國立交通大學

電控工程研究所

博士論文

從攻擊角度定量評估資訊系統安全性

Quantitative Assessments of Cyber Security from the Perspective of Attacks

研究生： 蔡欣宜

指導教授： 黃育綸博士

中華民國一百年十二月

從攻擊角度定量評估資訊系統安全性

Quantitative Assessments of Cyber Security from the Perspective of Attacks

A Dissertation

by

Hsin-Yi Tsai

Submitted to the Institute of Electrical Control Engineering

National Chiao Tung University

in partial fulfillment of the requirements

for the Degree of

Doctor of Philosophy

in

Electrical Control Engineering

Advisor: Dr. Yu-Lun Huang

December 2011

Hsinchu, Taiwan, Republic of China

從攻擊角度定量評估資訊系統安全性

研究生: 蔡欣宜　　指導教授: 黃育綸博士

國立交通大學電控工程研究所

摘要

資訊安全評估機制可以提供資訊系統的安全評估結果，協助系統管理者有效地瞭解系統之安全性，並成爲系統管理者管理該系統之參考依據。由於一個系統的安全性涉及許多因素，諸如系統設定、安全機制、現有攻擊方式等等，因此資訊安全的評估不能僅考慮單一面向，而必須要能同時考慮多項因素所造成的影響。本文分別由系統外部與內部攻擊的角度出發，探討資訊安全評估方法之設計，及其所能提供的評估結果。在外部攻擊方面，本文提出一個無線網路風險評估方法；該方法首先考慮網路系統的安全條件、攻擊手法與系統設定，以建立風險模型，接著本文再提出一套量測準則，藉以量化風險數值。在內部攻擊方面，本文提出一套量化分析軟體控制流程模糊化之方法，以評估控制流程模糊化對軟體強韌度之影響。該方法基於控制流程圖之概念，將控制流程模糊化轉換爲正規表示式。以此正規表示式爲基礎，本文進一步提出新的量測準則，以計算軟體控制流程模糊化所提供的保護能力。最後，本文利用數個範例，說明並驗證本文所提方法之可行性。我們相信本文所提之方法能提供系統管理者更全面的資訊安全評估結果，並進一步地協助系統管理者管理該系統。

Quantitative Assessments of Cyber Security from the Perspective of Attacks

Student: Hsin-Yi Tsai Advisor: Dr. Yu-Lun Huang

Institute of Electrical Control Engineering

National Chiao Tung University

Abstract

Assessment of cyber security is a long-standing and great challenge since multifarious factors and their reciprocal effects have to be considered in the meanwhile for the assessment. Due to its complexity, assessment of cyber security should be performed with multiple aspects. This dissertation presents the quantitative assessments from the perspectives of both external and internal attacks. Regarding assessing cyber security in terms of external attacks, we propose a wireless risk assessment method which consists of a risk model and an assessment measure. The risk model is in charge of modeling wireless network risk, and the assessment measure is an algorithm of determining the risk value per the risk model. As for internal attacks, we introduce a novel framework to evaluate software robustness in terms of control-flow obfuscating transformations. On the basis of this framework, we propose new metrics for quantifying the protection effect yielded by a control-flow obfuscating transformation. Moreover, we conduct the case studies to validate the proposed assessment methods. We believe that our methods are helpful for a system administrator to evaluate and manage the cyber security in a more effective way.

Acknowledgement

It has been a long journey, but I am fortunate and thankful where I am.

I am pleased to thank every one who supported, encouraged and accompanied me in the years of my Ph.D. study. First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Yu-Lun Huang, for her patience, motivation, enthusiasm, endless encouragement, immense knowledge and continuous support of my research. Her guidance helped me in all the time of research and writing of this thesis. Her encouragement widened my sight and let me see an amazingly different world. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Hahn-Ming Lee, Prof. Chin-Laung Lei, Prof. Yuan-Pei Lin, Prof. Shiuhpyng Shieh, Prof. Hung-Min Sun and Prof. Wen-Guey Tzeng, for their insightful comments and valuable time. Their valued feedback helped me improve the quality of my dissertation.

I am also grateful to Prof. Doug Tygar and Prof. David Wagner, who supervised me during my visit to UC Berkeley in my first year of Ph.D. study. I learned a lot from their passion for research, unique perspectives on academy and careful criticism. My appreciation also goes to Dr. André Miede, who hosted and supervised me when I was a visiting scholar at TU Darmstadt, Germany in my fourth year of Ph.D. study. André deeply impressed me with his sense to research ideas and insightful views.

I owe lots of gratitudes to my dearest RTES lab-mates, for their support, their sweet consideration and all the wonderful time we had together. How

lucky I am to be a member of RTES! Special thanks go to Yung-Wen and Borting. They always helped me a lot in not only project execution, research discussion but also administrative services of our laboratory.

Finally, I would like to give my deepest gratitudes to my parents, who made me who I am. I cannot go that far without their unconditional love. Many thanks to my brother as well for his support and caring during my Ph.D. study. I dedicate this dissertation to them, my beloved family.

Dedicated to my family

獻給我的家人

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Assessment of cyber security is important since *we cannot improve what we cannot measure* [1]. The assessment results are helpful for system administrators and users to understand system security easily. Then, the administrators are capable of designating countermeasures, applying protection mechanisms, or modifying system configurations to increase security according to the assessment results. Nevertheless, assessment of cyber security is critical since various factors (such as security countermeasures, system configurations, vulnerabilities and realistic attacks) are involved to pose individual effects on cyber security and yield different levels of security risk. It is thus difficult to assess cyber security from a holistic perspective because the multifarious factors and their reciprocal effects have to be considered in the meanwhile.

Cyber security can be compromised in many ways. Security mechanisms and configurations are designed and applied to fortify against different attacks. Hence, to plausibly assess security, the assessment should be performed from the aspect of attacks that attack targets, prerequisite config-

urations of an attack and attack impacts are involved. Cyber attacks take various forms and are coarsely classified into two types: external and internal attacks. An external attack is launched by an adversary outside a victim system. An internal attack is started by an attacker who is a legal user of the victim system. Upon attacking a network, an external attacker intends to gather information concerning the network system from the outside and then launches attacks accordingly, while an internal attacker, accessible to a victim, can control the system and assault the victim's data and programs. Security of a network system hence should be evaluated from both the external and internal aspects to better reflect the realistic situations.

There are many implementations of external attacks, such as penetration attacks, Denial-of-Service (DoS) attacks and eavesdropping attacks. According to the variations of attacks, various methods of assessing cyber security in terms of external attacks have been proposed. The methods include attack graph-based methods [2, 3, 4, 5, 6, 7, 8] and analytic hierarchy process (AHP)-based methods [9, 10, 11, 12, 13]. An attack graph-based method assesses the security of a network system based on analyzing the system's attack graph, which is drawn mainly from the aspect of penetration attacks. The attack graph-based methods are widely used in assessing security of wired networks, but they are not that appropriate for a dynamic network environment. The whole attack graph needs to be re-generated once the topology or configurations of a network system change. Such re-generation could cause a heavy load for assessing the cyber security due to the frequent change of a dynamic environment. The AHP is a structured technique for decision making problems [14, 15]. It has been applied to several realms, such

as planning, system designing and risk assessment. Zhao et al. applied the concept of AHP to modeling and assessing network security risk [9, 10, 11]. However, Zhao et al. developed a 3-layer hierarchical structure which is not sufficient to discuss the security impacts resulting from the incorrect configurations. [12] and [13] concentrate on the design of the methodology for risk assessment based on the AHP, but their focus does not lay in the design of the risk model to better represent the real security situation. Therefore, there is a need to establish a feasible risk model and design a practical risk assessment method which meets the ground truth.

Unlike an external attacker, an internal adversary obtains the privilege prior to launching an attack so that the adversary is authorized to manipulate the stored data and programs. Factors critical for assessing cyber security against external attacks may not be as crucial for the assessment from the internal attack perspective. In comparison, evaluating robustness of data and programs against internal attacks is the core of security assessment. Much research has been proposed to evaluate capabilities of data protection mechanisms, such as data encryption and digital watermarking. As for the protection of programs, comparatively little attention has been received in evaluating the program protection mechanisms like software obfuscation and software tamperproofing [16].

To distinguish the existing security assessment methods, this dissertation offers solutions to assessing cyber security in different scenarios and test cases. We present several quantitative assessments of cyber security in terms of both external and internal attacks. We develop a wireless risk assessment method, which is composed of a risk model and an assessment measure. The

risk model is in charge of modeling wireless network risk from the aspects of the security requirements, the wireless attacks and the configurations, where the wireless attacks fall into the category of the external attacks. The assessment measure is an algorithm for determining the risk value based on the risk model. To complement the deficiencies of the existing methods (attack graph-based and AHP-based methods), we extend an existing 3-layer AHP hierarchy into four layers with the considerations of device configurations. An additional layer is constructed to consider the impacts from incorrect configurations and to deal with the frequently changing configuration of a wireless network.

Our 4-layer hierarchy consists of the risk layer (1^{st} layer), the requirement layer (2^{nd} layer), the attack layer (3^{rd} layer) and the configuration layer (4^{th} layer) such that the vulnerabilities, the wireless attacks and the attack targets within a wireless network are considered by our method. The separate layers are advantageous to incorporating the dynamic configurations since only the 4^{th} layer is re-built on detecting the changes of the configurations. Further, since our hierarchy is developed per device, we can easily establish or remove a corresponding hierarchy when a device joins or leaves the network. Only the related hierarchy needs to be developed or removed, instead of all the hierarchies within the network. Therefore, the computing load, resulting from the dynamics of the network, can be reduced. On the basis of the hierarchy per device, we propose an assessment measure to calculate the value for wireless network risk.

In regard of program protection against the internal attacks, it is expected that after applying the protection mechanism, a program is more robust

against being understood or modified by attackers. Software obfuscation is a technique to shield a program from reverse engineering [17, 18, 19, 20, 21]. Collberg et al. [17, 22] classified software obfuscation and proposed several approaches. One approach is control-flow obfuscation, which tries to disguise the real control flow of an original program by re-ordering and obscuring its execution paths. Then, an obfuscated program with higher robustness than the original one is produced. Additionally, software tamperproofing is another well-known program protection mechanism. It not only aims at making tampering difficult but also tries to detect and respond to the modification as well [23]. Obfuscation is beneficial to tamperproofing, since an obfuscated program which is harder to understand increases the difficulty for an adversary to discover the exact software instructions that he would like to tamper. Tamperproofing is usually combined with obfuscation in practice. Therefore, this dissertation focuses on evaluating software obfuscation to analyze its effects upon software robustness. Then the evaluation result can lead to the further measurement of software robustness enhanced by a tamperproofing mechanism.

To evaluate various control-flow obfuscating transformations, we present an abstract framework for formalizing and modeling them. We describe a control-flow obfuscating transformation as a transformation on program control flow graphs (CFG) in this framework. A control-flow obfuscating transformation can be viewed as a function that accepts the original program's CFG as input and yields a modified CFG. By analyzing many existing transformations, we observed that many of them can be decomposed into a sequence of basic building blocks. Thus, we identify a set of atomic

operators for graph transformations that are guaranteed to preserve the functional behavior of the program and hence can be used as building blocks of a control-flow obfuscating transformation. By composing instances of these atomic operators in sequence, we can build many kinds of control-flow obfuscating transformations. This helps to understand and classify many prior control-flow obfuscation proposals and may help in devising new candidate obfuscating transformations.

On the basis of the formal representation of a transformation, we propose metrics that we conjecture may be related to software robustness of an obfuscated program, in comparison with the original program, against reverse engineering. Our framework with such metrics helps to statistically analyze and evaluate software robustness in terms of control-flow obfuscating transformations, while it does not support dynamic analysis of reverse engineering. In addition, we explain how to evaluate the overhead on code size introduced by a control-flow obfuscating transformation on the basis of our framework. Our approach works by characterizing the space penalty of each individual atomic operator. Then, we are able to estimate the overheads an obfuscating transformation yields according to the formalization of the transformation with ease.

The novel contributions of this dissertation are:

- We propose assessment methods of cyber security. The assessment methods concern the scenarios of both external and internal attacks.
- We present an extended AHP-based method for wireless risk assessment. The method models the wireless risk according to the widely

adopted definition of risk, the realistic attacks and the current system configurations. In addition, the method addresses the computing loads caused by the dynamics of a wireless network.

- We show a framework to evaluate software robustness enhanced by control-flow obfuscation. The framework can not only formalize existing control-flow obfuscating transformations but is also flexible enough to express new ones. In addition, our framework is helpful in evaluating not only software robustness but also space penalty caused by obfuscation at the design stage.

- We propose metrics that we conjecture they may be helpful in measuring wireless risk and capability of control-flow obfuscation. We reason the small risk values and the large capability values derived by our metrics are necessary but not sufficient for security. Then, the metrics can be a useful index for administrators to adjust network configurations or select proper protection mechanisms.

Synopsis Chapter 2 introduces the related work of cyber security assessment, including network risk assessment and evaluation of software obfuscation. Chapter 3 explains our risk assessment method which is designed based on the analytic hierarchy process. We also present metrics and a measure algorithm for assessing wireless network risk. In Chapter 4, we first review the background of CFGs, and describe the proposed atomic operators for formalizing control-flow obfuscation. The formalization of control-flow obfuscating transformations is specified in this chapter. Chapter 5 describes

the metrics for evaluating control-flow obfuscation. The metrics are devised based on the proposed formalization. Chapter 6 gives examples to illustrate our assessment methods and to validate our methods. Finally, the last chapter states the conclusions of this dissertation.

Chapter 2

Related Work

We review the existing methods of assessing cyber security in this chapter. We also discuss the advantages and insufficiencies of these methods to clarify the motivation of this dissertation again.

2.1 Security Assessment in terms of External Attacks

In most situations, an adversary has no access to a victim system. The attacker needs to start attacking without a given privilege. He may try to gather useful information by external exploration and to exploit vulnerabilities to gain a privilege illicitly. Attack graph-based methods assess cyber security based on analyzing potential or possible attack paths existing in a network. AHP-based methods focus on modeling security risk yielded by multifarious factors, including various kinds of attacks, system configurations, and so on.

2.1.1 Attack Graph-Based Assessment Methods

Traditionally, tree-based analyses such as event-tree analysis and fault-tree analysis are used in a quantitative risk assessment [24, 25]. The event-tree analysis produces a sequence of outcomes which may arise after the occurrence of a selected initiating event. In the fault-tree analysis, an undesired event is assigned as the root of a fault tree. Administrators deduce bottom events that may trigger the undesired event from top to down, and build a fault tree composed of the events. By traversing the event tree or the fault tree, we can ascertain the probability of occurrence of an undesired event. Both event-tree and fault-tree analyses, while useful, are less than satisfactory since they are not appropriate for assessing risk resulting from multiple criteria. That is because an administrator can select only one undesired event (initiating event) when build up a fault tree (an event tree). Therefore, the risk value deduced from the tree concerns a single criteria simply.

To improve the deficiencies of the tree-based methods, in 1999 Phillips et al. proposed an approach to modeling network risk based on an attack graph [2], which draws paths that may lead to an unexpected state of a network from various initial states. A node in a graph indicates a system state, and an edge is an action of transition from one state to another. An attack graph is generally developed with attack templates, system configurations and attack capabilities [2, 26, 27]. Attack templates mainly describe the pre and post conditions of attacks. The conditions may contain the information of user level, vulnerabilities, capabilities, etc. System configurations indicate the details of the network system. A configuration file should have the fol-

lowing information: machine type, operating system, ports opened, services, network type, and so on. In an attack graph, attack capabilities can be represented as the initial states. The attack capabilities are one of the factors leading to the probability of success of an attack.

Since attack graphs can provide thoroughly possible attack paths within a network, many researchers and professionals have proposed attack graph-based network security measures. Wang et al. [7, 6] presented a generic framework which considers disjunctive and conjunctive dependency relationship between exploits in an attack graph. An attack resistance metric has been proposed to calculate and compare the security of different network configurations based on the generic framework. In [4], Mehta et al. presented two algorithms of ranking attack graphs to determine the probability of an attacker reaching the goal states. The first algorithm is similar to Google's PageRank algorithm to determine the importance of webpages on the World Wide Web [28]. The authors modified Google's algorithm to find out the probability to reach a certain system state from the initial state. The second algorithm ranks individual states of an attack graph in a random simulation that the transition probability from state s_i to s_j equals the reciprocal of the number of successors of the state s_i.

[3] and [26] presented an analysis method of determining a minimal set of attacks that need to be prevented, otherwise the goal state will be reached. They also explained how to interpret an attack graph as a Markov Decision Process to perform quantitative reliability analysis. A number of researchers have proposed risk assessment and security analysis methods based on Bayesian network-based attack graphs [8, 29, 30]. Bayesian networks en-

able system administrators to determine the probability of a particular attack being executed from a given initial system state according to the conditional dependencies among passed states. Dantu et al. [31, 32] also used a Bayesian network-based attack graph for security risk management. The authors integrated behavior-based profiles with the Bayesian network-based attack graph to estimate the risk level based on an attacker's behavior.

The attack graph-based methods are widely used in network security analysis and assessment since an attack graph provides elaborate information about attacks which exploit vulnerabilities existing in a network. However, generation of an attack graph requires high time complexity. In [33], Ou et al. pointed out the complexity of the attack graph generation algorithm of Ammann et al. [34] is $O(N^6)$ in terms of network size. Ou's algorithm has $O(N^2)$ complexity under the assumption of constant table look-up time. In 2005, Ammann et al. [35] proposed an algorithm to track only "good" attack paths, instead of all possible attack paths. The algorithmic complexity is polynomial in the size of the network.

According to the discussion in the literature, complexity of generating an attack graph is a critical issue for the attack graph-based assessment methods. To assess security of a dynamic network environment, redrawing the whole attack graph is required because the paths of an attack graph are tightly dependent on the exploited vulnerabilities and on the nodes. Periodically redrawing an attack graph of a dynamic network, like wireless networks, could lead to a heavy load because topologies and configurations usually change in high frequency.

2.1.2 AHP-Based Assessment Methods

The AHP is a structured approach for solving decision-making problems. It is appropriate for complex decisions which involve various decision elements that are difficult to quantify. The AHP contains the steps in developing a hierarchy of decision elements and constructing the relationship between the elements. A weight is set for each element as the representation of the relationship. The AHP has been applied for many realms, including network risk assessment [9, 10, 11, 12, 13].

Wang and Zeng [12] presented a method of assessing information security risk based on the AHP. They quantified the security risk by integrating the AHP with the fuzzy mathematics and the artificial neural network. Zhang et al. [13] proposed an AHP-based risk assessment method for information security. They adopted a group decision making method to combine the assessing results from individual experts. [12] and [13] concentrate on the design of the methodology and does not mention much about the development of the risk model upon the AHP.

In [9, 10, 11], Zhao et al. constructed 3-layer hierarchical structures based on the AHP to model wireless network risk. The top layer of their structure is the goal of the risk assessment. The middle layer introduces the rules for weighting the risk factors with the aspects of probability, impact severity, and uncontrollability. The combination of these factors leads to a potential risk value of the network. Illegal actions and system faults which may influence the above elements are listed in the bottom layer. In [9], the entropy theory was introduced to determine the coherence of expert experiences. In 2007,

Zhao et al. extended their previous risk assessment method, which was proposed in 2005 [9], by including mobile IP security and wireless interferences in the bottom layer to assess security risk of a wireless network [11].

Compared to the attack graph-based methods, the AHP-based methods require lower time complexity to generate a network risk model. Thus, the AHP could be a convincing candidate basis for modeling and assessing security of a wireless network with changing configurations and dynamic topologies. Moreover, these hierarchical structures, composed of critical elements for the wireless network risk assessment, are useful to systematically measure network security. However, the existing work ([9], [10] and [11]) simply discusses how the risk factors affect network security without considering the impacts resulting from the practical configurations and network topologies. Because incorrect configuration is the main reason for system vulnerability for both wired and wireless networks, the existing 3-layer structures are deficient in modeling network risk.

2.2 Security Assessment in terms of Internal Attacks

Since an internal adversary has access to a victim system, most security mechanisms cannot forbid the adversary from reaching or stealing contents like data or software within the system. Nevertheless, there are some mechanisms which try to obstruct an adversary from understanding, interpreting or modifying the contents even the adversary obtains the contents. Therefore,

security assessment in terms of internal attacks should concern on security evaluation of the corresponding protection mechanisms. Software protection has received comparatively little attention, compared to data protection and evaluation of data protection. This dissertation aims at providing holistic security assessments that we concentrate on devising the evaluation of software protection.

Various software protection mechanisms have been proposed to accomplish distinct objectives. The mechanisms and their objectives are described as follows.

- *Software watermarking* targets on discouraging the intellectual property theft or proving the ownership of the software when the theft occurs by embedding a watermark into the software.
- *Software tamperproofing* tries to increase difficulty in tampering software and is able to detect changes if the software is tampered.
- *Software obfuscation* aims at obscuring software to protect the software from being understood or reverse engineering.

Software watermarking and software tamperproofing are generally combined with software obfuscation since an obfuscated program which is harder to understand increases the difficulty for an adversary to figure out the embedded watermarks, or to discover the exact software instructions that he would like to tamper. Therefore, the result of evaluating obfuscation can not only indicate the capability of software obfuscation but also be introduced to further security assessment of software watermarking and tamperproofing.

It is crucial to dive in the evaluation of software protection starting from evaluating obfuscation.

2.2.1 Evaluation of Software Obfuscation

Software obfuscation increases difficulty in reverse engineering by transforming an original program into an obfuscated one which thwarts reverse engineering but preserves the original functionality [17]. Despite the theoretic proof of the impossibility of omnipotent obfuscation [36], obfuscation is still able to reach positive results in specific situations [37] and implementation of obfuscation have been widely discussed [17, 18, 19, 20, 21]. According to [17], obfuscation is classified into three types: control-flow obfuscation, data obfuscation and layout obfuscation. Control-flow obfuscation disguises the real execution under scrambled control flow of a program to make reverse engineering difficult. Various implementations have been introduced to accomplish control-flow obfuscation [18, 38, 39, 40]. Data obfuscation transforms data and data structures in a program without modifying the original functionality. [21] and [41] presented obfuscating transformations by extending the concept of data obfuscation. Layout obfuscation removes the information that an attacker can seize from the program. Most of the commercial obfuscators such as Dotfuscator [42], DashO [43], Zelix [44] and ProGuard [45] adopt the basic idea of layout obfuscation.

Each type of obfuscation provides effective though limited resistance against malicious reverse engineering. In recent years, many researchers proposed various evaluation methods to assess the effectiveness of an obfuscating trans-

formation. The methods are mostly based on empirical analysis, which evaluates an obfuscating transformation by running practical experiments to observe how much the obfuscated program resists against deobfuscators or how much time a human subject takes to interpret it [46, 47, 48, 49, 50, 51].

Udupa et al. [52] examined control flow flattening, a control-flow obfuscating transformation, by measuring the time required by automatic deobfuscation. Anckaert et al. [47] introduced a framework to evaluate an obfuscating transformation based on software complexity metrics, which calculate the complexity with respect to instructions, control flow, data flow and data. The authors implemented three obfuscating transformations (control flow flattening, static disassembly thwarting and binary opaque predicates) and applied the transformations to eleven C programs of the SPECint 2000 benchmark suite, and the obfuscated programs were produced from the benchmark suite. The results of the complexity analysis show that all of the three transformations can provide non-negative effects, but the transformation, binary opaque predicates, is less potent than two others. Majumdar et al. [48, 53] considered a specific reverse-engineering technique, slicing, and developed metrics to evaluate the capability of obfuscation against that technique. [48] and [53] presented three obfuscating transformations (bogus predicate, adding to a `while` loop, and variable encoding) and applied them to five example programs to derive the values by the defined metrics. The metric values imply that these transformations significantly make reverse engineering difficult.

Ceccato et al. [49, 50] assessed the difficulty an attacker would encounter in examining identifier renaming, one of the obfuscation techniques, by ques-

tionnaires. The authors asked human subjects to interpret the original and obfuscated programs and to fulfill a comprehension task. The subjects were also asked to fill in a post-task survey questionnaire to describe their behavior during the task and the confidence about it. Certain types of statistical tests, such as the Mann-Whitney test and the Wilcoxon test, were adopted to analyze the task results and the questionnaires. The analysis results point out that identifier renaming effectively reduces the capability of the subjects to understand the source code.

The existing work [47, 48, 49, 50, 52] evaluated the effectiveness of obfuscation by empirical studies. Practical experiments were performed to measure individual obfuscating transformations according to the defined metrics or the perception of human subjects. These experiment results indicate the relation specifically between a designated original program and a single obfuscating transformation. While the same obfuscating transformation is intended to be applied to another program, the experiment results may not be applicable to determine the capability of that transformation in the case. In addition, the existing experiment results of evaluating individual transformations cannot help determine the effectiveness of a compound obfuscating transformation, which comprises several separate transformations. It thus requires a formal method for evaluating obfuscating transformations in a high-level of abstraction.

Preda and Giacobazzi [54] proposed a formal method for analyzing the effect of a control-flow obfuscating transformation based on program semantics. They considered a specific control-flow obfuscating transformation, which obscures the control flow by inserting opaque predicates. They eval-

uated the transformation by analyzing the effects of the opaque predicates inserted. They also modeled attackers for comparing obfuscating transformations. Their method is the closest to ours, which evaluates control-flow obfuscating transformations based on formal analysis as well. However, our method can formalize and evaluate more types of control-flow obfuscating transformations, not limited to the type of inserting opaque predicates.

2.3 Summary

We reviewed and analyzed the attack graph-based and the AHP-based methods for network risk assessment in terms of external attacks. The analysis showed that there is a need to propose a new risk assessment method, which can represent the risk in the real-world and is capable of addressing the dynamics of a network. We also discussed the existing methods of evaluating obfuscation. Most of them are empirical-based and examine the effects of obfuscating transformations by experiments; however, a formal method is necessary to help system administrators systematically and effectively assess the capability of obfuscation.

