NAIST-IS-DD461044

Doctoral Dissertation

Computation Theoretic Approaches in Intrusion Detection and Access Control

Jing Wang

December 6, 2006

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Jing Wang

Thesis Committee:

Professor Hiroyuki Seki	(Supervisor)
Professor Minoru Ito	(Co-supervisor)
Associate Professor Yuichi Kaji	(Co-supervisor)

Computation Theoretic Approaches in Intrusion Detection and Access Control *

Jing Wang

Abstract

In this thesis, we focus on two problems in system security: the IDS (intrusion detection system) partition deployment problem and the verification problem of HBAC (history-based access control) programs. The common methodology used in this thesis is computation theory. A version of the IDS partition deployment problem can be captured as a special kind of matching problem in graph theory. To solve the verification problem of HBAC programs, on the other hand, we use the formal language theory, especially the theory of finite state automata and context-free grammars.

In chapter 2, the IDS partition deployment problem is defined and an efficient algorithm for a simplified version of the problem is proposed. The IDS partition deployment problem is the problem of computing the number and deployment positions of distributed IDSs and dividing a given attack scenario to minimize the load of each IDS on a given network topology without sacrificing the detection capability of the original attack scenario. We successfully reduce the deployment problem to a newly introduced matching problem for weighted bipartite graphs. It is shown that the deployment problem for any state transition IDS can be solvable in deterministic polynomial time. We also prove a related problem, the minimum IDS partition deployment problem, is NP-complete.

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD461044, December 6, 2006.

In chapter 3, we first propose a formal model for Abadi-Fournet style access control called HBAC program. We also show that the expressive power of HBAC programs is stronger than programs with stack inspection. Next, we define the security verification problem for HBAC programs. The verification problem is reduced to the emptiness test of context-free languages. It is shown that the problem is solvable in deterministic polynomial time under a reasonable assumption while the problem is EXPTIME-complete in general. Finally, we propose a few optimization techniques used in verification of HBAC programs. Experimental results show that practical HBAC programs can be verified within reasonable time and space.

Keywords:

computer security, intrusion detection system, stack inspection, history-based access control, verification, model checking, computation theory

List of Publications

1. Journal Papers

   (1) Jing Wang, Naoya Nitta and Hiroyuki Seki: "An Efficient Method for Optimal Probe Deployment of Distributed IDS," IEICE Transactions on Information and Systems, E88-D(8), pp.1948–1957, Aug. 2005.

2. International Conference (Reviewed)

   (3) Jing Wang, Yoshiaki Takata and Hiroyuki Seki: "HBAC: A Model for History-based Access Control and Its Model Checking," 11th European Symposium on Research in Computer Security (ESORICS 2006), Lecture Notes in Computer Science 4189, pp.263–278, Sept. 2006.

3. Workshops

   (4) Jing Wang, Naoya Nitta and Hiroyuki Seki: "Towards an Optimal Probe Deployment for Network IDS," Technical Report of IEICE, SS2003-31, Nov. 2003 (in Japanese).

   (5) Jing Wang, Naoya Nitta and Hiroyuki Seki: "An Efficient Method for Optimal Probe Deployment of Distributed IDS," 2004 Symposium on Cryptography and Information Security (SCIS2004), pp.1035–1040, Jan. 2004 (in Japanese).

   (6) Jing Wang, Yoshiaki Takata and Hiroyuki Seki: "A Formal Model for Access Control Based on Execution History," Technical Report of IEICE, SS2004-63, March 2005.

   (7) Jing Wang, Yoshiaki Takata and Hiroyuki Seki: "A Formal Model for Access Control Based on Execution History and Its Model Checking," 8th Programming and Programming Language Workshop (PPL2006), p.183, March 2006.

   (8) Jing Wang, Yoshiaki Takata and Hiroyuki Seki: "An Efficient Model Checking Method for Programs with History-based Access Control,"

Technical Report of IEICE, SS2006-38, pp.34–39, Aug. 2006.

Acknowledgements

First, and foremost, I would like to thank Professor Hiroyuki Seki for his continuous support and encouragement and guidance of the work. Also foremost, I would like to thank Professor Minoru Ito for his invaluable comments and helpful suggestions concerning this thesis. I also wish to thank Associate Professor Yuichi Kaji for his valuable comments. I would like to express my sincere gratitude to Assistant Professor Yoshiaki Takata for his support and advice throughout the research. I am very grateful to Assistant Professor Naoya Nitta at Konan University for his support and advice in an early stage of the research. Finally, I would like to thank all the members of Seki laboratory.

Contents

List of Publications iii
Acknowledgements v

1 Introduction **1**

2 An Efficient Method for Optimal Probe Deployment of Distributed IDS **8**

2.1. Introduction 8
2.2. Distributed Network-oriented IDS 9
2.3. IDS Partition Deployment Problem 10
2.3.1 Load Minimization 11
2.3.2 Definition of IDS Partition Deployment Problem 12
2.3.3 Simplified IDS-PDP 14
2.4. Approach to IDS-PDP 16
2.4.1 Algorithm for Message Monitor Problem 16
2.4.2 Correctness of the Algorithm 22
2.4.3 Relation to the bipartite b-matching 26
2.5. Probe Number Minimization IDS-PDP 27
2.5.1 Definition of Probe Number Minimization IDS-PDP . . . 27
2.5.2 Complexity of Probe Number Minimization IDS-PDP . . . 28
2.6. Conclusion of chapter 2 30

3 HBAC: A Model for History-based Access Control and Its Model Checking **32**

3.1. Introduction 32
3.2. HBAC Program 33
3.3. Comparison with Stack Inspection 40
3.4. Model Checking HBAC Program 42
3.5. Optimization of Model Checking Algorithm 47
3.5.1 Basic Idea 47
3.5.2 Optimization 1: Rules with Reachable Symbols 48
3.5.3 Optimization 2: Precomputing Current Permissions 48
3.5.4 Optimization 3: Exact Computation of Current Permissions 50
3.6. Experiments 52
3.7. Conclusions of chapter 3 55

4 Conclusion **57**

References 59

List of Figures

1.1 A UDP race attack [38] 4
1.2 A scenario of a UDP race attack 4
1.3 Decomposition of the scenario in Fig. 1.2 5

2.1 NetSTAT 10
2.2 A structure of a probe 11
2.3 Example of IDS-PDP 12
2.4 An approach of IDS-PDP for state transition type IDS 15
2.5 Different solutions of message monitor problem and IDS-PDP for the same instance 16
2.6 Construction of a bipartite graph 17
2.7 A computation of matching 21
2.8 An example of probe number minimization IDS-PDP 28
2.9 Reduction from L_{vc} to L_{pids} 30

3.1 An HBAC program 35
3.2 Chinese wall policy 40
3.3 A basic program 42
3.4 Method $P[q,k]$ 45
3.5 Method $R[k,\gamma_j,q',k']$ 46
3.6 Method I 46
3.7 Sample HBAC program. 50
3.8 Initial dependency graph for $X_{1,\{p_1,p_3,p_4\}}$. 51
3.9 Final dependency graph for $X_{1,\{p_1,p_3,p_4\}}$. 51

3.10 On-line banking system 53
3.11 Verification time for $\pi_c(k)$ and $\pi_o(k)$ 55

List of Tables

2.1 Time complexity 30

3.1 Modification of current permissions 37
3.2 Verification profiles of sample programs 54

Chapter 1

Introduction

Nowadays, with rapid growth of computer networks and diversity of computer systems, our secret resources are exposed to various kinds of malicious behaviors and attacks from both inside and outside of a system. Information and computer security is an important research area, of which the aim is to protect users' private information against those threats and to retain the safe and effective function of the system. Security technologies are roughly divided into two sub-areas, namely, modern cryptographic technology and system security. The former, including secret-key and public-key cryptographic systems, has been widely used in today's computer and network systems and many protocols based on cryptography have been developed to make so-called e-business, e-trade and e-government safer and more secure. System security, on the other hand, mainly concerns controlling the behavior of a user, its process, and other related objects, based on the system's security policy in a few layers in the system such as the network, the operating system, and the language runtime environment. Among others, *intrusion detenction* and *access control* are major research topics in system security.

Intrusion Detection System (IDS) is a security protection mechanism that detects malicious accesses to an intra-network or secret resources by monitoring the traffic of a network, the state of a program, and so on. If the observed behavior matches the pre-defined signature or deviates from the pre-defined normal behavior, then the IDS alarms us of the danger. As expained in detail below, IDSs

can be devided into host-based IDSs and network-based IDSs.

Access control has its origin in protection of resources such as files and devices in operating systems based on file permission or access control lists (ACL). Recently, this kind of control technology has been evolved very much. As explained later, stack inspection is one of the well-known security protection mechanisms in such runtime environments as the Java virtual machine and the Common Language Runtime. History-based access control has also been introduced as an extension of stack inspection to overcome the problems of the latter. However, it is difficult to manually check whether these security protection mechanisms are sufficient to assure the security of a system against malicious accesses when the behaviors of the system and the attacker are complicated.

In this thesis, we focus on two problems in system security. One is the IDS partition deployment problem and the other is the security verification problem of HBAC programs. The common methodology used in this thesis is computation theory. A version of the IDS partition deployment problem can be captured as a special kind of matching problem in graph theory. We successfully reduce the deployment problem to a newly introduced matching problem for weighted bipartite graphs. To solve the seruity verification problem of HBAC programs, on the other hand, we use the formal language theory, especially the theory of finite state automata and context-free grammars. The verification problem is reduced to the emptiness test of context-free languages.

In both of the topics, we first define the problems formally, describe the algorithms to solve them and then analyse the computational complexity of these algorithms as well as analysing the lowerbound of the complexity of the problems themselves in the latter topic. In the rest of this chapter, the background and research motivation will be explained in some detail.

IDS partition deployment problem. An Intrusion Detection System (IDS) is a known mechanism that warns a network administrator whenever it detects a misuse access by monitoring the entire network traffic or scanning server log files. According to the underlying detection technique, IDSs are roughly divided into

anomaly detection type and *misuse detection* type. An anomaly detection IDS detects a impersonaton of a legitimate user u by comparing the current behavior of u on the network with the information in a behavior profile of u that has been extracted in advance. On the other hand, a misuse detection IDS searches for a misuse access of an anonymous user by comparing the user's behavior with known attack scenarios. An attack scenario defines what kind of event sequences on a network or servers is misuse. According to the method used for expressing attack scenarios, misuse detection IDSs are further divided into four classes[31]: (1) single event, (2) event sequence, (3) state transition diagram, and (4) others. Class (3) is wider than class (2) and class (2) is wider than class (1).

A misuse detection IDS that monitors network traffic is called a *network-oriented* IDS (e.g., EMERALD[32], NetSTAT[38]). With respect to a network-oriented IDS, sometimes it is unrealistic to detect all intrusions with a single host because:

- when the network traffic becomes heavy, it becomes harder to monitor the entire network traffic in real time with a single host;
- if the network events such as packets and messages related to an intrusion are visible only in different links on the network, then we cannot detect the whole intrusion with a single host (see the following example).

For these reasons, it is necessary for network-oriented IDSs that the detection tasks should be distributed to several hosts while preserving detection capability.

For example, let us recall a UDP race attack shown in [38] (Figs.1.1 and 1.2). There are four hosts, **kubrick**, **fellini**, **chaplin** and **jackson** on the network. **Kubrick**, **fellini** and one network interface of **chaplin** are on link L_1, and another network interface of **chaplin** and **jackson** are on link L_2. **Fellini** is a server that provides an NFS service to **kubrick**. In this case, since authentication of a UDP-based service such as NFS is based on the host address or host name, **jackson** can execute a UDP race attack by the following message sequence:

1. First, **kubrick** sends a message $m1$ including an NFS request to **fellini**.

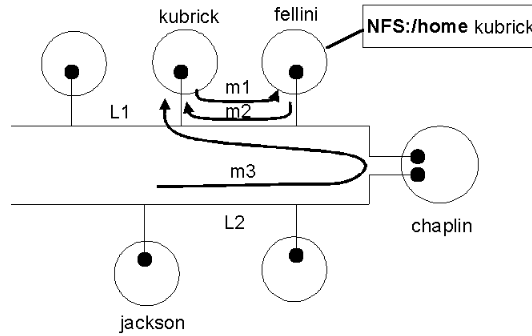


Figure 1.1. A UDP race attack [38]

2. **Jackson** sends to **kubrick** a message $m3$ the IP source address of which is changed to **fellini**'s.

3. If **kubrick** receives $m3$, before the reply message $m2$ from **fellini**, then **kubrick** will accept $m3$ as a legitimate reply.

