controlled by attackers (because of keeping $\theta_{\ell e}$ as the same as it in trust-based algorithms). To implement this optimal router selection, the user u_i is required to have the knowledge of a priori distributions that other users have to select routers for this position in their circuits.

3.2.2.2 Optimization for Multiple Hops Router Selection

Now, we seek the solution of the optimization problem described by Eq. (3.4) in general. In this scenario, multiple positions of u_i's onion circuit can be observed by inference attackers (i.e., $|O| \geq 1$). This solution can be used to derive an optimal multi-hop routing algorithm that takes trust degree into account.

Intuitively, we expect the optimal solution for multiple hops can be achieved by applying the optimal single hop solution to each hop independently. That is, we expect the optimal $Pr[C_k|u_i]$ solved by Theorem 4.1 can be used for each hop $k \in O$ independently and hence lead to the minimized $E(Y[u_i|C_O])$. However, this intuitive solution does not work, because the router selections for different hops are correlated.

Figure 3.3 gives an example to show the correlated router selections in multiple hops. In this example, u_i equally trusts routers r_1, r_2 and r_3. If attackers can only observe the position 3 in u_i's onion circuit, we should use a larger probability to select r_1 than r_2 in the position 3, because r_1 is trusted by two other users (i.e., u_1 and u_2) but r_2 is trusted by only one (i.e., u_3). However, if attackers can also observe the position 2 and u_i has already selected the router r_3 in this position (i.e., $C_2 = r_3$), the attackers can deanonymize u_i immediately if u_i selects r_1 in the position 3. The reason is that, except u_i, no other users trust both r_1 and r_3 in Figure 3.3. As a result, to minimize inference attacks, the router selection algorithm for the position 3 should be different if the router readily used in the position 2 is different.

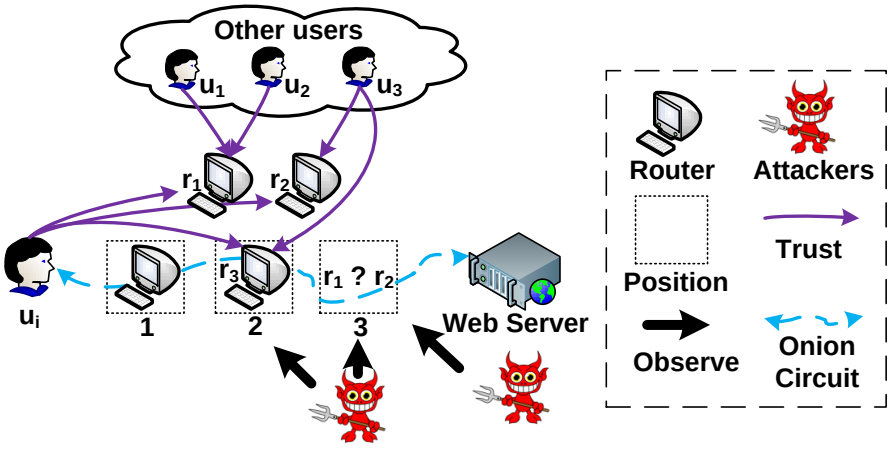


FIGURE 3.3: Router selections in multiple hops are correlated.

With no loss of generality, we consider the elements in set O are indexed in an ascending order. Let (n) be the n-th element in the set O. Let $(n-)$ be the first $n-1$ elements in the set O and $(n+)$ be the last $|O| - n$ elements in set O. As a result, $C_{(n)} = C_k$ if the n-th element in set O is the position k. Moreover, $C_{(n-)}$ denotes the sub sequence $\{C_{(1)}, C_{(2)}, \cdots, C_{(n-1)}\} \sqsubseteq C_O$, while $C_{(n+)}$ stands for the sub sequence $\{C_{(n+1)}, C_{(n+2)}, \cdots, C_{(|O|)}\} \sqsubseteq C_O$.

For example, if $O = \{1, 3\}$ (this indicates $|O| = 2$), we have the following assignments. First, $(1) = 1$ and $(2) = 3$. Second, $(1-) = \varnothing$, $(2-) = \{1\}$ and $(3-) = \{1, 3\}$. Third, $(0+) = \{1, 3\}$, $(1+) = \{3\}$ and $(2+) = \varnothing$. Fourth, $C_{(1)} = C_1$ and $C_{(2)} = C_3$. Fifth, $C_{(1-)} = \varnothing$, $C_{(2-)} = C_{(1)} = C_1$ and $C_{(3-)} = \{C_{(1)}, C_{(2)}\} = \{C_1, C_3\}$. And last, $C_{(0+)} = \{C_{(1)}, C_{(2)}\} = \{C_1, C_3\}$, $C_{(1+)} = C_{(2)} = C_3$ and $C_{(2+)} = \varnothing$.

As the router selections for multiple hops are correlated, we cannot calculate $Pr[C_O|u]$ using $Pr[C_O|u] = \prod_{k \in O} Pr[C_k|u]$, because this calculation assumes the multi-hop router selections are independent. Instead, $Pr[C_O|u]$ can be calculated in a more general version as follows.

$$Pr[C_O|u] = \prod_{n=1}^{|O|} Pr[C_{(n)}|u, C_{(n+)}]. \tag{3.6}$$

Where $Pr[C_{(n)}|u, C_{(n+)}]$ represents the conditional probability that the user u uses to select the router $C_{(n)}$ for the (n)-th position in his onion circuit in case the user u has already selected routers $C_{(n+1)}, C_{(n+2)}, \cdots, C_{(|O|)}$ for the $(n+1), (n+2), \cdots, (|O|)$-th positions in the circuit respectively.

Based the Eq. (3.6), the optimization problem that is described in Eq. (3.4) can be transformed to a general version as:

$$\min\ E(Y),\ s.t.\ \forall (n) \in O, \sum_{C_{(n)} \in R_e} Pr[C_{(n)}|u_i, C_{(n+)}] = \theta_{(n)e}. \tag{3.7}$$

Note that, since trust-based onion routing algorithms implement router selections for multiple hops according to trust independently, $Pr[C_{(n)}|u_i, C_{(n+)}] = Pr[C_{(n)}|u_i]$ if the user u_i uses trust-based routing algorithms. To derive inference attack countermeasures without tradeoffs, we find the solution of Eq. (3.7) in the isolated design space $H(O, R_e^{|O|})$, in which the $\theta_{(n)e}$ is pre-defined when u_i runs trust-based routing algorithms (i.e., $\theta_{(n)e} = \sum_{C_{(n)} \in R_e} Pr[C_{(n)}|u_i]$). This value of $\theta_{(n)e}$ is kept as a constant in the design space $H(O, R_e^{|O|})$. As a result, in the process of solving the optimization problem described in Eq. (3.7), for a given n, no matter which routers $C_{(n+1)}, C_{(n+2)}, \cdots, C_{(|O|)}$ have been readily used in the positions $(n+1), (n+2), \cdots, (|O|)$, the summarized probability that u_i uses to select routers $C_{(n)} \in R_e$ cannot be changed. That is, $\sum_{C_{(n)} \in R_e} Pr[C_{(n)}|u_i, C_{(n+)}]$ should be continuously equal to the constant $\theta_{(n)e}$ that is pre-determined by trust-based routing algorithms.

Theorem 3.6 gives the solution of the general optimization problem described in Eq. (3.7). The proof of this theorem is presented at Appendix A.2.

Theorem 3.6. *Subject to $\theta_{(n)e}, 1 \leq n \leq |O|$, the expectation $E(Y[u_i|C_O])$ that inference attackers have to deanonymize the user u_i by observing a sub sequence C_O of the onion circuit*

can be minimized to:

$$min\ E(Y) = \frac{\prod_{n=1}^{|O|} \theta^2_{(n)e}}{\prod_{n=1}^{|O|} \theta_{(n)e} + \sum_{C_O \in R_e^{|O|}} \sum_{u \in U \setminus u_i} Pr[C_O|u]}.$$

Since the selection order makes no effects to the solution of the optimization problem, we simply give the optimal distribution for $\forall(n) \in O, Pr[C_{(n)}|u_i, C_{(n+)}]$ *by considering* u_i *selects routers* C_O *in an descending order (i.e., selecting* $C_{(n)}$ *in the order from* $n = |O|$ *to* $n = 1$ *). In this case, when we consider* $Pr[C_{(n)}|u_i, C_{(n+)}]$*, the routers* $C_{(n+1)}, C_{(n+2)}, \cdots, C_{(|O|)}$ *that are used in the positions* $(n+1), (n+2), \cdots, (|O|)$ *are readily known (i.e.,* $C_{(n+)}$ *is a known value). The optimal distribution of* $Pr[C_{(n)}|u_i, C_{(n+)}]$ *for each* $(n) \in O$ *is as below.*

Given $\sum_{C_{(n)} \in R_e} Pr[C_{(n)}|u_i, C_{(n+)}] = \theta_{(n)e}$,

for $\forall C_{(n)} \in R_e$*:*

$$Pr[C_{(n)}|u_i, C_{(n+)}] \propto \sum_{C_{(n-)} \in R_e^{n-1}} \sum_{u \in U \setminus u_i} \prod_{n=1}^{|O|} Pr[C_{(n)}|u, C_{(n+)}].$$

Theorem 3.6 describes the optimal probability distributions for selecting the sequence of routers C_O from $R_e^{|O|}$. By implementing Theorem 3.6 to all the $\{R_1, R_2, \cdots, R_\nu\}^{|O|}$ respectively, u_i can have an optimal distribution for selecting the sequence of routers C_O over the whole set $R^{|O|}$.

Theorem 3.6 derives an optimal routing algorithm. This algorithm does not make any tradeoffs between the capability against inference attacks and the capability against attackers' routers. It can be embedded into existing trust-based routing algorithms to further enhance the anonymity protection.

Steps for the use of the optimal multi hops router selection algorithm: By using Theorem 3.6, users can apply the optimal multi hops router selection strategy to existing trust-based routing algorithms with four steps:

Step 1: The user u_i can first divide the set of routers R into several mutually disjoint sub sets R_es according to different trust level t_es, such as $R = \{R_1, R_2, \ldots, R_\nu\}$. Given the set of circuit positions O observed by inference attackers, u_i can choose $R_e^{|O|} \in \{R_1, R_2, \ldots, R_\nu\}^{|O|}$, where each $R_e \in R_e^{|O|}$ contains routers with the same trust level t_e from u_i's point of view (i.e., for each $r \in R_e$, $t(u_i, r) = t_e$).

Step 2: The user u_i determines $\theta_{(n)e}$ for any (n) and $R_e \in R_e^{|O|}$ according to trust-based routing algorithms.

Step 3: For each $R_e^{|O|}$, u_i finds the optimal distribution of $Pr[C_O|u_i]$ using Theorem 3.6.

Step 4: After having the optimal distribution of $Pr[C_\ell|u_i]$ for all the $R_e^{|O|} \in \{R_1, R_2, \ldots, R_\nu\}^{|O|}$, u_i can obtain the optimal distribution of $Pr[C_O|u_i]$ for $\{R_1, R_2, \ldots, R_\nu\}^{|O|}$ by concatenating the optimal distribution of $Pr[C_O|u_i]$ for all the $R_e^{|O|} \in \{R_1, R_2, \ldots, R_\nu\}^{|O|}$.

Using these steps, u_i can minimize the expectation of being deanonymized by the inference attackers who can observe the set of positions O in onion circuit. To use our algorithm, the user u_i is required to have the knowledge of which positions can be observed by attackers (i.e., the set O), as well as a priori distributions that other users have to select routers for these positions in their onion circuits (i.e., $Pr[C_{(n)}|u, C_{(n+)}]$ for $(n) \in O$ and $u \in U \setminus u_i$).

3.2.2.3 Algorithm Limitation

We use the optimal solution described in Theorem 4.1 and Theorem 3.6 as a proof of concept routing algorithm to show the effectiveness of trust degree based routing algorithm in theory, although we acknowledge this algorithm contains several limitations in practice. In this section, we describe these limitations and discuss possible solutions to work around them. This discussion gives a perspective of practical algorithms in the future.

Large requirements of a priori knowledge: The user u_i who selects routers using our algorithm is required to have the knowledge of the hypergraph $H(O, R_e^{|O|})$. This knowledge is two-fold. One is the set O and the other is $Pr[C_{(n)}|u, C_{(n+)}]$s for $(n) \in O, u \in U \setminus u_i$. We note that inference attackers are also required to have the knowledge of $Pr[C_{(n)}|u, C_{(n+)}]$s. As these inference attackers are deemed to exist, the users who have the same knowledge are certainly existing as well. In addition to $Pr[C_{(n)}|u, C_{(n+)}]$s, these users also need to know which positions of onion circuits can be observed by attackers (i.e., the set O). This requirement is practical and depends on how users construct onion circuits according to trust. For example, if users select routers with high level trust for all the positions in their onion circuits, the set O is likely to only contain the last position (i.e., $O = \{\ell\}$). However, if users adopt the downhill algorithm for router selection and the trust thresholds after the k-th position is low enough, O could contain the last $\ell - k + 1$ positions with a high probability (i.e., $O = \{k, k+1, \cdots, \ell\}$). Due to the large requirements of a priori knowledge about the onion routing network, the users who can benefit from our optimal solution could be very few (i.e., the number of users who can apply Theorem 4.1 and Theorem 3.6 is very small). They are usually the government users or security officers. Compared with normal users, these favored users could be highly capable of collecting the information about other users and require stronger anonymity protection.

Deadlock problem: We design our algorithm by investigating a user u_i in the context of a population of other users whose router selection probabilities are already known. This consideration

implicitly assumes that other users' router selection probabilities are stable. However, this assumption does not hold if there are more than one user who adopts our algorithm in the network. For example, consider two users u_i and u_i' who could select the router C_k for the k-th position of their circuits using our algorithm. The probability $Pr[C_k|u_i]$ that u_i uses to select C_k is determined according to the probability $Pr[C_k|u_i']$ that u_i' uses to select C_k, while the $Pr[C_k|u_i']$ on the other hand is calculated based on $Pr[C_k|u_i]$. Therefore, a deadlock occurs. To work around this issue, we suggest two possible solutions. First, each user u_i who adopts our algorithm can initiate $Pr[C_k|u_i]$ by assuming $Pr[C_k|u]$s for $u \in U \setminus u_i$ are calculated using trust-based algorithms at first, and then update $Pr[C_k|u_i]$ periodically afterwards. When u_i updates $Pr[C_k|u_i]$, he is required to know the $Pr[C_k|u]$s of the users u who also adopt our algorithm in current time period. This can be achieved by sharing the $Pr[C_k|u]$s among all the users u who adopt our algorithm in each period. By this way, all the users who adopt our algorithm can work together to play a race condition game, and their router selection probabilities approximate to the optimal values as time passing. Second, we can adjust our algorithm to let user u_i select routers according to other users' trust rather than other users' selection probabilities. For example, Theorem 4.1 can use $Pr[C_k|u_i] \propto \sum_{u \in U \setminus u_i} t(u, C_k)$ to replace $Pr[C_k|u_i] \propto \sum_{u \in U \setminus u_i} Pr[C_k|u]$. This method could cause the optimal solution degenerating to a near-optimal solution.

Impacts on other users: When the user u_i adopts our algorithm, he could reduce the probability of selecting some routers. Consider the probability $Pr[C_k|u_i]$ that u_i uses to select the router C_k is reduced, the other users who also trust and select C_k will become more likely to be guessed through C_k. We argue this impact in two aspects. First, the government users or security officers who have the knowledge of $H(O, R_e^{|O|})$ can directly adopt our algorithm to avoid this impact. Second, since the users who can run our algorithm in the network are few, the selection probability reduction of a particular router is usually small and therefore makes a little impact. Moreover, compared with the users who can adopt our algorithm (they are usually government users or security officers), the normal users require relatively weak anonymity protection. As a result, although these normal users cannot avoid this impact using our algorithm, they could tolerate such a little impact.

Although our optimal algorithm suffers from the three aforementioned limitations, we still choose it to investigate the effectiveness of trust degree in protecting anonymity because: 1), as a proof of concept routing algorithm, it provides the upper bound of trust degree's effectiveness in protecting anonymity; 2), these limitations can be worked around or only make tolerable impacts.

3.3 Investigating the Effectiveness of Trust Degree

In this section, we investigate the effectiveness of trust degree in resisting inference attacks using two real-world social networking datasets from [1]. We show this effectiveness by comparing $E(Y)$ between a raw routing algorithm with and without considering trust degree. The raw algorithm with considering trust degree represents the algorithm embedded with our optimal solution (see Theorem 4.1 and Theorem 3.6), while the raw algorithm without considering trust degree is the algorithm itself. In this investigation, we consider two raw algorithms. One is the traditional trust-based onion routing algorithm that is proposed to defeat correlation-like attacks by [12, 14], and the other is the downhill algorithm that is proposed to mitigate inference attacks by [14].

The organization of this section is as follows. We first describe the two real-world datasets that we use for our investigation in Section 3.3.1. We then investigate trust degree's effectiveness in enhancing the traditional trust-based routing algorithm and the downhill algorithm in Section 3.3.2 and 3.3.3, respectively.

3.3.1 Datasets

We employ two public social networking datasets [1] for our investigation. The authors of these two datasets crawled the New Orleans regional network in Facebook from December 29th, 2008 to January 3rd, 2009. One dataset contains a friendship graph and the other is an interaction graph. In a friendship graph, a directed edge represents a person is recorded in another person's friend list in Facebook. While in an interaction graph, if there is a directed edge, beside that a person must be another person's friend, these two people also need to have direct communications through social medias (e.g., post walls in Facebook). Recent studies [1, 125] argued that the interaction graph can better represent the trust among human beings than the friendship graph in the real world. However, to investigate the effectiveness of trust degree in diverse trust settings, we conduct our evaluation using both of the two graphs.

In these two datasets, we regard the start point of each directed edge as a user and the end point of the edge as a volunteer who deploys an onion router. We therefore map the friendship graph or the interaction graph to a trust graph $G = (U \bigcup R, U \times R)$, where U is the set of users and R is the set of routers (or the set of routers' owners). Note that, a person can be a user and a router's owner simultaneously (i.e., $U \bigcap R \neq \varnothing$). As discussed in Section 3.1.2, we consider two distinct levels of trust in our evaluation (i.e., $\nu = 2$). In particular, for $\forall(u, r) \in U \times R$, if the edge (u, r) exists in the dataset, we consider the user u trusts the router r (i.e., $t(u, r) = 1$). Otherwise, we regard the user u distrusts the router r (i.e., $t(u, r) = 0$).

We exclude the users who trust only one router from our evaluation, because trust-based routing algorithms cannot benefit from trust degree for these users. Table 3.2 summarizes basic statistics of our evaluated datasets which do not include the users who trust only one router.

TABLE 3.2: The basic statistics of the two datasets [1].

Dataset	# of users	# of routers	# of edges
Friendship graph	53,609	63,406	1,539,193
Interaction graph	30,988	37,467	262,684

3.3.2 Trust-based Algorithm benefits by incorporating Trust Degree

In this section, we investigate the effectiveness of trust degree by embedding our optimal solution into the traditional trust-based onion routing algorithm. This algorithm is proposed by [12, 14] and can be used to construct trust-based onion circuits to defeat correlation-like attacks. In our investigation, we consider each user in the real-world social networking datasets (listed in Table 3.2) as u_i one by one. u_i is the user in the context of a population of other users whose router selection distributions are known by u_i in advance. We show the effectiveness of trust degree by comparing $E(Y[u_i|C_O])$ between the traditional trust-based onion routing algorithm and this algorithm embedded with our optimal solution. A smaller $E(Y[u_i|C_O])$ in a particular algorithm indicates this algorithm is more effective in resisting inference attacks. For the ease of reference, we call the traditional trust-based onion routing algorithm as trust-based algorithm in the remainders of this section.

Given a user u_i, the set of routers R can be divided into two disjoint sub sets R_1 and R_2 according to the two distinct trust levels $t_1 = 1$ and $t_2 = 0$. $\forall r \in R_1, t(u_i, r) = t_1 = 1$ while $\forall r \in R_2, t(u_i, r) = t_2 = 0$. Using the trust-based algorithm, the user u_i can select the routers from R_1 uniformly at random but cannot use the routers from R_2 in any positions of his onion circuits [12, 14]. For this reason, u_i has the same probability $1/|R_1|$ for selecting each router from R_1.

When embedding our optimal solution (see Theorem 4.1 and Theorem 3.6) into the trust-based algorithm, we have $\theta_{k1} = 1$ and $\theta_{k2} = 0$ where $k \in O$. Note that, as our optimal solution enables u_i to select routers according to other users' router selection distributions (i.e., $Pr[C_O|u]$ for $\forall u \in U \setminus u_i$), we consider other users employ the trust-based algorithm for their router selections.

Figure 3.4 illustrates the distributions of $E(Y[u_i|C_O])$ when u_i uses trust-based algorithm (shown in y-axis) and this algorithm embedded with our optimal solution (shown in x-axis). Each point in the figures represents a user that we consider as u_i in the datasets. If a point is

located at the left-top side of the red diagonal line, it indicates this u_i's anonymity can be better protected against inference attacks by taking advantage of our optimal solution. It can be seen in Figure 3.4, all the points are located at the left-top side of the red diagonal lines regardless how many positions can be observed by inference attackers (we consider $|O| = 1, 2, 3$ in our investigation).

We measure trust degree's effectiveness in protecting anonymity in terms of ΔE, which can be calculated by using the $E(Y[u_i|C_O])$ in trust-based algorithm to divide the $E(Y[u_i|C_O])$ in trust-based algorithm embedded with our optimal solution.

$$\Delta E = \frac{E(Y[u_i|C_O])\ in\ trust\ based\ algorithm}{E(Y[u_i|C_O])\ in\ trust\ degree\ based\ solution}. \tag{3.8}$$

$\Delta E > 1$ indicates the trust-based algorithm that takes advantage of trust degree can better protect anonymity against inference attacks than the trust-based algorithm without considering trust degree. A larger ΔE means more effectiveness of trust degree.

As shown in Table 3.3, trust degree leads all the users to have smaller, or at least the same, $E(Y[u_i|C_O])$ compared with the trust-based algorithm which does not take advantage of trust degree (i.e., 100% users have $\Delta E \geq 1$). Moreover, for more than 99.6% users, trust degree can help them reduce $E(Y[u_i|C_O])$ (i.e., more than 99.6% users have $\Delta E > 1$). The largest reduction is up to 340.6 times (i.e., $max(\Delta E) = 340.6$). Our results confirm that trust-based algorithm can benefit a lot from trust degree in resisting inference attacks under real-world settings.

TABLE 3.3: The maximum ΔE and the percentage of users who meet a particular condition of ΔE in each graph.

Datasets and $\|O\|$	max ΔE	$\Delta E \geq 1$	$\Delta E > 1$	$\Delta E > 2$	$\Delta E > 10$
Friendship graph, $\|O\| = 1$	31.1	100%	99.6%	37.4%	0.4%
Friendship graph, $\|O\| = 2$	340.6	100%	100%	85.4%	17.1%
Friendship graph, $\|O\| = 3$	186.4	100%	100%	85.9%	7.5%
Interaction graph, $\|O\| = 1$	15.2	100%	99.8%	35.5%	0.8%
Interaction graph, $\|O\| = 2$	61.8	100%	100%	77.7%	5.7%
Interaction graph, $\|O\| = 3$	33.6	100%	100%	78.1%	0.8%

3.3.3 Downhill Algorithm benefits by incorporating Trust Degree

In this section, we investigate the effectiveness of trust degree by embedding our optimal solution into the downhill algorithm. The downhill algorithm is proposed by [14] and can be used to mitigate inference attacks by sacrificing the capability of evading attackers' routers. The same as we have done in Section 3.3.2, we consider all the users in the two datasets as to be u_i one

by one. We compare $E(Y[u_i|C_O])$ among the trust-based algorithm, the downhill algorithm and the downhill algorithm embedded with our optimal solution. A smaller $E(Y[u_i|C_O])$ in a particular algorithm means this algorithm can better protect anonymity against inference attacks.

Unlike trust-based algorithm which limits u_i to only select the routers u_i trusts (i.e., u_i has the probability to select r if and only if $t(u_i, r) = t_1 = 1$), the downhill algorithm enables u_i to select routers uniformly at random from sets with a decreasing trust threshold along onion circuits [14]. Let $t(n)$ be the trust threshold used in the position $(n) \in O$ of u_i's onion circuits, where (n) represents the n-th element in the observed position set O. Using the downhill algorithm, we have $t(n) \leq t(n')$ if the position (n') is closer to u_i than the position (n) (i.e., $(n') < (n)$). Since we consider only two distinct trust levels in our evaluation (i.e., $t_1 = 1$ and $t_2 = 0$), u_i uses the probability $1/|R_1|$ to select routers from R_1 for the position $(n) \in O$ if $t(n) = 1$, and uses the probability $1/|R|$ to select routers from $R = \{R_1, R_2\}$ for the position $(n) \in O$ if $t(n) = 0$.

When embedding our optimal solution (see Theorem 4.1 and Theorem 3.6) into the downhill algorithm, we have $\theta_{(n)1} = 1$, $\theta_{(n)2} = 0$ for the position with threshold $t(n) = 1$, and $\theta_{(n)1} = |R_1|/|R|$, $\theta_{(n)2} = |R_2|/|R|$ for the position with threshold $t(n) = 0$. Where R_1 is the set of routers with $\forall r \in R_1, t(u_i, r) = t_1 = 1$ and R_2 is the set of routers with $\forall r \in R_2, t(u_i, r) = t_2 = 0$.

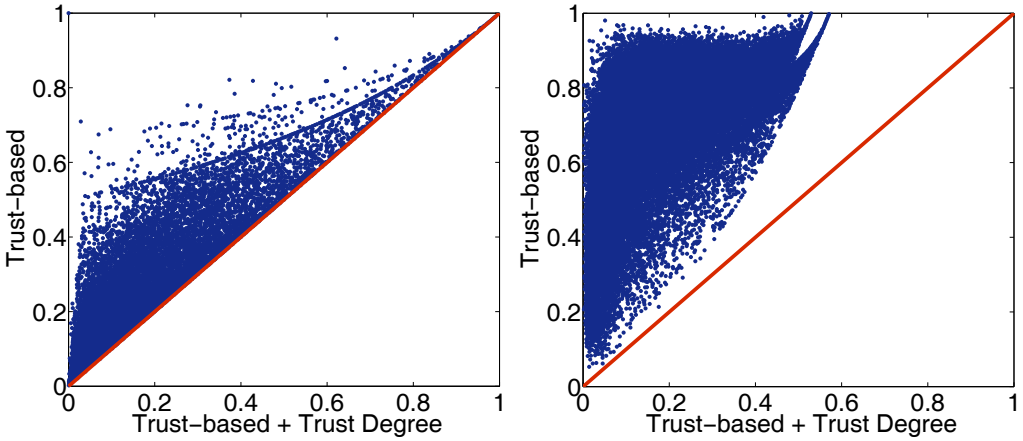
In this evaluation, we consider three positions of onion circuits can be observed by inference attackers (i.e., $|O| = 3$). According to the number of observed positions using trust threshold $t(n) = 1$, we have two different cases in our evaluation. Table 3.4 lists these two cases. The case ① has only one observed position with $t(n) = 1$ while the case ② has two observed positions with $t(n) = 1$.

TABLE 3.4: The two cases for the three observed positions.

Position in O	(1)	(2)	(3)
Thresholds in Case ①	$t(1) = 1$	$t(2) = 0$	$t(3) = 0$
Thresholds in Case ②	$t(1) = 1$	$t(2) = 1$	$t(3) = 0$

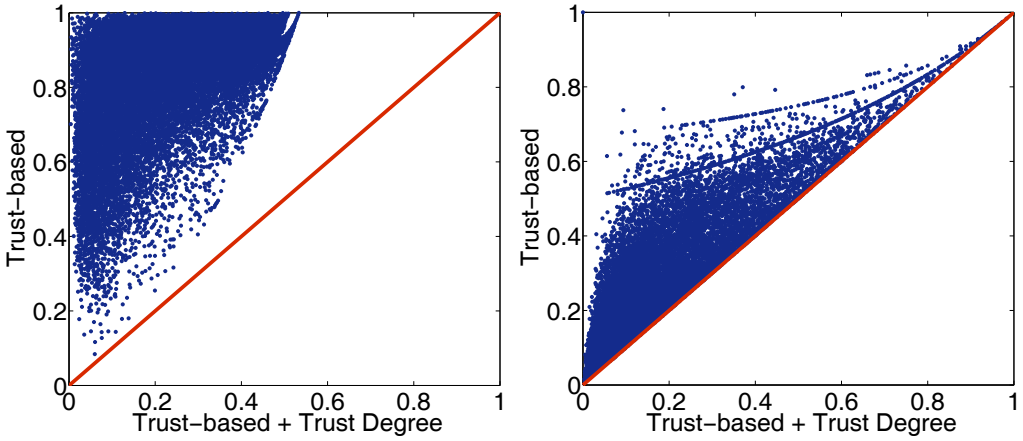
Figure 3.5 shows the $E(Y[u_i|C_O])$ in the downhill algorithm embedded with our optimal solution, the downhill algorithm and trust-based algorithm for each u_i in the two datasets listed in Table 3.2. It can be seen, although the downhill algorithm can better protect anonymity against inference attacks than trust-based algorithm, our optimal solution can be embedded into the downhill algorithm to further improve the effectiveness of resisting inference attacks. Unlike the downhill algorithm that sacrifices the capability against attackers' routers to mitigate inference attacks, our optimal solution makes no tradeoffs because it takes advantage of trust degree among equally trusted routers.

To sum up, we have successfully demonstrated that trust degree is very effective in resisting inference attacks and can be used to further enhance the protection of anonymity in trust-based onion routing under real-world settings.



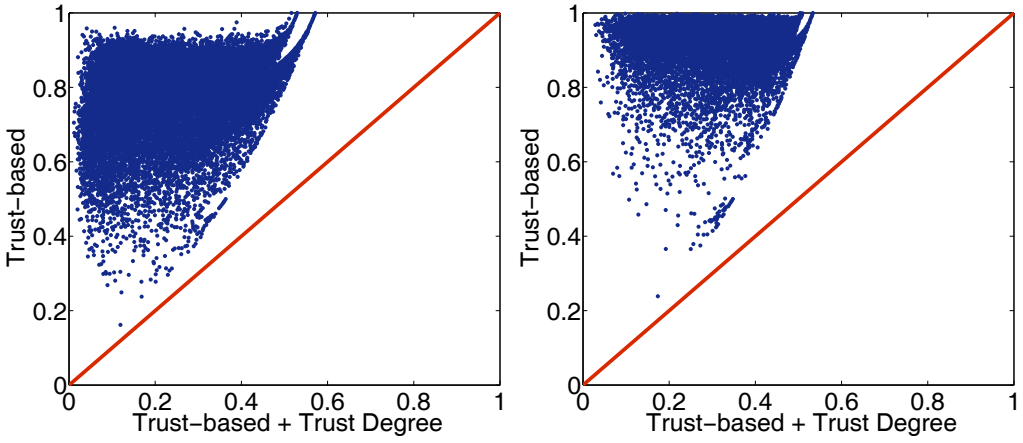
(a) One position (i.e., $|O| = 1$) observed by inference attackers in the friendship graph.

(b) Two positions (i.e., $|O| = 2$) observed by inference attackers in the friendship graph.



(c) Three positions (i.e., $|O| = 3$) observed by inference attackers in the friendship graph.