Chapter 3

Risk Assessment of Wireless Networks

Risk assessment is critical for risk mitigation and security enhancement. It can be applied to several different realms to address risk management, such as information technology, chemical industry and financial industry. Despite variation in the application realms, risk assessment takes into account calculations of two components of risk, the magnitude of the potential loss and the probability that the loss will occur. Then the assessment result is used as a reference for identifying proper controls in treating or eliminating risk during the following process, such as risk treatment and risk mitigation, in a security risk management standard [55, 56, 57].

The dynamics of wireless networks make network security evaluation and management a critical challenge. To help a network administrator effectively manage wireless network security, it is essential to design a risk assessment method which derives a risk value for the administrator to easily understand

the security of the managed network. A feasible risk assessment method has to reasonably model wireless network risk and measure the risk value according to the characteristics of the network. Network risk is defined as "*a function of the likelihood of a given threat-source's exercising a particular potential vulnerability, and the resulting impact of that adverse event on the organization*" [55]. According to the definition, network risk can be interpreted as the resulting impact which results from the likelihood, the threat sources and the vulnerabilities.

To fulfill the definition, we propose a risk model (4-RAH), shown in Figure 3.1, to describe the wireless network risk. The top layer of our model represents the impact severity which threatens the security requirements (2^{nd} layer) of a wireless network. The impact severity is determined in terms of the factors: likelihood, threat sources and vulnerabilities by the definition. Our model introduces the attack layer (3^{rd} layer) and the configuration layer (4^{th} layer) to indicate the threat sources and the vulnerabilities, respectively, where the edges between the layers represent the likelihood mentioned in the definition. We construct a hierarchy for each device, and then based on the hierarchy, we propose an assessment measure which contains a newly defined historical vulnerability metric and an algorithm to determine the network risk value. Our risk assessment takes the real-world situation into account and the evaluated result helps an administrator understand a network's weak points and their impacts. Therefore, our risk assessment can be a useful reference in designating security policy and improving network security.

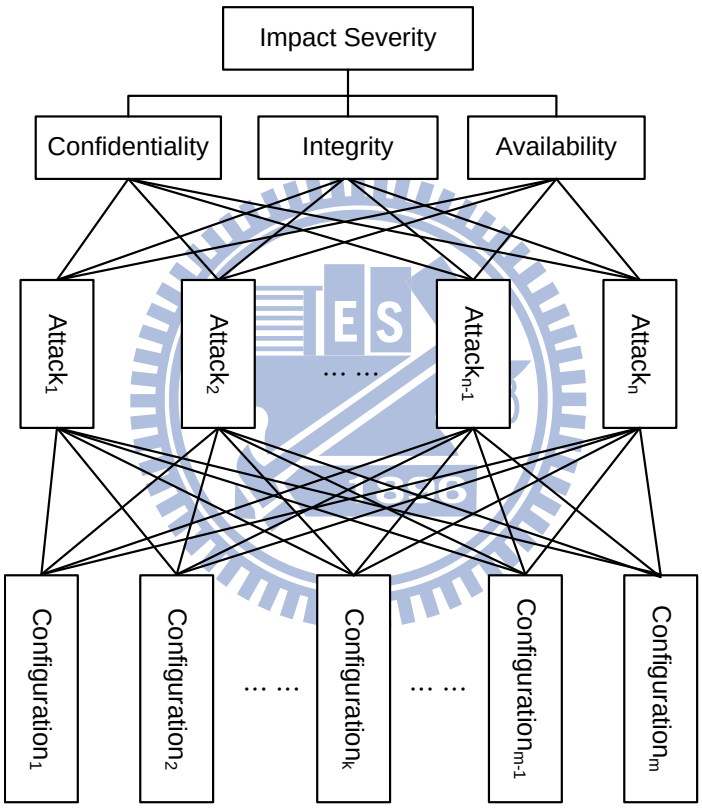


Figure 3.1: The proposed hierarchy per device: general case

3.1 Preliminaries

This section defines the symbols used in Chapter 3.

α_i	Severity of the i^{th} vulnerability
β	Decaying speed of the exponential function
λ_i	Age of the i^{th} vulnerability
A_i^{ap}	i^{th} attack targeting on an access point (AP)
A_i^{sta}	i^{th} attack targeting on a wireless station (STA)
$ahvm(dev_i)$	Value of the i^{th} device (dev_i), determined by the aggregated historical vulnerability measure (AHVM)
D	Degree matrix of a given device. The matrix dimension is n_a-by-n_r. The entry d_{ij} is used to represent the impact that the i^{th} attack A_i imposes on the j^{th} security requirement.
$hvm(ser_i)$	Value of the i^{th} service, determined by the historical vulnerability measure (HVM)
$\overline{hvm}(ser_i)$	Normalized $hvm(ser_i)$
I	Impact severity upon a device
$ihvm(dev_i)$	Value of the i^{th} device, determined by the integrated historical vulnerability metric (IHVM)
$\overline{ihvm}(dev_i)$	Normalized $ihvm(dev_i)$
n_a	Number of attacks
n_a^{ap}	Number of attacks targeting on APs
n_a^{sta}	Number of attacks targeting on STAs
n_d	Number of wireless devices in a network
n_d^{ap}	Number of APs in a network

n_d^{sta}	Number of STAs in a network
n_g	Number of configurations
n_r	Number of security requirements
n_s	Number of services running on a device
n_v	Number of vulnerabilities of a service
$\hat{\mathbf{p}}$	Probability vector. The i^{th} entry p_i denotes the probability of acquiring the i^{th} configuration.
$\hat{\mathbf{r}}$	Risk level vector. The i^{th} entry r_i reflects the help that a captured configuration may offer to an attacker.
T	Total impact severity upon a wireless network
$\hat{\mathbf{w}}_\mathbf{g}$	Weight vector of configurations, an n_a-dimension column vector. The i^{th} entry w_{g_i} reveals the impact leading to the i^{th} attack A_i, where the impact varies with the configurations of a wireless system.
$\hat{\mathbf{w}}_\mathbf{r}$	Weight vector of requirements. The vector is an n_r-dimension column vector. The i^{th} entry w_{r_i} represents the weight of the i^{th} security requirement when deriving the total impact severity.

3.2 Risk Model: Four-Layer Risk Analytic Hierarchy

To accomplish the definition of network risk [55], 4-RAH is proposed to model the wireless network risk with four layers: risk, requirements, attacks and configurations, respectively.

3.2.1 Risk Layer

The first layer (risk layer) only contains a root node, representing the impact severity of a wireless network as the security requirements of the network are not achieved.

3.2.2 Requirement Layer

We introduce the credible network security requirements, confidentiality, integrity and availability, into the 2^{nd} layer of 4-RAH.

- *Confidentiality* is imperiled when information is available or disclosed to unauthorized users. Different attacks aim for different targets. For instance, an eavesdropping attack launches impacts on network traffic confidentiality, while a penetration attack causes damage to memory data confidentiality. In this paper, loss of confidentiality can occur in multifarious targets according to the types of attacks.

- *Integrity* is damaged if data or messages are executed, modified, suspended, copied, replayed or deleted by an illicit user. Because attackers may be interested in attacking different targets such as network traffic or memory data, the integrity mentioned in this dissertation varies with the types of attacks.

- *Availability* mainly focuses on whether a service operation is affected by an attack, or whether an authorized user can access a network service they should. The availability mentioned in this dissertation is endangered if the service or server is spoofed, penetrated, or suspended, and

cannot operate as expected.

3.2.3 Attack Layer

In 4-RAH, the third layer (attack layer) represents attacks which may damage the security requirements listed in the second layer. An attack may pose different impacts on different security requirements, which have specific concerns on various targets, such as bandwidth, network traffic, programs, or computers. The targets may suffer different risks even though they are under the same attack. Taking a beacon flood attack as an example, the attack succeeds when targeting on the bandwidth, but fails if it intends to attack a program. In our model, the attack layer analyses the attacks, not only in terms of their behavior, but also the impacts with respect to the attack targets, and the security requirements. In addition, the impact varies with the sequence of attacks. Because the impacts of attacks are dependent on the sequence in which they are carried out, we define two types of impacts to express the relationship in the attacking sequence: direct, and indirect.

- *Direct impact*: the impact lays on the security requirements initially targeted by an attack.
- *Indirect impact:* the impact is a side effect accompanied by the direct impact from the previous attack.

For example, an eavesdropping attack imperils traffic confidentiality by maliciously sniffing wireless network packets. It poses the direct impact upon traffic confidentiality, and no direct impact on other targets, such as a file or a program. The packets sniffed by an eavesdropper can become a requirement

for a subsequent attack, such as a replay attack, and thus further endangers traffic integrity. Hence, an eavesdropping attack results in the indirect impact on traffic integrity. When evaluating the impacts caused by an attack, the union of direct and indirect impacts should be considered.

After analyzing the existing wireless attacks, we categorize wireless attacks into five types, including scan or monitor, masquerade, Denial of Service (DoS), key cracking, and penetration attacks, with respect to their behavior and intentions.

- Type I: Scan or Monitor attacks

  Scan attacks intend to search for accessible wireless networks. The monitor attacks aim at gaining useful, critical information of a victim network by intercepting aerial packets, and analyzing network traffic. Such kind of attacks includes war driving, eavesdropping, active scan attacks, etc. Because Type I tries to obtain critical information, most of the attacks of this type directly impact network traffic confidentiality.

- Type II: Masquerade attacks

  An attacker masquerades as a legitimate user to access a wireless network, or as a legitimate device to pirate network traffic or disable a functioning access point (AP). Once the attacker has snatched the identity of a victim successfully, the victim can no longer access the network, or the attacker can then provide network service to other illicit users. Thus this type of attack directly impacts availability. With the counterfeit identity, the masqueraded user can easily capture or reach private information so that confidentiality and integrity are usu-

ally threatened as well.

- Type III: DoS attacks

  Denial of Service (DoS) attacks aim at making computers or network resources unavailable to legitimate users. Attackers take advantage of the paralysis period to launch other attacks. Then, they can devastate the network security severely. Because service requests are denied under this type of attack, the direct impact is against availability.

- Type IV: Key cracking

  Key cracking attacks try to recover WEP [58, 59] or WPA [60] keys by analyzing numerous packets. After cracking the protection keys, all requirements (confidentiality, integrity, and availability) are harmed.

- Type V: Penetration attack

  This kind of attack attempts to penetrate a victim system through system vulnerabilities. After the success of the attack, the attacker can control the files, the programs, even the computer such that data confidentiality, data integrity, or service availability may be destroyed. All three security requirements are threatened under this type of attack.

3.2.4 Configuration Layer

To launch some attacks toward a wireless network, an attacker needs to obtain certain network information or device configurations, such as IP addresses of wireless stations (STA) or APs, Multimedia Access Control (MAC) addresses of STAs or APs, Service Set Identifiers (SSIDs), wireless channels,

OS versions, running services, etc. In 4-RAH, the 4^{th} layer (configuration layer) exhibits configurations of wireless devices and wireless networks. The following paragraphs discuss some configurations required to launch certain attacks. More configurations can be added to this layer when needed.

- IP address is one of the prerequisite configurations for an attacker to identify a victim in an IP network. Attacks of Type II, III, and V require such a configuration.
- MAC address is one of the configurations required to identify the physical address of a victim. Attacks of Type II, III, and IV require this configuration.
- SSID is one of the prerequisite configurations when an attacker attempts to connect or scan a specific wireless local area network. Attacks of Type II, III, and IV need this configuration.
- Wireless channel is one of the configurations required to launch key cracking attacks. Attacks of Type IV require such a configuration.
- OS version is one of the configurations required to obtain the possible vulnerabilities of a victim. Type V attacks require this configuration.
- Running services and open ports are useful configurations to penetrate a victim. Type V attacks need this configuration.

Table 3.1 lists the five attack types, and the relations with the security requirements and prerequisite configurations. Note that an attacker can start Type I attacks without prerequisite configurations, though the performance

Table 3.1: Types of Attacks

Types	Impacts		Prerequisite configurations	Attacks
	Direct	Indirect		
I	C	I, A	None	War driving, eavesdropping, etc
II	C, I, A	-	STA IP, AP IP, STA MAC, SSID, etc	Evil twin, IP spoofing, TCP hijacking, etc
III	A	-	STA MAC, AP MAC, SSID, etc	Beacon flood, association flood, etc
IV	C, I, A	-	AP MAC, SSID, channel, etc	WEP/WPA key cracking
V	C, I, A	-	STA IP, ports, running services, etc	Penetration attack, etc

C: confidentiality I: integrity A: availability

of the attacks can be enhanced if the attacker obtains more network configurations.

3.3 Integrated Historical Vulnerability Metric

In our risk assessment method, we define an integrated historical vulnerability metric (abbreviated to IHVM), evolving from the historical vulnerability measure (HVM) and the aggregated historical vulnerability measure (AHVM) proposed in [61], to determine the risk value of a device based on existing vulnerabilities.

3.3.1 HVM and AHVM

HVM measures the risk level of a service imposed by vulnerabilities of the service, and weights the vulnerabilities in terms of their ages [61]. The authors of [61] assumed that a vulnerability discovered a long time ago should take a small weight because the vulnerability may be understood and patched with a high probability as time passes by. Therefore, the age of a vulnerability is introduced in the decaying function of Eq. 3.1. [61] showed that $hvm(ser)$ can imply the probability that service ser will become vulnerability-prone in the future.

$$hvm(ser) = \ln\left(1 + \sum_{i=1}^{n_v} \alpha_i \times \exp\left(-\beta \times \lambda_i\right)\right). \tag{3.1}$$

α_i and λ_i indicate the severity and the age of the i^{th} vulnerability, and β denotes the decaying speed of the exponential function.

Not all of the vulnerabilities of service ser should be counted because the vulnerability effect usually declines with age, approaching zero. If only the latest n vulnerabilities of service ser are considered, then we can derive $\overline{hvm}(ser)$ by $hvm(ser)$, as represented in (3.2).

$$\overline{hvm}(ser) = \frac{hvm(ser)}{\ln\left(1 + 10 \times n\right)}, \text{ where } 0 \leq \overline{hvm}(ser) \leq 1. \tag{3.2}$$

A combination of $hvm(ser_i)$ for all services running on a device dev is defined by the AHVM [61]. AHVM is useful in calculating the vulnerability

threats that a device dev faces.

$$ahvm(dev) = \ln\left(\sum_{i=1}^{n_s} \exp\left(hvm\left(ser_i\right)\right)\right), \text{ for all services } ser_i \text{ running on } dev. \tag{3.3}$$

However, if there is no vulnerability detected in dev, AHVM outputs an undefined value, $\ln 0$. To address such an error, a new metric (IHVM) is proposed with our four-layer risk assessment model.

3.3.2 IHVM

IHVM is proposed to ensure the existence of the boundary values. In this metric, the notation $ihvm(dev)$ represents the value for a device dev, calculated by IHVM, while $\overline{ihvm}(dev)$ stands for the normalized $ihvm(dev)$.

$$ihvm(dev) = \ln\left(1 + \sum_{i=1}^{n_s} \exp\left(\overline{hvm}\left(ser_i\right)\right)\right). \tag{3.4}$$

All services ser_i running on the device dev contribute to $ihvm(dev)$. The number of services is denoted by n_s. The higher $ihvm(dev)$ implies that the running services may contribute more severity to the device dev. If no service is running on dev, then $ihvm(dev)$ will be set to 0.

After sorting $\overline{hvm}(ser_i)$, $\forall$ service ser_i running on dev, if we only consider the top m highest $\overline{hvm}(ser_i)$, then the maximum $ihvm(dev)$ becomes $\ln\left(1 + m \times \exp\left(1\right)\right)$. So, we can obtain the risk level of a single device $\overline{ihvm}(dev)$ according to the service vulnerabilities by Eq. 3.5.

$$\overline{ihvm}(dev) = \frac{ihvm(dev)}{\ln(1 + m \times \exp(1))}. \tag{3.5}$$

As a result, $\overline{ihvm}(dev)$ falls into the range $[0, 1]$.

3.4 Risk Assessment Algorithm

This section explains the algorithm of our assessment measure and represents a step-by-step progress toward the wireless network risk.

Step 1. Establish risk model.

Initially, an administrator needs to build up a 4-RAH, and generate degree matrices (D) of devices within a wireless network by investigating possible attacks.

Step 2. Develop experience mapping tables.

Because mobile wireless devices have certain sociological orbit, the security requirements and risks may differ by the position of a sociological orbit. This step intends to introduce expert experiences to adjust factors, and to achieve scenario-adaptive assessment.

To provide a fair or even close to fair assessment, multiple experts could be consulted, and several databases can be imported. In 2005, Zhao et al. proposed a method to evaluate the consistency of expert opinions by the entropy theory [9] . In our method, once an administrator develops the experience mapping tables, experts could be consulted

to approve the experiences shown in the tables. Because the degrees of approval may be categorized into several levels, the consistency of the degrees should be further evaluated. If all the experts show the same degree level of approval, the consistency reaches the maximum. On the contrary, the consistency reaches the minimum if the degree levels distribute equally. In the end, an administrator can obtain the weighted importance from the consistency.

Step 3. Assess network risk.

This step can be further decomposed into several sub-steps.

1. Specify probability vector $\hat{\mathbf{p}}$, and risk level vector $\hat{\mathbf{r}}$.

   According to network configurations, expert experiences, and vulnerability databases, we obtain $\hat{\mathbf{p}}$, and $\hat{\mathbf{r}}$, where $\hat{\mathbf{p}}$ relies on the encryption method used in a wireless network, and $\hat{\mathbf{r}}$ is determined with three aspects: 1) adoption of a default value of the configuration, 2) the number of attacks that view the configuration as a prerequisite, and 3) the $\overline{ihvm}$ value for the configuration of "running services."

2. Determine weight vector of configurations $\hat{\mathbf{w}}_{\mathbf{g}}$.

   We can obtain the i^{th} entry of $\hat{\mathbf{w}}_{\mathbf{g}}$ by Eq. 3.6. Each entry w_{g_i} reveals the impact leading to the i^{th} attack A_i, where the impact

varies with the configurations of a wireless system.

$$w_{g_i} = \frac{\sum_{j=1}^{n} r_j \times p_j}{n}. \tag{3.6}$$

where n indicates the number of configurations. If no prerequisite configuration is required, w_{g_i} is set to 1, which is the maximum weight.

3. Determine weight vector of requirements $\hat{\mathbf{w}}_\mathbf{r}$.

   We determine the value of each entry of $\hat{\mathbf{w}}_\mathbf{r}$ in terms of the functionalities of a device. For example, the "availability" of an access point should have a heavier weight than "confidentiality" and "integrity" because the AP is in charge of providing Internet access for wireless devices. $\hat{\mathbf{w}}_\mathbf{r} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \end{bmatrix}^T$.

4. Determine impact severity upon a device I.

   Because the security of a device may suffer more as the number of attacks that pose interests to the device raises, the range of the impact severity upon a device (I) is designed based on the number of attacks (n_a) targeting on a device *dev*. We then obtain the impact severity of the device as

$$I = \hat{\mathbf{w}}_\mathbf{g}{}^T \times D \times \hat{\mathbf{w}}_\mathbf{r}. \tag{3.7}$$

   Because entries of $\hat{\mathbf{w}}_\mathbf{g}$, D, and $\hat{\mathbf{w}}_\mathbf{r}$ all fall within $[0, 1]$, and the

summation of all entries of $\hat{\mathbf{w}}_\mathbf{r}$ equals 1, I falls within $[0, n_a]$.

5. Calculate total impact severity upon a wireless network T.

   Because any device in a network may jeopardize the network security, we accumulate the contribution of each device towards the total impact severity (T) by Eq. 3.8.

$$T = \log_{10}\left(\sum_{i=1}^{n_d} 10^{I^{dev_i}}\right) \tag{3.8}$$

   Because a compromised device or a device with weak configurations is usually viewed as a stepping stone by an attacker to propagate attacks, the maximum I dominates the result of Eq. 3.8 while the other smaller values are also introduced. We conjecture that the value of T increases as the network becomes risky.

   T, which depends on the number of devices and their configurations, varies with different network topologies. If there are more devices within a network, the possible maximum value of T becomes larger. If there are n_d^{ap} APs and n_d^{sta} STAs in a wireless network, T then falls within

$$[\log_{10}(n_d^{ap} + n_d^{sta}),\ \log_{10}\left(n_d^{ap} \times 10^{n_a^{ap}} + n_d^{sta} \times 10^{n_a^{sta}}\right)].$$

   It might be difficult for an administrator to interpret a linguistic meaning from a numerical value of T since T is dynamic with the variation of n_d^{ap}, n_d^{sta}, n_a^{ap}, and n_a^{sta}. In spite of the dynamics in a possible value of T, the range of T offers a scale to help grasp the linguistic meaning. An administrator may be able to comprehend

the level of risk (relatively) easily if there is additional information about the scale. Therefore, $\frac{T}{[\log_{10}(n_d^{ap}+n_d^{sta}),\log_{10}\left(n_d^{ap}\times 10^{n_a^{ap}}+n_d^{sta}\times 10^{n_a^{sta}}\right)]}$ could be a solider index of risk.

Besides, we also devise a referable mapping table between a linguistic meaning and a numerical value of T. We first calculate the maximum impact severity of devices in a network, and then define the thresholds for low, medium, and high threats. If all the devices have their impact severity with the maximum value, then we conjecture in such a situation that the network is undoubtedly unreliable, and absolutely insecure. However, not all the networks require such a strict condition. If a very strict condition is set, an administrator may over-ignore unexpected events, and may not deal with the wrong configurations in real-time. Hence, both the ratio of the maximum value of the total impact severity and the ratio of the number of all the devices have to be contemplated for a plausible mapping. The mapping between the numerical risk values and the semantic risk levels is suggested as shown in Table 3.2. The numerical thresholds can be adjusted according to an administrator's expertise, experience, or sociological orbits if needed.

6. Refresh the topology snapshot.

   If new devices or new configurations are detected, the topology snapshot should be refreshed. In our method, it is not necessary to re-calculate the corresponding values of all devices. An

Table 3.2: Numerical Impact Severity vs. Linguistic Meanings

Numerical impact severity (T)	Linguistic meanings (Threats)
$\left[\log_{10}\left(\frac{2n_d^{ap}}{3}\cdot 10^{\frac{n_a^{ap}}{2}}+\frac{2n_d^{sta}}{3}\cdot 10^{\frac{n_a^{sta}}{2}}\right),\log_{10}\left(n_d^{ap}\cdot 10^{n_a^{ap}}+n_d^{sta}\cdot 10^{n_a^{sta}}\right)\right]$	High (insecure)
$\left[\log_{10}\left(\frac{n_d^{ap}}{3}\cdot 10^{\frac{n_a^{ap}}{2}}+\frac{n_d^{sta}}{3}\cdot 10^{\frac{n_a^{sta}}{2}}\right),\log_{10}\left(\frac{2n_d^{ap}}{3}\cdot 10^{\frac{n_a^{ap}}{2}}+\frac{2n_d^{sta}}{3}\cdot 10^{\frac{n_a^{sta}}{2}}\right)\right)$	Medium
$\left[\log_{10}\left(n_d^{ap}+n_d^{sta}\right),\log_{10}\left(\frac{n_d^{ap}}{3}\cdot 10^{\frac{n_a^{ap}}{2}}+\frac{n_d^{sta}}{3}\cdot 10^{\frac{n_a^{sta}}{2}}\right)\right)$	Low (secure)

administrator simply executes the sub-steps 3.1 through 3.5 to determine the impact severity upon changing devices, such as the device newly entering the network, and the device whose configurations have been changed. Then, sub-step 3.6 is performed to re-calculate the total risk of the wireless network.

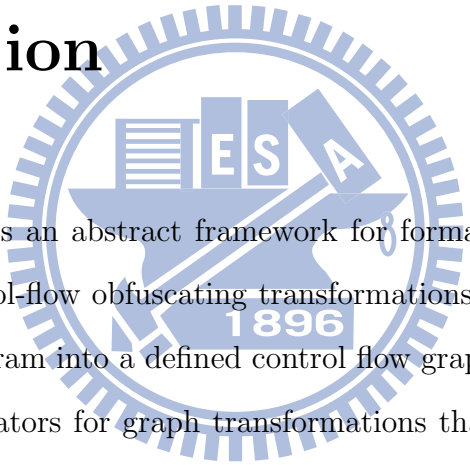
3.5 Summary

In Chapter 3, we presented a risk assessment method for wireless networks. We described the design of a risk model and explained a newly proposed metric (IHVM). We also introduced a risk assessment algorithm to measure the risk value of a network. The risk model and the algorithm are designed to address the dynamics of a wireless network. Not all layers of the risk model or not all steps of the algorithm are re-generated and re-calculated when changes occur in the network. The idea underlying our risk assessment

method help produce a real-time reference for a system administrator to manage network security.

Chapter 4

Formalization of Control-Flow Obfuscation



This chapter presents an abstract framework for formalizing and modeling many kinds of control-flow obfuscating transformations. In this framework, we first parse a program into a defined control flow graph. Then we identify a set of atomic operators for graph transformations that are guaranteed to preserve the functional behavior of the program. These operators can thus be used as building blocks of a control-flow obfuscating transformation. By composing instances of these atomic operators in sequence, we can formalize many kinds of existing control-flow obfuscating transformations and devise new candidate obfuscating transformations.

4.1 Preliminaries

This section defines the notations used in Chapter 4.