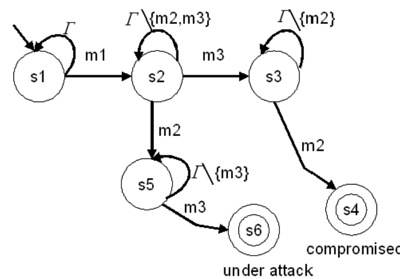


Figure 1.2. A scenario of a UDP race attack

In Fig. 1.2, we show an attack scenario of a UDP race attack as a state transition diagram based on [38]. Generally, an attack scenario is specified independently from the properties of individual network. The states $s1, s2, \ldots$, and $s6$ of the attack scenario represent the snapshots of the security-relevant properties and resources of the network (e.g., file ownership, the states of the network services, and so on). The state $s1$ is the initial state and $s4$ is a success state of the attack. A transition between states is represented as a directed edge labeled

with a delivered message. In this thesis, for technical convenience, we regard the set of all edge labels in each attack scenario as the set of all messages (denoted as Γ in Fig. 1.2) delivered in the network rather than the set of signature actions which are messages needed to complete the attack successfully. Hence, a loop is placed at each state of a state transition diagram to preserve the meaning of the attack scenario.

In the example shown in Figs. 1.1 and 1.2, since we can detect spoofing of the message $m3$ only at link $L2$ on the delivery route of $m3$, an IDS should be deployed on $L2$. Another IDS should be deployed on link $L1$ because only there both the messages $m1$ and $m2$ can be detected. Hence, it is necessary to deploy two IDSs on the network to detect the whole of the attack specified by the attack scenario in Fig. 1.2. For these IDSs, the attack scenario in Fig. 1.2 is decomposed into two subscenarios shown in Fig. 1.3. In this case, new messages $m1'$, $m2'$ and $m3'$ are exchanged between the two IDSs to synchronize the detection tasks of these IDSs. For example, the IDS deployed on link $L1$ monitors the message $m1$, and sends message $m1'$ to the IDS deployed on link $L2$. If the IDS on $L2$ accepts $m1'$ and monitors $m3$ in this order, then it sends a message $m3'$ to the IDS on $L1$. If the IDS on $L1$ receives $m3'$ from the IDS on $L2$ and monitors $m2$, then it warns a network administrator.

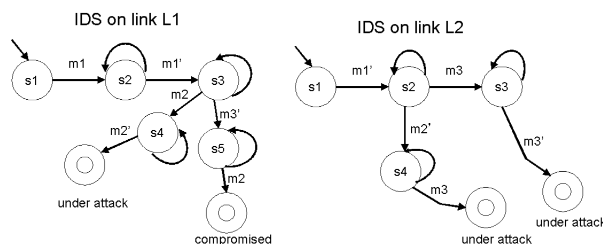


Figure 1.3. Decomposition of the scenario in Fig. 1.2

Although NetSTAT proposes a technique for decomposing attack scenarios and allocating detection tasks to several distributed IDSs on a network, the algorithm used in NetSTAT is based on heuristics [38] and is not always optimum.

In chapter 2, we formally define this problem as the IDS partition deployment problem, and propose an efficient algorithm for a simplified version of the problem. The IDS partition deployment problem is the problem of computing the number and deployment positions of distributed IDSs and dividing a given attack scenario to minimize the load of each IDS on a given network topology without sacrificing the detection capability of the original attack scenario. The simplified version of the problem is defined as a new graph theoretic problem by extracting out the essential part of the original problem. We successfully reduce the deployment problem to a newly introduced matching problem for weighted bipartite graphs. It is shown that the deployment problem for any syate transition IDS can be solvable in deterministic polynomial time. We also prove a related problem, the minimum problem IDS partition deployment problem, is NP-complete.

Verification problem for HBAC programs. Stack inspection is now broadly used as a dynamic access control infrastructure in such runtime environments as the Java virtual machine [17] and the Common Language Runtime. However, it has been pointed out that stack inspection is not sufficient for security assurance since the stack does not retain security information on the invoked methods for which execution is finished. To solve this problem, a few access control models have been proposed [1, 15, 36]. Common feature of these works is that the history of execution such as method invocation and resource access is used for access control, and the history is not always forgotten even if the surrounding method execution is completely finished. Schneider [36] defines an enforceable security policy as a prefix-closed nonempty set of event sequences, and also defines security automata, which exactly recognize enforceable policies. Fong [15] introduces several subclasses of security automata and compares the expressive power of these subclasses. In particular, Fong defines shallow history automata with finite state space, and shows that the class of policies recognized by shallow history automata is incomparable with that of stack inspection. Another novel approach is proposed by Abadi and Fournet [1]. As in stack inspection, a target system for access control is an object-oriented recursive program: A set of permissions is

assigned statically (before runtime) to each method; current permissions are modified each time a method is invoked. Generally, current permissions can depend on all the methods executed so far. This forms a contrast to access control based on stack inspection, which completely cancels the effect of the finished method execution. In [1], an implementation built on the top of $C^\sharp$ runtime environment is reported. However, formal verification methods for the model of [1] have not been investigated, except [3].

In chapter 3, we propose a formal model for Abadi-Fournet style access control called HBAC program. An HBAC program is a directed graph where a node represents a program point and an edge represents a control flow. We also show that the expressive power of HBAC programs is stronger than that of programs with stack inspection. We also define the security verification problem for HBAC programs and show that the problem is solvable in deterministic polynomial time under a reasonable assumption while the problem is EXPTIME-complete in general. Finally, we propose a few optimization techniques used in verification of HBAC programs. Experimental results show that practical HBAC programs can be verified within reasonable time and space.

Chapter 2

An Efficient Method for Optimal Probe Deployment of Distributed IDS

2.1. Introduction

In this chapter, we formally define this problem as the IDS partition deployment problem, and propose an efficient algorithm for a simplified version of the problem. The IDS partition deployment problem is the problem of computing the number and deployment of distributed IDSs and dividing a given attack scenario to minimize the load of each IDS on a given network topology without sacrificing the detection capability of the original attack scenario. The simplified version of the problem is defined as a new graph theoretic problem by extracting out the essential part of the original problem.

The rest of the chapter is organized as follows. In section 3, we define the IDS partition deployment problem and the simplified version of the problem. In section 4, we design an algorithm which computes the optimal solution of the simplified problem and we prove the correctness of the algorithm. In section 5, we define the Probe number minimization IDS partition deployment problem, and prove this problem is NP-complete. We provide conclusions and address future

work in section 6.

Related work:

Another problem concerned with misuse detection IDS is vulnerability analysis which verifies the safety of network system and attack scenarios. Generally, it is difficult to set up appropriate attack scenarios that can prevent both false acceptance and false rejection in a misuse detection IDS[2]. Therefore, to decide whether the attack scenarios are appropriately set up, verification techniques [23][33] using model checking [8] are proposed. Model checking is one of the automatic verification techniques which search the state space of a system exhaustively to decide whether the system satisfies given specifications. It also generates an execution sequence of the system as a counterexample if the system does not satisfy one of the specifications. Ritchey et al. [33] propose a method which decides by model checking whether a network system with vulnerability satisfies safety specifications, and outputs counterexamples as successful attacks if the specifications are not satisfied. Furthermore, Jha et al.[23] propose a method which extracts safe attack scenarios for IDS from all the successful examples outputted by model checkers. However, in these researches, only single event type attack scenarios are considered, and this type of attack scenarios do not have enough detection capability. On the other hand, if we are allowed to use a more powerful attack scenario such as USTAT [21] and NetSTAT [38], then from the input of model checkers [23] [33] (i.e., state transition diagrams which represent the behavior of the network and the successful states of attacks) we can directly construct a safe attack scenario, and thus safety verification would become meaningless.

2.2. Distributed Network-oriented IDS

IDSs are classified into host-oriented IDSs, which monitor the log files of the operating system, and network-oriented IDSs, which monitor the network traffic such as TCP/IP packets. Furthermore, according to the number of IDSs deployed on the network, we can classify IDSs into stand-alone type and distributed type.

NetSTAT ([38]) is proposed as a network-oriented distributed IDS. In NetSTAT, each deployed IDS is called a *probe*. NetSTAT detects a misuse access with each probe analyzing the traffic based on distributed attack subscenario and sending results to each other (Fig. 2.1).

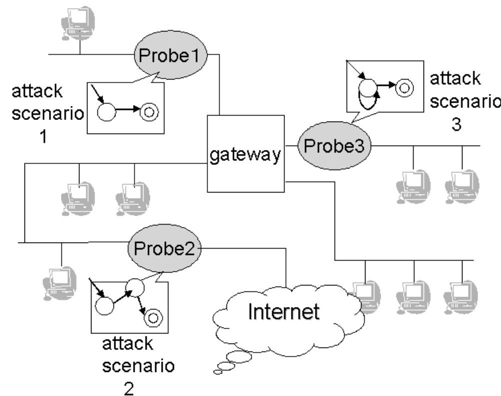


Figure 2.1. NetSTAT

Since the most suitable deployment of probes on the network and decomposition of an attack scenario greatly depend on the topology of the network, NetSTAT provides a tool (Analyzer) to acquire such information from an original attack scenarios and the topology of the network. However, the algorithm is based on heuristics.

2.3. IDS Partition Deployment Problem

Before defining the IDS partition deployment problem, we should discuss what are required for this problem. There are two objectives to decompose an attack scenario and to allocate them to many probes:

1. the inevitability of message passing;

2. the load minimization.

The inevitability of message passing means that for a given network topology, every message in the original attack scenario must be monitored at least one probe under any routing condition. The load minimization is discussed in the next section. In addition to these objectives, the following property must be satisfied:

(3) the preservation of detection capability.

The preservation of detection capability means that the detection capability should remain the same even if an attack scenario is decomposed.

2.3.1 Load Minimization

Many network-oriented IDSs consist of the message filter part, which extracts only messages related to attack scenarios from the traffic, and the misuse detection processing part, which performs misuse detection based on extracted messages (Fig. 2.2). We can assume that the processing time of message filter part can be

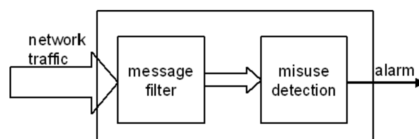


Figure 2.2. A structure of a probe

ignored compared with the processing time of misuse detection processing part. Now, let c denote the time needed by a certain IDS to perform misuse detection on one message. If the number of messages extracted per unit time exceeds 1/c, then it is impossible for this IDS to perform misuse detection in real time. On the other hand, if we assume all different kinds of messages appear at the same rate in ordinal traffic, then the number of the messages extracted per unit time is proportional to the kinds of messages that should be monitored. Thus, in this paper, we define the load minimization as the minimization of the maximum number of the different kinds of messages which each probe monitors. A typical

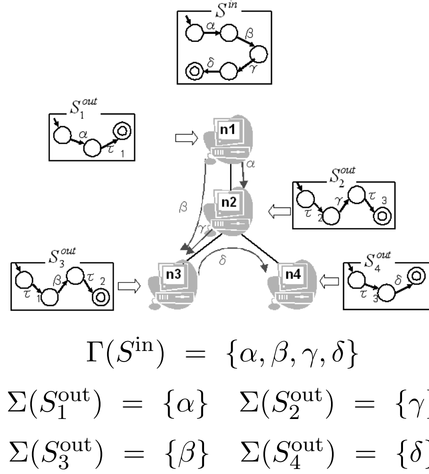


Figure 2.3. Example of IDS-PDP

case that minimizing the maximum number of the kinds of messages does not mean minimizing the load is a case that few kinds of messages appear extremely frequently in the network such as a situation under DoS (Denial of Service) attack. However, we consider such a situation under Dos attack should be detected and avoided by techniques completely different from the one discussed here, and we do not take such a situation into account in this paper.

2.3.2 Definition of IDS Partition Deployment Problem

As mentioned in section 1, IDSs can be classified by the class of the attack scenarios which can be handled. To make the IDS partition deployment problem independent of the type of attack scenarios, in the following, we first define the generalized IDS partition deployment problem. The topology of the network is represented as an undirected graph $T = (N, E)$ (Fig. 2.3), where the set N of vertices represents the set of network interfaces provided by the hosts on which a probe can be deployed, and the set $E \subseteq N \times N$ of edges represents a connection relation between the interfaces. We denote the original attack scenario by S^{in}, and the set of attack subscenarios which are distributed to probes by $\mathcal{S}^{\text{out}} = \{S_1^{\text{out}}, S_2^{\text{out}}, ..., S_n^{\text{out}}\}$ where $n = |\mathcal{S}^{\text{out}}|$ (without concrete descriptions). A probe

deployment mapping $a : \{1, 2, ..., |\mathcal{S}^{\rm out}|\} \rightarrow N$ is an injection. For example in Fig. 2.3, $a(1) = n_1$ since $S_1^{\rm out}$ is assigned to n_1. The set of all messages contained in the attack scenario S is denoted by $\Gamma(S)$ and the set of all message sequences detected by S is denoted by $L(S)$. In addition, extending the definition of L, for the set $\mathcal{S}$ of attack subscenarios, the set of message sequences detected by cooperating subscenarios in $\mathcal{S}$ is also denoted by $L(\mathcal{S})$. The source and destination of each message are given by mappings $r_1, r_2 : \Gamma(S^{\rm in}) \rightarrow N$, respectively.