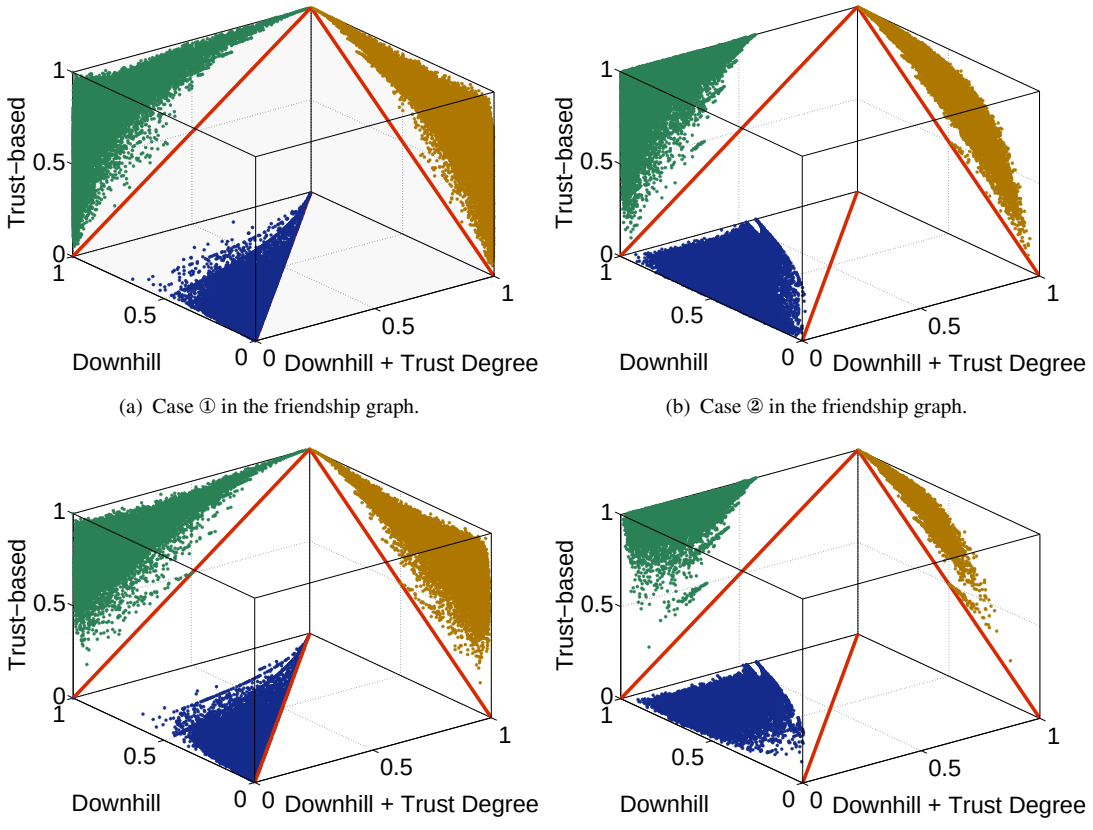
(d) Single position (i.e., $|O| = 1$) observed by inference attackers in the interaction graph.



(e) Two positions (i.e., $|O| = 2$) observed by inference attackers in the interaction graph.

(f) Three positions (i.e., $|O| = 3$) observed by inference attackers in the interaction graph.

FIGURE 3.4: The comparison of $E(Y[u_i|C_O])$ between trust-based algorithm and this algorithm embedded with our optimal solution in the two real-world social networking datasets [1]. Each point represents a u_i. y-axis indicates the $E(Y[u_i|C_O])$ when u_i adopts the trust-based algorithm while x-axis is the $E(Y[u_i|C_O])$ when u_i runs the trust-based algorithm embedded with our optimal solution. A smaller $E(Y[u_i|C_O])$ means a better protection of anonymity against inference attacks.



(a) Case ① in the friendship graph.

(b) Case ② in the friendship graph.

(c) Case ① in the interaction graph.

(d) Case ② in the interaction graph.

FIGURE 3.5: The comparison of $E(Y[u_i|C_O])$ among the trust-based algorithm, the downhill algorithm and the downhill algorithm embedded with our optimal solution in the two real-world social networking datasets [1]. Each point represents a u_i. x-axis indicates the $E(Y[u_i|C_O])$ using the downhill algorithm embedded with our optimal solution, y-axis is the $E(Y[u_i|C_O])$ using the downhill algorithm and z-axis is the $E(Y[u_i|C_O])$ using the trust-based algorithm. A smaller $E(Y[u_i|C_O])$ indicates a better protection of anonymity against inference attacks.

Chapter 4

Trust Graph based Onion Routing

In contrast to trust degree based onion routing, we also propose a more complete solution, called trust graph based onion routing, to further harden existing trust-based onion routing systems. This new trust graph based routing algorithm can address two key limitations that state-of-the-art trust-based onion routing protocols suffer from in protecting anonymity. One is the incorrect trust assignments and the other is biased trust distributions. To the best of our knowledge, no prior research has provided solutions that enhance the correctness of trust assignments in trust-based onion routing. Moreover, although several inference attack countermeasures have been proposed, the root cause of this attack has not been addressed. For example, to thwart inference attacks, a downhill algorithm [14] employs a decreasing trust threshold along onion circuits. Even our previous trust degree based algorithm takes only other users' trust into consideration. These counters cannot reduce the bias of a priori trust distributions, hence missing the opportunity to further enhance the protection against inference attacks.

We call our new algorithm SGor, a novel trust graph based onion routing that mitigates key limitations in the use of trust for protecting anonymity. SGor is designed based on two key insights. First, if people can assign trust to others according to their own knowledge independently, the trust from a group of honest people is more likely to be correct than the trust from a single honest person. Based on this observation, SGor aggregates group trust from mutual friends. The group trust enables users to verify the correctness of their trust assignments by consulting their friends' independent opinions. Second, although users have no immediate knowledge for their unfamiliar routers, these routers are not necessarily controlled by adversaries. This observation motivates us to propose an *adaptive trust propagation* algorithm. By guiding users to discover more honest routers even if they have no immediate knowledge about these routers, the proposed algorithm can derive global trust from a trust graph. The global trust leads honest routers to be trusted and selected by more users and thus impedes inference attacks by alleviating the bias of trust distributions.

Moreover, SGor is designed to run in a fully decentralized manner. This design benefits SGor in two aspects. First, our decentralized algorithms do not require users to expose their local trust relationships to other third parties (e.g., a centralized server). Since inference attacks can only be successfully launched with the knowledge of a priori trust relationships, this design assists SGor to evade inference attacks by mitigating the leakage of a priori trust relationships. Second, the removal of a centralized server can help SGor adapt efficiently to a large-scale trust graph with low costs.

To sum up, our contributions in this chapter are three-fold:

1. We propose SGor, a new onion routing protocol that protects anonymity using a trust graph.
2. We design novel decentralized algorithms to derive group trust and global trust from the trust graph. The group trust can be used to enhance the correctness of trust assignments, and the global trust is effective in reducing the bias of trust distributons. Using these new trust features, SGor mitigates key limitations in the use of trust for protecting anonymity.
3. We evaluate SGor with extensive simulation-based experiments using real-world social trust datasets. The experimental results confirm that SGor can make an effective use of the trust graph to protect anonymity but introduce only a few overheads.

The remainder of this chapter is organized as follows. We first present a high level overview of SGor in Section 4.1. After elaborating on the design of SGor in Section 4.2, we evaluate it using real-world social trust datasets in Section 4.3.

For the ease of reference, we summarize important notations used by this chapter in Table 4.1.

4.1 SGor Overview

In this section, we describe SGor in a high level. We first set design goals for SGor in Section 4.1.1. We then list basic assumptions in Section 4.1.2. After elaborating on threat model and trust model in Section 4.1.3 and Section 4.1.4 respectively, we present an overview of SGor with its architecture and major components in Section 4.1.5.

4.1.1 Design Goals

To effectively protect anonymity using trust, SGor is expected to meet the following four key requirements:

TABLE 4.1: Important Notations in Chapter 4.

Notation	Definition
G	A trust graph. $G = (V, E)$
V, E	V is a set of nodes in G, E is a set of edges in G
Φ_{ij}	The *group trust* that a person v_i has in another person v_j
Φ_h	A minimum group trust threshold
t_{ij}	A random token the person v_i generates and sends to his friend v_j
C_{ij}	The number of routers to which v_i can propagate trust through the friend v_j
L_i	The maximum number of hops to which v_i can propagate trust
R_i	The number of routers a person v_i expects to collect through trust propagation
T_{ij}	The number of v_i's tickets that the person v_j receives
$P(A_j\|\Phi_{ij})$	The probability that v_j is adversary given v_i has group trust Φ_{ij} for v_j
r_j	The number of routers that the person v_j deploys in SGor
$P(A_j\|L_i)$	The probability that v_j is adversary given v_i propagates trust to v_j through L_i hops
λ	$P(\Phi_{ij}\|A_j)$ follows a Poisson distribution with a parameter λ
β	$P(A_j)$ is independent and identically distributed with a parameter $\beta \in (0, 1)$

Trust Graph: To evade malicious routers and thwart correlation-like attacks, SGor is required to have the capability of constructing onion circuits using trust from a trust graph.

Group Trust: To verify the correctness of trust assignments, SGor is required to have the capability of aggregating group trust from mutual friends.

Global Trust: To reduce the bias of trust distributions and hence defeat inference attacks, SGor is required to have the capability of deriving global trust from a trust graph.

Decentralization: To prevent users' trust relationships from leaking to any third parties and adapt efficiently to a large scale trust graph, SGor is required to have a fully decentralized architecture.

These requirements work together to distinguish SGor from past works in the literature [11–15].

4.1.2 Basic Assumptions

We make three basic assumptions for the design of SGor.

1. Users and routers' owners in an onion routing network are also members of a trust network.

2. Users and routers' owners can assign trust to others according to their own knowledge independently.

3. Users and routers' owners can run user-agents in their own machines (e.g., in-browser add-ons, desktop or mobile phone installs) to support SGor's decentralized router selection process in a fully automatic manner.

Assumption 1 defines the potential population who can use SGor. For example, if we derive a trust graph from an online social network and apply it to SGor, the members of this online social network can use SGor to protect their anonymity when they visit Internet services. As reported by SocialBakers (www.socialbakers.com) in July 2012, Facebook, the most popular online social network, has more than 870 million members. Moreover, there are more than 2.2 billion Internet users in the world according to a report from InternetWorldStats (www.internetworldstats.com) in December 2011. Apparently, if SGor adopts Facebook as its trust graph, at least 40% Internet users who have accounts in Facebook can use SGor when they access Internet through onion routing.

With assumption 2, SGor is built upon a trust graph with independent trust assignments. If users can assign trust to others independently, they can avoid the influence of incorrect trust over a group of people. Otherwise, the group trust could have weak capability of verifying the correctness of trust assignments. For example, if Bob assigns trust to Pete due to the reason that Bob's friend Alice has already trusted Pete (i.e., Bob's trust on Pete is correlated with Alice), Alice cannot rely on the group trust from Bob to verify the correctness of her trust in Pete, because Bob's trust is inherited from Alice. In real world, the friendship graph of an online social network is more likely to have correlated trust assignments, because people usually make friends by consulting their old friends' friendship circles. Unlike that, the interaction graph [1, 125] admits trust assignments if and only if users have direct communications, hence being more independent. For this reason, we evaluate SGor by adopting the interaction graph in Section 4.3. We acknowledge that, even we adopt interaction graph, it is still very difficult to have completely independent trust assignments, since collusion attacks are popular in many social context and their defenses are also well studied. For this reason, we could adopt some typical solutions to address this problem, although it is out of the scope of this thesis.

With assumption 3, users can select onion routers from others and onion routers' owners can provide onion routers to others in a fully decentralized and automatic manner. For example, if SGor adopts the interaction graph of Facebook as its trust graph, we can develop a Facebook App (developers.facebook.com) for SGor. All the users and routers' owners run this App in their own machines. The running Apps communicate with each others in the Internet to accomplish the decentralized router selection process automatically. The communication traffic is encrypted to prevent potential leakage.

4.1.3 Threat Model

To compromise users' anonymity protected by (trust-based) onion routing networks, we consider adversaries have the following attacking capabilities:

The capability of compromising onion routers: First, we assume adversaries can arbitrarily deploy their own malicious routers to existing onion routing networks. This assumption holds in real-world onion routing networks such as Tor [9], because these networks are open to accept any volunteer's routers without checking their identities. Although some networks have runtime monitoring systems to exclude malicious routers, these systems are usually based on the uptime of normally behaving. As a result, smart adversaries can easily bypass these monitoring systems by operating their routers in normal behaviors for a long period. Second, we assume adversaries can exploit hacking techniques to compromise some honest routers which contain security vulnerabilities. This assumption adapts to advanced adversaries who master powerful computer network skills. Third, we simply assume that Internet services visited by onion routing users are already controlled by adversaries. This assumption allows us to design SGor by considering the worst scenario, because users can visit any Internet servers through onion routing and it is nearly impossible to confirm which servers are compromised and which are not.

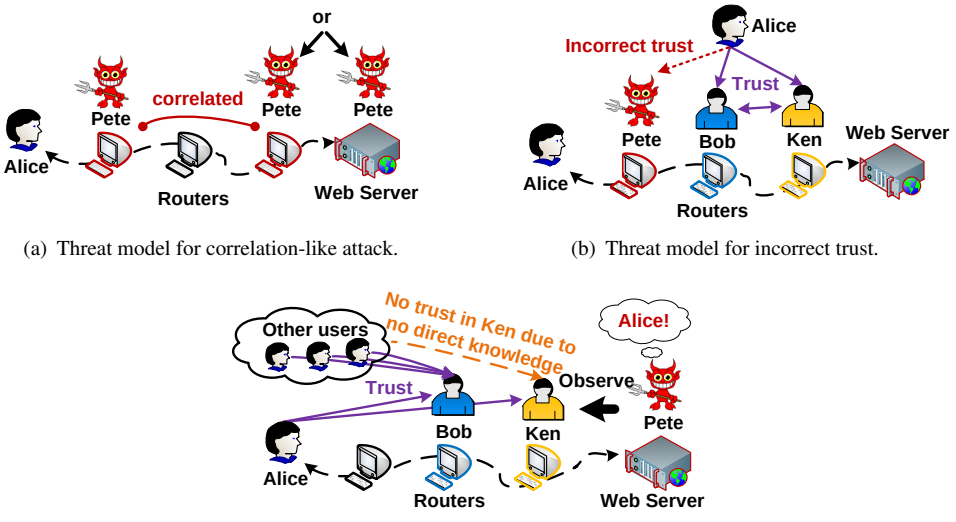
The capability of locally observing onion routers: We assume that adversaries can observe onion routers in users' onion circuits. This capability is required by the adversaries who perform inference attacks. However, since existing onion routing protocols are not designed to resist global adversaries who can monitor the whole communication infrastructure of onion routing networks (e.g., ISP-level or state-level adversaries) [7, 9], we also assume that adversaries can only employ compromised onion routers to observe adjacent routers in onion circuits or use compromised Internet servers to observe the last router in onion circuits.

The capability of correlating malicious onion routers: We assume that adversaries can correlate the onion routers and Internet servers under their control in the same onion circuit (i.e., the capability of performing correlation-like attacks). This assumption holds in real-world attacking scenarios, because several prior studies have demonstrated that various traffic watermarking and analysis techniques are effective in correlating two compromised endpoints (e.g., onion routers or Internet servers) in the same onion circuit [24–26, 126–128]. Figure 4.1(a) illustrates an example of the correlation-like attack against onion routing networks. In this example, Alice is a user who has no effective method to verify router identities. Pete is an adversary who control onion routers. If Alice selects Pete's routers as the first router in her onion circuit and visits an Internet server under Pete's control, Pete can correlate Alice and the server Alice is visiting.

The capability of attracting incorrect trust: We assume that adversaries can exploit users' inaccurate knowledge to attract incorrect trust assignments. Since trust-based onion routing computes trust according to users' own knowledge about routers' owners, if users mistakenly

trust an adversary, they could trust all the routers controlled by this adversary. Figure 4.1(b) shows an example of an incorrect trust assignment in trust-based onion routing. If Alice makes a mistake to trust an adversary Pete, she could select Pete's routers in her onion circuit. Moreover, although we agree that adversaries can easily deploy a large number of fake accounts in trust graph (i.e., Sybil attack [45]), it is very difficult for them to gain incorrect trust from a large number of honest humans. Previous social network based Sybil defences also adopt this assumption because this assumption widely holds in real-world social networks [44].

The capability of collecting a priori trust relationships: We assume that adversaries can collect a priori trust relationships of a trust graph. For example, if we apply an online social network as a trust graph to SGor, adversaries could employ a crawler to run a brute scan of this social network. This approach can reveal the local trust relationships of users who open their private information to the public [123, 129]. To obtain the trust relationships of users who could only expose their private information to their friends, adversaries could employ socialbots [130] to make friends with these users, or estimate these users' relationships using the leakage of their friends' trust relationships (i.e., link privacy leakage [131]). Using a priori trust relationships, adversaries can perform inference attacks and hence largely reduce users' anonymity protected by trust-based onion routing. Figure 4.1(c) demonstrates an example of inference attack against trust-based onion routing. In this demonstration, Pete is an adversary who knows a priori trust relationships. If Pete observes Ken's routers in an onion circuit, he can guess Alice as the initiate user of this circuit because Ken is only trusted by Alice. However, Ken's routers are honest in fact. Other users do not trust Ken just because they have no knowledge about Ken. It is not due to that they confirm Ken is an adversary.



(a) Threat model for correlation-like attack.

(b) Threat model for incorrect trust.

(c) Threat model for inference attack.

FIGURE 4.1: Threat models.

4.1.4 Trust Model

We model SGor's trust graph as a directed graph $G = (V, E)$. V is the set of nodes in G. A node $v_i \in V$ represents a person in the trust graph. E is the set of edges in G. An edge $v_i \rightarrow v_j = e_{ij} \in E$ indicates that the person $v_i \in V$ assign local trust to another person $v_j \in V$ in the trust graph (i.e., a trust edge represents a local trust assignment). $|V|$ and $|E|$ are the numbers of nodes and edges in G, respectively. As discussed in [12, 14], users can only assign coarse level trust to others. SGor therefore only considers two levels of local trust assignments, *trust* and *distrust*. If $v_i \rightarrow v_j$ exists, v_i trusts v_j. Otherwise, v_i distrusts v_j. Though this binary trust definition is enough for our design, we note that SGor can be extended to a continuous trust definition (e.g., trust value is a real number between 0 and 1). We will investigate this extension in our future work. Moreover, we consider the trust is positively transitive [132]. That means a friend of Alice's friend is still a friend of Alice.

We can sample SGor's trust graph G using any real-world trust networks. For example, we can derive a social trust graph from an online social network or adopt a reputation-based trust graph from an online community. We can also design an open community that facilitates users and routers' owners to evaluate the trustworthiness of each other independently, hence generating a customized trust graph to support SGor. In our research, we mainly focus on the design of SGor using online social networks because this choice makes SGor easy to be applied in the Internet.

In the literature, there are two acknowledged models that can be used to sample the trust graph G from an online social network. One is the friendship model and the other is the interaction model [1, 125]. They have different capability to express the trust that already exists in online social networks. We take an example from Facebook to show the difference. We first assume Alice and Ken are two persons with accounts in Facebook. If Alice attempts to make a friend with Ken, she sends a *friend request* to Ken. If Ken confirms this request, they are shown in each other's friendship list. The friendship model take this kind of relationship as trust. The interaction model is built on top of the friendship model. If Alice has local trust with Ken in an interaction model, only being a friend of Ken is not enough. Alice also needs bi-directional communications with Ken through online social networks (e.g., posting messages with each other in walls or tagging each other in photos).

SGor employs the interaction graph to sample G because of two reasons. First, the local trust in the interaction graph is more independent. A person may be easy to make a new friend who is already a friend of his old friends, but they are less likely to have bi-directional communications with each other if they are not acquaintances. Second, recent research has advocated that interaction graphs can better represent real world trust than friendship graphs [1, 125].

For the ease of description in the following sections, although we do not use the friendship graph to sample G, we also regard v_j as v_i's friend if v_i assigns local trust to v_j (i.e., $v_i \rightarrow v_j$ exists).

4.1.5 SGor Architecture and Major Components

In this section, we gives a high level overview to introduce SGor's architecture and major components.

4.1.5.1 SGor Architecture

SGor provides trust graph based onion routing by "overlaying" a trust network on top of the onion routing infrastructure. SGor has a two layered hierarchical architecture. The upper layer is for trust graph based router selection, and the underlayer is for onion routing. To communicate with remote Internet servers through SGor, users should first select onion routers according to trust in the trust graph layer. They then use the selected routers to establish onion circuits in the onion routing layer. We note that SGor is novel due to its trust graph layer. The onion routing layer runs the same protocol as traditional onion routing networks.

In SGor, each person in the trust network layer can arbitrarily play any roles in the onion routing layer. In particular, a person can act as onion routing user who requires onion routers from others. This person can also operate as a router owner who provides onion routers to others. Moreover, a person can work as both of an onion routing user and a router owner concurrently, or play neither of the two roles in the onion routing layer. In the last case, the person merely assists other people to propagate trust over the trust graph.

Figure 4.2 illustrates an example of the architecture of SGor. In this example, a person v_1 attempts to visit a (sensitive) Internet server through SGor. People v_2-v_6 provide onion routers in SGor. v_7 and v_8 are two adversaries. In the trust graph layer, although v_1 makes a mistake to trust the adversary v_7, SGor has a chance to exclude v_7 because no other v_1's friends trust v_7 (i.e., no other ways can be used to verify the correctness of $v_1 \rightarrow v_7$). Moreover, although v_1 only has local trust in v_2 and v_3, SGor can propagate v_1's trust to other persons v_4-v_6 in the trust graph. Through v_2, v_1's trust can be propagated to v_4 and v_5, and further to v_6. However, for security concern, v_3 cannot propagate v_1's trust to v_8 because no means can be used to verify the correctness of the local trust $v_3 \rightarrow v_8$. In the onion routing layer, v_1 constructs onion circuits using onion routers derived from the trust graph layer. Since SGor calculates trust features in the trust graph layer, onion routing users can evade all the onion routers deployed by the person whom they do not trust (e.g., v_8 has two routers in Figure 4.2. If v_1 does not trust v_8, he can circumvent the two routers deployed by v_8).

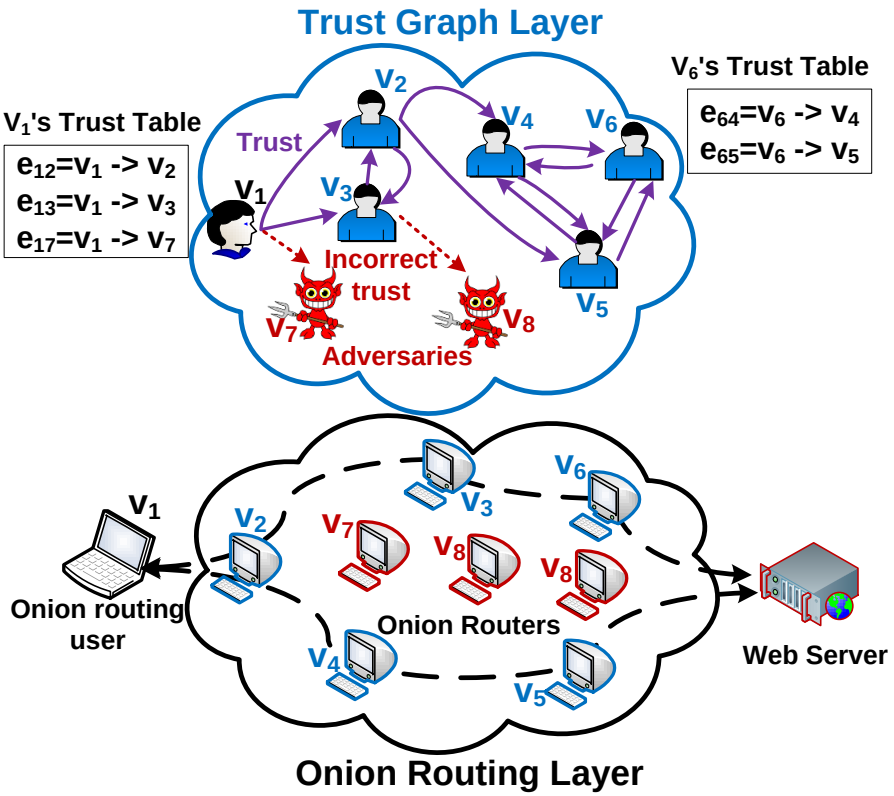


FIGURE 4.2: Two layered hierarchical architecture of SGor.

4.1.5.2 Major Components

SGor consists of three major components in its trust graph layer: trust management, trust aggregation and trust propagation. They work together to meet aforementioned design requirements for SGor (Refer to Section 4.1.1).

Trust Management: This component provides functionalities for managing local trust relationships. Since SGor is designed to run without any central servers, SGor users have to manage local trust relationships for themselves. As shown in Figure 4.2, each SGor user has a trust table to store local trust for their friends. This design results in two advantages. First, this design does not require any people to expose their local trust to any others, hence mitigating the leakage of a priori trust relationships. Second, this design adapts SGor to a large scale trust graph with a low cost in the storage. Rather than all the trust relationships in the entire trust graph, each SGor user is only required to store trust relationships for their friends.

Trust Aggregation: This component aggregates group trust from mutual friends, hence offering countermeasures to remove incorrect trust assignments. A person's friend receives group trust when this friend is also trusted by many other friends. The group trust can be used to remove incorrect trust assignments, because people can verify the correctness of their local trust assignments by consulting other friends' independent opinions. This component has a basic function to aggregate group trust from mutual friends. Moreover, since people could join and leave the

trust network dynamically, and update their local trust assignments when their knowledge is renewed, this component also provides an additional function to handle this dynamic behavior.

Trust Propagation: This component computes global trust by propagating local trust though trust graph, hence leading honest routers to be trusted by more users. This component plays the key role in reducing the bias of trust distribution and hence resisting inference attacks. We implement an *adaptive trust propagation* algorithm for this component. Using this algorithm, a person can propagate trust from the people who trust him to the other people whom he trusts. For example, Alice trusts Bob and Bob trusts Ken. If Bob propagates the trust from Alice to Ken, Alice could also trust Ken. Moreover, to mitigate the risk of mistakenly propagating trust to adversaries, SGor takes a counter that adapts trust propagation capacity to group trust (i.e., people use group trust to limit the maximum number of people to whom they can propagate trust through a friend). Back to the foregoing example, if Alice has a higher level group trust in Bob, she can propagate trust through Bob to more other people. But if Alice has a lower level group trust in Bob, she could just propagate trust through Bob to fewer, or even none, other people. By adaptively propagating trust over the entire trust graph, people can discover more honest routers and thus mitigate the bias of trust distributions. In return, these honest routers can be trusted and selected by more users and become more effective in thwarting inference attacks.

4.2 SGor Design

In this section, we elaborate on the design of SGor. First, we apply an algorithm to aggregate group trust from mutual friends in Section 4.2.1. Second, we propose an adaptive trust propagation algorithm to derive global trust from trust graph in Section 4.2.2. Based on these two new trust features, we design a trust graph based router selection algorithm in Section 4.2.3. All our proposed algorithms run in a fully decentralized manner. We also analyze SGor's capability of protecting anonymity using a probabilistic model in Section 4.2.4.

4.2.1 Group Trust

SGor employs group trust to verify the correctness of local trust assignments. In this section, we discuss the details for aggregating group trust. We first describe the concept of robust trust path by analyzing security concerns of trust path. We then give the definition of group trust based on robust trust path. Afterwards, we design a decentralized algorithm to aggregate group trust on top of a trust graph.

4.2.1.1 Robust Trust Path

In trust graph $G = (V, E)$, we consider v_i has a trust path to v_j if v_i can reach v_j through a sequence of successive trust edges. A trust path from v_i to v_j implicitly indicates that v_i trusts v_j, hence providing an unique way for v_i to confirm that v_j is not an adversary (the local trust $v_i \rightarrow v_j$ is correct).

We consider that a trust path is robust if this path cannot be arbitrarily forged through a single incorrect trust edge. However, not all the trust paths are necessarily robust. For example, if v_i assigns incorrect local trust to two adversaries v_j and v_k (i.e., $v_i \rightarrow v_j$ and $v_i \rightarrow v_k$ exist in G), the adversary v_k can forge unlimited number of trust paths from v_i to v_j because adversaries can arbitrarily trust other adversaries. Figure 4.3 demonstrates forged trust paths. v_1 has assigned an incorrect local trust to v_2 (i.e., $v_1 \rightarrow v_2$ is incorrect). If v_3 is also an adversary (i.e., $v_1 \rightarrow v_3$ is incorrect), he can arbitrarily forge trust paths from v_1 to v_2. As shown in this figure, the adversary v_3 has forged 2, 3 and 4 trust paths from v_1 to v_2.

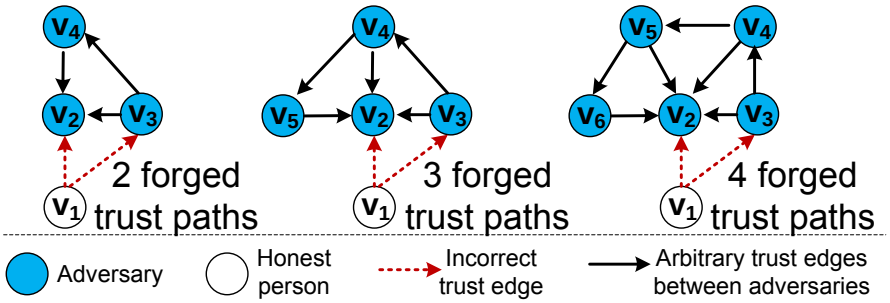


FIGURE 4.3: Examples of forged trust paths.

Here, we find that the trust paths consisting of no more than two trust edges are robust. We give an example to explain this finding. Considering two trust paths from v_i to v_j, $v_i \rightarrow v_k \rightarrow v_j$ is a robust trust path while $v_i \rightarrow v_k \rightarrow v_m \rightarrow v_j$ is not. The reason is that, if v_k is an adversary, he can arbitrarily trust any other adversaries like v_m, hence forging an unlimited number of trust paths like $v_i \rightarrow v_k \rightarrow v_m \rightarrow v_j$. However, the adversary v_k can only forge one robust trust path $v_i \rightarrow v_k \rightarrow v_j$ from v_i to v_j.

Since robust trust paths cannot be arbitrarily forged by adversaries (i.e., one incorrect trust edge can only corrupt one robust trust path), they can provide robust ways to confirm that a local trust assignment is correct.

4.2.1.2 Group Trust Definition

Let Φ_{ij} be the *group trust* v_i has in v_j. Φ_{ij} can be calculated by counting up the number of robust trust paths from v_i to v_j. We have $\Phi_{ij} = 0$ if the trust edge $v_i \rightarrow v_j$ does not exist.

If v_i is an honest person but v_j is an adversary, Φ_{ij} reflects the number of incorrect trust edges accompanied with the incorrect trust edge $v_i \rightarrow v_j$. Theorem 4.1 proves this nature.

Theorem 4.1. *If v_i is an honest person, v_j is an adversary and $\Phi_{ij} = N$, there must exist N incorrect trust edges in the robust trust paths from v_i to v_j.*

Proof. We use mathematical induction for the proof.

Base case: Consider $\Phi_{ij} = 1$, v_i can only have one robust trust path to v_j, i.e., $v_i \rightarrow v_j$. Meanwhile, this robust trust path consists of only one trust edge $v_i \rightarrow v_j$. As a result, if v_j is an adversary, we have 1 incorrect trust edge $v_i \rightarrow v_j$.

Inductive step: Assuming Theorem 4.1 holds for $\Phi_{ij} = N$, we show that Theorem 4.1 also holds for $\Phi_{ij} = N + 1$. Compared with $\Phi_{ij} = N$, v_i has an additional robust trust path to the adversary v_j when $\Phi_{ij} = N + 1$. We can simply assume this additional path is $v_i \rightarrow v_k \rightarrow v_j$. We consider two cases for this path: (i) v_k is an adversary and (ii) v_k is an honest person. For case (i), $v_i \rightarrow v_k$ is an additional incorrect trust edge. While for case (ii), $v_k \rightarrow v_j$ is incorrect because v_j is an adversary. As a result, if $\Phi_{ij} = N$ indicates N incorrect trust edges in the robust trust paths from v_i to the adversary v_j, $\Phi_{ij} = N + 1$ can lead to $N + 1$ incorrect trust edges. □

Figure 4.4 gives examples for Theorem 4.1. It can be seen that if $\Phi_{12} = 1$ and v_2 is an adversary, we have only 1 incorrect trust edge, i.e., $v_1 \rightarrow v_2$. However, if $\Phi_{12} = 2$, 2 incorrect trust edges are accompanied. In particular, $v_1 \rightarrow v_2$ and $v_3 \rightarrow v_2$ are incorrect if v_3 is an honest person, or $v_1 \rightarrow v_2$ and $v_1 \rightarrow v_3$ are incorrect if v_3 is an adversary. Similarly, $\Phi_{12} = 3$ confirms that there are 3 incorrect trust edges.