Notations

$\xi(C_i)$	Equivalent block of the i^{th} code block C_i
τ	Control-flow obfuscating transformation
ϕ	Termination block
ψ	Parsed program
B_i	The i^{th} branch
B_{ij}	The j^{th} piece split from the branch B_i
C_i	The i^{th} code block. A code block can be a branch, a fork, a join or a simple block.
C^E	The entry point of a parsed program
C_{ij}	The j^{th} piece split from the code block C_i
C^A	Any code block
C^D	Dummy code block
C^{False}	False target of a branch
C^T	Target code block of an atomic operator
C^{True}	True target of a branch
E	Edge set of a directed graph
F_i	The i^{th} fork
G	Directed graph
J_i	The i^{th} join
O	Atomic operator for control-flow obfuscation
O_D^L	Atomic operator of inserting dummy loops
O_D^S	Atomic operator of inserting dummy simple blocks
O_E	Atomic operator of replacing a code block with its equivalent code

O_F^n	Atomic operator of inserting folks, where n represents the number of code blocks expected to be run in parallel
O_G	Atomic operator of inserting folk edges
O_{Op}^F	Atomic operator of inserting type I opaque predicates
O_{Op}^T	Atomic operator of inserting type II opaque predicates
$O_{Op}^?$	Atomic operator of inserting type III opaque predicates
O_R	Atomic operator of reordering code blocks
$O_S^{S,n}$	Atomic operator of splitting a simple block into n pieces
$O_S^{B,n}$	Atomic operator of splitting a branch into n pieces
P^F	Type I opaque predicate. It is an obscure branch, which always evaluates false.
P^T	Type II opaque predicate. It is an obscure branch, which always evaluates true.
$P^?$	Type III opaque predicate. It is an obscure branch, which sometimes evaluates false and sometimes true.
S_i	The i^{th} simple block
S_{ij}	The j^{th} piece split from the simple block S_i
V	Vertex set of a directed graph

4.2 Control Flow Graphs

Control flow graphs (CFGs) were developed by Cota et al. [62, 63] as a representation of the control flow structure of a program, thus can help an analyzer

understand the program easily [64, 65]. In this dissertation we use CFGs to facilitate the formalization of control-flow obfuscating transformations. As a high-level abstraction, a program can be parsed into a directed graph whose vertices are code blocks of the program. There is an edge between two code blocks if the second code block can be executed immediately after the first.

This dissertation considers both sequential and parallel programs, so a code block in a CFG can be defined by our program parser as one of the followings:

- Branch: A branch refers to an instruction that can cause execution to transfer, either conditionally or unconditionally, to some statement other than the immediately following statement. In high-level programming languages, branch instructions may be found in `for`, `while`, `do-while`, `if-else`, and `goto` statements.

- Fork: A fork is the code block that creates parallel execution. The immediate successors of a fork can run concurrently until the paths converge.

- Join: A join is the code block at which parallel execution paths converge.

- Simple block: A simple block is defined as an ordered sequence of statements with no outgoing or incoming branch, fork or join instructions inside this code block.

We use the following notations for several special kinds of code blocks.

- Equivalent block ($\xi(C_i)$): a code block that is functionally equivalent to the code block C_i.

- Termination block (ϕ): the exit point of a source program.

The edges in a CFG represent possible execution paths that the program may take. Our program parser also specifies the following types of edges.

- Sequential edge: A sequential edge, denoted by (C_i, C_j), exists between two code blocks C_i and C_j. Here, C_i can be only a simple block or a join.

- Branch edge: Since a branch B_i may jump to either its true or false target, there are two code blocks that could be executed immediately after the branch. The two branch edges leaving B_i are denoted by $\left(B_i, C^{True}\right)^T$ and $\left(B_i, C^{False}\right)^F$. C_j is executed while B_i evaluates to true, so C^{True} represents the true target of B_i. Similarly, C^{False} is the false target of B_i.

- Fork edge: Since several code blocks can be executed concurrently right after a fork F_i, there may be several code blocks as the immediate successors of F_i. A fork edge is represented as (F_i, C_j), and C_j can be a simple block, a branch or a fork.

With the definitions of the code blocks and the edges, we represent a directed graph by the pair (V, E) where V is the vertex set and E is the edge set. V contains all the code blocks of a program, including simple blocks, branches, forks, joins and a termination. E is composed of sequential edges, branch edges and fork edges. Then, a parsed program ψ is a pair of an entry

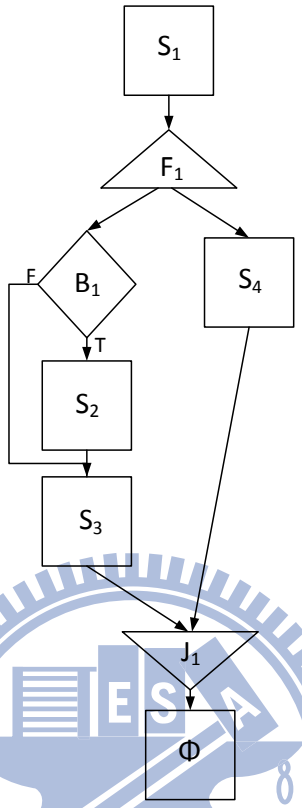


Figure 4.1: Example of the formalization of a parsed program

block of a directed graph and the graph. Figure 4.1 shows an example of a CFG of ψ. The CFG contains four simple blocks, one branch, one fork and one join. A rectangular indicates a simple block; a diamond denotes a branch; a base-down triangular and a base-up triangular represent a fork and a join, respectively. In Figure 4.1, S_1 is the entry block, so we obtain a parsed program $\psi = (S_1, G)$, where $G = (V, E)$, $V = \{S_1, S_2, S_3, S_4, B_1, F_1, J_1, \phi\}$, and $E = \{(S_1, F_1), (F_1, B_1), (F_1, S_4), (B_1, S_2)^T, (B_1, S_3)^F, (S_2, S_3), (S_3, J_1), (S_4, J_1), (J_1, \phi)\}$. ϕ is an indication of the end of execution path without code existing in this vertex. Hence, this type is not counted into the number of vertex set V.

4.3 Atomic Operators

A program graph is a complete representation of a source program. Obfuscating control flow of a program can be viewed as converting one program graph to another. For graph conversion, we can use *deletion*, *insertion* and *update*. With deletion, a vertex or an edge is removed. As deletion always alters the functionality of the original code if there is no other following action (insertion and update), we do not use deletion for program obfuscation. Addition inserts additional edges or vertices, and update means to modify the existing vertices or edges in the graph. Although addition and update may also change the execution result, dummy or redundant codes can be used to maintain the original functionality. Therefore, control-flow obfuscation may involve two classes of operators: insertion and update. We describe these two sets of atomic operators, called "operators" and denoted by "O" hereafter. Since a control flow graph consists of nodes and edges, the sets of atomic operators can be further classified into four categories: insertion of nodes, insertion of edges, update of nodes and update of edges. These categories cover all the possible atomic operators for control-flow obfuscation.

4.3.1 Insertion

Insertion of Nodes

Here we define four operators of insertion according to the types of the code blocks. To insert simple blocks without affecting the original functionality, we can insert dummy blocks that do nothing but resemble real code. Inserting branches can be realized by inserting opaque predicates. Regarding the

operator of inserting forks, a pair of a fork and a join is inserted. We also devise an additional operator of inserting dummy loops to help generate a more obscure control flow easily.

Insert Dummy Simple Blocks The insertion of dummy code blocks changes the control flow of a source program. Figure 4.2 exhibits the operator O_D^S representing the insertion of a dummy simple block C^D in front of the target code block C^T. The graph on the right-hand side is the obfuscated CFG, which represents a result after applying $O_D^S(\psi, C^T)$ to C^T in ψ. In ψ, all edges whose successor or true/false target is C_T would be replaced. An additional sequential edge (C^D, C^T) is also inserted to the edge set E. Algorithm A.1 describes the steps to achieve $O_D^S(\psi, C^T)$.

Algorithm A.1

Inert Dummy Simple Blocks, $O_D^S(\psi, C^T)$

$V \leftarrow V \cup \{C^D\}$;
IF C^T **is the entry point THEN**
  Replace the entry point with C^D**;**
END IF;
Replace edges (C^T, C_i) **with** (C^D, C_i) $\forall i$**;**
Insert (C^D, C^T) **to E;** □

Insert Opaque Predicates An opaque predicate is a Boolean valued expression whose value is known a priori to an obfuscator but is difficult for a deobfuscator to deduce [17, 66]. These opaque predicates can be categorized into three types [17]: a type I opaque predicate always evaluates to false; a

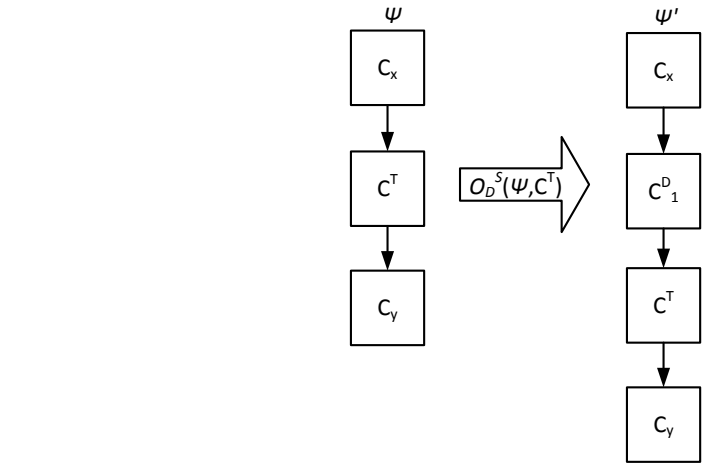


Figure 4.2: Atomic operator of inserting a dummy simple block. After insertion, E becomes $\{(C_x, C_1^D), (C_1^D, C^T), (C^T, C_y)\}$.

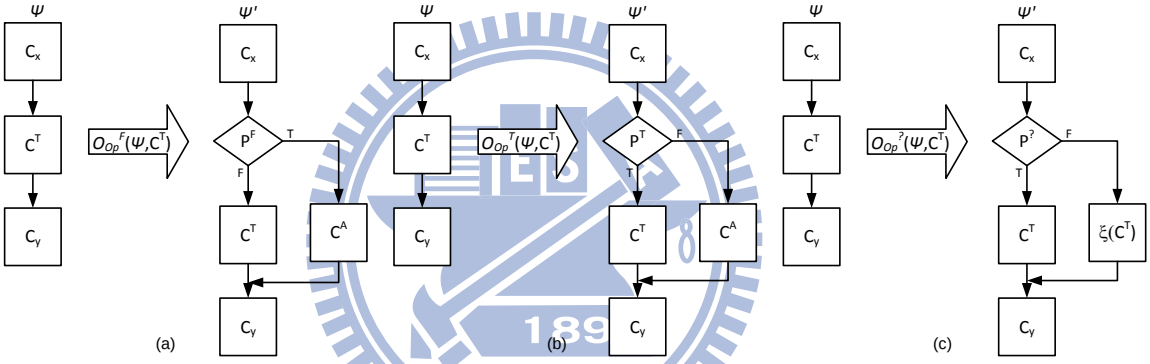


Figure 4.3: Atomic operators of inserting opaque predicates: (a) Type I, (b) Type II, and (c) Type III

type II predicate always evaluates to true; and a type III predicate can sometimes evaluate to true and sometimes to false. In this dissertation, we denote these predicates by P^F, P^T, and $P^?$, respectively. The opaque predicates can be applied in inserting branches for obfuscation to preserve the same execution result. The insertion can be accomplished by inserting the three types of opaque predicates to hide the real control flow of a source program. O_{Op}^F, O_{Op}^T and $O_{Op}^?$ represent the three types of insertion, respectively.

- O_{Op}^{F}: As P^F is inserted in front of the target block C^T, C^T should be moved to the false target of P^F to maintain the same functionality (see Figure 4.3(a)). Since the execution result of P^F is always false, any code block C^A may be specified as the true target of P^F. C^A can be an existing or a dummy code block by applying the operator O_D.

- O_{Op}^{T}: The procedure for inserting P^T is similar to that for P^F (see Figure 4.3(b)). Since P^T always evaluates to true, C^T is placed as the true target of P^T. C^A, any code block, can be its never-reached false target.

- $O_{Op}^{?}$: Figure 4.3(c) shows the actions of $O_{Op}^{?}$. To ensure the same functionality, the equivalence of C^T is placed on one of the targets of $P^?$.

The algorithms are explained in Algorithm A.2, A.3 and A.4.

Algorithm A.2

Insert Type I Opaque Predicates, $O_{Op}^{F}(\psi, C^T)$

$V \leftarrow V \cup \{P^F\}$;
IF C^T **is the entry point THEN**
  Replace the entry point with P^F;
END IF;
Replace edges (C^T, C_i) **with** (P^F, C_i) $\forall i$;
Insert $(P^F, C^A)^T$ **and** $(P^F, C^T)^F$ **to** E; □

Algorithm A.3

Insert Type II Opaque Predicates, $O_{Op}^{T}(\psi, C^T)$

$V \leftarrow V \cup \{P^T\}$;
IF C^T **is the entry point THEN**

Replace the entry point with P^T**;**

END IF;

Replace edges (C^T, C_i) **with** (P^T, C_i) $\forall i$**;**

Insert $(P^T, C^A)^T$ **and** $(P^T, C^T)^F$ **to** E**;** □

Algorithm A.4

Insert Type III Opaque Predicates, $O^?_{Op}(\psi, C^T)$

$V \leftarrow V \cup \{P^?, \xi(C^T)\}$**;**

IF C^T **is the entry point THEN**

Replace the entry point with $P^?$**;**

END IF;

Replace edges (C^T, C_i) **with** $(P^?, C_i)$ $\forall i$**;**

Insert $(P^?, C^T)^T$ **and** $(P^?, \xi(C^T))^F$ **to** E**;**

Find (C^T, C_j)**, insert** $(\xi(C^T), C_j)$ **to** E**;** □

Insert Forks Figure 4.4 shows the concept of inserting a fork. The insertion of a fork denoted by $O^n_F\left(\psi, C^T\right)$ indicates that a fork is inserted as the immediate predecessor of the code blocks, C^T and C^T's following $n-1$ code blocks. Assume C^T and the following code blocks are the indexed code blocks C_k – C_{k+n-1}. C_k – C_{k+n-1} are executed in parallel immediately after C_x. In addition to inserting a fork, we should also insert a join to guarantee that the execution of these parallel code blocks is completed before C_y to maintain the original functionality. The execution does not continue until the concurrent paths converge at a join. Before applying O_F, the execution dependency between C^T and its successors should be checked. If dependency exists, O_F may result in an incorrect execution. Since the immediate successors of a fork, a join or a branch cannot be executed when these code blocks

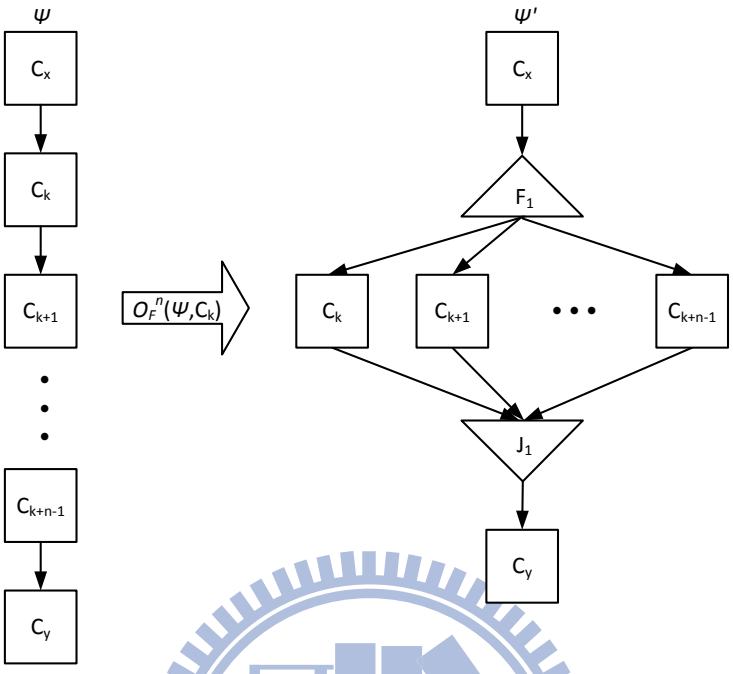


Figure 4.4: Atomic operator of inserting a fork. After insertion, E becomes $\{(C_x, F_1), (F_1, C_k), (F_1, C_{k+1}), ..., (F_1, C_{k+n-1}), (C_k, J_1), (C_{k+1}, J_1), ..., (C_{k+n-1}, J_1), (J_1, C_y)\}$.

are not finished, i.e. execution dependency exists, the target of the operator O_F cannot be either a fork, a join nor a branch.

Algorithm A.5

Insert forks, $O_F^n(\psi, C^T)$**,** $C^T = C_k$

$V \leftarrow V \cup \{F, J\}$;
$E \leftarrow E \cup \{(F, C_k), (F, C_{k+1}), \cdots, (F, C_{k+n-1})\}$;

Find (C_{k+n-1}, C^A) **and insert** (J, C^A) **to** E**;**

IF C^T **is the entry point THEN**

  Replace the entry point with F**;**

END IF;

Replace edges (C_i, C^T) **with** (C_i, F) $\forall i$**;**

Replace edges (C_i, C_j) **with** $(C_i, J) \forall C_i \in \{C_k, C_{k+1}, \cdots, C_{k+n-1}\}$**;** □

Insert Dummy Loops A loop can be achieved by combining simple blocks and branches. The operator O_D^L inserts an extra loop, composed of a dummy

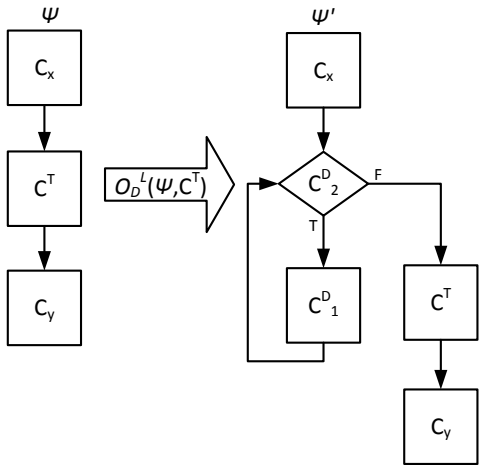


Figure 4.5: Atomic operator of inserting a dummy loop. After insertion, $E = \{(C_x, C_2^D), (C_2^D, C_1^D)^T, (C_2^D, C^T)^F, (C_1^D, C_2^D), (C^T, C_y)\}$.

simple block (C_1^D) and a dummy branch (C_2^D), in front of the target block C^T as shown in Figure 4.5. If C^T is the successor of a sequential edge or the true/false target of a branch edge, it will be replaced by the dummy branch. Then a new sequential edge (C_1^D, C_2^D) and two additional branch edges $(C_2^D, C_1^D)^T$, $(C_2^D, C^T)^F$ are inserted into the edge set where C_1^D is a dummy simple block. In this way, a loop composed of C_1^D and C_2^D is constructed.

Algorithm A.6

Insert Dummy Loops, $O_D^L(\psi, C^T)$

$V \leftarrow V \cup \{C_1^D, C_2^D\}$;
IF C^T **is the entry point THEN**
  Replace the entry point with C_2^D
END IF;
Replace edges (C^T, C_i) **with** (C_2^D, C_i) $\forall i$**;**
Insert (C_1^D, C_2^D) **to** E**;**
Insert $(C_2^D, C_1^D)^T$ **and** $(C_2^D, C^T)^F$ **to** E**;** □

Insertion of Edges

In a simple control flow graph of a sequential program, it is infeasible to insert an edge because each node has only one outgoing edge for a sequential program. On the contrary, the operator of inserting an edge may be able to be applied to the control flow graph of a parallel program because there are multiple edges outgoing from a folk, and an additional folk edge can be inserted in this case.

Insert Fork Edges The operator O_G inserts a folk edge between the first target C_1^T, which has to be a folk, and the second target code block C_2^T. Note that dependency between the two targets and other successors of C_1^T should be checked before the insertion. If dependency exists, incorrect execution may occur.

Algorithm A.7
Insert Fork Edges, $O_G(\psi, C_1^T, C_2^T)$

IF dependency exists THEN
  BREAK;
END IF;
IF $C_1^T \in F$ **THEN**
  $E \leftarrow E \cup \{(C_1^T, C_2^T)\}$**;**
END IF;

□

4.3.2 Update

Update of Nodes

The operators in this category modify the existing nodes and change the contents of the nodes. We introduce two specific operators here:split code blocks and replace code blocks.

Split Code Blocks Splitting a code block into pieces increases the number of vertices in the CFG. The split pieces are advantageous to creating more variations of the control flow. Combining the splitting operator with other operators helps implement more complex obfuscating transformations. In the following, the actions of the operators of splitting simple blocks and splitting branches are explained.

- $O_S^{S,n}$: The operator splits a simple block into n pieces. In Figure 4.6, $O_S^{S,n}(\psi, C^T)$ splits C^T into n pieces. Assume that C^T is originally indexed as C_k. Then the newly split pieces are denoted by C_{k1} – C_{kn}, where n is limited to the instruction count of C^T. Algorithm A.8 shows the algorithm for this operator.

- $O_S^{B,n}$: The operator splits a target branch into smaller pieces. Similarly, the parameter n is limited to the numbers of condition expressions in C^T. We take Figure 4.7 as an example that C^T is expressed as

$$cond_1 \texttt{ AND } ((cond_2 \texttt{ AND } cond_3) \texttt{ OR } cond_4).$$

  Since there are four condition expressions in C^T, n is limited to 4. After splitting C^T, the original CFG is then converted to the CFG on

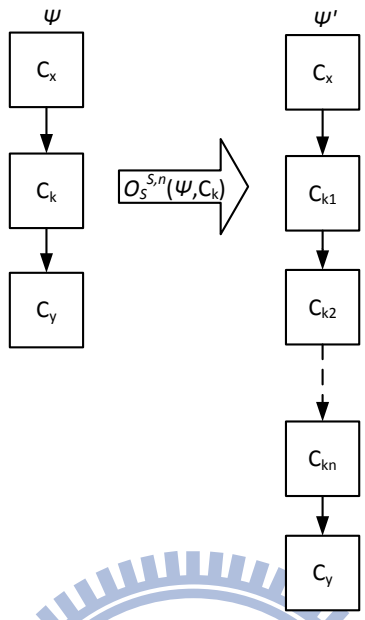


Figure 4.6: Atomic operator of splitting a simple block. Assume $C^T = C_k$. After splitting, $E = \{(C_x, C_{k1}), (C_{k1}, C_{k2}), \cdots, (C_{k(n-1)}, C_{kn}), (C_{kn}, C_y)\}$.

the right-hand side, where C_{ki} represents $cond_i$ with the assumption that C^T is indexed as C_k.

Algorithm A.8 and A.9 are the algorithms for splitting code blocks. In Algorithm A.9, the condition expressions in a branch are first parsed and converted to postfix orders. Each parsed element is denoted by $item_i$.

Algorithm A.8

Split Simple Blocks, $O_S^{S,n}(\psi, C^T)$**,** $C^T = C_k$

IF $C_k \in B$ **THEN**
  BREAK;
END IF;
IF $n >$ **instruction count of** C_k **OR** $n < 2$ **THEN**
  BREAK;
END IF;
IF C_k **is the entry point THEN**
  Replace the entry point with C_{k1}**;**
END IF;
$V \leftarrow V - \{C_k\}$;

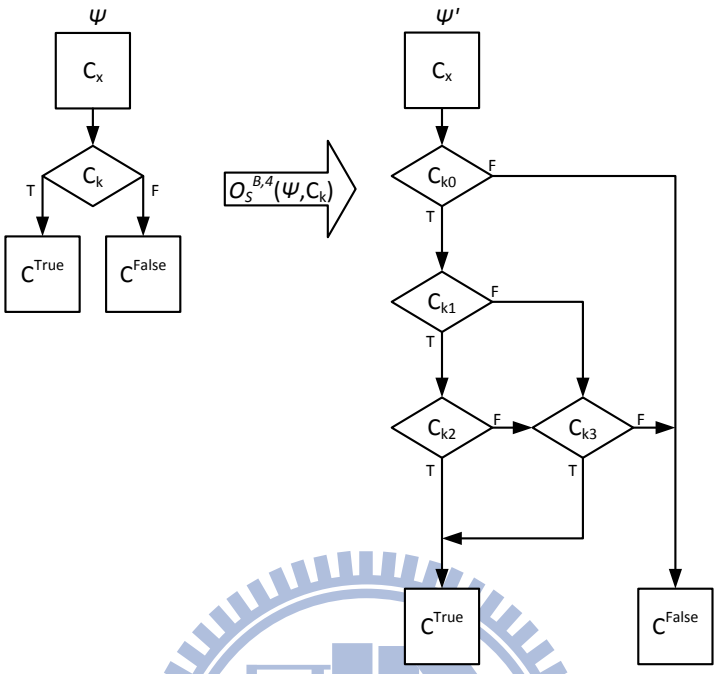


Figure 4.7: Atomic operator of splitting a branch. The example splits C^T into four pieces. Assume $C^T = C_k$. After splitting, $E = \{(C_x, C_{k1}), (C_{k1}, C_{k2})^T, (C_{k1}, C^{False})^F, (C_{k2}, C_{k3})^T, (C_{k2}, C_{k4})^F, (C_{k3}, C^{True})^T, (C_{k3}, C_{k4})^F, (C_{k4}, C^{True})^T, (C_{k4}, C^{False})^F\}$.