Definition 1 *Generalized IDS partition deployment problem (IDS-PDP)*
input : $S^{\rm in},\ T,\ r_1, r_2$
output : $n,\ a,\ \mathcal{S}^{\rm out}$,
where

- $n = |\mathcal{S}^{\rm out}|$,
- $L(S^{\rm in}) = L(\mathcal{S}^{\rm out})$ *(preservation of detection capability),*
- $max\{|\Gamma(S_i^{\rm out})|\ \ |\ 0 < i \leq n\}$ *is minimum (load minimization),*
- *if* $\alpha \in \Gamma(S_i^{\rm out})$*, then* $a(i)$ *exists on every path of* T *from* $r_1(\alpha)$ *to* $r_2(\alpha)$ *(inevitability of message passing).*

Note that the meaning of the preservation of detection capability depends on formulation of the attack scenario.

Example 2.3.1 In Fig. 2.3, we show an instance of IDS-PDP for the state transition type IDS. In a state transition type IDS, an attack scenario is given by a state transition diagram. $\mathcal{S}^{\rm out} = \{S_1^{\rm out}, S_2^{\rm out}, S_3^{\rm out}, S_4^{\rm out}\}$ and the deployment of probes shown in Fig. 2.3 is a solution of this instance. For example, as scenario $S_3^{\rm out}$ monitors message β, $S_3^{\rm out}$ is assigned to n_3, which is on the path from source n_1 to destination n_3 of β. The probes are cooperated as follows. If the probe deployed at the host n_1 monitors the message α, then the probe sends a synchronous message τ_1 to the probe deployed at the host n_3. If the latter probe receives τ_1 and monitors the message β, then it sends a synchronous message τ_2 to the probe deployed at the host n_2. In this way, a sequence $\alpha\beta\gamma\delta$ of messages

can be detected by these four probes. As the number of the kinds of messages monitored by $S_1^{\mathrm{out}}, S_2^{\mathrm{out}}, S_3^{\mathrm{out}}$ and S_4^{out} are $2, 3, 3$ and 2, respectively, the maximum load is 3. This satisfies the condition of the load minimization.

2.3.3 Simplified IDS-PDP

Without specifying the type of attack scenario in generalized IDS-PDP, the preservation of the detection capability cannot be formally defined. However, even if the formulation of the attack scenario is not specified, we can see that the preservation of the detection capability condition requires the following condition to hold:

$$\Gamma(S^{\mathrm{in}}) \subseteq \bigcup_i \Gamma(S_i^{\mathrm{out}}).$$

Furthermore, by ignoring [1] the load yielded by the synchronization between probes, we can define another type of IDS partition deployment problem independently of type of the attack scenarios as follows. Here, let the distribution function $\Sigma : \mathcal{S}^{\mathrm{out}} \rightarrow 2^{\Gamma(S^{\mathrm{in}})}$, which assigns a subset of messages to each attack subscenario, be a function such that $\Sigma(S_i^{\mathrm{out}}) = \Gamma(S_i^{\mathrm{out}}) \cap \Gamma(S^{\mathrm{in}})$ and if $i \neq j$ then $\Sigma(S_i^{\mathrm{out}}) \cap \Sigma(S_j^{\mathrm{out}}) = \emptyset$.

Definition 2 *Simplified IDS-PDP (Message monitor problem)*
input : $\Gamma(S^{\mathrm{in}})$, T, r_1, r_2
output : n, a, Σ,
where

- $n = |\mathcal{S}^{\mathrm{out}}|$,
- $\Gamma(S^{\mathrm{in}}) = \bigcup_i \Sigma(S_i^{\mathrm{out}})$,
- $max\{|\Sigma(S_i^{\mathrm{out}})| \mid 0 < i \leq n\}$ *is minimum,*

[1] Since the probability that a synchronization τ occurs between probes is exponentially decreasing in proportion to the length of message sequence from the initial state to τ in the attack scenrio, we can ignore it.

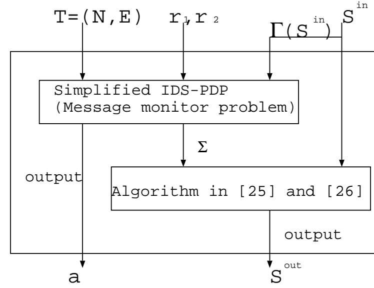


Figure 2.4. An approach of IDS-PDP for state transition type IDS

- *if* $\alpha \in \Sigma(S_i^{\text{out}})$*, then* $a(i)$ *exists on every path of* T *from* $r_1(\alpha)$ *to* $r_2(\alpha)$*.*

Example 2.3.2 Example 2.3.1 is also an instance of the message monitor problem. As an input, the set $\Gamma(S^{\text{in}})$ of the kinds of messages included in the scenarios S^{in} is given. Σ and the deployment of probes shown in Fig. 2.3 is a solution of this instance. Each message included in S^{in} is monitored by at least one subscenario in $\mathcal{S}^{\text{out}}$. As the numbers of the kinds of messages monitored by $S_1^{\text{out}}, S_2^{\text{out}}, S_3^{\text{out}}$ and S_4^{out} are $1, 1, 1$ and 1, respectively, the maximum load is 1.

In example 2.3.2, the deployment of probes determined by the message monitor problem agrees with that by the generalized IDS-PDS. However they do not always agree in general. In case of a single event type IDS, a solution of the generalized IDS-PDP just corresponds to a solution of the message monitor problem. On the other hand, in case of the state transition type IDS, from S^{in} and a solution Σ of the message monitor problem, we can compute the partition of the attack scenario in polynomial time by using the algorithm presented in [24, 25](see Fig. 2.4). The time complexity of the algorithm in [24, 25] is $O(|N| \cdot R^2)$, and the number of synchronization messages generated by the algorithm is $O(F \cdot |\Gamma(S^{in})|)$, where R is the number of the transitions of S^{in} and F is the number of the states of S^{in}. However, in general, a solution obtained by this method is not always a solution of the generalized IDS partition deployment problem (see Fig. 2.5).

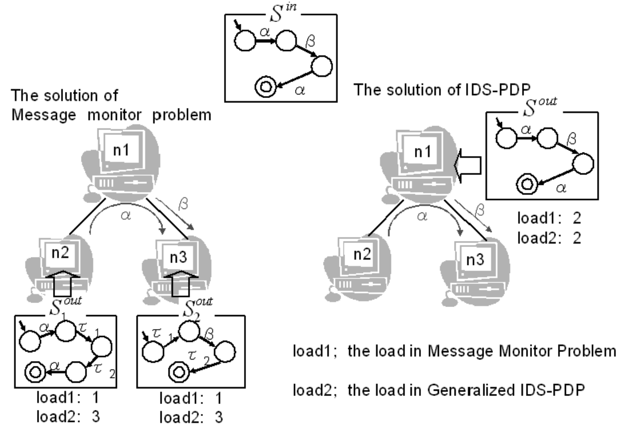


Figure 2.5. Different solutions of message monitor problem and IDS-PDP for the same instance

2.4. Approach to IDS-PDP

2.4.1 Algorithm for Message Monitor Problem

The approach proposed in this paper to solve the message monitor problem is as follows.

1. First, we construct a bipartite graph $G = (V^+, V^-, E')$ from the input $\Gamma(S^{\text{in}})$, $T = (N, E)$, and r_1, r_2 of the message monitor problem as follows. Let deployment mapping $\rho : \Gamma(S^{\text{in}}) \to 2^N$ be a mapping which maps a given message m to the set of hosts which m inevitably passes through (it is possible to compute ρ in T by depth first search in polynomial time).

   - $V^- = \Gamma(S^{\text{in}})$.
   - $V^+ = \bigcup_{m \in \Gamma(S^{\text{in}})} \rho(m)$,
     where $V^+ \subseteq N$, and $V^- \cap V^+ = \emptyset$.
   - $E' = \{(v_i, w_i) \mid v_i \in \Gamma(S^{\text{in}}), w_i \in \rho(v_i)\}$.

2. Second, for G, compute a minimum maximum overlapped matching defined below.

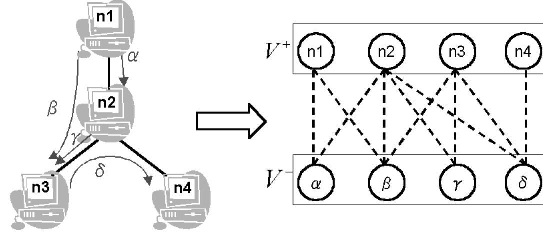


Figure 2.6. Construction of a bipartite graph

Example 2.4.1 In Fig. 2.6, we show the bipartite graph constructed from the input of the message monitor problem shown in Fig. 2.3.

- Let the messages α, β, γ and δ correspond to the vertices of V^- in the bipartite graph.
- Also, let the inevitably passed hosts n_1, n_2, n_3 and n_4 for the messages correspond to the vertices of V^+ in the bipartite graph.
- Put edges between each message v of V^- and each inevitably passed hosts for v. For example, since the message δ must pass the host n_2, n_3 and n_4, the edges between δ and n_2, n_3 and n_4 exist.

Note that for each $v_i \in V^-$ there exists at least one edge which is incident to v_i since each message v_i inevitably passes through its source $r_1(v_i)$ and destination $r_2(v_i)$. In the rest of the paper, however, we allow the case that some vertices in V^- has no incident edge since our algorithm can find a minimum maximum overlapped matching even in such a case.

Definition 3 *Let $G = (V^+, V^-, E')$ be a bipartite graph and $v \in V^+ \cup V^-$ be a vertex in G. The number of edges incident to v is called the degree of v and written by $deg(v)$. For a subset M of E', the number of edges which belong to M and are incident to v is denoted as $deg(v, M)$. A subset M of E' is called a $(n:1)$-matching and n is called the overlap of M in G if $deg(v^+, M) \leq n$ for every $v^+ \in V^+$ and $deg(v^-, M) \leq 1$ for every $v^- \in V^-$. When we are not*

interested in n of a $(n:1)$*-matching M in G, we call M an overlapped matching in G. A* $(n:1)-$*matching M in G is maximal if there is no* $(n:1)-$*matching* M' *in G such that* $|M'| > |M|$*. An overlapped matching M in G is maximum if there is no overlapped matching* M' *in G such that* $|M'| > |M|$*. A minimum maximum overlapped matching in G is a maximum overlapped matching in G such that the overlap is minimum. A vertex* $v^+ \in V^+$ *(or* $v^- \in V^-$*) is M-free if* $deg(v^+, M) < n$ *(or* $deg(v^-, M) = 0$*).*

Although a $(n:1)$-matching is a special case of the b-matching[37], to our purpose, it is sufficient to use $(n:1)$-matching. Relation between the problem in $(n:1)$-matchings and the problem in b-matchings is discussed in section 4.3.

By definition, if both of a $(n_1:1)$-matching M_1 and a $(n_2:1)$-matching M_2 are maximal and $n_1 \leq n_2$, then $|M_1| \leq |M_2|$. Also, for two maximum overlapped matchings M_1 and M_2, $|M_1| = |M_2|$ holds. Since we only consider a finite bipartite graph G, there always exists a minimum maximum overlapped matching in G.

Let $G = (V^+, V^-, E')$ be the bipartite graph constructed from an instance $\langle \Gamma(S^{\text{in}}), T = (N, E), r_1, r_2 \rangle$ of the message monitor problem and let M be a minimum maximum overlapped matching in G. A solution $\langle a, \Sigma \rangle$ of the original problem can be obtained from M as follows: Let $N' = \{v^+ \in V^+ \mid deg(v^+, M) \geq 1\}$ and $m = |N'|$. A probe deployment mapping a is defined as an arbitrary bijection from $\{1, \ldots, m\}$ to N'. The distribution function Σ is defined as $\Sigma(S_i^{\text{out}}) = \{v^- \mid (a(i), v^-) \in M\}$ for $1 \leq i \leq m$. By definition, $deg(a(i), M) = |\Sigma(S_i^{\text{out}})|$. Since M is minimum maximum, the conditions of definition 3.2 are satisfied.

To our purpose, we extend the definition of M-augmenting path [34] as follows.

Definition 4 *Let G be a bipartite graph and M be a* $(n:1)$*-matching in G. Then, a* $(n:1)$*-M-alternating path in G is a path whose edges are alternately in* $E' \backslash M$ *and M. A* $(n:1)$*-M-augmenting path in G is a* $(n:1)$*-M-alternating path such that both of its start edge and end edge are in* $E' \backslash M$*, and both of its start vertex and end vertex are M-free.*

Let $G = (V^+, V^-, E')$ be a bipartite graph. In the following, we show a *matching* algorithm of finding a minimum maximum overlapped matching. The algorithm

keeps track of a subset M of edges and a value k in internal variables. Intuitively, k represents the current lower limit of the overlap of maximum overlapped matchings in G. In the following, an edge belonging to M is denoted as a *solid line* and an edge belonging to $E' \backslash M$ is denoted as a *dashed line*. The solution is the set of solid lines when the algorithm halts. The algorithm runs as follows.

