The Pennsylvania State University
The Graduate School
College of Engineering

NEW TECHNIQUES FOR TRUSTWORTHY MOBILE COMPUTING

A Dissertation in
Computer Science and Engineering
by
Xin Chen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2015

The dissertation of Xin Chen was reviewed and approved* by the following:

Sencun Zhu
Associate Professor of CSE & IST
Dissertation Advisor, Chair of Committee

Guohong Cao
Professor of CSE

Wang-Chien Lee
Professor of CSE

Le Bao
Assistant Professor of STAT

Mahmut Kandemir
Professor of CSE
Graduate Program Chair

*Signatures are on file in the Graduate School.

Abstract

Technology advances in wireless networking have engendered a new era of computing, called *mobile computing*, in which users carrying portable devices have access to shared networks regardless of their physical locations. The new computing paradigm provides users with seamless access to networked services, and therefore, revolutionizes the way how computers are used. While more and more users enjoy the convenient networked services brought by mobile computing, the unique characteristics of mobile computing in communication channels (e.g., WiFi, GSM, Bluetooth, NFC, SMS), in hardware (e.g., PDA, smartphone and wearable device) and in software (e.g., Palm OS, Apple iOS, Google Android) also have raised many new concerns on trust, security and privacy.

In this dissertation, we present our studies on two types of mobile networks: *tactical networks* and *mobile phone networks*. For tactical networks, we elaborate our study about operational trust management and attack-resilient reputation management. Specifically, we first present ZIGZAG, a partial mutual revocation based trust management scheme, which allows rapid impeachment of identified malicious nodes, and then propose GLOBALTRUST, an attack-resilient reputation system for tactical networks, which aims at optimizing reputation assessment by identifying malicious nodes, and meanwhile, providing the consistency and resiliency.

For mobile phone networks, we first present DROIDLID, an automated functionality-aware privacy leakage analysis for Android applications. Prior approaches to detecting privacy leakage on smartphones primarily focused on the discovery of sensitive information flows but did not justify whether the leaked sensitive information flow is intended or not. In this study, we formulate it as a justification problem, which aims to justify the purpose of every sensitive information transmission in an app. We solve the justification problem by bridging the gap between the sensitive information transmission and application functions. Moreover, we propose SWEETDROID, a calling-context-sensitive, fine-grained privacy policy enforcement framework for Android OS. Our policy enforcement framework is able to distinguish sensitive data requests at different calling contexts and applies different pol-

icy rules automatically. The policy enforcement framework takes an important step towards applying the contextual integrity theory for mobile applications.

The design, implementation, demonstrations, and evaluation of proposed studies are elaborated in the dissertation.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deepest appreciation to my supervisor Dr. Sencun Zhu for his support throughout my dissertation research. Dr. Zhu has created a motivating, comfortable, and innovative research environment that I have benefited a lot such as the valuable research experience and expertise. Without all his guidance and help, this dissertation would not have been possible. It has been a great pleasure and honor for me to work under his supervision.

I am also grateful to Dr. Guohong Cao, Dr. Wang-Chien Lee and Dr. Le Bao for attending my Ph.D. dissertation committee and providing the great inspiration and insightful comments on my dissertation.

I would like to thank my co-author Zhi Xu, Wei Xu, Heqing Huang, Jin-Hee Cho, Mudhakar Srivatsa, and Harshal Patankar for numerous discussions, insightful suggestions, and spending valuable time helping me on my research works.

Dedication

Special thanks to my parents and my wife, Xiangyun. I am deeply indebted to them for spending so many years in the Ph.D. program. Their love, encouragement, understanding, and support give me strength and courage.

Chapter 1 — Introduction

1.1 Trust Management in Tactical Networks

Trust plays an important role in our daily life, as it enables people to assess a degree of uncertainty by utilizing their past experience. Let us take as an example the use of trust in electronic commerce. When customers in eBay and Amazon select sellers and purchase products, they often pay attention to the rating scores of sellers and products since rating scores indicate trustworthiness of sellers and products. As a result, sellers try to keep their rating scores as high as possible to seize the market. Besides, the concept of trust has been widely applied to various research areas, from social science, psychology, economics, media to computer science in the past few years.

In mobile ad hoc networks, trust also plays a crucial role in reinforcing security, reliability and availability. Trust management is a central issue in mobile ad hoc networks that are often deployed in adversarial settings and disaster management scenarios [1–3]. In such settings, compromised nodes can divert and monitor traffic, influence quorum-based decisions or spread harmful information. Moreover, since mobile ad hoc network routing involves a cooperative process among nodes, a secure routing mechanism must evaluate the trustworthiness of every node involved. Therefore, to limit the damage caused by compromised nodes and to provide a secure routing mechanism, agile trust management schemes that allow rapid impeachment of malicious nodes are vital for the network security.

Reputation assessment is another important issue in mobile ad hoc networks. In particular, many studies define reputation as a global perception of a node's

trustworthiness in a network, whereas trust indicates an individual node's perception of any other node's trustworthiness based on its direct observation. A desirable reputation system should be consistent, resilient and accurate. However, maintaining a consistent global view towards a node's reputation is challenging with uncertain or incomplete evidence in such hostile, distributed tactical network environments.

1.2 Privacy Threats in Mobile Phone Networks

Mobile devices such as smartphones and tablets have been an essential part of our modern life. However, at the same time, the large amount of sensitive information stored and processed on them, from device serial number, location to private messages, is of serious privacy concerns. Based on the summary from Mobile-Sandboxing [4], a malware analysis research company, user-privacy threatening malware families are prominent, which contribute to 63.1% of all malware families for the year 2014, and the privacy threat has increased by 3% as compared to 2013 and by 13% as compared to 2012. To remedy against privacy violations arising from mobile applications, the need for effective and efficient privacy leakage detection and prevention approaches is extremely emergent.

Privacy leakage detection on mobile applications has been heavily studied in recent years. Prior research primarily focused on the discovery of sensitive information flows [5–13]. However, as more and more benign apps send out users' sensitive information for legitimate application functions, these approaches cannot easily justify the purposes of sensitive information transmissions in an app, and hence may not detect privacy leakage effectively. For example, Google Maps sends out users' location information to a remote server for driving navigation and location-based recommendation services. To continue to be effective and adapt to the growing application markets, the development of more intelligent analysis approaches to detecting privacy leakage on smartphones is strongly desired.

Another crucial research direction to fight against serious privacy threats in mobile devices is to prevent privacy leakage and thus protect users' sensitive information. Although there are lots of systems for mobile device side privacy protection [11, 14–24], most of them are permission-based. None of them is based on individual information flows, which means that they do not distinguish the con-

textual information of the access to sensitive data. While one may force users to make a decision per sensitive data request, it is very impractical since it may cause dialog fatigue for users by handling every sensitive data request. Moreover, sometimes it is not obvious even to a tech-savvy user the reason for sensitive information access, leading to many decision errors [25, 26]. A calling-context-sensitive privacy protection solution for mobile devices is under high demand.

1.3 Contributions

In this dissertation, we study trust management in tactical networks, and privacy leakage detection and prevention in mobile phone networks. The major contributions of this dissertation are as follows.

1.3.1 Zigzag: Partial Mutual Revocation Based Trust Management in Tactical Ad Hoc Networks

In Chapter 2, we present ZIGZAG − a partial mutual revocation approach wherein we design a trust update function to temporarily punish both the accuser and accused node (without involving a quorum) − however, the trust update function does not essentially set their trust values to zero; instead it partially lowers the trust values of both the accuser and the accused. Similar to the complete mutual revocation approach a trusted authority or a quorum may (periodically) review such partial mutual revocations and update the trust values of the accuser and the accused nodes accordingly.

Specifically, the key contributions of our work in Chapter 2 include:

- We propose a partial mutual revocation based trust management, which not only offers the node a certain degree of freedom to tradeoff the extent of sacrifice with the global good of the network, but more importantly, reinforce the network against strategic (false) accusations made by bad nodes and erroneous accusations made by good nodes (e.g., due to benign errors in network monitoring).

- We present both analytical solutions that model evolving trust using recurrence equations over time and experimental evaluation to quantify the

efficacy of the proposed approach using three important metrics: revocation immediacy, accuracy, and abuse resistance. Both the analytical and experimental results show that the partial mutual revocation based trust management is able to rapidly impeach malicious nodes and highly resilient to malicious attacks targeting the proposed revocation scheme.

- Our proposed scheme encourages honest nodes to accuse malicious nodes by incentivizing them but at the same time discourages malicious nodes to do the same by penalizing them for making false accusations.

1.3.2 GlobalTrust: An Attack-Resilient Reputation System for Tactical Networks

In Chapter 3, we propose GlobalTrust, a reputation system for tactical networks with the goal of maximizing correct decision-making for identifying malicious entities.

Specifically, the key contributions of our work in Chapter 3 include:

- GlobalTrust provides an accurate, consistent view on the reputations of all nodes and detects malicious nodes in tactical networks.

- GlobalTrust can effectively deal with various types of attacks where all entities except the commander node may be compromised.

- GlobalTrust outperforms the existing reputation schemes (i.e., two schemes in PeerTrust [27]) in terms of view consistency and resilience against various types of attacks.

1.3.3 DroidJust: Automated Functionality-Aware Privacy Leakage Analysis for Android Applications

In Chapter 4, we formulate the problem of sensitive information leakage in mobile devices as a justification problem, which aims to justify if a sensitive information transmission in an app serves any purpose, either for intended functions of the app itself or for other related functions such as advertisements and analytics. To solve the justification problem, we propose an automated, functionality-aware approach,

called DroidJust. DroidJust not only identifies sensitive information flows but also tries to link each flow with certain application function to provide the evidence for justification.

Specifically, the key contributions of our work in Chapter 4 include:

- We propose a novel approach to automatically justify an app's sensitive information transmission by bridging the gap between the sensitive information transmission and application functions. Different to the previous work that utilize the evidence arising before or at the release point [25, 26], we are (probably) the first to consider the evidence arising after the release point for privacy leakage detection.

- Our approach overcomes several challenges to integrate all three types of PScout [28] resources (API, URI and Intent Actions) into DroidJust for labeling almost all (if not all) sensitive information sources in Android (in Section 4.2).

- We implement a prototype of DroidJust and evaluate it with more than 6000 Google Play apps and more than 300 known malware featuring privacy leakage. Our evaluation results demonstrate that DroidJust can effectively distinguish benign apps delivering sensitive information for application functions from the malware harvesting users' sensitive information.

- DroidJust identified 15 Google play apps that send out users' sensitive information but not for any application functions. Most of them cannot be detected by any anti-virus engine in VirusTotal and are still available for download in Google Play.

1.3.4 SweetDroid: Calling-Context-Sensitive Privacy Policy Enforcement Framework for Android

In Chapter 5, we aim at addressing these challenges by proposing a calling-context-sensitive privacy policy enforcement framework, named SweetDroid. Extending the existing Android framework, SweetDroid generates very fine-grained privacy policies targeting installed Android apps on Android devices, and enforces these privacy policies at the calling context level in application runtime to effectively enhance user privacy yet retain app's usability.

Specifically, the key contributions of our work in Chapter 5 include:

- We propose a novel privacy policy enforcement framework to enhance user privacy yet retaining application usability. Based on our best knowledge, our framework is the first to enforce privacy policies at the calling context level. It is able to distinguish sensitive data requests arising at different calling contexts, and correspondingly apply different privacy policies.

- Different to conventional policy enforcement schemes, our framework automatically generates privacy policies based on security analysis rather than forcing users to specify policies, which not only largely release normal users from the pain in making policies but also optimize the effectiveness of policies to achieve the sweet point balancing privacy protection and app usability.

- We implement a prototype of SweetDroid and evaluate it with Android apps from a third-party app market and known malware from VirusTotal [85]. Our evaluation results demonstrate that SweetDroid can effectively distinguish different sensitive data requests within an app, and respond appropriately to enhance user privacy yet retaining app usability.

1.4 Dissertation Outline

In summary, the rest of this dissertation is organized as follows: In Chapter 2, we present Zigzag, a partial mutual revocation based trust management scheme in tactical ad hoc networks. We propose GlobalTrust, an attack-resilient reputation system for tactical networks in Chapter 3. In Chapter 4, we present DroidJust, an automated functionality-aware privacy leakage analysis for Android applications. In Chapter 5, we propose a calling-context-sensitive privacy policy enforcement framework, named SweetDroid. Finally, Chapter 6 concludes our studies and present future works.

Chapter 2

Zigzag: Partial Mutual Revocation Based Trust Management in Tactical Ad Hoc Networks

Trust management is a central issue in ad hoc networks that are often deployed in adversarial settings and disaster management scenarios [1–3]. In such settings, compromised nodes can divert and monitor traffic, influence quorum-based decisions or spread harmful information. Also, since mobile ad hoc network (MANET) routing involves a cooperative process where route information is relayed between nodes, any secure routing mechanism must evaluate the trustworthiness of other nodes. Therefore, to limit the damage caused by compromised nodes and to provide a secure routing mechanism, agile trust management schemes that allow rapid impeachment of malicious nodes are vital for the security of the network.

One of the key challenges in operational trust management is to continually monitor the behavior of a node and update its trust score accordingly − evidently, both *speed* and *accuracy* is of great importance here. Past work has explored quorum based approaches wherein a group of nodes deliberate on trust revocation decisions and update the trust values of the accuser and the accused nodes accordingly − unfortunately, in the context of ad hoc networks such an approach lacks speed and agility. To address this problem, several papers have explored the concept of mutual revocation (also termed *suicide* [29–33]) wherein the trust value of both the accuser and the accused node are temporarily set to zero without involving a quorum; a trusted authority or a quorum (periodically) reviews such mutual revocations (suicides) and updates the trust values of the accuser and the

accused nodes accordingly − immediate revocation offers higher agility and speed which generally favors ad hoc network operation, albeit at the cost of lowering the accuracy (i.e., both good and bad nodes may get revoked).

In this chapter, we present ZIGZAG − a partial mutual revocation approach wherein we design a trust update function to temporarily punish both the accuser and accused node (without involving a quorum) − however, the trust update function does not essentially set their trust values to zero; instead it partially lowers the trust values of both the accuser and the accused. Similar to the complete mutual revocation approach a trusted authority or a quorum may (periodically) review such partial mutual revocations and update the trust values of the accuser and the accused nodes accordingly. In doing so we intuitively interpret trust score as a currency: a currency that is expended by both the accuser and the accused when they participate in (partial) mutual revocation; a currency that is earned when a quorum or a trusted authority agrees with the accuser's decision to (partially) revoke the accused; a currency that allows a node to exert higher influence in network operations (e.g., leader election, routing and forwarding decisions, etc.).

We remark that one of the key benefits of mutual revocation is its inherent ability to work in sparse dynamic networks where global (network-wide) evidences are hard to collect. Essentially, partial revocation offers the node a certain degree of freedom to tradeoff the extent of sacrifice with the global good of the network. In particular, we construct trust update mechanisms for partial mutual revocation that not only enables this tradeoff, but also is robust to strategic (false) accusations made by bad nodes and erroneous accusations made by good nodes (e.g., due to benign errors in network monitoring). We present both analytical solutions that model evolving trust using recurrence equations over time and experimental evaluation to quantify the efficacy of the proposed approach using three important metrics: revocation immediacy, accuracy, and abuse resistance. Revocation immediacy is the time taken for a node to be revoked from the network once it has been identified as malicious. Accuracy is mainly concerned with minimizing the effects caused due to misidentification of nodes. And finally, abuse resistance deals with avoiding malicious nodes from taking advantage of the proposed trust revocation scheme for their own benefit. Our scheme encourages honest nodes to accuse malicious nodes by incentivizing them but at the same time discourages malicious nodes to do the same by penalizing them for making false accusations.

2.1 Related work

The process of arriving at a revocation decision is the primary focus of the majority of revocation schemes presented to-date in the ad hoc networking literature [30,31, 33–37]. Assuming that a node has amassed sufficient evidence, various approaches have been introduced that require differing amounts of participation from other nodes in the network. That is, revocation decision making may be the result of a *collaborative* or *unilateral* decision process.

In collaborative schemes, nodes accuse other nodes of misbehaving by casting negative votes against them. If a predetermined threshold of negative votes is cast, then the offending node is considered revoked. By contrast, systemic revocation decision making has been proposed for use in Identity-based Public Key Infrastructures (ID-PKIs) for ad hoc networks [38]. As part of an ID-PKI, a validity period can be expressed in deriving a node's identifier. Once a node's identifier expires, the node must contact its (possibly distributed) TA and request a new private key, with a new expiry time. The TA in turn can decide whether to issue new keys during this re-enrollment process. In systemic-based decision making, the frequency of renewal (the longevity of an expiry period) is an important parameter: the higher the frequency, the less impact a compromised key may have on the network, but the greater is the effort that must be expended on key issuance procedures. This approach requires an on-line TA and may significantly increase traffic and thus energy consumption. The concept of unilateral decision making as a method of revocation was first introduced by Rivest in dealing with key compromise [39] in Public Key Infrastructures (PKIs). A user, upon detecting that his key has been exposed, declares his key invalid by issuing a signed message using the compromised key (indicating that this key is no longer to be trusted). This notion of suicide has recently been extended for use in ad hoc networks [29–31, 33]. A node commits suicide by broadcasting a signed instruction to revoke both its own key and the key of the misbehaving node. Suicide as a method of revocation in ad hoc networks has a number of attractive features when compared with collaborative decision making. With suicide, nodes can act immediately to a perceived threat. Additionally, suicide as a method of revocation is resistant to abuse due to the high cost associated with revoking another node.

It was first pointed out by Raya *et al.* [31] that in order for the suicide scheme to

work properly, the node should value the network utility more that his own utility. Raya *et. al.* [31] and Reidt *et. al.* [**?**] have developed methods to incentivize good nodes to participate in mutual revocation schemes. However, as duly acknowledged by both Raya *et. al.* [31] and Reidt *et. al.* [**?**], complete mutual revocation takes a heavy toll on the node. For example, consider a small network that contains only 10 nodes out of which two good nodes get involved in a mutual revocation. Raya *et al.* loose both the good nodes; Reidt *et al.* revive one of these nodes, but the other node is permanently revoked. Therefore, an accidental revocation profits the adversary. Every honest node revoked helps the adversary strengthen its influence on the network; also, in a non-cooperative environment it is more likely that the malicious nodes will collude to bring down an honest node than honest nodes cooperating to bring down a bad node.

2.2 Network and Security Models

Trust Model: We assume every node in the tactical ad hoc network has a public/private key pair, and the public key is known by every other node (or through public key certificate signed by a well-known CA). We further assume that each node in the network may have one or more identifiers along with its corresponding private keys. The trust level associated with an identifier is a fuzzy trust value which ranges between 0 (untrusted) and 1 (trusted).

Every node has an embedded Intrusion Detection System (IDS), which monitors its neighbors for any kind of malicious activity. For the sake of simplicity in this work, we focus on simple packet dropping attacks. Irrespective of the nature of such malicious activity, we assume that the IDS may be imperfect (typically represented by false positive and false negative rates). Hence, given an input from an imperfect IDS, a node may decide to launch an accusation against the purported bad node by broadcasting a digitally signed accusation message into the entire network.

In our model, a trusted authority (TA), as a network manager or administrator, can join the network in need. For example, in a tactical network, a TA could be the commander. Ideally most of the accusations that might take place in a network would be the result of malicious activity (e.g., actual packet dropping witnessed by nodes). But the rest of the accusations could be a result of the intentional false accusations made by the malicious nodes and unintentional false accusations

made by the good nodes (due to IDS imperfection). This can indeed be true as malicious nodes with the goal of disrupting the operation of the entire system may attempt to accuse as many honest nodes as possible. Therefore, to control any random or unjustified accusations, the TA reviews past accusations and helps identify whether an accusation was justified or not. The TA does so by making probabilistically correct decisions by, for example, posthumously interrogating witnesses or collecting evidences from other nodes in the network. Also, to incentivize nodes to make correct accusations, the TA rewards a node for a justified accusation by providing it additional trust and thus rewarding the node for its actions. After every judgment round, TA will permanently revoke any identifier with trust below a predefined threshold.

Adversary Model: In our model, we assume that the main goal of an adversary is to bring down the throughput of the entire network by dropping as many packets as possible. The simplest way to achieve this goal would be to drop all the packets that reach the malicious nodes. But, by doing so they also risk of being detected. So in order to maximize their overall influence on the network during their own lifetime, the malicious nodes can carefully choose a packet dropping rate. This would not only enable them to drop packets in an effective manner but also help them remain undetected by honest nodes for an otherwise longer time. Also based on different strategies, the adversary may even choose to abuse the scheme by making false accusations on honest nodes with the goal of reducing the average trust of honest nodes. If the traffic in the network is trust-driven, then this could lower the throughput.

2.3 Zigzag: Partial Mutual Revocation

2.3.1 Overview

In contrast to complete mutual revocation, during a partial mutual trust revocation, both accuser and the accused lose partial trust. For example, if node A accuses node B on its forwarding behavior, then both of them may partially lose data forwarding capability. The benefits of this are at least two-folds. First, intrusion detection systems (IDS), especially those based on forwarding behavior monitoring, are prone to errors because of network and systems complexity. The

network loses two benign nodes completely when a false accusation occurs, while by partial revocation the impact of such errors is limited. Second, even if a node is not completely trusted in data forwarding, it may still be safe to involve it in forwarding less critical messages. With appropriate replication through either multi-path routing or forward error correction, it would be possible to leverage the remaining resources of a suspicious node for enhancing network throughput.

2.3.2 Detailed Design of Zigzag

At a high level our protocol for partial mutual revocation can be summarized as follows: (i) Initially when the nodes are deployed in the environment they are all assumed to be benign. (ii) Whenever a node behaves maliciously, some of its neighboring nodes will accuse it of being malicious. This will lead to a drop in trust levels of both the nodes (accuser & accused). (iii) After a while, the TA would come online and analyze all the accusations occurred during its absence. It would then pass its own judgment based on the information it gathers. Based on that judgment the accuser and accused node's trust levels would be modified again. If the judgment is taken in favor of the accuser, then it will be given a small bonus in the form of trust and the trust level of the accused node will be left as it is. However, if the judgment is taken in favor of the accused node then the trust level of the accused node will be brought back to the original value and the trust level of the accuser node will not be recovered as a punishment of false accusation. Steps (ii) and (iii) are repeated as long as the network remains operational.

2.3.2.1 Trust Reduction

In our scheme once the accusation takes place, trust levels of both the nodes (viz. accuser and the accused) are reduced as follows:

$$T'_{Accuser} = T_{Accuser}(1 - \beta T_{Accused}) \tag{2.1}$$

$$T'_{Accused} = T_{Accused}(1 - \beta T_{Accuser}) \tag{2.2}$$

$\beta \in [0, 1]$ is a parameter to control the severity of accusation. Normally, an accuser could choose β based on the observed attack intensity—the more malicious the observed attack is, the larger β should be (e.g., using a linear or an exponential

function). An accuser may also choose β based on other technical or nontechnical considerations. Ultimately it gives the accuser the flexibility to decide the level of sacrifice it is willing to make. We will show how β impacts on the system in our evaluation section.

A key intuition here is that trust should be reduced in the same amount for both nodes (i.e., $\delta T_{Accuser} = \delta T_{Accused} = \beta T_{Accused} T_{Accuser}$) to prevent malicious nodes from taking advantage of the system. As soon as a node finds out that a neighbor is acting in a malicious way, it makes an accusation. For a malicious node, it can accuse another node at will. The accusation involves partially reducing both the accuser and accused node's trust and broadcasting a signed message to the entire network indicating the identifiers of both the accuser and the accused node. After the accusation takes place, each node carries on with it tasks – may it be forwarding packets or even making further accusations.

2.3.2.2 Judgment Criteria

Many such accusations can take place until the TA comes online. When the TA does come online, it collects all the accusations that took place since its last visit[1]. In order to pass a judgment on a specific accusation, the TA takes a probabilistic decision based on collected opinions from other nodes on the accused node.

		TA Judgment	
		Good	Bad
Reality	Good	p_t	p_f
	Bad	q_f	q_t

Table 2.1. TA's Judgment Probabilities

We abstractly quantify the efficacy of TA's decision using its false positive and false negative probabilities as described in Table 3.2. p_t denotes probability that TA classifies an honest node as an honest node. q_t denotes probability that TA classifies a malicious node as a malicious node. Also p_f and q_f are the false positive and false negative probabilities respectively. They are the probabilities with which TA misclassifies an honest node as a malicious one and malicious one as honest. Here $p_t + p_f = q_t + q_f = 1$.

[1]How to collect information in a MANET is a well-studied problem, so we do not study it here.

Event	Honest node profit	Malicious node profit
Honest node makes no accusation	0	0
Honest node accuses another honest node	$bp_f - \delta T p_t$	positive
Honest node accuses malicious node	1)$bq_t - \delta T q_f > 0$	2)$(\frac{M}{H-\delta T} - \frac{M}{H})q_f - (\frac{M}{H} - \frac{M-\delta T}{H+b})q_t < 0$
Malicious node makes no accusation	0	0
Malicious node accuses another malicious node	0	3) negative
Malicious node accuses honest node	$-\delta T p_f$	2)$(\frac{M+b}{H-\delta T} - \frac{M}{H})p_f - (\frac{M}{H} - \frac{M-\delta T}{H})p_t < 0$

Table 2.2. Profits made by an honest and malicious node for different kinds of accusations events. The profits stated represent an honest node's local view of the network and a global view for malicious node. In the column of "honest node profit", the inequalities should hold true for the scheme to be beneficial for honest nodes, whereas in the column of "malicious node profit", the inequalities should hold true for our scheme to be disadvantageous for the malicious nodes.