$V \leftarrow V \cup \{C_{ki} | \forall i, 1 \leq i \leq n\}$;

FOR edges (C_x, C_y) **DO**

IF $C_x = C_k$ **THEN**

Replace (C_k, C_y) **with** (C_{kn}, C_y);

ELSE IF $C_y = C_k$ **THEN**

Replace (C_x, C_y) **with** (C_x, C_{k1});

END IF;

$E \leftarrow E \cup \{(C_{k1}, (C_{k2}), ((C_{k2}, (C_{k3}), \ldots, (C_{k(n-1)}, C_{kn})\}$; □

Algorithm A.9

Split Branches, $O_S^{B,n}(\psi, C^T)$, $C^T = C_k$

$m \leftarrow 1$; $k \leftarrow 0$; $n \leftarrow 0$;
$N \leftarrow$ `number of condition expressions in` C_k;

IF C_k **is the entry point THEN**

Replace the entry point with C_{k1};

END IF;

$V \leftarrow V - \{C_k\}$;

$V \leftarrow V \cup \{C_{kj} | \forall j, 1 \leq j \leq N\}$;

FOR $m \leq 2N - 1$ **DO**

IF $item_m$ **is a condition THEN**

$n \leftarrow n + 1$; $m \leftarrow m + 1$;

$stack_n \leftarrow item_m$;

ELSE IF $item_m$ **is an operator THEN**

$m \leftarrow m + 1$;

IF $n \geq 2$ **THEN**

$tmp_1 \leftarrow stack_{n-1}$; $tmp_2 \leftarrow stack_n$;

$stack_{n-1} \leftarrow$ **NULL**; $stack_n \leftarrow$ **NULL**;

$C_{kj} \leftarrow tmp_1$; $C_{k(j+1)} \leftarrow tmp_2$;

SWITCH $item_m$

CASE AND:

$tmpE = tmpE \cup$

$\{(C_{kj}, C_{k(j+1)})^T, (C_{kj}, C^{False})^F,$

$(C_{k(j+1)}, C^{True})^T, (C_{k+1}, C^{False})^F\}$;

CASE OR:

$tmpE = tmpE \cup$

$\{(C_{kj}, C^{True})^T, (C_{kj}, tmp_2)^F,$

$(C_{k(j+1)}, C^{True})^T, (C_{k(j+1)}, C^{False})^F\}$;

END SWITCH;

$n \leftarrow n - 2$; $j \leftarrow j + 2$;

ELSE IF $n = 1$ **THEN**

$tmp_1 \leftarrow stack_n$; $tmp_2 \leftarrow$ **NULL**;

$stack_n \leftarrow$ **NULL**; $C_{kj} \leftarrow tmp_1$;

SWITCH $item_m$

CASE AND:

Replace C^{True} **with** tmp_1;

CASE OR:

Replace C^{False} **with** tmp_1;

END SWITCH;

$tmpE = tmpE \cup \{(C_{kj}, C^{True})^T, (C_{kj}, C^{False})^F\}$;

$n \leftarrow n - 1$; $j \leftarrow j + 1$;

ELSE IF $n = 0$ **THEN**

SWITCH $item_m$

CASE AND:

Replace C^{True} **with** tmp_1**;**
CASE OR**:**
Replace C^{False} **with** tmp_1**;**
END SWITCH;
END IF;
END IF;
END FOR;
Replace (C_x, C_k) **with** (C_x, C_{k1})**;**
$E \leftarrow E - \{(C_k, C^{True})^T, (C_k, C^{False})^F\}$**;**
$E \leftarrow E \cup tmpE$**;** □

Replace with Equivalent Codes Equivalent codes are those with the same execution result as the origins while their implementations are different. The equivalent codes conduce to confuse reverse engineers. The operator $O_E(\psi, C^T)$ replaces C^T in ψ with its equivalent code $\xi(C^T)$.

Algorithm A.10
Replace with Equivalent Codes, $O_E(\psi, C^T)$

$V \leftarrow V \cup \{\xi(C^T)\}$**;**
$V \leftarrow V - \{C^T\}$**;**
IF C^T **is the entry point THEN**
Replace the entry point with $\xi(C^T)$**;**
END IF;
Replace C^T **with** $\xi(C^T)$**;** □

Update of Edges

The category focuses on modifying edges without affecting the existing code blocks and their contents.

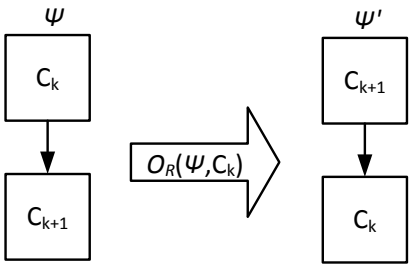


Figure 4.8: Atomic operator of reordering code blocks. Assume $C^T = C_k$. After reordering, $\{(C_k, C_{k+1})\}$ is replaced with $\{(C_{k+1}, C_k)\}$.

Reorder Code Blocks Randomizing the placement of instructions helps to hide the original execution logics from being reversely engineered. The reordering operator O_R then becomes one of the operators in obfuscating programs (see Figure 4.8). Assume that the target code block C^T is also indexed as C_k. Before applying O_R, the execution dependency between C_k and its immediate successor C_{k+1} should be checked. If dependency exists, then O_R may result in an incorrect execution.

Here is the algorithm for the reordering operator.

Algorithm A.11

Reorder Code blocks, $O_R(\psi, C^T)$, $C^T = C_k$

IF C_k **is a branch THEN**
  BREAK;
END IF;
FOR edges (C_k, C_i) **DO**
  IF C_i **is a branch THEN**
    BREAK;
  END IF;
  IF dependency exists between C_k **and** C_i**,**
    BREAK;
  END IF;
  Replace edge (C_k, C_i) **with** (C_i, C_k)**;**
END FOR; IF C_k **is the entry point THEN**
  Replace the entry point with C_{k+1}**;**
END IF; □

4.4 Formalization of Obfuscating Transformations

A control-flow obfuscating transformation $\mathcal{T}$ can be decomposed into a sequence of operators. Different sequences of operators lead to different transformations. Even with the same sequence, specifying different target blocks to these operators may obtain different results. Hence we represent a transformation $\mathcal{T} = \langle f_1, f_2, \cdots, f_m \rangle$ as the composition of m operators $f_1, \ldots, f_m$, where $f_x \in \{O^F_{Op}(\cdot, C_a), O^T_{Op}(\cdot, C_b), O^?_{Op}(\cdot, C_c), O_E(\cdot, C_d), O^{S,n}_S(\cdot, C_e), O^{B,n}_S(\cdot, C_f), O^L_D(\cdot, C_g), O^S_D(\cdot, C_h), O_R(\cdot, C_i), O^n_F(\cdot, C_j), O_G(\cdot, C_k, C_l)\}$, for $x = 1, \ldots, m$. Note that $\langle\rangle$ stands for an ordered set of functional composition, where $\langle f_1, f_2, \ldots, f_m \rangle$ represents the function g defined by $g(x) = f_m(\ldots f_2((f_1(x))\ldots)$. $C_a, C_b, \ldots, C_l$ represent code blocks, specified as targets of the operators, from the source program.

This formal model can be used to describe many existing control flow transformations [17, 18, 20, 38, 39, 67, 68], according to their algorithms. Decomposing these transformations into a sequence of operators also enables further analysis. Table 4.1 classifies 17 existing transformations according to whether they can be represented as a functional composition of our operators. As the table shows, twelve transformations can be decomposed into a sequence of the proposed operators, but five of them cannot. In this section, we explain the decomposition, justify each entry in the table, and interpret

Table 4.1: Feasibility of Decomposition

Control-Flow Obfuscating Transformations	Decomposable?
Basic Block Fission Obfuscation [38]	Y
Intersecting Loop Obfuscation [38]	Y
Replacing goto Obfuscation [38]	Y
Branch Insertion Transformation [17]	Y
Loop Condition Extension Transformation [17]	Y
Language-Breaking Transformation [17]	Y
Parallelize Code [17]	Y
Add Redundant Operands [17]	Y
Aggregation Transformations [17]	N
Ordering Transformations [17]	Y
Remove Library Calls and Programming Idioms [17]	Y
Table interpretation [17]	N
Degeneration of control flow [18]	Y
Obfuscation Scheme Using Random Numbers [67]	Y
Obfuscating C++ Programs via Flattening [39]	N
Control Flow Based Obfuscation [68]	N
Binary Obfuscation Using Signals [20]	N

Y: can be expressed N: cannot be expressed

these results.

Basic Block Fission Obfuscation [38]

This obfuscation tries to subvert the structures of programs such that decompiling the transformed programs would be unsuccessful. This transformation splits the chosen code blocks into more pieces, and inserts opaque predicates and `goto` instructions into these pieces. In the example presented in [38], to protect the original program against the decompilation attack, a few more blocks were generated and inserted after splitting the chosen code blocks. Then, a type I opaque predicate was inserted to make sure the unreachabil-

ity of the newly inserted code blocks, and thus functionality of the original program was preserved.

In this case, according to the type of the chosen code blocks, we can apply $O_S^{S,n}$ to a simple block or $O_S^{B,n}$ to a branch. Moreover, O_D^S and O_D^L can be used to insert dummy code blocks. Type II opaque predicates are used to perform the functionality of `goto` instructions while any one of three O_{Op} operators can be inserted as opaque predicates to realize the basic block fission obfuscation. Thus, this transformation can be expressed as $\mathcal{T} = \langle f_1, f_2, f_3, f_4 \rangle$, where $f_1 \in \{O_S^{S,n}, O_S^{B,n}\}$, $f_2 \in \{O_D^S, O_D^L\}$, $f_3 = O_{Op}^T$ and $f_4 \in \{O_{Op}^F, O_{Op}^T, O_{Op}^?\}$.

Intersecting Loop Obfuscation [38]

This obfuscation inserts two intersected loops to a source program to make control flows unrecognizable for decompilers. Also, a type I opaque predicate is inserted to skip the newly inserted intersected loops and to avoid any influence upon the original execution. Since a loop consists of a simple block and a branch, we use two simple blocks and two opaque predicates to create the two intersected loops. To preserve the same execution, the newly inserted loops are followed by a type I opaque predicate. Hence, this transformation can be expressed as $\mathcal{T} = \langle O_D^S, O_D^S, O_{Op}, O_{Op}, O_{Op}^F \rangle$, where $O_{Op} \in \{O_{Op}^F, O_{Op}^T, O_{Op}^?\}$.

Replacing `goto` Obfuscation [38]

This obfuscation replaces `goto` instructions with conditional branch instructions that do not influence the original control flow. This can be realized by replacing the `goto` instructions with their equivalent codes. The trans-

formation can be represented as $\mathcal{T} = \langle f_1, f_2, \ldots, f_m \rangle$, where $f_x = O_E$ for $x = 1, \ldots, m$.

Branch Insertion Transformation [17]

This transformation is designed based on one of the three opaque predicate insertion operators, O^F_{Op}, O^T_{Op} and $O^?_{Op}$. It can be expressed as $\mathcal{T} = \langle O^2_S, O_{Op}, [O_D] \rangle$. The target block is first split into two pieces by $O^2_S \in \{O^{S,2}_S, O^{B,2}_S\}$. The second step is to apply O_{Op} to the split pieces, where $O_{Op} \in \{O^F_{Op}, O^T_{Op}, O^?_{Op}\}$. Finally, the insertion of dummy codes $O_D \in \{O^S_D, O^L_D\}$ is optional in this transformation.

Loop Condition Extension Transformation [17]

A loop can be obfuscated by complicating the loop condition. The idea is to extend the loop condition using opaque predicates that do not affect the iterations when the loop is executed. The targets of opaque predicates are the branch blocks forming the loop condition. These opaque predicates are inserted immediately in front of the branch blocks. Optionally, a dummy code block can also be placed in its never-reached target. The formal representation of this transformation can be defined as $\mathcal{T} = \langle O_{Op}, [O_D] \rangle$, where $O_{Op} \in \{O^F_{Op}, O^T_{Op}, O^?_{Op}\}$ and $O_D \in \{O^S_D, O^L_D\}$ (optional).

Language-Breaking Transformation [17]

This transformation converts a reducible flow graph to a non-reducible one by turning a structured loop into a loop with multiple headers. For obscurity,

the loop body is split into two pieces. A type I or type II opaque predicate is inserted in front of the original loop to make a never-executed jump into the second split piece. Since it is a never-executed jump, the second split piece is placed on the never-executed target of the inserted opaque predicate. The expression in terms of the operators is defined as $\mathcal{T} = \langle O_S^2, O_{Op}, [O_D] \rangle$ where O_S^2 is the operator to split a code block into two halves, $O_{Op} \in \{O_{Op}^F, O_{Op}^T\}$, and $O_D \in \{O_D^S, O_D^L\}$ is optional.

Parallelize Code [17]

A reverse engineer may find a parallel program more difficult to understand than a sequential one. To increase parallelism for obscuring the control flow of a program, we can either create dummy processes or split a code block into multiple data-independent blocks executed in parallel. According to our formalization framework, the expression in terms of the operators is defined as $\mathcal{T} = \langle O_D^S, O_S, O_F \rangle$. We first insert dummy simple code blocks or split a block into several pieces. Then we make them run in parallel by the atomic operator of inserting forks.

Add Redundant Operands [17]

Algebraic laws can be used to add redundant operands to arithmetic expressions. The logic of the original expression is modified, and the operation becomes more complex. The transformation is formalized by $\mathcal{T} = \langle f_1, f_2, \ldots, f_m \rangle$, where $f_x = O_E$ for $x = 1, \ldots, m$. Only the method "add redundant codes" can be used as the technique of creation of equivalent codes

for the operator O_E.

Aggregation Transformations [17]

This transformation falls into two categories. One is to break up codes which programmers aggregated them into a method and scatter the codes over the program. The other is to aggregate the codes which seem not to belong together into one method. Since operators are mainly applied to code blocks, this transformation with the basis of methods cannot be represented using our operators.

Ordering Transformations [17]

To eliminate useful spatial clues to understanding the execution logics of a program, ordering obfuscation was proposed to randomize the placement of any code block in a source program. The operator $O_R(\psi, C_T)$ is used to express the ordering transformations in the form $\mathcal{T} = \langle f_1, f_2, \ldots, f_m \rangle$ where $f_x = O_R$ for $x = 1, \ldots, m$. Note that O_R exchanges the two target blocks if no dependency exists between them.

Remove Library Calls and Programming Idioms [17]

It is known that in some programming languages like Java, the standard library calls may provide useful clues to reverse-engineers. To impede this problem from being exacerbated, an obfuscator may provide its own versions of the standard libraries. The versions can be generated by applying the operator O_E to the code blocks of the libraries. Therefore, $\mathcal{T} = \langle f_1, f_2, \ldots, f_m \rangle$

where $f_x = O_E$ for $x = 1, \ldots, m$.

Table Interpretation [17]

This transformation converts a code block into different virtual machine code which is then executed by a virtual machine interpreter within the obfuscated program. Since we do not talk about interpreters in this dissertation, it fails to formalize this transformation with the proposed operators.

Degeneration of Control Flow [18]

This transformation converts high-level control structures into equivalent *if-then-goto* constructs. Then, `goto` statements are modified such that the target addresses of the `goto` statements are computed at runtime. In the first step, the expected construct can be developed according to the proposed CFG. Since the transformation replaces control flow with computed-`goto` statements, equivalence techniques can be used to generate the target blocks of the `goto` statements. Subsequently, O_E can be applied to branches of the construct to dynamically determine the target address of the `goto` instructions. Thus, the transformation can be expressed as $\mathcal{T} = \langle f_1, f_2, \ldots, f_m \rangle$, where $f_x = O_E$ for $x = 1, \ldots, m$.

Obfuscation Scheme Using Random Numbers [67]

In this transformation, a dispatcher uses a random number (RN) to determine its target method while a method point (MP) is used to check whether the selected target method should be executed or not. If RN $\neq$ MP, the selected

method is not executed. The transformation regenerates a random number to select another method until RN matches MP.

The concept of using a dispatcher and a random number can be accomplished by the obscurity and randomness of type III opaque predicates. Here, a type III opaque predicate is inserted in front of each method designated as the true target of the predicate. If the predicate evaluates to true, its corresponding method is reached; otherwise, the execution jumps to another predicate with the same functionality as the former. Since MP is used to determine the accurate execution path, we insert other type III opaque predicates for each method, where the newly inserted predicates play the same role as MP. Hence, the transformation can be expressed in the form $\mathcal{T} = \langle f_1, f_2, \ldots, f_m \rangle$, where $f_x = O^?_{Op}$ for $x = 1, \ldots, m$.

Obfuscating C++ Programs via Flattening [39]

The transformation is to firstly break up the function body into several smaller blocks and then make the blocks in the same nesting level. Besides, a dispatcher determines which equal-leveled blocks are to be executed. Although we can adopt the same way for the implementation of the dispatcher, we cannot carry out the main idea of this transformation that the split blocks are in the same nesting level. Therefore, it is unable to express the transformation with the operators.

Control Flow Based Obfuscation [68]

Two processes, P and M, are used in this transformation. P-process performs the main functionality and acts as the original program. M-process handles and saves the control flow information extracted from the original program. P-process queries M-process for the correct addresses whenever P-process reaches a point with missing control flow information. Since additional information is needed to achieve this transformation, we fail to decompose it.

Binary Obfuscation Using Signals [20]

This transformation replaces an unconditional jump with code, attempting to access an illegal memory location that raises a signal. The signal handling routine determines the target address of the original unconditional jump and takes over the control flow of the program. Since we do not refer to any signals and signal handling routines, this transformation cannot be expressed with our operators. However, the idea of using the signal handling routine can be introduced as an approach to generating an equivalent code block.

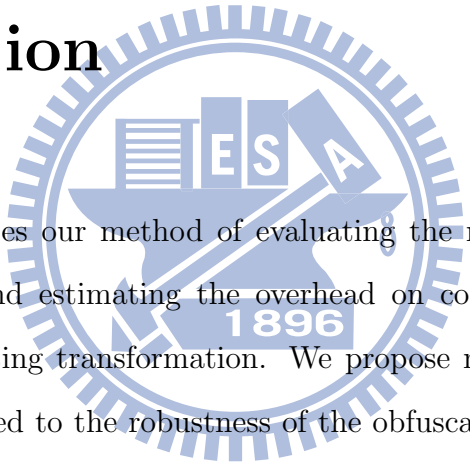
4.5 Summary

We defined a control flow graph in this chapter. The control flow graph is able to represent both a simple and a parallel program. We then identified atomic operators as the basic building blocks for formalizing control-flow obfuscation based on the control flow graph. We examined the feasibility of the

formalization method by decomposing the existing control-flow obfuscating transformations. Twelve of the seventeen transformations can be decomposed into the sequence of the operators. The formalization is feasible in high-level abstraction.

Chapter 5

Evaluation of Control-Flow Obfuscation



This chapter describes our method of evaluating the robustness of an obfuscated program and estimating the overhead on code size caused by a control-flow obfuscating transformation. We propose metrics that we conjecture may be related to the robustness of the obfuscated program against reverse engineering. As for estimating the overhead, our approach works by characterizing the space penalty of each individual atomic operator based on the formalization of control-flow obfuscation. We believe that these evaluation techniques can help to analyze the tradeoff between the effectiveness and the overhead of different obfuscating transformations.

5.1 Preliminaries

Here, we introduce the notations used in Chapter 5.

Notations

$comp(\psi)$	Complexity of a parsed program ψ
$cs(G_i, G_j)$	Common subgraph of graph G_i and G_j
$dis(G_i, G_j)$	Distance between graph G_i and graph G_j
$DP(\psi, \tau)$	Distance-Potency (DP) vector of applying a transformation τ to a parsed program ψ
$max(num_i, num_j)$	Maximum of num_i and num_j
$mcs(G_i, G_j)$	Set of vertices of the maximal common subgraph of graph G_i and graph G_j
n_c	Number of condition expressions contained in a branch
$pcomp(\psi)$	Level of parallelism of ψ
$pot(\psi, \psi')$	Potency, indicating the increment of complexity between ψ and ψ'
$scomp(\psi)$	Sequential complexity of ψ
$scope(\psi)$	Value, determined by the complexity measure SCOPE, of ψ
$range(\psi, B_i)$	Set of vertices in the loop led by branch B_i in ψ or on the paths branching out at B_i in ψ until the paths converge
$range(\psi, F_i)$	Set of vertices on the parallel execution paths led by fork F_i in ψ until the paths join
$\|crange(\psi, B_i)\|$	Size of compound range of B_i in ψ
$\|edge(G_i)\|$	Number of edges of graph G_i
$\|G_i\|$	Size of graph G_i, i.e. the number of vertices of G_i
$\|range(\psi, B_i)\|$	Size of $range(\psi, B_i)$
$\|range(\psi, F_i)\|$	Size of $range(\psi, F_i)$

$\bar{C}$	Average size of code block C

5.2 Evaluation Metrics

Reverse engineers generally follow the following process to reverse-engineer a program [69]:

- Identify the component that will be reverse engineered.
- Observe the execution flow, read manuals, and disassemble the code.

The difficulty of reverse engineering an obfuscated program depends on the relationship between the original and transformed program. The exact amount of effort required is difficult to quantify, because it depends upon the experience and skill level of the reverse engineer: it may take some people significantly longer than others to reverse engineer the same program.

We propose a measure that tries to eliminate factors varying from person to person. Our measure does not compare the difficulty of reverse engineering the same program between different reverse engineers; rather, it is intended to estimate the difficulty of reversing different obfuscated programs, if we hold constant the person who is performing the reverse engineering.

To measure the complexity and overhead of obfuscated programs, Collberg et al. [17] proposed several metrics for evaluating an obfuscating transformation, including cost, resilience and potency. The cost metric is defined to measure the additional run-time resources required to execute an obfuscated program. The resilience metric is intended to measure how well an

obfuscating transformation holds up against attacks from an automatic de-obfuscator. The potency metric is supposed to be related to the degree to which an obfuscating transformation confuses a human trying to understand the obfuscated program. Of these three metrics, only potency is intended to measure the difficulty for a reverse engineer to compromise and deduce an obfuscated program. The potency indicates the increment of software complexity after obfuscation. It is defined as Eq. 5.1 shows.

$$pot(\psi, \psi') = \frac{comp(\psi')}{comp(\psi)} - 1. \tag{5.1}$$

Here $comp(\psi)$ and $comp(\psi')$ denote the complexity of the original program ψ and the obfuscated program ψ'.

Potency $pot()$ implies the difficulty in reverse engineering from the perspective of depth, i.e. the increments of software complexity. Nevertheless, for completeness of evaluating robustness of software obfuscation, the difficulty should be assessed from the perspective of width as well. We adopt a distance metric that determines the degree of disparity between an original program and an obfuscated program. The degree, offering the proportion of the original execution paths to the paths of the obfuscated program, indicates the difficulty of reverse engineering. We use both the distance and potency metrics to evaluate robustness of an obfuscated program compared with the original one.

5.2.1 Distance Metric

In this subsection, we introduce certain existing metrics for measuring distance between graphs and discuss their suitability for further implication of

robustness of obfuscation.

MCS Measure Bunke et al. [70] proposed a distance metric based on the maximal common subgraph (MCS). A common subgraph G_{sub} of graph G_1 and graph G_2 is defined as that if there exists subgraph isomorphisms from G_{sub} to G_1 and from G_{sub} to G_2. G_{sub} is the MCS of G_1 and G_2 if there exists no other common subgraph G'_{sub} of G_1 and G_2 that has more nodes than G_{sub}. The distance between two graphs is given in terms of the number of nodes of their MCS:

$$dis(G_1, G_2) = 1 - \frac{|\text{mcs}(G_1, G_2)|}{\max(|G_1|, |G_2|)}.$$

Here $|G|$ is the number of nodes of the graph G, and $\text{mcs}(G_1, G_2)$ represents the MCS of G_1 and G_2. This distance metric could be used to measure the robustness of an obfuscated program obtained from a control-flow obfuscating transformation by letting G_1 denote the CFG of the original program and G_2 the CFG of the obfuscated one.

Graph Union Measure Wallis et al. [71] proposed another distance metric:

$$dis(G_1, G_2) = 1 - \frac{|\text{mcs}(G_1, G_2)|}{|G_1| + |G_2| - |\text{mcs}(G_1, G_2)|}.$$

We refer to this as the graph union measure, since $|G_1| + |G_2| - |\text{mcs}(G_1, G_2)|$ is loosely related to the size of the graph union. It is exactly the size of the union, if G_1 and G_2 have only one common subgraph.

These two metrics (MCS measure and graph union measure) only consider the size of the MCS, and do not reflect any changes in other common

subgraphs. They may fail to speculate on fine changes that a control-flow obfuscating transformation produces. As a result, they may not accurately measure the robustness of obfuscation.

Measure of Graph Edge To measure the robustness of obfuscation in finer-grained, our distance metric differs from those of earlier work. Our metric takes all common subgraphs into account, not merely the MCS. In addition, our metric counts the number of edges in these common subgraphs, instead of the number of nodes, to deliberate on the impacts upon execution paths due to control-flow obfuscation. Our metric (Measure of Graph Edge, MGE) quantifies the distance between two graphs G_1 and G_2:

$$dis(G_1, G_2) = 1 - \sum_i \frac{2\,|edge(cs_i(G_1, G_2))|}{|edge(G_1)| + |edge(G_2)|} \tag{5.2}$$

where $cs_i(G_1, G_2)$ denotes the i^{th} common subgraph of G_1 and G_2, $edge(G)$ is the set of edges within graph G, and $|edge(G)|$ is the number of edges within G. The minimum value of $dis(G_1, G_2)$ is "0" if the two graphs are exactly the same. The maximum value of $dis(G_1, G_2)$ is "1" if no common subgraph exists between G_1 and G_2.

Assume that we know which vertices in G_2 correspond to which vertices in G_1, then the common subgraphs can be uniquely identified and the distance metric is well-defined. Figure 5.1 displays examples of graphs G_1 and G_2, both containing 8 nodes and 7 edges. There are two common subgraphs of G_1 and G_2. One comprises 1 edge, and the other comprises 3. We obtain $dis(G_1, G_2) = \frac{3}{7}$ by Eq. 5.2.

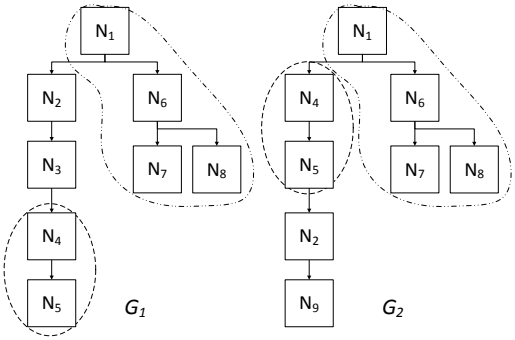


Figure 5.1: Two common subgraphs of G_1 and G_2 are circled.

5.2.2 Potency Metric

The potency, defined by Eq. 5.1, is the other indicator of robustness of obfuscation. Eq. 5.1 calculates the increment of software complexity due to obfuscation, so we need an appropriate measure of software complexity to accomplish the equation. Since our formalization framework considers not only sequential programs but also parallel ones, the complexity measure needs to take both of the sequential and parallel program into account.