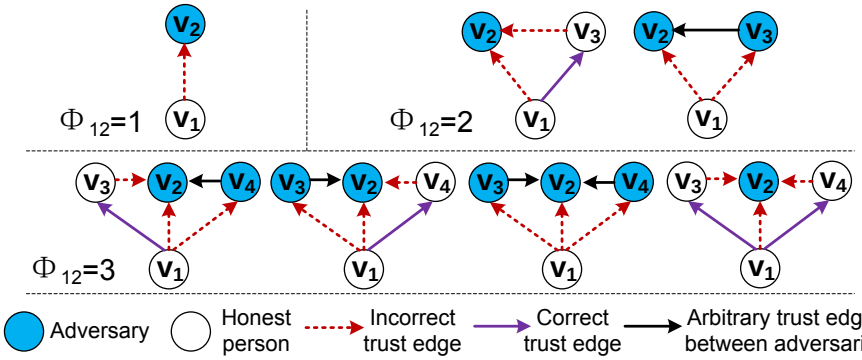


FIGURE 4.4: If v_1 is an honest person but v_2 is an adversary, the group trust Φ_{12} equals to the number of incorrect trust edges in the robust trust paths from v_1 to v_2.

Since group trust Φ_{ij} is the total number of robust trust paths from v_i to v_j, a larger Φ_{ij} indicates that v_i can obtain more robust means to confirm that v_j is not an adversary. SGor therefore use Φ_{ij} to validate the correctness of $v_i \rightarrow v_j$. In the design of SGor, users can set a minimum group

trust threshold Φ_h, and have more confidence to confirm that $v_i \rightarrow v_j$ is correct (i.e., v_j is not an adversary) if $\Phi_{ij} \geq \Phi_h$.

4.2.1.3 Group Trust Aggregation Algorithm

SGor applies a decentralized algorithm to aggregate the group trust Φ_{ij} by counting the number of v_i's friends who have v_j as a mutual friend of v_i. This algorithm is based on two observations. First, if v_i has a robust trust path $v_i \rightarrow v_k \rightarrow v_j$ through v_k to v_j, v_j is a mutual friend of v_i and v_k. Second, the group trust Φ_{ij} is the total number of robust trust paths from v_i to v_j.

This group trust aggregation algorithm runs three steps in a fully decentralized manner. The communications between any two people in the trust graph layer are assumed to be encrypted.

Step 1: At first, an initiate person generates random tokens on the fly, and maps different tokens to different friends. The initiate person sends each token to this token's mapped friend. Since every token is marked by the initiate person, the receivers can confirm these tokens are initial tokens because they are sent and marked by the same person. As shown in Figure 4.5, v_1 has trust edges to v_2, v_3, v_4 and v_5. To aggregate group trust for these friends, v_1 generates and maps random tokens t_{12}, t_{13}, t_{14} and t_{15} to v_2, v_3, v_4 and v_5, respectively. v_1 then sends these initial tokens to their mapped friends (i.e., sending t_{1x} to v_x, where $x = 2, 3, 4, 5$).

Step 2: When a person receives an initial token, he will further forward this token to his friends. These forwarded tokens are sent and marked by different people. Receivers regard them as second hand tokens. As shown in Figure 4.5, v_2 forwards t_{12} to v_3, while v_3 and v_5 forward t_{13} and t_{15} to v_4. Actually, v_2, v_3, v_4 and v_5 should forward the tokens received from v_1 to all of their friends, even if these friends are not v_1's friends. We omit this process in Figure 4.5 to make a clear demonstration.

Step 3: When a person receives second hand tokens, he can take actions depending on whether he has already received an initial token with the same marker of the second hand tokens. If it is not the case, this person will discard these second hand tokens. Otherwise, he will send the initial token and all the second hand tokens back to the initiate person who marks them. The initiate person calculates the group trust for each of his friends by counting the number of tokens returned from these friends. In Figure 4.5, v_2, v_3, v_4 and v_5 return 1, 2, 3 and 1 tokens to v_1, respectively. Hence, v_1 obtains $\Phi_{12} = 1$, $\Phi_{12} = 2$, $\Phi_{13} = 3$ and $\Phi_{14} = 1$.

Since the proposed group trust aggregation algorithm operates in a fully decentralized manner, there are three scenarios that a person has to start running this algorithm: (i), a fresh person first joins SGor; (ii), a person changes local trust assignments to his friends; (iii), a person has friends who change their local trust assignments to these friends' friends.

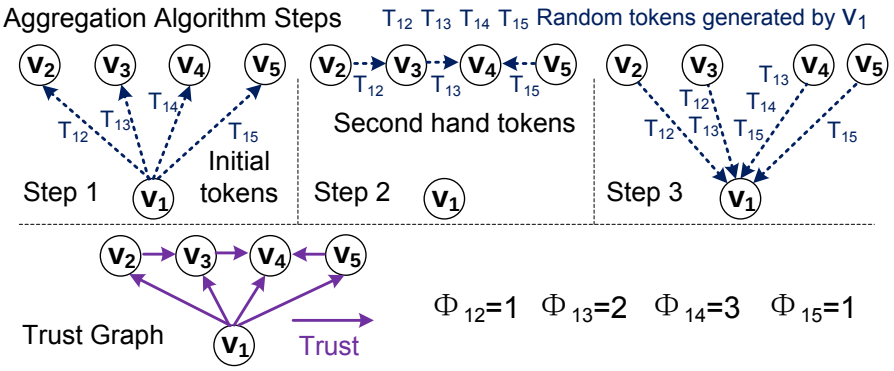


FIGURE 4.5: The Steps of Group Trust Aggregation Algorithm.

4.2.2 Global Trust

SGor employs global trust from trust graph to thwart inference attacks. We consider a person v_i has global trust in another person v_j if v_i has at least one trust path to v_j. Using global trust, onion routing users can trust routers even if they have no direct knowledge for these routers' owners.

SGor derives global trust through trust propagation. Considering people v_i and v_j, v_i has global trust in v_j if and only if v_i can propagate his trust to v_j through a trust path. However, if people naively propagate trust through any trust paths without any constraints, they cannot limit the number of adversaries who receive global trust during trust propagation. For example, if v_k can propagate v_i's trust but v_k is an adversary, v_k can arbitrarily propagate v_i's trust to any other adversaries.

To address this problem, we propose an adaptive trust propagation algorithm. The key idea is to limit the number of people to whom a person can propagate trust through a friend. The limitation is determined by the group trust that this person has in this friend. This algorithm defends trust propagation against adversaries in two aspects. First, even if an honest person mistakenly propagates trust to an adversary, this adversary can only further propagate trust to a limited number of other adversaries. Second, since a larger group trust received by a friend indicates a lower probability that this friend is an adversary, the friend who is less likely to be an adversary can propagate trust to more others than the friend who is more likely to be an adversary.

We denote the trust propagation capacity C_{ij} as the number of routers (note that one person could deploy multiple routers) to which v_i can propagate trust through the friend v_j. In our proposed adaptive trust propagation algorithm, C_{ij} must be proportionate to Φ_{ij}, such as:

$$C_{ij} = \lceil \frac{\Phi_{ij}}{\sum_{\Phi_{ik} \geq \Phi_h} \Phi_{ik}} \cdot \sum_{\Phi_{ik} \geq \Phi_h} C_{ik} \rceil. \tag{4.1}$$

where, $\lceil * \rceil$ is a ceiling function. Φ_h is a group trust minimum threshold. SGor employs Φ_h to filter out ineligible friends because they are more likely to be wrongly trusted. $\sum_{\Phi_{ik} \geq \Phi_h} C_{ik}$ is the total number of routers to which v_i can propagate trust through eligible friends, while $\sum_{\Phi_{ik} \geq \Phi_h} \Phi_{ik}$ is the sum of group trust that v_i has in the eligible friends.

Using the adaptive trust propagation algorithm, if an adversary v_k can propagate v_i's trust, he can only propagate v_i's trust to at most C_{ik} other adversaries.

4.2.3 Trust Graph based Router Selection

Based on group trust and global trust, SGor proposes a novel trust graph based router selection algorithm. This algorithm has three parameters to be set in advance. One is group trust minimum threshold Φ_h. The other two are R_i and L_i. The parameter R_i determines the number of candidate routers that a person v_i expects to collect through trust propagation. v_i constructs onion circuits by selecting routers uniformly at random from these R_i candidate routers. The parameter L_i limits the maximum number of hops to which v_i can propagate trust.

SGor implements the trust graph based router selection algorithm using a ticket distribution mechanism. This algorithm consists of three steps:

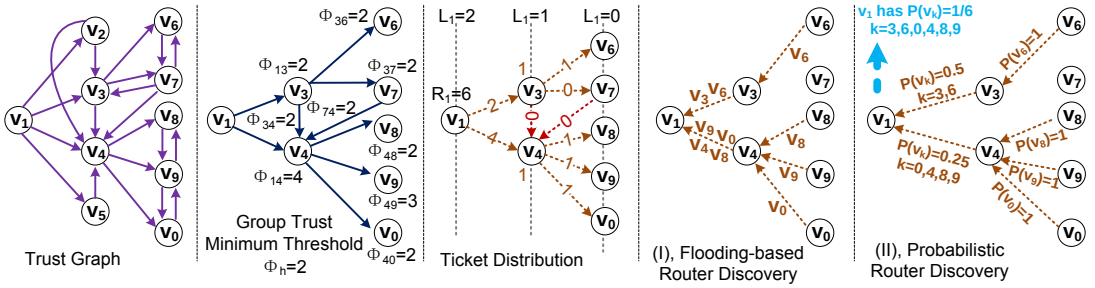


FIGURE 4.6: An example of trust graph based router selection.

Initial Step: When a person v_i (i.e., an onion routing user) attempts to construct onion circuits, he should create R_i tickets at first. Each ticket has a time-to-live field which specifies the maximum number of hops the ticket can be transmitted. v_i initializes this field to L_i. Afterwards, v_i distributes these R_i tickets to v_i's friends in whom v_i has group trust no smaller than Φ_h. The amount of tickets that can be distributed to each eligible friend is calculated according to Eq. (4.1), where $\sum_{\Phi_{ik} \geq \Phi_h} C_{ik} = R_i$.

Ticket Distribution Step: If a person v_j receives T_{ij} v_i's tickets, v_j consumes r_{ij} tickets to provide r_{ij} candidate routers to v_i. v_j determines r_{ij} depending on T_{ij} and r_j (i.e., r_j is the amount of onion routers that v_j deploys in SGor). If $T_{ij} \leq r_j$, $r_{ij} = T_{ij}$ and v_j stops the ticket distribution. Otherwise, $r_{ij} = r_j$. For the remaining $T_{ij} - r_j$ tickets, v_j first sets $L_i = L_i - 1$. If $L_i = 0$, v_j discards these remaining tickets and stops the ticket distribution. Otherwise, v_j will

further distribute the remaining tickets to v_j's eligible friends who will repeat the ticket distribution step. The amount of tickets distributed to each of v_j's eligible friends is also computed using Eq. (4.1), where $\sum_{\Phi_{jk} \geq \Phi_h} C_{jk} = T_{ij} - r_j$. When distributing v_i's tickets, the ticket distribution step is performed by different people in parallel. Hence, some people could receive v_i's tickets more than one time (i.e., the duplicated ticket distribution).

To eliminate the duplicated ticket distribution, people should only distribute tickets to the eligible friends who have not received v_i's tickets before. The intuitive mechanism is to allow people querying their friends whether they have already received v_i's tickets before they distribute tickets. However, this mechanism is not resilient to attacks. Adversaries can simply claim they have never received v_i's tickets and repeatedly defraud tickets.

Here, we propose a *regressive checking mechanism* to effectively avoid both the duplicated ticket distribution and the ticket frauds. Using this mechanism, the ticket owner v_i maintains a list that records the people who have already received v_i's tickets. For each person v_j who attempts to distribute v_i's tickets, v_j should first check with v_i and exclude the friends who have already recorded in v_i's list. v_j then distributes v_i's tickets to v_j's remaining friends. Meanwhile, v_i adds these remaining friends to v_i's list.

Router Discovery Step: SGor has two candidate mechanisms for this step: (I), flooding-based router discovery; and (II), probabilistic router discovery.

Using the flooding-based router discovery, any person v_j who receives v_i's tickets should provide r_{ij} candidate routers to v_i. These candidate routers are sent to v_i alongside the backward path from v_j to v_i. Therefore, v_i can receive $\sum_{v_j \in V_i} r_{ij} = R_i^*$ candidate routers in total, where V_i is the set of people who receive v_i's tickets. $R_i^* \leq R_i$ because some v_i's tickets could be discarded in the ticket distribution step. v_i constructs onion circuits by selecting routers uniformly at random from these R_i^* candidate routers, hence resulting in a selection probability $\frac{1}{R_i^*}$ for each candidate router. However, since all the R_i^* candidate routers are sent back alongside backward trust paths to v_i, the flooding-based router discovery runs a high risk of exposing honest routers to others.

To mitigate the exposure of honest routers, we propose a probabilistic router discovery mechanism. Using this mechanism, a person $v_j \in V_i$ can only response one candidate router to the person who distributes tickets to v_j. Let V_{ij} be the set of people who receive v_i's tickets from v_j (i.e., V_{ij} is the set of v_j's eligible friends who receive v_i's tickets). Let $P(v_k)$ be the probability that v_j sends back a router from v_k to v_j's predecessor, where $v_k \in V_{ij} \vee v_j$. The probabilistic router discovery sets $P(v_k)$ to $P(v_k) = \frac{r_{ik}}{\sum_{v_m \in V_{ij} \vee v_j} r_{im}}$. The same as the flooding-based router discovery, this probabilistic router discovery can also result in a selection probability $\frac{1}{R_i^*}$ for each candidate router. Since each person only sends back a single router to its predecessor, the probabilistic router discovery mitigates the chance of exposing honest routers to adversaries.

Figure 4.6 demonstrates an example of the trust graph based router selection algorithm. In this example, each person is assumed to deploy a single router in SGor (i.e., $r_i = 1, i = 0 \ldots 9$). v_1 is the person who attempts to construct onion circuits. The parameters are set as $\Phi_h = 2$, $R_1 = 6$ and $L_1 = 2$. In the initial step, v_1 generates $R_1 = 6$ tickets and set $L_1 = 2$ to the time-to-live field of these tickets. v_1 then distributes 2 tickets to v_3 and 4 tickets to v_4 according to Eq. (4.1) (i.e., $C_{13} = \lceil \frac{\Phi_{13}}{\Phi_{13}+\Phi_{14}} \rceil \cdot R_1 = 2$ and $C_{14} = \lceil \frac{\Phi_{14}}{\Phi_{13}+\Phi_{14}} \rceil \cdot R_1 = 4$). When v_3 and v_4 receive $T_{13} = C_{13} = 2$ and $T_{14} = C_{14} = 4$ tickets respectively, each of them consumes one because each person deploys a single router. For the remaining tickets (v_3 has $T_{13} - r_3 = 1$ and v_4 has $T_{14} - r_4 = 3$), since $L_1 = L_1 - 1 = 1 > 0$, v_3 and v_4 can further distribute these tickets to their eligible friends. v_3 has two eligible friends v_6 and v_7, while v_4 has three (v_8, v_9 and v_0). Although $\Phi_{34} = 2 \geq \Phi_h$, v_3 can exclude v_4 from the set of v_3's eligible friends using the regressive checking mechanism as v_4 has already received v_1's tickets from v_1.

Using Eq. (4.1), v_3 can calculate $C_{36} = \lceil \frac{\Phi_{36}}{\Phi_{36}+\Phi_{37}} \rceil \cdot (T_{13} - r_3) = 1$ and $C_{37} = \lceil \frac{\Phi_{37}}{\Phi_{36}+\Phi_{37}} \rceil \cdot (T_{13} - r_3) = 1$. However, v_3 only has $T_{13} - r_3 = 1$ remaining tickets. In this case, v_3 can simply decide to distribute 1 ticket to v_6 but 0 ticket to v_7, or vice versa. Upon receiving tickets, v_6, v_8, v_9 and v_0 stop the ticket distribution due to two reasons. One is $L_1 = L_1 - 1 = 0$, and the other is that they have no remaining tickets after they consume one. v_1 collects onion routers from the users who consume v_1's tickets. As can be seen in Figure 4.6, both of the two router discovery mechanisms can result in $\frac{1}{6}$ selection probability for each candidate router, although v_3 and v_4 differently act in these two mechanisms. In the flooding-based router discovery, v_3 and v_4 simply send back all the routers they receive to v_1. While in the probabilistic router discovery, v_3 uses the probability 0.5 to choose v_3's router or v_6's router for responding v_1, and v_4 employs the probability 0.25 to chose responded routers.

We note that SGor is not the first one that uses ticket distribution mechanism. The Sybil-resilient rating system has already used it to defeat Sybil attacks [42]. However, SGor's ticket distribution is novel in several aspects. First, SGor distributes tickets according to group trust, hence having better capability of preventing tickets being distributed to adversaries. Second, since people consume tickets based on the number of routers they deploy, SGor can have better capability of evading adversaries' routers if honest people can deploy more routers. Third, to evade duplicated ticket distribution, the Sybil-resilient rating system requires a central server for calculating the shortest path in advance. Unlike that, SGor proposes a novel regressive checking mechanism to evade the duplicated ticket distribution. This checking mechanism does not require a centralized server.

4.2.4 SGor Analysis

In this section, we analyze SGor's capability of protecting anonymity using a probabilistic model. We first discuss whether SGor can effectively evade adversaries' routers from users' onion circuits. We then investigate whether SGor can effectively defend against inference attacks.

4.2.4.1 The Capability of Evading Adversaries' Routers

Since SGor employs group trust and global trust to discover honest routers, we analyze how group trust and global trust affect the capability of evading adversaries' routers, respectively.

Group Trust: In this analysis, we focus on answering the question how likely a person v_j is an adversary if an honest person v_i has group trust Φ_{ij} in v_j.

Let I_{ij} be the event that an honest user v_i assigns incorrect trust to an adversary v_j (i.e., $v_i \rightarrow v_j$ is an incorrect trust assignment). $P(I_{ij})$ is the probability that the event I_{ij} occurs. Since local trust assignments (i.e., trust edges) in SGor's trust graph $G = (V, E)$ are independent (see Section 4.1.2 and 4.1.4), the events I_{ij} for different v_i, v_j are independent with each other. We therefore have $P(I_{ij}, I_{kl}) = P(I_{ij}) \cdot P(I_{kl})$ for $\forall v_i, v_j, v_k, v_l \in V$, where $P(I_{ij}, I_{kl})$ is the joint probability that both events I_{ij} and I_{kl} happen concurrently.

Let A_j be the event that the user $v_j \in V$ is an adversary. $P(A_j)$ is the probability that v_j is an adversary.

Let F_{ij} be the set of the people who are trusted by v_i and meanwhile trust v_j (i.e., for $\forall v_k \in F_{ij}$, there must exist $v_i \rightarrow v_k \rightarrow v_j$). Let $\Lambda_{ij} \subseteq F_{ij}$ be the set of people who are adversaries belonging to F_{ij}.

$P(\Phi_{ij} = N | A_j)$ is the probability that v_i has group trust $\Phi_{ij} = N$ for v_j on condition that v_j is an adversary. We can calculate $P(\Phi_{ij} = N | A_j)$ as:

$$\begin{aligned} P(\Phi_{ij} = N | A_j) = \; & P(I_{ij}) \cdot \sum_{\Lambda_{ij} \subseteq F_{ij}} \prod_{v_k \in \Lambda_{ij}} P(A_k) P(I_{ik}) \cdot \\ & \prod_{v_k \in F_{ij} \setminus \Lambda_{ij}} (1 - P(A_k)) P(I_{kj}). \end{aligned} \tag{4.2}$$

where, $P(I_{ij})$ is the probability that v_i assigns incorrect trust to the adversary v_j. $P(A_k)P(I_{ik})$ represents the probability that v_i assigns incorrect trust to v_k and v_k is another adversary who can forward v_i's trust to the adversary v_j. $(1 - P(A_k))P(I_{kj})$ is the probability that v_i assigns trust to an honest person v_k but v_k assigns incorrect trust to the adversary v_j.

Since $\Phi_{ij} = N$ indicates N robust trust paths from v_i to v_j (see Theorem 4.1), it is intuitive to have a corollary as below.

Corollary 4.2. $P(\Phi_{ij} = N+1|A_j) \leq P(\Phi_{ij} = N|A_j)$.

Proof. Compared with $\Phi_{ij} = N$, v_i has one more robust trust path to v_j when $\Phi_{ij} = N+1$. We can simply assume this additional path is $v_i \to v_k \to v_j$. Hence, we have:
$P(\Phi_{ij} = N+1|A_j)$
$= (P(A_k)P(I_{ik}) + (1 - P(A_k))P(I_{kj})) \cdot P(\Phi_{ij} = N|A_j)$
$\leq (P(A_k) + (1 - P(A_k))) \cdot P(\Phi_{ij} = N|A_j)$
$= P(\Phi_{ij} = N|A_j)$. □

$P(A_j|\Phi_{ij} = N)$ is the probability that v_j is an adversary on condition that v_i has group trust $\Phi_{ij} = N$ for v_j. According to Bayes' theorem, we can have:

$$P(A_j|\Phi_{ij} = N) = \frac{P(\Phi_{ij}=N|A_j) \cdot P(A_j)}{P(\Phi_{ij}=N)}. \tag{4.3}$$

If the probabilities $P(A_j)$ and $P(\Phi_{ij})$ follow uniform distribution, $P(A_j|\Phi_{ij} = N)$ is proportionate to $P(\Phi_{ij} = N|A_j)$. Hence $P(A_j|\Phi_{ij} = N+1) \leq P(A_j|\Phi_{ij} = N)$. That is to say, if an honest people v_i has a larger group trust in another people v_j, v_j is less likely to be an adversary. As a result, SGor has a better capability of evading adversaries' routers from users' onion circuits by using a larger Φ_h.

However, it is not the fact that the probabilities $P(A_j)$ and $P(\Phi_{ij})$ are uniformly distributed in practice. Hence, to show the effectiveness of group trust, we will do more practical evaluations using real-world datasets in Section 4.3.

Global Trust: In this analysis, we focus on answering the question how likely a person v_j is an adversary if an honest person v_i can propagate global trust to v_j through a L_i-hop trust path.

If an honest person v_i propagates global trust to v_j through a L_i-hop trust path, the probability that v_j is an adversary can be calculated using a recursive function:

$$P(A_j|L_i) = P(A_k|L_i - 1) + (1 - P(A_k|L_i - 1)) \cdot P(A_j|\Phi_{kj}). \tag{4.4}$$

where, v_k precedes v_j in the trust path from v_i to v_j. If v_k is an adversary, v_j must be an adversary. Otherwise, v_j has the probability $P(A_j|\Phi_{kj})$ to be an adversary. The base case of the recursive Eq. 4.4 is $P(A_m|L_i = 1) = P(A_m|\Phi_{im})$, where v_m is the first person after v_i in the trust path from v_i to v_j.

Since a larger L_i leads to a higher $P(A_j|L_i)$, a smaller L_i can help SGor achieve a better capability of evading adversaries' routers from users' onion circuits.

4.2.4.2 The Capability of Defeating Inference Attacks

To analyze SGor's capability of defeating inference attacks, we investigate how many users an onion router can be trusted and selected by in average. In SGor, global trust is used to guide people to trust more others in the trust graph, hence allowing honest routers to be trusted and selected by more users in return. We note that, if a router can be selected by more users, inference attackers are hard to guess the initiate user of an onion circuit by observing this router.

SGor uses an adaptive trust propagation algorithm to derive global trust. This algorithm has two parameters L_i and Φ_h (see Section 4.2.3). L_i is used to limit the distance of trust propagation and Φ_h is the group trust minimum threshold that can be used to prevent trust propagation from adversaries.

Since a larger L_i and a smaller Φ_h could lead more routers' owners to be globally trusted, SGor obtains a better capability of thwarting inference attacks by using a larger L_i and a smaller Φ_h. However, as discussed in Section 4.2.4.1, a larger L_i and a smaller Φ_h also result in a worse capability of evading adversaries' routers. As a result, SGor should choose appropriate Φ_h and L_i to balance the capability of evading adversaries' routers and the capability of defeating inference attacks. Section 4.3 will evaluate SGor on top of real-world social trust datasets and show appropriate Φ_h and L_i for these datasets.

4.3 Evaluation

In this section, we evaluate SGor using two real-world social trust datasets. We first describe the datasets in Section 4.3.1. We then evaluate SGor's capability of evading adversaries' routers and the capability of defeating inference attacks in Section 4.3.2 and 4.3.3, respectively. We compare SGor with other global trust-based routing protocols in terms of deanonymization expectation in Section 4.3.4. After evaluating the leakage of a priori trust relationships in Section 4.3.5, we also evaluate the overheads introduced by SGor in terms of storage and communication round trips in Section 4.3.6.

4.3.1 Datasets

We adopt two real-world social interaction graph datasets to evaluate SGor. One dataset contains post wall interactions between users in a New Orleans regional network in Facebook [1]. This interaction graph has $46,952$ nodes and $876,993$ trust edges. The other dataset is the collection of one month interactions from another Facebook anonymous regional network [125]. It includes $431,995$ nodes and $781,862$ trust edges.

In our evaluated interaction graphs, each node represents a person who registers an account in Facebook. Each trust edge indicates a local trust that one person assigns to another person (see Section 4.1.4). We assume any person can play the role as an onion routing user or an onion router owner, or both or none of them (see Section 4.1.5.1).

We preprocess these two datasets following a similar manner as X-Vine [133] and SybilLimit [40] did. In particular, we remove the nodes with low degrees (e.g., the sum of outdegree and indegree is less than 5). This process can guarantee all the nodes in our evaluation have reasonable connectivity to the trust graph. We also eliminate the self-linked edges (e.g., $v_i \rightarrow v_i$) because these trust edges could mislead group trust computation. After these appropriate preprocesses, we use the truncated datasets to evaluate SGor.

Table 4.2 summarizes basic statistics of these two truncated datasets. The average degree takes both outdegree and indegree into consideration. It is calculated by using the number of edges to divide the number of nodes in the graph (i.e., $\frac{|E|}{|V|}$). The density of a graph is defined as the number of edges in the graph divided by the number of edges in a complete graph (i.e., $\frac{|E|}{|V|\cdot(|V|-1)}$). Both the average degree and graph density are graph properties that can be used to show whether a graph is well connected or not. A larger average degree and density leads to a more tight-knit graph.

TABLE 4.2: The Two Interaction Social Graphs after Preprocess.

Dataset	# of Nodes	# of Edges	Avg. degree	Graph density
New Orleans [1]	$27,601$	$231,035$	8.37	3×10^{-4}
Anonymous [125]	$83,034$	$305,711$	3.68	4.4×10^{-5}

To have a better understanding of how graph properties (i.e., average degree and graph density) affect the parameter selection in SGor (the selection of Φ_h and L_i), we generate six synthetic graphs using a tool NetworkX [134] for investigation. We put the results based on synthetic graphs in Section 4.4.

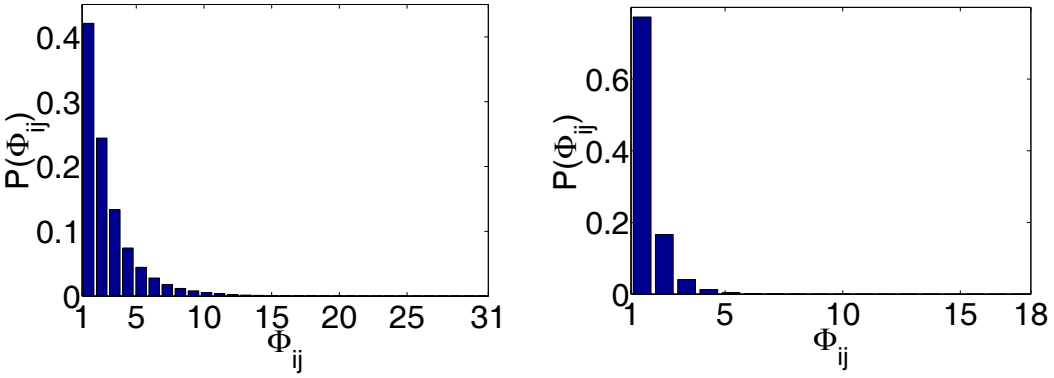
4.3.2 Evaluating The Capability of Evading Malicious Routers

In this section, we first evaluate group trust's effectiveness in evading adversaries' routers. We then show global trust's impacts to this effectiveness. Afterwards, we compare SGor with state-of-the-art trust-based onion routing to show SGor's improvement in protecting users from adversaries' routers.

4.3.2.1 Group Trust's Effectiveness in evading Adversaries' Routers:

We measure group trust's effectiveness in evading adversaries' routers in terms of the probability $P(A_j|\Phi_{ij} = N)$. A smaller $P(A_j|\Phi_{ij} = N)$ indicates a smaller probability that v_j is an adversary given group trust $\Phi_{ij} = N$. According to Eq. 4.3, $P(A_j|\Phi_{ij} = N)$ is determined by $P(\Phi_{ij} = N)$, $P(\Phi_{ij} = N|A_j)$ and $P(A_j)$.

Based on the two real-world datasets listed in Table 4.2, we calculate practical distributions of $P(\Phi_{ij})$. In New Orleans dataset, Φ_{ij} ranges from 1 to 31. While in the anonymous dataset, $1 \leq \Phi_{ij} \leq 18$. Figure 4.7 illustrates histograms of $P(\Phi_{ij})$ for the two datasets. It can be seen that the two datasets have the similar distributions of Φ_{ij}. A larger Φ_{ij} results in a smaller $P(\Phi_{ij})$. Moreover, there are more Φ_{ij}s having a smaller value in the Anonymous dataset than those in the New Orleans dataset. This is probably due to the lower graph density in the Anonymous dataset. We have further investigated this phenomenon using syntactic graphs in Appendix A.2.



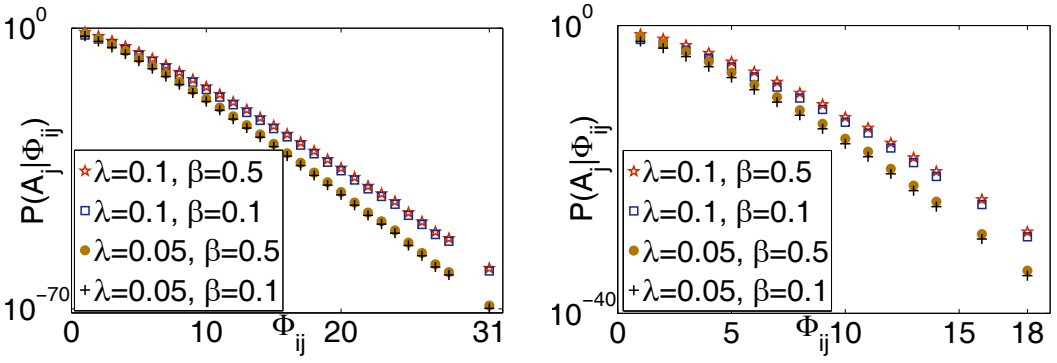
(a) New Orleans Dataset [1]. (b) Anonymous Dataset [125].

FIGURE 4.7: The real-world group trust distributions of $P(\Phi_{ij})$.

According to Theorem 4.1 and Eq. 4.2, $P(\Phi_{ij} = N|A_j)$ is determined by the probabilities of events that honest people assign incorrect trust to adversaries. Since incorrect trust assignments are human behaviors, we can model these behaviors as a Poisson process with an average incorrect trust assignments probability λ (i.e., $P(\Phi_{ij} = N|A_j)$ follows a Poisson distribution with a parameter λ). A large λ indicates that a large number of people in SGor assign incorrect trust to adversaries. Since it is very difficult for adversaries to attract incorrect trust from a large number of honest human beings [44], SGor is not designed for a large λ. Therefore, we only consider a small λ in our evaluation.

We consider adversaries as independent and identically distributed, i.e., the events A_j are i.i.d. In this case, $P(A_j)$ for $\forall v_j \in V$ is equal to a constant $\beta \in (0, 1)$. A large β indicates a large fraction of the network is compromised by adversaries.

We evaluate the effectiveness of group trust Φ_{ij} in protecting an honest person v_i from an adversary v_j's routers by investigating $P(A_j|\Phi_{ij})$ for different Φ_{ij}s in the two real-world datasets. A smaller $P(A_j|\Phi_{ij})$ means a local trust assignment $v_i \rightarrow v_j$ is less likely to be incorrect, hence indicating a better capability of evading adversaries' routers. In this evaluation, we consider $\lambda = 0.1$ or $\lambda = 0.05$ and $\beta = 0.5$ or $\beta = 0.1$. Figure 4.8 shows the evaluation results. It can be seen that a larger Φ_{ij} leads to a smaller $P(A_j|\Phi_{ij})$ (i.e., a better capability of evading adversaries). In particular, for the dataset [1], $P(A_j|\Phi_{ij})$ drops about 10^{68} times when the group trust climbs from $\Phi_{ij} = 1$ to 31. For the dataset [125], $P(A_j|\Phi_{ij})$ declines around 10^{35} times when the group trust grows up from $\Phi_{ij} = 1$ to 18. Moreover, $P(A_j|\Phi_{ij})$ with a setting of smaller λ and β is smaller than that with a setting of larger λ and β. This indicates that group trust is more effective in evading adversaries when honest people make less incorrect local trust assignments in SGor and adversaries control a smaller fraction of the SGor network.