1. Let the initial value of k be $\lceil |V^-|/|V^+| \rceil$ and all edges be dashed lines.
2. For every vertex in V^-, perform the following steps $(3), (4), (5)$.
3. Find a $(k:1)$-M-augmenting path from a vertex in V^- to a vertex in V^+ with depth-first search ((d) of Fig. 2.7).
4. If a $(k:1)$-M-augmenting path is found, then all dashed lines are changed to solid lines and all solid lines are changed to dashed lines on the path, which results in a new $(k:1)$-matching M' with $|M'| = |M| + 1$.
5. If such a path is not found, then k is increased by one, per from the step (3) for the vertex specified by step (3).

```
function dfs(v :vindex; ρ :path) :path;
    var z :vindex; p :↑edge; r :path;
begin
  vertex[v].state :=visited;
  p := vertex[v].adjlist; (adjacent vertex list of v)
  while p ≠ nil do begin
    z := p ↑ .dest;
    if vertex[z].state =novisit then
      if z ∈ V+ and vertex[z].sat < k and
          p ↑ .edgestyle =dashedline then begin
            add z to path ρ; return(ρ)
      end;
      if (z ∈ V+ and vertex[z].sat = k and
         p ↑ .edgestyle =dashedline) or
```

```
            (z ∈ V⁻ and p ↑ .edgestyle =solidline)
        then begin
            add z to path ρ;  r := dfs(z, ρ);
            if r ≠ nil then return(r)
            else delete v from path ρ
        end;
      p := p ↑ .next
    end;
    return(nil)
end;
```

($p \uparrow .edgestyle$ denotes the style of the edge (v, z)).

```
 procedure matching;
    var i, j :vindex;  r :path;
begin
    k := ⌈|V⁻|/|V⁺|⌉; let all edges be dashed lines;      /*  step(1)
    for each vertex i do vertex[i].sat := 0;
    for each i ∈ V⁻ do      /*  step(2)
      if vertex[i].adjlist ≠ nil then begin
        for each vertex j do
            vertex[j].state :=novisit;
        r := dfs(i, nil);      /*  step(3)
        if r ≠ nil then begin      /*  step(4)
          dashed lines are changed to solid lines and
          solid lines are changed to dashed lines in r;
          vertex[i].sat := vertex[i].sat + 1;
          vertex[end vertex of r].sat :=
                      vertex[end vertex of r].sat + 1;
        end
        else begin      /*  step(5)
            k := k + 1;
```

for each vertex j **do**
 $vertex[j].state$:=novisit;
$r := dfs(i, \mathbf{nil})$;
dashed lines are changed to solid lines and
solid lines are changed to dashed lines in r;
$vertex[i].sat := vertex[i].sat + 1$;
$vertex$[end vertex of r]$.sat :=$
 $vertex$[end vertex of r]$.sat + 1$;
end
end
end;

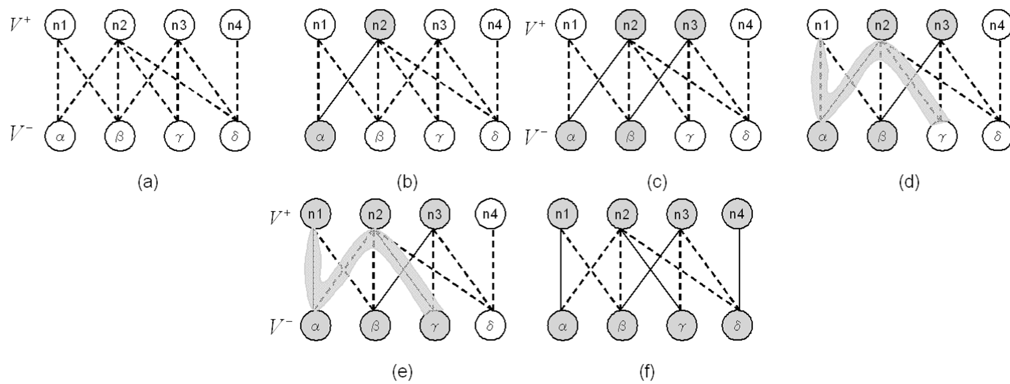


Figure 2.7. A computation of matching

Example 2.4.2 In Fig. 2.7, we show the computation of *matching* algorithm for a bipartite graph given in Fig. 2.6.

(a) Let the initial value k be 1 (since $|V^-| = |V^+| = 4$ and $\lceil |V^-|/|V^+| \rceil = 1$), and all of the edges be dashed. At this time, all of the vertices of V^- and V^+ are M-free.

(b) For a M-free vertex α of V^-, consider edges (α, n_1) and (α, n_2) incident to α. Since both adjacent vertices n_1 and n_2 are M-free, we can choose either

(α, n_1) or (α, n_2). In this example, (α, n_2) is chosen. Then change the type of (α, n_2) from dashed to solid. As a result, α and n_2 become not M-free.

(c) Do the same as α for β. In this case, we choose (β, n_3) and change the dashed line to solid line. Thus, β and n_3 become not M-free.

(d) For the M-free vertex γ, the edges incident to γ are (γ, n_2) and (γ, n_3). Furthermore, n_2 and n_3, the adjacent vertices of γ, are not M-free. In this case, we can find a $(1:1)$-M-augmenting path $\gamma n_2 \alpha n_1$ from γ to a M-free vertex of V^+ by depth first search.

(e) Change the dashed lines to the solid lines and the solid line to the dashed line in the path $\gamma n_2 \alpha n_1$. Then γ and n_1 become not M-free.

(f) After all of the vertices of V^- become not M-free, the solid lines in the bipartite graph is a minimum maximum overlapped matching $\{(\alpha, n_1), (\beta, n_3), (\gamma, n_2), (\delta, n_4)\}$. Consequently, as a solution of this problem, four probes are deployed in n_1, n_2, n_3 and n_4 and monitor the messages α, γ, β and δ, respectively.

2.4.2 Correctness of the Algorithm

In this section, we show the correctness of the algorithm provided in the previous section, and also analyze its time complexity. The first lemma shows a basic property of $(n:1)$-M-augmenting path.

Lemma 1 *Let $G = (V^+, V^-, E)$ be a connected bipartite graph and both M and M' be $(n:1)$-matchings. If $E = M \cup M', M \cap M' = \phi$ and $|M'| > |M|$, then G has a $(n:1)$-M-augmenting path.*

Proof. Since M and M' are $(n:1)$-matchings, for all $v^+ \in V^+,\ 0 \leq deg(v^+, M)$, $deg(v^+, M') \leq n$ hold, and for all $v^- \in V^-, 0 \leq deg(v^-, M)$, $deg(v^-, M') \leq 1$ hold. Since $|M'| > |M|$ and G is a bipartite graph,

$$\sum_{v^+ \in V^+} deg(v^+, M) < \sum_{v^+ \in V^+} deg(v^+, M'),$$

$$\sum_{v^- \in V^-} deg(v^-, M) < \sum_{v^- \in V^-} deg(v^-, M').$$

Hence,

$$\exists v^+ \in V^+, deg(v^+, M) < deg(v^+, M') \leq n,$$
$$\exists v^- \in V^-, deg(v^-, M) < deg(v^-, M') \leq 1. \quad (2.1)$$

That is, both V^+ and V^- include at least one M-free vertex. Since G is a connected graph, for an arbitrary M-free vertex v_0 in V^-, there exists a trail (a path on which no edge appears more than once) from v_0 to every M-free vertex in V^+. Let the longest one of these trails be $v_0 v_1 ... v_{2i-1}$. Note that v_0 is in V^- and v_{2i-1} is in V^+, Here, we prove that a $(n:1)$-M-augmenting path with length no more than $2i-1$ exists by the induction with i.
Basis) If $i = 1$, then $(v_0, v_1) \notin M$. Thus, $v_0 v_1$ is a $(n:1)$-M-augmenting path.
Induction) Assume $i > 1$. Also, let a trail between v_0 and a M-free vertex v' in V^+ with length $2i-1$ be $v_0 ... v''' v'' v'$. Delete all edges $(v''', v''), (v'', v')$ included in all these trails from G. Then, the remaining graph is not connected, and let them be $G_1, G_2, \ldots, G_m$, where for each $j (1 \leq j \leq m)$, let $G_j = (V_j^+, V_j^-, E_j), M_j = M \cap E_j$ and $M'_j = M' \cap E_j$. Since $v'' \in V^-$, the numbers of edges deleted from M' and that from M are the same, $|\cup_j M'_j| > |\cup_j M_j|$ is satisfied. Thus, for some connected component G_j, $|M'_j| > |M_j|$ is satisfied. Since G_j has at least one M-free vertex both in V^+ and in V^- by (1) and it includes v_0 or v' which satisfies the above conditions. For, if G_j does not have v', then there exists a pair of deleted edges $(v''', v''), (v'', v')$ such that v''' is included in G_j. In this case v''' has a trail to v_0 with length $2i-3$ in G_j. Thus, the longest trail which connects two M-free vertices included in G_j is shorter than $2i-1$. By induction hypothesis, a $(n:1)$-M-augmenting path with length no more than $2i-1$ exists. □

The next lemma provides a condition for an overlapped matching to be maximal in terms of augmenting path, which is proved by using lemma 4.1.

Lemma 2 *A $(n:1)$-matching M of a bipartite graph $G = (V^+, V^-, E)$ is maximal if and only if G does not include a $(n:1)$-M-augmenting path.*

Proof. First, we show the only if part. Let $v_0, v_1, ..., v_{2i-1}$ be a $(n:1)$-M-augmenting path, and $M_1 = \{(v_{2k-1}, v_{2k}) \mid 1 \leq k < i\} \subseteq M$ and $M_2 = \{(v_{2k-2}, v_{2k-1}) \mid 1 \leq k \leq i\} \cap M = \emptyset$. Since $M' = (M \cup M_2) \backslash M_1$ is a $(n:1)$-matching of G and $|M'| = |M| + 1$ holds, M is not maximal.

Second, we assume that M is not maximal and show that G includes a $(n:1)$-M-augmenting path. Let $M'(|M'| > |M|)$ be a $(n:1)$-matching, and consider the subgraph G_1 of G whose edges are $(M \cup M') \backslash (M \cap M')$. G_1 contains at least one connected graph $G_2 = (V_2^+, V_2^-, E_2)$ such that $|M_2'| > |M_2|$ where $M_2 = E_2 \cap M$ and $M_2{}' = E_2 \cap M'$. By lemma 4.1, G_2 has a $(n:1)$-M-augmenting path.

□

Definition 5 *Let G be a bipartite graph and M be a $(n:1)$-matching in G. A $(n:1)$-M-augmenting path $v_0 v_1 \ldots v_{2k-1}$ is called normal if (1) $v_0 \in V^-$, (2) every $v_{2i-1} \in V^+$ is not M-free $(1 \leq i < k)$, and (3) $i \neq j$ implies $v_i \neq v_j (0 \leq i, j \leq 2k-1)$.*

Lemma 3 *Let G be a bipartite graph and M be a $(n:1)$-matching in G. A normal $(n:1)$-M-augmenting path exists in G whenever a $(n:1)$-M-augmenting path exists in G.*

Proof. We can construct a normal $(n:1)$-M-augmenting path from a $(n:1)$-M-augmenting path $v_0 v_1 \ldots v_{2k-1}$ in G as follows. (1) If a pair of vertices v_i and v_j which satisfies $v_i = v_j$ appears in the path, remove the path between v_i and v_j, and construct a simple path $v_0 ... v_i v_{j+1} ... v_{2k-1}$. Let $v_{2k'-1}$ be the M-free vertex which occurs at first in V^+ along with the path obtained by (1). Then, $v_0 v_1 ... v_{2k'-1}$ is a normal $(n:1)$-M-augmenting path. □

Lemma 4.3 is used for proving the following lemma, which claims that the algorithm can find an augmenting path whenever it exists.

Lemma 4 *Let $G = (V^+, V^-, E)$ be a bipartite graph and M be a $(n:1)$-matching in G. If some $(n:1)$-M-augmenting paths P started in $v_0 \in V^-$ exists, then the algorithm dfs outputs one of P. Otherwise, the algorithm eventually outputs nil.*

Proof. dfs makes exactly one vertex visited by recursive call. Since the number of vertices is limited, dfs halts. That is, dfs outputs a path and halts only if it finds a normal $(n:1)$-M-augmenting path. Consequently, if a normal $(n:1)$-M-augmenting path in G does not exist, equivalently by lemma 4.3, if a $(n:1)$-M-augmenting path does not exist, then dfs outputs nil.

Conversely, suppose that there exists a $(n:1)$-M-augmenting path $v_0v_1...v_{2k-1}$ in G. We show that *dfs* does not output *nil* by contradiction. Assume that *dfs* outputs *nil* and halts. Then it visits all the vertices connected with v_0 through a normal augmenting path. Since v_{2k-1} is connected with v_0 through $v_0v_1...v_{2k-1}$, v_{2k-1} has already been visited by *dfs*. However, since v_{2k-1} is M-free, whenever *dfs* visits v_{2k-1}, it outputs an augmenting path and halts. □

The following theorem states the correctness of the algorithm, which can be proved by using lemmas 4.2 and 4.4.