Sequential Complexity The complexity metric underlying the potency metric must evaluate complexity from the perspective of obfuscation so that the derived potency value is able to represent the capability of obfuscation. In 1981 Harrison and Megal presented the measure SCOPE [72] to calculate complexity of a control flow graph of a sequential program. The measure SCOPE determines complexity in terms of the size and the depth of nests involved in a control flow graph. Since transforming a CFG into another usually leads to changes of the graph size or the nesting level, the measure SCOPE can be an appropriate base for the potency metric. The measure

SCOPE is defined as Eq. 5.3 shows [72].

$$scope(\psi) = \sum_{B_i \in \mathcal{B}} |range(\psi, B_i)| \tag{5.3}$$

where $\mathcal{B}$ is the set of branches in ψ and $|range(\psi, B_i)|$, size of the range of B_i, stands for the nesting level that B_i contributes. $|range(\psi, B_i)|$ represents the number of code blocks in the loop led by B_i or on the paths branching out at B_i until the paths converge. *scope* increases as the number of nodes in the nests of a program increases. The complexity of the entire control flow graph equals the summation of the complexity of each split sub-graph when the measure SCOPE is introduced. This feature is especially advantageous to a complicated CFG or a CFG with some changing sub-graphs.

Condition expressions which dominate control flow of a program play a crucial role in analyzing and understanding the execution logic of the program. In addition to the nesting level, the number of condition expressions within a branch also contributes complexity of a program. The large the number of condition expressions is, the more effort should be taken to understand the control flow. We extend the measure SCOPE to contemplate the effects resulting from the number of the condition expressions in a branch.

From a high-level programming perspective, a branch can be treated as a building block for two types of control flow structures, `conditional jump` and `loop`, which pose individual effects upon software complexity.

- Conditional jump

  The fundamental control flow graph of a conditional jump is shown in Figure 5.2. It contains a branch with two target code blocks which

are followed by C^A. Assume that in this fundamental control flow graph the branch B_i contains only one condition expression so that $|range(\psi, B_i)|$ is 2. However, it is with a high possibility that there are more than one condition expression contained in a branch. We should further consider the nesting level of a branch in this situation.

While B_i contains n_c condition expressions which are connected by the relation "AND" and "OR," B_i can be split into n_c branches (B_{i1} to $B_{i(n_c)}$), each branch of which involves only one condition expression. The new flow graph deduced from Figure 5.2 now includes n_c branches and 3 simple code blocks. The nesting level of branch B_{i1}, in the deepest nest, is equal to 2 since two code blocks C^{True} and C^{False} are on the divergent paths branching out at B_{i1}. $|rang(\psi, B_{i1})|$ also equals $|range(\psi, B_i)|$. For branch B_{i2}, B_{i1} is moved to either B_{i2}'s true or false target, while B_{i1}'s true and false targets are unchanged. So, before the paths which branch out at B_{i2} meet, three code blocks (B_1, C^{True} and C^{False}) are likely executed, i.e. $|range(\psi, B_{i2})| = 3$. Similarly, B_{i2} is placed as B_{i3}'s true/false target according to the relation between them. In this way, $|range(\psi, B_{i3})|$ is obtained as 4, while 4 code blocks (B_{i2}, B_{i1}, C^{True} and C^{False}) are included. By induction, for branch B_{ij}, $|range(\psi, B_{ij})| = |range(\psi, B_{i1})| + j - 1 = |range(\psi, B_i)| + j - 1$. We define the compound range of B_i to consider the situation that B_i

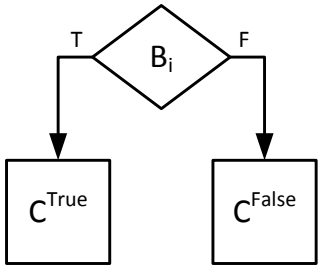


Figure 5.2: Control flow graph of a conditional jump

contains n_c condition expressions by Eq. 5.4.

$$
\begin{aligned}
|crange(\psi, B_i)| &= \sum_{j=1}^{n_c} |range(\psi, B_{ij})| \\
&= \sum_{j=1}^{n_c} (|range(\psi, B_i)| + j - 1) \\
&= n_c \times |range(\psi, B_i)| + \sum_{j=2}^{n_c} (j-1)\,.
\end{aligned}
\tag{5.4}
$$

- Loop

  In a high-level program, a loop may be generated by `for`, `while` and `do-while` statements. The control flow graph of a loop is as Figure 5.3 shows, where the graph contains one branch and two simple code blocks. If B_i in the loop contains only one condition expression, the nesting level, denoted by $|range(\psi, B_i)|$, is 2 since 2 code blocks (C^{True} and B_i itself) are on the paths branching out at B_i until the paths converge. B_i can be further split into n_c pieces (B_{i1} – $B_{i(n_c)}$) when B_i contains n_c condition expressions (see Figure 5.4). A new flow graph thus contains two simple code blocks and n_c branches. For each branch in the loop in Figure 5.4, before its divergent paths meet at C^A, all the branches in the loop and C^{True} are executed. That is, the number of the code blocks

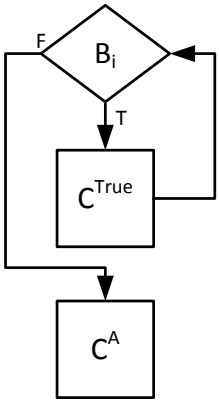


Figure 5.3: Control flow graph of a loop

on the divergent paths is $n_c + 1$. Hence $\forall 1 \leq j \leq n_c$, $|range(\psi, B_{ij})| = n_c + 1 = n_c + |range(\psi, B_i)| - 1$. The compound range of B_i, containing n_c condition expressions, within a loop is thus derived as

$$|crange(\psi, B_i)| = n_c \times (|range(\psi, B_i)| + n_c - 1) \tag{5.5}$$

Parallel Complexity The measure SCOPE was proposed for a sequential program, and thus unable to measure complexity of a parallel program. In 1988, Shatz [73] suggested a framework of measuring a distributed program's complexity, which is calculated based on complexity of each local task and complexity stemming from interaction between the tasks. In Shatz's opinion, complexity of each local task can be calculated by the existing complexity measures of sequential programs, while he proposed the number of concurrently active rendezvous as a useful measure in deriving the complexity of the interaction. According to [73], we define the total complexity of a parallel

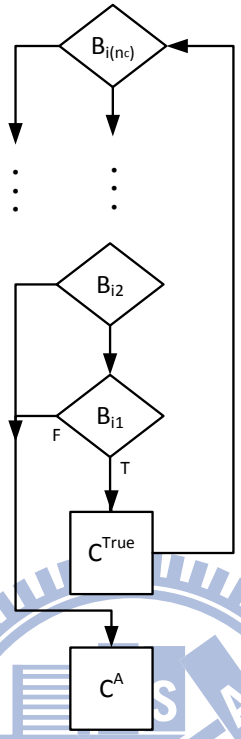


Figure 5.4: Extended control flow graph of a loop: B_i contains n_c condition expressions

program (ψ) as Eq. 5.6 shows:

$$comp(\psi) = w_s \times scomp(\psi) + w_p \times pcomp(\psi) \tag{5.6}$$

$comp(\psi)$ is the total complexity of ψ. $comp(\psi)$ involves two parts: $scomp(\psi)$ and $pcomp(\psi)$, indicating the sequential complexity and the level of parallelism, respectively. w_s and w_p are adjustable weights. We can use existing measures, such as SCOPE, to calculate $scomp(\psi)$. However, there has been little discussion about the metric for calculating $pcomp(\psi)$. Assume that the number of code blocks executed in parallel in a program contributes the level of parallelism of the program; we define a metric $pcomp(\psi)$ by extending

Shatz's concept. Eq. 5.7 expresses the definition of the level of parallelism.

$$pcomp(\psi) = \sum_{F_i \in \mathcal{F}} |range(\psi, F_i)| \tag{5.7}$$

where $\mathcal{F}$ is the set of forks in ψ and $|range(\psi, F_i)|$ represents the parallelism that the fork F_i is conducive to. $|range(\psi, F_i)|$ indicates the number of code blocks on the parallel execution paths led by F_i until the paths join. If no forks exist in ψ, then $pcomp(\psi)$ is zero.

5.2.3 DP Vector

In evaluating the difficulty that a reverse engineer may encounter after obfuscation, Collberg et al. proposed the potency metric (Eq. 5.1) as an estimate of the degree of the difficulty. However, the potency metric, based on the software complexity metric, fails to detect all changes to execution paths and may not accurately measure the robustness of some obfuscating transformations. One way to remedy this kind of shortcoming is to introduce another metric which measures the difficulty caused by obfuscation from a different dimension. We devise a special distance metric for quantifying the difference between two programs after obfuscation. Evaluating the robustness of obfuscation from both the potency and distance perspectives considers more factors and can provide a relatively holistic analysis. Therefore, we suggest using both the potency metric and our distance metric to evaluate the robustness, namely,

$$DP(\psi, \mathcal{T}) = (dis(\psi, \mathcal{T}(\psi)), pot(\psi, \mathcal{T}(\psi))) \tag{5.8}$$

where ψ represents the CFG of the original program, $\mathcal{T}$ the obfuscating transformation, and $\mathcal{T}(\psi)$ the obfuscated CFG. Here $dis(\psi, \mathcal{T}(\psi))$ is computed using the MGE defined in Eq. 5.2, and $pot(\psi, \mathcal{T}(\psi)))$ denotes the potency computed using SCOPE (Eq. 5.3). We expect that larger distance and potency values are correlated to better robustness against reverse engineering.

5.3 Space Penalty

Control-flow obfuscation uses techniques such as creating buggy loops and inserting dummy codes to disturb the real execution path. After obfuscating transformations, a source program can better forbid malicious tampering and reverse engineering. However, it suffers from space penalty. The more transformations applied to a program, the more code size overheads are suffered. Thus, estimation of space penalty is important for assurance whether the increment of code size due to the designated transformations is tolerable. Through the proposed formal representation, estimation of space penalty can be efficiently determined in advance such that administrators can decide whether to apply more transformations or not. In this section we analyze overheads on code size resulting from each atomic operator.

Assuming that an original parsed program ψ has n code blocks where the size of the i^{th} block is denoted as z_i, $\forall i \in [1, n]$, the total code size of ψ is $\sum_{i=1}^{n} z_i$. After obfuscating transformations, α simple blocks, β branches, γ forks and δ joins are inserted into ψ where the size of the i^{th} simple block, the j^{th} branch, k^{th} fork and l^{th} join are respectively indicated as zs_i, zb_j, zf_k and zj_l, $\forall i \in [1, \alpha]$, $\forall j \in [1, \beta]$, $\forall k \in [1, \gamma]$ and $\forall l \in [1, \delta]$. Now, the total code size of

the obfuscated program is $\sum_{i=1}^{n} z_i + \sum_{i=1}^{\alpha} zs_i + \sum_{i=1}^{\beta} zb_i + \sum_{i=1}^{\gamma} zf_i + \sum_{i=1}^{\delta} zj_i$ and the space penalty is $\sum_{i=1}^{\alpha} zs_i + \sum_{i=1}^{\beta} zb_i + \sum_{i=1}^{\gamma} zf_i + \sum_{i=1}^{\delta} zj_i$.

For simplicity of analysis, the summation of the sizes of all inserted blocks is replaced with the product of the average size and the number of blocks. Since the gap between the average size of each type of code blocks may be too large to be ignored, they should be individually denoted by $\bar{S}$, $\bar{B}$, $\bar{F}$ and $\bar{J}$. The space penalty becomes $\alpha \cdot \bar{S} + \beta \cdot \bar{B} + \gamma \cdot \bar{F} + \delta \cdot \bar{J}$. We describe the space penalty with respect to each proposed operator in the following, and Table 5.1 makes the arrangement.

- O_{Op}^{F} or O_{Op}^{T} introduces an extra predicate which results in a space penalty of $\bar{B}$.
- $O_{Op}^{?}$ inserts a new predicate and an equivalent block. This operator yields a space penalty which can be one of the followings depending on the type of C^T: $\bar{B} + \bar{S}$, $2 \cdot \bar{B}$, $\bar{B} + \bar{F}$ or $\bar{B} + \bar{J}$. For example, if C^T is a simple block, then the space penalty is $\bar{B} + \bar{S}$.
- $O_{S}^{S,n}$ or $O_{S}^{B,n}$ splits C^T into smaller pieces. The space penalty is "0", but the number of nodes increases.
- O_R adds nothing and has "0" space penalty.
- O_E replaces C^T with its equivalence $\xi(C^T)$. Since it is a replacement, there is no space penalty.
- O_{D}^{S} inserts an extra simple block and acquires an space penalty $\bar{S}$.

Table 5.1: Space Penalty of Each Atomic Operator

Atomic Operators	Space Penalty
Insert type I/II opaque predicates, O_{Op}^{F}/O_{Op}^{T}	$\bar{B}$
Insert type III opaque predicates, $O_{Op}^{?}$	$\bar{B}+\bar{S}$ or $2\cdot\bar{B}$ or $\bar{B}+\bar{F}$ or $\bar{B}+\bar{J}$
Split code blocks, $O_{S}^{S,n}/O_{S}^{B,n}$	0
Reorder code blocks, O_{R}	0
Replace with equivalent codes, O_{E}	0
Insert dummy simple blocks, O_{D}^{S}	$\bar{S}$
Insert dummy loops, O_{D}^{L}	$\bar{S}+\bar{B}$
Insert forks, O_{F}^{n}	$\bar{F}+\bar{J}$
Insert edges, O_{G}	0

- O_{D}^{L} inserts a dummy loop containing a branch and a simple block. Thus the space penalty is $\bar{S}+\bar{B}$.
- O_{F}^{n} inserts a fork and a join to increase the level of parallelism of a program that leads to a space penalty of $\bar{F}+\bar{J}$.
- O_{G} inserts an edge without any code instructions. Therefore, no space penalty is produced.

5.4 Summary

In this chapter, we introduced the evaluation of control-flow obfuscation based on the formalization of control-flow obfuscating transformations. We

evaluated a control-flow obfuscating transformation in terms of the robustness and the overhead on code size. We proposed the DP vector, composed of the distance and the potency metrics, to calculate the robustness of obfuscation. Moreover, we provided a light-weight approach to estimating the space penalty caused by an obfuscating transformation on the basis of the formalization in Chapter 4.

We recognize our metrics of the DP vector serve as merely heuristic, general indicators of security. However these metrics can still be the first step towards the evaluation of obfuscation. We do not claim that a large value of our metric implies that the obfuscation will necessarily be secure against reverse engineering; we expect that large values of this metric are necessary but not sufficient for security. Our metrics are only intended to reflect the difficulty of reverse engineering through static analysis – it does not reflect information that might be gained by running the program and observing its execution, or by performing some other kind of dynamic analysis. Nonetheless, we conjecture that the metrics are helpful in comparing different approaches to obfuscation. We also realize that the calculation of space penalty in this dissertation is trivial to some extent. However, we believe the simplicity of the calculation is indeed advantageous for an administrator to approximate the overhead, especially in advance of implementing an obfuscated program. Therefore, the DP vector and the space penalty metrics still offer useful information to determine the balance between the protection capability and accompanying overheads.

Chapter 6

Case Studies

We validate our assessment methods of network security by case studies. In assessing network security from the viewpoint of external attacks, we demonstrate the effectiveness and feasibility of our wireless risk assessment method by two examples. In Example I, we assess the risks of two different networks, and then launch a practical eavesdropping attack against the networks. The measured risk values are consistent with the realistic attack results. We illustrate how our method handles the wireless dynamics by Example II, in which configuration snapshots of a wireless network at different timing points are introduced. In assessing network security from internal attacks, Example III and Example IV explain how our framework formalizes and evaluates a control-flow obfuscating transformation. The capability and the overhead of a control-flow obfuscating transformation can be effectively estimated by our framework.

The symbols used in this chapter are the same as those defined in Section 3.1, Section 4.1 and Section 5.1.

6.1 Case Study of Wireless Risk Assessment

This section describes two examples for validating our wireless risk assessment method. In these examples, we should first build up a risk analytic hierarchy, and then develop the experience mapping tables to further determine the risk levels of configurations, the probabilities of acquiring device configurations, etc. With the hierarchy and the tables, our assessment algorithm derives the risk values.

6.1.1 Establish Risk Model

To build up a four-layer risk hierarchy, an administrator needs to select and analyze possible attacks in a wireless network. In the following two examples, we consider 12 known wireless attacks to establish the hierarchy: war driving, eavesdropping, active scan, evil twin, MAC spoofing, IP spoofing, TCP hijacking, beacon flood, association flood, de-authentication flood, key cracking attacks and penetration attacks [74, 75, 76, 77]. These attacks are first classified in terms of their types. Then, we analyze their impacts and prerequisite configurations as listed in Table 6.1. With these analyses, we finally construct the four-layer risk model accordingly.

Type I: The war driving, eavesdropping and active scan attacks fall into this category.

1. War driving targets on exposing and locating accessible wireless networks while driving around a city without a priori information about the target network. With the exposed locations, illicit users

can abuse the networks to interfere services for legitimate users.

2. Eavesdropping imperils traffic confidentiality. Even more, the attacker is capable of replaying or deciphering the packets captured to strike network security violently.

3. After an illicit user actively sends a probe request to a target AP, the user may receive a response from the AP. The response provides designate configurations, such as SSID, MAC address and channel, which can be used to inflict additionally severe damages to the network security requirements.

Type II: The evil twin, MAC spoofing, IP spoofing and TCP hijacking attacks are classified as Type II attacks.

1. Masquerade of a physical AP is referred as an evil twin attack. An attacker sets its SSID to be the same as an AP at a local hotspot. A user may accidentally connect to this malicious AP (called the evil twin), allowing the attacker to intercept all the packets which should be transmitted to the victim AP. Traffic confidentiality, packet integrity and service availability are all jeopardized.

2. An illicit STA can access a network by replacing its MAC address with a permitted one. The MAC spoofing attack obstructs granted access rights and destroys the AP's service availability.

3. An attacker who alters the source IP address in the packet headers can cheat a router into forwarding the modified packets. Hence, the attacker is allowed to access the network.

4. An attacker utilizes the regularity of SYN and ACK numbers during a TCP session and then hijacks the TCP session to eavesdrop the secrets exchanged between the two communication parties.

Type III: Flooding attacks, such as beacon flood, association flood and de-authentication flood, are classified as DoS attacks.

1. In a beacon flood attack, a great amount of counterfeit 802.11 beacons are generated to consume wireless resources and to make legitimate users difficult to access the network. The impact severity hence suffers from the network unavailability caused by the attack.
2. In 802.11, the association requests from STAs are kept in the association table of an AP. Since the memory size of the association table is limited, the AP cannot deal with more association requests when the table is full. By taking advantage of the limited storage capacity of an association table, the impact severity again suffers from the network unavailability caused by a great amount of forged association requests.
3. An attacker floods a victim STA with repeatedly masqueraded de-authentication or disassociation packets to disconnect the STA from its associated AP. This attack forbids the network availability.

Type IV: Key cracking attacks attempt to recover WEP or WPA keys, which were proposed to protect data confidentiality and data integrity

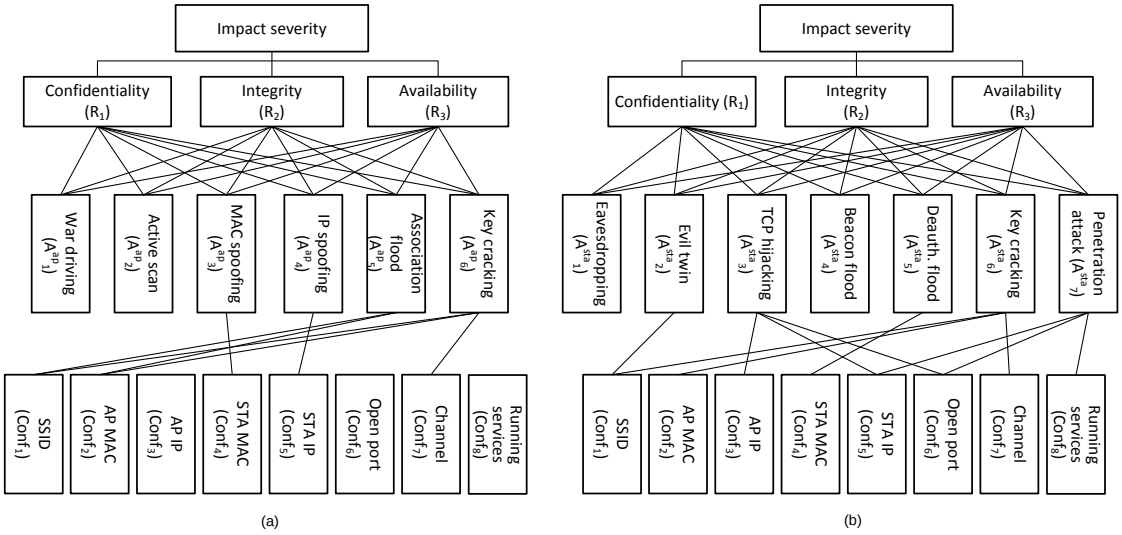


Figure 6.1: Example of four-layer risk analytic hierarchies (4-RAH):(a)4-RAH of an access point (b)4-RAH of a station

and to prevent unauthorized access to APs.

Type V: The penetration attack exploits existing security flaws and vulnerabilities in software, which can be but not limited to internet browsers, drivers or media players. In general, an attacker should possess prerequisite knowledge of the target machine, like its IP address, OS version, software version and running services, before launching this kind of attack against a selected software program.

Next, we further analyze the above attacks and list the victim devices of the above attacks, where A_i^{ap} means the i^{th} attack targeting on an access point and A_i^{sta} denotes the i^{th} attack aiming at a wireless station. Table 6.1 states the analysis results. Then, we construct the 4-RAH for each type of device according to Table 6.1. Figure 6.1 shows the 4-RAH for an AP and that for a station.

Table 6.1: Attack Analysis

Types	Attacks	Target victims	Configurations	Direct impact	Indirect impact
I	War driving (A_1^{ap})	AP	None	-	A
	Eavesdropping (A_1^{sta})	STA	None	C	I, A
	Active scan (A_2^{ap})	AP	None	C	I, A
II	Evil twin (A_2^{sta})	STA	SSID ($Conf_1$)	C, I, A	-
	MAC spoofing (A_3^{ap})	AP	STA MAC ($Conf_4$)	A	-
	IP spoofing (A_4^{ap})	AP	STA IP ($Conf_5$)	A	-
	TCP hijacking (A_3^{sta})	STA	STA IP ($Conf_5$), AP IP ($Conf_3$), open port ($Conf_6$)	C, I, A	-
III	Beacon flood (A_4^{sta})	STA	None	A	-
	Association flood (A_5^{ap})	AP	SSID ($Conf_1$), AP MAC ($Conf_2$),	A	-
	Deauth. flood (A_5^{sta})	STA	STA MAC ($Conf_4$)	A	-
IV	WEP/WPA key cracking (A_6^{ap},A_6^{sta})	STA, AP	SSID ($Conf_1$), AP MAC ($Conf_2$), channel ($Conf_7$)	C, I, A	-
V	Penetration attack (A_7^{sta})	STA	STA IP ($Conf_5$), open port ($Conf_6$), running services ($Conf_8$)	C, I, A	-

C: confidentiality I: integrity A: availability

6.1.2 Develop Experience Mapping Tables

Expert experience is mandatory to assess network risk. To derive the risk value which can be the representative of the practical situation, expertise and real-world experiences are introduced into our risk assessment method. In this step we inject expert experiences and dependable databases for 1) converting the expert experiences to crisp numbers, 2) defining the risk level of a device configuration, 3) defining the probability of acquiring a configuration and 4) assigning each impact a numeric value.

- Linguistic to numeric conversion

  Table 6.2 exhibits an example of the linguistic-to-numeric conversion. In the conversion table, 9 linguistic terms are mapped to crisp numbers falling within the range $[0, 1]$. The crisp numbers assigned in Table 6.2 can be adjusted according to the experience of an administrator or the sociological orbit.

- Risk levels of device configurations

  The risk level of a device configuration is determined according to the following factors.

  1. Configuration management: A device is risky if it adopts default configuration values. If an administrator adopts the default configuration without changing periodically, then it is easy for an attacker to guess the setting. The configuration is hence viewed as a risky configuration. In Figure 6.1, configurations $Conf_1$ (SSID), and $Conf_6$ (open port) are of "High" risk, if default settings are

taken; otherwise "Low" risk levels are assigned.

2. Number of effective attacks: An attack may require a certain configuration for a successful launch. Such an attack is called an effective attack of the configuration. The risk level of a configuration increases with the number of its effective attacks, which take this configuration as a prerequisite. In Figure 6.1, the risk level of $Conf_1$, $Conf_2$, $Conf_3$, $Conf_4$, $Conf_5$, $Conf_6$, and $Conf_7$ is determined by the number of their effective attacks.

   Table 6.2 describes an example conversion between the number of effective attacks and the risk level of a configuration. An administrator may adjust the conversion between the number of effective attacks and the risk level of a configuration according to his or her expert experience and the sociological orbit of a wireless network.

3. $\overline{ihvm}(dev)$: $\overline{ihvm}(dev)$ stands for the risk level of a device dev caused by the vulnerabilities of services running on dev. Table 6.3 lists the vulnerabilities of some services, the severity of each vulnerability, and the age of each vulnerability. We obtain the information from NVD (National Vulnerability Database) [78]. Then, we derive the $hvm(ser)$ by Eq. 3.1 with $\beta = 1$. Solving Eq. 3.5, we finally obtain the risk level stemming from $Conf_8$.