(a) New Orleans Dataset [1]. (b) Anonymous Dataset [125].

FIGURE 4.8: The probability $P(A_j|\Phi_{ij})$ with different Φ_{ij} in four settings of λ and β.

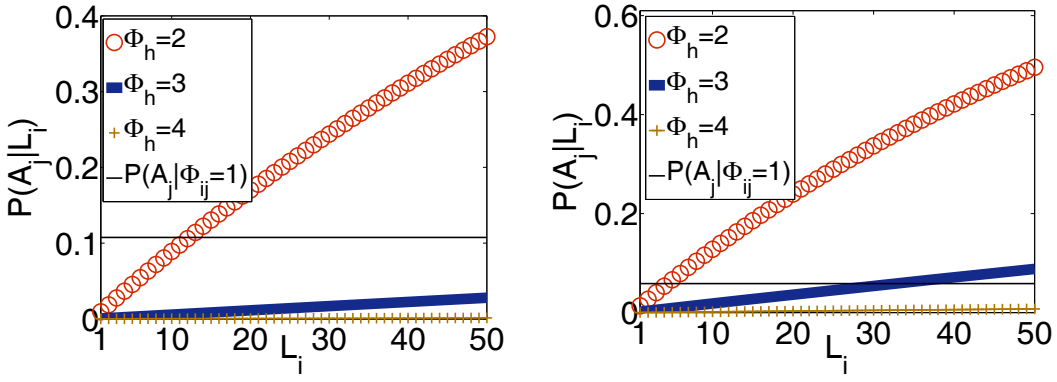
4.3.2.2 Global Trust's Impact:

We evaluate the global trust's impacts on the capability of evading adversaries' routers by investigating $P(A_j|L_i)$. A smaller $P(A_j|L_i)$ indicates that v_j is less likely to be an adversary if an honest person v_i propagate global trust to v_j through a trust path consisting of L_i trust edges. When SGor derives global trust using adaptive trust propagation algorithm, $P(A_j|L_i)$ is determined by the length of the trust path L_i and the group trust of each trust edge in the trust path. The group trust of each trust edge must be no smaller than a minimum threshold Φ_h (see Section 4.2.3). A larger L_i results in a larger $P(A_j|L_i)$, while a larger Φ_h leads to a smaller $P(A_j|L_i)$.

To evaluate global trust's impact, we investigate $P(A_j|L_i)$ when L_i varies in different cases of Φ_h. We compare $P(A_j|L_i)$ with $P(A_j|\Phi_{ij} = 1)$. $P(A_j|\Phi_{ij} = 1)$ represents how likely v_j is an adversary if v_i has local trust in v_j. $P(A_j|L_i) < P(A_j|\Phi_{ij} = 1)$ indicates that the

people who receive global trust through a trust path consisting of L_i trust edges are less likely to be adversaries than the people who receive local trust. As a result, if SGor can achieve $P(A_j|L_i) < P(A_j|\Phi_{ij} = 1)$ using appropriate L_i and Φ_h, the global trust could not degrade the capability of evading adversaries' routers.

Figure 4.9 shows the probability $P(A_j|L_i)$ when $\Phi_h = 2, 3, 4$ and L_i increases from 1 to 50. In this evaluation, we use the setting of $\lambda = 0.1$ and $\beta = 0.5$, and consider the worst case that the trust propagation is through the trust paths where any trust edges $v_k \rightarrow v_m$ have $\Phi_{km} = \Phi_h$. In the dataset [1], for the case $\Phi_h = 2$, $P(A_j|L_i)$ becomes larger than $P(A_j|\Phi_{ij} = 1)$ when $L_i > 12$. For the other cases $\Phi_h = 3$ and $\Phi_h = 4$, $P(A_j|L_i)$ remains smaller than $P(A_j|\Phi_{ij} = 1)$ even if L_i grows up to 50. While in the dataset [125], for the cases $\Phi_h = 2$ and $\Phi_h = 3$, $P(A_j|L_i)$ is larger than $P(A_j|\Phi_{ij} = 1)$ when $L_i > 5$ and $L_i > 30$ respectively. Based on this evaluation, to maintain the capability of evading adversaries' routers, SGor should propagate global trust through the trust paths no longer than $L_i = 12$ if $\Phi_h = 2$ in the dataset [1], and no longer than $L_i = 5$ if $\Phi_h = 2$ or $L_i = 30$ if $\Phi_h = 3$ in the dataset [125]. It can be seen, SGor can choose a larger L_i for trust propagation in the graph with higher density (i.e., the dataset [1] has a higher graph density than the dataset [125]).



(a) New Orleans Dataset [1]. (b) Anonymous Dataset [125].

FIGURE 4.9: The probability $P(A_j|L_i)$ when L_i is from 1 to 50.

4.3.2.3 Simulation of SGor and Trust-based Onion Routing:

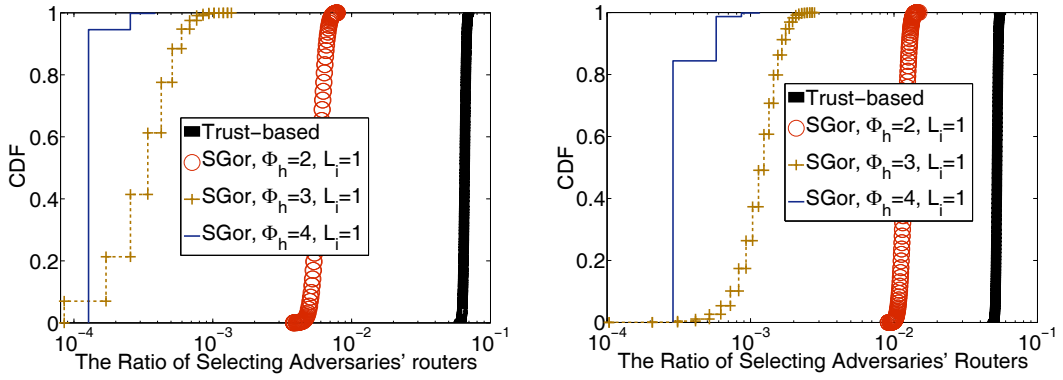
To show SGor's advantage in evading adversaries' routers compared with trust-based onion routing, we simulate router selections based on the two real-world datasets with setting of $\lambda = 0.1$ and $\beta = 0.5$. In this simulation, SGor sets $L_i = 1$, $\Phi_h = 2, 3, 4$ and R_i to a large enough value, respectively. SGor uses Φ_h to filter the routers whose group trust is smaller than Φ_h and propagates global trust to at most L_i hops. We note that the trust-based onion routing operates in the same manner as SGor with $L_i = 1$ and $\Phi_h = 1$. We also assume each person deploys a single router in the network. We generate a pseudo random value $\pi \in [0, 1]$ for each person

v_j who deploys an onion router. From an honest person v_i's point of view, v_j is regarded as an adversary if $\pi < P(A_j|L_i)$.

We simulate a person v_i as an honest person to select a router deployed by other people using existing trust-based onion routing algorithms [12, 14] or the trust graph based algorithm proposed in Section 4.2.3. We define a round of simulation by iterating all the people in the two real-world datasets to operate as v_i one by one. We conduct $10,000$ rounds of router selection simulations for trust-based onion routing and SGor with $L_i = 1$, $\Phi_h = 2, 3, 4$, respectively.

In each round of simulation, we measure the capability of evading adversaries' routers using the ratio of selecting adversaries' routers. This ratio is calculated by using the number of people who select adversaries' routers in each round to divide the total number of people. A smaller ratio means a better capability of evading adversaries' routers.

Figures 4.10(a) and 4.10(b) show the cumulative distribution function (CDF) of the ratio of selecting adversaries' routers in our simulations. It can be seen that SGor outperforms trust-based onion routing in both the New Orleans dataset [1] and the anonymous dataset [125]. In particular, SGor with parameters $\Phi_h = 2$ and $L_i = 1$ has more than 10 times improvement for evading adversaries' routers. When the parameter Φ_h is increased to 4, the improvement extends to around $1,000$ times.



(a) New Orleans Dataset [1].

(b) Anonymous Dataset [125].

FIGURE 4.10: The CDFs of the ratio of selecting adversaries' routers in $10,000$ rounds of router selection simulation in SGor and trust-based onion routing.

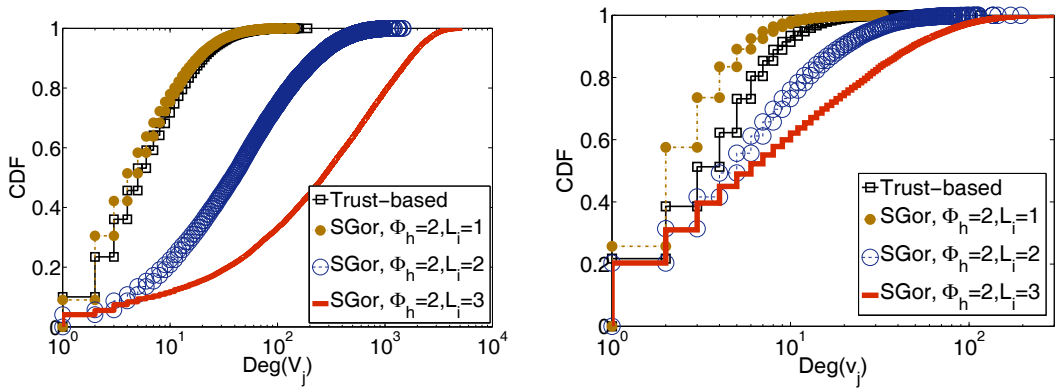
4.3.3 Evaluating The Capability of Defeating Inference Attacks

We evaluate an honest onion router's capability of defeating inference attacks by investigating the number of users who can select this router. A network (SGor or trust-based onion routing) has a better capability of resisting inference attacks if more honest onion routers can be selected by more users.

Let $Deg(v_j)$ be the number of people who can select the onion routers deployed by v_j. In trust-based onion routing, a person v_i can select another person v_j's routers if and only if v_i has local trust in v_j (i.e., $v_i \rightarrow v_j$ exists in G). Hence, $Deg(v_j)$ is the in-degree of the node v_j in the trust graph G. While in SGor, v_i can select v_j's routers in two cases. One is $\Phi_{ij} \geq \Phi_h$ if $v_i \rightarrow v_j$ exists. The other is v_i can propagate global trust to v_j using the adaptive trust propagation algorithm described in Section 4.2.2 and 4.2.3.

We use $Deg(v_j)$ as a measure to compare the capability of defeating inference attacks between SGor and trust-based onion routing. A larger $Deg(v_j)$ indicates a better capability of defeating inference attacks. We consider SGor with parameters $L_i = 2, 3$ and $\Phi_h = 2$. We also choose R_i as a value which is large enough to guarantee SGor can propagate trust to $L_i = 2$ or 3. Our evaluation adopts the two real-world datasets listed in Table 4.2 and considers all the people to be as v_j one by one.

Figure 4.11 shows the CDF of $Deg(v_j)$ for SGor and trust-based onion routing in our evaluation. Compared to trust-based onion routing, SGor with $\Phi_h = 2$ and $L_i = 1$ suffers a distribution of slightly smaller $Deg(v_j)$s because $\Phi_h = 2$ filter trust edges whose group trust is smaller than 2. This slight degradation does not make a significant impact on the capability of defeating inference attacks, because these removed trust edges are likely to be incorrect trust assignments. When L_i is increased to 2, SGor reaches a distribution with much larger $Deg(v_j)$s than trust-based onion routing. The value of $Deg(v_j)$ is further increased when L_i grows up to 3 in SGor. These results show that SGor with a larger L_i can have a better capability of defeating inference attacks. By referring back to Figure 4.9, we note that SGor with $\Phi_h \geq 2$ and $L_i \leq 5$ retains a better capability of evading adversaries' routers than trust-based onion routing. As a result, SGor apparently demonstrates a better capability of defeating inference attacks and evading adversaries' routers simultaneously.



(a) New Orleans Dataset [1]. (b) Anonymous Dataset [125].

FIGURE 4.11: The CDFs of $Deg(v_j)$ for SGor and trust-based onion routing.

4.3.4 Comparing SGor with Other Global Trust-based Schemes

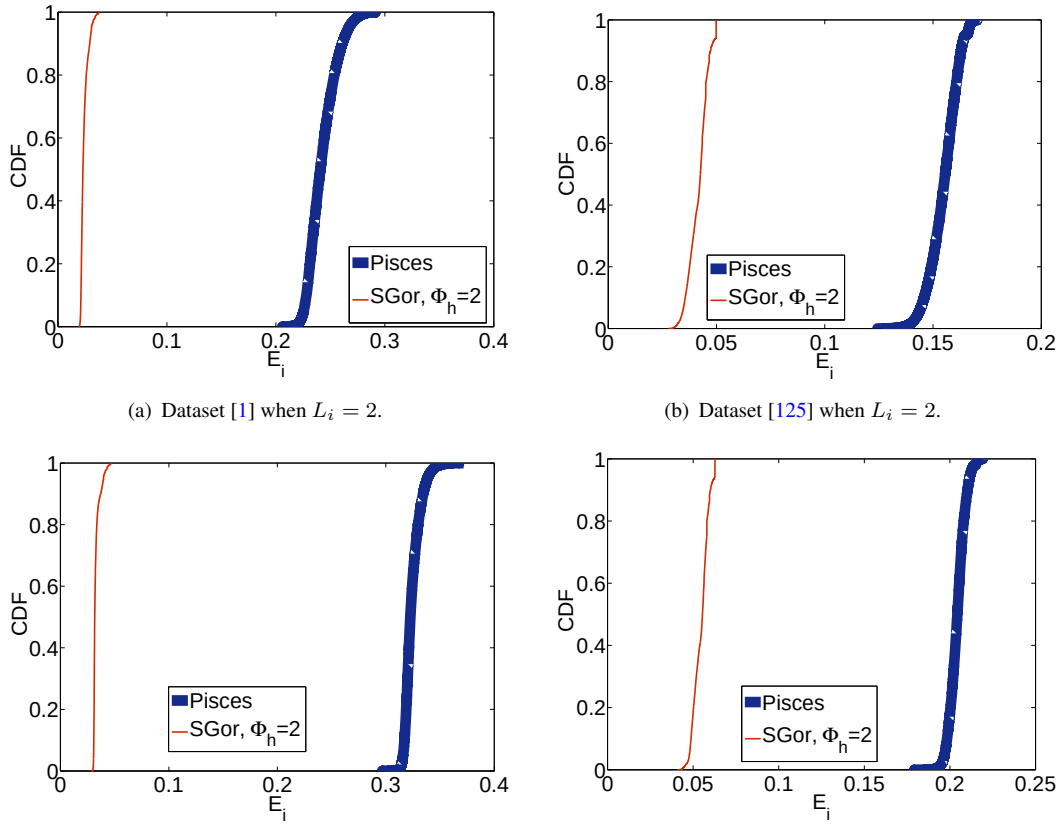
Although Section 4.3.2 and 4.3.3 have proved SGor's better performance of protecting anonymity than trust-based routing approaches that utilize local trust [11, 12, 14], we need to further compare SGor with prior studies which also apply global trust to onion routing. In the literature, there are two projects, Drac [13] and Pisces [15], that fall into this category. However, since Pisces has been proved to have better capability of protecting anonymity than Drac [15], we only compare SGor with Pisces because Pisces represents the state-of-the-art techniques using global trust.

To have a fair comparison between SGor and Pisces, we propose the use of deanonymization expectation to measure the anonymity each user can obtain from the two schemes. This measure takes both the capabilities of evading adversaries' routers and defeating inference attacks into account.

Let E_i be the person v_i's deanonymization expectation. E_i can be calculated as $E_i = \sum_{v_j \in \Gamma_i} P_{ij} \cdot E_{ij}$. Where Γ_i is the set of persons whose router can be selected by v_i. P_{ij} is the probability that the person v_i uses to select the person v_j's router. It is determined by different trust-based routing algorithms. E_{ij} is the average probability that v_i can be deanonymized due to the selection of v_j's router. We have $E_{ij} = P(A_j|L_i) \cdot 1 + (1 - P(A_j|L_i)) \cdot P(v_i|v_j) \cdot \mu$. The constant 1 in $P(A_j|L_i) \cdot 1$ is the probability that v_i can be deanonymized when v_j is an adversary. This probability equals to 1 because the adversary v_j can launch correlation-like attacks to deanonymize v_i directly. $P(v_i|v_j)$ represents the probability that adversaries can guess v_i by observing v_j's router (i.e., inference attacks) when v_j is not an adversary. μ is the probability that adversaries can observe routers in onion circuits. We choose a small $\mu = 0.01$ in our evaluation, because adversaries can only observe routers by exploiting adversaries' routers or web servers (see Section 5.1), but the web servers adversaries can control are relatively few and users rarely select adversaries' routers for onion routing after using trust-based algorithm (see Figure 4.8). L_i which is the number of hops v_i can propagate trust in SGor equals to the number of random walk steps in Pisces. Apparently, a smaller E_i indicates a better capability of protecting anonymity.

In our experiments, we compare SGor and Pisces by considering the setting of $\lambda = 0.1$ and $\beta = 0.5$. We choose $\Phi_h = 2$ for SGor. Moreover, we consider $L_i = 2$ and $L_i = 3$ in our comparison. That is, SGor propagates trust through 2 or 3-hop trust path while Pisces launches 2 or 3-step metropolis-hastings random walks on top of the New Orleans dataset [1] and the Anonymous dataset [125], respectively. Although we cannot guarantee the random walk can globally converge within $L_i = 2$ or $L_i = 3$ steps (it depends on whether the graph is fast-mixing), Pisces employs metropolis-hastings algorithm to guarantee the reach probability of each router during the random walk is approximately uniform distributed [15].

Figure 4.12 shows the results of our comparison. Apparently, SGor achieves a better capability in protecting anonymity (in terms of a smaller deanonymizaiton expectation E_i). SGor outperforms Pisces because SGor employs group trust to rate limit trust propagation. This idea can effectively prevent adversaries from compromising global trust in trust propagation.



(a) Dataset [1] when $L_i = 2$.

(b) Dataset [125] when $L_i = 2$.

(c) Dataset [1] when $L_i = 3$.

(d) Dataset [125] when $L_i = 3$.

FIGURE 4.12: The CDF of E_i in Pisces and SGor with $\Phi_h = 2$.

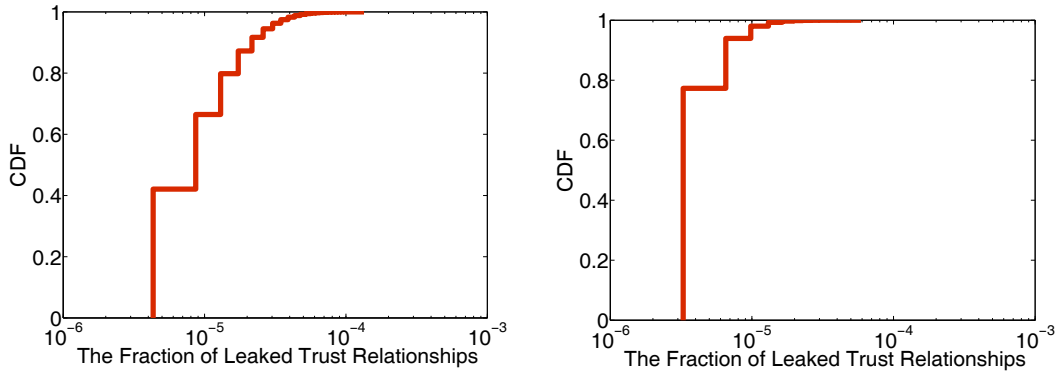
4.3.5 Evaluating The Leakage of A Priori Trust Relationships

Since SGor is designed to run in a fully decentralized manner, it is not required to leak a priori trust relationships to any third parties, such as a centralized server. However, as the calculation of group trust is based on mutual friends information, people's friends could expose some of their trust edges to a third person who computes group trust. For example, if v_i has two robust trust paths to v_j, one is $v_i \rightarrow v_j$ and the other is $v_i \rightarrow v_k \rightarrow v_j$, to calculate Φ_{ij}, v_k's trust edge $v_k \rightarrow v_j$ could be leaked to v_i. As a result, SGor could leak some of people's trust relationships to other people.

We measure the trust relationships leaked to a third person in terms of the fraction of leaked trust edges. The fraction can be calculated by using the number of trust edges that are leaked to an

individual third person during group trust calculation to divide the total number of trust edges in SGor's trust graph. A larger fraction indicates a larger amount of trust edges leaked.

Figure 4.13 shows the CDF of the fraction of trust relationships that are leaked to each third person who computes group trust. It can be seen that the leaked fraction is negligible (less than 0.0002). Although a large number of adversaries could collude with each others (i.e., Sybil attacks), they cannot effectively enlarge the fraction of leaked trust relationships, because adversaries should compromise trust with two honest people to leak these two people's trust relationship through group trust computation. However, it is very difficult for adversaries to compromise trust with a large number of honest people. Previous Sybil defences (e.g., SybilLimit [40]) are usually designed based on this observation. As a result, the adversaries who perform inference attacks cannot benefit a lot from the decentralized design of SGor when they collect a priori trust relationships.



(a) New Orleans Dataset [1]. (b) Anonymous Dataset [125].

FIGURE 4.13: The CDF of the fraction of trust relationships that are leaked to each third person who calculates group trust.

4.3.6 Evaluating SGor's Overheads

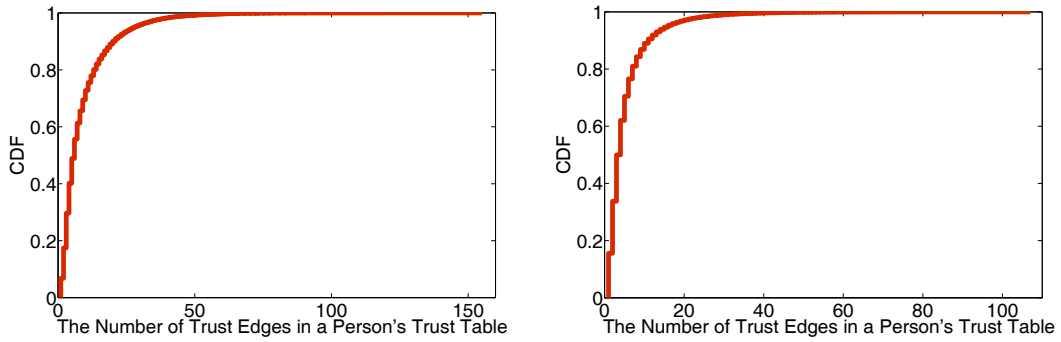
To support trust graph based onion routing, SGor introduces overheads because it performs additional storage and communication in the trust graph layer. In this section, we evaluate these overheads in terms of storage in Section 4.3.6.1, communication round trips in Section 4.3.6.2, and additional traffic in Section 4.3.6.3. The results confirm that SGor only induces very few overheads due to its decentralized design.

4.3.6.1 Storage Overheads

Benefit from the decentralized design, SGor is not required to deploy a centralized server to manage the entire trust graph. Instead, each person in SGor has a trust table to store his own local trust assignments, i.e., trust edges from this person to this person's friends (see Section

4.1.5.2). As a result, the storage overheads introduced by SGor are due to the additional storage space required by the trust table.

We evaluate the storage overheads by investigating the number of trust edges stored in each person's trust table. Figure 4.14(a) and 4.14(b) plot the CDF of this number for the people in datasets [1] and [125], respectively. It can be seen that most of people (around 80%) have less than 20 trust edges stored in their trust table. Compared to the total number of trust edges in these two datasets (as shown in Table 4.2, the total number of trust edges is $231,035$ in dataset [1] and $305,711$ in dataset [125]), the number of trust edges stored in each person's trust table is relatively small. As a result, SGor introduces a few storage overheads for each person.



(a) New Orleans Dataset [1].

(b) Anonymous Dataset [125].

FIGURE 4.14: The CDF of the number of trust edges stored in each person's trust table in SGor.

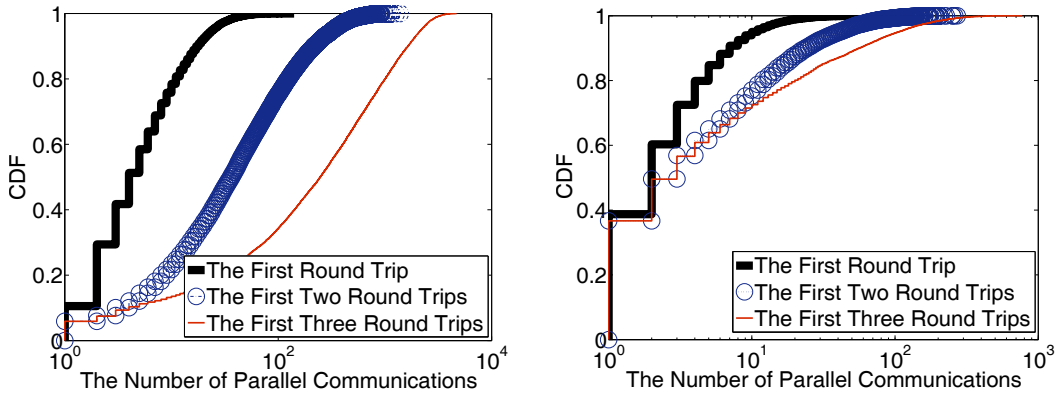
4.3.6.2 Communication Overheads

As described in Section 4.2.3, onion routing users need to run a trust graph based algorithm to discover honest routers in SGor. This algorithm induces additional communications among people (through people's user-agents) in the trust graph layer.

We evaluate SGor's communication overheads in terms of communication round trips. A communication round trip is the time delay of a bi-directional communication between two people in the trust graph layer. Since SGor operates in a fully decentralized manner, most communications between different two people can be performed in parallel. According to SGor's decentralized algorithms (see Sections 4.2.1.3 and 4.2.3), the number of additional communication round trips introduced by SGor can be calculated by summing up one and a half round trips (for group trust computation) and the length of the longest trust path through which SGor propagates global trust (i.e., $L_i + 1.5$). These additional communication round trips are only determined by the parameter L_i and cannot be affected by the size of trust graph. Since $L_i + 1.5$ is a very small value compared with the size of a trust graph, SGor introduces a few communication overheads even if it adopts a large scale trust graph.

We investigate the number of parallel communications when people run trust graph based router selection algorithm in SGor. This number indicates the benefits that SGor can obtain from the decentralized design to reduce communication overheads. A larger number of parallel communications means more efficiency benefits.

Figure 4.15 shows the CDFs of the number of parallel communications during different number of communication round trips. In this evaluation, people are simulated to run decentralized ticket distribution algorithm to discover honest routers on top of the datasets [1] and [125]. We consider $\Phi_h = 2$ and $L_i = 3$. It can be seen that more communication round trips could result in more parallel communications. This is the reason why a large scale trust graph does not introduce a serious communication overhead in SGor.



(a) New Orleans Dataset [1].

(b) Anonymous Dataset [125].

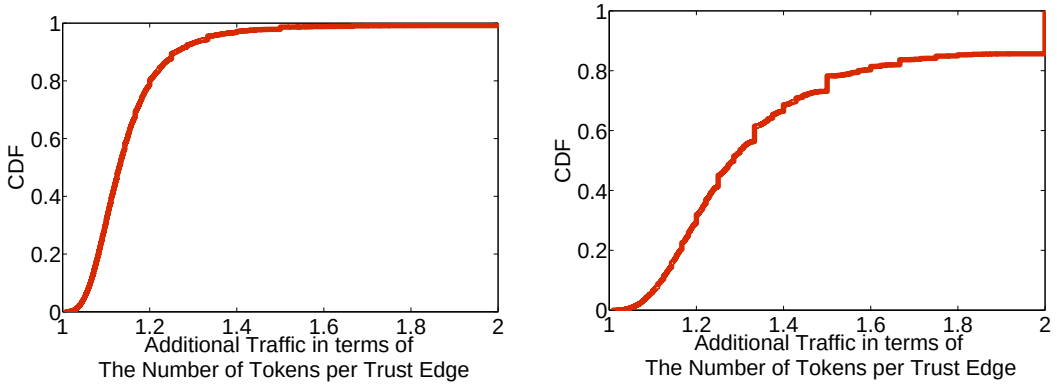
FIGURE 4.15: The CDFs of the number of parallel communications during different number of communication round trips in SGor with $\Phi_h = 2$ and $L_i = 3$.

4.3.6.3 Additional Traffic

In contrast to the additional communications, SGor also introduces additional traffic to the network. This overhead is caused by group trust aggregation and adaptive global trust propagation.

When SGor aggregates group trust using the decentralized token-based algorithm (see Section 4.2.1.3), we measure the additional traffic in terms of the number of tokens transmitted in each trust edge. When a person calculates group trust for his friends, two kinds of trust edges are involved. One includes the edges from this person to his friends, and the other includes the edges from this person's friends to the friends of this person's friends. The tokens can be distributed through the first kind of edges at first, and then further forwarded via the second kind of edges. At last, appropriate tokens can be sent back to the initiate user through the first kind of edges again. The number of tokens transmitted in each trust edge is computed by using the number of tokens transmitted on the fly to divide the number of trust edges involved.

Figure 4.16 illustrates the CDF of the average number of tokens transmitted in each trust edge. Apparently, the additional traffic introduced by group trust calculation is not large because no more than 2 tokens per trust edge are inserted into the network. However, there is an interesting observation that the New Orleans dataset introduces less additional traffic than the Anonymous dataset, although the group trust in the former dataset is statistically larger than that in the latter dataset. The reason of this observation is because a large number of people in the Anonymous dataset have none friends even if they are friends of other people. These people cannot induce the second kind of trust edges during group trust calculation, hence resulting a small number of trust edges when we compute the number of tokens per trust edge.

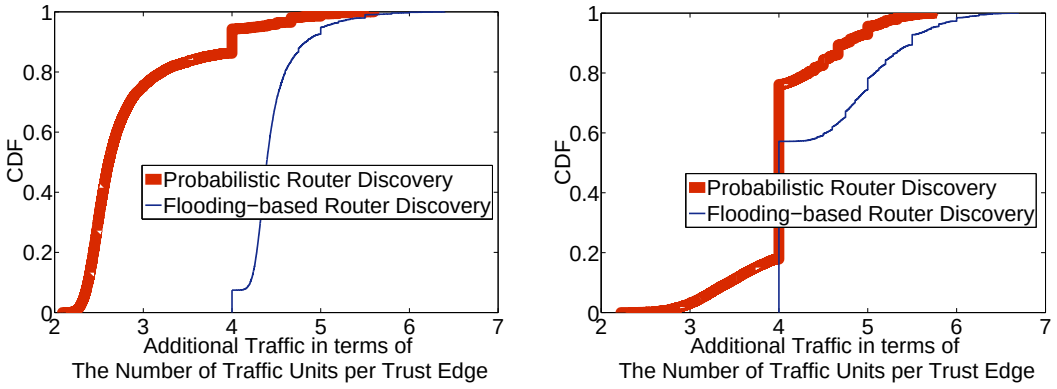


(a) New Orleans Dataset [1]. (b) Anonymous Dataset [125].

FIGURE 4.16: The CDFs of additional traffic introduced by group trust aggregation (in terms of the number of tokens per trust edge) in SGor.

Beside group trust aggregation, the adaptive global trust propagation also injects additional traffic into the network. We measure this kind of additional traffic in terms of traffic units that are conveyed in each trust edge. We use traffic unit as the measure, because global trust propagation could induce three different kinds of additional traffic. As described in Section 4.2.3, our global trust propagation algorithm consists of a tickets forwarding, a regressive checking mechanism for evading duplicated ticket distribution and a router collection. Each tickets forwarding injects one unit of additional traffic as it only forwards the number of remaining tickets through trust edges. For the regressive checking mechanism, the checking request and the checking response consume one traffic unit each. For the router collection, each router transmitted in the trust edge induces one additional traffic unit.

Figure 4.17 demonstrates the CDFs of the number of traffic units transmitted in each trust edge during global trust propagation in SGor when $\Phi_h = 2$ and $L_i = 3$. It can be seen that the additional traffic is relatively small (no more than 7 traffic units are transmitted in each trust edge). Moreover, although the probabilistic router discovery mechanism is originally proposed to mitigate the chances of router exposure during trust propagation, our result confirms that this mechanism is also effective in reducing additional traffic compared with the flooding-based router discovery.