If an accusation was deemed correct, then the trust of the accuser is restored and it is given a small incentive in the form of additional trust. However, if the accusation was deemed incorrect, then the trust level of accused is brought back to the original value. The trust level of the accuser, however, is not brought back to the original value. This is done in order to penalize for making false accusations.

One can use several past approaches for trust assessment to provide us the functionality of a TA. In this work, we assume that the TA collects evidence from several nodes in the network and builds a classifier based on the k-means clustering algorithm [40]. The classifier outputs a 0 or 1 which indicates that the accused node is indeed guilty or not based on available evidences. The details of our approach are excluded from this work. We note that the correctness of our partial mutual revocation protocol depends only on the false positive and false negative probabilities of the judgment system; indeed one could in general use any judgment mechanism as long as one can quantify all the parameters in Table 2.1.

2.3.2.3 Trust Update

After the TA passes its judgment on an accusation, the trust levels of both the nodes (accuser and accused) are updated as follows. If the TA rules in favor of the accused, the trust level of the accused node needs to be brought back by adding the reduction value in the accusation. However, if the TA rules in favor of the accuser, then a bonus needs to be provided to the accuser node besides restoring the reduction value. The bonus encourages honest nodes and gives them a reason to make correct accusations and expose the nodes behaving maliciously. Now,

bonus can be calculated considering many aspects including: accused node's trust level prior to the accusation $T_{Accused}$, reduction in trust level of the accuser due to the accusation $\delta T_{Accuser}$, trust level of the accuser prior to accusation $T_{Accuser}$, node's previous streak (either winning or losing), etc. Now among these various aspects, if the bonus is based on $\delta T_{Accused}$, then the amount of bonus received would be based on the amount of trust lost by the accused node.

Intuition: If an honest node correctly accuses a malicious node which had a high trust level for some reason, then the honest node needs to get due credit for it. The bonus function to achieve this is:

$$b = \gamma \cdot \delta T_{Accused} \tag{2.3}$$

where $\gamma \in (0, 1)$ is a system set parameter. Therefore, the $T^{New}_{Accuser}$ can be written as:

$$T^{New}_{Accuser} = T_{Accuser} + \gamma \cdot \delta T_{Accused} \tag{2.4}$$

The reason for limiting the parameter to 1 is to ensure that accuser never receives bonus more than the amount of trust lost by the accused during the accusation. If on the other hand, $\gamma > 1$ were permitted then two malicious nodes could take undue advantage of this and build their own trust by simple accusing each other over and over.

2.3.2.4 Profit Evaluation

Table 2.2 lists the expected profit for honest and malicious nodes for each type of accusation events. These expected profits are based on the TA judgment probabilities listed in Table 3.2. For an honest node, the main motive is to be as useful as possible. This can be achieved by increasing its trust or by making correct accusations (thereby pinpointing malicious nodes in the network). On the other hand, malicious nodes want to disrupt network operations by reducing the average trust value of honest nodes or increasing their own average trust value. Therefore, we count malicious nodes' profit as the ratio of total trust values of malicious nodes to those of honest nodes. Then, we examine all possible accusation events to ensure the following requirements:

1. An honest node can obtain profit by following rules to accuse a malicious

node;

2. Malicious nodes cannot gain profit by accusing honest nodes or being accused by honest nodes;

3. Malicious nodes cannot gain profit by accusing with each other.

Let M be the total trust values of malicious nodes in the network and let H be the total trust values of honest nodes. δT is the amount of trust reduction of the accuser node or accused node in the accusation. An honest node can gain a bonus b if the accusation is justified whereas it will lose δT in trust level if not justified. Then for condition 1), we have $bq_t - \delta T q_f > 0$. For condition 2), in the case that an honest node accuses a malicious node, the ratio of total malicious nodes' trust values to total honest nodes' trust values is increased by $\frac{M}{H-\delta T} - \frac{M}{H}$ if TA favors the malicious node whereas the ratio is reduced by $\frac{M}{H} - \frac{M-\delta T}{H+b}$ if TA favors the honest node; in the other case that a malicious accuses an honest node, the ratio of total malicious nodes' trust values to total honest nodes' trust values is increased by $\frac{M+b}{H-\delta T} - \frac{M}{H}$ if TA favors the malicious node whereas the ratio is reduced by $\frac{M}{H} - \frac{M-\delta T}{H}$ if TA favors the honest node. Hence, we can conclude two inequalities for condition 2) as shown in Table 2.2. For condition 3), recall from the trust update section that the choice of parameter γ ($\gamma < 1$) ensures that malicious nodes do not gain by accusing other malicious nodes. For condition 1) and 2), we derive the lower bounds for the TA's judgment probability p_t and q_t:

$$p_t > \frac{bH + \delta T M}{\delta T(H - \delta T)} p_f,\ q_t > \max(\frac{(H+b)\delta T M}{(H-\delta T)(bM + \delta T H)}, \frac{\delta T}{b}) q_f$$

The loosely derived lower bounds for both are as follows:

$$p_t > \frac{\gamma H + M}{H - 1} p_f,\ q_t > \max(\frac{M}{H-1}, \frac{1}{\gamma}) q_f$$

2.3.2.5 Trust-Aware Partial Data Access

Different with the situation in complete mutual revocation, a node being partially revoked in partial mutual revocation still retains some extent of network capability based on its trust level, and hence can contribute to the network. However, to

differentiate nodes with various trust levels, it is necessary to associate a node's trust level with its network capability. For example, trust is widely used to assist route selection [41, 42] and therefore nodes with low trust levels are more likely to be excluded from packet forwarding. Here, we propose trust-aware partial data access, which aims to tie a node's trust level to its capability in data access. A group key is used to protect the confidentiality of communication within the network. We require that a node with low trust can only have partial group key and thus understand partial data packets encrypted by the group key.

Problem Model: We assume every data packet has a secrecy level which is ranged from 0 (least secrecy) to $M-1$ (most secrecy). Every node j has a hierarchical rank R_j from 0 (least capability) to $M-1$ (most capability) in the MANET corresponding to its trust level T_j. Hierarchical ranks of nodes are evaluated by TA periodically by using a linear function: $R_j = \lfloor T_j \cdot M \rfloor$. The design goal is to ensure that a node can only decrypt any data packet whose secrecy level is at most the node's rank evaluated by TA recently.

Scheme: Whenever TA finishes updating all nodes' trust levels after accusation judgment, TA evaluates all nodes' ranks and distributes new group keys based on the new ranks. In each group rekeying, TA generates a key chain of size M. Let the keys in the key chain generated for the rekeying at time t be $K^{M-1}(t), K^{M-2}(t), \cdots, K^1(t), K^0(t)$, where $K^0(t) = H(K^1(t)) = H^2(K^2(t)) = \cdots = H^{M-1}(K^{M-1}(t))$ and H is a one-way hash function such as SHA-1. Due to the one-wayness of the hash function, a node that knows $K^i(t)$ can iteratively compute the keys $K^{i-1}(t), \cdots, K^0(t)$ and understand any packets encrypted by these keys, but cannot compute any of the keys $K^{i+1}(t), \cdots, K^{M-1}(t)$ and understand the associated packets. Then, TA sends to each node j key $K^{R_j}(t)$ for the next communication session. Each node derives its own sub keychain based on the one-way function. During the session, a sender will encrypt a data packet whose secrecy level is j with group key $K^j(t)$. And, only the nodes whose ranks are at least j can decrypt and understand the packet. Besides, the scheme ensures that a node cannot properly encrypt any data packet whose secrecy level is higher than its own rank.

2.3.3 Basic Analytical Model

To make our analysis tractable, we assume that TA comes online every interval time T_1 to handle all the accusation events that happened during its absence. T_1 is a system parameter that the authority can adjust by considering both the efficiency and overhead. Also, each honest node takes a time period of T_2 to gather enough evidence, via its IDS, to launch an accusation against a malicious node. The choice of this parameter mainly depends on the performance of IDS by considering a tradeoff between accusation immediacy and accuracy. We define the timeline of TA's online round i as $[i \cdot T_1, (i+1) \cdot T_1)$. Hence, the maximum number of accusations that can be launched by a node in each TA's online round is given by: $\omega = T_1/T_2$. To avoid malicious nodes from repeatedly accusing honest nodes for abusing the whole system and limit the overhead of accusation traffic, any node that make more than ω accusations in one round will be directly revoked by the TA. For analysis purpose, we assume that the IDS of an honest node will accuse a malicious node with a probability of α_i when the malicious node's attack intensity is α_i: in the context of packet forwarding α_i is the probability with which a malicious node drops packets in round i. Hence, the average number of accusations made by each honest node in i is:

$$\lambda_i = \sum_{k=0}^{\omega} k \cdot \binom{\omega}{k} \cdot \alpha_i^k \cdot (1-\alpha_i)^{\omega-k} = \omega\alpha_i \tag{2.5}$$

Denote the *average* trust levels of honest nodes and malicious nodes as two-dimension arrays T_h and T_m, respectively. Specifically, $T_h(i, j)$ and $T_m(i, j)$ are the respective average trust levels of honest nodes and malicious nodes after the jth accusation round during round i. i starts from 0 and continues till the network is operational. j starts from 0 and ends at λ_i.

Given that two malicious nodes have no incentive to accuse each other and two honest nodes rarely accuse each other, the total number of accusations that involve honest nodes must equal those that involve malicious nodes. Hence, in each round for every accusation that involves an honest node, there are $\frac{n}{m}$ accusations that involves a malicious node (where n is the number of honest nodes and m is the number of malicious nodes and typically, $n > m$). Therefore, based on

Formula (1) and (2), trust value of malicious nodes evolves as follows [2]: $T_m(i,\, j) = T_m(i,\, j-1) \cdot (1 - \beta T_h(i,\, j-1))^{\frac{n}{m}}$

It is a little more complex to approximate $T_h(i,\, j)$ because the average trust value of malicious nodes varies considerably after every m accusations. Hence, we divide an accusation round into $\frac{n}{m}$ phases, and at each phase m of n honest nodes accuse malicious nodes for a total of m times. Hence, the average trust of m honest nodes at each phase k (from 1 to $\frac{n}{m}$) is formulated as below:

$$T_{h\_k}(i,\, j) = T_h(i,\, j-1)(1 - \beta T_m(i,\, j-1) \cdot (1 - \beta T_h(i,\, j-1))^{k-1})$$

To approximate the average trust of n honest nodes, we add up the average trust of m honest nodes at each phase with a weight of $\frac{m}{n}$, i.e., $T_h(i,\, j) = \sum_{k=1}^{\frac{n}{m}} \frac{m}{n} \cdot T_{h\_k}(i,\, j)$. After deduction, we have:

$$\begin{aligned} T_h(i,\, j) \;=\; & T_h(i,\, j-1) \cdot (1 - \\ & \frac{m T_m(i,\, j-1)(1 - (1 - \beta T_h(i,\, j-1))^{\frac{n}{m}})}{n \beta T_h(i,\, j-1)}) \end{aligned}$$

To recursively solve the above two formulas, we first need to determine $T_h(i,\, 0)$ and $T_m(i,\, 0)$. Clearly, $T_h(0,\, 0)$ and $T_m(0,\, 0)$ are the initial trust levels assigned to good nodes and bad nodes, respectively. For analysis purpose, let us assume they are both 0.8. More general, $T_h(i,\, 0)$ is determined by judgment decisions based on λ_{i-1} accusations at the $(i-1)$th round. We can formulate $T_h(i,\, 0)$ by considering all of λ_{i-1} accusations one by one in a probabilistic way. It is much simpler to get the average trust level of malicious nodes $T_m(i,\, 0)$ as TA passes its judgment only once for each malicious node and all accusation events containing the same accused node follow this judgment decision. If accusations against malicious nodes are justified, the malicious nodes would keep the trust level as in the end of TA's previous online round; otherwise, the average trust level of malicious nodes would be restored to that in the beginning of TA's previous online round. To summarize, we have the following formulas:

[2] We acknowledge that the function approximates the expectation of trust levels by directly involving the expected value for the simplicity.

$$
\begin{aligned}
T_h(i,\,0) &= T_h(i-1,\,0) \\
&\quad + \sum_{j=1}^{\lambda_{i-1}} (q_t \cdot \gamma \cdot (T_m(i-1,\,j-1) - T_m(i-1,\,j)) \\
&\quad -(1-q_t) \cdot (T_h(i-1,\,j-1) - T_h(i-1,\,j))) \qquad (2.6) \\
T_m(i,\,0) &= q_t \cdot T_m(i-1,\,\lambda_{i-1}) + (1-q_t) \cdot T_m(i-1,\,0) \qquad (2.7)
\end{aligned}
$$

Now that we have captured the evolution of trust for malicious and honest nodes through recurrence equations over time, we quantify the expected reward for a malicious node as follows. For the sake of simplicity we assume that in each round i, the expected reward for a malicious node is proportional to the product of their average trust level and their attack intensity α. The reward for a malicious node at each round i is evaluated as $\frac{\sum_{j=0}^{\lambda_i - 1} T_m(i,j)}{\lambda_i} \cdot \theta^i \cdot \alpha_i$, where $0 < \theta < 1$ is a discount factor[3] that weighs immediate reward for a malicious node more than its future rewards. Hence, the total profit of a malicious node over the network lifetime is:

$$
R = \sum_{i=0}^{\infty} \frac{\sum_{0}^{\lambda_i - 1} T_m(i,\,j)}{\lambda_i} \cdot \theta^i \cdot \alpha_i \qquad (2.8)
$$

The malicious nodes could strategically choose a vector $\alpha = (\alpha_1,\, \alpha_2,\, \ldots)$ to maximize their expected long-term reward.

2.3.4 Evaluation

This subsection contains two revocation cases: honest nodes accusing malicious nodes and malicious nodes accusing honest nodes. For the first case, our evaluation is based on the basic analytical formulas derived in Section 2.3.3. For the second, we wrote C simulators based on our trust update formulas (Formulas 1-4) without considering node mobility.

[3]Discount factor is commonly used approximation for infinite horizon problems [43, 44].

2.3.4.1 Honest Nodes Accusing Malicious Nodes

In this part of the evaluation, we assume only honest nodes accuse malicious nodes. Figure 2.1 shows how the TA's accuracy (q_t and p_t) changes with the attack intensity α based on the aforementioned k-means clustering algorithm. As the attack intensity α increases, the probabilities that TA correctly identifies a malicious and an honest node dramatically increase, and they both nearly reach 1 when α is over 0.7.

2.4 Security analysis

The analytical results shown next were carried out with a fixed malicious-to-honest node ratio $(\frac{m}{n}) = 0.5$ and the varying TA's accuracy which changes with the attack intensity based on the curves shown in Figure 2.1. The number of accusation rounds (i.e. ω) in a TA's online interval is set to 10. β is set to 1. Except for Figure 2.4, γ is 0.1 by default. To validate our analytical model, we also did a number of simulations with the same parameter settings as the above and the validation results are shown in Figure 2.2 and 2.3.

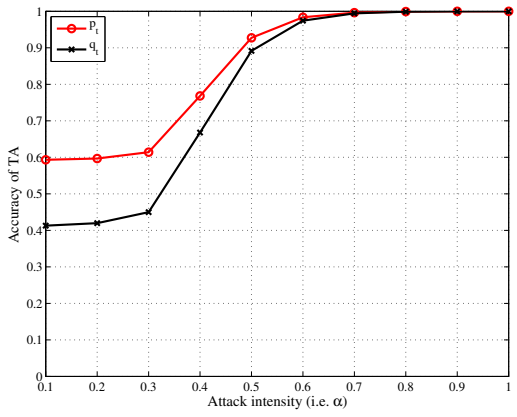


Figure 2.1. TA accuracy vs. intensities

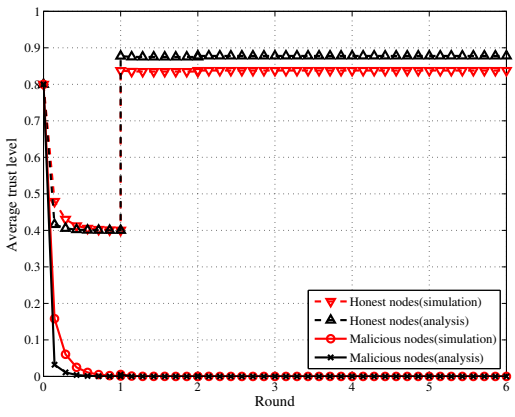


Figure 2.2. Avg. Trust for $\alpha = 0.7$

Figure 2.2 shows how the average trust of honest and malicious nodes changes over different rounds. Here the attack intensity for a malicious node is set to 0.7. First, we can see the simulation results match with the analytical results very well. Second, as the rounds progress, the average trust associated to malicious nodes

decreases quickly whereas the average trust associated to honest nodes does not. This is due to the fact that during each round many accusations against malicious nodes take place and at the end of each round all those accusations are judged by the TA. When the TA passes its judgment, malicious nodes hardly recover to their trust of the previous round. On the other hand, the honest nodes are awarded a bonus for making correct accusations. Since the deducted trust of accused nodes also depends on the trust level of accusers, with higher trust levels, honest nodes can bring down the average trust level of malicious nodes more in later rounds.

Figure 2.3 shows how the average trust of malicious nodes changes over different rounds with varying attack intensities. Each curve in the figure, either by simulation or by analysis, corresponds to a fixed attack intensity. First, we can see that the analysis and simulation results are very close, thus validating our analytical models. Second, as the attack intensity α increases, the average trust of malicious nodes decreases more quickly over each round. This happens because honest nodes have a higher probability to accuse bad nodes at each accusation round as α increases. Higher accusation probability reflects higher accusation frequency in the figure. Another reason is that as α increases, the TA's accuracy q_t also increases (as shown in Figure 2.1). Thus, the trust level of malicious nodes is less likely to be restored by TA. We can also see that the change of trust level over rounds is a zigzag shape, so we name our revocation protocol ZIGZAG.

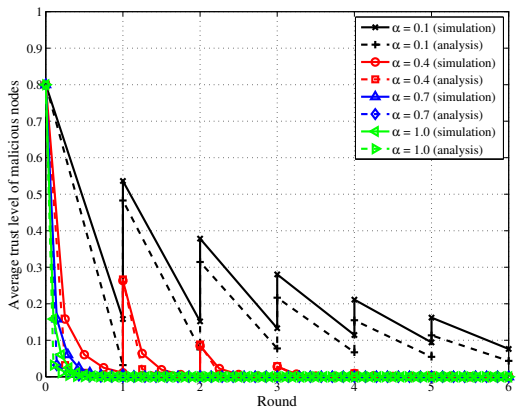


Figure 2.3. Avg. Trust of malicious nodes vs. α

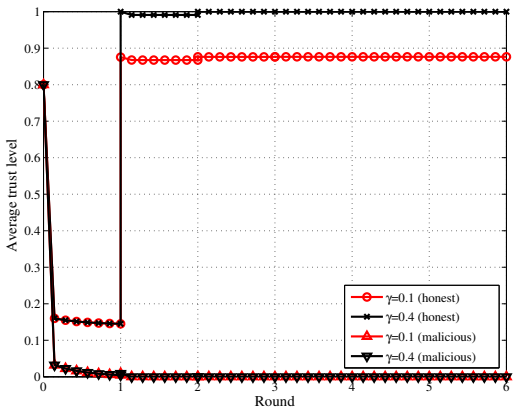


Figure 2.4. Avg. Trust of honest and malicious nodes vs. γ

Figure 2.4 shows how the average trusts of malicious nodes and honest nodes

change under different values of γ. It is very clear that average trust of honest nodes increases as the bonus parameter γ increases. Consequently, the average trust of malicious nodes drops much faster.

Figure 2.5 shows the total profit earned by a malicious node over its entire lifetime with various attack intensities. Here, the attack intensity α is fixed to a certain value during the entire lifetime, which means $\alpha_1, \alpha_2, \ldots$ are equal. It reveals that in our scheme malicious nodes can achieve higher long-term profit with a relatively low attack intensity. Especially, the optimal α is around 0.2. This is because q_t and p_t are affected by the attack intensity. Here the functionality of bad nodes is assumed to be proportional to their trust levels.

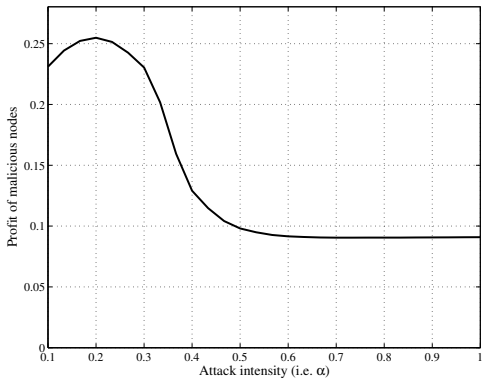


Figure 2.5. Profit of mal. nodes vs. α

Here an interesting question is: assuming the global knowledge of the system, would the malicious nodes gain a higher long-term profit by adopting a different attack intensity α_i at a different round? That is, the malicious nodes may drop packets at different rates at different rounds so that they may drop the maximal number of packets. We use an efficient heuristic search algorithm to figure out this optimal attack intensity vector under the searching granularity of 0.1. Interestingly, our result suggests the optimal attack intensity vector α_{opt} in this setting is the same as the previous optimal α, that is, all $\alpha_i = 0.2$.

2.4.0.2 Malicious Nodes Accusing Honest Nodes

As noted previously, the main motive of malicious nodes is to disrupt the network. Accusing honest nodes repeatedly and bringing their average trust level down could

be one of the effective ways as this can lower the network throughput. We present two different types of attack strategies below.

Figure 2.6 represents the *One-to-One* attack scenario where each malicious node accuses a different honest node at each accusation round. No two malicious nodes accuse the same honest node. Their main motive is to bring down as many honest nodes as possible in a single round. As the TA updates trust levels at the end of each round, a malicious node keeps accusing the same honest node until it is evicted from the network. Here, we compare the average trusts of malicious nodes and honest nodes under three different settings of $\frac{m}{n}$. It can be seen from the figure that as $\frac{m}{n}$ increases, i.e., with more malicious nodes, the one-to-one accusation is more effective by the end of TA's round. However, as the TA updates the trust levels, the trust of malicious nodes decreases whereas the trust of honest nodes increases. This happens because the TA judgment probability (p_t) is usually higher than 0.5 to support honest nodes. As the rounds progress, the average trust of honest nodes is not affected much but the average trust of malicious nodes reaches zero.

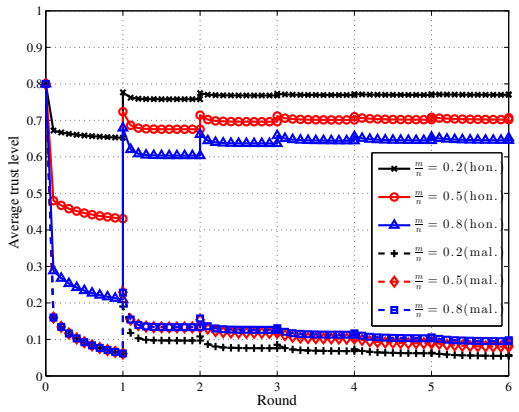


Figure 2.6. One-to-One accusation (p_t = 0.8)

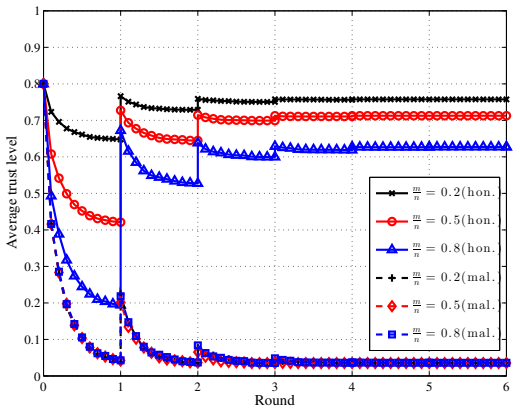


Figure 2.7. Many-to-One accusation (p_t = 0.8)

Figure 2.7 represents a *Many-to-One* attack scenario where malicious nodes collude and keep on accusing honest nodes sequentially. Specifically, at each accusation round, each malicious node will pick to accuse the active honest node whose trust level is the lowest. As a result, an honest node will receive multiple accusations until its trust level is below a threshold and considered inactive. The main objective is to break down as many honest nodes as possible. A predefined

threshold (0.05 in our simulation) is set to determine whether an honest node is active according to its trust level. It can be seen from the figure that many honest nodes are brought down in the first couple of rounds. When the round comes to an end the TA updates all the trust levels, the average trust of malicious nodes falls below that of honest nodes. This trend continues in the first few rounds until their difference becomes rather large. Once that happens malicious nodes are not able to make any sizable impact on honest nodes' trust levels, let alone bring them down. As they continue this type of sequential accusation, their own trust level decreases and after few rounds it approaches zero.

Based on the comparison of 2.6 and Figure 2.7, it can be observed that from the adversary's point of view, the Many-to-One attack and the One-to-One attack do not have much difference in their accusation effectiveness. This observation has the following important indication. As these two accusation strategies represent two extreme cases for an adversary, any other accusation strategy, which we do not enumerate here, would be a hybrid version of these two cases, so its attack effectiveness would also be similar.