Theorem 1 *For a given bipartite graph $G=(V^+,V^-,E)$, the algorithm matching always outputs a minimum maximum overlapped matching M in G and the overlap k of M.*

Proof. By induction on $i=|V^-|$, we prove both the theorem and the claim that every vertex in V^- either is not M-free or has no incident edge when the algorithm halts. Let v be the last visited vertex in V^- by the algorithm *matching*.
Basis) The proof is similar to that of induction step. Hence we omit it here.
Induction) Let G' be a subgraph of G which is obtained by removing v and the set of edges incident to v from G. Let k' and M' be the content of internal variables k and M, respectively, when the algorithm begins to process $v \in V^-$. By induction hypothesis, M' be a minimum maximum $(k':1)$-matching of G'. There are two cases to consider. First, if v has no incident edge in G, then M' is also a minimum maximum overlapped matching in G. In this case, *dfs* is never called, and M' and k' are outputted by *matching*. Second, if v has an incident edge in G, then M' is not a maximum overlapped matching in G because $M' \cup \{e\}$ is also an overlapped matching where e is an edge incident to v. Hence, this case

is further divided into two cases. One case is that G has $(k' : 1)$-matching M''s.t. $|M''| > |M'|$. Then by lemma 4.2, there is a $(k' : 1)$-M'-augmenting path σ in G. By induction hypothesis, every vertex in $V^{-}\backslash\{v\}$ either is not M'-free or has no incident edge. Hence, σ is started from v. By lemma 4.4, *dfs* can find σ and consequently *matching* outputs $(M'\backslash(E(\sigma) \cap M')) \cup (E(\sigma)\backslash M'))$ as a maximum overlapped matching, where $E(\sigma)$ represents the set of all edges in σ. The other case is that G has $(k'' : 1)$-matching M'' s.t. $|M''| > |M'|$ and $k'' > k'$. In this case, it is sufficient to consider $k'' = k' + 1$ because for $k' + 1$, every vertex in V^{+} is M'-free. By lemma 4.2, there is a $(k' + 1 : 1)$-M'-augmenting path σ in G. Similarly to the former case, *dfs* can find σ and consequently *matching* outputs a maximum overlapped matching. □

Lastly, we present the time complexity of the algorithm.

Theorem 2 *The time complexity of the algorithm of message monitor problem is* $O(|\Gamma(S^{\text{in}})| \cdot |N| \cdot (|N| + |E| + |\Gamma(S^{\text{in}})|))$.

Proof. The time complexity of determining the mapping deployment is $O(|\Gamma(S^{\text{in}})| \cdot |N| \cdot (|N| + |E|))$. The time complexity of finding a minimum maximum overlapped matching is $O(|\Gamma(S^{\text{in}})|^2 \cdot |N|)$. Thus, the time complexity of the algorithm of message monitor problem is $O(|\Gamma(S^{\text{in}})| \cdot |N| \cdot (|N| + |E| + |\Gamma(S^{\text{in}})|))$. □

2.4.3 Relation to the bipartite b-matching

As stated before, $(n : 1)$-matching is a special case of b-matching. Since b-matching has been researched widely for a long time, it is valuable to discuss about relation between the the simplified IDS-PDP and the bipartite b-matching problem.

- A maximal $(n : 1)$-matching is considered as a special case of the maximum weighted capacitated b-matching. No b-matching algorithm which are specialized to $(n : 1)$-matching has been reported as far as we know.

More precisely, the optimization criterion which we want to obtain to solve the simplified IDS-PDP is the " minimum maximum overlapped" matching. However, minimum maximum overlapped matching in $(n : 1)$-matching cannot be directly defined in b-matching because b-matching has no parameter corresponding to n.

- The simplified IDS-PDP can be solved by performing binary search based on a general b-matching algorithm. However, the algorithm proposed in this paper is more efficient than such a naive search method.

2.5. Probe Number Minimization IDS-PDP

As a problem concerned with IDS partition deployment problem, in this section, we consider probe number minimization IDS-PDP which minimizes the number of deployed probes instead of minimizing the maximum number of different messages monitored by each deployed probe. In the following, we show that the problem is NP-complete.

2.5.1 Definition of Probe Number Minimization IDS-PDP

A probe cannot be deployed on a certain kind of hosts such as a router. To take such a situation into account, we let $N_\Phi \subseteq N$ denote the set of all hosts on which a probe can be deployed.

Definition 6 *The probe number minimization IDS partition deployment problem.*
input : S^{in}, T, r_1, r_2, N_Φ
output : a, $\mathcal{S}^{\text{out}}$ *, where*

- $L(S^{\text{in}}) = L(\mathcal{S}^{\text{out}})$,
- *minimize* n $(n = |\mathcal{S}^{\text{out}}|)$,
- $a(i) \in N_\Phi$ *holds for every* $i (1 \leq i \leq n)$,

- *if* $\alpha \in \Gamma(S_i^{\mathrm{out}})$*, then* $a(i)$ *exists on every path of* T *from* $r_1(\alpha)$ *to* $r_2(\alpha)$*.*

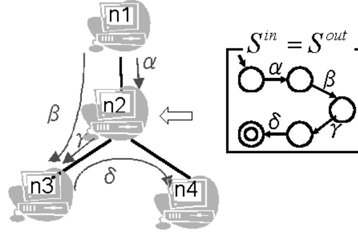


Figure 2.8. An example of probe number minimization IDS-PDP

An example of the probe number minimization IDS-PDP is shown in Fig. 2.8. It is a solution of the probe number minimization IDS-PDP that a probe is deployed in $n2$ and monitors all of the messages.

2.5.2 Complexity of Probe Number Minimization IDS-PDP

The probe number minimization IDS partition deployment problem is an optimization problem. Here, we consider it as a decision problem for a given probe number. This problem is shown NP-complete.

Theorem 3 *The probe number minimization IDS partition deployment problem is NP-complete.*

Proof. We denote the probe number minimization IDS partition deployment problem by L_{pids}, and the vertex cover problem [16] by L_{vc}.

In the following, we first show that L_{pids} is in *NP*. Let the network topology $G' = (N, E')$, the sources and the destinations $(r_1(1), r_2(1)), ..., (r_1(m), r_2(m))$ of messages, and a positive integer k be given as an instance of L_{pids}.

- Nondeterministically guess a deployment of k probes on the hosts. The computation time is $O(k \cdot log(|N_\Phi|))$

- Check whether all of the messages are monitored by all probes deployed the on hosts. The computation time is $O(m \cdot k \cdot log(|N_\Phi|))$

Now, we show how to reduce L_{vc} to L_{pids} in polynomial time. Let $G = (V, E)$ and k be an instance of L_{vc}, where $V = \{v_1, v_2, ...v_n\}$ and $E = \{e_1, e_2, ..., e_m\}$. Then we construct from G an instance of L_{pids} as follows.

- $G' = (N, E')$, where
  $N = V \cup \{v'\}(v' \notin V)$ and $E' = \{(v_i, v') \mid v_i \in V\}$.
- $N_\Phi = V$.
- $\Gamma(S) = E$.
- $r_1 = \langle (v_i, v_j) \mapsto v_i, (v_i, v_j) \in E \rangle$.
- $r_2 = \langle (v_i, v_j) \mapsto v_j, (v_i, v_j) \in E \rangle$.
- $S^{\text{in}} = (\{q_0, q_1, ..., q_m\}, \{e_1, e_2, ..., e_m\}, \delta, q_0, \{q_m\})$, where
  $q_0, q_1, ...$ and q_m are states, q_0 is an initial state, q_m is a final state, δ is a transition function.

An example of the conversion is shown in Fig. 2.9. We first add a new node v', and put the edges between v' and all of the other nodes. Let the source and the destination of each message corresponds to each edge in G. Then we construct an instance of L_{pids}, where the new node is a host on which a probe can not be deployed. This conversion can be performed in polynomial time in the size of G and k.

We show that the above construction correctly reduces L_{vc} to L_{pids}. That is, G has a vertex cover of size k if and only if G' has a deployment of k probes.

(If) Suppose G' has a deployment of k probes. Let N' be the set of probes deployed on the hosts, note that $v' \notin N'$ since $N_\Phi = V$. Since the source and the destination of each message corresponds to each edge in G and every message is monitored by a probe in N', every edge in G is covered by N'.

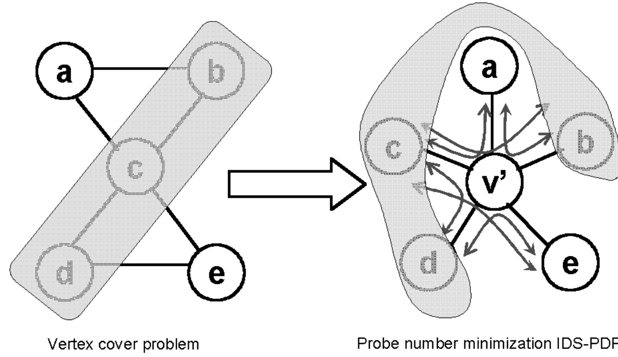


Figure 2.9. Reduction from L_{vc} to L_{pids}

(Only if) Suppose G has a vertex cover V' of size k. Let $e = (v_i, v_j) \in \Gamma(S) = E$ be an arbitrary message. By the definition of r_1 and r_2, the unique path from $r_1(e)(= v_i)$ to $r_2(e)(= v_j)$ is v_i, v', v_j. Since V' is a vertex cover, either $v_i \in V'$ or $v_j \in V'$ holds. Thus, V' is a correct deployment of k probes for the constructed instance.

□

Table 2.1. Time complexity

	IDS partition deployment problem		
Type of IDS	Generalized	Simplified	probe Number
Single	in $O(n^2)^\dagger$		NP-complete
State Transition	solvable	in $O(n^2)^\dagger$	NP-complete

$^\dagger n = |N| + |E|$ ($T = (N, E)$ is an input graph). The size of the attack scenarios is considered as constant.

2.6. Conclusion of chapter 2

We have defined a generalized IDS partition deployment problem which computes the deployment of IDSs, the set of messages which should been monitored by each IDS and a partition of an attack scenario. Furthermore, we simplified this problem and designed an efficient algorithm which computes the optimal

solution of the problem. The complexities of the related problems are summarized in table 2.1. For any state transition IDS, the IDS partition deployment problem (simplified) can be solvable in P, and the probe number minimization IDS partition deployment problem is NP-complete, while nontrivial upperbound and lowerbound of the complexity of the generalized IDS-PDP are unknown at present.

Chapter 3

HBAC: A Model for History-based Access Control and Its Model Checking

3.1. Introduction

In this chapter, we propose a formal model for Abadi-Fournet style access control, called HBAC program (program with History-Based Access Control). An HBAC program is a directed graph where a node represents a program point and an edge represents a control flow. Next we show that the expressive power of HBAC programs is stronger than that of programs with stack inspection. Also we define the security verification problem for HBAC programs and show that the problem is solvable in deterministic polynomial time under a reasonable assumption while the problem is EXPTIME-complete in general. Finally, we propose a few optimization techniques used in verification of HBAC programs. Experimental results show that practical HBAC programs can be verified within reasonable time and space.

Related works. There have been studies on verification of history-based access control [3, 4, 5, 11, 18]. The program model proposed in [4, 5] is a call-by-value λ-

calculus augmented with local policy defined as a regular language of events. For each function (method) call, a new (but statically bound to the function) local policy is imposed in a nested way. They propose a model checking algorithm for a given program and a global security property by reducing the problem to the traditional model checking problem for basic process algebra by removing duplicated local policies caused by recursive calls. The access control mechanism of [4, 5] is an extension of [36, 15] and differs from [1] and ours , i.e., their model do not have explicit dynamic check on permissions. Another access control mechanism based on [36, 15], which simulates a security automaton by inserting dynamic access control codes into a target program, was proposed in [11]. In their later work [18], a type system is used to guarantee that the rewritten program adheres to security policies. Also their model differs from [1] and ours, and they do not deal with model checking problems. The previous work most related to ours is [3] where a program model with explicit dynamic check on permissions and grant/accept constructs is defined. They propose a type and effect system of Volpano-Simth-style [39] and show that a type safe program has a noninterference property. This property is important because one of the main purposes of access control is to avoid leaking undesirable information flow. However, they also do not deal with model checking problems. Moreover, unlike our study, all of these works do not discuss computational complexity needed for verification or optimization issues that are important for implementing a useful verification tool.

3.2. HBAC Program

We will define the syntax and operational semantics of an HBAC program, which resembles but more general than the model in [22, 30]. An HBAC program is just a control flow graph with nodes of three types, call nodes, return nodes and check nodes. The graph is decomposed into methods and each method is given a subset of permissions for access control, called *the static permissions* of the method. A (local) state of a program is a pair $\langle n, C\rangle$ of the current program point n and a subset of permissions C called *the current permissions.* A (global)

configuration is represented by a stack, which is a finite sequence of local states. A call node has two parameters, grant permissions and accept permissions. When a method is called from a configuration $\langle n_1, C_1 \rangle : \ldots : \langle n_k, C_k \rangle$ with the current (call) node n_1 with grant permissions P_G and accept permissions P_A, a new local state $\langle m, C' \rangle$ is pushed onto the stack where m is the entry point of the callee method and C' is the updated current permission obtained by intersecting C_1 with the static permission of the callee method. Furthermore, P_G is temporarily added to the current permissions during the execution of the callee method and P_A is added to the current permissions when returned from the callee method. A check node tests whether the current permissions include a specified subset of permissions, and if not, the execution is aborted. For simplicity, we do not include an exception handling mechanism in our HBAC model although it is not difficult to incorporate a throw-catch-style exception handling into the model and extend the model checking algorithm presented in Section 4 as was done in our previous work [27].