(a) New Orleans Dataset [1]. (b) Anonymous Dataset [125].

FIGURE 4.17: The CDFs of additional traffic introduced by global trust propagation (in terms of the number of traffic units transmitted in each trust edge) in SGor when $\Phi_h = 2$ and $L_i = 3$.

4.4 Syntactic Graph based Analysis

Our syntactic graph based analysis is presented here to investigate how graph properties (e.g., average degree and density) affect the parameter (e.g., Φ_h and L_i) selection in SGor. In particular, we generate syntactic graphs by calling the function `barabasi_albert_graph()` in NetworkX [134]. This function returns a random scale-free graph based on a Barabási-Albert preferential attachment model [135]. The trust graphs following this model are widely observed in nature and human-made systems, which include social networks.

We have generated six syntactic graphs with different sizes, average degrees and graph densities. Table 4.3 summarizes the statistics of these six graphs.

TABLE 4.3: Statistics in Six Syntactic Graphs.

Dataset	# of Nodes	# of Edges	Avg. degree	Graph density
Syntactic Graph ①	$1,000$	$4,975$	4.98	5×10^{-3}
Syntactic Graph ②	$1,000$	$10,879$	10.9	1.1×10^{-2}
Syntactic Graph ③	$1,000$	$51,975$	52.0	5.2×10^{-2}
Syntactic Graph ④	$1,000$	$97,900$	98.0	9.8×10^{-2}
Syntactic Graph ⑤	$2,000$	$9,975$	5.00	2.5×10^{-3}
Syntactic Graph ⑥	$2,000$	$106,975$	53.5	2.7×10^{-2}

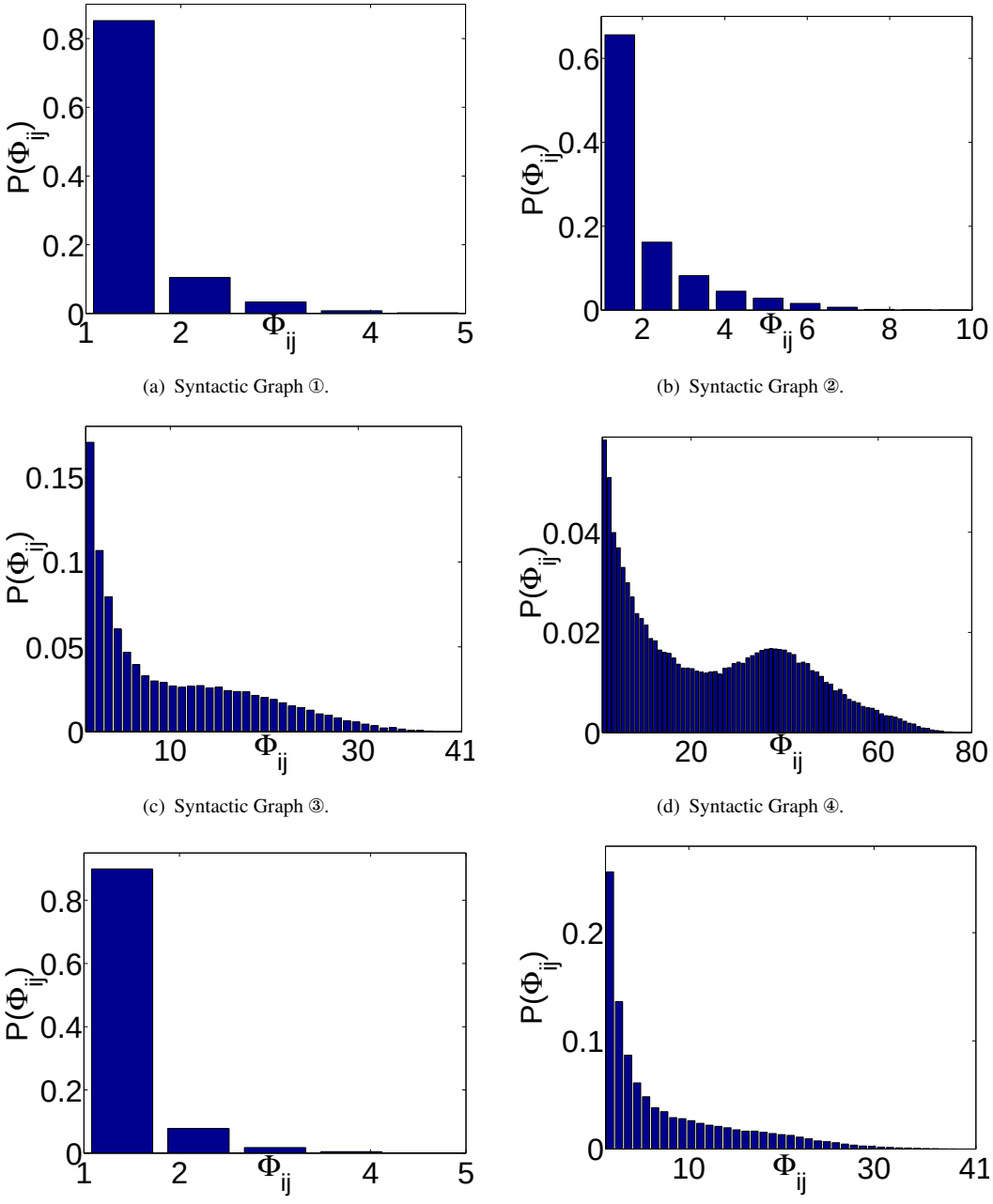
We show the group trust (i.e., $P(\Phi_{ij})$) distributions of these six graphs in Figures 4.18(a)-4.18(f). By analyzing these figures as well as consulting Table 4.3, we confirm that a larger average degree or a higher graph density results in a larger value of $P(\Phi_{ij})$. Moreover, compared with graph density, the average degree is more appropriate for indicating the group trust distribution. For example, although the density of syntactic graph ⑤ is roughly a half of the

density in the graph ①, their group trust distributions are similar because they have roughly the same average degree.

Based on the group trust distributions, we also plot the probability $P(A_j|\Phi_{ij})$ with different Φ_{ij} in four settings of λ and β in Figures 4.19(a)-4.19(f). These figures show the $P(A_j|\Phi_{ij})$ consistently decreases when Φ_{ij} increases. Moreover, the $P(A_j|\Phi_{ij})$ can reach a smaller value in the graph which has higher average degree and graph density.

To investigate how graph properties (i.e., average degrees and graph densities) affect the global trust propagation, we show $P(A_j|L_i)$ in these six graphs in Figures 4.20(a)-4.20(f). In these figures, the $P(A_j|\Phi_{ij} = 1)$ represents how likely v_j is an adversary if v_i has local trust in v_j. If $P(A_j|L_i) < P(A_j|\Phi_{ij} = 1)$, that means the people who receive global trust through a trust path consisting of L_i trust edges are less likely to be adversaries than the people who receive local trust. As a result, when SGor can achieve $P(A_j|L_i) < P(A_j|\Phi_{ij} = 1)$ using appropriate L_i and Φ_h, the trust propagation could not degrade the capability of resisting adversaries.

To maintain the capability of resisting adversaries during trust propagation, we summarize appropriate L_i according to different Φ_h in SGor for these six syntactic graphs in Table 4.4. Apparently, the graph with larger average degree and higher density supports a larger L_i for trust propagation without reducing the capability of evading adversaries.



(a) Syntactic Graph ①.

(b) Syntactic Graph ②.

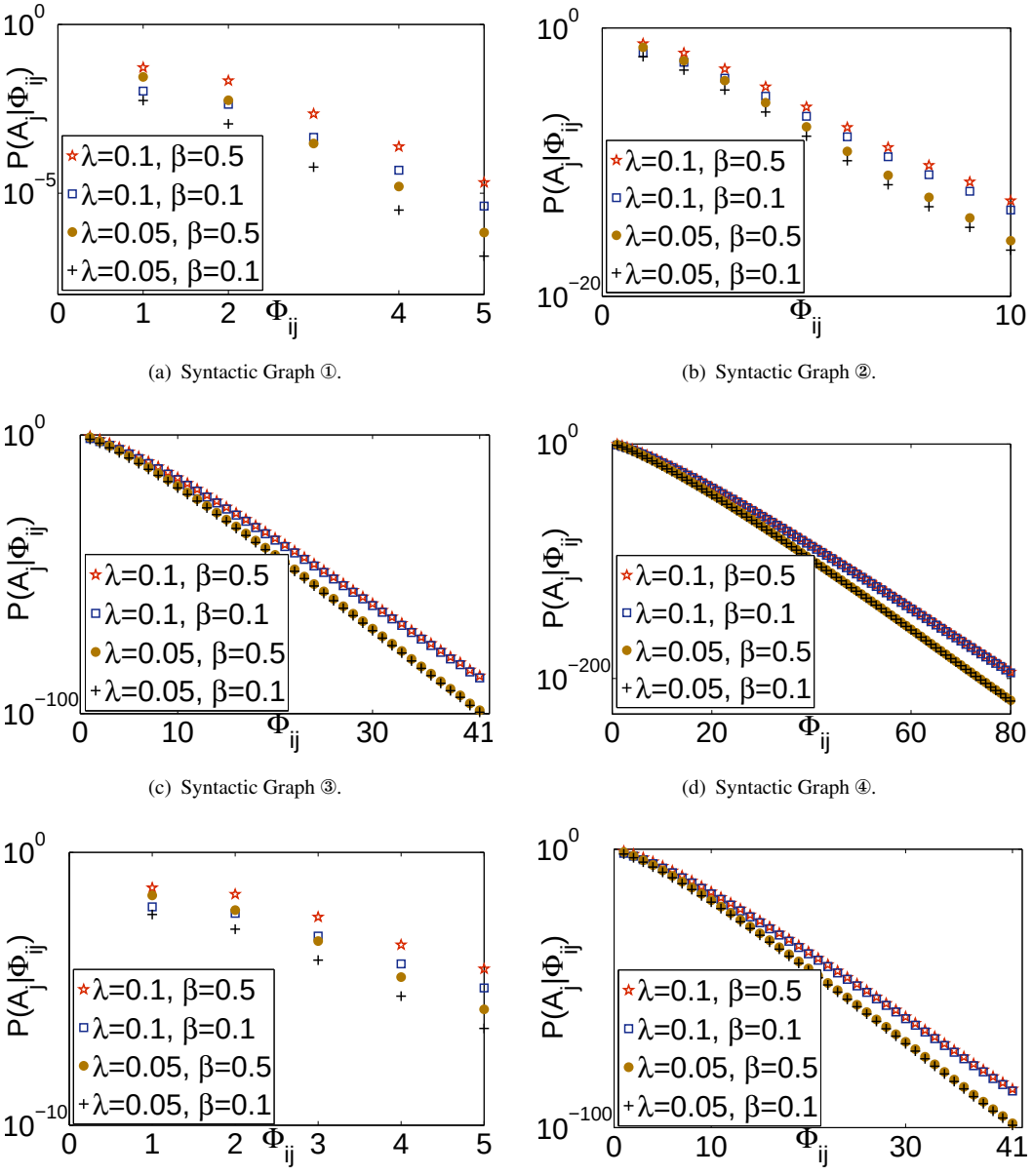
(c) Syntactic Graph ③.

(d) Syntactic Graph ④.

(e) Syntactic Graph ⑤.

(f) Syntactic Graph ⑥.

FIGURE 4.18: The group trust distributions of $P(\Phi_{ij})$ in syntactic graphs.



(a) Syntactic Graph ①.

(b) Syntactic Graph ②.

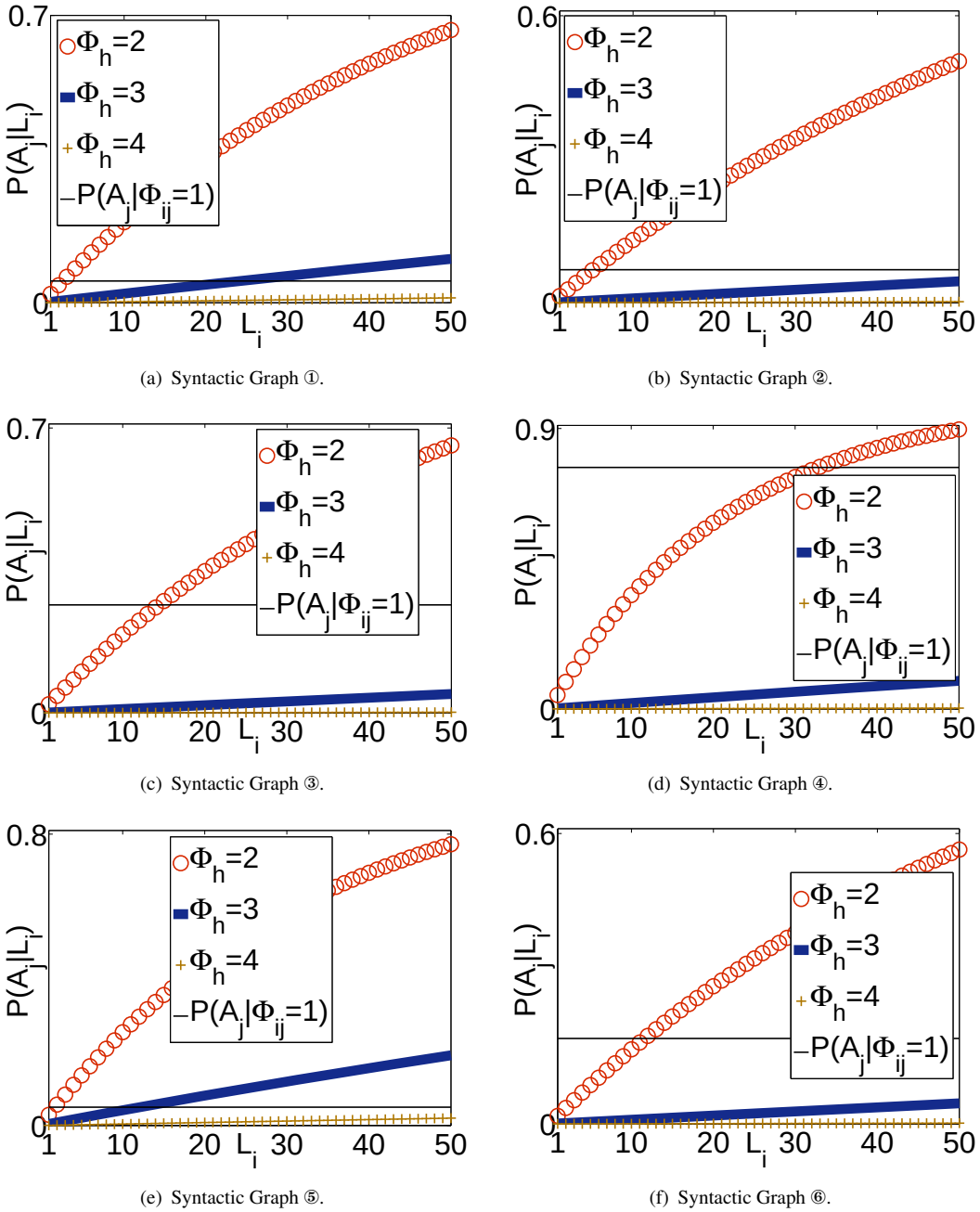
(c) Syntactic Graph ③.

(d) Syntactic Graph ④.

(e) Syntactic Graph ⑤.

(f) Syntactic Graph ⑥.

FIGURE 4.19: The probability $P(A_j|\Phi_{ij})$ with different Φ_{ij} in four settings of λ and β.



(a) Syntactic Graph ①.

(b) Syntactic Graph ②.

(c) Syntactic Graph ③.

(d) Syntactic Graph ④.

(e) Syntactic Graph ⑤.

(f) Syntactic Graph ⑥.

FIGURE 4.20: The probability $P(A_j|L_i)$ when L_i is from 1 to 50.

TABLE 4.4: The selection of L_i given a Φ_h for SGor to guarantee $P(A_j|L_i) < P(A_j|\Phi_{ij} = 1)$ in different Syntactic Graphs.

Dataset	Avg. degree	Graph density	Φ_h	L_i
Syntactic Graph ①	4.98	5×10^{-3}	$\Phi_h = 2$	$L_i \leq 2$
			$\Phi_h = 3$	$L_i \leq 26$
			$\Phi_h = 4$	L_i up to 50
Syntactic Graph ②	10.9	1.1×10^{-2}	$\Phi_h = 2$	$L_i \leq 5$
			$\Phi_h = 3$	L_i up to 50
			$\Phi_h = 4$	L_i up to 50
Syntactic Graph ③	52.0	5.2×10^{-2}	$\Phi_h = 2$	$L_i \leq 14$
			$\Phi_h = 3$	L_i up to 50
			$\Phi_h = 4$	L_i up to 50
Syntactic Graph ③	98.0	9.8×10^{-2}	$\Phi_h = 2$	$L_i \leq 33$
			$\Phi_h = 3$	L_i up to 50
			$\Phi_h = 4$	L_i up to 50
Syntactic Graph ③	5.00	2.5×10^{-3}	$\Phi_h = 2$	$L_i \leq 1$
			$\Phi_h = 3$	$L_i \leq 14$
			$\Phi_h = 4$	L_i up to 50
Syntactic Graph ③	53.5	2.7×10^{-2}	$\Phi_h = 2$	$L_i \leq 11$
			$\Phi_h = 3$	L_i up to 50
			$\Phi_h = 4$	L_i up to 50

Chapter 5

Active Approach for Certification based Trust Model

The authentication for Internet websites is critically important for web security, because it enables end users to verify the identities of Internet websites and hence prevent their sensitive data from leaking to the sites they do not trust. Today's web deploys a certification based trust model for site authentication [16, 17]. This model relies on a group of *trust anchors* (known as certificate authorities) to establish trust by means of issuing legal certificates to websites. End users authenticate remote websites by checking whether the site certificates are (directly or indirectly) issued by a trust anchor. Web servers implement this model using the HTTPS protocol (HTTP over SSL/TLS), and web browsers have already pre-installed a default set of trust anchors for website authentication purposes.

Although the certification based trust model has been placed on the web for more than two decades, it is recently found with a disastrous security flaw: the compromise of a single trust anchor can subvert the entire trust model globally. The root cause is that end users do not have the knowledge about which certificate authority is the legal issuer of which site certificate. As a result, the attackers who compromise one certificate authority (usually the weakest one) can issue faked site certificates which can be used to launch man-in-the-middle attacks. For example, two real-world certificate authorities, Comodo [32] and DigiNotar [33], were found compromised in 2011. Attackers employed them to issue faked certificates for popular websites, and launched man-in-the-middle attacks to successfully hijack around $300,000$ Iranian's Google mail connections [34].

Patching this dangerous flaw is a very challenging problem, because the man-in-the-middle attacks with different vantage points and attacking patterns pose different levels of threats to the trust model. The state-of-the-art countermeasures do not make full use of trust the certification based trust model provides, hence probably becoming ineffective against man-in-the-middle

variants. On the one hand, notary based solutions (such as Perspectives [35] and Crossbear [36]) deploy a third-party service to collect different copies of a suspicious site certificate through globally distributed hosts and alert a possible man-in-the-middle attack to end users if these copies are not the same. This countermeasure explores a portion of Internet path diversity to detect man-in-the-middle. However, the third-party service can be regarded as an additional trust anchor and the compromise of this trust anchor can also subvert the entire trust model. Even if the third-party cannot be compromised, this solution is still ineffective against the man-in-the-middle attacks whose vantage points are near to web servers [36]. On the other hand, some recent studies [37, 38] propose the use of pre-shared secrets (e.g., a password in [37] or an HTTP cookie in [38]) to enhance site authentication. Although this kind of countermeasures introduces new information from the design space outside trust model, it cannot establish trust on the first time authentication, which is a very important property the certification based trust model intends to protect.

In this chapter, we propose an active approach to harden the flawed certification based trust model. The basic idea is to maximize the protection against SSL/TLS man-in-the-middle attacks by actively seeking a priori trust from trust model (including available trust anchors and Internet path diversity) as much as possible. Our approach consists of four countermeasures, each of which has a unique tradeoff between the difficulty of deployment and the capability against man-in-the-middle variants. We believe our approach is the first solution that can maximize the use of trust for website authentication purposes.

In summary, we list our three major contributions in this chapter as follows:

1. We study how to maximize the use of trust in the certification based trust model for the first time.

2. We design four distinct countermeasures against SSL/TLS man-in-the-middle attacks with few compromised trust anchors. Our countermeasures have different capabilities against man-in-the-middle variants and require different efforts for deployment.

3. We have evaluated our approach using a public certificate data set and Internet experiments. The experimental results confirm the effectiveness of our approach.

The remainder of this chapter is organized as follows. We first the threat model of compromised trust anchor based man-in-the-middle attacks in Section 5.1. We then design our approach which consists of four countermeasures in Section 6.2.1, After that, we evaluate our countermeasures using a real-world certificate data sets and Internet experiments in Section5.3.

5.1 Threat Model

In this section, we describe the threat models for different man-in-the-middle variants against the certification based trust model on the web. These variants contain different man-in-the-middle vantage points and attacking patterns, hence posing different threats to the trust model.

Figure 5.1 illustrates three man-in-the-middle variants considered in this letter. All three are assumed with a single compromised trust anchor (i.e., the trust anchor B in Figure 5.1). For the man-in-the-middle variant ①, the vantage point is close to the victim web user (e.g., a gateway or a wireless access point of the victim user is compromised by man-in-the-middle attackers). Moreover, this threat model considers that the attackers indiscriminately hijack all the HTTPS connections from the victim user to any remote websites (i.e., the attacking pattern is non-selective hijacking). Compared with variant ①, the man-in-the-middle variant ② is smarter because it will only intercept the connections to its interested websites (i.e., the attacking pattern is selective hijacking). The real-world attack reported by [34] falls into threat model ②. In contrast to ① and ② in which the vantage point is near the web user, threat model ③ considers that the man-in-the-middle vantage point is nearby web servers. This man-in-the-middle variant is harder to detect, because nearly all the Internet paths to the victim web server necessarily pass through this kind of vantage point. There is also another new variant if ③'s hijacking pattern becomes selective. But we do not discuss this variant in this letter, because it makes lesser impact as compared with variant ③ and it is also easier to be detected.

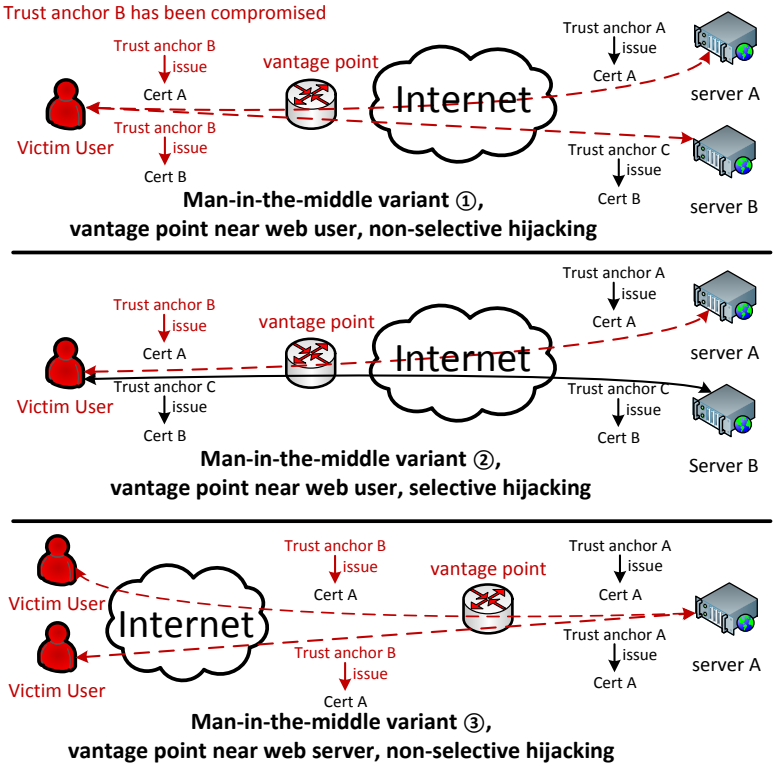


FIGURE 5.1: Three threat models for different man-in-the-middle variants.

5.2 Design of Active Approach

We elaborate on the details of our active approach in this section. Our approach contains three client-side countermeasures and one server-side countermeasure against the three man-in-the-middle variants described in Section 5.1. These countermeasures maximize the protection by exhausting available trust anchors and exploiting Internet path diversity.

5.2.1 Formalization

Let T be the set of trust anchors in the certification based trust model on the web. Let W be the set of web servers and U be the set of web user hosts in the Internet. Let $P = \{u \rightsquigarrow w | u \in U, w \in W\}$ be the set of Internet paths from any user host to any web server. Let S be the set of websites in the Internet. Since a website can be served by multiple web servers behind, we have a subset of web servers $W_s \subseteq W$ to serve a website $s \in S$. We use $|*|$ to denote the size of a set, where $*$ could be T, W, U, P, S or any other set. Let C_w be the certificate placed on the web server $w \in W$ for site authentication. Let $I(C_w)$ be the issuer of C_w. In certification based trust model, a web server w can be authenticated as a legal server that serves website $s \in S$ if Eq. (5.1) is satisfied.

$$I(C_w) \in T, \ where \ w \in W_s. \tag{5.1}$$

In this letter, we design countermeasures against man-in-the-middle attacks by actively exhausting available trust anchors in T and exploiting Internet path diversity from P.

5.2.2 Client-side Countermeasures

To counter the three different man-in-the-middle variants, we propose three corresponding client-side solutions.

5.2.2.1 Client-side Countermeasure ①

We propose client-side countermeasure ① to defeat man-in-the-middle variant ①. This countermeasure is based on a key insight that a large number of site certificates a web user actively collects could be issued by the same trust anchor if man-in-the-middle variant ① is launched with a single compromised trust anchor. As a result, we design a client-side countermeasure ① by enabling a web user to actively connect n different websites for requesting site certificates and confirm a man-in-the-middle if these n certificates are issued from less than μ trust anchors. These n websites can be randomly selected or intended to choose if the web user knows

some site certificates' issuers in advance. Let $S_n \subseteq S$ be the set of the n selected websites. A man-in-the-middle is confirmed if Eq. (5.2) is satisfied.

$$|\{I(C_w) \in T | w \in W_s, s \in S_n\}| < \mu. \tag{5.2}$$

The parameter n indirectly determines the number of trust anchors that can be used to defeat man-in-the-middle variant ①. A larger n leads to a higher probability that the collected site certificates are issued by more different trust anchors. If we choose n websites whose certificates are issued from $|T|$ different trust anchors, all the available trust anchors can be used to resist man-in-the-middle. The parameter μ defines the number of compromised trust anchors our countermeasure can tolerate (i.e., our countermeasure can detect the man-in-the-middle with up to μ compromised trust anchors). A larger μ can protect the trust model against man-in-the-middle with more compromised trust anchors, but meanwhile suffers from a higher false positive, especially if n is not large enough. We will show how to select appropriate n and μ in Section 5.3.1.

5.2.2.2 Client-side Countermeasure ②

Client-side countermeasure ① is ineffective against man-in-the-middle variant ②, because this variant only intercepts few site certificates the attackers are interested in.

To detect this variant on the client side, additional SSL/TLS tunnels are required. We therefore propose a client-side countermeasure ② by actively deploying m globally distributed SSL/TLS tunnel points to defeat the man-in-the-middle variant ②. The Internet paths from tunnel points to websites can bypass the man-in-the-middle vantage points which is near the web user. However, the Internet paths from a web user to tunnel points are still subject to the man-in-the-middle. To take care of this issue and maximize the protection, we install certificates issued from different trust anchors in different tunnel points. Let H be the set of tunnel points deployed in the Internet. Let C_h^1 be the certificate installed in SSL/TLS tunnel point $h \in H$ and C_h^w be the web server w's certificate received through tunnel point $h \in H$. Our countermeasure detects a man-in-the-middle if the received tunnel point certificates are issued from less than m trust anchors or a web server's certificates received through two different tunnels are different. To be precise, a man-in-the-middle can be confirmed if Eq. (5.3) is satisfied.

$$|\{I(C_h^1) \in T | h \in H\}| < m \text{ or } \exists h, h' \in H, C_h^w \neq C_{h'}^w. \tag{5.3}$$

By this way, to hijack the connections between a web user and m deployed tunnel points, man-in-the-middle attackers are required to compromise m additional trust anchors. Obviously, deploying $m = |T|$ tunnel points can maximize the protection because all the available trust anchors are used.

5.2.2.3 Client-side Countermeasure ③

Countermeasure ② cannot defeat man-in-the-middle variant ③, because it can hardly set up a tunnel point to access a web server without passing the man-in-the-middle vantage point closed to this server. To address this problem, we propose a client-side countermeasure ③ by enabling web users to actively collect site certificates from other web servers of the same website. The key insight of this countermeasure is that popular websites usually deploy a large distributed system of servers across the Internet to serve end users from different regions with high availability and performance (i.e., content delivery network). These web servers are likely to use the certificates issued from the same trust anchor, while the man-in-the-middle vantage point near one web server is unlikely to be close to other servers, especially the ones located in a different geographical region. This countermeasure detects a man-in-the-middle if the certificates received from different web servers of the same website are issued by different trust anchors. Recall that W_s is a set of web servers serving behind website s, then a man-in-the-middle can be confirmed if Eq. (5.4) is satisfied.

$$\exists w, w' \in W_s, I(C_w) \neq I(C_{w'}). \tag{5.4}$$

5.2.3 Server-side Countermeasure

We propose a unified solution to defeat all the three man-in-the-middle variants on the server side. Our solution enables web servers to actively deploy η certificates issued by η different trust anchors for site authentication. Web users recognize a man-in-the-middle if the η received certificates from the same web server are issued by less than η trust anchors. Let $C_w[j]$ be the j-th certificate received from the server w, where $j = 1, ..., \eta$. A man-in-the-middle is confirmed if Eq. (5.5) is satisfied.

$$|\{I(C_w[j]) \in T | j = 1, 2, ..., \eta\}| < \eta. \tag{5.5}$$

By this mean, to subvert the entire trust model, man-in-the-middle attackers need to compromise at least η trust anchors. Obviously, deploying $\eta = |T|$ certificates in a web server can maximize the protection of this web server as available trust anchors are exhausted.

5.2.4 Comparison in the Literature

We compare our approach with existing countermeasures in Table 5.1. As can be seen, our approach is the first solution that can defeat all three man-in-the-middle variants with up to $|T|$ compromised trust anchors and protect both the first-time authentication and subsequent authentications. We achieve this result by utilizing a priori trust knowledge from the certification based trust model as much as possible. Although the server-side countermeasure has the best capability against man-in-the-middle attacks as compared with the other three client-side countermeasures, this countermeasure is more difficult to deploy, because it requires the modification of SSL/TLS internals in both client side and server side.

5.3 Evaluation

Although we can fully control the parameters of client-side countermeasure ② (i.e., m) and server-side countermeasure (i.e., η) for guaranteing their effectiveness, some of the parameters in client-side countermeasures ① (i.e., μ) and ③ (i.e., W_s) are out of our direct control. For this reason, we evaluate client-side countermeasures ① and ③ in this section. We also evaluate the performance overhead introduced by our approach.

5.3.1 Evaluation of client-side countermeasure ①

We employ a publicly available certificate data set released by [70] for our evaluation. The authors of [70] scanned the entire IPv4 space in 2010 and successfully collected more than 1.5 million valid certificates. These certificates are issued from 963 certificate authorities. In our experiments, we only consider the authorities which issue more than $5,000$ certificates in the data set as trust anchors. The resulted data set contains $1,426,139$ valid site certificates issued from 32 trust anchors. That is, $|T| = 32$ in our experiments.

We regard an experiment round as a uniformly random selection of n certificates from the data set. We consider $n = 100$ to $1,000$ and conduct $1,000$ rounds of experiment for each n. In each round, we investigate the number of distinct issuers (i.e., trust anchors) of these n certificates. The parameter μ is determined according to this investigation. Figure 5.2 shows the results. We can see that, for $n = 100$, we should not use $\mu > 15$ because it could result in false positives. While for $n > 800$, it is safe for us to use $\mu = 30$, which approximates the maximum protection $\mu = |T| = 32$.