- Probabilities of acquiring configurations

  The probability of acquiring a configuration strongly depends on the encryption method adopted in a wireless network. It takes different

Table 6.2: Effective Attacks and Risk levels

Number of effective attacks	Risk level in linguistics	Risk level in crisp numbers
0	Absolutely low (AL)	0
0	Very low (VL)	0.1
0	Low (L)	0.2
0	Fairly low (FL)	0.3
1	Medium (M)	0.5
2 – 4	Fairly high (FH)	0.7
5 – 8	High (H)	0.8
9 – 11	Very high (VH)	0.9
12	Absolutely high (AH)	1

Table 6.3: Vulnerabilities of Running Services

Running service (ser)	Vulnerabilities*	Severity (α)	Age in year (λ)	$hvm(ser)$
Windows Live Messenger	CVE-2010-0278	4.3	0.32	2.3951
	CVE-2009-2544	6.8	0.81	
	CVE-2009-0647	5.0	1.24	
	CVE-2008-5828	5.0	1.37	
	CVE-2008-5179	5.0	1.49	
Wireshark	CVE-2010-0304	7.5	0.25	3.2299
	CVE-2009-4378	4.3	0.37	
	CVE-2009-4377	4.3	0.37	
	CVE-2009-4376	9.3	0.37	
	CVE-2009-4211	9.3	0.42	
Skype	CVE-2009-4741	10	0.11	2.8013
	CVE-2009-4567	3.5	0.33	
	CVE-2009-5697	4.2	1.37	
	CVE-2009-4875	6.8	1.51	
	CVE-2009-1805	9.3	1.92	
FireFtp	CVE-2009-3478	6	0.6	1.7242
	CVE-2008-2399	9.30	1.96	

*The vulnerabilities are named by the Common Vulnerabilities and Exposures (CVE) standard [79].

Table 6.4: Probability of Acquiring Configurations

Encryption method	Probability		Vulnerable configurations
	Linguistic	Crisp	
No encryption	Absolutely high	1	$Conf_1$, $Conf_2$, $Conf_3$, $Conf_4$, $Conf_5$, $Conf_6$, $Conf_7$, $Conf_8$
WEP	Absolutely high	1	$Conf_1$, $Conf_2$, $Conf_4$, $Conf_7$
	Medium	0.5	$Conf_3$, $Conf_5$, $Conf_6$, $Conf_8$
WPA-PSK, WPA2-PSK	Absolutely high	1	$Conf_1$, $Conf_2$, $Conf_4$, $Conf_7$
	Low	0.2	$Conf_3$, $Conf_5$, $Conf_6$, $Conf_8$
Stronger encryption methods*	Absolutely high	1	$Conf_1$, $Conf_2$, $Conf_4$, $Conf_7$
	Very low	0.1	$Conf_3$, $Conf_5$, $Conf_6$, $Conf_8$

*: WPA-EAP TLS, WPA-EAP AES, etc

efforts to decrypt packets ciphered by different methods. However, in some cases, the attacker may obtain some configurations that cannot be protected by the activated encryption method. By analyzing the configurations displayed in Figure 6.1, we present an example of probabilities of obtaining configurations under protection by various encryption methods in Table 6.4.

- Impact level

  The impacts on the security requirements can be classified into three levels: direct, indirect and no impact. According to the expert experience, an administrator can assign each impact a numeric level. In this example, we assign 1, 0.5 and 0 to direct, indirect and no impact. Then, we produce the degree matrices for each type of victim device

according to Table 6.1. Since 6 attacks target on victim APs, and 7 attacks shoot for stations, a 6-by-3 matrix and a 7-by-3 matrix are built for an AP and a STA, respectively (see Eq. 6.1). By definition, each row of a degree matrix represents the impacts against the security requirements launched by an attack. The number of the elements in a row relies on the number of the security requirements. The element d_{ij} in an AP's degree matrix D stands for the level of impact that attack A_i^{ap} launches upon the j^{th} security requirement. Taking "war driving (A_1^{ap})" as an example, it only has indirect impact on availability of a victim AP, so the 1^{st} row of D is [0 0 0.5]. The meaning of each element in a station's degree matrix is similar to that in an AP's.

$$\texttt{for } AP, \quad D = \begin{bmatrix} 0 & 0 & 0.5 \\ 1 & 0.5 & 0.5 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{6.1}$$

$$\texttt{for } STA, \quad D = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

6.1.3 Assess Network Risk

Example I: Eavesdropping Attack

In the first example, we design two experiments (Ex1-1, and Ex1-2) with similar wireless topologies, one AP, and two STAs. STA_1 runs Windows Live Messenger, and STA_2 maliciously eavesdrops the conversation of STA_1 by running Wireshark. In this example, no security mechanism is applied in Ex1-1, but WPA2-PSK encryption is introduced in Ex1-2 to protect the network traffic. Due to the different configurations, STA_2 successfully eavesdrops the traffic of STA_1 in Ex1-1, but fails to steal the MSN conversations of STA_1 in Ex1-2. Figure 6.2 shows the scenarios and results in Example I.

In the following, we evaluate the risk values of the two networks by the proposed method.

1. Derive $\hat{\mathbf{r}}$, and $\hat{\mathbf{p}}$. The rules of calculating the risk levels of different configurations are mentioned in Section 6.1.2.

   (a) For $Conf_1$, and $Conf_6$, their risk levels should be determined by 1) the configuration management, and 2) the number of effective attacks. In this example, $Conf_1$ does not adopt a default setting, and hence a "Low" risk level is assigned. In addition, $Conf_1$ is a prerequisite for three attacks, including "evil twin," "association flood," and "key cracking" attacks. By Table 6.2, a "fairly high" risk level may be assigned. In the end, we convert these possible risk levels to crisp numbers, and select a maximum value, $max(0.2, 0.7)$, for $Conf_1$. Similarly, we acquire the risk

level of $Conf_6$, $max(0.8, 0.7) = 0.8$, by assuming a default setting is adopted for $Conf_6$.

(b) The risk levels of $Conf_2$, $Conf_3$, $Conf_4$, $Conf_5$, and $Conf_7$ depend on the number of effective attacks. For example, $Conf_2$ is required by 2 attacks, and its risk level is then set to "fairly high," where "fairly high" implies 0.7.

(c) The risk level of $Conf_8$ is determined by the IHVM, as mentioned in Section 6.1.2. In this example, STA_1 is running a service, Windows Live Messenger (ser_1), and STA_2 is running a service, Wireshark (ser_2), while no service is run on AP_1. According to NVD, there are 8, and 93 known vulnerabilities of Windows Live Messenger, and Wireshark, respectively. Table 6.3 displays the newest 5 vulnerabilities of each. Assume the administrator concerns only the latest 5 vulnerabilities of each service, and introduces the highest three $hvm(ser_i)$ to $ihvm(dev)$; according to Eq. 3.1, Eq. 3.2, Eq. 3.4, and Eq. 3.5, we obtain $\overline{ihvm}(AP_1) = 0$, and we derive $\overline{ihvm}(STA_1)$ and $\overline{ihvm}(STA_2)$ by

$\overline{hvm}(ser_1)=\frac{2.3951}{ln(1+10\times5)}=0.6092$ # ser_1: Windows Live Messenger

$ihvm(STA_1)=ln(1+exp(\overline{hvm}(ser_1)))=1.0434$

$\overline{ihvm}(STA_1)=\frac{ihvm(STA_1)}{ln(1+3\times exp(1))}=0.4712$

$\overline{hvm}(ser_2)=\frac{3.2299}{ln(1+10\times5)}=0.8215$ # ser_2: Wireshark

$ihvm(STA_2)=ln(1+exp(\overline{hvm}(ser_2)))=1.1860$

$\overline{ihvm}(STA_2)=\frac{ihvm(STA_2)}{ln(1+3\times exp(1))}=0.5356$

Hence, in both Ex1-1 and Ex1-2, the risk levels of configurations for AP_1, STA_1 and STA_2 are shown as follows.

$$AP_1: \quad \hat{\mathbf{r}} = \begin{bmatrix} 0.7 & 0.7 & 0.5 & 0.7 & 0.7 & 0.8 & 0.5 & 0 \end{bmatrix}^T \tag{6.2}$$

$$STA_1: \quad \hat{\mathbf{r}} = \begin{bmatrix} 0.7 & 0.7 & 0.5 & 0.7 & 0.7 & 0.8 & 0.5 & 0.4712 \end{bmatrix}^T \tag{6.3}$$

$$STA_2: \quad \hat{\mathbf{r}} = \begin{bmatrix} 0.7 & 0.7 & 0.5 & 0.7 & 0.7 & 0.8 & 0.5 & 0.5356 \end{bmatrix}^T \tag{6.4}$$

We calculate the probability of acquiring configurations ($\hat{\mathbf{p}}$) by analyzing Table 6.2, and Table 6.4.

In Ex1-1 (no security protection), we obtain $\hat{\mathbf{p}}$ for each device:

$$\hat{\mathbf{p}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T. \tag{6.5}$$

In Ex1-2 (the WPA2-PSK encryption is applied), the probabilities of acquiring the configurations for each device are adjusted according to the encryption method.

$$\hat{\mathbf{p}} = \begin{bmatrix} 1 & 1 & 0.2 & 1 & 0.2 & 0.2 & 1 & 0.2 \end{bmatrix}^T. \tag{6.6}$$

2. Derive the weight vector of configurations ($\hat{\mathbf{w}}_\mathbf{g}$) of AP_1, STA_1, and STA_2 by Eq. 3.6.

In Ex1-1,

$$\begin{aligned}
\texttt{for } AP_1: \quad \hat{\mathbf{w}}_g &= \begin{bmatrix} 1 & 1 & 0.7 & 0.7 & 0.7 & 0.6333 \end{bmatrix}^T \\
\texttt{for } STA_1: \quad \hat{\mathbf{w}}_g &= \begin{bmatrix} 1 & 0.7 & 0.6667 & 1 & 0.7 & 0.6333 & 0.6571 \end{bmatrix}^T \\
\texttt{for } STA_2: \quad \hat{\mathbf{w}}_g &= \begin{bmatrix} 1 & 0.7 & 0.6667 & 1 & 0.7 & 0.6333 & 0.6785 \end{bmatrix}^T .
\end{aligned} \tag{6.7}$$

In Ex1-2,

$$\begin{aligned}
\texttt{for } AP_1: \quad \hat{\mathbf{w}}_g &= \begin{bmatrix} 1 & 1 & 0.7 & 0.14 & 0.7 & 0.6333 \end{bmatrix}^T \\
\texttt{for } STA_1: \quad \hat{\mathbf{w}}_g &= \begin{bmatrix} 1 & 0.7 & 0.1333 & 1 & 0.7 & 0.6333 & 0.1314 \end{bmatrix}^T \\
\texttt{for } STA_2: \quad \hat{\mathbf{w}}_g &= \begin{bmatrix} 1 & 0.7 & 0.1333 & 1 & 0.7 & 0.6333 & 0.1357 \end{bmatrix}^T .
\end{aligned} \tag{6.8}$$

3. Derive the weight vector of requirements ($\hat{\mathbf{w}}_\mathbf{r}$) for each network device. For example, "availability" of an access point should have a heavier weight than "confidentiality" and "integrity" because the AP is in charge of providing Internet access for wireless devices. Hence, in Ex1-1 and Ex1-2, we have

$$\hat{\mathbf{w}}_\mathbf{r} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \end{bmatrix}^T \quad \texttt{for } AP_1. \tag{6.9}$$

On the other hand, confidentiality, integrity, and availability could be

weighted equally for a wireless station, so that for STA_1 and STA_2

$$\hat{\mathbf{w}}_\mathbf{r} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}^T \tag{6.10}$$

4. Derive the impact severity of each device. By Eq. 3.7, Eq. 6.1, Eq. 6.7, Eq. 6.9, and Eq. 6.10, we obtain the following risk values for devices in Ex1-1.

$$AP_1: \quad I = \hat{\mathbf{w}}_\mathbf{g} \times D \times \hat{\mathbf{w}}_\mathbf{r} = 2.5583$$
$$STA_1: \quad I = \hat{\mathbf{w}}_\mathbf{g} \times D \times \hat{\mathbf{w}}_\mathbf{r} = 3.8904$$
$$STA_2: \quad I = \hat{\mathbf{w}}_\mathbf{g} \times D \times \hat{\mathbf{w}}_\mathbf{r} = 3.9118.$$

Similarly, we can obtain the impact severity of each device in Ex1-2:

$$AP_1: \quad I = 2.2783$$
$$STA_1: \quad I = 2.8313$$
$$STA_2: \quad I = 2.8356.$$

5. Determine the risk value of the wireless network by Eq. 3.8. We obtain the risk values $T = \log_{10}(10^{2.5583} + 10^{3.8904} + 10^{3.9118}) = 4.2120$ for Ex1-1, and $T = \log_{10}(10^{2.2783} + 10^{2.8313} + 10^{2.8356}) = 3.1911$ for Ex1-2.7. Ex1-1 is at HIGH risk because the total impact severity of Ex1-1 is larger than the high threshold 3.6887 based on Table 3.2. Similarly,

Ex1-2 falls into the LOW category because its total impact severity is smaller than the medium threshold 3.3877.

According to Table 3.2, the total impact severity in Ex1-1 and Ex1-2 should fall into the range $[0.4771, 7.3222]$. The scaling information can provide more semantic meanings for an administrator, in addition to the suggested mapping table. Thus, we import the scaling information into the risk value to better describe the semantics. Therefore, for Ex1-1, the risk value can be further represented as $\frac{4.2120}{[0.4771,7.3222]}$. Since 4.2120 falls in the upper half of the range, an administrator can reason that Ex1-1 is at HIGH risk. Similarly, $\frac{3.1911}{[0.4771,7.3222]}$ is the implication for the risk value of Ex1-2. Ex1-2 is at LOW risk because 3.1911 falls in the lower half.

Such a result is close to the real situation because the derived risk value for Ex1-1 is larger when the eavesdropping attack succeeds, and the risk value for Ex1-2 is smaller when the network Ex1-2 can resist the attack.

Example II: Dynamic Topologies

In the second example, we show how our risk assessment method incorporates the dynamic topologies of a wireless network. The example presents snapshots of a wireless network at times τ_1, τ_2, and τ_3. Initially (at time τ_1), the network contains one AP, and two STAs. Then, a new station STA_3 enters the network at τ_2. Finally, STA_1 leaves at τ_3. Figure 6.3 shows the network topologies, and the device configurations. With the proposed method, we

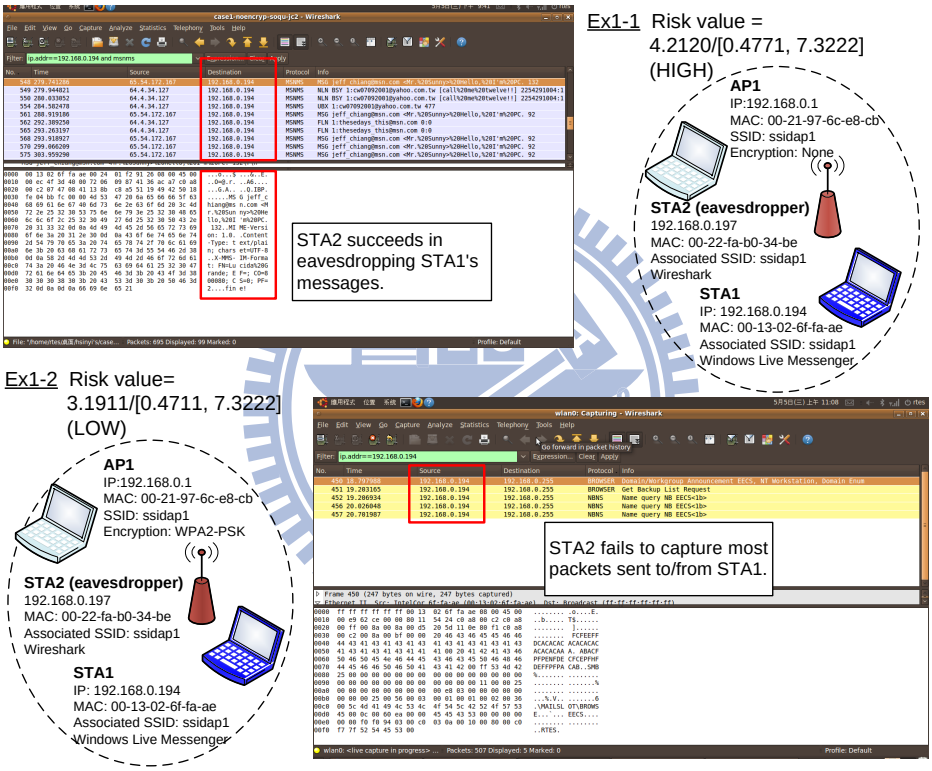


Figure 6.2: Example I: No security mechanism is applied in Ex1-1, but the network is protected by WPA2-PSK in Ex1-2. The eavesdropper (STA_2) successfully captures STA_1's MSN messages in Ex1-1, but fails to sniff the communication session in Ex1-2.

can assess the network risk efficiently by performing the following steps.

Initially, at time τ_1: Ex2-1

Because the two networks in Ex1-2 and Ex2-1 are exactly the same, we derive the total risk value for Ex2-1 the same as those for Ex1-2, $T = 3.1911$. The risk value is represented as $\frac{3.1911}{[0.4771,7.3222]}$ in details.

At time τ_2: Ex2-2

STA_3 joins the wireless network (as shown in Figure 6.3) at time τ_2. Because no changes are made in AP_1, STA_1, and STA_2, we do not need to re-calculate the corresponding impact severities, but perform the following steps.

1. Derive the risk levels of configurations of STA_3. Assume that STA_3 runs the services Windows Live Messenger (ser_1), Skype (ser_3), and FireFtp (ser_4); and the administrator intends to consider the latest five vulnerabilities of each service. According to the service vulnerabilities listed in Table 6.3, we derive $\overline{hvm}(ser_i), i \in \{1, 3, 4\}$ by Eq. 3.1 and Eq. 3.2, and solve $\overline{ihvm}(STA_3)$ by Eq. 3.2 and Eq. 3.4.

$\overline{hvm}(ser_1)=\frac{2.3951}{ln(1+10\times5)}=0.6092$ # ser_1: Windows Live Messenger

$\overline{hvm}(ser_3)=\frac{2.8013}{ln(1+10\times5)}=0.7125$ # ser_3: Skype

$\overline{hvm}(ser_4)=\frac{1.7242}{ln(1+10\times5)}=0.4385$ # ser_4: FireFtp

$ihvm(STA_3)=1.8607$

$\overline{ihvm}(STA_3)=0.8403$

Hence, we obtain the risk level vector of STA_3, i.e.

$$\hat{\mathbf{r}} = \begin{bmatrix} 0.7 & 0.7 & 0.5 & 0.7 & 0.7 & 0.8 & 0.5 & 0.8403 \end{bmatrix}^T.$$

Because Ex2-2 uses WPA2-PSK encryption, we acquire

$$\hat{\mathbf{p}} = \begin{bmatrix} 1 & 1 & 0.2 & 1 & 0.2 & 0.2 & 1 & 0.2 \end{bmatrix}^T.$$

2. Derive the weight vector of configurations of STA_3. By Eq. 3.6, for STA_3,

$$\hat{\mathbf{w}}_g = \begin{bmatrix} 1 & 0.7 & 0.1333 & 1 & 0.7 & 0.6333 & 0.1560 \end{bmatrix}^T \tag{6.11}$$

3. Assign the weight vector of requirements. In this example, we apply the same vector, $\hat{\mathbf{w}}_\mathbf{r}$, given in Example I.

4. Derive the impact severity of STA_3: $I = 2.8559$.

5. Derive the total risk value for Ex2-2, T, from AP_1's impact severity, STA_1's impact severity, STA_2's impact severity and STA_3's impact severity. By Eq. 3.8, $T = \log_{10}(10^{2.2783} + 10^{2.8313} + 10^{2.8356} + 10^{2.8559}) = 3.3561$. Since more devices are within the network at time τ_2, the range of T becomes $[0.6021, 7.7924]$. The risk value can be further represented as $\frac{3.3561}{[0.6021,7.7924]}$.

Compared with the experiment Ex2-1, there are more devices and vulnerabilities in Ex2-2; hence, the total risk value of Ex2-2 is larger than that of Ex2-1, but the risk still falls into the LOW category.

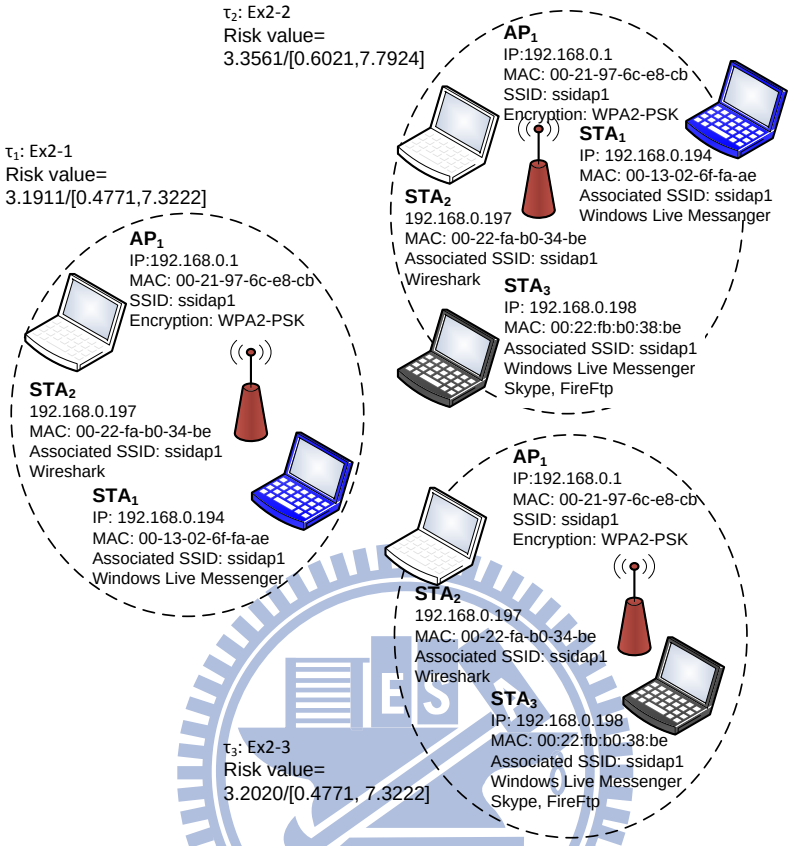


Figure 6.3: Example II: snapshots of a wireless network at different time

At time τ_3: Ex2-3

STA_1 leaves the network with nothing changed for other devices. We can easily calculate the risk value at τ_3 by re-calculating the total impact severity of Ex2-3 with the known impact severities of AP_1, STA_2 and STA_3, the same as those in Ex2-2. As a result, for Ex2-3, $T = 3.2020$. The current risk value is $\frac{3.2020}{[0.4771,7.3222]}$.

6.2 Case Study of Evaluating Control-Flow Obfuscation

This section states our method of evaluating control-flow obfuscation. We introduce two examples (Example III and Example IV) to demonstrate the feasibility and flexibility of our method. Example III models two existing control-flow obfuscating transformations by the atomic operators and then evaluates them by the DP vector and the space penalty metric. Example IV introduces two transformations which are composed of the same sequence of atomic operators but with different target code blocks. The examples indicate that our evaluation method are flexible and sensitive enough to distinguish the differences and to produce fine results.

6.2.1 Graph Conversion

We introduce Program I, a prime number generator, as an original program used in the examples in this section. We first parse Program I into ψ, and then we can apply the atomic operators to make ψ obscure. Figure 6.4 displays ψ and the contents of each code block.

```
/* Program I. Prime number generator */
int _PrimeGen(int tmp) {
     int i;
     for (i=2; i<=tmp/2; i++)
      if (tmp %i == 0)
      return 0;
        return 1;
}
int main() {
     int num, tmp, sum;
```

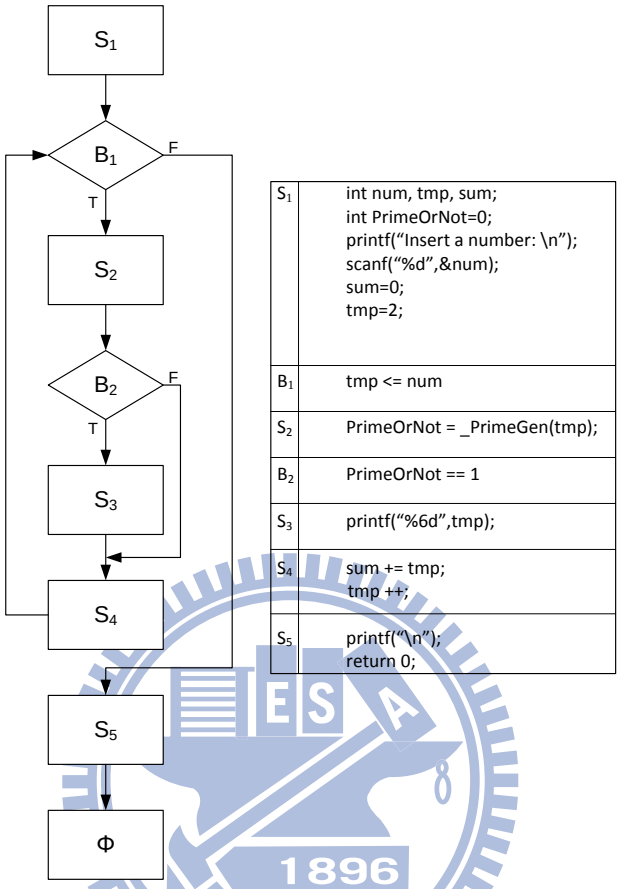


Figure 6.4: CFG of Program I: $\psi = (S_1, (V, E))$, where $V = \{S_1, S_2, S_3, S_4, S_5, B_1, B_2, \phi\}$, and $E = \{(S_1, B_1), (B_1, S_2)^T, (B_1, S_5)^F, (S_2, B_2), (B_2, S_3)^T, (B_2, S_4)^F, (S_3, S_4), (S_4, B_1), (S_5, \phi)\}$.

```
        int PrimeOrNot=0;
        printf("insert a number \n");
        scanf("%d", &num);
        for(sum=0, tmp=2; tmp<=num; tmp++) {
             PrimeOrNot = _PrimeGen(tmp);
             if( PrimeOrNot == 1)
              printf("%6d", tmp);
             sum += tmp;
        }
        printf("\n");
        return 0;
}
```

6.2.2 Obfuscation Formalization and Evaluation

Example III: Existing Control-Flow Obfuscating Transformations

We apply two control-flow obfuscating transformations, the basic block fission obfuscation [38] and the branch insertion transformation [17], in the example:

$$\mathcal{T}_1 = \langle O_S^{S,2}(\cdot, S_1), O_D^S(\cdot, S_{11}), O_{Op}^T(\cdot, B_1), O_{Op}^F(\cdot, C_1^D)\rangle$$
$$\mathcal{T}_2 = \langle O_S^{S,2}(\cdot, S_1), O_{Op}^F(\cdot, S_{12}), O_E(\cdot, S_{12}), O_D^S(\cdot, \xi(S_{12}))\rangle$$

Apply the specified basic block fission obfuscation, $\mathcal{T}_1$

- Running $O_S^{S,2}(\cdot, S_1)$:

$$C^E \leftarrow S_{11},$$
$$V \leftarrow (V - \{S_1\}) \cup \{S_{11}, S_{12}\},$$
$$E \leftarrow (E - \{(S_1, B_1)\}) \cup \{(S_{11}, S_{12}), (S_{12}, B_1)\}.$$

In this example, S_{11} is

```
int num, tmp, sum;
int PrimeOrNot=0;
printf("insert a number \n");
scanf("%d", &num);
```

and S_{12} is

```
sum=0; tmp=2;
```

Since splitting S_1 does not contribute to the nesting level, *scope* remains the same as the original so far.