2.4.0.3 Mutual Accusation

Finally, we consider the case that malicious nodes and honest nodes accuse each other. Here, we mix the above scenarios to simulate a more realistic setting. Under this attack scenario, malicious nodes perform malicious activities dropping packets with the attack intensity α of 0.7, which lead to the potential accusations by honest nodes. Besides, at each accusation round, they will randomly choose an active honest node (i.e., whose trust level is above 0.05 in our setting) to accuse. It can be seen from Figure 2.8 that compared with Figure 2.7 and Figure 2.6, the average trust of malicious nodes falls more quickly because their dropping behavior is sensed and hence they are accused by honest nodes.

2.5 Comparative Study

We further evaluate ZIGZAG through simulations and show its *revocation immediacy* and *accuracy* in comparison with the complete revocation scheme [32]. Besides, we study how β impacts security.

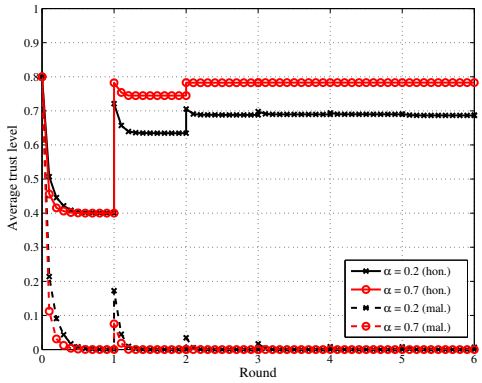


Figure 2.8. Mutual accusations

Our simulation is based on the simulator GloMoSim 2.03. Table 2.3 describes the general parameter settings. We choose DSR as the routing protocol and 802.11 as the MAC layer protocol. The mobility model is random waypoint; node's velocity is between 0 and 10 m/s and its pausing time is 30 time units. In the simulation, periodically each node requests a simple response from a neighbor chosen at random. An honest node will always send back an ACK in response whereas a malicious node only responds to the request at a probability pre-determined by its attack intensity. At each accusation round, an honest node will choose to accuse a node that is identified to be malicious by its IDS, if any. Because of the imperfection of its IDS, an accused node might or might not be an actual bad node. The TA will come online to judge all accusations and update the trust levels of all accuser and accused nodes every T_2 time. By default, the judgment accuracy is fixed. Both ZIGZAG and complete revocation are implemented and evaluated under this setting. The results are averaged over 20 independent runs.

Revocation Immediacy is the time taken (or the number of accusations needed) for a node to be revoked from network once it is identified as malicious. *Accuracy* is mainly concerned with minimizing the effect caused by faulty IDS (leading to false accusations). Figure 2.9 provides a comparison on these two metrics between partial and complete revocation. Figure 2.9 shows that the average trust level of malicious nodes drops quickly. In the complete revocation scheme, it reaches almost 0 in 3 rounds, and in ZIGZAG it reaches to 0.05 in 3 rounds. Although this indicates the former has higher revocation immediacy, in our partial revocation scheme, a node is considered revoked after its trust level drops below 0.05. So in

	Parameters	Description	Default
Scenario	N	# of nodes in the network	100
	n	# of honest nodes in the network	80
	m	# of malicious nodes in the network	20
	α	attack intensity of malicious nodes	1
	T_2	interval of TA online round	100 units
Zigzag	β	coefficient of trust reduction	0.1
	γ	coefficient of trust reward	1
Complete revocation	b	coefficient of trust reward	1
IDS	p_{f_IDS}	false positive of IDS	0.05
	q_{f_IDS}	false negative of IDS	0.05
	T_1	interval of accusation round	2 units
TA	p_f	false positive of TA judgment	0.2
	q_f	false negative of TA judgment	0.2

Table 2.3. Parameter Setting

this sense, the revocation immediacy of ZIGZAG is close to that of the complete revocation scheme.

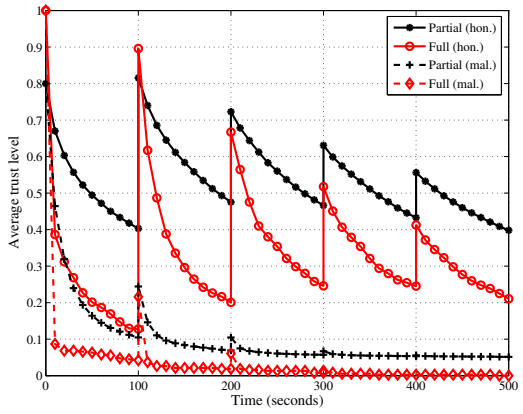


Figure 2.9. Partial vs. complete revocation

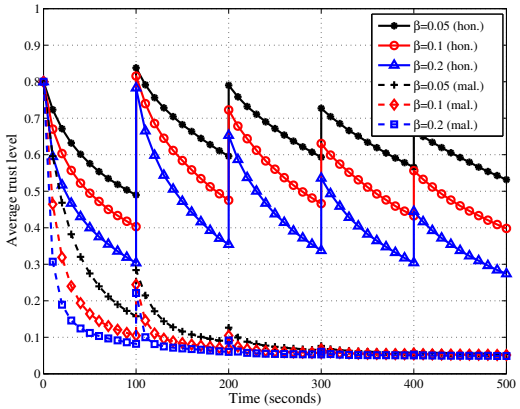


Figure 2.10. Ave. trust level vs. β

On the other hand, the figure also shows the big advantage of ZIGZAG over the complete revocation scheme in terms of accuracy. Due to false positives of IDSs in mobile nodes, false accusations have a much greater influence on the complete revocation scheme, as its average trust level of good nodes is far below that of ZIGZAG. This demonstrates ZIGZAG works much better in tolerating IDS inaccuracies.

The parameter β denotes the degree of trust reduction in ZIGZAG. From Figure 2.10, we can observe the tradeoff between accusation immediacy and accuracy. When β decreases, the revocation speed is lower but accuracy is better. To the

opposite, when β increases, the revocation speed is faster but accuracy is worse. Hence, β can be flexibly selected based on the context of a network, the severity of observed attacks, and other factors.

2.6 Summary

In this chapter, we have introduced ZIGZAG, a new scheme for trust management in ad hoc networks. First, based on a node's fuzzy trust value, its network privileges are modulated under a model of partial revocation. Second, for better revocation immediacy and abuse resistance, we explored the idea of mutual trust revocation. The partial revocation approach presents its trade-offs between revocation immediacy and accuracy. Third, by providing trust in the form of incentives, it encourages honest nodes to make right accusations but at the same time also discourages malicious nodes by penalizing them for making false accusations. Our future work will study other possible attack strategies as well as more extensive simulations to compare Zigzag with other existing revocation schemes.

Chapter 3 — GlobalTrust: An Attack-Resilient Reputation System for Tactical Networks

Military tactical networks often face challenges in designing security protocols because they require additional precautions compared to civilian networks, including high hostility, distributed network characteristics, node subversion, and node heterogeneity. The mixture of wired/wireless communication mediums and high tempo operations cause rapid changes in network topology and service requirements. Since communities of interest (e.g., mission/task teams) are formed dynamically, participating nodes may not have any pre-defined trust relationships to each other. A tactical network may consist of heterogeneous entities characterized by humans (e.g., soldiers), robots, or unmanned/manned vehicles equipped with devices such as machines and/or sensors. In this work, we use the terms *a node* and *an entity* interchangeably to represent heterogeneous entities (or nodes) above. Military tactical networks typically have a hierarchical structure where a commander makes critical decisions to control all other entities in the network [45]. In this scenario, for the commander, it is critical to perceiving an accurate view towards other entities for making right decisions. For example, when the commander wants to form a temporary mission team, called *a military coalition*, based on an acceptable trust level of nodes, the accuracy of trust assessment towards each node significantly impacts mission success.

One of the common applications using trust management mechanisms is to identify malicious entities in order to protect the network from attackers. The malicious entities may disrupt system security goals by performing network attacks such as loss of service availability (e.g., denial-of-service, packet dropping), and/or

loss of data integrity (e.g., good/bad mouthing, message forgery/modification). In this work, we propose an attack-resilient reputation management mechanism that can accurately assess nodes' trustworthiness in the presence of highly hostile entities.

Trust or reputation management has been extensively studied in various domains [46]. In particular, many studies define *reputation* as a global perception of a node's trustworthiness in a network, whereas *trust* indicates an individual node's perception of any other node's trustworthiness based on its direct observation. A desirable reputation management should be able to provide the following features in the network:

- **Consistency**: provide a consistent view of the reputation of a node based on the consensus of honest nodes.
- **Resiliency**: be resilient to common security threats.
- **Accuracy**: derive valid reputation values based on accurate trust assessment.

Maintaining a consistent global view towards the node's reputation is challenging with uncertain or incomplete evidence in hostile, distributed tactical network environments.

In this chapter, we propose a reputation system, the so called GlobalTrust, for tactical networks for maximizing correct decision-making by identifying malicious entities.

3.1 Related Work

Trust or reputation management (TRM) schemes have been extensively studied in various domains. In general, the term *trust management* is interchangeably used with the term *reputation management* [47]. However, a slight difference between trust and reputation has been clarified in the literature. According to Liu *et al.* [48], trust is a node's belief in trusting a peer, a subjective view towards its peer. Reputation is the perception that peers form about a node. Thus, reputation can be estimated based on the aggregation of peer nodes' trust values. Again in our work, we use trust to indicate a node's subjective opinion towards other nodes

based on its own observations (i.e., LTOs) while reputation means the converged view towards a particular node, computed based on multiple opinions of other nodes such as the commander node's reputation values about other nodes.

Cho *et al.* [46] discussed the unique properties of trust, which can be distinguished from reputation, such as subjectivity, incomplete transitivity, asymmetry, and context-dependency. Fundamentally, trust is a subjective belief a node has towards another node. On the other hand, reputation is a relatively objective concept because it is computed based on evidence from the majority of third parties.

Aberer and Despotovic [1] presented a trust-based reputation management scheme that is scalable for data management without any centralized control without considering collusive attacks. Kamvar *et al.* [49] proposed a distributed and secure method for reputation management that effectively identifies and isolates malicious nodes using the pre-trusted authority. Xiong and Liu [27] proposed two reputation-based trust models to evaluate a node's reputation in a fully distributed manner: trust-value based credibility measure (TVM) and personalized similarity measure (PSM). However, TVM is vulnerable to collusion attack while PSM generates discrepancies in reputation about the same entity for different evaluators.

Zhou and Hwang [50] introduced a reputation system using the power-law feedback provided by power nodes to aggregate reputation values in order to build a robust P2P system. Bella *et al.* [51] proposed a reputation management scheme that enables a node to exchange and update other nodes' reputation values in mobile ad hoc networks (MANETs). Arboit *et al.* [52] introduced a computational reputation model that considered accusations against nodes in MANETs. These works [51, 52] do not deal with the false recommendation attack that significantly deters accurate reputation assessment. Some other existing reputation management schemes [53–56] evaluate the reputation of a node subjectively based on the evaluator's direct observation, ultimately leading to inconsistent global reputation view in the network.

Quorum-based decision on detecting malicious nodes has been extensively studied based on *k-out-of-n* threshold signatures [57,58]. The key idea to these schemes is to determine the threshold $k + 1$ that is an upper bound of negative votes to diagnose a node as compromised. However, it has a limitation in obtaining a sufficient number of votes under highly dynamic network environments and also did not consider any collusive attack. Later, Reidt *et al.* [32] proposed a revocation

decision making scheme by employing the k-means clustering algorithm for trust management. Nevertheless, the k-means-based judgment scheme is vulnerable to inconsistent (or conflicting) recommendation attack.

3.2 Preliminaries

3.2.1 Problem Statement and Challenges

We assume each node in a tactical network is pre-installed with a monitoring mechanism [59] characterized by detection error probability ε. This enables a node to directly observe its neighboring nodes' behavior. With this monitoring capability, each node can derive Local Trust Opinions (LTOs) about its neighboring nodes based on direct observations. For example, $LTO_{w,u}$ is node w's trust opinion towards node u based on *direct* observations. If node w has not encountered with node u, there will be no LTO. Let $p_{w,u}$ and $n_{w,u}$ be the total number of positive events and total number of negative events that node w observed about node u, over the period of encountering time. The LTO of node w towards node u during this time period, $LTO_{w,u}$, is calculated as:

$$LTO_{w,\,u} = \frac{p_{w,\,u}}{p_{w,\,u} + n_{w,\,u}} \tag{3.1}$$

$LTO_{w,u}$ is a real number scaled in [0, 1]. Note that if the total number of observed events, $p_{w,u} + n_{w,u}$, is 0 (i.e., no direct observation), $LTO_{w,u}$ will be set as a *null* value. These LTOs form an LTO matrix where each entry $LTO_{i,j}$ is the LTO of node i towards node j. The following is a simple example of an LTO matrix with six nodes in the network, where the fourth and sixth nodes are malicious nodes giving false (dishonest) LTOs. Here empty entries indicate *null* values.

$$\text{LTO} = \begin{pmatrix} & 0.82 & 1 & 0 & & 0.26 \\ 0.93 & & & & 0.88 & 0.20 \\ 0.96 & & & 0 & 0.93 & \\ 0.05 & 0 & 0 & & 0 & 1 \\ & & 0.89 & 0.18 & & 0.23 \\ 0 & 0 & 0.07 & 1 & 0 & \end{pmatrix} \tag{3.2}$$

We define the density of an LTO matrix, denoted as d, as the proportion of non-null LTOs (i.e., real values) in the matrix and calculate d as follows:

$$d = \frac{|\{(i,\, j) : LTO_{i,j} \neq null\}|}{N(N-1)} \tag{3.3}$$

where N is the number of nodes. Besides, every LTO is time-stamped to keep track of its freshness. Every node may store its LTOs using, for example, in-network storage technology with multiple copies to mitigate the potential data loss in a distributed network environment. That is, LTOs are stored and fetched in a distributed hash table (DHT) like P-Grid [1].

Given an LTO matrix for a given time period, our goal is to develop a reputation management scheme that can provide the network authority (e.g., a commander) with the capability of *consistent* and *accurate* assessment on the reputation of every node. That is, the proposed scheme aims to meet the following key requirements: (1) providing a consistent reputation value towards a node based on an LTO matrix; and (2) minimizing the inaccuracy of reputation evaluation introduced by intentionally injected false LTOs and imperfect monitoring error. To achieve these goals, we face two major challenges:

- **No pre-trusted LTOs**: The nodes which provide LTOs are not pre-trusted, so their LTOs cannot be trusted. In other words, the commander node cannot directly use these LTOs to derive reputation values for nodes.
- **Incomplete/Sparse LTO matrix**: The LTO matrix may be incomplete and even sparse due to the lack of observations or malicious nodes suppressing their LTO reports during an evaluation period.

3.2.2 Networw Model and Assumptions

GlobalTrust is a very generic framework, as long as it has an LTO matrix as the input. The LTO matrix can be generated from any group where members rate each other. Hence, GlobalTrust can be applied to the context of MANETs, peer-to-peer networks, Internet, or social networks. For concreteness, we assume that the targeted environment is a tactical network consisting of multiple mobile nodes communicating through multiple hops. For secure communication, each node is pre-loaded with a public/private key pair or pairwise shared keys.

In our work, a network is allowed to be hierarchical in that nodes may have different ranks in the structure. Node k's hierarchical rank, HR_k, represents the importance of its role in the network. For instance, it is a very common scenario in a tactical network where entities with different ranks, such as a commander and his/her members, collaborate in a common mission.

In the considered military scenario, we allow a trusted authority (TA), such as a commander node, to be online periodically or as needed to collect evidence to assess reputation of other nodes and make trust decisions. None of the regular nodes is pre-trusted. We note that if in certain scenarios, when a single TA involves a security, safety, and/or performance concern, standard protocols [60] can be hold to distribute such a trust role into multiple regular nodes in the network, leading to reaching a consensus on the trustworthiness of all nodes. Such extensions are orthogonal to the reputation management algorithm in GlobalTrust.

A node may behave honestly, or may be compromised and perform various types of attacks. Now we describe various types of attack behaviors considered in this work below. We assume that honest nodes are a majority in the network, not allowing the Byzantine Failure condition due to too many malicious entities in the network. we demonstrate the impact of the ratio of malicious nodes on decision accuracy in Section 3.4.3. As a measure of reputation, we consider the degree of compliance with a given network protocol (i.e., not performing network attacks and reporting honest LTOs compared against those of majority entities) by an entity.

3.2.3 Adversary Model

A malicious node (*aka* a compromised node or attacker) is defined as a node not complying with a given network protocol by either denying requested services or providing false LTOs. We model the degree of an attacker's misbehavior with attack intensity, α, ranged in [0, 1]. With this attack intensity, we can model a random attack behavior where an attacker performs an attack with probability α while exhibiting honest behavior with probability $(1 - \alpha)$. Malicious nodes may collude to promote their reputations via good mouthing attacks while demoting honest nodes' reputations via bad mouthing attacks. Malicious nodes may provide false LTOs that are opposite to their actual observations. In this work, we consider

the following attack behaviors:

- **Naïve Malicious Attack (NMA)**: A compromised node may provide improper services, not complying with a given network service protocol. However, it does not lie in reporting its LTOs.
- **Collusive Rumor Attack (CRA)**: In addition to providing improper services, malicious nodes collude to report false LTOs (i.e., good/bad mouthing attacks) for disrupting accurate trust or reputation assessment.
- **Non-collusive Rumor Attack (NRA)**: Without colluding with other malicious nodes, a malicious node can report a false LTO that is opposite to the observed evidence. For example, if an LTO is evaluated as p, the malicious node may report $1 - p$ for the LTO.
- **Malicious Spy Attack (MSA)**: Some malicious nodes misbehave while other malicious nodes, called *malicious spies*, behave normally by providing proper services. These malicious nodes may collude and form an attacker community to perform good/bad mouthing attacks by reporting false LTOs, in order to subvert the entire trust and reputation system [61].
- **Conflicting Behavior Attack (CBA)**: Malicious nodes can behave inconsistently to different parties. This attack aims to disseminate conflicting (or inconsistent) LTOs. For example, they may misbehave only to a subset of honest nodes (referred to as *target nodes*) to intensify the LTO discrepancy between targeted and non-targeted honest nodes. This attack may reduce the overall attack intensity due to the nature of intermittent misbehavior.

3.3 GlobalTrust

3.3.1 Overview

With a commander node taking the role of TA, GlobalTrust is deployed on TA to evaluate the global reputations of all nodes. Whenever TA comes online, it collects all LTOs with timestamps during the last offline interval. In this section, we discuss how to evaluate global reputation values of all nodes by aggregating

both true and false LTOs, without any prior knowledge of which LTOs are true or false.

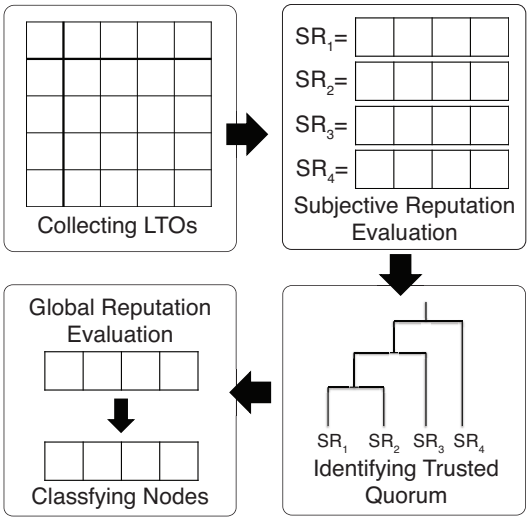


Figure 3.1. Workflow of GlobalTrust

Assuming that honest nodes are a majority in the network, they are expected to form a consistent view on other nodes even in the presence of conflicting evidence. We call the view that node i has towards node j a *subjective reputation* (SR); it is computed by TA based on both the $LTO_{i,j}$ and the LTOs that other nodes have over node j. Section 3.3.2 will detail how to compute the subjective reputation. We use a machine learning technique, called *hierarchical clustering*, to identify a minimum dominating set of nodes as a trusted quorum based on the similarity among their subjective views. Then we evaluate the reputation of a node by converging the subjective reputations of nodes in the quorum. Based on the computed reputation value of each node, TA judges the trustworthiness status of each node according to three reputation statuses: *honest*, *malicious* or *unknown*. Fig. 3.1 summarizes the key processes of the GlobalTrust.

3.3.2 Subjective Reputation Evaluation

In our model, each node can compute LTOs only for other nodes it has directly interacted with, not for remote nodes because no direct evidence is available. However, TA can evaluate all nodes based on the LTOs provided by all nodes in the network. With the LTO matrix, TA will first calculate the subjective reputations

(SR) of each node based on subjective trust values, LTOs, provided by all nodes in the network. Let $SR_{w,u}$ denote the reputation of node u evaluated by TA as it could have been subjectively assessed by node w. In the evaluation, node w trusts its own LTO. We use a weighted average to compute $SR_{w,u}$:

$$SR_{w,u} = \sum_{j \in S_u} LTO_{j,u} \cdot \frac{HR_j \cdot Sim(w, j)}{\sum_{j \in S_u} HR_j \cdot Sim(w, j)} \tag{3.4}$$

where S_u is the set of nodes that have non-*null* LTOs over node u (including w if w has one), $LTO_{j,u}$ is the LTO of node j over node u, HR_j is node j's hierarchical rank, and $Sim(w, j)$ is the similarity between LTOs reported by node w and node j. The rationale behind the formula is as follows. From node w's viewpoint, to evaluate the reputation of another node u, besides its own direct observation (if any), the LTOs over node u reported by other nodes can be taken into consideration too. Node w weighs other nodes' $LTO_{j,u}$ values based on the similarity between its own view and node j's view. That is, it weighs more the opinions from others with more similar views to its own. The similarity of LTOs between node w and node j is measured based on a *cosine* function with the input of their LTO vectors:

$$Sim(w, j) = max(\cos(\mathbf{LTO'}_w, \mathbf{LTO'}_j), 0). \tag{3.5}$$

Here we adopt a *cosine* function to capture the similarity of two LTOs represented by two vectors. The *cosine* similarity result is ranged in $[-1, 1]$, where -1 refers to complete dissimilarity in the two opinions, 1 complete similarity, and 0 ignorance (uncertainty), indicating orthogonal opinions. Before computing the *cosine* similarity of two vectors, the LTOs in both vectors are linearly mapped to the scale of $[-1, 1]$, re-scaled from the original scale in $[0, 1]$. The re-scaled **LTO** vector is denoted as $\mathbf{LTO'}$. Note that if there is no common set between two vectors, the *cosine* similarity value is set to 0. Further, the similarity result is adjusted to 0 if the *cosine* similarity value of the two vectors is negative, which excludes evidence

provided by untrusted nodes due to the dissimilarity. $SR_{w,u}$ is evaluated by:

$$SR_{w,u} = \begin{cases} \text{if } \sum_{j \in S_u} HR_j \cdot Sim(w, j) \neq 0, \\ \sum_{j \in S_u} LTO_{j,u} \cdot \frac{HR_j \cdot Sim(w,j)}{\sum_{j \in S_u} HR_j \cdot Sim(w,j)} \\ \text{else if } S_u \neq \varnothing, \\ \sum_{j \in S_u} LTO_{j,u} \cdot \frac{HR_j}{\sum_{j \in S_u} HR_j} \\ \text{else} \\ null \end{cases} \tag{3.6}$$

When $\sum_{j \in S_u} HR_j \cdot Sim(w, j) = 0$ (i.e., the denominator in Equation 3.4), this indicates that node w does not have directly observed evidence towards u, nor did any other nodes with whom node w shares positive similarity. In this case, we average the existing LTOs on node u with the HR of each recommender as the weight for $SR_{w,u}$, if any. If none of the nodes in the network has LTOs on node u (i.e., $S_u = \varnothing$), we set it to a *null*. Note that if $S_u = \varnothing$, $SR_{w,u}$ is *null* for any w.

3.3.3 Assessment of Trusted Quorum

After computing the SR for each pair of nodes, TA generates an SR matrix. The SR tuple in node w's view is denoted as vector $\mathbf{SR}_w = (SR_{w,1}, \cdots, SR_{w,N})$. There are N SR tuples in total. Our next step is to identify a subset of the SR tuples as TA's *trusted quorum*. Intuitively, SR tuples from honest nodes tend to be similar and hence form a cluster, while those from malicious nodes may form another cluster or are irregularly distributed subject to specific false recommendation attack patterns. We call a cluster *dominating* if the number of nodes in the cluster exceeds the half of a network size. We aim to find the *minimum dominating cluster* to represent the trusted quorum. The reasons are two folds: the dominating size guarantees that the SR tuples in malicious nodes' views cannot form such a big cluster while the minimum requirement contributes to excluding inaccurate SR tuples, due to false reported LTOs and imperfect direct observations, as much as possible. We use the *agglomerative hierarchical clustering* technique to build a hierarchy of clusters based on all the SRs and find a minimum dominating cluster.