TABLE 5.1: A comparison of our approach and the existing solutions in the literature.

man-in-the-middle variants	# of compromised trust anchors	Implementation Level	Implementation Side
Certification based trust model [16, 17]	≤ 1	N/A	N/A
Notary-based solutions [35, 36]	≤ 1	Application	Client
Notary-based solutions [35, 36]	≤ 2	Application	Client
Pre-shared secrets [37, 38]	$\leq \|T\|$	SSL/TLS internal	Client+Server
Our approach (client-side ①)	$\leq \mu \rightarrow \|T\|$	Application	Client
Our approach (client-side ②)	$\leq m \rightarrow \|T\|$	Application	Client
Our approach (client-side ③)	$\leq \|T\|$	Application	Client
Our approach (server-side)	$\leq \eta \rightarrow \|T\|$	SSL/TLS internal	Client+Server

man-in-the-middle variants	First-time authentication variant ①	variant ②	variant ③
Certification based trust model [16, 17]	×	×	×
Notary-based solutions [35, 36]	✓	✓	×
Notary-based solutions [35, 36]	×	×	×
Pre-shared secrets [37, 38]	×	×	×
Our approach (client-side ①)	✓	×	×
Our approach (client-side ②)	✓	✓	×
Our approach (client-side ③)	×	×	✓
Our approach (server-side)	✓	✓	✓

	Subsequent authentications variant ①	variant ②	variant ③
Certification based trust model [16, 17]	×	×	×
Notary-based solutions [35, 36]	✓	✓	×
Notary-based solutions [35, 36]	×	×	×
Pre-shared secrets [37, 38]	✓	✓	✓
Our approach (client-side ①)	✓	×	×
Our approach (client-side ②)	✓	✓	×
Our approach (client-side ③)	×	×	✓
Our approach (server-side)	✓	✓	✓

5.3.2 Evaluation of client-side countermeasure ③

We conduct Internet experiments to evaluate the client-side countermeasure ③. We choose Google as the victim website because it has been attacked by real-world man-in-the-middle [34]. Figure 5.3 shows the Internet paths from a machine located in Hong Kong to two Google servers in Hong Kong and one Google server in Japan. The site certificates collected from these

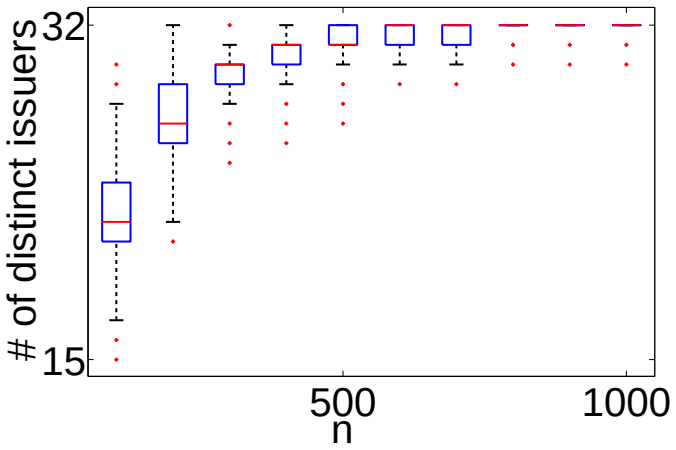


FIGURE 5.2: The number of issuers of n randomly selected certificates.

three Google servers are issued from the same trust anchor. It can be seen that if attackers have compromised the router 216.239.43.17 which nears the Google server 1 in Hong Kong, we can detect this man-in-the-middle attack by fetching the certificate from the Google server 2 in Hong Kong or the Google server in Japan.

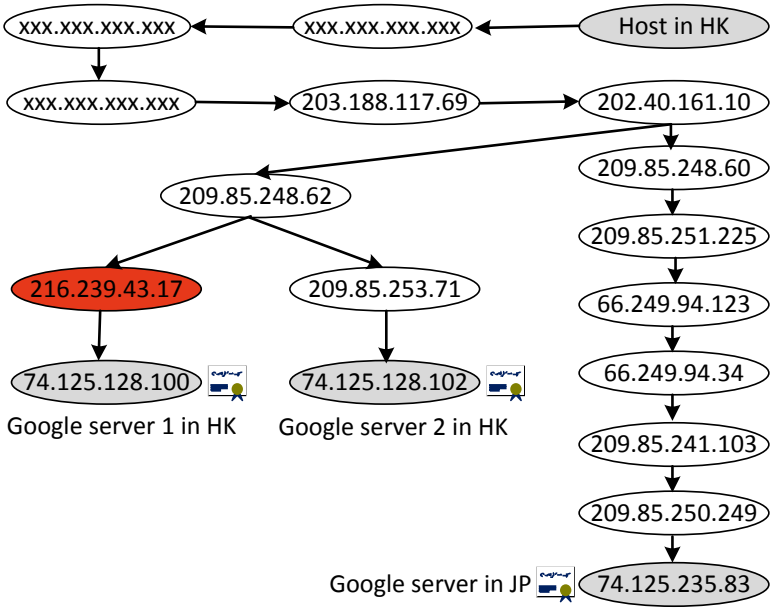


FIGURE 5.3: Internet paths from a host in Hong Kong to three different Google servers.

We conduct further experiments to traceroute the Google servers deployed in six other countries, including Singapore, Australia, America, Canada, German and Russia. We measure Internet path diversity between two paths using the number of different routers between them. Table 5.2 shows the diversity between the paths to Google server 1 in Hong Kong and eight other Google servers located in different regions. A larger number indicates a higher probability of evading man-in-the-middle vantage point near the Google servers.

TABLE 5.2: The number of different routers between the paths to the Google server 1 in Hong Kong and other Google servers located in different regions.

Country	HK(2)	SG	JP	AU	US	CA	DE	RU
# of different routers	1	2	6	6	13	6	16	13

We also note that not all administrative domains have deployed CDN. Therefore, we investigate the Alexa top 1000 domains (500 of them open HTTPS services) and report the CDN support rate based on these 500 HTTPS domains. In particular, we scan these 500 domains' underlying web servers from seven regions including Hong Kong, Japan, Singapore, Australia, Canada, German and Russia. We successfully find 266 domains supported by more than one web server. We also examine the certificates deployed in these domains' web servers. We confirm that for 256 domains the web servers belonging to the same domain have deployed site certificates issued from the same trust anchor. For the other 10 domains, we conduct a careful check and find the certificates that are issued from different trust anchors are either self-signed certificates or the certificates signing for other domains. Our client-side countermeasure ③ cannot be impeded by these erroneous certificates, because they can be easily identified and removed. After excluding the web servers deploying erroneous certificates, these 10 domains still contain more than one web server. As a result, our client-side countermeasure ③ can be carried out in these 266 out of 500 (i.e., 53.2%) domains.

5.3.3 Evaluation of performance overhead

Our approach could introduce additional connection delay and computation cost due to additional certificate verification. We evaluate these performance overheads by simulating additional tasks in parallel. Our performance evaluation is performed using a virtual machine located in our laboratory with 8 core 2.67GHz CPU and 6 GB memory.

We run our evaluation for each countermeasure 100 times. For the evaluation of the client-side countermeasure ①, we randomly choose $n = 100$ to $n = 1000$ IP addresses from the certificate data set [70]. For the evaluation of the client-side countermeasure ②, we randomly select $m = 32$ PlanetLab nodes around the world as SSL/TLS tunnel points and connect back to the Hong Kong Google server through these nodes. For the client-side countermeasure ③, we conduct the evaluation based on the Google servers listed in Table 5.2. For the server-side countermeasure, we consider $\eta = 32$. We present our result in Figure 5.4 and Table 5.3. As can be seen, if we do not choose a large n (i.e., $n \leq 100$) for the client-side countermeasure ①, the performance cost (computation cost plus connection delay) is relatively small (i.e., less than 1.2 seconds). Note that, the connection delay for server-side countermeasure is N/A, because this countermeasure does not introduce connections to additional web servers. Instead, it only

introduces additional connections to the same web server, and these connections cannot result in larger RTTs. We also note that our approach could introduce different overheads if we choose different countermeasure parameters.

TABLE 5.3: Performance overheads introduced by our approach.

Countermeasure	Client-side ①	Client-side ②	Client-side ③	Server-side
Computation cost	Fig. 5.4	461 ± 39 ms	55 ± 10 ms	202 ± 30 ms
Connection delay	Fig. 5.4	744 ± 25 ms	341 ± 17 ms	N/A

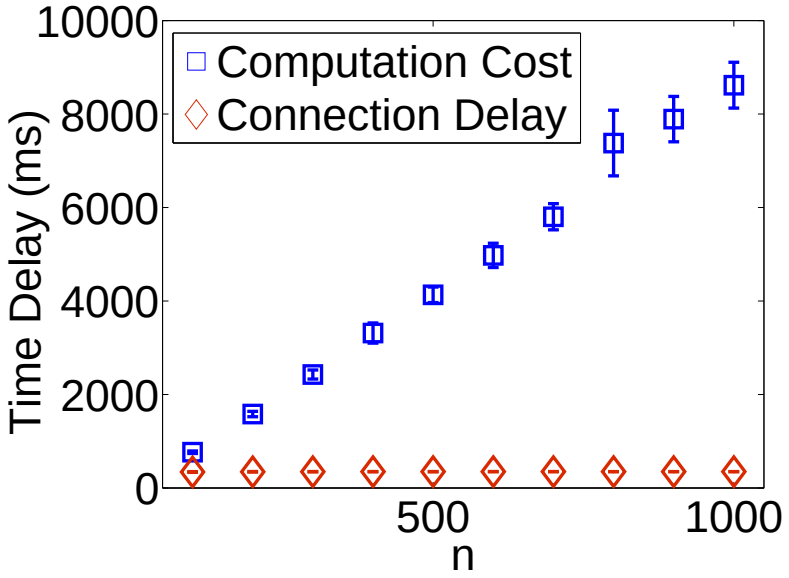


FIGURE 5.4: The performance overheads in terms of connection delay and computation cost for client-side countermeasure ①.

Chapter 6

File Download Vulnerability Study and New Defense

Today's web is now becoming a readable and writable medium for sharing and managing files that are stored on globally distributed web servers. Along with this trend, today's web consists of a large number of web applications which provide file management services (such as file upload, download and online editing etc.). These applications usually implement a range of file operation functions by means of server-side scripts, and releases these functions in the form of dynamic URLs. Among these functions, file download is a very basic primitive that is responsible for publishing and delivering web managing files to remote web users. For the ease of description, we regard the files stored on web server's filesystem but managed by web applications as *web managing files*, and other files on the filesystem as *local files*.

A file download vulnerability occurs if remote web users can download unauthorized (local) files from a web server through a file download script. These files could be source code of server-side scripts, web configuration files, critical system files and unauthorized web managing files. The root cause of this vulnerability is the ambiguous permission server-side scripts use for local filesystem access. More precisely, server-side script runs on web server permission with a particular user and group privilege. This permission is too coarse-grained to isolate web managing files from local filesystem and cannot differentiate the files managed by different web applications.

Defenses against file download vulnerability have been existing for a long time. As a result, some may believe this is a solved problem. However, we find that existing defenses still have a number of practical drawbacks, hence making this vulnerability still prevalent in today's web. On the one hand, sanitizing sensitive user inputs before using these inputs as file names to access local filesystem is an effective defense. However, this defense requires customized implementations for different file download logics and therefore becomes easy to introduce insufficient or

erroneous protections. Moreover, since an enterprise web server often consists of legacy scripts that are deployed by different web administrators, at different times, and for different purposes, it is very difficult to assure all these scripts are not vulnerable. On the other hand, directory based permission control is another kind of mainstream defenses. Although this defense can restrict web server permission to some specified directories and forbids any file read operations outside, it is not flexible and falls victim to multiple file download scripts. For example, if the web server permission is specified to a web application's home directory, the local files such as server-side scripts and web configuration files inside this home directory cannot be protected. Even if the specified directories do not contain any local files, the files managed by one web application can also be illegitimately downloaded through other irrelevant file download scripts running in the same web server.

By sampling the landscape of file download vulnerabilities across different domains and countries, we confirm the weak protection of file download scripts in today's web. In particular, we have collected $19,137$ suspicious file download scripts by means of Google search engine, and successfully discovered $6,060$ (or 31.7% of our sample set) vulnerabilities. Among these vulnerabilities, we have found $5,807$ (or 30.3%) can download script source codes, $4,285$ (or 22.4%) can retrieve web configurations, and $1,987$ (or 10.4%) can expose critical system files. Our further experiments confirm a high probability of disastrous consequences that our discovered vulnerabilities can induce (i.e., 12.8% of our discovered vulnerabilities can result in system intrusion and 17.8% can lead to database intrusion). Our results motivate us to reconsider new defense to enhance the protection of file download scripts.

The design of our defense targets to provide (1) error-proof protection (providing unified protection regardless of specific file download logics), (2) flexible protection (providing per-file permission control), and (3) mutually independent protection (providing effective protection against web managing files leaking through irrelevant file download scripts). We therefore propose a new defense called FileGuard, which takes advantage of script engine to embed dedicated ownership information into web managing files. This ownership information is different from the ownership information recognized by operating systems, as well as can be used to isolate web managing files from local filesystem and differentiate the files managed by different web applications. Our design chooses extended file attributes as the storage to embed the dedicated ownership, although we note that other file properties can also be used for this purpose.

FileGuard is novel in three aspects. First, compared with user input sanitization, FileGuard is error-proof because it provides a unified protection in script engine layer and thus cannot be affected by specific file download logics. Second, in contrast to directory based permission control, FileGuard is flexible since its permission control can be performed on file basis. Third, FileGuard can embed different ownership to the files managed by different web applications, hence preventing web managing files from being downloaded through irrelevant file download

scripts. We have implemented a prototype of FileGuard on top of PHP5 script engine and evaluated this prototype in terms of effectiveness, performance overhead and development costs. Our experimental results show that, compared with existing defenses, FileGuard can provide better protection against file download vulnerabilities but induces only a few performance overheads and development costs.

To sum up, we make two major contributions in this chapter:

- We report on $6,060$ (or 31.7% of our sample dataset) file download vulnerabilities across different domains and countries. Our results reveal the prevalence of file download vulnerabilities in today's web, and implicitly confirm the practical drawbacks of existing defenses.
- We propose FileGuard, a new defense that can provide error-proof, flexible and mutually independent protections against file download vulnerabilities.

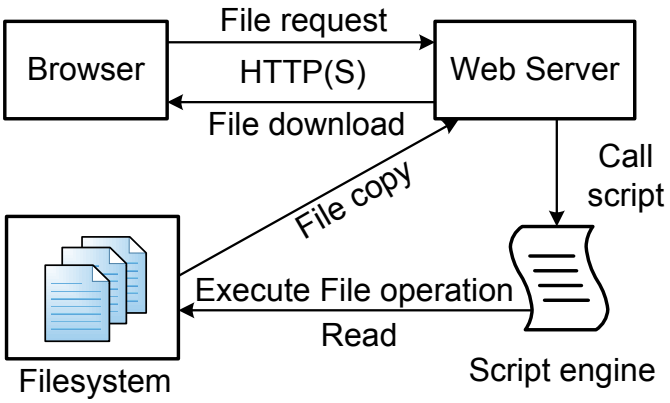
The remainder of this chapter is organized as follows. Section 6.1 revisits the background of file download scripts and practical drawbacks of existing defenses. Section 6.2 reports our sampling results of file download vulnerabilities in today's web and potential attacks using these vulnerabilities. Section 6.3 presents the design of FileGuard and the implementation of its prototype. Section 6.4 evaluates the performance of our FileGuard prototype implementation.

6.1 Background

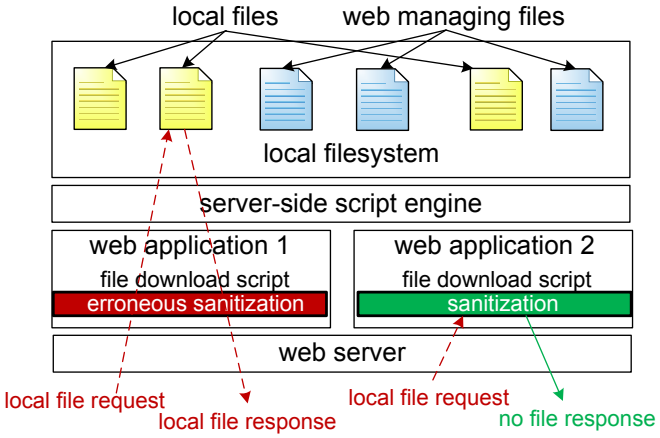
In this section, we elaborate on why file download scripts on the web are vulnerable, as well as survey existing defenses for this issue.

6.1.1 Threats to File Download Scripts

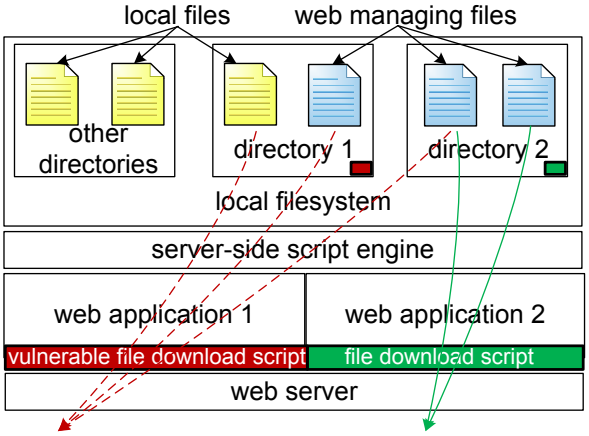
Modern web servers are designed to run on a particular user and group permission for accessing local filesystem. Server-side scripts inherit this permission to perform authorized file system operations, such as reading allowed files as requested by remote web users and sending these files back to the users for their downloading. Figure 6.1(a) illustrates a file download process on the web. A remote web user initiates a file download request through his browser. This request is sent to web server using HTTP or HTTPS protocol in the Internet. Upon receiving this request, web server calls corresponding server-side script read functions to copy the file from local filesystem to memory, and then send it back to the web user in HTTP response packets.



(a) A file download process on the web.



(b) User input sanitization.



(c) Directory based permission control.

FIGURE 6.1: File download process and existing defenses.

According to Figure 6.1(a), we find that ① remote web users are in charge of determining which file (possibly including file path) they attempt to download, ② server-side scripts use web server permission to read files in response to remote web users. These two findings indicate a potential threat to file download scripts. That is, a malicious web user can request arbitrary files the web server has read permission on. A higher permission the web server runs on can lead to a more

serious threat. For example, if a web server is configured with root privilege, server-side scripts can use this privilege to read entire local filesystem. Although modern web servers use a lesser privilege in default, this privilege is still too coarse-grained to protect file download scripts.

First, this permission cannot isolate all the local files from web managing files. We consider an Apache web server running in Linux operating system as an example to show the coarse privilege issue. In this example, Apache is configured to run with a user and group account *www-data* by default. Although this setting is intended to only allow read permissions on the files whose owner or group is www-data (these files are web managing files because they are uploaded and created by server-side script which runs on web server permission), it also causes implicit read access to other world-readable files in the system. A file is world-readable if the *others reference mode* of this file is set to "r". In Linux, a number of local files are necessarily world-readable, because they are shared with all the Linux users. For example, any Linux user can perform the command "ls -l" for specified files and directories to display their ownership information, but this information should be read from a password file `/etc/passwd`. As a result, the password file is necessarily world-readable and runs a high risk of being illegally downloaded with the default web server permission. The coarse privilege issue is not limited to Apache and Linux, any other web servers and operation systems suffer from the same problem. Second, the default web server permission can neither differentiate web managing files managed by different web applications, because all the web applications run on top of the same web server.

6.1.2 Drawbacks of Existing Defenses

In today's web, a number of defenses have been proposed to protect file download scripts. These defenses can be roughly grouped into two categories. One is user input sanitization, and the other is directory based permission control.

6.1.2.1 User Input Sanitization

From script developers' point of view, user input sanitization is an effective defense that they can implement to protect their own file download scripts. The basic idea is to sanitize dangerous characters from the URL parameters which convey user specified file and path names. However, since different file download scripts usually have different download logics, the implementations of user input sanitization are quite different and easy to introduce errors. Figure 6.1(b) shows a file download vulnerability due to the erroneous sanitization. We note that there necessarily exist a large number of possibilities that could lead to erroneous sanitization, but we discuss only four typical ones here to help readers understand this problem.

Absolute Path or Relative Path: Some file download scripts sanitize parent path operand (i.e., ../) to prevent unauthorized local filesystem access. Although this kind of sanitization is effective for relative path traversal, it cannot protect the scripts which treat user inputs as absolute file paths for filesystem access, because absolute path can traversal the entire filesystem without requiring parent path operands. Therefore, if a file download script only sanitizes parent path operand but use absolute file path to access local filesystem, this sanitization is an erroneous protection.

Partial Sanitization: Some file download scripts can use multiple URL parameters to convey file paths, but only sanitize some of these parameters. The partial sanitization could lead to file download vulnerabilities. Code listing 6.1 shows a real-world example. As can be seen, although the vulnerable script prevents the download of php source codes by sanitizing the parameter "file_extension" (Lines 3-8 in code listing 6.1), malicious users can easily craft a download URL with parameters like "source=/download.php&file_extension=pdf" to bypass this protection, because the effective file path is conveyed through the parameter "source" (Lines 2 and 10 in code listing 6.1).

```
1  <?php ... ...
2  $file = substr(trim(urldecode($_GET['source'])),1);
3  $file_extension = trim(strtolower(urldecode($_GET['file_extension'])));
4  ... ...
5  switch( $file_extension ) {
6  ... ...
7  //The following are for extensions that shouldn't be downloaded (sensitive stuff, like
       php files)
8  case "php": die("<b>Cannot be used for ". $file_extension ." files!</b>"); break;
9  ... ... }
10 @readfile($file);
11 ... ... ?>
```

LISTING 6.1: Partial Sanitization Issue

Incomplete Black List: Some file download scripts handle a black list of extension names to prevent local files with these extension names from being illegitimately downloaded. However, it is very difficult to have a complete black list. The code listing 6.2 shows a real-world example of incomplete black list. In this example, the script uses a black list *not_to_be_dloaded* for sanitization. This list is apparently incomplete, because the Linux password file `/etc/passwd` and many other critical system files cannot be protected using this list.

Programming Errors: Human-made errors are the primary cause of erroneous user input sanitization. This kind of errors is nearly impossible to be eliminated, because it depends on

the programming capabilities and experiences the script developers have. Back to the example in the code listing 6.2, this script misuses the function *substr()* by setting its second parameter to 1. This setting could strip the first character of the extension name. That is, "fname=download.php" can result the extension name to "php", and hence bypass the protection because the forbidden extension name stored in the *not_to_be_dloaded* array is ".php".

```
1  <?php ... ...
2  $fn = $_GET['fname'];
3  ... ...
4  $not_to_be_dloaded = array(".htm", ".html", ".shtml", ".dhtml", ".php");
5  $extensie = strrchr($fn, ".");
6  if (!in_array(substr($extensie, 1), $not_to_be_dloaded) && (strpos($fn, "..") ===
       false)) {
7  ... ...
8  readfile($fn);
9  } else {
10 // message when downloading a forbidden file
11 echo "$msg_not_allowed";}
12 ?>
```

LISTING 6.2: Incomplete Black List and Programming Errors

6.1.2.2 Directory based Permission Control

From web administrators' point of view, server-side permission control is a very promising defense, because it can protect file download scripts without considering the detailed logics of these scripts. This kind of defenses usually restricts web server permission to some predefined directories and cannot perform fine-grained access control inside these directories. That is, this defense cannot protect the local files stored on these pre-defined directories. Moreover, the directory based permission control cannot differentiate which directory is configured for which web application, hence becoming ineffective to prevent web managing files from being illegitimately downloaded through irrelevant file download scripts.

Figure 6.1(c) shows an example of directory based permission control. In this example, web server is configured to have read permission for directories 1 and 2. The directory 1 is intended to store web managing files of web application 1, while the directory 2 is for web application 2. We assume web application 1 has a vulnerable file download script. Due to the directory based permission control, malicious web users cannot retrieve local files outside the directory 1 through the vulnerable script. However, the local files inside directory 1 are still possible to be downloaded. Even worse, malicious users can exploit this vulnerable script to bypass web application 2's authentication and illegally download web managing files stored on the directory 2.

6.2 Vulnerability Survey in Today's Web

In this section, we sample the landscape of file download vulnerabilities across different domains and countries, and surprisingly confirm the weak protection of file download scripts in today's web. In particular, we first describe the methodologies that we use for sampling file download vulnerabilities in Section 6.2.1. We then report the vulnerabilities we have discovered in Section 6.2.2. We also discuss the possibly disastrous consequences of file download vulnerabilities in Section 6.2.3.

6.2.1 Sampling Methodologies

We sample file download vulnerabilities from today's web in two steps. In step one, we collect suspicious file download URL samples by means of Google search engine. In step two, we test these URL samples and check whether they are vulnerable.

6.2.1.1 Step One, Collecting Suspicious URL Samples

We collect suspicious file download URLs from the Internet by means of Google search engine. We are interested in the sample URLs which are named as *download.ext*. The *ext* could be *php*, *jsp*, *aspx* or *asp*. They represent four types of server-side script language accordingly: PHP (i.e., Hypertext Preprocessor), JSP (i.e, Java Server Pages), ASP.NET (with a ".aspx" extension) and its predecessor ASP (i.e., Active Server Pages). We choose these four because they are used by more than 95% Internet websites in today's web [136].

We implement a crawler to automate our collection process. Our crawler manipulates Firefox to access Google search pages using Selenium WebDriver [137]. However, since Google sets up several restrictions to prevent illegal use of their results, we should work around these restrictions before running our collection process. On the one hand, Google will present CAPTCHA if it suspects the user of the search engine is not a human. Solving Google's CAPTCHA without human efforts is very challenging. Although several automated tools (such as CAPTCHA Monster [138]) has been released for this problem, the success rate of breaking Google's CAPTCHA cannot be guaranteed. For this reason, we do not use this kind of tools for our experiment. Instead, we propose two methods to reduce human interactions during the crawl of Google results. One is setting the number of results that can be shown in one page from the default value 10 to the maximum value 100. This setting can largely reduce the frequency that our crawler should interact with Google, hence mitigating the probability of encountering CAPTCHA. The other method is to set up a list of Firefox profiles in advance. These profiles contain different HTTPS proxies and user session settings. In case a CAPTCHA appears, our crawler can

choose a new profile to continue the collection because Google's detection is bound to IP address and user account. After all the prepared profiles are exhausted, manual efforts are required to fix CAPTCHA for all these profiles in one time. We note that our crawler works in a semi-automated manner but it is easy to make our crawler fully automated if we can find an effective Google CAPTCHA solver.

On the other hand, Google search engine will only report the first one thousand relevant results even if there are thousands of millions of results that match users' key words. Even worse, these reported results usually contain a large number of duplicated URLs (the values of URL parameters are different). As a result, to collect suspicious URLs as much as possible, we should append dummy key words associated with *download.ext* to drive Google search engine to report more results. Our used key word pattern is like below:

```
allinurl:download.ext? key filetype:ext site:top.country
```

The *allinurl*, *filetype* and *site* are Google search operators [139], while the blue words are the parameters in our pattern. Each parameter can be enumerated as different values in our experiments as follows. *ext* = (*php*,*jsp*,*aspx*,*asp*), *key* = (*Empty*,*file*,*path*,*filename*,*filepath*,*fname*), *top* = (*Empty*,*edu*,*ac*,*gov*,*org*,*com*,*co*,*net*,*biz*) and *country* could be *Empty* or 254 country domain suffixes such as *uk*, *cn*, and *ca*. We note that we can collect more suspicious URL samples if we choose more values for these pattern parameters.

We run our crawler to collect suspicious URL samples from the 7th May 2013 to the 12th May 2013. We remove the URLs whose domains cannot be connected and finally have 19,137 valid suspicious URL samples in total.

6.2.1.2 Step Two, File Download Vulnerability Discovery

In this step, we target to detect file download vulnerabilities from the URL samples we have collected in step one. We first employ regular expression to recognize the URL parameters that appear to transmit file name or path or path concatenating file name. We then replace original file name with a representative server-side script name and enumerate possible paths. We confirm a vulnerability if we can successfully download the representative script in our test. We also choose representative web configuration file and critical system file, and test whether these files can be downloaded through the suspicious URL.

In our experiment, a suspicious download URL usually has the following format:

```
http(s)://domain/path_to_script/download.ext?para1=value1&para2=value2&...&paraN=
    valueN
```

In this example, there exist N URL parameters (i.e., para$n$ is the name of a parameter while value$n$ is the value of the parameter para$n$, $n = 1, ..., N$). To download an expected file, web users need to specify the file name and sometimes the *path_to_file* in corresponding URL parameters. We apply regular expressions to identify which parameter is likely to contain this kind of information. We recognize the file name according to the pattern of its extension name (i.e., *.pdf*, *.doc*, *.mp3* etc..). We identify the *path_to_file* using different regular expressions for two different cases, one is absolute path and the other is relative path. To recognize absolute *path_to_file*, we employ regular expression to match the string that is started with / or $[A-Za-z]:$ (e.g., $/uploads$ and $C:/uploads$ are both absolute paths). For the relative *path_to_file*, it should be a string that contains the special character / but does not match the regular expression of absolute *path_to_file* (e.g., $uploads/news$ is a relative path). Unfortunately, some relative *path_to_file*s may not contain /, such as *download.php?dir=news* where *news* is a relative directory but does not contain the character /. To take care of this issue, we have two candidate approaches. One is to blindly consider all the parameters whose value does not contain / and extension name as the relative *path_to_file*, and the other is to regard a parameter as relative *path_to_file* if this parameter's name contains sub string "dir" or "path" or "fold". We choose the second approach because it can cover almost cases in our experiment. Moreover, since some paths could use \ rather than /, we preprocess parameter values by replacing all the \ with / before the parameter recognition.

Table 6.1 lists the regular expressions that we use for parameter recognition. If a parameter matches both the file name and *path_to_file*, this parameter is recognized as a path concatenating file name, such as $/uploads/a.pdf$.

TABLE 6.1: Regular expressions for parameter recognition.

	Parameter	Regular expression
File name	value	$\backslash.[A-Za-z0-9]\{3,4\}\$$
Abs. path_to_file	value	$^\Lambda(/\|([A-Za-z]:))$
Rel. path_to_file	value	$^\Lambda(?!/\|([A-Za-z]:))[^\Lambda/]+/$
Rel. path_to_file	name	$(dir\|path\|fold)$
	value	$^\Lambda[^\Lambda/\backslash.]+\$$

To test whether a suspicious download URL is vulnerable, we utilize the URL to download a representative sensitive file. If the suspicious URL can successfully download the representative file, we confirm it as a vulnerability. Otherwise, it is not.

In our experiment, we group file download vulnerabilities into three categories according to their capabilities of downloading different sensitive files. If a vulnerability can leak the source code of server-side scripts, we call it *s-vul*. If a vulnerability can be used to download web configuration files, we call it *w-vul*. And if a vulnerability can retrieve critical system files, we

call it *c-vul*. Table 6.2 presents the details of our definitions. Apparently, a vulnerability can simultaneously belong to more than one category.

TABLE 6.2: The definition of three types of vulnerabilities.

Type	s-vul	w-vul	c-vul
Sensitive Files	Source Codes	Web Configurations	System Files

To detect s-vul, we choose the download script itself as a representative server-side script (i.e., `download.ext`). We note that a successful retrieval of `download.ext` does not guarantee a s-vul has the capability of downloading the entire set of server-side script source codes in the website. However, we choose it as representative because it shows s-vul's minimal capability.

To perform a valid test, we have to replace recognized file name with the representative script name. We also need to prepare an appropriate path to replace the `path_to_file` because the location of our representative file is usually different from the location of original file. We enumerate possibly appropriate paths to find the path that can lead to the exploit (we call it `path_to_exploit`). Our enumeration is based on the `path_to_file` and the representative script's leading path (i.e., `path_to_script`).

Let s be the number of directories within `path_to_script` and let f be the number of directories within `path_to_file`. Since absolute `path_to_file` is started with the top directory but relative `path_to_file` implicitly means the current path (i.e., `path_to_script`) or some other paths specified in server-side script is leading to this `path_to_file`, our path enumeration is different for absolute `path_to_file` and relative `path_to_file`. For absolute `path_to_file`, we use path pattern like $path\_to\_file/\{../\}^{f+k}path\_to\_script$, where $\{../\}^{f+k}$ denotes $f+k$ consecutive $../$. While for relative `path_to_file`, we use pattern like $path\_to\_file/\{../\}^{s+f+k}path\_to\_script$, where $\{../\}^{s+f+k}$ indicates $s+f+k$ consecutive $../$. In our experiment, we enumerate k from 0 to 8. Although these patterns can detect a large portion of s-vul, they heavily rely on the parent path operator (i.e., $../$). As a result, these patterns cannot disclose the vulnerable script that has sanitized the parent path operator.