- Running $O_1^D(\cdot, S_{12})$:

$$V \leftarrow V \cup \{C_1^D\},$$
$$E \leftarrow (E - \{(S_{11}, S_{12})\}) \cup \{(S_{11}, C_1^D), (C_1^D, S_{12})\}.$$

  We choose $\texttt{sum} = \texttt{num} + (\texttt{tmp}\%4)$ as C_1^D. Inserting C_1^D in front of S_{12} does not impose influence on the nesting level, so that *scope* is not modified by this operation.

- Running $O_{Op}^T(\cdot, B_1)$:

$$V \leftarrow V \cup \{P_1^T\},$$
$$E \leftarrow (E - \{(S_4, B_1), (S_{12}, B_1)\}) \cup$$
$$\{(S_4, P_1^T), (S_{12}, P_1^T), (P_1^T, B_1)^T, (P_1^T, C_1^D)^F\}.$$

  We choose $(\texttt{num}^3 - \texttt{num})\%3 == 0$ for P_1^T, which only works with an integer *num*. After applying this atomic operator, *scope* raises from 6 to 16.

- Running $O_{Op}^F(\cdot, C_1^D)$:

$$V \leftarrow V \cup \{P_1^F\},$$
$$E \leftarrow (E - \{(S_{11}, C_1^D), (P_1^T, C_1^D)^F\}) \cup$$
$$\{(S_{11}, P_2^F), (P_2^F, S_{12})^T, (P_2^F, C_1^D)^F, (P_1^T, P_2^F)^F\}.$$

  We choose $\texttt{7} \times \texttt{tmp}^2 - \texttt{1} == \texttt{num}^2$ as P_2^F.

Applying $O_{Op}^F(\cdot, C_1^D)$ to ψ not only results in $|range(\psi, P_2^F)|$ but also produces the increment of $|range(\psi, P_1^T)|$ and $|range(\psi, B_1)|$, so *scope* becomes 19.

Now, we obtain the obfuscated CFG ψ_1 after applying $\mathcal{T}_1$ and can derive the obfuscated program (Program II) from ψ_1, which is displayed in Figure 6.5.

Apply the specified branch insertion transformation, $\mathcal{T}_2$

- Running $O_S^{S,2}(\cdot, S_1)$:

$$
\begin{aligned}
&C^E \leftarrow S_{11},\\
&V \leftarrow (V - \{S_1\}) \cup \{S_{11}, S_{12}\},\\
&E \leftarrow \big(E - \{(S_1, B_2)\}\big) \cup \{(S_{11}, S_{12}), (S_{12}, B_2)\}.
\end{aligned}
$$

  Here, we use the same S_{11} and S_{12} as those used in $\mathcal{T}_1$.

- Running $O_{Op}^F(\cdot, S_{12})$:

$$
\begin{aligned}
&V \leftarrow V \cup \{P_1^F\},\\
&E \leftarrow (E - \{(S_{11}, S_{12})\}) \cup \{(S_{11}, P_1^F), (P_1^F, B_1)^T,\\
&\quad (P_1^F, S_{12})^F)\}.
\end{aligned}
$$

  We choose $\texttt{7} \times \texttt{tmp}^{\texttt{2}} - \texttt{1} == \texttt{num}^{\texttt{2}}$ as P_1^F. $O_{Op}^F(\cdot, S_{12})$ inserts a branch and makes *scope* increased by one.

- Running $O_E(\cdot, S_{12})$:

$$
\begin{aligned}
&V \leftarrow (V - S_{12}) \cup \{\xi(S_{12})\},\\
&E \leftarrow (E - \{(S_{12}, B_1), (P_1^F, S_{12})^F\}) \cup \{(\xi(S_{12}), B_1),\\
&\quad (P_1^F, \xi(S_{12}))^F\}.
\end{aligned}
$$

  Here, $\xi(S_{12})$ can be generated by several techniques, such as inserting dummy instructions and creating parallel execution. We assume the replacement with $\xi(S_{12})$ in this example does not change *scope*.

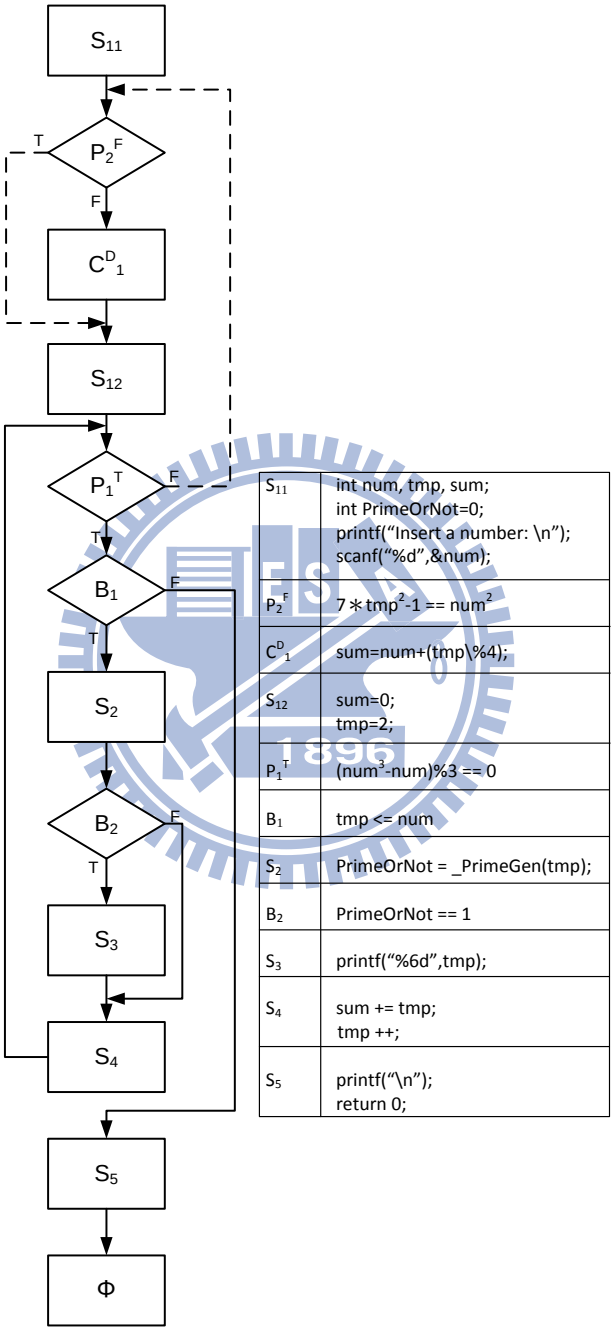


Figure 6.5: ψ_1: obfuscated result of ψ after applying $\mathcal{T}_1$

- Running $O_D^S(\cdot, \xi(S_{12}))$:

$$V \leftarrow V \cup \{C_1^D\},$$
$$E \leftarrow (E - \{P_1^F, \xi(S_{12}))^F\}) \cup$$
$$\{(C_1^D, \xi(S_{12})), (P_1^F, C_1^D)^F\}.$$

  We choose $\texttt{sum} = \texttt{num} + (\texttt{tmp}\%4)$ as C_1^D. Since C_1^D is inserted and located on a branched path, *scope* raises from 7 to 8.

Now, we regenerate the obfuscated program (Program III) according to ψ_2. Figure 6.6 shows ψ_2 and the instructions in each code block.

Evaluation of Example III Comparing ψ, ψ_1, and ψ_2, we obtain the edges of the common subgraphs of ψ, ψ_1 and ψ, ψ_2:

$$edge(cs(\psi, \psi_1)) = \{(B_1, S_2)^T, (B_1, S_5)^F, (S_2, B_2), (B_2, S_3)^T, (B_2, S_4)^F, (S_3, S_4), (S_5, \phi)\}$$
$$edge(cs(\psi, \psi_2)) = \{(B_1, S_2)^T, (B_1, S_5)^F, (S_2, B_2), (B_2, S_3)^T, (B_2, S_4)^F, (S_3, S_4), (S_4, B_1),$$
$$(S_5, \phi)\}$$

The number of edges in ψ, ψ_1 and ψ_2 are 9, 15 and 13. The distances between these graphs are $dis(\psi, \psi_1) = \frac{5}{12}$ and $dis(\psi, \psi_2) = \frac{3}{11}$.

In ψ, there are two branches B_1 and B_2 with range values 5 and 1. Hence, $scope(\psi) = 6$. In ψ_1, the range values of P_1^T, B_1, B_2 and P_2^F are 8, 9, 1 and 1. So, we obtain $scope(\psi_1) = 19$ and $pot(\psi, \psi_1) = \frac{13}{6}$. Similarly, in ψ_2, the range values of P_1^F, B_1 and B_2 are 2, 5 and 1. Therefore, $scope(\psi_2) = 8$ and $pot(\psi, \psi_2) = \frac{1}{3}$.

Positive potency values imply that both $\mathcal{T}_1$ and $\mathcal{T}_2$ achieve obscurity from the perspective of the nesting level of a program. With distance computed

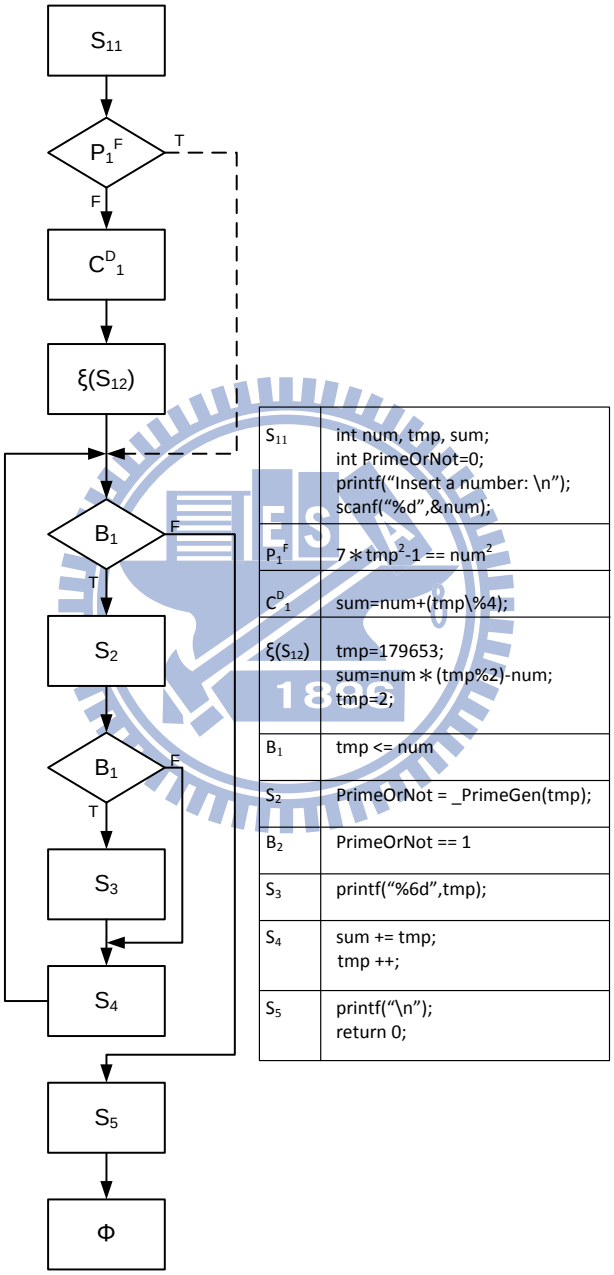


Figure 6.6: ψ_2: obfuscated result of ψ after applying $\mathcal{T}_2$

using the proposed MGE, we then acquire two DP vectors, $DP(\psi, \mathcal{T}_1) = (\frac{5}{12}, \frac{13}{6})$ and $DP(\psi, \mathcal{T}_2) = (\frac{3}{11}, \frac{1}{3})$. Since both distance and potency values of ψ_1 are larger than those of ψ_2. We conclude that $\mathcal{T}_1$ provides the better robustness than $\mathcal{T}_2$ to the original program ψ.

The space penalty caused by $\mathcal{T}_1$ is estimated as $(\bar{S}+2\cdot\bar{B})$, where $O_S^{S,2}(\cdot, S_1)$ results in no overheads, $O_D^S(\cdot, S_{12})$ leads to $\bar{S}$, $O_{Op}^T(\cdot, B_1)$ and $O_{Op}^F(\cdot, C_1^D)$ lead to $2\cdot\bar{B}$ in total. The space penalty resulting from $\mathcal{T}_2$ is $\bar{B}+\bar{S}$, where $O_S^{S,2}(\cdot, S_1)$ and $O_E(\cdot, S_{12})$ derives no overheads, but $O_{Op}^F(\cdot, S_{12})$ and $O_D^S(\cdot, \xi(S_{12}))$ leads to $\bar{B}$ and $\bar{S}$, respectively.

The metrics tell us that $\mathcal{T}_1$ presents better protection to ψ than $\mathcal{T}_2$ while an administrator suffers more space penalty if he decides to adopt $\mathcal{T}_1$. An administrator can make a decision about the balance between the robustness and the overhead by referring to the evaluation results.

Example IV: Parallel Control-Flow Obfuscating Transformations

We introduce a parallel control-flow obfuscating transformation, proposed in [38], as an example. The original program used in Example IV is the same one used in Example III. To formalize the transformation, we first create and insert a dummy code block into ψ. Then a fork is inserted to generate parallel execution. Example IV presents two specific transformations with the same composition and sequence of the atomic operators but with different target code blocks.

$$\mathcal{T}_3 = \langle O_D^S(\cdot, B_2), O_F^2(\cdot, S_2)\rangle$$

$$\mathcal{T}_4 = \langle O_D^S(\cdot, S_3), O_F^2(\cdot, C_1^D)\rangle$$

Apply the 1^{st} specified parallel control-flow obfuscation, $\mathcal{T}_3$

- Running $O_D^S(\cdot, B_2)$:

$$V \leftarrow V \cup \{C_1^D\},$$
$$E \leftarrow (E - \{(S_2, B_2)\}) \cup \{(S_2, C_1^D), (C_1^D, B_2)\}.$$

We create a dummy function, *kidfunc()*, as C_1^D, where *kidfunc()* is described in the following.

```
void kidfunc (int* t)
{
     int t1=*t;
     while(t1-- & (t1%10)!=0);
}
```

Since C_1^D is placed in the loop led by B_1, $|range(\psi, B_1)|$ is increased by 1. We obtain $comp(\psi) = |range(\psi, B_1)| + |range(\psi, B_2)| = 6 + 1 = 7$.

- Running $O_F^2(\cdot, S_2)$:

$$V \leftarrow V \cup \{F_1, J_1\},$$
$$E \leftarrow (E - \{(C_1^D, B_2), (S_2, C_1^D), (B_1, S_2)^T\}) \cup$$
$$\{(F_1, S_2), (F_1, C_1^D), (J_1, B_2), (B_1, F)^T, (S_2, J_1), (C_1^D, J_1)\}.$$

After inserting $O_F^2(\cdot, S_2)$, S_2 and C_1^D are specified as the immediate successors of the fork F_1 such that S_2 and C_1^D could be executed in parallel. Now ψ_3 has been generated. By Eq. 5.3 and Eq. 5.7, $scomp(\psi_3) = |range(\psi_3, B_1)| + |range(\psi_3, B_2)| = 9$ and $pcomp(\psi_3) = |range(\psi_3, F_1)| = 2$. $comp(\psi_3)$ is equal to 11 by summing up $scomp(\psi_3)$

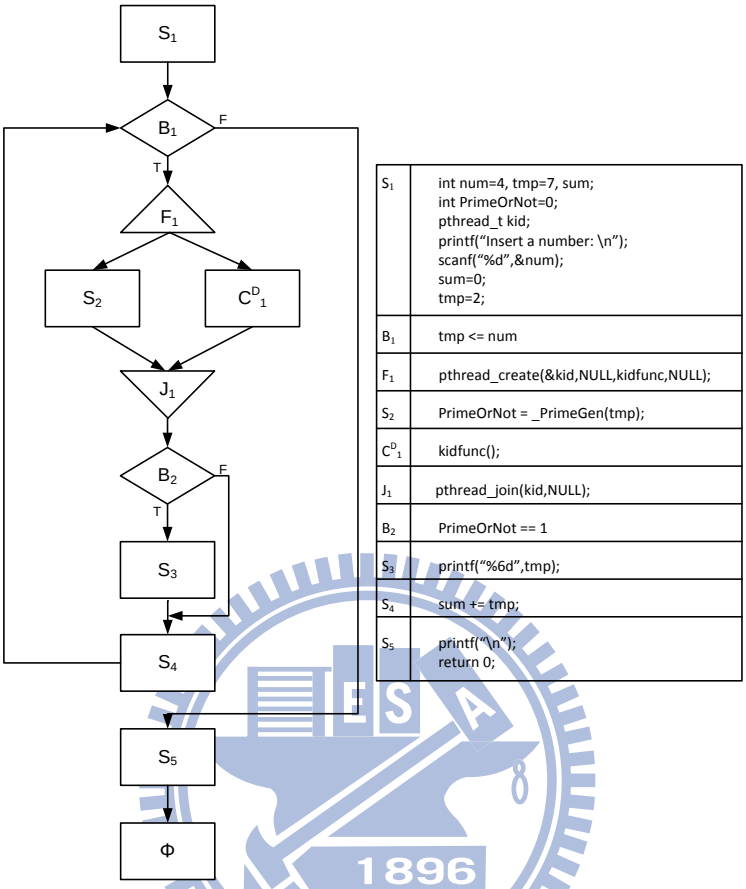


Figure 6.7: ψ_3: obfuscated result of ψ after applying $\mathcal{T}_3$

and $pcomp(\psi_3)$ according to Eq. 5.6 with $w_s = 1$ and $w_p = 1$. Figure 6.7 shows ψ_3 after applying $\mathcal{T}_3$ to ψ.

Apply the 2^{nd} specified parallel control-flow obfuscation, $\mathcal{T}_4$

- Running $O_D^S(\cdot, S_3)$:

$$V \leftarrow V \cup \{C_1^D\},$$
$$E \leftarrow (E - \{(B_2, S_3)^{True}\}) \cup \{(B_2, C_1^D)^{True}, (C_1^D, S_3)\}.$$

The example code of C_1^D here is the same as that of $\mathcal{T}_3$. C_1^D is inserted as the true target of B_2 that C_1^D and S_3 are in the loop led by B_2. Hence

$|range(\psi, B_1)|$ and $|range(\psi, B_2)|$ are increased by one, and $comp(\psi)$ is derived as $|range(\psi, B_1)| + |range(\psi, B_2)| = 6 + 2 = 8$.

- Running $O_F^2(\cdot, C_1^D)$:

$$
\begin{aligned}
V \leftarrow & V \cup \{F_1, J_1\}, \\
E \leftarrow & (E - \{(B_2, C_1^D)^{True}, (C_1^D, S_3), (S_3, S_4)\}) \cup \\
& \{(F_1, C_1^D), (F_1, S_3), (J_1, S_4), (B_2, F_1)^{True}, (C_1^D, J_1), (S_3, J_1)\}.
\end{aligned}
$$

$O_F^2(\cdot, C_1^D)$ makes C_1^D and S_3 executed in parallel after F_1. The obfuscated CFG (ψ_4) after applying $\mathcal{T}_4$ is shown in Figure 6.8. The insertion of F_1 contributes the parallelism such that $pcomp(\psi_4) = |range(\psi_4, F_1)| = 2$. The total complexity of ψ_4 is obtained as $|range(\psi_4, B_1)| + |range(\psi_4, B_2)| + |range(\psi_4, F_1)| = 8 + 4 + 2 = 14$.

Evaluation of Example IV To measure the distance between ψ, ψ_3 and ψ_4, we first analyze the edges of their common subgraphs:

$$edge(cs(\psi, \psi_3)) = (S_1, B_1), (B_2, S_3)^T, (B_2, S_4)^F, (S_3, S_4), (S_4, B_1), (B_1, S_5)^F, (S_5, \phi)$$

$$edge(cs(\psi, \psi_4)) = (S_1, B_1), (B_1, S_2)^T, (B_1, S_5)^F, (S_2, B_2), (B_2, S_4)^F, (S_4, B_1), (S_5, \phi)$$

In this example, $|edge(cs(\psi, \psi_3))| = |edge(cs(\psi, \psi_4))| = 7$, and $dis(\psi, \psi_3) = dis(\psi, \psi_4) = \frac{4}{11}$, even the common subgraphs of ψ, ψ_3 and ψ, ψ_4 are not the same. We then evaluate the robustness of ψ_3 and ψ_4 from the potency perspective by Eq. 5.1. We derive the potency that $\mathcal{T}_3$ results in: $pot(\psi, \psi_3) = \frac{11}{6} - 1 = \frac{5}{6}$. Similarly, $pot(\psi, \psi_4) = \frac{14}{6} - 1 = \frac{4}{3}$. We obtain $DP(\psi, \mathcal{T}_3) = (\frac{4}{11}, \frac{5}{6})$ and $DP(\psi, \mathcal{T}_4) = (\frac{4}{11}, \frac{4}{3})$. According to the DP vectors, we conjecture that $\mathcal{T}_4$ provides more robust protection to ψ than $\mathcal{T}_3$ since the potency value $\mathcal{T}_4$

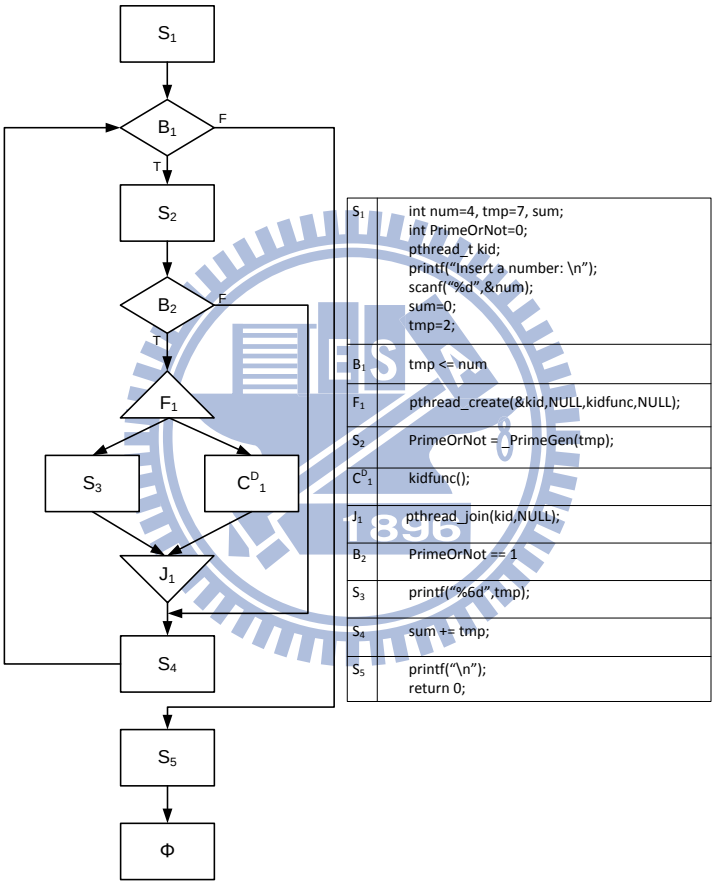


Figure 6.8: ψ_4: obfuscated result of ψ after applying $\mathcal{T}_4$

contributes is larger than that $\mathcal{T}_3$ produces, while the two transformations lead to the same distance value.

Advanced Evaluation of Example IV In the previous paragraphs, we calculate the DP vectors from a coarse-grained perspective since we view the called function, $kidfunc()$, as a single code block without further parsing the function into a separate sub-CFG. In this way, we may underestimate the potency that the newly inserted $kidfunc()$ contributes. Here, we present a relatively fine-grained evaluation of the total complexity of ψ_3 by Eq. 5.4, Eq. 5.5 and Eq. 5.6.

As the steps taken in the previous sections, we first parse $kidfunc()$ into ψ_5 that $\psi_5 = (C^E, (V, E))$ where $C^E = S_1^K$, $V = \{S_1^K, B_1^K, S_2^K, \phi\}$ and $E = \{(S_1^K, B_1^K), (B_1^K, S_2^K)^T, (S_1^K, B_1^K), (B_1^K, \phi)^F\}$. Here, we use a superscript K to emphasize that the code blocks are parsed from $kidfunc()$ in C_1^D. ψ_5 is exhibited in Figure 6.9. Since there are two condition expressions in B_1^K, we further consider the complexity that multiple condition expressions yield by computing the compound range of B_1^K. We obtain $|crange(\psi_5, B_1^K)| = 2 \times (2 + 2 - 1) = 6$ by Eq. 5.5.