Fig. 3.2 is a simple example of *hierarchical clustering dendrogram*. In this method, each node starts with its own cluster, and the pairs of clusters with the

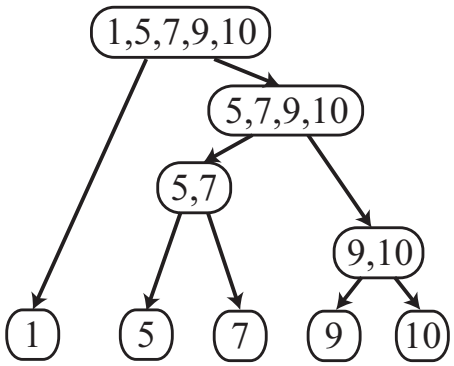


Figure 3.2. An example of hierarchical clustering dendrogram

nearest distance are merged continuously until only one cluster remains. Eventually, it forms a hierarchical clustering tree. Here, the distance of two values a and b, denoted as $dist(a, b)$, is $|a - b|$, and the distance of two clusters A and B is defined as $\max\{dist(a, b) : a \in A, b \in B\}$. Fig. 3.2 describes the example procedures of hierarchical clustering dendrogram as follows: (1) the cluster $\{9\}$ and the cluster $\{10\}$ are merged since their distance of 1 is the smallest; (2) the cluster $\{5\}$ and the cluster $\{7\}$ are merged because the current smallest distance is 2; (3) the cluster $\{5, 7\}$ and the cluster $\{9, 10\}$ are combined since the smallest distance becomes 5 after that; (4) the cluster $\{5, 7, 9, 10\}$ merges with the cluster $\{1\}$ to complete the hierarchical clustering.

Applying this method, we categorize N SR tuples into a hierarchical clustering tree by assigning each SR tuple into a leaf node. In our case, the distance between any two SR tuples is their *Euclidean distance* and the distance between two clusters follows the same definition above. Therefore, the minimum dominating cluster, denoted as D, is the first cluster formed in the *agglomerative clustering* whose size is over $N/2$. This cluster D becomes TA's trusted quorum. To compute the *agglomerative clustering*, we use the *nearest-neighbor chain algorithm* [62]. The overall time and space complexity for the nearest nearest-neighbor chain algorithm is $O(N^2)$ and $O(N)$, respectively, where N is the number of nodes in the network.

3.3.4 Global Reputation Evaluation

We compute the global reputation of each node considering two aspects of reputation: *behavioral reputation* (BR) and *credibility reputation* (CR). Node u's behavioral reputation, BR_u, reflecting how other nodes view node u's network behavior,

is computed by averaging the SR tuples in D:

$$BR_u = \begin{cases} \text{unknown} & \text{if } S_u = \varnothing \\ \frac{\sum_{w \in D} SR_{w,\,u}}{|D|} & \text{otherwise} \end{cases} \tag{3.7}$$

S_u is the set of nodes that have LTOs over node u, $SR_{w,\,u}$ is the SR of node u in node w's opinion. When no LTOs towards node u are available in the network (i.e., $S_u = \varnothing$), BR_u is set to *unknown*. In the case, $SR_{w,\,u}$ must be *null* for any w, as mentioned previously.

Node u's credibility reputation, denoted as CR_u, indicates how trustworthy u's reported LTOs (i.e., $\mathbf{LTO}_u$) are. It is estimated based on the difference between u's reported LTOs and BRs of the nodes that node u has reported LTOs over. This implies that if the behavioral reputation of a node j is evaluated to be good, node u also has a very positive LTO over j, meaning u's LTO is more credible. The credibility of node u's LTOS, CR_u, is estimated by:

$$CR_u = \begin{cases} \text{unknown} & \text{if } \mathbf{LTO}_u = \text{null} \\ 1 - \sqrt{\frac{\sum_{j \in \{LTO_{u,j} \neq null\}} (LTO_{u,j} - BR_j)^2}{|\{j | LTO_{u,j} \neq null\}|}} & \text{otherwise} \end{cases}$$

Note that when node u does not report any LTOs (i.e., $\mathbf{LTO}_u = \text{null}$), *unknown* is assigned to CR_u. In this case, its global reputation is solely computed based on its behavior.

Finally, TA computes the global reputation of node u by:

$$GR_u = \begin{cases} \gamma BR_u + (1-\gamma) CR_u & \text{if both } \textit{known} \\ \text{unknown} & \text{if both } \textit{unknown} \\ CR_u & \text{if only } BR_u = \textit{unknown} \\ BR_u & \text{if only } CR_u = \textit{unknown} \end{cases} \tag{3.8}$$

Here $\gamma \in [0, 1]$ is used to normalize the global reputation values.

After TA computes global reputation (GR) values of all nodes, it can judge the trustworthiness of each node u as one of three statuses: *malicious*, *honest*, or

unknown by:

$$Decision(u) = \begin{cases} \text{unknown} & \text{if } GR_u = unknown \\ \text{honest} & \text{if } GR_u \geq \theta \\ \text{malicious} & \text{if } GR_u < \theta \end{cases} \tag{3.9}$$

where θ is a decision threshold selected from the range in $[0, 1]$ that may be adjusted to minimize detection errors (we will examine the impact of θ in our simulation experiments).

3.3.5 Security Analysis

For security analysis, let us first consider the case that a malicious node behaves consistently to other nodes (i.e., NMA, CRA, NRA and MSA attacks). In this case, honest nodes have high consistent views (LTOs) on every malicious node as well as on every honest node, meaning high similarity of LTOs between two honest nodes. On the other hand, the similarity of LTOs between an honest node and a malicious node depends on how faithfully the malicious node reported its LTOs. The more faithfully, the higher the similarity. Therefore, by converging the LTOs with their similarity as weight, $SR_{w,u}$ is highly accurate to reflect node u's behavioral reputation when node w is honest. That is, SR tuples in honest nodes' views are highly consistent and accurate. Note that for a malicious node w, $SR_{w,u}$ could be inaccurate if node w reports false LTOs, or accurate if *node* reports LTOs honestly to actually contribute to reputation aggregation. By leveraging hierarchical clustering, the consistent and accurate SR tuples with a minimum dominating size will form a trusted quorum to eventually evaluate the behavioral reputations of all nodes accurately, which can effectively identify malicious nodes in the attacks including NMA, CRA and NRA. With the help of accurate behavioral reputation, the scheme can accurately evaluate the credibility reputation of nodes and hence effectively identify malicious spies in MSA.

There is a case that malicious nodes behave inconsistently to different honest nodes (i.e., CBA). Even in this case, since honest nodes have high consistent views on honest nodes consisting of a majority of the nodes in the network, they are more likely to form the trusted quorum even if malicious nodes may exhibit inconsistent

network / reporting behavior. Note that if malicious nodes report their honest LTOs, they are likely to be involved into the trusted quorum and contribute to accurate reputation assessment. This, thus, enables their credibility reputations (CR) to maintain high. However, their behavioral reputations (BRs) will be low, which causes their overall global reputations lower than those of honest nodes. In this sense, our scheme is resilient against malicious nodes performing CBA and accordingly can effectively identify honest and malicious nodes, except the case with the following two cases: (1) when the ratio of malicious nodes is very close to 50% (see Section 3.4.3 for the analysis); and (2) the LTO matrix is too sparse, leading to the case the LTOs of malicious nodes form the majority in the LTO matrix.

3.4 Performance Evaluation

3.4.1 Simulation Setup

We evaluate GlobalTrust through extensive simulations using C. The network model uses a set of human-mobility traces from CRAWDAD [63]. In the collection of the datasets, all participants were equipped with Global Positioning System (GPS) receivers to log their positions per 30 seconds. We use the dataset by KAIST (Korea Advanced Institute of Science and Technology) which uses mobility traces of 92 nodes. The nodes of a simulated network are split into two types of nodes, honest or malicious nodes. The numbers of malicious and honest nodes are denoted as m and h, respectively. The ratio of malicious nodes is denoted as k ($= \frac{m}{m+h}$) and its default value is set to 0.3. Each node is assumed to have an equal hierarchical rank, except TA, taking the role of a higher rank commander.

The networking traffic is simulated based on packet forwarding behavior. Every node randomly requests one of its neighboring nodes to forward a packet as a relay for 100 times per minute, where the one-hop wireless radio range is 250 meters. For honest nodes, they cooperate in forwarding packets with probability $(1 - e) = 0.95$ and drop packets with probability $e = 0.05$. Honest nodes are supposed to provide LTOs of other nodes based on their direct observations. After a node forwards a packet to a neighbor node, it would monitor the neighbor's behavior on packet forwarding. Packet forwarding is regarded as positive behavior while packet

dropping is counted as a negative behavior. We consider the inherent detection error probability in the monitoring mechanism with $\varepsilon = 0.05$, providing falsely observed report towards the observed events (e.g., reporting opposite results). We summarize attackers' behavior pattern discussed in Section 3.2.3 in Table 3.1. Note that α is the probability that a malicious node drops a packet and $\alpha = 0.5$ as the default.

Model	Behavior	Recommendation
NMA	misbehaving with prob. α	honestly reporting LTOs
NRA	misbehaving with prob. α	reporting opposite LTOs, $1 - \alpha$
CRA	misbehaving with prob. α	reporting LTOs of 1 to malicious nodes and LTOs of 0 to honest nodes
MSA	half malicious nodes misbehaving with prob. α; the other half malicious nodes behaving honestly	reporting LTOs of 1 to malicious nodes and LTOs of 0 to honest nodes
CBA	misbehaving with prob. α to half honest nodes; behaving honestly to the other half honest nodes	reporting LTOs of 1 to malicious nodes and LTOs of 0 to honest nodes

Table 3.1. Malicious attack patterns

TA computes the reputation of each node every 30 minutes. Based on TA's online interval and mobility traces, we observe that on average a node encounters with 39% of all nodes as a 1-hop neighbor. The LTOs submitted in the previous offline time frame (i.e., the last 30 minutes) are collected to estimate global reputations and make decisions about nodes' statuses (i.e., honest, malicious, or unknown). The coefficient γ is set to 0.7 to weigh the behavioral reputation (BR) higher than the credibility reputation (CR) because malicious behavior is able to cause direct attacks to the network performance (e.g., throughput), whereas false LTOs may be filtered out by GlobalTrust and hence introduce less negative impacts on the network. We set the decision threshold, $\theta = 0.8$, to determine whether a node is malicious or honest. The simulation is run 1000 for each scenario for the results shown here.

3.4.2 Performance Metrics

In this work, we consider detection errors (i.e., FPs and FNs) on trust decisions evaluated by TA as performance metrics. We show all possible decision cases in Table 3.2. For the nodes classified as *unknown*, this is the case when they neither provide any recommendation nor interact with any other nodes, regarded as inactive in the network. We do not consider them for our performance analysis. For an

active node, four outcomes are possible, as in Table 3.2. We mainly use both false positive (FP) and false negative (FN) probabilities as our performance metrics to indicate judgment (decision) errors. Besides, we use receiver operating characteristics (ROC) analysis as a performance metric, indicating correct detection probability.

		TA Decision	
		Malicious	Honest
Ground Truth	Malicious	true positive (TP)	false negative (FN)
	Honest	false positive (FP)	true negative (TN)

Table 3.2. Detection types

3.4.3 Comprehensive Evaluation

This subsection gives a comprehensive evaluation of decision errors with respect to three factors: the probability of attack intensity (α), ratio of malicious nodes (k), and malicious attack patterns. Finally, we show how the selection of decision threshold (θ) affects the decision accuracy.

Fig. 3.3 illustrates how the decision errors vary as attack intensity, α, increases. We observe that the maximum FP is less than 0.03 over the entire range of α except for CBA. Under NMA, the FP is close to zero because of no false (dishonest) recommendations. Under CBA, the FP increases slightly up to 0.07 when α increases. This is because the increasing α can increase the dissimilarity between honest nodes' LTOs and malicious nodes' LTOs, which ultimately affects the subjective reputation evaluation. However, the negative impact is small because the similarity of honest nodes' LTOs over honest nodes ensures the credibility of LTOs.

Fig. 3.3 also shows that the observed FN is close to 0, except the case of NMA with $\alpha < 0.3$ showing a significant number of malicious nodes is falsely identified as honest. With small α, malicious nodes under NMA do not exhibit misbehavior much, accordingly leading to high FN. Under all other attacks, malicious nodes' global reputations are downgraded due to their misbehavior and false LTOs, leading to the situation that most of them are classified as malicious even with low α. Besides the spies in MSA are well identified even if they show consistent honest behaviors. Therefore, considering credibility reputation (CR) in deriving the overall global reputation significantly helps identify malicious nodes showing inconsistent behavior such as intermittent reporting of false recommendations.

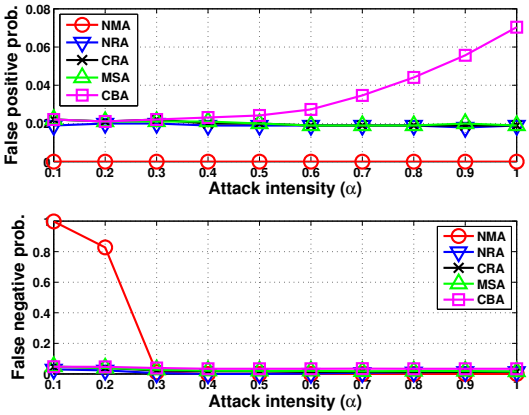


Figure 3.3. Decision error vs. α

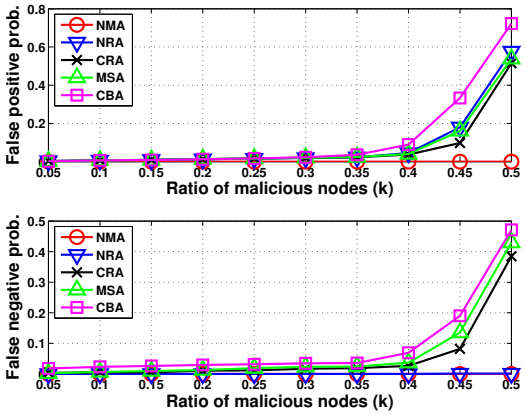


Figure 3.4. Decision error vs. k

Fig. 3.4 reveals how the increasing ratio of malicious nodes, k, degrades the decision errors. The FP increases but stays below 0.1 under all types of attacks with $k < 0.4$. When k increases up to 0.5, the FP increases rapidly and stays around 0.7 at the end. This is because the trusted quorum derived from the hierarchical clustering may include more malicious nodes than honest nodes when malicious nodes become a majority of the network. We also observe that CBA is more detrimental than NRA, CRA and MSA when k increases. For attacks such as NRA, CRA and MSA, a higher k means more malicious nodes in the trusted quorum. However, when malicious nodes perform the CBA attack, a higher k not only increases the degree of malicious activities, but also affects the subjective reputation evaluation by honest nodes, because the CBA attackers generate more conflicting LTOs.

When k is below 0.4, the FN increases with k under MSA, CRA and CBA and it reaches 0.06 at its maximum under CBA whereas it is almost zero under NMA and NRA. For higher $k > 0.4$, the FN increases as quickly as the FP increases because the trusted quorum tends to include more malicious nodes. Again, CBA impacts more detrimentally than CRA and MSA over a wider range of k. This is because malicious nodes performing the CBA can obtain high trust values in behavior reputation (BR) while still performing attacks, compared to malicious nodes performing CRA and MSA. Overall GlobalTrust is fairly resilient to the attacks considered above when $k < 0.4$.

Fig. 3.5 shows how the density of an LTO matrix, d, affects the decision errors.

The result shows that both FP and FN stay low and they fluctuate a little bit as the density of an LTO matrix, d, increases under all other attacks considered. Hence, GlobalTrust is adaptive to a wide range of LTO density, d, as low as 20%.

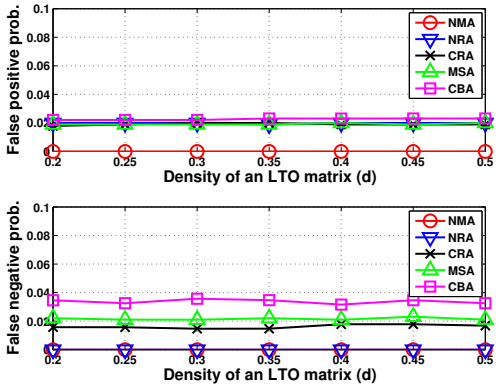


Figure 3.5. Decision error vs. d

Fig. 3.6 visualizes how GlobalTrust performs w.r.t. θ using the ROC metric. We consider three collusion attacks including CRA, MSA and CBA. The y-axis is the TP probability, referring to the probability of correctly detecting malicious nodes, while the x-axis denotes the FP, meaning the probability of detecting a good node as bad. The value labeled with each point is the decision threshold, θ. The observed general trend is that the TP probability increases with θ and FP (< 0.05). In Fig. 3.6, to ensure that ROC (detection probability) is above 0.7, θ should be as low as 0.4, 0.4 and 0.6 under CRA, MSA and CBA, respectively. Similar to our observation in previous results, malicious nodes performing CBA have higher reputation values than those performing CRA and MSA.

When the threshold, θ, increases from 0.7 to 0.8 under MSA, ROC significantly increases by 0.13 between these two thresholds. This implies that sufficiently high θ is required to maximize ROC. We observe that $\theta = 0.8$ is optimal under the given condition because this ensures the smallest fluctuation of FP and FN, 0.05, under the considered attacks.

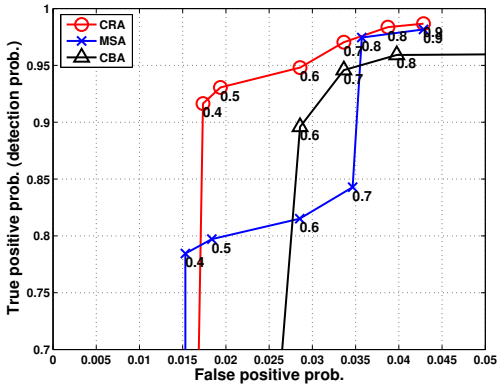


Figure 3.6. ROC curve by varying θ

3.4.4 Comparative Performance Analysis

This subsection presents two performance comparison studies: (1) GlobalTrust vs. *k-means clustering-based judgment* scheme [32]; and (2) GlobalTrust vs. two existing reputation methods (i.e., TVM and PSM) in *PeerTrust* [27].

3.4.4.1 GlobalTrust vs. K-Means Clustering-Based Judgment

Reidt *et al.* [32] introduced a *k-means clustering-based judgment scheme* (KMS-JS) on a trust overlay network. In KMS-JS, TA collects all LTOs to form LTO matrix O, in which $o_{i,j}$ represents the LTO of node i about node j. All $N \times N$ entries are assumed to be full after a sufficiently long time elapsed, where N is the number of nodes in the network. The LTOs over node j are placed in its column vector of the matrix O, $\mathbf{o_j} = (o_{1,j}, \ldots, o_{N,j})$. The values in column vectors of honest nodes tend to be close to each other and thus can often be clustered together. The judgment system uses a $N-1$ dimensional hyper-plane to maximally separate two clusters based on nodes' column vectors, and the larger cluster is categorized as honest. Unfortunately, the decision made may not be true, showing severe security vulnerability due to conflicting recommendation attacks. For example, a collusive community divides all honest nodes (i.e., nodes out of the community) into two groups equally, denoted as $\mathcal{G}_1$ and $\mathcal{G}_2$; collusive malicious nodes provide highest LTOs about themselves and honest nodes in $\mathcal{G}_1$, while they provide lowest LTOs about honest nodes in $\mathcal{G}_2$; also, malicious nodes control their attack intensity α in a proper level. This attack pattern tends to maximize the difference between

vectors $\mathbf{o_j}$ of two different honest groups and minimize the difference between malicious nodes and nodes in $\mathcal{G}_1$. Under this attack, the judgment system may cluster malicious nodes and nodes $\mathcal{G}_1$ into the honest class while nodes in $\mathcal{G}_2$ are clustered into the malicious class. Fig. 3.7 shows how detection error (FP and FN) varies with respect to α, when the ratio of malicious nodes k is 0.3. Fig. 3.7 shows that KMS-JS performs very poorly with FN close to 1 and FP close to 0.5 for $\alpha < 0.9$. In contrast, GlobalTrust performs significantly better than KMS-JS, with both FN and FP less than 0.1 in most cases when $\alpha > 0.1$.

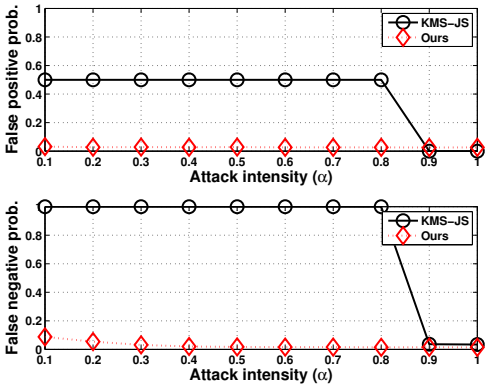


Figure 3.7. Comparison with KMS-JS

3.4.4.2 GlobalTrust vs. Existing Reputation Schemes

Here we compare GlobalTrust with the existing reputation schemes [27, 49, 50, 53, 54, 64] based on two criteria: consistency and resilience, shown in Table 3.3. In Fig. 3.8, we compare GlobalTrust with two reputation techniques used in *PeerTrust* [27], trust value based credibility measure (TVM) and personalized similarity measure (PSM), with respect to the accuracy of trust assessment.

Consistency: For fully distributed tactical networks, reputation evaluation is normally performed either in a distributed, cooperative way [49] or in an independent, uncooperative way [27]. In the former case, the evaluated reputation of a node must be consistent through the network. In the latter case, the evaluated reputation towards a node may be inconsistent in the network if an evaluator differentiates direct observations from indirect observations in deriving reputation values. Table 3.3 shows if existing reputation schemes have considered view consistency.

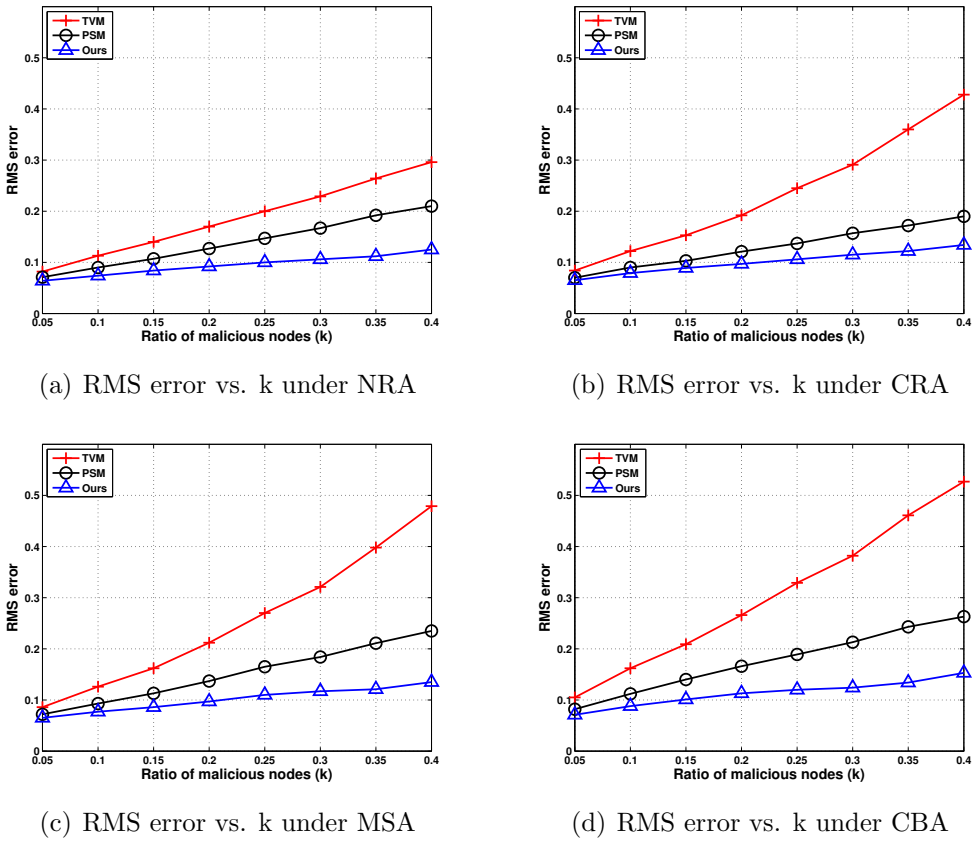
TBRM	Cons.	NMA	NRA	CRA	MSA	CBA
CORE	Yes	√	√	•	•	•
EigenTrust	Yes	√	√	⋆	•	⋆
SORI	No	√	√	•	•	•
Robust	No	√	√	√	⋆	•
PSM	No	√	√	√	⋆	√
PowerTrust	Yes	√	√	√	⋆	√
GlobalTrust	Yes	√	√	√	√	√
√: resilient; ⋆: partially vulnerable; •: vulnerable						

Table 3.3. Comparison between our GlobalTrust and existing TBRM schemes w.r.t. consistency and resilience

Resilience: We compare GlobalTrust with existing reputation schemes w.r.t. their resilience to the types of attacks in Table 3.3. *CORE* and *SORI* do not deal with collusion attacks such as CRA, MSA and CBA. *EigenTrust* is able to resist CRA to some extent with the help of pre-trusted nodes; however, for those nodes that the pre-trusted nodes have not had a chance to interact or observe (i.e., high uncertainty), the reputation evaluation would be highly distorted. Besides, MSA is an attack that can effectively defeat *EigenTrust* based on two reasons: (1) *EigenTrust* has no way to identify spies since their reputations are overestimated with high reputation values; and (2) false recommendations provided by spies are regarded as trustworthy information because the spy nodes do not show other abnormal behavior except passing false recommendations. *EigenTrust* may be vulnerable to CBA when pre-trusted nodes may be cheated by malicious nodes showing inconsistent behavior. *Robust*, *PSM* and *PowerTrust* devise trust models to effectively filter out false recommendations by collusion attacks; however, they do not consider credibility of recommendations for reputation evaluation and hence cannot identify spies in MSA. *Robust* is vulnerable to CBA with a malicious node showing inconsistent behavior because of the lack of capability to detect them by honest nodes. Since GlobalTrust can filter out false recommendations using the subjective reputation of nodes based on the identified trust quorum. In addition, GlobalTrust uses credibility reputation (CR) to consider credible recommendations that can help correctly measure global reputation, ultimately leading to effectively identifying spies in MSA.