To detect s-vul in the scripts that have stripped $../$, we enumerate additionally possible paths. For absolute `path_to_file`, the additional path pattern is $\{path\_to\_file\}^{-m}/path\_to\_script$, where the $\{path\_to\_file\}^{-m}$ indicates a `path_to_file` in which the last m directories have been stripped. Apparently, $0 \leq m \leq f$. While for the relative `path_to_file`, we only consider two additional paths. One is $./$ and the other is an empty path.

To have a more comprehensive study of file download vulnerabilities in the web, we try further exploits to detect w-vul and c-vul. For detecting w-vul, we choose different representative web configuration files for different server-side scripting languages. We choose `WEB-INF/web.xml` in JSP and `web.config` in ASP.NET. Since developers write web configuration files for ASP

and PHP in an ad hoc manner, we use `index.(asp|php)` or `default.(asp|php)` as the representative. These configurations are usually stored in web server home directory. To test whether a suspicious URL can download web configuration, we use representative web configuration file to replace the original file name in corresponding URL parameters. We try similar path enumerations as we choose for testing s-vul but remove `path_to_script` in these paths, because web configurations are usually stored in website home directory. If we can successfully download the representative configuration, we regard this URL as a w-vul.

For discovering c-vul, we choose `win.ini` as the representative in Windows and `/etc/passwd` (or `/etc/shadow`) in Linux. To discover c-vul, we first use `win.ini` or `passwd` to replace the original file name in the URL parameters. We also enumerate similar paths as we use for the discovery of s-vul, but replace `path_to_script` with `sys_path`. The `sys_path` could be `/windows/` or `C:/windows/` in Windows system and `/etc/` in Linux.

6.2.2 Empirical Study of File Download Vulnerabilities in the Web

In our experiments, we have successfully discovered 6.060 vulnerabilities from the $19,137$ suspicious URLs (i.e., 31.7% URLs are vulnerable in our dataset). Among these vulnerabilities, $5,807$ are s-vul, $4,285$ are w-vul and $1,987$ are c-vul.

It can be seen, a number of vulnerabilities can be used to only download source codes or web configurations or critical system files or some of them. There are two major reasons causing this result. First, many vulnerabilities exist due to insufficient sanitization. Compared with the totally missing of sanitization, insufficient checks could limit the consequences of vulnerabilities. For example, if a server-side script only sanitizes parent path operators, the resulted vulnerability can still download sensitive files in current directory or sub directories. Moreover, if a script applies an incomplete black list to only sanitize script's extension names, the corresponding vulnerability may be able to still retrieve critical system files. Second, server-side script engines have their own mechanisms to protect filesystem although these protections are usually disabled in default. We take PHP as an example to explain this kind of protection. Website administrators who take care of filesystem security can enable the `safe mode` and specify an `open_basedir` string in `php.ini`[1]. Since the PHP safe mode only allows an PHP script to read the file which has the same UID (owner) as this script, this mode is very effective in preventing vulnerable scripts from reading world-readable critical files. Moreover, the open_basedir string can limit the files that PHP scripts can read to a specified directory-tree.

We conduct empirical study of file download vulnerabilities based on our experiment results. Our study covers three aspects. We first analyze our discovered vulnerabilities according to different scripting languages. We then illustrate the geographical distribution of vulnerabilities

[1] http://www.php.net/manual/en/ini.core.php

for different countries. At last, we study the popularity of the websites where the vulnerabilities are found according to the Alexa top 1 million site ranking list [140].

6.2.2.1 Scripting Language Study

Our experiments cover four popular scripting languages (i.e., PHP, JSP, ASP.NET and ASP). Figure 6.2 illustrates the distribution of the three types of vulnerabilities (including s-vul, w-vul and c-vul) among the four scripting languages. We also plot the number of suspicious URLs for each scripting language in this figure. PHP contains the largest number of s-vul, w-vul and c-vul, while JSP is involved with fewest s-vul and w-vul. This result is probably due to the biased samples collected by our crawler which collects much more suspicious PHP URLs than JSP URLs. Moreover, it collects roughly the same number of suspicious URLs for ASP and ASP.NET. Since ASP is the predecessor version of ASP.NET, it is not surprising to see more vulnerabilities in ASP than those in ASP.NET.

An interesting observation is that, although JSP has fewest s-vul and w-vul, its c-vul are more than those in ASP and ASP.NET. The reason is relevant to the type of operation systems the scripting languages run. In our experiment result, we only discover 285 `win.ini` while harvest 1,702 `/etc/passwd`. This result indicates that the stolen of critical system file in Windows system is more difficult than that in Linux system. Since JSP is a platform-independent scripting language but ASP and ASP.NET can only work in Windows system, it is reasonable for JSP to have more c-vul even if its s-vul and w-vul are less than those in ASP and ASP.NET.

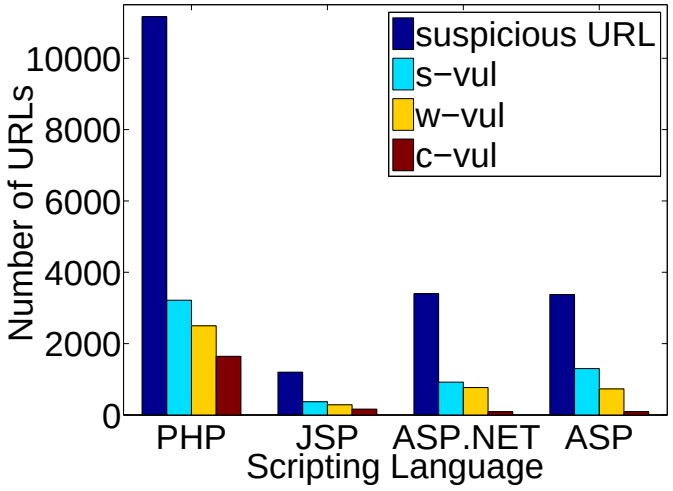


FIGURE 6.2: The distribution of file download vulnerabilities for scripting languages.

6.2.2.2 Global Distribution Study

In this section, we study global distribution of file download vulnerabilities among different countries. We recognize the country code of a suspicious URL according to the top domain of this URL (e.g., .kr, .cn, .it etc.). If the top domain name is not a country code (such kind of top domain could be .edu, .org, .com etc.), we query such URLs' country code by means of `whois` services. If even the `whois` service can neither recognize a URL's country code, we regard this URL as from an *unknown* country. We finally have 793 out of the 19,137 suspicious URLs from *unknown* country.

In our experiment results, we have successfully collected suspicious URLs from 190 countries. We have discovered s-vul, w-vul and c-vul from 133, 119 and 101 countries respectively. We sort these countries according to the number of suspicious URLs within each country in descending order. Note that we simply exclude the *unknown* country from our ranking.

Table 6.3 lists the number of suspicious URLs and the three types of vulnerabilities that belong to the top 15 countries and other countries. Although the top 15 countries are less than 10% of all the 190 countries, they contain more than a half of suspicious URLs. The file download vulnerabilities in the top 15 countries are also majority.

TABLE 6.3: The proportion of suspicious URLs and vulnerabilities for top 15 countries.

	Total	Top 15 countries	Other countries
suspicious URL	19,137	10,758 (56.2%)	8,379 (43.8%)
s-vul	5,807	3,491 (60.1%)	2,316 (39.9%)
w-vul	4,285	2,532 (59.1%)	1,753 (40.9%)
c-vul	1,987	1,112 (56.0%)	875 (44.0%)

We perform an in-depth study for the three types of vulnerabilities in the top 15 countries, and aim to understand the distribution of vulnerabilities among these countries in details. Figure 6.3 demonstrates this distribution. As clearly shown, Korea (kr) is the top country that contains the most suspicious URLs. America (us) takes the second place and Mainland China (cn) is behind. The order of the three types of vulnerabilities is slightly different from the order of suspicious URLs. For example, even the number of suspicious URLs from German (de) is the fourth largest, its vulnerabilities are less than those in Italy (it) which is ranked as the sixth place in terms of the number of suspicious URLs.

6.2.2.3 Popularity Study

In this section, we study the popularity of file download vulnerabilities according to the ranking of vulnerable websites in the Alexa top 1 million domain list [140]. We plot three CDF curves to

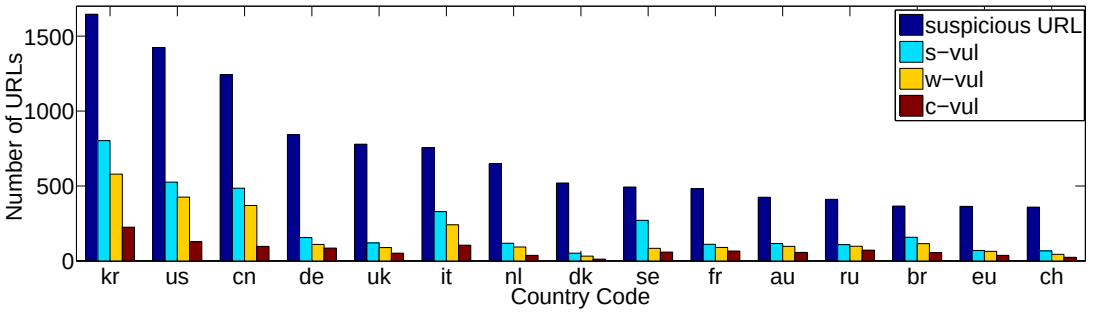


FIGURE 6.3: The distribution of file download vulnerabilities for top 15 countries.

show the distribution of our discovered s-vul, w-vul and c-vul among the top 1 million domains in Figure 6.4. It can be seen, for all the three types of vulnerabilities, around 15% of them belong to the top 1 million domains, and their distributions are very similar. Moreover, there are 23 s-vul, 13 w-vul and 2 c-vul belonging to the top 1000 domains. Our study confirms the existence of file download vulnerabilities in top domains, and some are among the very popular sites.

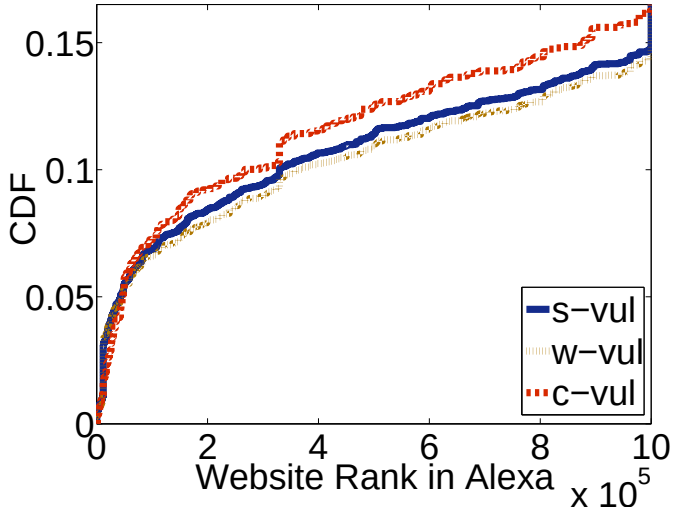


FIGURE 6.4: The popularity of file download vulnerabilities.

6.2.3 Attacks Using File Download Vulnerabilities

In this section, we discuss potential attacks that exploit the file download vulnerability. Except the system intrusion which has been studied by [141], other attacks we describe here have not been comprehensively discussed in prior academic research.

6.2.3.1 System Intrusion

If the system password files (e.g., `/etc/shadow` or `passwd` in Linux or `C:/windows/system32/config/SAM` in Windows) are leaked, the system runs a very high risk of being compromised. Since the file download vulnerability cannot read SAM file from Windows, we only discuss the use of c-vul to break into Linux in our study. In Linux systems, the leak of `/etc/shadow` and the leak of `/etc/passwd` show different levels of security risks. If the `/etc/shadow` is exposed, attacker can employ password breaking tools such as the John the Ripper password cracker[2] to crack system passwords offline. However, if attackers can only have `/etc/passwd` through the c-vul, they must guess and try passwords online.

Beside system password files, a successful intrusion also requires the targeted systems to have at least one system remote login service opening to the Internet. For the sake of security, website administrators usually close this kind of services in their websites or restrict these services to the Intranet. To investigate which websites are likely to be compromised, we employ `nmap`[3] to scan the default ports of three popular remote login services for the websites that have c-vul. The three services are SSH (default port is 22), Telnet (default port is 23) and VNC remote controller (default port is 5900). We put our scanning results in Table 6.4. It can be seen, more than 30% c-vul have chances to result a successful system intrusion. We acknowledge that our results could overestimate the possibility of system intrusion because websites could be deployed behind a NAT and share the same public IP address with many other Intranet machines.

TABLE 6.4: The websites containing c-vul that open default ports of SSH, Telnet and VNC in the Internet.

Critical File	# of c-vul	SSH (22)	Telnet (23)	VNC (5900)
`/etc/passwd`	1,702	746 (43.8%)	34 (2.00%)	23 (1.35%)
`/etc/shadow`	96	31 (32.3%)	1 (1.04%)	1 (1.04%)

6.2.3.2 Database Intrusion

Compared with system intrusion, the use of w-vul to break into back-end database is more feasible because the database credentials (including the account name and password) are usually stored using plain texts in web configuration files. The password cracking is not required for this kind of intrusion. However, to perform a successful database intrusion, attackers must have a connection to administrative management services of back-end database in the Internet. For this reason, we scan official database administration services for the websites which suffer w-vul. We consider three popular database softwares. They are MySql (default port of administration

[2] http://www.openwall.com/john/
[3] http://nmap.org/

service is 3306), Microsoft SQL server (default port is 1433) and Oracle database (we only consider the default port 1521 in Oracle. We note our results underestimate the Oracle intrusion possibility because Oracle could have many other default ports for administrative management).

Table 6.5 presents our scanning results. In average, there are more than 25% websites suffering from w-vul having at least one open database administrative management service in the Internet. In particular, PHP and JSP suffer more public MySql services, while ASP.NET and ASP are more common to be involved with public Microsoft SQL server services.

TABLE 6.5: The websites containing w-vul that open default ports of MySql, MSSql and Oracle in the Internet.

Script	# of w-vul	MySql (3306)	MSSql (1433)	Oracle (1521)
PHP	2,500	888 (35.5%)	38 (1.52%)	23 (0.92%)
JSP	286	35 (12.2%)	13 (4.55%)	19 (6.64%)
ASP.NET	769	72 (9.36%)	96 (12.5%)	4 (0.52%)
ASP	730	84 (11.5%)	111 (15.2%)	17 (2.32%)
Total	4,285	1,079 (25.2%)	258 (6.02%)	63 (1.47%)

Moreover, the management of back-end databases can also be achieved through the web. For example, the phpMyAdmin[4] is a widely used administration tool for managing MySql over the web. It provides direct access to back-end MySql database using HTTP interfaces. As a result, even if databases' official administrative management services are not open to the Internet, attackers still have chances to access the back-end database in case the web-based management services are enabled. We investigate the 2,500 PHP websites which have w-vul and successfully discover 81 (3.24%) sites running phpMyAdmin.

6.2.3.3 White-box Analysis

In contrast to the system and database intrusion, s-vul have a basic capability of downloading server-side script source codes. This capability enables white-box analysis of victim websites. Although the vulnerability is not indispensable for performing white-box analysis on open source websites, it is necessary for analyzing commercial websites. Compared with black-box analysis, the white-box analysis can cover all possible execution paths and hence have higher probabilities to reveal a complete set of web vulnerabilities in the targeted site. As a result, the difficulty of attacking commercial websites is largely reduced.

To demonstrate the effectiveness of white-box analysis in attacking commercial sites, we exploit the s-vul to download server-side script source codes from a vulnerable website. This site deploys commercial PHP applications and belongs to the top 1000 Alexa site ranking list. We illegitimately download 13 important server-side scripts using the s-vul. These scripts include

[4] http://www.phpmyadmin.net/

admin pages, login pages etc.. To analyze these PHP source codes, we employ RIPS [2], an open source static detection tool that can analyze PHP source codes for finding vulnerabilities. We have successfully discover 17 severe vulnerabilities (among them, 10 are SQL injection while the other 7 are cross-site scripting) by running white-box analysis of the 13 important PHP scripts. We put the analysis results in Figure 6.5. Moreover, we believe a careful human analysis of these source codes can yield more sophisticated vulnerabilities.

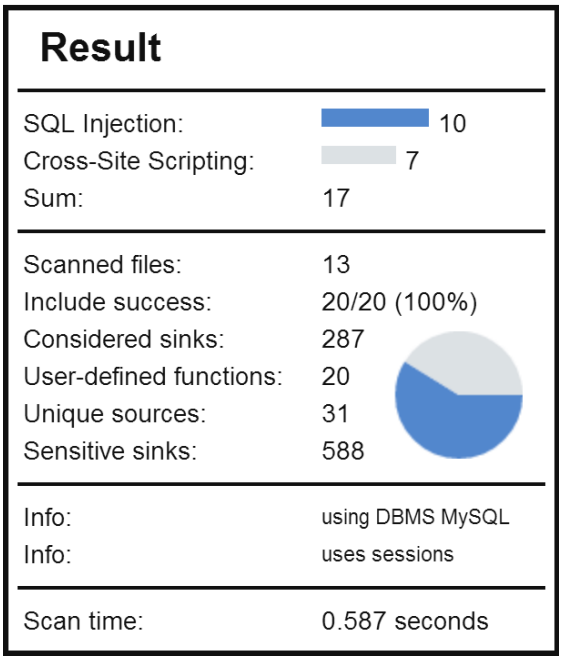


FIGURE 6.5: White-box analysis results of PHP scripts using RIPS-0.54 [2]. We download these PHP source codes by exploiting the s-vul in a vulnerable website.

6.2.3.4 Other Attacks

The use of file download vulnerabilities can also induce many other attacks, such as FTP intrusion, SMTP abusing, known vulnerabilities discovery and the breaking of file download authentication. We will just briefly describe some of these exploits in this chapter and we believe our discussion only covers a sub set of potential attacks.

There are two kinds of FTP intrusion that could be induced by the file download vulnerabilities. One is the intrusion of FTP server which is deployed in the victim website, while the other is the intrusion of third-party FTP servers that the victim website is configured to access automatically. For the first case, the FTP intrusion is similar as system intrusions we have discussed in Section 6.2.3.1. Attackers can exploit `/etc/passwd` or `/etc/shadow` or FTP configuration file to perform this intrusion. But for the second case, attackers should use file download vulnerabilities to download the files which contain the preset third-party FTP credentials. This information is likely written in server-side scripts. For example, PHP provides a function *ftp_login()* to log

into third-party FTP servers. The second parameter of this function is the username of the FTP server and the third parameter is the password. As a result, locating the function *ftp_login()* and tracking its parameters are very useful in revealing third-party FTP credentials.

Some websites also have the functionality of sending e-mails through SMTP services automatically. This functionality cause the possibility of SMTP credentials leakage through the file download vulnerabilities. Using SMTP credentials, attackers can abuse SMTP services to send faked e-mails (such as broadcasting spam).

The use of file download vulnerabilities to discover vulnerable softwares is also attractive, because these vulnerable installs usually have known vulnerabilities. To detect vulnerable softwares, attackers can enumerate the latest vulnerable softwares and test to download some representative files (e.g., readme files, version files, configuration files etc.) of these softwares from `C:/Program Files` in Windows or `/usr/local/` in Linux. If the download is successful, attackers can search known vulnerabilities of this software from Internet. By exploiting the known vulnerabilities, attackers have chances to break into the system and escalate privileges.

More importantly, the use of vulnerable file download scripts to download web files that are managed by other web applications, even if these applications are well protected and only allow authenticated users to download. Attackers can exploit vulnerable scripts running in the same web server to download any web files without authentication. This kind of exploit cannot be prevented by any of existing defenses (such as user input sanitization and directory based access control), and therefore motivate us to design new defence to address this problem.

6.3 FileGuard Design

To address the challenges in defending against vulnerable file download scripts, we propose a new system, called FileGuard, which implements a fine-grained permission control in script engine layer. Compared with ad hoc user input sanitization, FileGuard is more resilient to erroneous implementations since it resides in script engine and does not depend on specific file download logics. Moreover, FileGuard controls permission on file basis which is finer-grained than existing directory based control methods.

In this section, we elaborate on the design of FileGuard. In particular, we first describe the threat model, basic assumption and design goals in Section 6.3.1. We then present the detailed design of FileGuard in Section 6.3.2. After that, we demonstrate the feasibility and effectiveness of our design using a prototype implementation in Section 6.3.3. This prototype of FileGuard is implemented by modifying the source code of PHP5, which is one of the most popular server-side script engine in today's web.

6.3.1 Preliminaries

In the following sub sections, we present preliminaries of FileGuard which include the threat model (Section 6.3.1.1), basic assumption (Section 6.3.1.2) and its design goals (Section 6.3.1.3).

6.3.1.1 Threat Model

We design FileGuard by considering a practical threat model where the operating system of web server has not been compromised. Attackers cannot create, delete or modify any files in the targeted web server. Instead, they behave as malicious web users who can only exploit vulnerable file download script to steal sensitive files. We assume this kind of attackers have two key capabilities. One is the capability of finding vulnerable file download script in targeted web servers, and the other is the capability of locating where the sensitive files they are of interest.

To discover vulnerable file download scripts, attackers can employ a crawler to scan the targeted web servers and test whether suspicious file download scripts are vulnerable, which is similar as the technique we have used for our empirical study (see Section 6.2.1). This technique is practical although it requires human efforts in some cases.

To successfully download sensitive files, attackers should identify where these files are located at first. However, the difficulty of such localization depends on which type of sensitive files they are of interest. For example, the critical system files always have fixed location in particular operating systems (such as `/etc/passwd` and `/etc/shadow` in Linux), while the web configuration files are also fixed in particular web servers and server-side script engine (such as web.config in ASP.NET framework running in IIS). Moreover, the location of any server-side scripts can also be detected by crawling the web sites, no matter they belong to which web applications. Based on these observations, we simply assume the attacker are capable to locate the path of any sensitive files they try to download in this paper.

6.3.1.2 Basic Assumption

We have a basic assumption for the design of FileGuard: all the web managing files are uploaded, modified and downloaded only through server-side scripts. However, we note that many web administrators could manage web files through other methods, such as FTP or SSH or any other remote login systems. To cope with this requirement, we develop a specific tool to enable any web files to be compatible with FileGuard, even if they are out of control by server-side scripts. We will elaborate the detailed techniques in Section 6.3.2.

6.3.1.3 Design Goals

To effectively limit file download vulnerabilities and significantly reduce the effects from human errors, we list three goals in the design of FileGuard as follows.

1. The separation of web files from the rest of local filesystem.
2. The separation of web files belonging to one web application from any other applications.
3. No side effects introduced to the web managing flies, hence not rendering them unable to be processed as usual.

The first goal is to protect local files from being downloaded illegally. These local files contain critical system files, web configuration files and server-side scripts themselves. The leakage of these local files could cause disastrous consequences to the web server (see Section 6.2.3). The second goal is to enable web applications to protect their web files from leaking through vulnerable scripts belonging to other web applications which runs on the same web server. Before FileGuard, this goal cannot be achieved by any existing defending methods. The last goal is to prevent FileGuard from damaging the web files. For example, FileGuard should not be able to change the contents or types of any web files because this kind of modification could render web files from being reading and managing by their traditional editing program.

6.3.2 FileGuard Design

We design FileGuard to prevent file download vulnerabilities by implementing identity based access control in script engine layer. FileGuard consists of two components. One is the identity insertion function. This function can add specific identity to web files when they are uploaded into local filesystem. The other is the identity based access control mechanism. This mechanism can determine whether they are allowed to be read and downloaded through script engine based on the identities of web files.

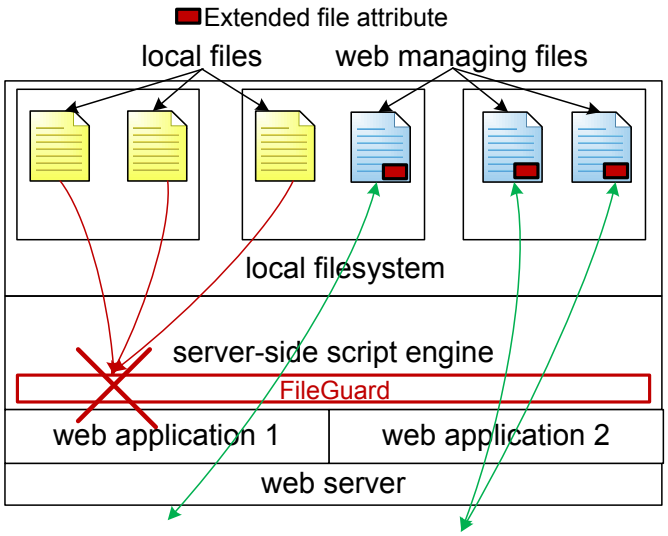
To implement FileGuard, a key challenge is how to embed additional identities into web files by script engine. As required by the third design goal (see 6.3.1.3), we cannot insert identities by modifying the contents of web files. Although we could use dedicated string pattern to embed additional identities into the names of web files, we do not choose this mean because it could limit the usability of web files. That is, if we do so, web administrators and web developers lose the opportunity to name the web files as they want. In this thesis, we use extended file attributes as the medium to store the additional identities for FileGuard. We note that there should exist other mediums for this purpose, although we choose extended file attributes here to demonstrate the feasibility of FileGuard design.

Extended file attributes is a file system feature that is supported by almost filesystems in mainstream operating systems (such as the UFS1 and UFS2 filesystems in FreeBSD, the ext2, ext3, ext4, JFS, ReiserFS, XFS, Btrfs and OCFS2 filesystems in Linux, HFS+ filesystem in MAC OS as well as FAT and HTFS filesystems in Windows). This attribute can associate computer files with meta information that are not interpreted by the filesystem, hence making nearly no side effects to the web files. FileGuard contains a modified version of file upload functions in script engine, which embed dedicated extended file attributes to any files uploaded through script engine. Since local files in the filesystem do not have this kind of attributes by default, FileGuard can easily differentiate which files are local and which are not by checking the extended file attributes when the file read function is called in script engine. Figure 6.6(a) demonstrates how FileGuard protects the local files from being downloaded through vulnerable scripts in the script engine layer. As can be seen, FileGuard can embed an extended file attribute into each web managing file during its upload process, and then allow the download of these files by checking corresponding attributes. In contrast, since the local files do not contain such kind of attributes, FileGuard can block the download of these local files as well.

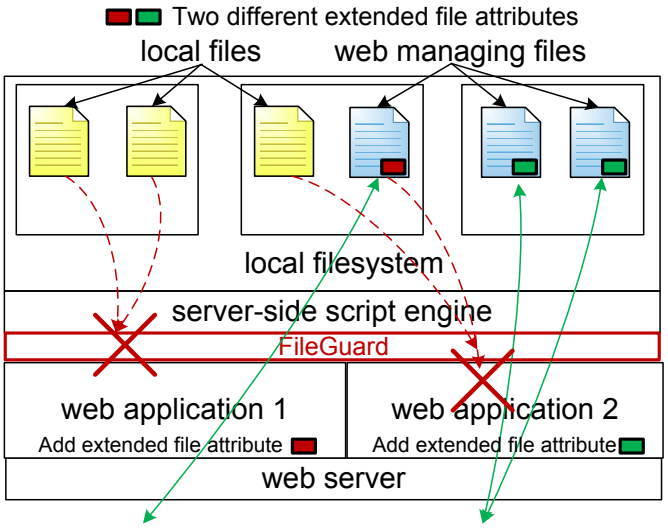
To protect web files from being downloaded through the vulnerabilities belonging to any other web applications, FileGuard also allows web developers to keep their own identity as a secret and set it to the extended file attributes in their own web applications. By this way, the file download scripts can be used to only download the web files which contain the attributes the download scripts have already known. The vulnerable scripts belonging to other web applications cannot be used to download the web files associated with secret attributes, because they cannot prove to FileGuard that they know the secret. As can be seen in Figure 6.6(b), the web files uploaded through web application 1 contain different attributes compared with those files uploaded via web application 2. In this case, if the file download script in web application 2 is vulnerable, it cannot be used to download the web files managed by web application 1 because this vulnerable script does not know web application 1's secret attribute.

To sum up, FileGuard has three distinct advantages compared with existing methods:

- Compared with ad hoc user input mechanism, FileGuard minimizes human efforts in application level development, hence mitigating the possibility of vulnerable implementation of file download logics.
- Compared with directory based permission control, FileGuard performs a finer-grained permission control on file basis.
- Compared with all the existing methods, FileGuard is the first solution that can prevent illegal downloads of web files through the vulnerabilities in other web applications.



(a) FileGuard to protect local files.



(b) FileGuard to protect the web files against vulnerable scripts residing in other web applications.

FIGURE 6.6: FileGuard to protect local files and web files against vulnerable file download scripts.

6.3.3 Prototype Implementation

To demonstrate the feasibility and effectiveness, we implement a prototype of FileGuard here by modifying the source code of PHP5 (version 5.5.6) script engine in Linux. Before the description of the implementation details, we present the background of file upload and download process handled by PHP5 at first. This background description helps us make a deep understanding of how file upload and download is implemented in PHP5.

PHP5 is designed to follow the RFC1867 [142] to handle form-based file upload process. Upon web server receiving remote files from web clients through HTTP POST command, the PHP5 script engine is called to write these files to local filesystem. The PHP5 engine implements

a function SAPI_POST_HANDLER_FUNC() for this task and first store the uploaded files in a temporary file path (default in `/tmp`, but can be changed in php.ini file). The file name is randomly generated using the pattern "php******" (each * represents a random character). To copy the uploaded files to specified directories and using specified names, server-side scripts should explicitly call a PHP function move_uploaded_file(). This function has two parameters, the first is for the temporary file path and the second is for the specified file path. Server-side script can access the glabal array $_FILES to locate the temporary file path. On the side of file download, PHP5 has a number of functions to open a file and read it to the web. In our implementation, we consider only the readfile() function since it is the most popular file read function used by existing server-side scripts.

In Linux, an extended file attribute can be added to a file using a system call [143]. To set an extended file attribute to a file using the name and path of this file, the system call number is 226. While to retrieve an extended file attribute from a file, the call number is 229. And to remove an extended file attribute, the call number is 235. Moreover, Linux has four namespaces for extended file attributes. They are user, trusted, security and system. We choose the user namespace in our implementation since it can be recognized in user space by any Linux system. To this end, we implement the prototype of FileGuard by adding the system call to insert extended file attributes in PHP function move_uploaded_file() and using the system call to read extended file attributes in PHP function readfile(). We also check whether the retrieved extended file attribute is correct, and refuse the read of a file without an expected attribute.

Figure 6.7 illustrates the layout of our FileGuard prototype implementation based on PHP5 script engine. It demonstrates how FileGuard handles the file upload process and performs access control to prevent illegal file download. Each file uploaded through FileGuard could be associated with an extended file attribute as its identity, and FileGuard can check the extended file attribute when the file is downloaded. Only the files which have legal attribute can be allowed to be downloaded. FileGuard will remove the extended file attribute from the copy of the file which is downloaded.