Formally, an HBAC program is a directed graph given by a 7-tuple $\pi = (NO, TG, CG, IS, IT, PRM, SP)$ where NO is a finite set of nodes, $TG \subseteq NO \times NO$ is a set of *transfer edges*, $CG \subseteq NO \times NO$ is a set of *call edges*, $IS : NO \to \{call[P_G, P_A] \mid P_G, P_A \subseteq PRM\} \cup \{check[P] \mid P \subseteq PRM\} \cup \{return\}$ is the labeling function for nodes, $IT \in NO$ is the *initial node*, which represents the entry point of the entire program, PRM is a finite set of *permissions*, and $SP : NO \to 2^{PRM}$ is the assignment of permissions to nodes. Each node $n \in NO$ corresponds to a program point and NO is divided into three subsets by IS as follows:

- $IS(n) = call[P_G, P_A]$ where $P_G, P_A \subseteq PRM$. The node n is a *call node* that represents a method call. Parameters P_G and P_A are called *grant permissions* and *accept permissions*, respectively.

- $IS(n) = return$. The node n is a *return node* that represents the return from a callee method.

- $IS(n) = check[P]$ where $P \subseteq PRM$. The node n is a *check node* that

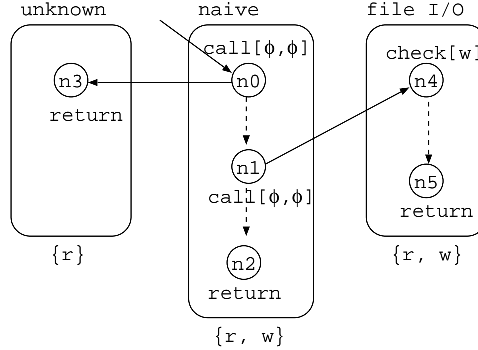


Figure 3.1. An HBAC program

represents a test for the current permissions. (The formal definition of the current permissions is given later.) If the current permissions include P as a subset, then the execution continues. Otherwise, the execution is aborted. For $p \in PRM$, $check[\{p\}]$ is abbreviated as $check[p]$. Similar abbreviation will be applied to $call[P_G, P_A]$.

A transfer edge (tg) represents a control flow within a method, and a call edge (cg) connects a method caller and a callee. In a figure, a solid arrow denotes a cg and a dotted arrow denotes a tg. A node that has an incoming edge without source node denotes the initial node.

Example 3.2.1 Fig 3.1 is an example of an HBAC program π_1 with the initial node n_0. There exists a tg from n_0 to n_1 (denoted as $n_0 \stackrel{TG}{\rightarrow} n_1$), which means the control can move to n_1 just after the execution of n_0. Likewise, there exists a cg from n_1 to n_4 (denoted as $n_1 \stackrel{CG}{\rightarrow} n_4$). This means that if the control reaches n_1, then the control is further passed to n_4 by a method call. If the control reaches n_5, the control returns to n_1. □

For a node n, $SP(n)$ specifies a subset of permissions that are assigned to n before runtime (*static permissions*). We assume that every node in the same method has the same static permissions, i.e.,

$$n \stackrel{TG}{\rightarrow} n' \Rightarrow SP(n) = SP(n').$$

Also, for every call node n such that $IS(n) = call[P_G, P_A]$, we require $P_G \subseteq SP(n)$ and $P_A \subseteq SP(n)$. In Fig 3.1, a method is represented by the set of nodes surrounded by a rectangle. A set beside the rectangle denotes the static permissions assigned to the nodes belonging to the method. For example, $SP(n_0) = SP(n_1) = SP(n_2) = \{r, w\}$ and $SP(n_3) = \{r\}$.

The description length of $\pi = (NO, TG, CG, IS, IT, PRM, SP)$ is defined as $\|\pi\| = |NO| \cdot |PRM| + |TG| + |CG|$. A state of π is a pair $\langle n, C\rangle$ of a node $n \in NO$ and a subset of permissions $C \subseteq PRM$. A *configuration* of π is a finite sequence of states, which is also called a *stack*. The concatenation of state sequences ξ_1 and ξ_2 is denoted as $\xi_1 : \xi_2$. The semantics of an HBAC program is defined by the transition relation $\rightarrow$ over the set of configurations, which is the least relation satisfying the following rules.

$$\frac{IS(n) = call[P_G, P_A], \;\; n \overset{CG}{\rightarrow} m}{\langle n, C\rangle : \xi \rightarrow \langle m, (C \cup P_G) \cap SP(m)\rangle : \langle n, C\rangle : \xi} \tag{3.1}$$

$$\frac{IS(m') = return, \;\; IS(n) = call[P_G, P_A], \;\; n \overset{TG}{\rightarrow} n'}{\langle m', C'\rangle : \langle n, C\rangle : \xi \rightarrow \langle n', C \cap (C' \cup P_A)\rangle : \xi} \tag{3.2}$$

$$\frac{IS(n) = check[P], \;\; P \subseteq C, \;\; n \overset{TG}{\rightarrow} n'}{\langle n, C\rangle : \xi \rightarrow \langle n', C\rangle : \xi} \tag{3.3}$$

For a configuration $\langle n_1, C_1\rangle : \ldots : \langle n_k, C_k\rangle$, the stack top is $\langle n_1, C_1\rangle$ where n_1 and C_1 are called the *current program point* and the *current permissions* of the configuration, respectively.

Rule (1) says that if the control is at a call node n where $IS(n) = call[P_G, P_A]$ and there exists a cg $n \overset{CG}{\rightarrow} m$, then $\langle m, (C \cup P_G) \cap SP(m)\rangle$ can be pushed onto the stack. That is, when control reaches the call node n, a method invocation can occur by passing the control to m and the current permissions become $(C \cup P_G) \cap SP(m)$. Rule (2) concerns with the return from the method. Assume that there exists a tg $n \overset{TG}{\rightarrow} n'$. If the current node is a return node m' in the callee method, then the next current node can be n'. The current permissions become $C \cap (C' \cup P_A)$. Note that if there is no cg from the call node n, then the control cannot proceed beyond n since rule (1) cannot be applied to n. Similarly, if there

Table 3.1. Modification of current permissions

	method call	return
(general case)	$(C \cup P_G) \cap SP(m)$	$C \cap (C' \cup P_A)$
$P_G = SP(n)$	$SP(n) \cap SP(m)$	$C \cap (C' \cup P_A)$
$P_G = \emptyset$	$C \cap SP(m)$	$C \cap (C' \cup P_A)$
$P_A = SP(n)$	$(C \cup P_G) \cap SP(m)$	C
$P_A = \emptyset$	$(C \cup P_G) \cap SP(m)$	$C \cap C'$
$P_A = \emptyset, P_G = \emptyset$	$C \cap SP(m)$	$C \cap C'(= C')$

is no tg from the call node n, then the control stops when it reaches a return node of the callee method. Although a program with such a node is pathological, for simplicity we do not make any syntactical restriction on cgs and tgs.

We can easily show that $C' \subseteq C \cup P_G$ whenever rule (2) can be applied to a configuration reachable from the initial configuration by induction on the rule application. The definition of the current permissions for some special cases helps us understand why they are defined as in rules (1) and (2).

- The case when $P_A = \emptyset$ and $P_G = \emptyset$ represents the basic design principle proposed in [1]: When a method is called, the current permissions C are intersected with the static permissions $SP(m)$ of the callee method; when the control returns to the caller method, the current permissions become $C \cap C'$, which is the intersection of the current permissions C' of the callee at the end of the method execution and the current permissions C of the caller when the method was invoked. Since $C' \subseteq C \cup P_G = C$ or equivalently $C \cap C' = C'$, the current permissions C' do not change at the return of the method call.

- In a general case, the parameter $P_G \subseteq SP(n)$ has the effect of adding the permissions in P_G to the current permissions before taking the intersection with $SP(m)$ at the method call. Especially, if $P_G = SP(n)$ then the current permissions simply become the intersection of the static permissions of the

caller and the callee, forgetting the execution history. In anyway, the effect of adding P_G to C is canceled since the current permission is intersected with C when returned.

- The effect of the execution of the callee method is partially canceled by adding the permissions in $P_A \subseteq SP(n)$ to C' before taking the intersection with C when returned. Especially, if $P_A = SP(n)$ then the current permissions become C when returned since $C' \subseteq C \subseteq SP(n)$. This means that the effect of the execution of the called method is totally canceled as in the case of stack inspection.
- A call node with $P_A = \emptyset$ corresponds to a grant statement in [1].
- A call node with $P_G = \emptyset$ corresponds to an accept statement in [1].

Finally, rule (3) says that if the control reaches a check node n with $IS(n) = \mathit{check}[P]$, there exists a tg $n \stackrel{TG}{\rightarrow} n'$ and the current permissions include P, then the control can be passed to n'.

The trace set of π is defined as

$$\begin{aligned} [\![\pi]\!] \;=\; & \{n_0 n_1 \ldots n_k \mid n_0 = IT,\ C_0 = SP(IT),\ \xi_0 = \varepsilon, \\ & \exists C_1, \ldots, C_k \subseteq PRM, \\ & \exists \xi_1, \ldots, \xi_k \in (NO \times 2^{PRM})^*, \\ & \langle n_i, C_i \rangle : \xi_i \rightarrow \langle n_{i+1}, C_{i+1} \rangle : \xi_{i+1} \text{ for } 0 \leq i < k\} \end{aligned}$$

where ε denotes the empty sequence.

For a set S of sequences, let prefix(S) denote the set of all nonempty prefixes of sequences in S.

Example 3.2.2 We return to HBAC program π_1 in Fig 3.1. When the method 'unknown' is called by n_0, the current permissions become $\{r, w\} \cap SP(n_3) = \{r, w\} \cap \{r\} = \{r\}$ since $IS(n_0) = \mathit{call}[\emptyset, \emptyset]$ (see Table 3.1). The test at node n_4 fails since $IS(n_4) = \mathit{check}[w]$ and the current permission $\{r\}$ does not include

$\{w\}$. Summarizing,

$$\begin{aligned}&\langle n_0, \{r, w\}\rangle \to \langle n_3, \{r\}\rangle : \langle n_0, \{r, w\}\rangle \to \langle n_1, \{r\}\rangle \\ &\to \langle n_4, \{r\}\rangle : \langle n_1, \{r\}\rangle \not\to \langle n_5, \{r\}\rangle : \langle n_1, \{r\}\rangle. \\ &[\![\pi_1]\!] = \{n_0, n_0n_3, n_0n_3n_1, n_0n_3n_1n_4\} \\ &\qquad = \mathit{prefix}(\{n_0n_3n_1n_4\}).\end{aligned}$$

Note that since there exists no node that has multiple outgoing tg or multiple outgoing cg (i.e., there is no nondeterminism) and there exists no cycle in π_1, the trace set can be represented as the prefixes of a single sequence $n_0n_3n_1n_4$.

Consider the situation that method 'naive,' calls 'unknown' and the latter method secretly change the content of a local variable of 'naive', say *fname*, to the name of a very critical file. Then, 'naive' requests 'file I/O' to delete *fname* without knowing the content of *fname* has been changed. If 'file I/O' performs $\mathit{check}[w]$ before deleting the file, the unintended file deletion can be avoided since the current permission does not include write permission $\{w\}$ as the effect of executing 'unknown'. As explained in the next section, however, this kind of access control cannot be realized by stack inspection.