Accuracy: In Fig. 3.8, we compare GlobalTrust with *PeerTrust* [27] w.r.t.

accuracy of trust assessment. We choose *PeerTrust* for the comparison because *PeerTrust* and GlobalTrust adopt the same definition of behavior reputation. The two evaluation models, TVM and PSM, in *PeerTrust* are devised based on different strategies to estimate recommendation credibility. TVM is known as vulnerable to collusion attacks while PSM is well designed to resist CRA. All parameters are set equally for these schemes in our simulation for fair comparison, as shown in Section 3.4.



(a) RMS error vs. k under NRA

(b) RMS error vs. k under CRA

(c) RMS error vs. k under MSA

(d) RMS error vs. k under CBA

Figure 3.8. Comparison with PeerTrust in accuracy

For PSM model, an honest node is randomly assigned as the evaluator to compute reputation-based trust values of all nodes. The evaluator's LTOs are *pre-trusted* in PSM when estimating the credibility of others' recommendations. We compare these three reputation evaluation methods (TVM, PSM and GlobalTrust) w.r.t. judgment accuracy under NRA, CRA, MSA and CBA. We use the root-mean-square (RMS) of the behavioral reputations of all nodes and the *actual* likelihood that all nodes behave honestly to measure reputation evaluation errors.

That is, we compare the behavioral reputations (BRs) in GlobalTrust with the reputation-based trust values in TVM and PSM since all of these values estimate the actual probability that all nodes behave honestly.

The actual behavioral reputation towards a malicious node's behavior is $1 - \alpha$ under NRA and CRA, 1 for spy and $1 - \alpha$ for non-spy under MSA, and $1 - \frac{\alpha}{2}$ under CBA. The actual reputation of an honest node's behavior is 1. Fig. 3.8 shows the results comparing GlobalTrust, TVM and PSM. We mainly observe the following trends:(1) TVM is severely vulnerable to collusion attacks including CRA, MSA and CBA as the RMS error has exceeded 0.4 when the ratio of malicious nodes, k, reaches 0.4; (2) GlobalTrust has about 0.25 to 0.4 lower RMS evaluation errors than PSM when k reaches 0.4 for each attack; and (3) PSM performs well, being resilient against NRA and CRA since the increased span of the RMS error is not significantly large (around 0.1) when k varies from 0.05 to 0.4. In contrast to PSM, GlobalTrust performs well in interpreting the behavioral reputation of a node under all these attacks as the maximum RMS error increases approximately up to 0.05. The results prove that GlobalTrust outperforms TVM and PSM in terms of the accuracy of trust assessment.

3.5 Summary

In this chapter, we proposed a trust-based reputation scheme, called GlobalTrust, to accurately evaluate the reputation of nodes with respect to both the behavioral trustworthiness and recommendation credibility in a tactical network environment in the presence of malicious entities and with no pre-trusted nodes in the network except for a commander node. Through the extensive simulation experiments, we compared GlobalTrust with other existing schemes and showed that GlobalTrust outperforms existing reputation schemes by highly being resilient against various types of attacks, maintaining high view consistency throughout the network and generating low reputation judgment errors.

Chapter 4 — DroidJust: Automated Functionality-Aware Privacy Leakage Analysis for Android Applications

Mobile devices, particularly smartphones and tablets, are becoming more and more prevalent in the world. While users enjoy the convenience and functions brought by smartphones and tablets, their privacy is severely threatened by malicious mobile apps that leak sensitive information to remote servers against users' intention. Based on the statistics from Genome [65] and Mobile-Sandbox [66], 55.8% and 59.7% Android malware families feature privacy leakage. Therefore, it is vital to have an effective approach for detecting such malicious apps.

Prior approaches to detecting privacy leakage on smartphones primarily focused on the discovery of sensitive information flows [5–13]. However, as more and more benign apps send out users' sensitive information for legitimate application functions, these approaches cannot easily justify the purposes of sensitive information transmissions in an app, and hence may not detect privacy leakage effectively. For example, Google Maps sends out users' location information to a remote server for driving navigation and location-based recommendation services. To continue to be effective and adapt to the growing application markets, the development of more advanced analysis approaches to detecting privacy leakage on smartphones is strongly desired.

In this chapter, we formulate the problem of sensitive information leakage as a justification problem, which aims to justify if a sensitive information transmission in an app serves any purpose, either for intended functions of the app itself or for

other related functions such as advertisements and analytics. To solve the justification problem, we propose an automated approach, called DroidJust. DroidJust not only identifies sensitive information flows but also tries to link each flow with certain application function to provide the evidence for justification. DroidJust uses various static taint analyses to automate the whole analysis process. We evaluate DroidJust on more than 6000 Google Play apps and more than 300 known malware collected from VirusTotal. Our experiments show that our tool can effectively and efficiently analyze Android apps for the purposes of their sensitive information flows, and hence can greatly assist in detecting privacy leakage.

4.1 Related Work

Most prior approaches to detecting privacy leakage in mobile apps use either static or dynamic analysis. TaintDroid [7] is a dynamic analysis tool for monitoring potential privacy leakage in Android apps by modifying Dalvik virtual machine and dynamically instrumenting Dalvik bytecode instructions. PiOS *et al.* [6] is a static analysis tool for discovering possible leaks of SI from a mobile device to third parties in iOS devices. Enck *et al.* [8] use ded [67], a re-targeting tool, to convert a Dalvik executable back to Java source code, and leverage a commercial Java source code static analysis tool named Fortify 360 [68] to detect suspicious information flow. AndroidLeaks [10] is another static analysis tool to detect potential privacy leakage in Android applications by leveraging the WALA [69] framework. Mann *et al.* [12] also proposed a static taint analysis based framework by using their self-crafted abstract Dalvik virtual machine instruction set and a security type system. Permlyzer [70] is a hybrid permission analysis tool which uses both dynamic and static analysis to identify the use of sensitive permissions. FlowDroid [5] is a precise context, flow, field, object-sensitive and lifecycle-aware static taint analysis tool to detect SI transmissions in Android apps. FlowDroid uses SuSi [71], a machine-learning approach to identifying an app's sensitive information sources and sinks. To summarize, all these approaches are capable of detecting an app's SI transmissions, but they are not designed to justify the SI transmission automatically.

Several approaches have focused on the justification of an app's SI transmission by examining the contextual information of the leakage. AppIntent [26] is an anal-

ysis tool to provide a human analyst with the contextual information of privacy data transmission, particularly, the chain of events leading to the triggering of a transmission, to help justify discovered SI transmissions. However, the approach still needs human effort to justify every discovered SI flow. Tripp *et al.*proposed a bayesian approach to statistically classify SI transmissions as legitimate or illegitimate based on the evidence arising at the release point [25]. The effectiveness of the approach highly depends on the select feature of the evidence for their statistical inference, which is the similarity between actual sensitive data and the data about to be released. Different to those two approaches, which consider the evidence arising before and at the release point, our approach uses the evidence arising after the release point for privacy leakage detection. Zhang *et al.*proposed Capper, a bytecode rewriting tool, to instrument Android apps to alert users on SI transmissions in runtime and enable users to allow/deny the transmission [72] . Market providers and antivirus vendors, however, cannot use the reactive approach to performing large-scale detection.

Past research work has also demonstrated the strong relationship between an app's meta information and its declared permissions. Pandita *et al.*and Qu *et al.*proposed WHYPER and AutoCog to automatically infer an app's necessary permissions from its description by using natural language processing [73, 74]. These approaches can be potentially used to provide additional useful information to justify an app's SI transmission. However, it is nearly impossible to only use meta information to justify an app's SI transmission because meta information is often very high-level, incomplete and sometimes inaccurate in reflecting all permission needs. Note that DroidJust is not designed for permission analysis, but rather a tool for sensitive information flow analysis. Because a privacy-sensitive permission might be needed in multiple sensitive information flows in an app, even if the purpose of a permission is justified, a dependent individual SI flow may still be unjustifiable. For example, a malicious weather forecast app may be justified for the location permission based on its description, but one of its SI flows stealthily sending to an unknown third party cannot be justified. This indicates that these tools and DroidJust work at different granularities and may complement each other.

4.2 Problem Statement and Design Goals

Recently, detecting privacy leaks in mobile apps has been one of the main research focuses on smartphone security, and it has led to development of many useful tools such as TaintDroid [7] for Android and PiOS [6] for iOS. Based on either static or dynamic taint analysis, such tools [5–13] can help discover potential sensitive information transmission. In a nutshell, these taint analysis approaches reduce the privacy leakage detection problem to the reachability problem. However, in reality, the existence of sensitive information transmission is *not* equal to privacy leakage, as real-world apps may send out users' sensitive information for their advertised functions. For example, a weather forecast app may send out users' location information to fetch the weather reports tailored to the locations; Google Maps also sends out GPS information for driving navigation. While these examples demonstrate obvious reasons for usage of users' sensitive information, there are also less obvious, sometimes even unpredictable, usage cases. For example, `com.pixeltech.imonline`, a trial Facebook messager app identified in our experiment, sends out users' Gmail addresses to a remote server for calculating the remaining trial days and then shows the number of days in the app. Judging this sort of sensitive information transmission is beyond the power of the conventional taint analysis approaches.

Realizing the fuzzy nature of the privacy leakage detection problem, prior research work has tackled the privacy leakage detection problem from different angles. For example, Yang *et al.* [26] proposed to use users' expectations as the indicator of privacy leakage. If the sensitive information transmission is expected by users, it will be considered as necessary, so not a leakage case; otherwise, if unexpected, it will be a privacy leakage case. However, *users' expectations are diverse*. For example, an advertisement library may send out a phone's geographic location for location-based advertisements. Depending on whether they like to receive targeted advertisements or not, different users may agree or disagree that disclosure of location information is expected in this context. Further, *we cannot assume all users are capable of comprehending system-level contextual information to provide their expectations*. The experiment in [26] has demonstrated that even security specialists had a discrepancy about the usage of device IDs in certain apps after they reviewed the generated event chains that lead to data transmission. This is

because app developers could potentially use a device ID, a phone number or even a Google account as a unique identifier of a device or a user, and such code-level information is often not available to the human specialists when they make decisions. Indeed, device IDs and phone numbers are the most common sensitive information that are delivered to the network [7, 65]. Different from the users' expectation angle, Tripp *et al.* [25] formulated the privacy-leakage detection problem as a machine learning problem based on certain features. Their approach, however, is probabilistic, and the effectiveness highly depends on the selected features and the training data sets.

In this work, we take a slightly different angle to tackle this privacy leakage detection problem. We formulate it as a *justification* problem, which aims to justify if a sensitive information transmission in an app serves any purpose, either for intended functions of the app itself or for other related functions such as advertisements and analytics. For example, if an app sends out the user's location to a remote server, later receives information from the server, and finally displays the information to the user in the phone screen, we consider this sensitive information transmission *justifiable*. On the contrary, if the app does not receive any information from the server after sending out users' location information, this sensitive information transmission is *unjustifiable*. Note that conceptually there are differences between *justifiability* and *privacy leakage*. According to our definition, a sensitive information transmission caused by an advertisement library is also justifiable because it does serve some known purpose, although privacy advocators may dislike it and consider it a privacy leakage. The merit of our formulation is that it separates technical issues from users' opinions. Rather than directly telling a user whether a sensitive information flow is a privacy leakage, we only report the purpose it serves, if any. The research problem now becomes more distinct and objective, and, therefore, more feasible to solve than before.

Following the formulation above, we aim to design an approach to justifying an app's sensitive information transmission. Specifically, we want to achieve the following design goals.

- **Fully automated analysis.** The proposed approach must be able to automatically justify an app's sensitive information transmission. The purpose is to minimize the involvement of human analysts in the middle. This task is challenging because it requires automatically extracting and understanding the

contextual information in order to bridge the gap between an app's sensitive information transmissions and its functions.

- **Complete and precise coverage.** Our approach needs to precisely cover almost all (if not all) users' sensitive information, restricted by the sensitive permissions of our interest. This is non-trivial due to the incomplete Android documentation and diverse permission enforcement mechanisms in Android.

- **High accuracy and scalability.** Our approach should minimize the inaccuracy incurred by possible under- and over-approximation during our implementation. Besides, our technique must be efficient for analyzing real-world apps at a large scale.

4.3 Approach Overview

In the section, we describe the rationale behind our justification approach and its workflow and then present an example to illustrate how our approach justifies the sensitive information transmission through a real-world app.

4.3.1 Design Rationale

The key to solving the justification problem is to identify if an app's sensitive information transmission could be used to fulfill some app function. To start, we study *how an app provides functions to mobile phone users*. We realize that the functions of a mobile app in smartphones are experienced by users during their interactions with the app. During the interactions, users are prompted by the changes of sensible phone states (SPS) (e.g., display, sound, vibration and light). Here sensible phones states are defined as phone output events that can be directly sensed by phone users. *In other words, app functions are provided to users via SPS*. Without leading to any *SPS directly or indirectly*, the function of an app is not meaningful to phone users as it cannot be experienced by users. Hence, if a sensitive information transmission cannot cause the change of any SPS, we consider it unnecessary and hence unjustifiable. Otherwise, we consider it justifiable. Note that in the PC world, similar rationale has been adopted by Privacy Oracle [75] and TightLip [76] to detect sensitive information leakage by

third-party apps. Their ideas are to apply black-box based differential testing to identify the existence of sensitive user inputs in outbound network traffic by mapping the discrepancy in output network traffic to different inputs.

Figure 4.1 shows our overall workflow, which answers *how an app's sensitive information transmission is used to provide functions to users* by linking users' sensitive information (SI) with app functions in terms of SPS. In the figure, SI is first read (often further transformed) and delivered to a remote server for computing or other purposes. This is an outbound information flow. Then, if a response from the server is received by the app and ultimately used to change some SPS *directly or indirectly*, we call this inbound information flow *sensible information reception (SIR)*. If an inbound information flow is not a SIR, it will not be sensed by the phone user, so we will not use it to justify any SI transmission. Note that without this rule, an attacker may easily introduce random inbound information flows to justify illegal SI transmissions. We will discuss this problem again in our Security Analysis section. Finally, once we have all the information flows of interest, we want to link inbound and outbound information flows. If a SI transmission cannot be linked to any SIR, it is unjustifiable; otherwise, it is justifiable.

From Figure 4.1, we can see that the app's sensitive information may also be consumed *locally* to change some SPS. Such local flow information could be useful for other analysis purpose, but for this work, we do not study it. Last, from our formalization, we can see a limitation of our approach: it cannot justify sensitive information flows in some service apps, which run in the background and have no user interface at all.

4.3.2 An Motivating Example

We present a motivating example to elaborate our proposed analysis approach, as shown in Figure 4.2.

`com.inspireadart.niceweather` is a popular weather forecast app in Google Play app store. We start from analyzing the app's SI transmission. In a discovered SI flow, the phone location information is first read, transformed to the locality information and returned from the `getLocation` method. Then, the locality information is passed to the `getForecastWeatherData` method in a background task

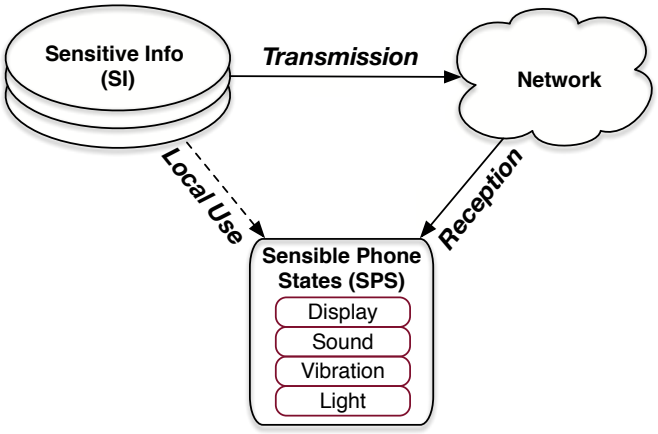


Figure 4.1. Design workflow: linking sensitive information with app functions

(`doInBackground`). Further, the locality information is put into a URL string, and the URL is finally used to open an HTTP connection after a few manipulations in the `getForecastWeatherData` method.

Next, we analyze the app's sensible information reception SIR. In the discovered SIR flow, the received information is first read from a HTTP connection and returned after a few manipulations from the `getForecastWeatherData` method. Then, the returned information is passed to the `getForecastWeather` method for parsing and the method returns a `WeatherForecast` object in a background task (`doInBackground`). Further, the background task returns the `WeatherForecast` object, which is in turn passed to the `onPostExecute` method as its parameter. In the `onPostExecute` method, the `WeatherForecast` object is passed to the `updateScreen` method. Finally, the extracted information from the `WeatherForecast` object flows into the framework API `setText` to change the text of a `TextView` in the `updateScreen` method. We note that here we show only one of the discovered information flows from the `WeatherForecast` object to SPS due to space limit. In reality, the extracted information from the `WeatherForecast` object also flows into several other framework APIs such as `setImageDrawable` and `setBackgroundResource` to change SPS. We also note that the intermediate representation in each message box in the figure shows an information flow in a method; hence, adjacent lines in a message box may not be adjacent in actual bytecode.

Last but not least, we can see that the discovered SIR is correlated to the

Figure 4.2. A motivating example with an Android app

discovered *SI* transmission because they use the exact same network connection (`<HttpURLConnection: void connect()>`). Therefore, through our two-stage information flow analysis, we conclude that this *SI* transmission is used to fulfill app's functions, and hence *justifiable*.

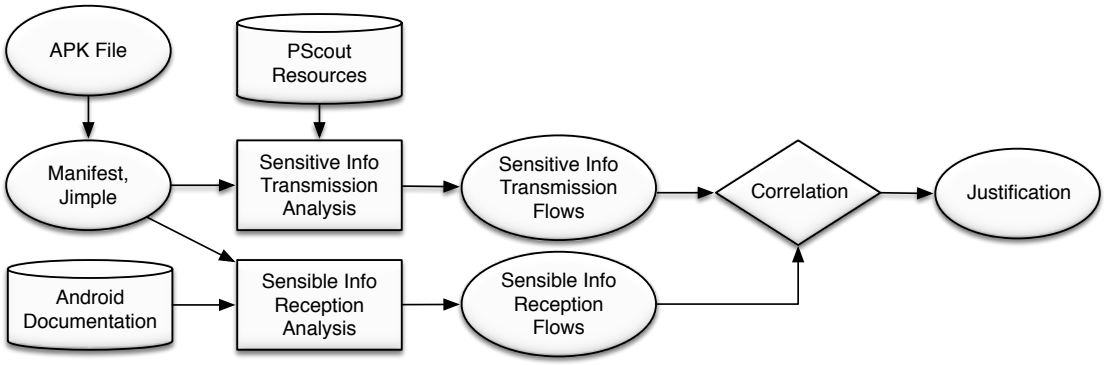


Figure 4.3. Overall Architecture of DroidJust

4.4 DroidJust: Overview and System Design

This section starts with an overview of the DroidJust's system design, and then describes its details.

4.4.1 Overview

Figure 5.1 depicts the overall architecture of DroidJust to justify an app's *SI* transmission. It takes the following major steps.

1. **Preprocessing.** An Android apk file consists of a Dalvik executable file, manifest files, native libraries, and resources. In this step, DroidJust decomposes the apk file and transforms the Dalvik bytecode executable file into the Jimple representation, which is a typed-3 address intermediate representation suitable for analysis and optimization on the Soot framework.
2. **Sensitive information transmission analysis.** In this step, DroidJust searches the app for *SI* flows by parsing the permission specifications from PScout [28] and the outgoing channels where the *SI* flows can reach, and using static taint analysis to identify the *SI* transmission from the *SI* (as sources) to the outgoing channels (as sinks).

3. **Sensible information reception analysis.** In the step, DroidJust searches the app for inbound network flows and the framework APIs that can change SPS. This is done by parsing the Android documentation, and using static taint analysis to identify the SIR, with inbound network flows as sources and the framework APIs that can change SPS as sinks.
4. **Correlation and justification.** After the SI transmission and SIR analysis, DroidJust correlates the identified transmissions and reception flows in an attempt to justify all the SI transmissions, and finally determines if a SI transmission is justifiable.

4.4.2 Sensitive Information Transmission Analysis

In the subsection, we define the users' SI and show how DroidJust identifies them as taint sources. In addition, we show how to identify the outgoing channels as taint sinks. The actual static taint analysis process will be explained in Section 4.4.4.

4.4.2.1 Sources

Sensitive information. There are many kinds of SI in Android apps, and currently DroidJust covers ten types of SI: phone information (such as device ID and phone number), contacts, messages, user profile, location information, social stream data, calendar information, user accounts, call logs, and browsing history and bookmarks. Android uses security permissions to restrict apps to access SI. Particularly, there are 12 Android permissions corresponding to the ten types of SI. Except for messages and location information, each type of the aforementioned SI is protected by one permission for *read* access. For example, `READ_PHONE_STATE` corresponds to phone information and `READ_CONTACTS` corresponds to contacts. For messages and location information, there are two permissions for each. `READ_SMS` and `RECEIVE_SMS` grant *read* access to SMS messages while `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION` grant *read* access to phone's location information.

Label actual sensitive data. Researchers have proposed many tools to identify sensitive data based on Android permissions [10,12,77–81]. However, it is still a big challenge to discover all possible sensitive data sources due to either the incomplete Android documentation or the diverse permission enforcement mechanisms

in Android. For better coverage, DroidJust utilizes the permission specification of PScout [28] to identify most (if not all) SI sources related to the above permissions. PScout is known as a tool that extracts a relatively complete permission specification from Android. In particular, there are three types of permission-related resources in PScout. The first type (T1) is documented and undocumented framework APIs that retrieve SI through returns or callbacks. The second type (T2) is privileged intent actions, which are associated with `IntentFilter` and `BroadcastReceiver` classes to request SI. The third type (T3) is URI fields and strings that are identifiers of content providers that manage SI in Android.

For T1 (i.e., framework APIs), a major challenge is to map it to actual sensitive data, given that not all return types are used to store actual sensitive data and the callbacks to retrieve sensitive data are very diverse. To overcome the challenge, we handle those framework APIs in the following manner. First, we filter out the non-return APIs and then, based on the Android source code, we manually check the remaining APIs to identify the return types that can be used to store actual sensitive data. As a result, we get a list of 39 return types (of which 21 return types are unique) for the 12 permissions, and we find that 575 framework APIs directly return actual sensitive data for the 12 permissions in Android 4.1.1.

Further, to identify all the possible callbacks in those APIs for retrieving actual sensitive data, we need to get an exhaustive list of the APIs that can retrieve the actual sensitive data through callbacks, as well as the mapping from the APIs to the callback classes, methods (i.e., handlers) and parameters. In practice, we use an automatic filtering method to identify the APIs' parameters that belong to or inherit from the following types: `Receiver`, `Listener`, `Callback`, `PendingIntent` and `Binder`, and then manually check their definitions to find the right callback methods and parameters based on the Android source code. Finally, we obtain a list of 254 framework APIs. By utilizing the refined framework APIs (575+254) and additional mapping information, DroidJust can label the actual sensitive data invoked by the framework APIs in PScout resources more accurately and completely.

Registering a `BroadcastReceiver` with an `IntentFilter` is another way to retrieve SI in Android. Hence, the second type of PScout resources (T2) is privileged intent actions that can be added by an `IntentFilter` to gain access to the corresponding SI. For example, `"android.provider.Telephony.SMS_`

RECEIVED" is a privileged intent action to receive all incoming SMS messages, and a BroadcastReceiver class can acquire the incoming messages by registration with an IntentFilter including the intent action. Particularly, an intent containing the incoming message is passed to the onReceive method of the registered BroadcastReceiver.

In Android, there are two ways to register a BroadcastReceiver with an IntentFilter. One way is to register in manifest files statically. In this case, DroidJust parses manifest files to identify the BroadcastReceiver classes that are registered with an IntentFilter, and then label the Intent parameters of their onReceive methods as the actual sensitive data. The second way is to register a BroadcastReceiver dynamically. Particularly, the app code can call the method registerReceiver at runtime to register a BroadcastReceiver with an IntentFilter. In this case, DroidJust searches the app for the strings that are equal to the privileged intent actions, then performs *static taint analysis* from the strings (as sources) to registerReceiver methods (as sinks) to identify the BroadcastReceiver classes that can receive the SI, and finally label the Intent parameters of their and their subclasses' onReceive methods as actual sensitive data.

The third type of resources (T3) in PScout is related to content providers. In Android, content providers also manage access to certain SI. To retrieve the referenced SI, app developers can use a ContentResolver object to resolve a content Uri object by calling its query method. Specifically, there are two ways to obtain a content Uri object in Android. One way is to directly construct it by encoding a string and the other way is to directly fetch a content Uri object from the field of a framework class. For example, constructing a Uri object by encoding the string "content://com.android.contacts" gives the exact same Uri object as android.provider.ContactsContract.AUTHORITY_URI gives. Hence, T3 is those strings and fields that can be used to construct or directly fetch Uri objects to retrieve SI. DroidJust searches an app for the Uri objects that are constructed by the strings or directly fetched from the fields, then performs *static taint analysis* from the found Uri objects (as sources) to the query method of the ContentResolver object (as sinks). It finally labels the result of the query (a Cursor type) as actual sensitive data.