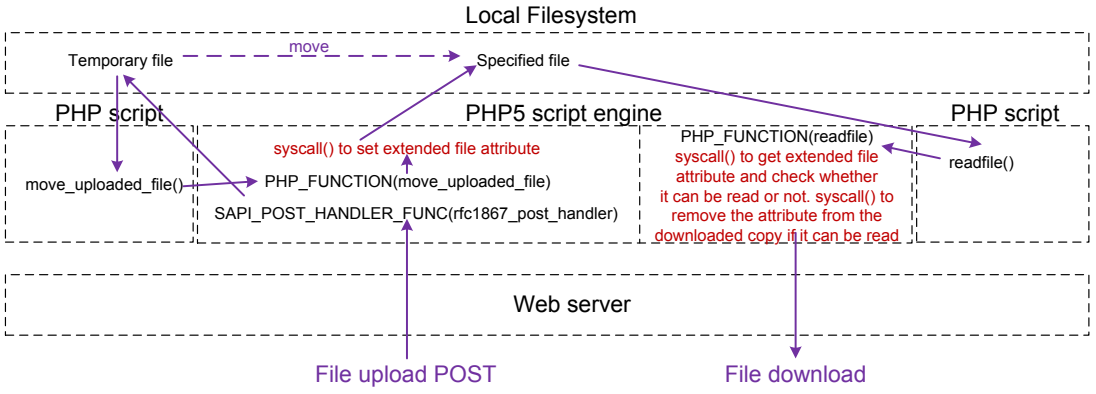
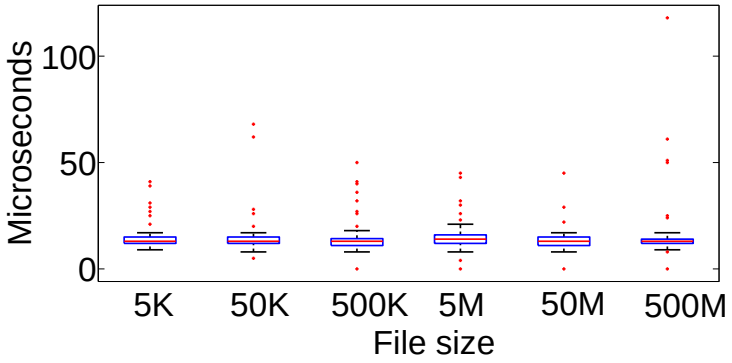


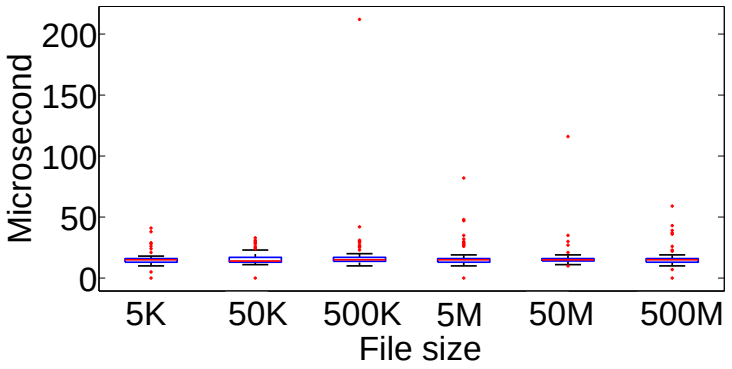
FIGURE 6.7: The layout of FileGuard prototype implementation.

6.4 Performance Evaluation

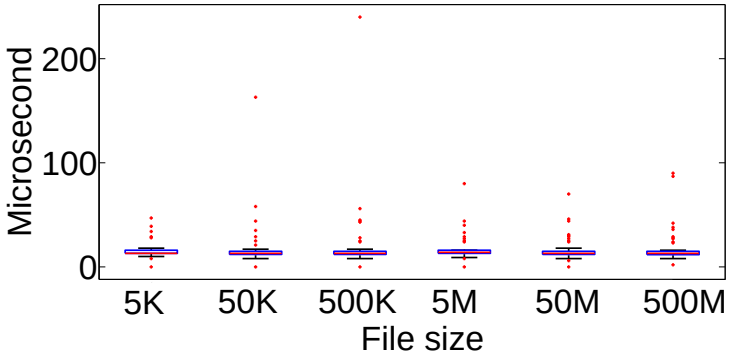
In this section, we evaluate the performance overhead introduced by FileGuard. In particular, FileGuard consumes more time for file upload and download since it needs to set, get or remove extended file attributes to the file managed by script engine. As a result, we investigate the timing cost required by the system calls for setting, getting and removing an extended file attribute in Linux. We test these system calls for the files in different size (from 5K to 500M), and run the system call for each file 100 times. We present the experimental results in Figure 6.8.



(a) The time cost of setting an extended file attribute.



(b) The time cost of getting an extended file attribute.



(c) The time cost of removing an extended file attribute.

FIGURE 6.8: The performance overhead introduced by FileGuard in terms of time cost (in microseconds).

As can be seen, the time cost is relatively small (around 20 microseconds in average) in setting, getting and even removing the extended file attributes. Moreover, this timing cost does not depend on the size of files. The 5K file has the similar timing cost compared with the 500M file in all the Figures 6.8(a), 6.8(b) and 6.8(c). Based on this experimental result, we can confirm that FileGuard can induce only negligible performance overhead for defending against file download vulnerabilities.

Chapter 7

Conclusion and Future Research

7.1 Conclusions

In this thesis, we have conducted an in-depth research to investigate the fundamental limitations of existing trust-based systems. We mainly focus our research on three trust-based systems that are designed for enhancing anonymity and security protections in the web. We have found that the use of trust for these systems does not meet the best practice in the state-of-the-art. Instead, trust usually introduces new problems in protecting anonymity and security, as well as does not achieve the best usage. Based on these findings, we propose more trustworthy solutions to further enhance the three trust-based systems we have studied in this thesis. Our research indicates that the use of trust for anonymous communications and web services stays in a very early stage, much more research efforts are required in this field.

For trust-based onion routing, we reveal two fundamental limitations for the use of trust in protecting anonymity for the first time. One is the biased trust distributions which could reduce the overall entropy of onion routing systems. While the other is the incorrect trust assignments that may render trust-based onion routing getting worse anonymity protection. To overcome these two limitations, we propose trust degree based and trust graph based solutions. The former solution enables users to hide using the routers trusted by more other users, hence increasing the overall entropy of the system even it cannot reduce the bias of trust distributions. In contrast, the latter solution proposes the use of global trust to alleviate the levels of biased trust distributions. Moreover, trust graph based onion routing employs group trust to address incorrect trust assignment problem. These two solutions are proved effective in enhancing anonymity protections through both mathematical proofs and experiments with real-world social data sets.

For the flawed certification based trust model in public key infrastructure, we have designed an optimal method to maximize the protection against SSL/TLS man-in-the-middle attacks. Our

method exhausts available trust anchors and exploit Internet diversity as much as possible, hence getting nearly the largest protection the trust model can provide. Our research confirms that: a priori trust within certification based trust model can be fully utilized to maximize the protection against SSL/TLS man-in-the-middle attacks.

For file download vulnerabilities, we perform a large-scale survey in the Internet and confirm the weak protection of this vulnerability in today's web. This bad result motivates us to design a new file download protection system, called FileGuard. FileGuard is implemented in script engine layer in order to mitigate the possibility of human errors, and can provide a finer-grained permission control on file basis. We implement a prototype of FileGuard by modifying the source code of PHP5, and confirm its negligible performance overhead.

Based on our research, we reveal a lot of problematic use of trust for securing anonymous communications and web services, hence shedding light that there still exists much room for further improvement in even the state-of-the-art trust-based systems. Our more trustworthy solutions are designed based on these findings, and have demonstrated better performance in protecting anonymity and security.

7.2 Future Directions

We have presented a comprehensive study of three popular trust-based systems in this thesis, and reveal that the use of trust to protect anonymity and security is a conundrum. On the one hand, trust is very effective in defeating a lot of traditional attacks in the Internet. But on the other hand, the use of trust always induces new attacks which target to the trust itself. Although our research leads to more trustworthy solutions to enhance the three selected trust-based systems, we cannot guarantee that our solutions reach the best results. Moreover, our studies do not cover the improvement of other trust-based systems. These two could be our future research directions.

First, we can do further research on the three trust-based systems we have studied in this thesis. For example, our proposed trust graph based onion routing still suffers from social engineering attacks. We thus call for new ideas to address this problem in the future. Moreover, although our active approach is designed to maximize the protection against SSL/TLS man-in-the-middle attacks, it requires the nearly re-design of existing SSL/TLS protocol internals. This requirement makes our approach very difficult to be deployed in the Internet quickly. For this reason, the studies on lightweight solutions are very attractive in the future research. Besides, the FileGuard stays in a very early design stage. We will extend the basic idea of FieGuard and design a more secure system which implements mandatory access control in script engine layer.

Second, we can extend our research area to other trust-based systems in our future studies. For example, we can do an in-depth analysis to discover fundamental problems in trust-based Sybil defenses [39–44]. These systems employ a priori trust from social networks to resist Sybil attack [45] in peer-to-peer networks [46]. However, they are usually designed based on the assumption that attackers are difficult to compromise honest people's trust and make friends with them. This assumption is too strong in many practical scenarios. Therefore, we will consider how to remove this assumption in our future work. Moreover, we will study how to build up a reliable trust infrastructure to enhance the whole Internet. This infrastructure is essentially required by the design of named data networking [144], which is a new architecture standard in the future Internet.

Appendix A

A.1 The proof of Theorem 1

Proof. We first consider a simple optimization problem as follows.

$$\min\ (x \cdot \frac{x}{x+a} + y \cdot \frac{y}{y+b}),\ s.t.\ x+y=\beta.$$

Where x and y are variables while a and b are constants. Let $f(x) = x \cdot x/(x+a) + (\beta - x) \cdot (\beta - x)/(\beta - x + b)$. Then the above optimization problem can be simplified as:

$$\min f(x) = \min(x \cdot \frac{x}{x+a} + (\beta - x) \cdot \frac{\beta - x}{\beta - x + b}).$$

It is known that, if $f(x)$ has an extreme value and $f(x)$'s second derivative is larger than 0, this extreme value is $f(x)$'s minimum value. And this minimum value can be obtained by letting $f(x)$'s first derivative equals to 0. Such that, if $f''(x) = d^2f(x)/d^2x > 0$, $f(x)$ can reach its minimum value when $f'(x) = df(x)/dx = 0$.

When $\beta = x + y > x$, $a > 0$ and $b > 0$, we can prove that $f''(x) > 0$. In this case, we find the minimum value of $f(x)$ by solving the quadratic equation $f'(x) = 0$.

We have two roots of $f'(x) = 0$. One is a positive value and the other is a negative value. We only consider the positive result, $x = \beta \cdot a/(a+b)$. Applying this x to $f(x)$, we have:

$$\min f(x) = \min\ (x \cdot \frac{x}{x+a} + (\beta - x) \cdot \frac{\beta - x}{\beta - x + b}) = \frac{\beta^2}{a+b+\beta}.$$

Since $x + y = \beta$, we have:

$$\frac{x^2}{x+a} + \frac{y^2}{y+b} \geq \frac{\beta^2}{a+b+\beta} = \frac{(x+y)^2}{a+b+x+y}. \tag{A.1}$$

To satisfy the equality, we have $x = \beta \cdot a/(a+b)$ and $y = \beta \cdot b/(a+b)$ (i.e., $x \propto a$ and $y \propto b$).

Now, we turn back to the optimization problem described in Eq. (3.5). By iteratively applying the inequality (A.1) to $E(Y[u_i|C_\ell])$, we have:

$$\begin{aligned} E(Y[u_i|C_\ell]) &= \sum_{C_\ell \in R_e} Pr[C_\ell|u_i] \cdot Y[u_i|C_\ell] \\ &= \sum_{C_\ell \in R_e} Pr[C_\ell|u_i] \cdot \frac{Pr[C_\ell|u_i]}{Pr[C_\ell|u_i] + \sum\limits_{u \in U \setminus u_i} Pr[C_\ell|u]} \\ &\geq \frac{(\sum\limits_{C_\ell \in R_e} Pr[C_\ell|u_i])^2}{\sum\limits_{C_\ell \in R_e} Pr[C_\ell|u_i] + \sum\limits_{C_\ell \in R_e} \sum\limits_{u \in U \setminus u_i} Pr[C_\ell|u]} \\ &= \frac{\theta_{\ell e}^2}{\theta_{\ell e} + \sum\limits_{C_\ell \in R_e} \sum\limits_{u \in U \setminus u_i} Pr[C_\ell|u]}. \end{aligned} \tag{A.2}$$

To satisfy all the equalities and achieve the minimal value, we have $\forall C_\ell \in R_e,\ Pr[C_\ell|u_i] \propto \sum_{u \in U \setminus u_i} Pr[C_\ell|u]$ subject to $\theta_{\ell e} = \sum_{C_\ell \in R_e} Pr[C_\ell|u_i]$.

Therefore, Theorem 4.1 is proved. □

A.2 The proof of Theorem 2

Proof. We first consider a function as follows.

$$\sum_{C_{(1)} \in R_e} \frac{\prod\limits_{n=1}^{|O|} Pr[C_{(n)}|u_i, C_{(n+)}]^2}{\prod\limits_{n=1}^{|O|} Pr[C_{(n)}|u_i, C_{(n+)}] + \sum\limits_{u \in U \setminus u_i} \prod\limits_{n=1}^{|O|} Pr[C_{(n)}|u, C_{(n+)}]}, \tag{A.3}$$

subject to $\sum\limits_{C_{(1)} \in R_e} Pr[C_{(1)}|u_i, C_{(1+)}] = \theta_{(1)e}$. As the operator $\sum\limits_{C_{(1)} \in R_e}$ is irrelevant to $\prod\limits_{n=2}^{|O|} Pr[C_{(n)}|u_i, C_{(n+)}]$, we can consider it as a constant in the function (A.3). Simply let $\Omega = \prod\limits_{n=2}^{|O|} Pr[C_{(n)}|u_i, C_{(n+)}]$. By applying the inequality (A.2) to the function (A.3), we can prove the minimal value of the function (A.3) as below.

$$\frac{(\theta_{(1)e}\Omega)^2}{\theta_{(1)e}\Omega + \sum\limits_{C_{(1)} \in R_e} \sum\limits_{u \in U \setminus u_i} \prod\limits_{n=1}^{|O|} Pr[C_{(n)}|u, C_{(n+)}]}. \tag{A.4}$$

To achieve this minimum value, for $\forall C_{(1)} \in R_e$, subject to $\sum_{C_{(1)} \in R_e} Pr[C_{(1)}|u_i, C_{(1+)}] = \theta_{(1)e}$, we have:

$$Pr[C_{(1)}|u_i, C_{(1+)}] \propto \sum_{C_{(1-)} \in R_e^{1-1}} \sum_{u \in U \setminus u_i} \prod_{n=1}^{|O|} Pr[C_{(n)}|u, C_{(n+)}]. \tag{A.5}$$

where, $C_{(1-)} = \varnothing$ and $R_e^{1-1} = R_e^0 = \varnothing$.

Now, we turn back to the optimization problem described in Eq. (3.7).

$$E(Y[u_i|C_O]) = \sum_{C_O \in R_e^{|O|}} \frac{Pr[C_O|u_i]^2}{Pr[C_O|u_i] + \sum_{u \in U \setminus u_i} Pr[C_O|u]}.$$

where, $Pr[C_O|u_i] = \prod_{n=1}^{|O|} Pr[C_{(n)}|u_i, C_{(n+)}]$, and the operator $\sum_{C_O \in R_e^{|O|}}$ can be extended to $\sum_{C_O \in R_e^{|O|}} = \sum_{C_{(|O|)} \in R_e} \cdots \sum_{C_{(1)} \in R_e}$.

By considering that each operator $\sum_{C_{(j)} \in R_e}$ is irrelevant to $\prod_{n=j+1}^{|O|} Pr[C_{(n)}|u_i, C_{(n+)}]$, we can iteratively apply the similar optimization process like (A.4) and (A.5) to $E(Y[u_i|C_O])$, and therefore prove:

$$min\ E(Y) = \frac{\prod_{n=1}^{|O|} \theta_{(n)e}^2}{\prod_{n=1}^{|O|} \theta_{(n)e} + \sum_{C_O \in R_e^{|O|}} \sum_{u \in U \setminus u_i} Pr[C_O|u]}.$$

To reach this minimal value, for $\forall C_{(n)} \in R_e$, subject to $\sum_{C_{(n)} \in R_e} Pr[C_{(n)}|u_i, C_{(n+)}] = \theta_{(n)e}$, we have:

$$Pr[C_{(n)}|u_i, C_{(n+)}] \propto \sum_{C_{(n-)} \in R_e^{n-1}} \sum_{u \in U \setminus u_i} \prod_{n=1}^{|O|} Pr[C_{(n)}|u, C_{(n+)}].$$

Therefore, Theorem 3.6 is proved. □

Bibliography

[1] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42. ACM, 2009.

[2] http://rips-scanner.sourceforge.net/, 2013.

[3] Internet world stats. http://www.internetworldstats.com/stats.htm, 2012.

[4] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[5] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[6] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.

[7] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[8] Micheal G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.

[9] Nick Mathewson, Paul Syverson, and Roger Dingledine. Tor: the second-generation onion router. In *Proceedings of the 12th USENIX Security Symposium*, 2004.

[10] Dave Cooper. Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile. 2008.

[11] Krishna Puttaswamy, Alessandra Sala, and Ben Y Zhao. Improving anonymity using social links. In *Proceedings of the 4th Workshop on Secure Network Protocols*, pages 15–20. IEEE, 2008.

[12] Aaron Johnson and Paul Syverson. More anonymous onion routing through trust. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium*, pages 3–12. IEEE, 2009.

[13] George Danezis, Claudia Diaz, Carmela Troncoso, and Ben Laurie. Drac: An architecture for anonymous low-volume communications. In *Proceedings of the 10th Privacy Enhancing Technologies Symposium (PETS)*, pages 202–219. Springer, 2010.

[14] Aaron M Johnson, Paul Syverson, Roger Dingledine, and Nick Mathewson. Trust-based anonymous communication: Adversary models and routing algorithms. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 175–186. ACM, 2011.

[15] Prateek Mittal, Matthew Wright, and Nikita Borisov. Pisces: Anonymous communication using social networks. *preprint arXiv:1208.6326*, 2012.

[16] Nada Kapidzic. Creating security applications based on the global certificate management system. *Computers & Security*, 17(6):507–515, 1998.

[17] Carl M Ellison. The nature of a useable pki. *Computer Networks*, 31(8):823–830, 1999.

[18] Helen Kapodistria, Sarandis Mitropoulos, and Christos Douligeris. An advanced web attack detection and prevention tool. *Information Management & Computer Security*, 19 (5):280–299, 2011.

[19] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114. Springer, 2001.

[20] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 183–195. IEEE, 2005.

[21] Lasse Overlier and Paul Syverson. Locating hidden servers. In *Proceedings of the IEEE Symposium on Security and Privacy*, page 15pp. IEEE, 2006.

[22] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the ACM workshop on Privacy in electronic society*, pages 11–20. ACM, 2007.

[23] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. A new cell counter based attack against tor. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 578–589. ACM, 2009.

[24] Nathan S Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on tor using long paths. In *Proceedings of the 17th USENIX Security Symposium*, pages 33–50, 2009.

[25] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. Correlation-based traffic analysis attacks on anonymity networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(7):954–967, 2010.

[26] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):13, 2010.

[27] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Weijia Jia, and Wei Zhao. Protocol-level attacks against tor. *Computer Networks*, 2012.

[28] Jason Mortensen. Website impersonation attacks: Who is really behind that mask. http://www.concise-courses.com/infosec/website-impersonation-attacks/, 2013.

[29] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.

[30] Stanley RM Oliveira and Osmar R Zaïane. Protecting sensitive knowledge by data sanitization. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pages 613–616. IEEE, 2003.

[31] Davide Balzarotti, Marco Cova, Viktoria Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 387–401. IEEE, 2008.

[32] Jacob Appelbaum. Detecting certificate authority compromises and web browser collusion. https://blog.torproject.org/blog/detecting-certificate-authority-compromises-and-web-browser-collusion, 2011.

[33] Black tulip report of the investigation into the Diginotar Certificate Authority Breach. Technical report, Fox-IT, 2012.

[34] Charles Arthur. Rogue web certificate could have been used to attack iran dissidents. http://www.guardian.co.uk/technology/2011/aug/30/faked-web-certificate-iran-dissidents, 2011.

[35] Dan Wendlandt, David G Andersen, and Adrian Perrig. Perspectives: Improving ssh-style host authentication with multi-path probing. In *Proceedings of the USENIX Annual Technical Conference*, 2008.

[36] Ralph Holz, Thomas Riedmaier, Nils Kammenhuber, and Georg Carle. X.509 forensics: Detecting and localising the SSL/TLS men-in-the-middle. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. 2012.

[37] Italo Dacosta, Mustaque Ahamad, and Patrick Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. 2012.

[38] Michael Dietz, Alexei Czeskis, Dirk Balfanz, and Dan S Wallach. Origin-bound certificates: A fresh approach to strong client authentication for the web. In *Proceedings of the 20th USENIX Security Symposium*, 2012.

[39] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. *ACM SIGCOMM Computer Communication Review*, 36(4):267–278, 2006.

[40] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–17. IEEE, 2008.

[41] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *Proceedings of the 16th Annual Network & Distributed System Security Conference (NDSS)*, 2009.

[42] Dinh Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient online content voting. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, volume 9, pages 15–28, 2009.

[43] Abedelaziz Mohaisen, Nicholas Hopper, and Yongdae Kim. Keep your friends close: Incorporating trust into social network-based sybil defenses. In *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1943–1951. IEEE, 2011.

[44] Haifeng Yu. Sybil defenses via social networks: a tutorial and survey. *ACM SIGACT News*, 42(3):80–101, 2011.

[45] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[46] Geoffrey Fox. Peer-to-peer networks. *Computing in Science & Engineering*, 3(3):75–77, 2001.

[47] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A reputation system for peer-to-peer networks. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 144–152. ACM, 2003.

[48] Michael Kinateder and Kurt Rothermel. Architecture and algorithms for a distributed reputation system. In *Trust Management*, pages 1–16. Springer, 2003.

[49] Yao Wang and Julita Vassileva. Trust and reputation model in peer-to-peer networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, pages 150–157. IEEE, 2003.

[50] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[51] Audun Jsang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th bled electronic commerce conference*, pages 41–55, 2002.

[52] Paul Resnick and Richard Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. *Advances in applied microeconomics*, 11:127–157, 2002.

[53] Sravanthi Kalepu, Shonali Krishnaswamy, and Seng Wai Loke. Verity: a qos metric for selecting web services and providers. In *Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops*, pages 131–139. IEEE, 2003.

[54] Le-Hung Vu, Manfred Hauswirth, and Karl Aberer. Qos-based service selection and ranking with trust and reputation management. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 466–483. Springer, 2005.

[55] Ziqiang Xu, Patrick Martin, Wendy Powley, and Farhana Zulkernine. Reputation-enhanced qos-based web services discovery. In *Proceedings of the IEEE International Conference on Web Services*, pages 249–256. IEEE, 2007.

[56] Chrysanthos Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 150–157. ACM, 2000.

[57] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7): 843–857, 2004.

[58] Tor metrics portal: Users. https://metrics.torproject.org/users.html, 2013.

[59] Ceki Gulcu and Gene Tsudik. Mixing e-mail with babel. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 2–16. IEEE, 1996.

[60] Markus Jakobsson. A practical mix. In *Advances in Cryptology*, pages 448–461. Springer, 1998.

[61] https://www.anonymizer.com/.

[62] https://freenetproject.org/.

[63] http://www.i2p2.de/.

[64] Matthew K Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):489–522, 2004.

[65] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2013.

[66] Zhen Ling, Junzhou Luo, Wei Yu, Ming Yang, and Xinwen Fu. Extensive analysis and large-scale empirical evaluation of tor bridge discovery. In *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM)*, pages 2381–2389. IEEE, 2012.

[67] Zhen Ling, Junzhou Luo, Kui Wu, and Xinwen Fu. Protocol-level hidden server discovery. In *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM)*, pages 1043–1051. IEEE, 2013.

[68] George H Weiss and Robert J Rubin. Random walks: theory and selected applications. *Advances in Chemical Physics*, 52:363–505, 1983.

[69] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995.

[70] Peter Eckersley and Jesse Burns. An observatory for the SSLiverse. *URL: www. eff. org/files/DefconSSLiverse. pdf*, 2010.

[71] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. The SSL landscape: a thorough analysis of the x. 509 pki using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 427–444. ACM, 2011.

[72] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. Analysis of the https certificate ecosystem. In *Proceedings of the Internet Measurement Conference*, 2013.

[73] http://www.openssh.com/.

[74] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the ACM conference on Computer and communications security*, pages 38–49. ACM, 2012.

[75] Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith, Lars Baumgärtner, and Bernd Freisleben. Why eve and mallory love android: An analysis of android SSL (in) security. In *Proceedings of the ACM conference on Computer and communications security*, pages 50–61. ACM, 2012.

[76] Moxie Marlinspike. Null prefix attacks against SSL/TLS certificates. *Blackhat*, 2009.

[77] http://www.oss.com/asn1/resources/asn1-made-simple/introduction.html.

[78] Moxie Marlinspike. Defeating ocsp with the character '3'. *Blackhat*, 2009.

[79] Moxie Marlinspike. New tricks for defeating SSL in practice. *Black Hat DC*, 2009.

[80] Moxie Marlinspike. More tricks for defeating SSL in practice. *Black Hat USA*, 2009.

[81] Jeff Hodges, Collin Jackson, and Adam Barth. Http strict transport security (hsts). *URL: http://tools. ietf. org/html/draft-ietf-websec-strict-transport-sec-04*, 2012.

[82] http://trends.builtwith.com/docinfo/HSTS.

[83] Adonis PH Fung and KW Cheung. SSLock: sustaining the trust on entities brought by SSL. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 204–213. ACM, 2010.

[84] Bernhard Amann, Robin Sommer, Matthias Vallentin, and Seth Hall. No attack necessary: The surprising dynamics of SSL trust relationships. 2013.

[85] https://www.eff.org/https-everywhere.

[86] http://convergence.io/.

[87] Gabor Toth and Tjebbe Vlieg. Public key pinning for TLS using a trust on first use model. 2013.

[88] Paul Hoffman and Jakob Schlyter. The DNS-based authentication of named entities (dane) transport layer security (TLS) protocol: TLSA. Technical report, RFC 6698, August, 2012.

[89] Moxie Marlinspike. Trust assertions for certificate keys. 2013.

[90] Chris Evans and Chris Palmer. Public key pinning extension for http. 2011.

[91] http://web.monkeysphere.info/.

[92] https://www.owasp.org/index.php/Path_Traversal.

[93] https://www.owasp.org/index.php/Main_Page.

[94] Chris Anley. Advanced sql injection in sql server applications. *White paper, Next Generation Security Software Ltd*, 2002.

[95] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. In *ACM SIGPLAN Notices*, volume 41, pages 372–382. ACM, 2006.

[96] William GJ 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*, pages 174–183. ACM, 2005.

[97] Bradley W Hill. Command injection in xml signatures and encryption. *Information Security Partners*, 2007.

[98] Gary Wassermann and Zhendong Su. Sound and precise analysis of web applications for injection vulnerabilities. In *ACM Sigplan Notices*, volume 42, pages 32–41. ACM, 2007.

[99] Adam Kieyzun, Philip J Guo, Karthick Jayaraman, and Michael D Ernst. Automatic creation of sql injection and cross-site scripting attacks. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 199–209. IEEE, 2009.

[100] Stephen W Boyd and Angelos D Keromytis. Sqlrand: Preventing sql injection attacks. In *Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[101] Gregory Buehrer, Bruce W Weide, and Paolo AG Sivilotti. Using parse tree validation to prevent sql injection attacks. In *Proceedings of the 5th international workshop on Software engineering and middleware*, pages 106–113. ACM, 2005.

[102] William GJ Halfond and Alessandro Orso. Combining static analysis and runtime monitoring to counter sql-injection attacks. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–7. ACM, 2005.

[103] William GJ Halfond, Alessandro Orso, and Panagiotis Manolios. Using positive tainting and syntax-aware evaluation to counter sql injection attacks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 175–185. ACM, 2006.

[104] WG Halfond, Jeremy Viegas, and Alessandro Orso. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA*, pages 13–15, 2006.

[105] Sruthi Bandhakavi, Prithvi Bisht, P Madhusudan, and VN Venkatakrishnan. Candid: preventing sql injection attacks using dynamic candidate evaluations. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 12–24. ACM, 2007.

[106] Xiang Fu, Xin Lu, Boris Peltsverger, Shijun Chen, Kai Qian, and Lixin Tao. A static analysis framework for detecting sql injection vulnerabilities. In *Proceedings of the 31st Annual International Computer Software and Applications Conference*, volume 1, pages 87–96. IEEE, 2007.

[107] Yuji Kosuga, K Kernel, Miyuki Hanaoka, Miho Hishiyama, and Yu Takahama. Sania: syntactic and semantic analysis for automated testing against sql injection. In *Proceedings of the 23rd Annual Computer Security Applications Conference*, pages 107–117. IEEE, 2007.

[108] Michael Martin and Monica S Lam. Automatic generation of xss and sql injection attacks with goal-directed model checking. In *Proceedings of the 17th conference on Security symposium*, pages 31–43. USENIX Association, 2008.

[109] Wei Gao, Thomas Morris, Bradley Reaves, and Drew Richey. On scada control system command and response injection and intrusion detection. In *eCrime Researchers Summit (eCrime)*, pages 1–9. IEEE, 2010.

[110] Kevin Spett. Cross-site scripting. *SPI Labs*, 2005.

[111] Jesse Ruderman. The same origin policy, 2001.

[112] http://en.wikipedia.org/wiki/Cross-site_scripting#Server-side_versus_DOM-based_vulnerabilities.

[113] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the ACM symposium on Applied computing*, pages 330–337. ACM, 2006.

[114] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In *Proceedings of the 14th Annual Network & Distributed System Security Conference (NDSS)*, 2007.

[115] Gary Wassermann and Zhendong Su. Static detection of cross-site scripting vulnerabilities. In *Proceedings of the 30th ACM/IEEE International Conference on Software Engineering*, pages 171–180. IEEE, 2008.

[116] Prithvi Bisht and VN Venkatakrishnan. Xss-guard: precise dynamic prevention of cross-site scripting attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–43. Springer, 2008.

[117] Matthew Van Gundy and Hao Chen. Noncespaces: Using randomization to enforce information flow tracking and thwart cross-site scripting attacks. In *Proceedings of the 16th Annual Network & Distributed System Security Conference (NDSS)*, 2009.

[118] Mike Ter Louw and VN Venkatakrishnan. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 331–346. IEEE, 2009.

[119] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM, 2003.

[120] Mehdi Kiani, Andrew Clark, and George Mohay. Evaluation of anomaly based character distribution models in the detection of sql injection attacks. In *Proceedings of the 3rd International Conference on Availability, Reliability and Security*, pages 47–55. IEEE, 2008.

[121] Rajagopal Gaarudapuram Sriraghavan and Luca Lucchese. Data processing and anomaly detection in web-based applications. In *Proceedings of the IEEE Workshop on Machine Learning for Signal Processing*, pages 187–192. IEEE, 2008.

[122] Xinwen Fu, Zhen Ling, J Luo, W Yu, W Jia, and W Zhao. One cell is enough to break tor's anonymity. *Black Hat DC*, 2009.

[123] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the ACM workshop on Privacy in the electronic society*, pages 71–80. ACM, 2005.

[124] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 173–187. IEEE, 2009.

[125] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna PN Puttaswamy, and Ben Y Zhao. User interactions in social networks and their implications. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 205–218. ACM, 2009.

[126] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. A new cell-counting-based attack against tor. *IEEE/ACM Transactions on Networking*, 20(4):1245–1261, 2012.

[127] Amir Houmansadr and Nikita Borisov. Swirl: A scalable watermark to detect correlated network flows. In *Proceedings of the 18th Annual Network & Distributed System Security Conference (NDSS)*, 2011.

[128] Xiapu Luo, Junjie Zhang, Roberto Perdisci, and Wenke Lee. On the secrecy of spread-spectrum flow watermarks. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 232–248. Springer, 2010.

[129] Yabing Liu, Krishna P Gummadi, Balachander Krishnamurthy, and Alan Mislove. Analyzing facebook privacy settings: User expectations vs. reality. In *Proceedings of the ACM SIGCOMM conference on Internet measurement conference*, pages 61–70. ACM, 2011.

[130] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. The socialbot network: when bots socialize for fame and money. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 93–102. ACM, 2011.

[131] Alan Mislove, Bimal Viswanath, Krishna P Gummadi, and Peter Druschel. You are who you know: inferring user profiles in online social networks. In *Proceedings of the 3rd ACM international conference on Web search and data mining*, pages 251–260. ACM, 2010.

[132] Ramanthan Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web*, pages 403–412. ACM, 2004.

[133] Prateek Mittal, Matthew Caesar, and Nikita Borisov. X-vine: Secure and pseudonymous routing using social networks. *arXiv preprint arXiv:1109.0971*, 2011.

[134] NetworkX. http://networkx.github.io/.

[135] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 1999.

[136] Usage of server-side programming languages for websites. http://w3techs.com/technologies/overview/programming_language/all, 2013.

[137] Selenium Projects. http://seleniumhq.org.

[138] Captcha Monster. http://captchamonster.com/.

[139] Google guide, search operators. http://www.googleguide.com/advanced_operators_reference.html.

[140] http://www.alexa.com/topsites, 2013.

[141] Ninj@S3c. Arbitrary file download: Breaking into the system. http://resources.infosecinstitute.com/arbitrary-file-download-breaking-into-the-system/, 2013.

[142] Rfc1867. http://www.ietf.org/rfc/rfc1867.txt.

[143] Linux system calls. http://linasm.sourceforge.net/docs/syscalls/filesystem.php.

[144] Named data networking. http://named-data.net/.