Let π_2 be the HBAC program that is the same as π_1 except that $IS(n_0) = \mathit{call}[\emptyset, \{r, w\}]$. Since the accept permissions of n_0 are $\{r, w\}$,

$$\begin{aligned}\langle n_0, \{r, w\}\rangle &\to \langle n_3, \{r\}\rangle : \langle n_0, \{r, w\}\rangle \to \langle n_1, \{r, w\}\rangle \\ &\to \langle n_4, \{r, w\}\rangle : \langle n_1, \{r, w\}\rangle \to \langle n_5, \{r, w\}\rangle : \langle n_1, \{r, w\}\rangle.\end{aligned}$$

rm Similarly, let π_3 *be the HBAC program that is the same as* π_1 *except that* $IS(n_1) = \mathit{call}[\{r, w\}, \emptyset]$. *Since the grant permissions of* n_1 *are* $\{r, w\}$,

$$\begin{aligned}\langle n_0, \{r, w\}\rangle &\to \langle n_3, \{r\}\rangle : \langle n_0, \{r, w\}\rangle \to \langle n_1, \{r\}\rangle \\ &\to \langle n_4, \{r, w\}\rangle : \langle n_1, \{r\}\rangle \to \langle n_5, \{r, w\}\rangle : \langle n_1, \{r\}\rangle.\end{aligned}$$

□

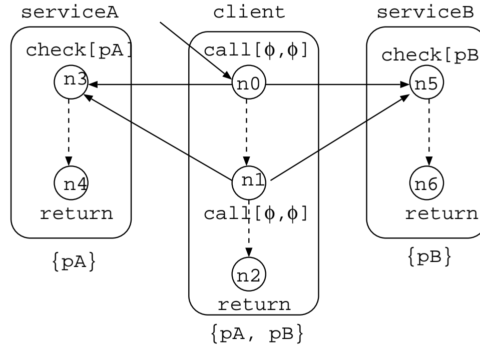


Figure 3.2. Chinese wall policy

Example 3.2.3 Chinese wall policy [6] is a policy such that a user has access permissions to any resources, but once the user has accessed one of the resources, (s)he loses access permissions to the resources belonging to the competing parties. A simplified Chinese wall policy can be represented by program π_4 in Fig 3.2. If n_0 calls 'serviceA,' the current permissions lose permission p_B. Thus, if n_1 calls 'serviceB' afterward, the check at n_5 fails. The same situation occurs when 'serviceB' and 'serviceA' are called in this order. In fact,

$$[\![\pi_4]\!] = \text{prefix}(n_0 n_3 n_4 n_1 (n_3 n_4 n_2 + n_5) + n_0 n_5 n_6 n_1 (n_5 n_6 n_2 + n_3)),$$

where the argument of 'prefix' is specified by a regular expression and + denotes the union operator. □

3.3. Comparison with Stack Inspection

A program with the Java stack inspection (abbreviated as SI program) can be represented by an 8-tuple $\pi = (NO, TG, CG, IS, IT, PRM, SP, PRV)$ where each component of π is the same as that of an HBAC program except that the label $IS(n)$ of each call node n is simply *call* without P_G and P_A, and a set of privileged nodes $PRV \subseteq NO$ is specified. The execution of a check node $check[P]$ succeeds if (a) for every node n on the stack, $P \subseteq SP(n)$, or (b) there exists a node $n_0 \in PRV$ on the stack such that $P \subseteq SP(n_0)$ and for every later node n in

the stack, $P \subseteq SP(n)$. By taking eager evaluation strategy, we can define the semantics of π by the following rules and rule (3) defined before (see [22, 30] for details).

$$\frac{IS(n) = call,\ n \stackrel{CG}{\to} m,\ n \notin PRV}{\langle n, C\rangle : \xi \to \langle m, C \cap SP(m)\rangle : \langle n, C\rangle : \xi} \tag{3.4}$$

$$\frac{IS(n) = call,\ n \stackrel{CG}{\to} m,\ n \in PRV}{\langle n, C\rangle : \xi \to \langle m, SP(n) \cap SP(m)\rangle : \langle n, C\rangle : \xi} \tag{3.5}$$

$$\frac{IS(m') = return,\ IS(n) = call,\ n \stackrel{TG}{\to} n'}{\langle m', C'\rangle : \langle n, C\rangle : \xi \to \langle n', C\rangle : \xi} \tag{3.6}$$

A program without check node is called a *basic program.* An HBAC (resp. SI) program π is an HBAC (resp. SI) *extension* of a basic program π_0 if π is obtained from π_0 by the following operations:

- Insert zero or more check nodes of HBAC (resp. SI) program into π_0;
- Add grant permissions and/or accept permissions to call nodes (in the case of HBAC extension); and
- Choose some of the nodes as privileged nodes (in the case of SI extension).

The formal definition of the extension is omitted. Let nc be a homomorphism over the set of nodes defined by $nc(n) = n$ for a call node and a return node n and $nc(n) = \varepsilon$ for a check node n. Let π_1 and π_2 be extensions of a basic program π_0. We say that π_1 and π_2 are *trace equivalent* if $nc([\![\pi_1]\!]) = nc([\![\pi_2]\!])$.

Comparing rules (4), (5), (6) with rules (1), (2), we can see that a non-privileged call node and a privileged node in an SI program can be simulated by $call[\emptyset, SP(n)]$ and $call[SP(n), SP(n)]$, respectively. This correspondence was informally described in [1]. However, the converse does not hold as shown in the next example.

Lemma 5 *Chinese wall policy in example 3.2.3 cannot be simulated by SI.*

(Proof sketch) Program π_4 in example 3.2.3 is an HBAC extension of basic program π_0 in Fig 3.3. Note that

$$nc([\![\pi_4]\!]) = \text{prefix}(n_0 n_4 n_1 n_4 n_2 + n_0 n_6 n_1 n_6 n_2).$$

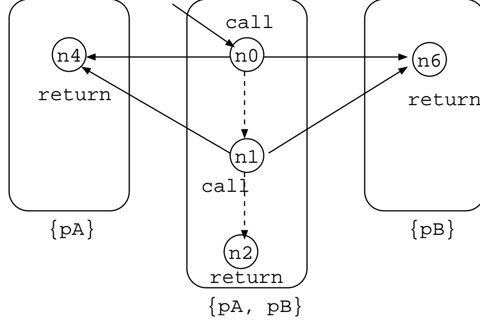


Figure 3.3. A basic program

Assume that there exists an SI extension π_{SI} of π_0 such that $nc([\![\pi_{\mathrm{SI}}]\!]) = nc([\![\pi_4]\!])$. Because the effect of executing 'serviceA' or 'serviceB' is canceled when the control reaches n_1 in π_{SI}, $nc([\![\pi_{\mathrm{SI}}]\!])$ must include a trace $n_0 n_4 n_1 n_6 n_2$, which is not in $nc([\![\pi_4]\!])$. □

Theorem 4 *For every basic program π_0 and every SI extension π of π_0, there exists an HBAC extension π' of π_0 that is trace equivalent to π. There exists a basic program π_0 and an HBAC extension π of π_0 such that there exists no SI extension π' of π_0 that is trace equivalent to π.* □

3.4. Model Checking HBAC Program

In this section, we discuss the verification problem (or model checking problem) defined as follows:

Inputs: An (HBAC) program $\pi = (NO, \ldots)$ and a verification property $\psi \subseteq NO^*$.

Output: Does every trace in $[\![\pi]\!]$ satisfy ψ? (i.e., $[\![\pi]\!] \subseteq \psi$?)

Example 3.4.1 Consider the verification problem for program π_4 of Example 3.2.3 and verification property $\psi = (\Sigma - \{n_4\})^* + (\Sigma - \{n_6\})^*$ where $\Sigma = (n_0 + n_1 + \cdots + n_6)$. As explained in Example 3.2.3, nodes n_4 and n_6 cannot be reached simultaneously in a single trace, and thus $[\![\pi_4]\!] \subseteq \psi$ holds. □

Let M be any representation of a language such as an automaton and a grammar. The description length of M is denoted by $\|M\|$ and the language expressed by M is denoted by $L(M)$.

Lemma 6 *For an arbitrary HBAC program π, we can construct a context-free grammar (cfg) G such that $L(G) = [\![\pi]\!]$ and $\|G\| = O(\|\pi\| \cdot c^{|PRM|})$ $(c > 1)$.*

(Proof sketch) We define the set of nonterminal symbols of G as $(NO \times 2^{PRM}) \cup (NO \times 2^{PRM} \times 2^{PRM})$. A nonterminal symbol $\langle n, C\rangle \in NO \times 2^{PRM}$ derives every trace starting from a node n with current permissions C. A nonterminal symbol $[n, C, C'] \in NO \times 2^{PRM} \times 2^{PRM}$ derives every trace starting from a node n with current permissions C and ending with a return node with current permissions C'. In the following, let C, C', and C'' be arbitrary subsets of PRM. For each node n, G has the rule $\langle n, C\rangle \to n$. For each pair (n, m) of nodes such that $IS(n) = call[P_G, P_A]$ and $n \overset{CG}{\to} m$, G has the rule $\langle n, C\rangle \to n\,\langle m, P_1\rangle$ where $P_1 = (C \cup P_G) \cap SP(m)$. Moreover, for each node n' such that $n \overset{TG}{\to} n'$, G has the following rules.

$$\langle n, C\rangle \;\to\; n[m, P_1, C']\langle n', P_2\rangle \tag{3.7}$$

$$[n, C, C''] \;\to\; n[m, P_1, C'][n', P_2, C''] \tag{3.8}$$

$$P_2 \;=\; C \cap (C' \cup P_A)$$

For each pair (n, n') of nodes such that $IS(n) = check[P]$ and $n \overset{TG}{\to} n'$, if $P \subseteq C$, then G has the following rule.

$$\langle n, C\rangle \;\to\; n\langle n', C\rangle$$

$$[n, C, C'] \;\to\; n[n', C, C']$$

For each return node n, G has the following rules.

$$[n, C, C] \;\to\; n$$

The start symbol of G is $\langle IT, SP(IT)\rangle$. □

Theorem 5 *Let π be an HBAC program and M be a finite automaton (fa). The verification problem for π and $\psi = \overline{L(M)}$ is solvable in deterministic $O(\|\pi\| \cdot c^{|PRM|} \cdot \|M\|^3)$ time ($c > 1$).*

(Proof sketch) By Lemma 6, we can construct a cfg G such that $L(G) = [\![\pi]\!]$. Thus, the verification problem is equivalent to deciding whether $L(G) \cap L(M) = \emptyset$. The latter condition can be checked in $O(\|G\| \cdot \|M\|^3)$ time. □

Corollary 6 *The verification problem for π and $\psi = \overline{L(M)}$ is solvable in deterministic $O(\|\pi\|^2 \cdot \|M\|^3)$ time if $|PRM| = O(\log \|\pi\|)$.* □

The assumption that $|PRM| = O(\log \|\pi\|)$ is realistic since the number of permissions is usually not so large compared with the program size.

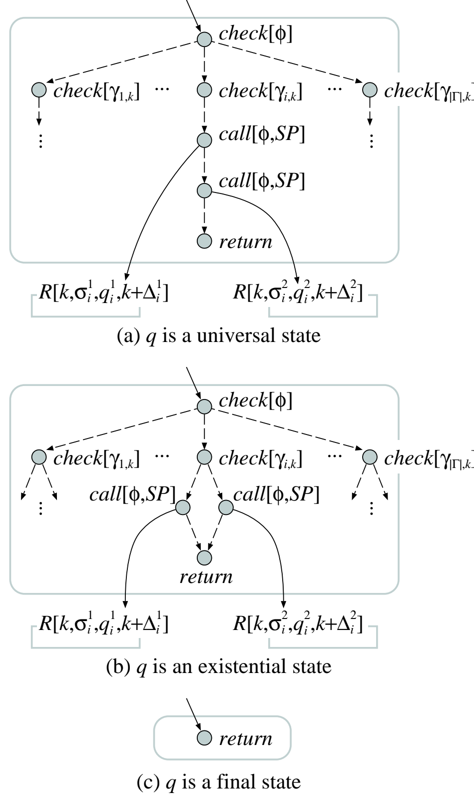
Let EXPTIME denote the class of decision problems solvable in deterministic $O(c^{p(n)})$ time for a constant c (> 1) and a polynomial p. The following theorem states that if the assumption that $|PRM| = O(\log \|\pi\|)$ does not hold, then the verification problem is EXPTIME-complete.

Theorem 7 *Let π be an HBAC program and M be an fa. The verification problem for π and $\psi = \overline{L(M)}$ is EXPTIME-complete.*

Proof. By Theorem 5, it suffices to show that the problem is EXPTIME-hard. It is known that a language L belongs to EXPTIME if and only if L is recognized by a polynomial space-bounded alternating Turing machine (ATM) [7]. For any given polynomial space-bounded ATM A and any input x of A, we can transform A and x into an HBAC program $\pi_{A,x}$ and a verification property ψ within polynomial time such that $[\![\pi_{A,x}]\!] \not\subseteq \psi \Leftrightarrow A$ accepts x.

Below we sketch the transformation. Assume that for any input x whose length equals n, A uses not more than $p(n)$ space for a polynomial p. Let $\Gamma = \{\gamma_1, \ldots, \gamma_{|\Gamma|}\}$ be the set of tape symbols of A and γ_1 be the blank symbol. Let δ be the transition function of A.

We define the set PRM of permissions in $\pi_{A,x}$ as $PRM = \{\gamma_{i,j} \mid 1 \leq i \leq |\Gamma|,\ 1 \leq j \leq p(n)\}$, and let current permissions C of each configuration of $\pi_{A,x}$



In this figure, we assume that $\delta(q,\gamma_i) = \{(q_i^1,\sigma_i^1,\Delta_i^1), (q_i^2,\sigma_i^2,\Delta_i^2)\}$.
SP denotes the static permissions of each call node.

Figure 3.4. Method $P[q, k]$

denote an instantaneous description (ID) (a string contained by the $p(n)$ tape squares) $\sigma_1\sigma_2 \ldots \sigma_{p(n)}$ of A, i.e., $\gamma_{i,j} \in C \Leftrightarrow \sigma_j = \gamma_i$. Program $\pi_{A,x}$ simulates a computation of A by altering current permissions according to the transition function δ of A.