4.4.2.2 Sinks

The retrieved SI can directly flow to the outer world through several channels. Below we describe two most common channels, which are currently covered in DroidJust.

Internet. Android apps can access the Internet and deliver SI in several ways. A common way is to employ a socket-like API or a high-level HTTP client to send out SI. We collect all such APIs from `java.net`, `javax.net` and `org.apache.http` packages. Besides, Android apps may embed SI into a URL and use the Android webkit APIs such as `<WebView: void loadUrl(String URL)>` to deliver SI to the network. Hence, we also collect the related framework APIs from the `android.webkit` package as potential sinks.

SMS. SMS is another popular channel to deliver users' SI, especially for malware. App developers can use the framework APIs in `SmsManager` package to send a message. Hence, we collect a list of the framework APIs from the `SmsManager` package as sinks.

4.4.3 Sensible Information Reception Analysis

Next, we identify both inbound information flows and sensible phone states (SPS) by parsing the Android documentation. We will delay the description on static taint analysis from the inbound information (as sources) to the SPS (as sinks) in Section 4.4.4.

4.4.3.1 Sources

Corresponding to the two types of sinks for outbound SI transmissions, we also consider inbound information flows from these two channels: the Internet and SMS.

Internet. Android apps can receive information from the network by employing a socket-like API or a high-level HTTP client. We collect a list of the related framework APIs from the `java.net`, `javax.net` and `org.apache.http` packages to identify the sources. Besides, Android apps can receive network data by calling the Android webkit APIs. We collect the related framework APIs from the `android.webkit` package to identify the sources.

SMS. We consider the incoming text messages as another source of SIR in our work. Android apps receive incoming text messages by registering a `BroardcastReceiver` with the intent action `android.provider.Telephony.SMS_RECEIVED`. To cover the source, we label the corresponding `onReceive` methods and identify the incoming `Intent` parameters as sources (as shown in 4.4.2.1).

4.4.3.2 Sinks

Android apps use framework APIs to change the SPS. For example, `<TextView: void setText(CharSequence)>` is a framework API to change the display of a text editor widget; `<Vibrator: void vibrate(long)>` is used to cause the phone to vibrate. We collect the framework APIs that can change the SPS in four different ways, including display, sound, vibration, and light, by parsing the Android 4.1.1 documentation. In general, many framework APIs can change the SPS via display. Our selection strategy for this type of APIs is to first label all the subclasses of `android.view.View`, because this class represents the most basic building block for UI components in Android. We then manually identify the methods that can change SPS by checking their functions in the Android documentation. Based on our observation, most APIs that can change SPS have a prefix of "set" in their method names. For sound, vibration and light, they have much fewer framework APIs than the display-related APIs. Hence, we manually find the related classes to collect their methods that can change SPS. Finally, we collect totally 249 Android framework APIs that can change SPS. Table 4.1 gives a summary of our collected Android framework APIs that are able to change SPS.

Type	Method Name	Quantity
display	setText, setTitle, setIcon, etc.	232
sound	setDataSource, setSound, etc.	11
vibration	setVibrate, vibrate	4
light	setLights	2

Table 4.1. Android framework APIs that are able to change sensible phone states

4.4.4 Static Taint Analysis

To identify the data flows from different kinds of sources to different kinds of sinks, DroidJust uses static taint analysis intensively. Specifically, we have

the following static taint analysis tasks: 1) from an intent action string to a `registerReceiver` method (in Section 4.4.2.1), 2) from a `Uri` object to a `query` method (in Section 4.4.2.1), 3) from the actual sensitive data to the outgoing channels (in Section 4.4.2), 4) from the inbound information to the SPS (in Section 4.4.3), and 5) from a URL string to network socket or a high-level HTTP client (in Section 4.4.5).

DroidJust models the static taint analysis problem within the IFDS [82] framework for inter-procedural distributive subset problems. In practice, DroidJust extends Soot [83], Heros [84] and FlowDroid [5] to provide inter-procedural data-flow analysis. Particularly, FlowDroid generates a dummy main method based on a precise modeling of Android lifecycle and flow functions, which define an IFDS analysis problem; Soot generates a call graph and an inter-procedural control-flow graph (ICFG) from the dummy main method; Heros provides template-driven inter-procedural data-flow analysis by taking as the input flow functions and the ICFG; and DroidJust identifies different kinds of sources and sinks for the inter-procedural data-flow analysis and supports additional indirect static taint analysis (as described below).

Additional Indirect Static Taint Analysis. In practice, we find that the state-of-the-art static taint analysis is ineffective to discover a significant amount of data flows, particularly in the aforementioned tasks 3) and 4), due to the heavy use of data medium in Android apps. That is, tainted data could be first stored into a data medium and later delivered to a sink through data medium. This is very common in Android development since app developers prefer to use data media (e.g., SQLite) as the backend of displayed content. To handle this challenge, DroidJust performs additional *indirect* data flow analysis at two stages: first from sources to the data media, and then from the tainted data media to sinks. In general, there are four types of data media in Android: SharedPreference, ContentProvider, SQLite database and File. Each type of data medium has its own *unique identifier*, and DroidJust taints data media at two stages according to the *unique identifier*. Specifically, SharedPreference uses both `Context` and a filename (a string) to uniquely identify a preference file; ContentProvider uses `Uri` to uniquely identify a data repository; SQLite uses a table name (a string) to identify a table on a default database; and File uses a filename (a string) to identify a stored file. By launching the two-stage static taint analysis, DroidJust is able to

discover almost all data flows.

4.4.5 Correlation and Justification

After identifying the app's SI transmissions and SIR, DroidJust tries to justify each of the SI transmission flows by linking it to an SIR flow. This correlation task is not easy since DroidJust cannot acquire and analyze the server-side logic. To try the best, DroidJust solves it in the following manner. SI transmission flows deliver the sensitive information either via the Internet or SMS. Considering a SI transmission flow delivering the sensitive information via the Internet, if the transmission flows into the Android webkit APIs, it is justifiable since the transmission displays a `WebView` to users in phone screen. Otherwise, it means the transmission flows into a socket-like API or a high-level HTTP client.

In the latter case, DroidJust first finds if the transmission is *synchronous* to any SPS flow. Specifically, DroidJust checks if the SI transmission flow and the SIR flow share the same network socket or HTTP client. If true, they are correlated. Otherwise, DroidJust continues to check if a SI transmission flow is *asynchronous* to any SIR flow. More specifically, DroidJust checks if the destination of the SI transmission and the source of the SIR are the same. In other words, DroidJust checks if the network server that delivers information to the SPS is the same server where the SI flow goes to.

There are two tasks to map the server names. The first task is to extract the network addresses from each of the inbound and outbound information flow. DroidJust first identifies all the URL- or IP-like strings and then uses *static taint analysis* to find the strings that flow into the network connection of the SI transmission or SIR. The identified strings are the network addresses of the SI transmission or reception. The second task is to check if these network addresses refer to the same network server. The simplest way is to compare the hostnames of the network addresses. However, in reality, an app may use different hostnames for the same server. To cope with this situation, we further check whether the IP addresses of the hostnames are equal. Finally, DroidJust justifies the SI transmission if either the hostnames or the IP addresses of the hostnames are the same. In its implementation, DroidJust uses the standard Java API (`<URI: String getHost()>`) to extract hostnames from URL- or IP-like strings and `nslookup` to

resolve hostnames to IP addresses.

On the other hand, considering a SI transmission is through SMS, DroidJust simply checks if the app receives any incoming messages to change SPS. If yes, the transmission is justifiable; otherwise, the transmission is unjustifiable.

4.4.6 Correlation

After identifying the app's sensitive information transmissions (SITs) and sensible information receptions (SIRs) by using static taint analysis, DroidJust tries to justify each of the sensitive information transmission flows by linking it to a sensitive information reception flow. This correlation task is not easy since DroidJust cannot acquire and analyze the server-side logic. To overcome the challenge, we propose to employ dynamic analysis approaches to collect execution traces for our correlation tasks. In particular, we use a testing tool, named MonkeyRunner, to generate and send user events to automate app's execution. At the same time, we start profiling, a functionality of the activity manager (am) in adb, on the app's running process, to collect execution call-stacks. By the end, we collect a few call-stack log files containing many method invocations with their timestamp information. Then, DroidJust identifies both the SIT flows and the SIR flows in the call-stack log files and correlates them based on their timing difference. Note that a specific SIT flow or a SIR flow could occur in multiple times. To justify an identified SIT flow, we try to validate whether there is always at least one SIR flow occurring after the SIT flows in a pre-defined time limit. If yes, the SIT flow is justified; otherwise, it is unjustified. We use a supervised machine learning approach to train the time limit since the parameter is critical to the accuracy of the correlation task.

4.5 Experimental Evaluation

To evaluate the effectiveness, accuracy and efficiency of DroidJust, we perform experiments on 6111 Google Play apps and 340 known SI-stealing Android malware collected from VirusTotal [85]. Next we report the detailed results and our findings.

Package Name	Leaked Sensitive Info	Dynamic Tainting [1]	Still on Google Play? [2]	VirusTotal Score [2]
com.controlaltkill.autoball	Phone number	No	Yes	0/57
com.jb.gosms.pctheme.loving_bears	IMEI	Yes	Yes	0/57
com.kingdom_card_wcm777_1	Location	No	Yes	0/57
com.kokovoin.homedesign	IMEI	No	Yes	1/56
com.mapnavigation	IMEI	Yes	Yes	12/57
com.necta.aircall_accept.free	IMEI	Yes	Yes	7/57
com.pixoplay.candyshooter	IMEI	Yes	Yes	0/57
com.topdisk.launcher	IMEI	N/A [3]	Yes	0/57
cz.prilozany.android.compass	IMEI, Location	Partial [4]	Yes	0/57
fr.pb.tvmobile	IMEI	No	No	0/56
lk.bhasha.sett.hindi	IMEI	Yes	Yes	3/57
lk.bhasha.vishwa	IMEI	No	No	0/57
me.chatcast.kaomoji	Gmail account name	No	Yes	0/57
me.zed_0xff.android.alchemy	IMEI	Yes	Yes	2/57
mx.websec.mac2wepkey.hhg5xx	Location	No	Yes	0/57

Table 4.2. (C3) Identified Google play apps that send out users' sensitive information not for functions. Notes: 1. we use Andrubis for dynamic taint analysis, whose dynamic taint analysis is based on TaintDroid; 2. we update till Feb. 13, 2015; 3. the file exceeds the maximum size limit (8MB) restricted by Andrubis; 4. only the IMEI leak is identified.

4.5.1 Evaluation on Google Play Apps

In the experiment, we evaluate DroidJust over 6111 apps, randomly downloaded from the Google play store during March 2014. Based on a report from Andrubis [86], only 1.6% Google Play apps are identified as malware by anti-virus vendors. Hence, we expect most of these downloaded apps to be benign and not leak user privacy. By scanning these apps with DroidJust, we evaluate whether DroidJust can precisely identify benign apps, particularly those delivering SI to the network.

We setup our evaluation on a cluster with hundreds of Intel Xeon E5-2665 2.40 GHz processors (16 cores per processor). Each analysis task (for analyzing an app) is assigned to a cluster node with 4 cores and 16 GB physical memory, which runs JDK 1.7.0_21. DroidJust takes about 85 hours of CPU time to analyze 6111 Google play apps and 12 hours of CPU time to analyze 340 known malware. On average, each Google play app takes about 50 seconds of CPU time and each known malware about 128 seconds of CPU time. Hence, DroidJust is definitely an affordable tool for antivirus vendors or Android market operators.

During the evaluation, we notice that DroidJust cannot analyze some apps due to either insufficient memory or failure of type resolving. Basically, DroidJust shares the same problem with other FlowDroid-dependent tools [87]. We start by analyzing 6111 apps, among which 1092 apps failed to go through. Thus,

below we show our experimental results over the remaining 5019 apps.

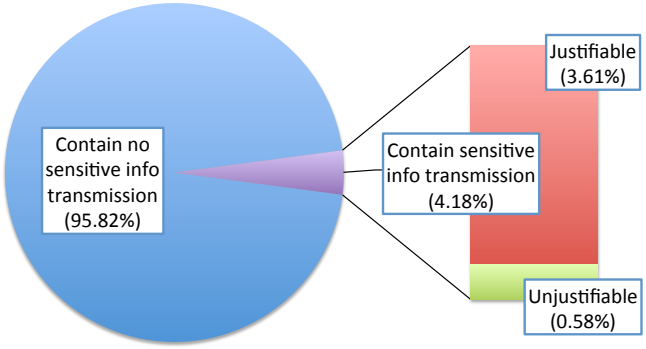


Figure 4.4. Analysis results for Google Play apps

Results. Figure 4.4 illustrates the analysis results. Among 5019 Google play apps, 95.82% apps do not send out users' SI while 4.18% (210) apps transmits users' SI via Internet or SMS. Among those transmitting SI, 3.61% (181) apps' SI transmissions are justifiable while 0.58% (29) apps' SI transmissions are unjustifiable.

Validation. We manually check these 29 apps by analyzing their intermediate representation (Jimple) and read their descriptions in Google Play to understand their functions and validate our detection results. After validation, we classify these 29 apps into the following categories.

(C1) Stealthily send out SI for app functions. There are ten apps that stealthily send out users' SI for application functions including anti-theft, location-tracking and spying. Those apps have clear descriptions about their stealthy behavior in Google Play. For anti-theft apps (e.g., `avsolution.version1`), once a thief changes the SIM card of a stolen phone, they immediately notify the original users of phone information (e.g., phone number, IMEI, and location) of the new SIM card via SMS in a background task. This communication is designed to be one-way. Location-tracking apps (e.g., `gaugler.backitude`) send out locations via Internet or SMS to track a mobile device in real-time. Besides, there are spyware apps whose main function is to spy on users, as stated in their descriptions. For example, `com.dona.messagespoofing` is an intentionally designed spyware to stealthily forward all incoming SMS messages to a designated phone number after the first-time setup. In summary, those apps are supposed to stealthily send

out SI without users' awareness. In terms of behavior, they are very similar to spyware that steal users' sensitive information. Hence, it is indeed appropriate for DroidJust to label them as unjustifiable from the perspective of code behavior. We note that privacy analysts can easily distinguish all these apps from malicious spyware by reading their descriptions.

(C2) Analytics libraries. Four apps containing analytics libraries are found sending out users' phone information to remote servers. We recognize these analytics libraries by checking the hostnames of their network servers. They have affiliations with the mobile application solution providers including *accelerator*, *crittercism* and *kontagent*. DroidJust does not justify the SI transmissions by these analytics libraries because they do not provide any function back to users. We note that, in practice, an analytics library is often bundled with the same provider's advertisement library, and therefore, our tool will justify the sensitive information transmission in the analytics library by identifying the SIR in the advertisement library.

(C3) Stealthily send out SI but not for app functions. We identify 15 apps that stealthily send out users' SI to remote servers, but such SI transmissions cannot be justified. Those apps do not describe anything about their stealthy behavior in their descriptions in Google Play. To avoid possible false alarms, we manually and carefully check their app logic by analyzing their intermediate representation (Jimple) to ensure that the discovered SI transmission does not provide any function to users. Besides, we use the dynamic taint analysis tool Andrubis to expose their privacy leak behavior in runtime.

Table 4.2 shows a summarized result for these 15 apps. From left to right, the table shows app's package name, leaked SI identified by DroidJust, if Andrubis can identify the same leak if the app is still available for download on Google Play, and how many antivirus engines in VirusTotal identify the app as malicious. From this table, our observations and findings are as follows. First, we can see that the most frequently leaked SI is device ID (IMEI). This is expected based on the past research work [7,65]. Second, Andrubis's dynamic taint analysis did not identify all the SI transmissions in 8 apps, mostly due to the failure in generating appropriate inputs. For example, `com.kingdom_card_wcm777_1` and `lk.bhasha.vishwa` require users to provide correct authentication information for a mobile payment account and the Facebook account, respectively, at the very beginning, and dynamic taint

analysis fails to bypass the authentication in runtime. Third, after almost one year, Google Play only removed two apps while the remaining 13 apps are still available for download. Last but not least, 10 of these 15 apps cannot be detected by any antivirus engines in VirusTotal.

4.5.2 Evaluation on Known Malware

In this experiment, we evaluate DroidJust on 340 malware known for stealing users' SI. To collect them, we first collect a list of malware families that are known for stealing user private information from Genome [65] and Forensics blog [66], and then download the apps related to these malware families from VirusTotal by using their advanced reverse search system [85]. We run DroidJust against these 340 apps to evaluate its detection precision. We start with 340 apps and unfortunately 42 apps fail to go through due to the same reason as we mentioned above. Thus, here we only show our experimental results for the rest 298 apps.

Results. Table 4.3 shows the analysis results for each malware family. There are 43 malware families in total. For each malware family, the number of samples is between 1 and 31. In the table, the column *Positive* gives the number of samples that are identified to contain unjustifiable SI transmission and column *Negative* gives the number of samples that are identified to contain only justifiable SI transmissions. The total number of positive outcomes is 274 while the total number of negative outcomes is 24. Thus, the detection rate is 91.94%. The malware samples leak five kinds of SI, as shown in the middle of the table: phone information (P), contacts (C), messages (S), locations (L) and accounts (A). We note that the marks in each row indicate the union of sensitive information types that are leaked by all samples in the malware family; as such, not every sample in a malware family leaks all marked types. We can see that phone information is the most leaked SI among the 43 malware families.

Validation. We manually inspect the 24 apps with negative outcomes by analyzing their intermediate representation. There are two main sources of false negatives for DroidJust. One is because of dynamic code loading. That is, some malicious apps download and install other malware after exploiting a certain root access vulnerability. It is the dynamically downloaded malware code that leaks user privacy. Static taint analysis inherently cannot detect the leakage behavior by the

Malware familiy	Sensitive information					Pos.	Neg.
	P	S	L	C	A		
anserver	x					30	1
avpass	x					1	0
backflash	x	x				2	0
basebridge	x	x				27	1
beanbot	x	x				6	0
bgserv	x	x				3	1
droiddreamlight	x	x		x	x	25	2
droidkungfu	x					20	2
extension	x					1	0
fakeangry	x					5	0
fakebank	x			x		8	0
fakemart		x				1	0
faketaobao		x				2	1
fjcon	x	x	x			4	0
fokonge	x					19	1
geinimi	x	x	x			17	1
ggtracker	x	x				6	1
gingermaster	x		x			16	3
godwon				x	x	3	1
golddream	x	x		x	x	29	0
hongtoutou	x	x				17	1
kmin	x	x				29	0
lena	x					2	0
loozfon	x			x		3	0
mobilespy	x	x		x		4	1
mobiletx	x					2	1
pjapps	x					22	1
plankton	x					12	1
roguelemon	x					1	0
roidsec		x	x	x		2	0
sinpon		x	x	x		2	0
skullkey	x					1	0
smspacem		x		x		1	0
sndapps	x				x	9	0
spitmo	x	x				3	1
spyoo	x					3	0
ssucl	x	x		x	x	1	0
tetus	x					1	0
typstu	x				x	9	0
usbcleaver	x					1	0
vdloader	x					1	0
yzhc	x					11	2
zitmo	x	x				2	1
Total (43)	37	20	5	10	6	274	24

Table 4.3. Malware families featuring privacy leakage
P: Phone information; C: Contacts; S: Messages;
L: Locations; A: Accounts.

malware installed later. The second reason is due to the inaccurate callback-based lifecycle modeling in FlowDroid [88].

4.6 Discussion

In this section, we discuss the limitations of our approach.

Security Analysis. In our design, we ignore the *meaningless* inbound flows (i.e., those not leading to any SPS) to prevent attackers from introducing noisy inbound flows to evade our detection. Determined attackers, however, may manage to introduce noisy inbound flows that indeed lead to some SPS. Since SPS is sensible to users, the attacker will need to ensure that such noisy flows will only cause minimal changes of phone states to not degrade the usability and functionality of the app. We will examine the practicality of this and other attacks and accordingly design possible countermeasures in our future work.

Implicit Information Flows. Sensitive information can propagate in other channels than direct channels, such as control flow and timing channels. It is very challenging to detect and track these channels. In this work, we do not consider tracking implicit information flows. The limitation is also shared by other taint analysis tools, such as TaintDroid [7] and PiOS [6]. We leave it as our future work to support the discovery of implicit information flows.

Java Reflection & Native Code. Static information flow analysis always has the trouble to handle Java reflection and native code due to the lack of full knowledge on Java reflective calls and JNI calls [77]. In our work, we use taint wrappers with various crafted function summaries to partially resolve the propagation through Java Reflection, which however may introduce some false positives. We do not deal with native code for data propagation. Potentially, we could model the well-known JNI calls and thereby create the corresponding taint wrappers for the calls to exercise static data propagation. We leave it as our future work to enhance our tool.

4.7 Summary

In this chapter, we present DroidJust, an automated approach to justifying an app's SI transmission by bridging the gap between SI transmission and app's func-

tionality. It uses static taint analysis to first discover the SI transmissions to the network, then discover the information receptions (from the network) that serve application functions, and finally justify the discovered SI information transmissions by correlating the outbound and inbound information flows. Our evaluation on real-world Android apps and known malware demonstrates that DroidJust can effectively and efficiently analyze both benign apps and malware.

Chapter 5 — SweetDroid: Calling-Context-Sensitive Privacy Policy Enforcement Framework for Android

Mobile devices such as smartphones and tablets have been an essential part of our modern life. However, at the same time, the large amount of sensitive information stored and processed on them, from device serial number, location to private messages, has raised many security and privacy concerns. Based on the summary from Mobile-Sandboxing [4], the user-privacy threatening malware families are prominent, which contribute to 63.1% of all malware families in 2014. The privacy threat has increased by 3% as compared to 2013 and by 13% as compared to 2012. To remedy against the privacy violations arising from mobile applications, the development of new approaches to protecting user privacy is extremely emergent.

Prior research effort on enhancing user privacy in mobile devices has mostly focused on permission model extension and enforcement [11, 14–24]. Those proposed solutions provided users with more flexible and fine-grained policy enforcement than the conventional Android's permission system, which is notorious for its all-or-nothing permission authorization during an app installation phase [16, 21]. For example, a number of solutions [15, 16, 18, 19, 21, 23, 24] enable users to turn off the permissions granted to the installed applications or intervene their access to sensitive resources based on user-defined security policies. In a similar fashion, the recent Android 6.0 introduces a new permission model that allows users to not grant any permissions during app installation. Instead, it shows a dialog to

users asking for a certain permission when needed at the first time [89], which is very similar to the iOS permission model. The new Android permission model also provides users a graphical user interface (GUI) to enable/disable the granted permissions after app's installation phase.

Most previous solutions, however, apply permissions to a whole app and hence may fail to prevent privacy violations, without considering and distinguishing the contextual information of the access to sensitive data. For example, users may grant a weather forecast app the permission to access the location data when the app needs to retrieve the weather forecast information tailored to the current location, but it is unclear whether the app may access the sensitive data in other contexts and whether the users will approve as well. To ensure the contextual integrity in accessing sensitive data, a simple solution is to force users to make a decision per sensitive data request. Nevertheless, there are two major weaknesses in practice. First, it may cause dialog fatigue since users have to permit every sensitive data request. Second, although there are clear examples of the use of sensitive data, there are also less obvious cases of the use of sensitive data where users may not be capable of making right decisions [25, 26]. For example, app developers could use device IDs, phone numbers or Google accounts as unique identifiers of devices or users for realizing application function, and such code-level information is not available to end users when they make decisions [90]. In a nutshell, *distinguishing the context of the access to sensitive data* and *reasoning about the use of sensitive data under the context* are two critical steps to enhancing user privacy yet retaining application usability for mobile applications.

In this chapter, we aim at addressing these challenges by proposing a calling-context-sensitive privacy policy enforcement framework, named SweetDroid. Extending the existing Android framework, SweetDroid generates calling-context-sensitive privacy policies targeting installed Android apps on Android devices, and enforces these privacy policies at the calling context level in runtime to effectively enhance user privacy yet retaining app's usability.

5.1 Related Work

We categorize the previous work about privacy protection and enhancement on smartphones into the following categories based on their primary mechanisms.

Permission Model Extension and Enforcement. Most past research works enhance user privacy by either modifying Android source code [16, 17, 21, 22, 91] or rewriting app code [11, 14, 18–20, 24] to extend and enforce Android permission model. Apex allows users to selectively grant permissions to applications and restrict the usage of resources [21]. Introducing a privacy mode, TISSA empowers users to flexibly control what kinds of personal information are accessible to an application [91]. CRePe can enforce fine-grained permission policies based on the contextual information of the mobile device such as time, location and user interaction [17]. Saint provides a security infrastructure that governs install-time permission assignment and enforce runtime application-centric security policies [22]. MockDroid modifies Android framework to allow users to mock an application's access to a resource [16].

I-ARM-Droid rewrites app's bytecode to interpose on the invocations of sensitive API methods in order to enforce desired security policies. [19]. RetroSkeleton is an app rewriting framework that supports retrofit of app's behaviors by statically and dynamically inception of method invocations [18]. AppGuard rewrites and repackages an app on the phone to mediate security-relevant methods [14]. Aurasium enforces user-defined policies by rewriting an app and low-level *libc.so* [24]. [20] provides fine-grained permissions on resource accessing by supporting parameterized permissions. AppFence enables users to feed shadow data to apps in place of data that users want to keep private and block data exfiltration [11]. Additionally, Boxify uses full-fledged app sandboxing to enforce security and privacy policies without modifying Android source code or app code [15].