We now integrate the total complexity of ψ_5 into the new total complexity of ψ_3. $range(\psi_3, F_1)$ is $\{S_2, S_1^K, S_2^K, B_1^K\}$ if we parse $kidfunc()$, contained in C_1^D, into ψ_5 with the code blocks S_1^K, S_2^K, B_1^K. $pcomp(\psi_3)$ thus becomes 4. In addition, $|range(\psi_3, B_1)|$ increases due to the code blocks of ψ_5 that $|range(\psi_3, B_1)| = 10$. Now, $scomp(\psi_3) = |range(\psi_3, B_1)| + |range(\psi_3, B_2)| + |crange(\psi_5, B_1^K)| = 10 + 1 + 6 = 17$. Hence a fine-grained total complexity of ψ_3, $comp(\psi_3) = 17 + 4 = 21$, has been obtained. $pot(\psi, \psi_3)$ is accordingly

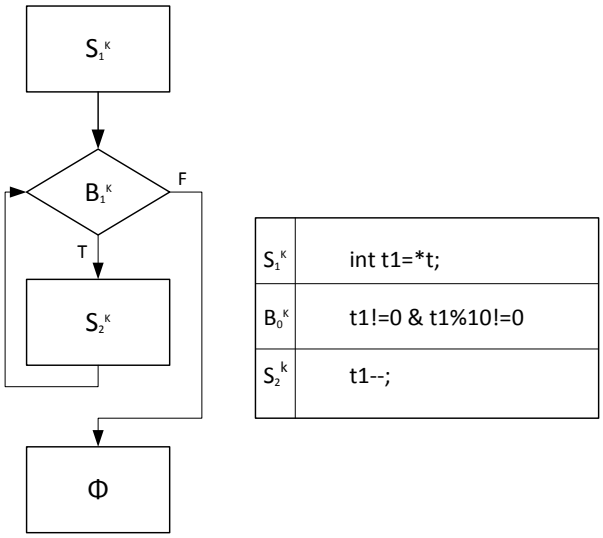


Figure 6.9: ψ_5: CFG of the function $kidfunc()$

derived as $\frac{21}{6} - 1 = \frac{5}{2}$ which is larger than $\frac{5}{6}$ described in the paragraph "Evaluation of Example IV." The discrepancy implies that the advanced evaluation adopts more information and derives a distinct result. By the advanced evaluation, we can measure the effect that a newly inserted dummy code block causes in more details.

6.3 Summary

This chapter presented several examples to demonstrate our assessment methods. In Example I and Example II, we clarified the usage and steps of the proposed method for wireless risk assessment. We conducted several experiments to launch an eavesdropping attack against two different wireless networks, Ex1-1 and Ex1-2, where Ex1-1 is unprotected, but Ex1-2 is protected by WPA2-PSK. The attack succeeds to sniff the communication sessions in Ex1-1, but fails in Ex1-2. We obtained the total impact severity of Ex1-1 (4.2120, HIGH risk) and of Ex1-2 (3.1911, LOW risk) by our risk assessment

method. The derived risk values confirm the realistic experiment results. In addition, we showed the risk value of the entire wireless network can be calculated without repeating the redundant steps when the topology or the configuration of the network changes. The examples verified that our wireless risk assessment method is feasible for the real-world situation.

In Example III and Example IV, we formalized and evaluated control-flow obfuscating transformations by our method. Example III explained how our method helps an administrator designate an effective transformation for protecting a specified program in terms of protection capability and space penalty. Moreover, we illustrated the evaluation of parallel control-flow obfuscating transformations by Example IV. This example verified the flexibility and distinguishability of our method. Different sequences of atomic operators and different specified target blocks lead to different transformations. Therefore, our method can not only formalize existing control-flow obfuscating transformations but also help design and measure new transformations.

Chapter 7

Conclusions and Future Work

Assessment of cyber security has been a long-standing challenge to the research community. Nevertheless, there is an imperative need for a practical security assessment method which is supportive of controlling and managing security. In this dissertation, we showed our first attempt at the quantitative assessments of cyber security. We proposed assessment methods to assist an administrator or a developer in assessing cyber security in a methodical manner, from establishing a formal representation to deriving a numerical assessment result.

Our assessment methods are separated along two dimensions, external and internal attacks, to meet specific requirements for distinct scenarios. We dived into the two dimensions and studied the deficiencies of the existing security assessment methods; then we presented a wireless risk assessment method and an evaluation method for estimating software robustness for each dimension.

Our wireless risk assessment method measures network risk in considera-

tion of dynamics of a wireless network. We designed a 4-layer risk analytical hierarchy to model wireless network risk from the perspectives of security requirements, external attacks and configurations. Due to the design of clearly separated layers and the design of a hierarchy per device, the computing load of assessing risk of a changing wireless network is reduced since only the related layers and hierarchies have to be calculated and developed. Our method diminishes the time complexity in the network assessment at a considerable sacrifice that wireless risk is assessed from a comparatively coarse-grained viewpoint. The assessment result can be used as the first perimeter of controlling and managing security of a network, especially a dynamic network. Then, the administrator can further use other methods to probe potential attack paths and to mitigate security risk. A holistic security assessment and management can thus be achieved by combining the existing solutions with ours.

As for evaluating cyber security in terms of internal attacks, we started from evaluating software robustness in terms of control-flow obfuscation. We presented a framework for representing control-flow obfuscating transformations and evaluating software robustness enhanced by the transformations. We showed that, with a graph-based representation, many existing control-flow obfuscating transformations can be represented as a composition of atomic operators. The atomic operators can not only describe the present transformations but also help to design and construct new ones which may offer different levels of software robustness to a program. We have also proposed new metrics (distance and potency) for quantifying the effects of these transformations upon a program. The metrics has been designed from the

viewpoint of static analysis, and we recognize they serve merely heuristic, general indicators of security. However, we view our approach as a first step towards evaluating the trade-off between the robustness and overheads caused by control-flow obfuscating transformations. We believe our formal framework with the metrics is beneficial in avoiding suffering intolerable overheads, which can be estimated at the design stage, prior to implementation of a more robust but too costly version of an obfuscated program.

We evaluated cyber security from multiple aspects and provided practical metrics for system administrators in a systematic manner. This dissertation is our first attempt, and we recognize there is still space for improvement to reach absolutely quantitative assessment. In order to fulfill the requirements of realistic security assessment, there are, however, some issues that are interesting and need to be further explored. We summarize the issues as follows:

- *Coherence of databases.* To provide a fair or even close to fair evaluation, risk assessment heavily depends on information collected from multiple databases and experts. A holistic risk assessment method should be able to consider the discrepancy between databases or expert opinions. More study is required to evaluate the consistency between the data, and to integrate the risk value with the consistency.

- *Estimation of obfuscation overheads.* The side effects of obfuscating a software program include not only the increased code size but also the slowed-down execution performance. Therefore, more study is needed on how best to compromise between security and performance over-

heads.

- *Attacks, scenarios and protection mechanisms.* We believe that no single security measurement or metric is able to satisfy the requirements of security control and management in the real world. A thorough security assessment has to be considered from aspects of different attacks, scenarios and protection mechanisms. In this dissertation, we simply discussed the evaluation of control-flow obfuscation, while a formal evaluation method for other types of obfuscation, such as data obfuscation and layout obfuscation, are also desired. Another interesting direction of future work would be the design of a framework for combining multiple security measurements or metrics.

- *Absolutely quantitative assessment.* In this dissertation, the range of a derived risk value varies with the number of attacks and number of devices because attacks are the threat sources impacting a network and more devices within a network sensibly imply more potential attack surfaces. We recognize the current design has not achieved absolutely quantitative assessment since we need extra information, such as a scale or a mapping table, to grasp the implication of a numerical risk value for an individual network. However, this is simply our first attempt at devising a quantitative and semantic reference for a network administrator. Further research on the design of a fixed-range variable representing risk of various networks could be conducted based on our present result.

- *Human efforts.* There is always a hope that system administrators

easily infer the realistic efforts that an attacker should invest, such as time and money, from the proposed security measurements and metrics. The inference is a pressing need for system administrators to designate security strategies in a more effective way. A solution to determining the inference is required despite it is still an open issue.

Bibliography

[1] A. Jaquith, *Security Metrics Replacing Fear, Uncertainty, and Doubt*. Addison Wesley, 2007.

[2] C. Phillips and L. P. Swiler, "A Graph-Based System for Network Vulnerability Analysis," *Proceedings of Workshop on New Security Paradigms (NSPW '99)*, pp. 71–79, Jan. 1999.

[3] S. Jha, O. Sheyner, and J. Wing, "Two Formal Analyses of Attack Graphs," *Proceedings of the 15th Computer Security Foundation Workshop*, pp. 49–63, 2002.

[4] V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing, "Ranking Attack Graphs," *Proceedings of Recent Advances in Intrusion Detection*, 2006.

[5] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup, "A Weakest-adversary Security Metric for Network Configuration Security Analysis," *Proceedings of the 2nd ACM Workshop on Quality of Protection*, pp. 31–38, 2006.

[6] L. Wang, A. Singhal, and S. Jajodia, "Toward Measuring Network Security Using Attack Graphs," *Proceedings of the 2007 ACM Workshop on Quality of Protection*, pp. 49–54, 2007.

[7] L. Wang, A. Singhal, and S. Jajodia, "Measuring the Overall Security of Network Configurations using Attack Graphs," in *Proceedings of the 21th IFIP WG 11.3 Working Conference on Data and Applications Security*, 2007.

[8] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic Security Risk Management Using Bayesian Attack Graphs," *IEEE Transactions on Dependable and Secure Computing (unpublished)*, 2011.

[9] D. M. Zhao, J. H. Wang, J. Wu, and J. F. Ma, "Using Fuzzy Logic and Entropy Theory to Risk Assessment of the Information Security," *Proceedings of 4th International Conference on Machine Learning and Cybernetics*, pp. 2248–2253, Aug. 2005.

[10] D. M. Zhao, J. H. Wang, J. Wu, and J. F. Ma, "Fuzzy Risk Assessment of the Network Security," *Proceedings of 2006 International Conference on Machine Learning and Cybernetics*, pp. 4400–4405, Aug. 2006.

[11] D. Zhao, C. Wang, and J. Ma, "A Risk Assessment Method of the Wireless Network Security," *Journal of Electronics*, vol. 24, no. 3, pp. 428–432, May 2007.

[12] Z. Wang and H. Zeng, "Study on the Risk Assessment Quantitative Method of Information Security," *Proceedings of 3rd International Con-*

ference on Advanced Computer Theory and Engineering (ICACTE), pp. 529–553, 2010.

[13] X. Zhang, Z. Huang, G. Wei, and X. Zhang, "Information Security Risk Assessment Methodology Research: Group Decision Making and Analytic Hierarchy Process ," *Proceedings of 2nd World Congress on Software Engineering (WCSE)*, pp. 157–160, 2010.

[14] T. L. Saaty, "How to Make a Decision: The Analytic Hierarchy Process," *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, 1990.

[15] O. S. Vaidya and S. Kumar, "Analytic Hierarchy Process: An Overview of Applications," *European Journal of Operational Research*, vol. 169, no. 1, pp. 1–29, 2006.

[16] P. Falcarin, C. Collberg, M. Atallah, and M. Jakubowski, "Guest Editor's Introduction: Software Protection," *IEEE Software*, vol. 28, no. 2, pp. 24–27, March–April 2011.

[17] C. Collberg, C. Thomborson, and D. Low, "A Taxonomy of Obfuscating Transformations," Univ. Auckland, New Nealand, Tech. Rep. 148, 1997.

[18] C. Wang, J. Davidson, J. Hill, and J. Knight, "Protection of Software-Based Survivability Mechanisms," *Foundations of Intrusion Tolerant Systems*, pp. 273–282, 2003.

[19] F. Kazuhide, K. Shinsaku, and T. Toshiaki, "An Obfuscation Scheme Using Affine Transformation and Its Implementation," *Transactions of*

Information Processing Society of Japan, vol. 47, no. 8, pp. 2556–2570, 2006.

[20] I. V. Popov, S. K. Debray, and G. R. Andrews, "Binary Obfuscation Using Signals," *Proceedings of the 16th USENIX Security Symposium*, pp. 275–290, 2007.

[21] A. Majumdar, S. Drape, and C. Thomborson, "Slicing Obfuscations: Design, Correctness and Evaluation," *Proceedings of the 2007 ACM workshop on Digital Rights Management (DRM'07)*, pp. 70–81, Oct. 2007.

[22] C. Collberg and C. Thomborson, "Watermarking, Tamper-proofing, and Obfuscation - Tools for Software Protection," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 735–746, 2002.

[23] C. Collberg and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison Wesley, 2009.

[24] M. E. Pate-Cornell, "Fault Trees vs. Event Trees in Reliability Analyssi," *Risk Analysis*, vol. 4, no. 3, pp. 177–186, 1984.

[25] W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie, "Fault Tree Analysis, Methods, and Applications: A Review," *IEEE Transactions on Reliability*, vol. 34, no. 3, pp. 194–203, 1985.

[26] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated Generation and Analysis of Attack Graph," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 273–284, May 2002.

[27] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-Attack Graph Generation Tool," *Proceedings of Information Survivability Conference & Exposition II*, pp. 307–321, June 2001.

[28] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," *Computer Networks and ISDN Systems*, vol. 30, pp. 107–117, 1998.

[29] Y. Liu and H. Man, "Network Vulnerability Assessment using Bayesian Networks," *Proceedings of the SPIE*, vol. 5812, pp. 61–71, 2005.

[30] M. Frigault and L. Wang, "Measuring Network Security Using Bayesian Network-Based Attack Graphs," *Proceedings of 32nd Annual IEEE International Computer Software and Applications Conference*, pp. 698–703, 2008.

[31] R. Dantu, K. Loper, and P. Kolan, "Risk Management Using Behavior Based Attack Graphs," *Proceedings of International Conference on Information Technology: Coding and Computing*, vol. 1, pp. 445–449, 2004.

[32] R. Dantu, P. Kolan, and J. ao Cangussu, "Network Risk Management Using Attacker Profiling," *Security and Communication Networks*, vol. 2, pp. 83–96, 2009.

[33] X. Ou, W. F. Boyer, and M. A. McQueen, "A Scalable Approach to Attack Graph Generation," *Proceedings of 13th ACM Conference on Computer and Communications Security (CCS'06)*, pp. 336–345, 2006.

[34] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," *Proceedings of 9th ACM Conference on Computer and Communication Security*, pp. 217–224, Nov. 2002.

[35] P. Ammann, J. Pamula, and R. Ritchey, "A Host-Based Approach to Network Attack Chaining Analysis," *21st Annual Computer Security Applications Conference (ACSAC'05)*, pp. 72–84, 2005.

[36] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, S. Vadhan, and K. Yang, "On the (Im)possibility of Obfuscating Programs ," *Lecture Notes in Computer Science*, pp. 1–18, 2001.

[37] B. Lynn, M. Prabhakaran, and A. Sahai, "Positive Results and Techniques for Obfuscation," in *EUROCRYPT '04*, 2004.

[38] T. Hou, H. Chen, and M. Tsai, "Three Control Flow Obfuscation Methods for Java Software," *IEE Proceedings Software*, vol. 153, no. 2, pp. 80–86, Jan 2006.

[39] T. László and A. Kiss, "Obfuscating C++ Programs via Control Flow Flattening," *Annales Universitatis Scientiarum de Rolando Etvs Nominatae - Sectio Computatorica*, May 2008.

[40] J. Cappaert and B. Preneel, "A General Model for Hiding Control Flow," *Proceedings of 10th ACM Workshop on Digital Rights Management*, pp. 35–42, 2010.

[41] S. Drape, A. Majumdar, and C. Thomborson, "Slicing Aided Design of Obfuscating Transformations," *Proceedings of the International Conference on Computng and Information Systems (ICIS 2007)*, pp. 1019–1024, 2007.

[42] "Dotfuscator," PreEmptive Solutions Inc. [Online]. Available: http://www.preemptive.com/products/dotfuscator

[43] "Dasho," PreEmptive Solutions Inc. [Online]. Available: http://www.preemptive.com/products/dasho

[44] "Zelix klassmaster," http://www.zelix.com/klassmaster/.

[45] "Proguard," http://proguard.sourceforge.net/.

[46] N. Naeem, M. Batchelder, and L. Hendren, "Metrics for Measuring the Effectiveness of Decompilers and Obfuscators," *Proceedings of 15th IEEE International Conference on Program Comprehension (ICPC'07)*, pp. 253–258, 2007.

[47] B. Anckaert, M. Madou, B. D. Sutter, B. D. Bus, K. D. Bosschere, and B. Preneel, "Program Obfuscation: A Quantitative Approach," *Proceedings of 3rd Workshop on Quality of Protection (QoP'07)*, pp. 15–20, 2007.

[48] A. Majumdar, S. Drape, and C. Thomborson, "Metrics-Based Evaluation of Slicing Obfuscations," *Proceedings of 3rd International Symposium on Information Assurance and Security*, pp. 472–477, 2007.

[49] M. Ceccato, M. D. Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, "Towards Experimental Evaluation of Code Obfuscation Techniques," *Proceedings of 4th Workshop on Quality of Protection*, pp. 39–46, 2008.

[50] M. Ceccato, M. D. Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, "The Effectiveness of Source Code Obfuscation: An Experimental Assessment," *Proceedings of IEEE 17th International Conference on Program Comprehension (ICPC'09)*, pp. 178–187, May 2009.

[51] M. Madou, B. Anckaert, B. D. Bus, K. D. Bosschere, J. Cappaert, and B. Preneel, "On the Effectiveness of Source Code Transformations for Binary Obfuscation," *Proceedings of the International Conference on Software Engineering Research and Practice (SERP06)*, 2006.

[52] S. Udupa, S. Debray, and M. Madou, "Deobfuscation: Reverse Engineering Obfuscated Code," *Proceedings of 12th Working Conference on Reverse Engineering (WCRE 2005)*, 2005.

[53] A. Majumdar, "Design and Evaluations of Software Obfuscations," Ph.D. dissertation, Department of Computer Science, University of Auckland, New Zealand, 2008.

[54] M. D. Preda and R. Giacobazzi, "Control Code Obfuscation by Abstract Interpretation," *Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, pp. 301–310, 2005.

[55] G. Stonebumer, A. Goguen, and A. Feringa, "Risk Management Guide for Information Technology Systems," National Institute of Standards and Technology, Special Publication 800-30, 2002.

[56] *ISO/IEC 27001:2005 – Information Technology – Security Techniques – Information Security Management Systems – Requirements*, ISO/IEC Std.

[57] *ISO/IEC 27005:2008 – Information Technology – Security Techniques – Information Security Risk Management*, ISO/IEC Std.

[58] "IEEE Standard for Information Technology-Telecommunications and Information Exchange between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, 2007.

[59] A. H. Lashkari, F. Towhidi, and R. S. Hosseini, "Wired Equivalent Privacy (WEP)," *Proceedings of International Conference on Future Computer and Communication (ICFCC 2009)*, pp. 492–495, 2009.

[60] A. H. Lashkari, M. Mansoor, and A. S. Danesh, "Wired Equivalent Privacy (WEP) versus Wi-Fi Protected Access (WPA)," *Proceedings of*

2009 International Conference on Signal Processing Systems, pp. 445–449, 2009.

[61] M. S. Ahmed, E. Al-Shaer, and L. Khan, "A Novel Quantitative Approach for Measuring Network Security," *Proceedings of 27th IEEE International Conference on Computer Communications (INFOCOM 2008)*, pp. 13–18, Apr. 2008.

[62] B. A. Cota and R. G. Sargent, "Automatic Lookahead Computation for Conservative Distributed Simulation," CASE Center, Tech. Rep. 8916, 1989.

[63] B. Cota, D. Fritz, and R. Sargent, "Control Flow Graphs as a Representation Language," *Proceedings of Simulation Conference*, pp. 555–559, 1994.

[64] P. D. Stotts and Z. N. Cai, "Hierarchical Graph Models of Concurrent CIM Systems," *Proceedings of IEEE Workshop on Languages for Automation: Symbiotic and Intelligent Robots*, pp. 100–105, 1988.

[65] J. Cheng, "Complexity Metrics for Distributed Programs," *Proceedings of 4th International Symposium on Software Reliability Engineering*, pp. 132–141, Nov. 1993.

[66] C. Collberg, C. Thomborson, and D. Low, "Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs," *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98)*, pp. 184–196, 1998.

[67] T. Toyofuku, T. Tabata, and K. Sakurai, "Program Obfuscation Scheme using Random Numbers to Complicate Control Flow," *IEIC Technical Report*, Jan 2005.

[68] J. Ge, S. Chaudhuri, and A. Tyagi, "Control Flow Based Obfuscation," *Proceedings of 5th ACM Workshop on Digital Rights Management*, pp. 83–92, Nov 2005, USA.

[69] "Chilling Effects Clearinghourse," Last updated: Dec. 12, 2008. [Online]. Available: http://www.chillingeffects.org

[70] H. Bunke and K. Shearer, "A Graph Distance Metric Based on the Maximal Common Subgraph," *Pattern Recognition Letters*, vol. 19, pp. 255–259, March 1998.

[71] W. Wallis, P. Shoubridge, M. Kraetz, and D. Ray, "Graph Distances Using Graph Union," *Pattern Recognition Letters*, Jan 2001.

[72] H. Zuse, *Software Complexity: Measures and Methods*. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1991.

[73] S. M. Shatz, "Towards Complexity Metrics for Ada Tasking," *IEEE Transactions on Software Engineering*, vol. 14, no. 8, pp. 1122–1127, 1988.

[74] W. A. Arbaugh, N. Shankar, and Y. C. J. Wan, "Your 802.11 Wireless Network Has No Clothes," *IEEE Wireless Communications*, vol. 9, no. 6, pp. 44–51, Dec. 2002.

[75] T. Karygiannis and L. Owens, "Wireless Network Security: 802.11, Bluetooth and Handheld Devices," National Institute of Standards and Technology, Special Publication 800-48, 2002.

[76] J. Bellardo and S. Savage, "802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions," *Proceedingsof 12th USENIX Security Symposium (SSYM'03)*, pp. 15–28, 2003.

[77] D. Welch and S. Lathrop, "Wireless Security Threat Taxonomy," *Proceedings of IEEE Workshop on Information Assurance United States Military Academy*, pp. 76–83, June 2003.

[78] "National Vulnerability Database," Last updated: 09/25/2011. [Online]. Available: http://nvd.nist.gov/

[79] "Common Vulnerabilities and Exposures," Last updated: 09/07/2011. [Online]. Available: http://cve.mitre.org/

Curriculum Vitae

Hsin-Yi Tsai received the B.S., and M.S. degrees in Electrical and Control Engineering from the National Chiao Tung University, Taiwan in 2005, and 2007 respectively. She is currently pursuing the Ph.D. degree at the Institute of Electrical Control Engineering of National Chiao Tung University. Hsin-Yi was a student visiting scholar at UC Berkeley in 2008. She was also a visiting researcher at the Multimedia Communications Lab (KOM), Darmstadt University of Technology, Germany during 2010 and 2011. Her research interests include evaluation of protection techniques, risk assessment of networks, and design of security metrics. She has been a member of the Phi Tau Phi Society since 2007.

List of Publications

Journal Papers

1. **Hsin-Yi Tsai**, Melanie Siebenhaar, André Miede, Yu-Lun Huang and Ralf Steinmetz, "Threat as a Service? The Impact of Virtualization on Cloud Security," *accepted by IT Professional*, Oct. 2011

2. **Hsin-Yi Tsai** and Yu-Lun Huang, "An Analytic Hierarchy Process-Based Risk Assessment Method for Wireless Networks," *accepted by IEEE Transactions on Reliability*, April 2011

3. Yu-Lun Huang, Alvaro Cardenas, Saurabh Amin, Song-Zyun Lin, **Hsin-Yi Tsai** and S. Shankar Sastry, "Understanding the Physical and Economic Consequences of Attacks on Control Systems," *International Journal of Critical Infrastructure Protection*, vol. 2, no. 3, pp. 73–83, Oct. 2009

4. **Hsin-Yi Tsai**, Yu-Lun Huang and David Wagner, "A Graph Approach to Quantitative Analysis of Control-Flow Obfuscating Transformations," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 2, pp. 257–267, June 2009

Conference Papers

1. **蔡欣宜**, 王繼偉, 陳柏廷, 黃育綸, 謝續平, "基於虛擬裝置之無線網路安全測試平台," in Crypotology and Information Security Conference 2010, Hsinchu, Taiwan

2. Zong-Syun Lin, Alvaro Cardenas, Saurabh Amin, **Hsin-Yi Tsai**, Yu-Lun Huang and Shankar Sastry, "Security Analysis for Process Control Systems," in the 16^{th} ACM Conference on Computer and Communications Security (CCS 2009), Nov. 9–13, 2009

3. Zong-Syun Lin, Alvaro Cardenas, **Hsin-Yi Tsai**, Saurabh Amin, Yu-Lun Huang and S. Shankar Sastry, "Understanding the Physical Consequences of Attacks on Control Systems," in Proceedings of the 3^{rd} Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection, March 22–25, 2009

4. Y. L. Huang, J. D. Tygar, H. Y. Lin, L. Y. Yeh, **H. Y. Tsai**, K. Sklower, S. P. Shieh, C. C. Wu, P. H. Lu, S. Y. Chien, Z. S. Lin, L. W. Hsu, C. W. Hsu, C. T. Hsu, Y. C. Wu, M. S. Leong, "SWOON: A Testbed for Secure Wireless Overlay Networks," in USENIX Workshop on Cyber Security Experimentation and Test (CSET'08), San Jose, July 28, 2008

5. Y. L. Huang, F. S. Ho, **H. Y. Tsai**, H. M. Kao, "A Control Flow Obfuscation Method to Discourage Malicious Tampering of Software Codes," in the proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS'06), March 21–24, 2006