Mandatory Access Control. Another way to enhance access control is to introduce Security Enhanced Linux (SELinux) [92], the most prominent mandatory access control solution, into Android. There are several research works supporting mandatory access control on both Android's middle and kernel layers [93, 94].

However, nearly all existing approaches in these two categories heavily rely on user-defined security or privacy policies to prevent privacy violations, which is not realistic for normal users since they lack professional knowledge on security. Besides, the contextual information used in the previous policies is too coarse-grained and/or heuristic-oriented that could interfere with user-desired functionality [11]. In contrast, the generated privacy policies by our framework distinguish sensitive API invocations in an app and handle them differently according to analysis re-

sults. It achieves the sweet point to enhance user privacy while retaining app function.

Permission Separation for Libraries. There are a few research works focusing on restricting permissions for one component [95] or a third-party library [96–98]. Compac distributes a narrowed set of permissions into every component and hence enforces access control at component level [95]. AdSplit extends Android to allow an application and its advertising to run as separate processes (under separate user-ids) in order to separate permission sets for different domains [98]. PEDAL uses a novel machine learning classifier to detect ad libraries and rewrite app's bytecode for privilege de-escalation [96]. AdDroid separates privileged advertising functionality from host applications, allowing apps to show advertisements without requesting privacy-sensitive permissions [97]. Though these approaches can restrict access control at component or library level to eliminate unnecessary sensitive data access (e.g., for advertisement), they cannot handle overly-curious or privacy-invasive apps because there is no clear boundary between the privacy-violating part and the user-defined function part.

5.2 Background and Example

In this section, we first describe our motivation by presenting an example and then explain the attack model. List 1 gives two snippets of code abstracted from a real-world Android application. The first snippet of code implements an activity of this application (`AppActivitiy`). The activity contains a private method `getUserProfile` to fetch users' profile from the application server by sending users' IMEI (returned by `getDeviceId()`) and put the profile data into a private field `mUserProfile` (line 9). The purpose is to frictionlessly authenticate a user [1] and fetch the user's app data remotely. This kind of IMEI usage is not rare in Android applications where the fetched app data could contain app record, game save, remaining trial days, etc. It could avoid local data loss caused accidentally (e.g., reset system for rescue) or intentionally (e.g., renew a trial app). The `AppActivity`, in the example, uses `UserProfile` to interact with users and accomplish its function; hence, impeding the IMEI retrieval could disable the application function or harm user experience. The second snippet of code, as a part of a third-party library,

[1] It actually authenticates a device rather than a user since a user may have multiple devices.

```
1  public class AppActivity extends Activity {
2      private UserProfile mUserProfile = null;
3      private void getUserProfile(){
4          TelephonyManager tm = (TelephonyManager)
5            getSystemService(Context.TELEPHONY_SERVICE);
6          String imei = tm.getDeviceId();
7          if (imei != null)
8              // get users' profile from the App server
9              mUserProfile = fetchUserProfile(imei);
10     }
11 }
12
13 public class LibraryClass extends Service{
14      public static void sendImei(){
15         TelephonyManager tm = (TelephonyManager)
16           getSystemService(Context.TELEPHONY_SERVICE);
17         String imei = tm.getDeviceId();
18         // harvest users' IMEI to their servers
19         sendToURL(imei, "http://www.foo.com");
20      }
21      @Override
22      public void create(){
23         sendImei();
24      }
25 }
```

Listing 1: Example Android Application

implements a public method `sendImei` to send out users' IMEI to a remote server in an Android service (line 19). It invokes the method when the service is created (line 23). The method is used by a third-party library to harvest users' sensitive information possibly for the purpose of statistics, and this kind of sensitive information does not serve application function and thus should be prevented.

To avoid such unnecessary sensitive information leak and enforce the principle of least privilege, an effective privacy policy enforcement scheme must be context sensitive: the contexts of the sensitive data access in the two snippets of code should be distinguished and the latter one should be considered as a privacy violation and hence prevented. The current Android permission system (up to Android 6.0) and existing policy enforcement approaches are agnostic to the exact calling context

within the app process, which means that their permission enforcement applies equally to all code (including Java and Native code) executing under the app's assigned UID.

Attack Model. SweetDroid can enforce privacy policies for different calling contexts of sensitive information access. We assume the following attack model in this work. The attacker supplies an untrusted app with arbitrary malicious Dalvik bytecode. The attacker's goal is to leak users' private data through a few dangerous communication channels such as the Internet and SMS. We assume that the Android OS is trusted, including the Linux kernel and the Android framework. That is, we assume that an application cannot compromise the integrity of kernel or Android framework. We assume that the attacker has no way of circumventing the security mechanism of the Android platform or exploiting system vulnerabilities to gain excessive privileges (e.g., root privilege).

Moreover, we assume that the untrusted app could be a repackaged version of a legitimate app to take advantage of the permissions declared by the original app for malicious purposes. We assume that the untrusted app has full control over its process and the associated memory address space. Moreover, the app's code and thus the app's behavior may be self-modified at runtime through techniques like native code or Java reflection.

5.3 SweetDroid Architecture

In this section, we present the design and implementation of SweetDroid.

The main idea of SweetDroid is to provide a calling-context-sensitive privacy policy enforcement on Android by leveraging state-of-the-art privacy analysis approaches. In particular, recent privacy analysis approaches can identify sensitive information flows and evaluate their appropriateness based on their contextual information for mobile applications [25, 26, 90, 99]. SweetDroid leverages the code-level observations and insights provided by those privacy analysis tools to construct privacy policies, and then regulate app's access to sensitive information and prevent privacy violations.

Depicted in Figure 5.1, there are mainly four technical components in SweetDroid. We briefly explain their functions and how they interact with other components in SweetDroid and the Android platform. First, SweetDroid will

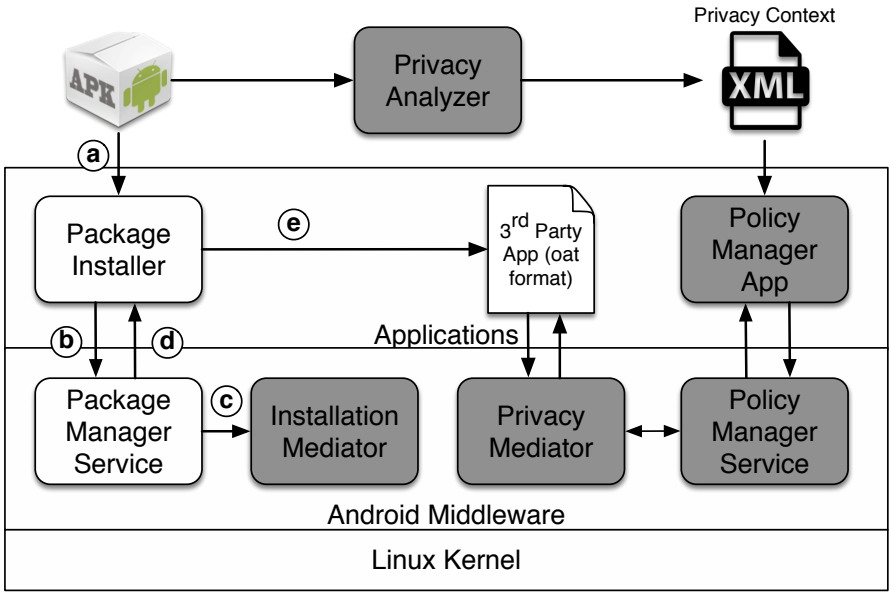


Figure 5.1. The design of SweetDroid framework (the SweetDroid components are colored as grey.)

leverage a Privacy Analyzer such as DroidJust [90] to analyze a given third-party app and automatically generates its *privacy context*. The privacy context file, in XML format, records the calling context of each sensitive API call (e.g., `getLastKnownLocation`), including the class and method where it is invoked, all its network sinks and whether the outbound flow to each sink is intended or unintended. This basically shows how risky each sensitive API is. The privacy context files are hosted in a dedicated server or separate servers for later retrieval.

Second, we mediate the app installation process through "patching". For instance, when a user downloads a third-party app from the Internet for installation, the apk file will be first passed to the `PackageInstaller`, which is the default application for Android to interactively install a normal package (step a). Upon the user confirms the installation request, the `PackageInstaller` will call the `PackageManagerService` (a system service in Android framework) through an `InstallAppProgress` activity (step b). Then, the `PackageManagerService` will verify the app, keep the metadata and call our Installation Mediator (IM) (step c). Just before the app is optimized and transformed into the Android runtime oat format for later execution, our IM scans the apk file to find all sensitive API calls and their calling contexts (including the class and method where each of them is in-

voked), and then rewrites each of such APIs with an extra parameter, which is the encrypted calling context information by a randomly-generated secret key. The IM applies the secret key throughout the entire app and sends the key associated with the app's package name to the PMS. The purpose of calling context information encryption is to prevent the original app from calling rewritten APIs with a legitimate calling context directly. We call this process of introducing the encrypted context information as an extra parameter as *API patching*. The patching is for future runtime access control. Note that this simple scanning cannot track information flows or identify privacy leakage. After that, the `PackageManagerService` will optimize the modified bytecode by calling the native `installd` daemon and then notify the `PackageInstaller` with installation success or failure (step d). The output of this installation is a modified oat file which is usually located at `/data/dalvik-cache/` in the Android internal storage (step e).

Third, once the app is successfully installed, a Policy Manager App, will download the privacy context file for this app from the server and send the file to Policy Manager Service (PMS), which is a system service to manage privacy policies in Android framework. Then, PMS will store the data of the privacy context file in a SQLite database. Moreover, PMS takes a role in parsing privacy context data and enforces a default privacy policy for each context. For example, if none of the network sinks for a `getLastKnownLocation()` API invocation is intended (according to the privacy context file), the default action, in this case, is DENY. Besides downloading privacy context files, Policy Manager App also provides a user-friendly interface for end users to view, comprehend and modify app's privacy policies if needed. For example, users can easily turn on or off the location information access for advertisement providers in an app. Note that Policy Manager App, as an interface in an application layer, does not maintain any privacy policy and all operations about privacy policy read and write go through PMS.

Fourth, during the actual execution of this app (i.e., when this oat file is running), whenever a (rewritten) sensitive API is invoked, the Privacy Mediator (PM) will handle the sensitive API invocation and check the PMS for this API, providing the app's package name, the signature of the sensitive API and the encrypted calling context (the last argument in the sensitive API invocation) to match the same API in the app's privacy context file in PMS and return the corresponding policy rule. We note that before matching, the PMS decrypts this encrypted call-

ing context with the corresponding secret key from the IM. If the action is DENY, the api will return a *null* value. If the action is MANIPULATION, the API will return totally random data (step 4).

Moreover, SweetDroid does not require third-party app repackaging and resigning before installation; hence, the whole API patching process is transparent to the user and the authorship of the mobile app remains for the Android shared UID feature [100] and hassle-free application upgrade.

5.4 Security Analysis

In this section, we analyze the security of SweetDroid by considering the following attack interfaces.

Java Code. SweetDroid relies on rewriting app's bytecode and adding sensitive API wrappers in Android framework to enforce privacy policies. SweetDroid implements these two parts directly in Android framework and thus they cannot be manipulated in runtime (since they are not under app's process).

In an attempt to bypass SweetDroid, a malicious app may try to use Java reflection or dynamic class loading to invoke original sensitive APIs instead of rewritten ones. However, this kind of attack could be easily detected by placing hooks in these original sensitive APIs because the Installation Mediator has previously replaced all sensitive API invocations in the app's bytecode and the triggered original sensitive API invocations must be caused by dynamic code behavior (i.e., Java reflection and dynamic loading). To defeat the attack, we simply redirect original sensitive APIs to their rewritten versions with a null context, and the rewritten APIs will deny the sensitive data access by default.

Moreover, a malicious app could use Java reflection or dynamic class loading to invoke our rewritten sensitive APIs with a fake calling context argument that matches a privacy policy rule as ALLOW. However, the malicious app cannot acquire the right calling context due to two reasons. First, the Installation Mediator encrypts the calling context with a randomly generated secret key and shares the secret key only with PMS. That means, even if the malicious app acquires a legitimate calling context of a sensitive API invocation, it cannot produce a valid encrypted version of the legitimate calling context. Second, the malicious app may try to look for a legitimate calling context directly from bytecode. For instance,

it could read its apk file to look for the legitimate calling context. Here we note that the app does have the read access to its apk file normally under `/data/app/`. However, this apk file is the original one which is untouched by our Installation Mediator and thus does not include the rewritten APIs. The file containing the rewritten APIs is the oat file under `/data/dalvik-cache/` wherein the app has no read access to read (unless the app acquires root privilege).

Native Code. Employing native code cannot bypass our privacy policy enforcement because SweetDroid intercepts neither IPC nor system calls, but enforces privacy policies inside the Android framework. A malicious app could leverage native code to tamper with the associated memory address space, but this cannot circumvent SweetDroid for privilege escalation.

Red Pill. SweetDroid is not designed to be invisible to an untrusted app. The untrusted app can use Java reflection to deduce that the app's code has been rewritten in runtime. Thus, a malicious app could hide its misbehavior and refuse to function in such a hardened environment. While it could harm the usability of an app for end users, the malfunction cannot lead to any privilege escalation.

5.5 Case Study

In this section, we present a case study to illustrate the usage of SweetDroid on a popular weather forecast app from Anzhi app market (a leading Android app market in China). SweetDroid can distinguish the different calling contexts of sensitive information requests and thus enhance user privacy.

We start the case study by installing the weather forecast application in a Nexus 5 device. Figure 5.2 shows the requested permissions when the app is installed. As we can see that the weather forecast app requests sensitive permissions to access users' phone and location information. From a user's point of view, it is reasonable to grant the location information access since the app could provide weather forecast tailored to the user's location. However, the user may have no clue about why the app needs to access phone information at the current stage.

After the user grants the permission requests, SweetDroid installs the weather forecast app through our *API patching* process. Once the app is successfully installed, the Policy Manager App (PMA) will download the privacy context for this application from our server (the privacy context is generated by Droidjust), and

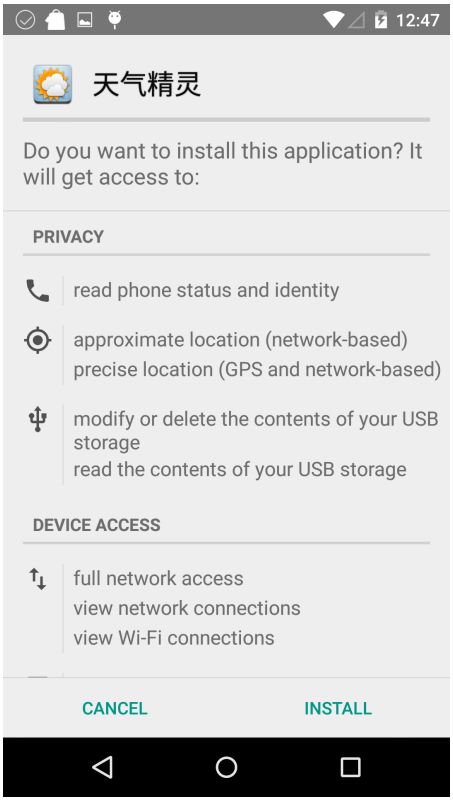


Figure 5.2. Permission request

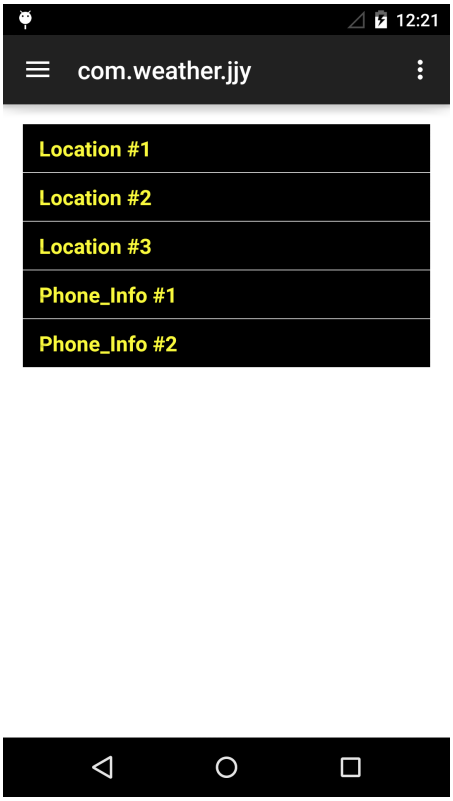


Figure 5.3. Policy manager app

send the privacy context file to the underlying Policy Manager Service (PMS). Figure 5.3 illustrates the user interface of PMA to show and modify privacy policies for installed applications. The top left toggle is used to switch between different apps. This figure particularly shows the privacy policy for the weather forecast app and the title is the app's package name. The following is a list of sensitive information requests, each representing a sensitive information request at a specific calling context. Figure 5.4 shows an example of a location information request. It contains the signature of the sensitive API, the signature of the API's calling context, the analysis result from the privacy context file, and current rule being applied to the sensitive information request. In the example, the location information request is justified (based on the analysis result) and the default rule is *allow*.

Figure 5.5 shows another location information request, which is used by an advertisement library, named *feiwo*, based on the analysis result. The default rule for the sensitive information request by advertisement libraries is *mock*. Figure 5.6

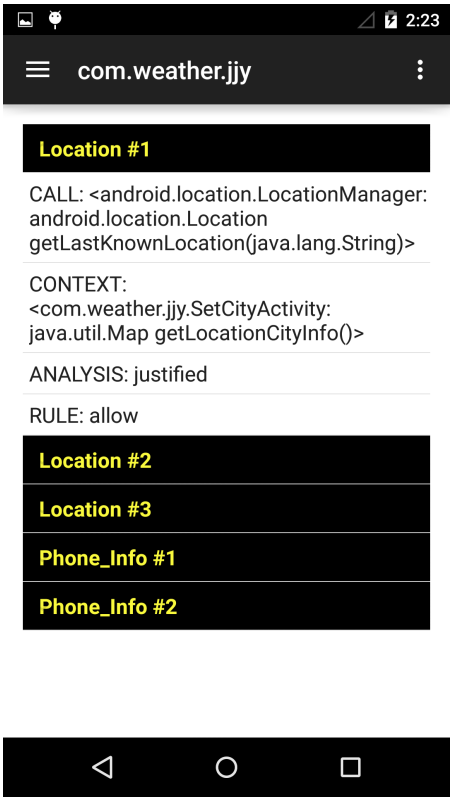


Figure 5.4. A location request by app

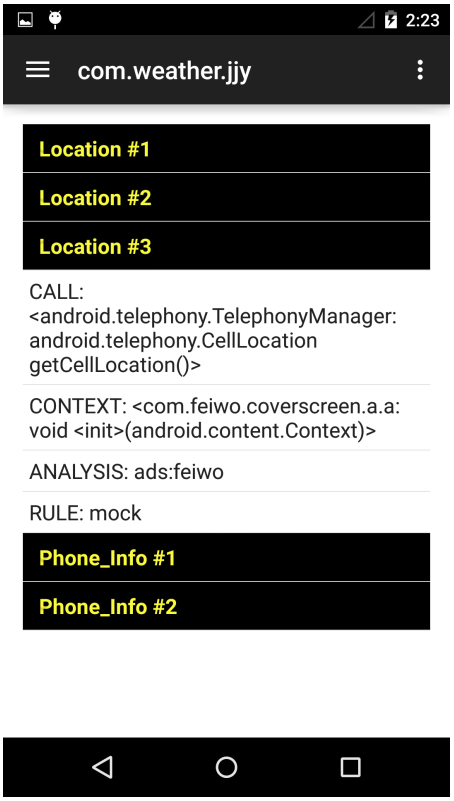


Figure 5.5. A location request by library

shows that the advertisement library also requests the device's IMEI information. Besides viewing the sensitive information requests in the app, the user can modify the policy rule for a sensitive information request at any time. Figure 5.7 shows an example of a popup menu after a long click on the rule that a user wants to modify. So far, SweetDroid provides three options including allow, mock, and deny. "Allow" returns genuine sensitive information, "mock" returns fake sensitive information, and "deny" simply returns null value. The "RECOMMENDED" flag shows the default rule based on the analysis result. After the modification, the rule takes effect immediately.

5.6 Evaluation

We evaluated SweetDroid on a collection of Android applications to ensure that application installations succeed and the added code does not impede the original functionality of applications. We conducted a broad evaluation which includes

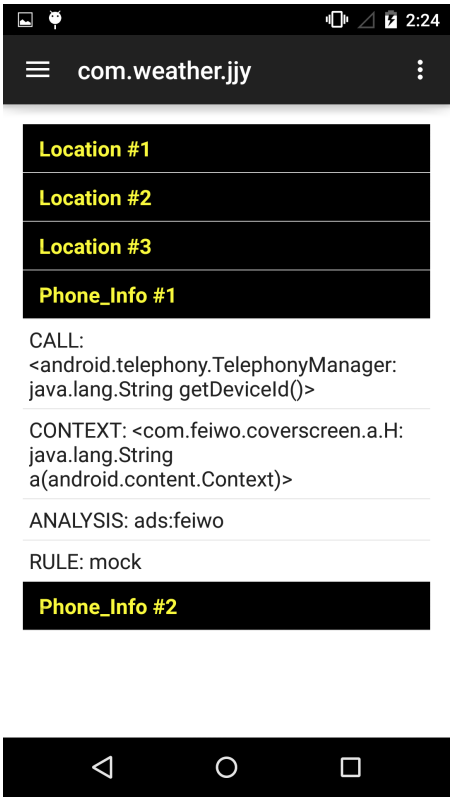


Figure 5.6. An IMEI request by library

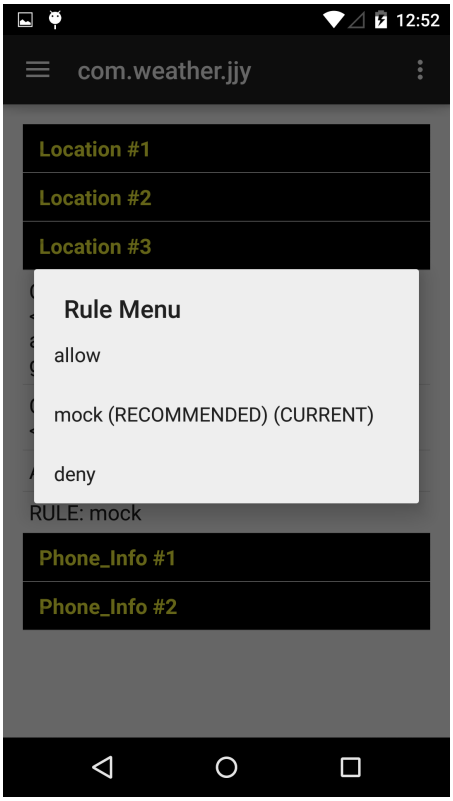


Figure 5.7. Modify current rule

3760 apps from Anzhi (a leading third-party app market in China) as well as more than 2759 malicious apps from VirusTotal. Our evaluation was conducted on an LG Nexus 5 phone running Android 5.0.1 "Lollipop".

5.6.1 Rewriting Evaluation

We first performed an evaluation to determine how many Android applications were successfully rewritten and installed by SweetDroid. We applied SweetDroid on 3760 apps from Anzhi and 2759 malicious apps from VirusTotal [85]. Table 5.1 shows the success rate of our rewritten process.

Source	# of App	Success Rate
Anzhi	3760	98.6%(3709)
VirusTotal	2759	95.9%(2646)

Table 5.1. Repackaging Evaluation Results

We have more than 95% success rate in application rewritten. The failures in

rewriting arbitrary applications are due to errors in `apktool` when disassembling APK files (e.g., error in decoding application resource files, error in opening zip files and invalid magic number in decoding). We are trying to work on improving `apktool` to achieve a 100% success rate.

5.6.2 Size Overhead

We also evaluated the application size increase due to SweetDroid's rewriting process, as shown in Figure 5.8. On average, SweetDroid increases the application size by only 11.6 KB, which is a very small overhead for the majority of applications.

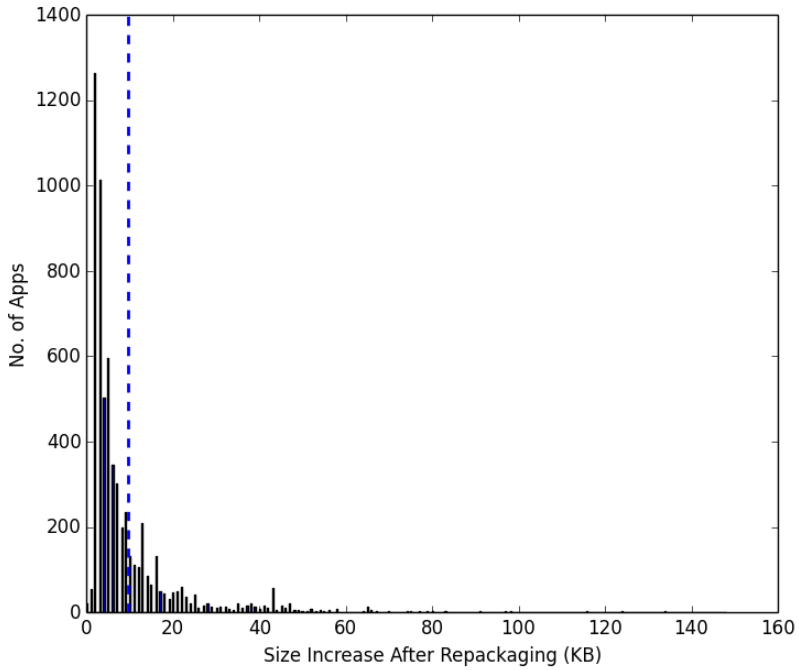


Figure 5.8. Application Size Increase After Repackaging

5.6.3 Privacy Policy Generation

We employed DroidJust [90] to automatically analyze privacy leakage and generate privacy policies for those Android applications. Table 5.2 shows the results, where the third column shows the number of apps that contain unjustified sensitive

information leak in their application logic (refer to [90] for more details about unjustified sensitive information leak), the fourth column shows the number of apps that contain sensitive information leak in their advertisement logic, and the last column shows the number of apps that contain either one.

The results demonstrate that more than 30% apps from both Anzhi and VirusTotal have privacy leakage issues. However, these two application datasets have a significant difference in the leaking area. Particularly, most privacy-invasive apps from Anzhi leak users' sensitive information through advertisement libraries rather than application logic, whereas nearly all privacy-invasive apps (887 of 989) from VirusTotal leak users' sensitive information directly in their application logic. There are two important observations. The first observation is that advertisement libraries are the major cause of users' sensitive information leakage since, as we know, most advertisement libraries send out users' sensitive information such as IMEI and location for location-based or targeted advertising. This second observation is that privacy-invasive apps are mostly not identified as malware by antivirus vendors unless they leak user privacy through their application logic.

Source	# of Apps	Unjustified Leak by App	Leak by Ads	Either
Anzhi	3760	488(13.0%)	1004(26.7%)	1254(33.4%)
VirusTotal	2759	887(32.1%)	199(7.2%)	989(35.8%)

Table 5.2. Privacy Leakage Analysis

Table 5.3 shows a breakdown of the leaked sensitive information for those apps. On the one hand, we can see that the privacy-invasive apps from Anzhi mostly leak users' phone information (e.g., IMEI, IMSI, phone number, etc.) and location but rarely leak other kinds of information such as messages and contacts. It validates our previous observation that the major cause of the leaks in this dataset is for advertising purpose. On the other hand, we can see that the privacy-invasive apps from VirusTotal not only leaks phone and location information but also leak messages and contacts significantly. Based on our study, we found that those malware leak contacts and messages for malicious attacks. For example, harvesting user contacts is used for message or email spam, and a malware stealthily calling premium numbers usually intercepts and forwards the subscription message to its malicious server.

Source	Phone Info	Location	Messages	Contacts	Call Logs	Accounts
Anzhi	1222	576	3	7	1	1
VirusTotal	719	79	457	102	2	2

Table 5.3. Leaked Sensitive Information

5.6.4 Performance Evaluation

We employed two most popular Android benchmark applications from the official market and applied SweetDroid to them in order to check the performance overhead that SweetDroid introduces to a real Android device. From Table 5.4, we can see that the benchmark scores are largely unaffected by SweetDroid. Note that higher scores mean better performance.

Benchmark App	Without SweetDroid	With SweetDroid
AnTutu Benckmark (5.7.1)	35621 Pts	35527 Pts
Geekbench 3 (3.3.2)	786 Pts	780 Pts

Table 5.4. Performance on Benchmark Applications

Because SweetDroid introduces the most overhead when an application invokes sensitive API invocations, we further evaluate an artificial app's runtime delay caused by SweetDroid when it invokes sensitive APIs. The artificial app calls each sensitive API invocations for 1000 times. Results in Table 5.5 show that SweetDroid introduces an overhead of 13-15% in both cases. We believe that the incurred overhead is acceptable since the averaging delay for each sensitive API invocation is less than 0.3 ms and such a short time interval will not affect user experience.

1000 API invocations	Without SweetDroid	With SweetDroid	Overhead
Get Device ID	1786 ms	2022 ms	13.2%
Get Last Location	278 ms	318 ms	14.4%

Table 5.5. Performance on API invocations

5.7 Summary

In this chapter, we present SweetDroid, a calling-context-sensitive privacy policy enforcement framework to automatically generate calling-context-sensitive privacy

policies for Android applications and enforce those privacy policies at the calling context level in runtime in order to effectively enhance user privacy yet retaining app's usability. Our evaluation demonstrated that SweetDroid can effectively distinguish users' sensitive information use in different contexts of an Android application and then apply proper privacy policies to prevent unnecessary privacy leakage.

Chapter 6
Conclusion and Future Works

6.1 Conclusion

In this dissertation, we present our studies focusing on mobile security with a particular interest on tactical networks and mobile phone networks. For tactical networks, we propose ZIGZAG, a partial mutual revocation based trust management scheme, which allows rapid impeachment of identified malicious nodes and offers a node a certain degree of freedom to tradeoff the extent of sacrifice with the global good of the network. Our analytical and experimental results have shown that the proposed partial mutual revocation approach is more accurate than conventional complete mutual revocation approaches by sacrificing revocation immediacy a bit. Also, our evaluation has shown that the partial mutual revocation is very robust against strategic (false) accusations made by bad nodes and erroneous accusations made by good nodes (e.g., due to benign errors) to retain the availability of the network.

In a tactical network environment, we also present GLOBALTRUST, a trust-based reputation management scheme, to accurately evaluate the reputation of nodes with respect to both the behavioral trustworthiness and recommendation credibility. Through the extensive simulation experiments, we have demonstrated that GLOBALTRUST outperforms existing reputation schemes by highly being resilient against various types of attacks, maintaining high view consistency through the network and generating low reputation judgment errors.

With a massive increase of smartphones and mobile devices over the past few years, mobile malware is exploding. Based on the stats from a malware analysis

research company, privacy leakage is the highest-ranked threat in smartphones and mobile devices. To mitigate the severe privacy threat in mobile phone networks, we present DroidJust, an automated, functionality-aware approach to justifying an app's sensitive information release. Our evaluation on real-world Android apps and known malware has demonstrated that DroidJust can effectively and efficiently analyze both benign apps and malware.

Further, in order to mitigate privacy leakage in the mobile device side, we present a calling-context-sensitive, fine-grained privacy policy enforcement framework, named SweetDroid. Our policy enforcement framework is able to enforce privacy policies at the calling context level. By doing so, SweetDroid can grant or deny sensitive permission for different sensitive information requests based on their specific calling contexts.

6.2 Future Works

For future works, we are interested in investing our efforts in the following directions.

Trust and Reputation Management in Tactical Ad Hoc Networks. Our future work will study other possible attack strategies as well as more extensive simulations to compare Zigzag or GlobalTrust with other existing revocation schemes.

Functionality-Aware Privacy Leakage Analysis for Android Applications. Our future work include: (1) improve the correlation accuracy between sensitive information transmission and sensible information reception by leveraging machine learning techniques; (2) support the discovery of implicit information flows; (3) enhance data propagation through native code by modeling the known JNI call in Android framework; and (4) host an online service to provide the privacy leakage analysis of Droidjust.

Calling-Context-Sensitive Privacy Policy Enforcement Framework for Android Applications. We plan our future work on further improving the security of SweetDroid, providing extensions for state-of-art privacy analysis tools (e.g., TaintDroid) to automatically generate privacy context, and hosting a public repository of privacy context for Android applications.

Bibliography

[1] Aberer, K. and Z. Despotovic (2001) "Managing trust in a peer-2-peer information system," in *Proc. CIKM*.

[2] Ooi, B., C. Liau, and K. Tan (2003) "Managing trust in peer-to-peer systems using reputation-based techniques," *Advances in Web-Age Information Management*, pp. 2–12.

[3] Wang, Y. and J. Vassileva (2003) "Trust and reputation model in peer-to-peer networks," in *Peer-to-Peer Computing, 2003.(P2P 2003). Proceedings. Third International Conference on*, IEEE, pp. 150–157.

[4] "An Android malware summary from Mobile-Sandboxing," `http://forensics.spreitzenbarth.de/2015/05/18/our-android-malware-summary-for-the-year-2014/`.

[5] Arzt, S., S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel (2014) "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the 35th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI 2014)*.

[6] Egele, M., C. Kruegel, E. Kirda, and G. Vigna (2011) "PiOS: Detecting Privacy Leaks in iOS Applications." in *NDSS*.

[7] Enck, W., P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth (2010) "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones." in *OSDI' 10*, vol. 10, pp. 1–6.

[8] Enck, W., D. Octeau, P. McDaniel, and S. Chaudhuri (2011) "A Study of Android Application Security." in *USENIX Security Symposium*.

[9] Gilbert, P., B.-G. Chun, L. P. Cox, and J. Jung (2011) "Vision: automated security validation of mobile apps at app markets," in *Proceedings of the second international workshop on Mobile cloud computing and services*, ACM, pp. 21–26.

[10] Gibler, C., J. Crussell, J. Erickson, and H. Chen (2012) "AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale," in *Trust and Trustworthy Computing*, Springer, pp. 291–307.

[11] Hornyack, P., S. Han, J. Jung, S. Schechter, and D. Wetherall (2011) "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM conference on Computer and communications security, CCS' 11*, ACM, pp. 639–652.

[12] Mann, C. and A. Starostin (2012) "A framework for static detection of privacy leaks in android applications," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ACM, pp. 1457–1462.

[13] Rastogi, V., Y. Chen, and W. Enck (2013) "AppsPlayground: automatic security analysis of smartphone applications," in *Proceedings of the third ACM conference on Data and application security and privacy*, ACM, pp. 209–220.

[14] Backes, M., S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky (2013) "AppGuard–enforcing user requirements on android apps," in *Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 543–548.

[15] Backes, M., S. Bugiel, C. Hammer, O. Schranz, and P. von Styp-Rekowsky (2015) "Boxify: Full-fledged app sandboxing for stock Android," in *Proc. USENIX Security*.

[16] Beresford, A. R., A. Rice, N. Skehin, and R. Sohan (2011) "Mockdroid: trading privacy for application functionality on smartphones," in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, ACM, pp. 49–54.

[17] Conti, M., V. T. N. Nguyen, and B. Crispo (2011) "CRePE: Context-related policy enforcement for Android," in *Information Security*, Springer, pp. 331–345.

[18] Davis, B. and H. Chen (2013) "RetroSkeleton: retrofitting android apps," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, ACM, pp. 181–192.

[19] Davis, B., B. Sanders, A. Khodaverdian, and H. Chen (2012) "I-arm-droid: A rewriting framework for in-app reference monitors for android applications," *Mobile Security Technologies*, **2012**.

[20] Jeon, J., K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein (2012) "Dr. Android and Mr. Hide: fine-grained permissions in android applications," in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, ACM, pp. 3–14.

[21] Nauman, M., S. Khan, and X. Zhang (2010) "Apex: extending android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ACM, pp. 328–332.

[22] Ongtang, M., S. McLaughlin, W. Enck, and P. McDaniel (2012) "Semantically rich application-centric security in Android," *Security and Communication Networks*, **5**(6), pp. 658–673.

[23] Wu, C., Y. Zhou, K. Patel, Z. Liang, and X. Jiang (2014) "Airbag: Boosting smartphone resistance to malware infection," in *Proceedings of the Network and Distributed System Security Symposium*.

[24] Xu, R., H. Saïdi, and R. Anderson (2012) "Aurasium: Practical Policy Enforcement for Android Applications." in *USENIX Security Symposium*, pp. 539–552.

[25] Tripp, O. and J. Rubin (2014) "A bayesian approach to privacy enforcement in smartphones," in *USENIX Security*.

[26] Yang, Z., M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang (2013) "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM, pp. 1043–1054.

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

[28] Au, K. W. Y., Y. F. Zhou, Z. Huang, and D. Lie (2012) "Pscout: analyzing the android permission specification," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ACM, pp. 217–228.

[29] Clulow, J. and T. Moore (2006) "Suicide for the Common Good: A New Strategy for Credential Revocation in Self-organizing Systems," *ACM SIGOPS Operating Systems Reviews*, **40**(3), pp. 18–21.

[30] Moore, T., M. Raya, J. Clulow, P. Papadimitratos, R. Anderson, and J.-P. Hubaux (2008) "Fast Exclusion of Errant Devices From Vehicular Networks," in *Proceedings of the 5th conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2008)*, pp. 135–143.

[31] Raya, M., M. H. Manshaei, M. Félegyhazi, and J.-P. Hubaux (2008) "Revocation games In ephemeral networks," in *Proc. CCS*.

[32] Reidt, S., M. Srivatsa, and S. Balfe (2009) "The fable of the bees: incentivizing robust revocation decision making in ad hoc networks," in *Proc. CCS*.

[33] T. Moore, R. A., J. Clulow and S. Nagaraja (2007) "New strategies for revocation in ad-hoc networks," in *Proc. ESAS*.

[34] Zhu, S., S. Setia, S. Xu, and S. Jajodia (2006) "GKMPAN: An efficient group rekeying scheme for secure multicast in ad-hoc networks," *Journal of Computer Security*, **14**(4), pp. 301–325.

[35] Liu, D., P. Ning, and K. Sun (2003) "Efficient self-healing group key distribution with revocation capability," in *Proc. CCS*.

[36] Liu, W. (2006) "Securing Mobile Ad Hoc Networks with Certificateless Public Keys," *IEEE Transactions on Dependable and Secure Computing*, **3**(4), pp. 386–399.

[37] Yi, S. and R. Kravets (2003) "MOCA: Mobile certificate authority for wireless ad hoc networks," in *Proc. PKI*.

[38] Matt, B. (2004) "Toward Hierarchical Identity-based Cryptography for Tactical Networks," in *Proceedings of the 2004 Military Communications Conference (MILCOM 2003)*, IEEE Computer Society, pp. 727–735.

[39] Rivest, R. (1998) "Can we eliminate certificate revocation lists?" in *Proc. FC*.

[40] Kanungo, T., D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu (2002) "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE TPAMI*.

[41] Marti, S., T. Giuli, K. Lai, and M. Baker (2000) "Mitigating routing misbehavior in mobile ad hoc networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, ACM, pp. 255–265.

[42] Pirzada, A., A. Datta, and C. McDonald (2004) "Trust-based routing for ad-hoc wireless networks," in *Proc. ICON*.

[43] McCune, J. M., E. Shi, A. Perrig, and M. K. Reiter (2005) "Detection of denial-of-message attacks on sensor network broadcasts," in *Proc. S&P*.

[44] White, D. J. and C. E. White (1993) *Markov decision processes*, 1 ed., Wiley, John & Sons, Incorporated.

[45] Reidt, S. and S. D. Wolthusen (2007) "Efficient Distribution of Trust Authority Functions in Tactical Networks," in *Information Assurance and Security Workshop*.

[46] Cho, J., A. Swami, and I. Chen (2011) "A survey of trust management in mobile ad hoc networks," *IEEE Communications Surveys and Tutorials*, **13**(4), pp. 562–583.

[47] Li, H. and M. Singhal (2007) "Trust management in distributed systems," *Computer*, **40**(2), pp. 45 –53.

[48] Liu, Z., A. W. Joy, and R. A. Thompson (2003) "A dynamic trust model for mobile ad hoc networks," in *Proc. 10th IEEE Int'l Workshop on Future Trends of Distributed Computing Systems*.

[49] Kamvar, S. D., M. T. Schlosser, and H. Garcia-Molina (2003) "The Eigentrust algorithm for reputation management in P2P networks," in *Proc. WWW*.

[50] Zhou, R. and K. Hwang (2007) "PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing," *IEEE Trans. Parallel Distrib. Syst.*, **18**(4), pp. 460–473.

[51] Bella, G., G. Costantino, and S. Riccobene (2008) "Managing reputation over mANETs," in *Information Assurance and Security*.

[52] Arboit, G., C. Crépeau, C. R. Davis, and M. Maheswaran (2008) "A localized certificate revocation scheme for mobile ad hoc networks," *Ad Hoc Netw.*, **6**(1), pp. 17–31.

[53] Buchegger, S. and J. Le Boudec (2003) "A robust reputation system for mobile ad-hoc networks," in *Proc. P2PEcon*.

[54] He, Q., D. Wu, and P. Khosla (2004) "SORI: a secure and objective reputation-based incentive scheme for ad-hoc networks," in *Proc. WCNC*.

[55] Teacy, W. L., J. Patel, N. R. Jennings, and M. Luck (2006) "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, **12**(2), pp. 183–198.

[56] Jsang, A. and R. Ismail (2002) "The beta reputation system," in *Proceedings of the 15th bled electronic commerce conference*.

[57] Chan, H., V. D. Gligor, A. Perrig, and G. Muralidharan (2005) "On the Distribution and Revocation of Cryptographic Keys in Sensor Networks," *IEEE Trans. Dependable Secur. Comput.*, **2**(3), pp. 233–247.

[58] Raya, M., M. H. Manshaei, M. Félegyhazi, and J.-P. Hubaux (2008) "Revocation games in ephemeral networks," in *Proc. CCS*.

[59] Buchegger, S. and J. Le Boudec (2002) "Performance analysis of the CONFIDANT protocol," in *Proc. MobiHoc*.

[60] Kong, J., P. Zerfos, H. Luo, S. Lu, and L. Zhang (2001) "Providing Robust and Ubiquitous Security Support for Mobile Ad-hoc Networks," in *IEEE ICNP*.

[61] Mármol, F. and G. Pérez (2009) "Security threats scenarios in trust and reputation models for distributed systems," *computers & security*, **28**(7), pp. 545–556.

[62] Benzécri, J. (1982) "Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques," *Les Cahiers de l'Analyse des Données*, **7**(2), pp. 209–218.

[63] Rhee, I., M. Shin, S. Hong, K. Lee, S. Kim, and S. Chong (2009), "CRAWDAD, data set ncsu/mobilitymodels (v. 2009-07-23)," .

[64] Michiardi, P. and R. Molva (2002) "Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," in *Advanced Communications and Multimedia Security*.

[65] Zhou, Y. and X. Jiang (2012) "Dissecting android malware: Characterization and evolution," in *IEEE Symposium on Security and Privacy, S&P' 12*, IEEE, pp. 95–109.

[66] "Mobile-Sandbox," `http://forensics.spreitzenbarth.de/android-malware/`.

[67] Octeau, D., S. Jha, and P. McDaniel (2012) "Retargeting Android Applications to Java Bytecode," in *Proceedings of the 20th International Symposium on the Foundations of Software Engineering*, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA.
URL `http://siis.cse.psu.edu/dare/papers/octeau-fse12.pdf`

[68] "Fortify 360 source code analyzer," http://www8.hp.com/us/en/software-solutions/software.html?compURI=1338812#.U3U1YlhdXKo.

[69] "WALA, T. J. Watson libraries for analysis," http://wala.sourceforge.net/.

[70] Xu, W., F. Zhang, and S. Zhu (2013) "Permlyzer: Analyzing permission usage in android applications," in *IEEE 24th International Symposium on Software Reliability Engineering, ISSRE' 13*, IEEE, pp. 400–410.

[71] Rasthofer, S., S. Arzt, and E. Bodden (2014) "A machine-learning approach for classifying and categorizing android sources and sinks," in *Network and Distributed System Security Symposium, NDSS' 14*.

[72] Zhang, M. and H. Yin (2014) "Efficient, Context-Aware Privacy Leakage Confinement for Android Applications without Firmware Modding," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2014)*.

[73] Pandita, R., X. Xiao, W. Yang, W. Enck, and T. Xie (2013) "WHYPER: Towards Automating Risk Assessment of Mobile Applications." in *USENIX Security*.

[74] Qu, Z., V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen (2014) "AutoCog: Measuring the Description-to-permission Fidelity in Android Applications," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS' 14*, ACM, pp. 1354–1365.

[75] Jung, J., A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. Kohno (2008) "Privacy oracle: a system for finding application leaks with black box differential testing," in *Proceedings of the 15th ACM conference on Computer and communications security, CCS' 08*, ACM, pp. 279–288.

[76] Yumerefendi, A. R., B. Mickle, and L. P. Cox (2007) "TightLip: Keeping Applications from Spilling the Beans." in *NSDI' 07*.

[77] Felt, A. P., E. Chin, S. Hanna, D. Song, and D. Wagner (2011) "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, pp. 627–638.

[78] Hoffmann, J., M. Ussath, T. Holz, and M. Spreitzenbarth (2013) "Slicing droids: program slicing for smali code," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ACM, pp. 1844–1851.

[79] Kim, J., Y. Yoon, K. Yi, J. Shin, and S. Center (2012) "ScanDal: Static analyzer for detecting privacy leaks in android applications," *MoST*.

[80] Lu, L., Z. Li, Z. Wu, W. Lee, and G. Jiang (2012) "Chex: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM conference on Computer and communications security, CCS' 12*, ACM, pp. 229–240.

[81] Yang, Z. and M. Yang (2012) "Leakminer: Detect information leakage on android with static taint analysis," in *WCSE' 12*, IEEE, pp. 101–104.

[82] Reps, T., S. Horwitz, and M. Sagiv (1995) "Precise interprocedural dataflow analysis via graph reachability," in *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM, pp. 49–61.

[83] Vallée-Rai, R., P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan (1999) "Soot-a Java bytecode optimization framework," in *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, p. 13.

[84] Bodden, E. (2012) "Inter-procedural data-flow analysis with ifds/ide and soot," in *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis*, ACM, pp. 3–8.

[85] "VirusTotal," `https://www.virustotal.com/`.

[86] Weichselbaum, L., M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. van der Veen, and C. Platzer (2014) "Andrubis: Android Malware Under The Magnifying Glass," *Vienna University of Technology, Tech. Rep. TRISECLAB-0414-001*.

[87] Li, L., A. Bartel, J. Klein, and Y. L. Traon (2014) "Automatically exploiting potential component leaks in android applications," in *Trust, Security and Privacy in Computing and Communications, TrustCom' 14*, IEEE, pp. 388–397.

[88] Gordon, M. I., D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard (2015) "Information-flow analysis of Android applications in DroidSafe," in *Proceedings of the 22nd Annual Network and Distributed System Security Symposium, NDSS'15*.

[89] "Android 6.0 permission model," `https://developer.android.com/preview/features/runtime-permissions.html?utm_campaign=m-developer-launch&utm_source=dac&utm_medium=blog`.

[90] Chen, X. and S. Zhu (2015) "DroidJust: Automated Functionality-Aware Privacy Leakage Analysis for Android Applications," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*.

[91] Zhou, Y., X. Zhang, X. Jiang, and V. W. Freeh (2011) "Taming information-stealing smartphone applications (on android)," in *Trust and Trustworthy Computing*, Springer, pp. 93–107.

[92] Smalley, S., C. Vance, and W. Salamon (2001) "Implementing SELinux as a Linux security module," *NAI Labs Report*, **1**(43), p. 139.

[93] Bugiel, S., S. Heuser, and A.-R. Sadeghi (2013) "Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies." in *Usenix security*, pp. 131–146.

[94] Smalley, S. and R. Craig (2013) "Security Enhanced (SE) Android: Bringing Flexible MAC to Android." in *NDSS*, vol. 310, pp. 20–38.

[95] Wang, Y., S. Hariharan, C. Zhao, J. Liu, and W. Du (2014) "Compac: Enforce component-level access control in Android," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, ACM, pp. 25–36.

[96] Liu, B., B. Liu, H. Jin, and R. Govindan (2015) "Efficient Privilege De-Escalation for Ad Libraries in Mobile Apps," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, ACM, pp. 89–103.

[97] Pearce, P., A. P. Felt, G. Nunez, and D. Wagner (2012) "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ACM, pp. 71–72.

[98] Shekhar, S., M. Dietz, and D. S. Wallach (2012) "AdSplit: Separating Smartphone Advertising from Applications." in *USENIX Security Symposium*, pp. 553–567.

[99] Huang, J., X. Zhang, L. Tan, P. Wang, and B. Liang (2014) "AsDroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction," in *Proceedings of the 36th International Conference on Software Engineering*, ACM, pp. 1036–1046.

[100] "Android shared UID feature," `http://developer.android.com/guide/topics/security/permissions.html`.

Vita

Xin Chen

Xin Chen enrolled in the Ph.D. program in Computer Science and Engineering at Pennsylvania State University in 2010. Prior to that, he received the B.S. degree in Computer Science from Nanjing University, P.R.China in 2010. His research interests include trust and reputation management in mobile ad hoc networks, and various security and privacy issues in mobile phones.

Publications during the Ph.D. study include:

- **X. Chen**, and S. Zhu. "DroidJust: Automated Functionality-Aware Privacy Leakage Analysis for Android Applications". Proceedings of the 8th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec), 2015.

- **X. Chen**, J. Cho, and S. Zhu. "GlobalTrust: An Attack-Resilient Reputation System for Tactical Networks". Proceedings of the 11th IEEE Conference on Sensing and Communication in Wireless Networks (SECON), 2014.

- **X. Chen**, H. Patankar, S. Zhu, M. Srivatsa, J. Opper. "Zigzag: Partial Mutual Revocation Based Trust Management in Tactical Ad Hoc Networks". Proceedings of the 10th IEEE Conference on Sensing and Communication in Wireless Networks (SECON), 2013.

- Z. Xu, H. Hsu, **X. Chen**, S. Zhu and A. Hurson. "AK-PPM: An Authenticated Packet Attribution Scheme for Mobile Ad Hoc Networks". Proceedings of the 15th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2012.

Awards during the Ph.D. study include:

- Student travel grants for WiSec' 15.

- First place in Penn State Cisco Innovation Challenge, March 2012.