Program $\pi_{A,x}$ consists of three types of method: $P[q, k]$, $R[k, \gamma_j, q', k']$, and I. $P[q, k]$ (in Fig 3.4) simulates a computation of A from any ID with state q and head position k. It first examines whether the current contents of the tape square k equals γ_i by $check[\gamma_{i,k}]$, and then calls $R[k, \gamma_j, q', k']$ for each $(q', \gamma_j, \Delta) \in \delta(q, \gamma_i)$ and $k' = k + \Delta$. $P[q, k]$ calls all these $R[k, \gamma_j, q', k']$ sequentially if q is a universal state of A. If q is an existential state of A, $P[q, k]$

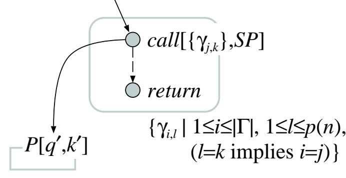


Figure 3.5. Method $R[k, \gamma_j, q', k']$

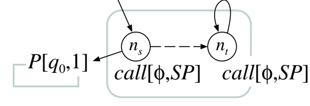


Figure 3.6. Method I

calls any one of these $R[k, \gamma_j, q', k']$ and returns. Otherwise, i.e., if q is a final state of A, $P[q, k]$ calls no $R[k, \gamma_j, q', k']$ and simply returns. Every node m of $P[q, k]$ has all the permissions as the static permissions, i.e., $SP(m) = PRM$. Every call node m_c in $P[q, k]$ is labeled as $IS(m_c) = call[\emptyset, SP(m_c)]$.

$R[k, \gamma_j, q', k']$ (in Fig 3.5) replaces the contents of tape square k with γ_j and calls $P[q', k']$ for simulating further moves. Every node m of $R[k, \gamma_j, q', k']$ has the static permissions $SP(m) = \{\gamma_{i,l} \mid 1 \le i \le |\Gamma|,\ 1 \le l \le p(n),\ (l = k \text{ implies } i = j)\}$ and the call node n_c in $R[k, \gamma_j, q', k']$ that calls $P[q', k']$ is labeled as $IS(n_c) = call[\{\gamma_{j,k}\}, SP(n_c)]$. Thus, when $R[k, \gamma_j, q', k']$ is called, every $\gamma_{i,k}$ for $1 \le i \le |\Gamma|$ and $i \ne j$ is removed from the current permissions, and when $R[k, \gamma_j, q', k']$ calls $P[q', k']$, $\gamma_{j,k}$ is added into the current permissions.

By the above construction, $P[q, k]$ returns if and only if the configuration of A that consists of q, k and the contents of the tape (ID) represented by the current permissions is a yes-configuration.

Method I (in Fig 3.6) simulates the initial configuration of A. It consists of two call nodes n_s and n_t and n_s is the initial node of $\pi_{A,x}$. Let the input $x = \gamma_{x_1}\gamma_{x_2}\ldots\gamma_{x_n}$ and q_0 be the initial state of A. The nodes in I have the static permissions $SP(n_s) = SP(n_t) = \{\gamma_{x_1,1}, \gamma_{x_2,2}, \ldots, \gamma_{x_n,n}, \gamma_{1,n+1}, \gamma_{1,n+2}, \ldots, \gamma_{1,p(n)}\}$. (Note that γ_1 is the blank symbol.) The initial node n_s calls $P[q_0, 1]$ with the current permissions $SP(n_s)$, and an execution of $\pi_{A,x}$ reaches the node n_t if and only if $P[q_0, 1]$ returns, i.e., A accepts x. We can simply let $\psi = (NO - \{n_t\})^*$, and thus $L(M) = NO^* n_t NO^*$.

□

3.5. Optimization of Model Checking Algorithm

From the proof of Theorem 2, we obtain the following algorithm for solving the verification problem.

Algorithm 1. For a given HBAC program π and an fa M such that $\psi = \overline{L(M)}$, perform the following three steps in this order.

1. Construct a cfg G such that $L(G) = [\![\pi]\!]$ based on the proof of Lemma 1.
2. Construct a cfg $\widehat{G}$ such that $L(\widehat{G}) = L(G) \cap L(M)$.
3. Decide whether $L(\widehat{G}) = \emptyset$.

The size of the G constructed in Step 1 is exponential to $|PRM|$. In most cases, however, G contains useless rules. In this section, we describe techniques for avoiding the construction of useless rules so that we can greatly reduce verification time and space.

3.5.1 Basic Idea

The following is traditional algorithm for eliminating useless rules in a cfg [20]. Nonterminal symbol X is *generating* if there exists a derivation from X to some string of terminal symbols. X is *reachable* if a derivation exists from the start symbol of G to $\alpha X \beta$ for some α and β. A rule r is *useless* if r contains a symbol that is not generating or not reachable. The traditional algorithm finds set V of all the symbols that are generating and reachable and then removes all rules involving one or more symbols not in V. While this algorithm *eliminates* useless rules of a given cfg, we want to *avoid* constructing such rules in the cfg construction. From the definition of G in the proof of Lemma 1, we can show the following lemma:

Lemma 7 *Let π be an HBAC program and G be the cfg constructed for π in Step 1 of Algorithm 1. For each $n \in NO$ and $C, C' \subseteq PRM$, $\langle n, C \rangle$ and $[n, C, C']$ are not reachable if $C \not\subseteq SP(n)$, and $[n, C, C']$ is not generating if $C' \not\subseteq C$.*

By this Lemma, we can avoid constructing rules involving $\langle n, C\rangle$ or $[n, C, C']$ such that $C \not\subseteq SP(n)$ or $C' \not\subseteq C$. However, the number of remaining rules is still exponential to $|PRM|$ in most cases, and thus we need further optimization.

3.5.2 Optimization 1: Rules with Reachable Symbols

We can exactly construct the rules involving only reachable symbols through the following breadth-first search algorithm: Construct every rule whose left-hand side is the start symbol $\langle IT, SP(IT)\rangle$. Then construct every rule whose left-hand side has appeared in the right-hand side of one of the constructed rules. Repeat this step until the run out of newly discovered nonterminal symbols.

The algorithm always halts since the number of nonterminal symbols is finite. Obviously, this algorithm constructs rule r of G if and only if r only contains reachable symbols. If the following conditions hold for some constant c, then the number of rules constructed through this algorithm is polynomial to $\|\pi\|$.

1. $|SP(n) \cap SP(m)| < c$ for each n in the main method (i.e., the method to which IT belongs) and each m such that $n \overset{CG}{\rightarrow} m$.

2. $|P_G(n)| < c$ for each call node n, where $P_G(n)$ is the set of the grant permissions of n.

The HBAC program of the Chinese wall policy in Example 3 satisfies these conditions.

3.5.3 Optimization 2: Precomputing Current Permissions

Every symbol of the form $\langle n, C\rangle$ is generating since the rule $\langle n, C\rangle \rightarrow n$ exists; however, some of the rules constructed through the algorithm in 3.5.2 may contain reachable but nongenerating symbols of the form $[n, C, C']$. For example, if the algorithm finds that $\langle n, C\rangle$ is reachable for some n with $IS(n) = call[P_G, P_A]$ and some C, then it constructs $\langle n, C\rangle \rightarrow n\,[m, P_1, C']\langle n', P_2\rangle$, where $P_1 = (C \cup P_G) \cap SP(m)$ and $P_2 = C \cap (C' \cup P_A)$, for every m and n' such that $n \overset{CG}{\rightarrow} m$ and $n \overset{TG}{\rightarrow} n'$ and

every $C' \subseteq P_1$. However, if $IS(m) = \mathit{return}$, then only $[m, P_1, P_1]$ is generating among the $2^{|P_1|}$ symbols of the form $[m, P_1, C']$. Hence we want to compute the set $X_{m,P_1} = \{C' \subseteq PRM \mid [m, P_1, C'] \text{ is generating}\}$ for given m and P_1 and to construct $\langle n, C\rangle \to n\,[m, P_1, C']\langle n', P_2\rangle$ only for each $C' \in X_{m,P_1}$. From the definition of the cfg G, $X_{n,C}$ is the least solution of the following equation:

$$X_{n,C} = \begin{cases} \{C\} & \text{if } IS(n) = \mathit{return}, \\ \emptyset & \text{if } IS(n) = \mathit{check}[P] \text{ and } P \not\subseteq C, \\ \bigcup_{n' \in TG(n)} X_{n',C} & \\ & \text{if } IS(n) = \mathit{check}[P] \text{ and } P \subseteq C, \\ \bigcup_{m \in CG(n)} \bigcup_{n' \in TG(n)} & \\ \quad \bigcup_{C' \in X_{m,(C \cup P_G) \cap SP(m)}} X_{n', C \cap (C' \cup P_A)} & \\ & \text{if } IS(n) = \mathit{call}[P_G, P_A], \end{cases}$$

where $TG(n) = \{n' \mid n \overset{TG}{\to} n'\}$ and $CG(n) = \{n' \mid n \overset{CG}{\to} n'\}$.

If a given HBAC program π is acyclic (as a directed graph with set of edges $CG \cup TG$), then we can compute $X_{n,C}$ for given n and C by regarding the above equation as a recursive definition of a function of n and C; i.e., we define and use a procedure for computing $X_{n,C}$ that answers the value of the right-hand side of the equation computed through recursive calls. However, if π has a cycle, then the recursion may not terminate. To avoid this problem, we modify this procedure as follows: When it is invoked to compute $X_{n,C}$ for some n and C during the computation of $X_{n,C}$ itself, then it temporarily assumes $X_{n,C} = 2^C$, which is the most conservative answer, and continues the computation. This modified procedure thus answers an over-estimation of $X_{n,C}$, and we write this estimated value as $X^*_{n,C}$.

In our implementation, the procedure computing $X^*_{n,C}$ also constructs rules consisting of symbols that are reachable from $[n, C, C']$ for some $C' \in X^*_{n,C}$ and are generating (or correctly, are the form $[m, P, P']$ such that $P' \in X^*_{m,P}$). For some n with $IS(n) = \mathit{call}[P_G, P_A]$ and some C, for instance, the procedure constructs $[n, C, C''] \to n\,[m, P_1, C'][n', P_2, C'']$, where $P_1 = (C \cup P_G) \cap SP(m)$ and $P_2 = C \cap (C' \cup P_A)$, for each m, n', C' and C'' such that $n \overset{CG}{\to} m$, $n \overset{TG}{\to} n'$, $C' \in X^*_{m,P_1}$ and

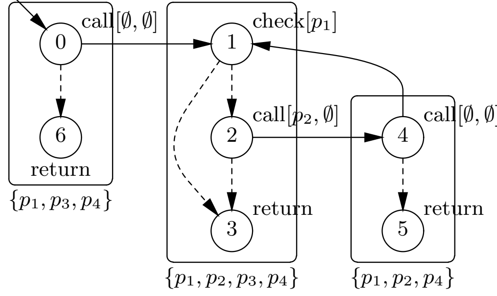


Figure 3.7. Sample HBAC program.

$C'' \in X^*_{n',P_2}$. The other rules reachable from $[n, C, C'']$ are constructed during the computation of X^*_{m,P_1} and X^*_{n',P_2}.

3.5.4 Optimization 3: Exact Computation of Current Permissions

The exact value of $X_{n,C}$ can be computed through the following iterative procedure: Consider each $X_{n,C}$ to be a variable, and let $V = \{X_{n,C} \mid n \in NO$ and $C \subseteq PRM\}$. Initialize every variable in V to $\emptyset$. Compute the value of the right-hand side of the equation in 3.5.3 for each n and C using the current values of the variables in V, and then assign it to $X_{n,C}$. Repeat this step until the values of the variables do not change any more. Since the domain of $X_{n,C}$ is finite and the right-hand side of the equation is monotonic, i.e., it only consists of the union operation, this procedure obtains its least fixpoint, which equals the least solution of the equation. However, managing all the variables in V is too expensive, and hence we want to avoid the computation for unnecessary variables. Moreover, when the value of some $X_{n,C}$ changes, we want to efficiently find $X_{n',C'}$ whose value should change according to the change of $X_{n,C}$.

To satisfy the above requirements, we propose the following algorithm:

1. Construct a directed graph that represents the dependency among a subset of variables in V, through depth-first search starting from a variable that we want to compute. The graph resembles the inputted HBAC program except

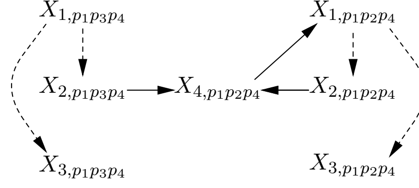


Figure 3.8. Initial dependency graph for $X_{1,\{p_1,p_3,p_4\}}$.

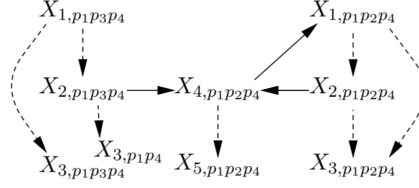


Figure 3.9. Final dependency graph for $X_{1,\{p_1,p_3,p_4\}}$.

that each node is augmented by current permissions. In this step, we ignore tgs from call nodes because we do not know the current permissions at the return from a callee method. Consider the HBAC program in Fig 3.7 for example. To construct rules reachable from the start symbol $\langle 0, \{p_1p_3, p_4\}\rangle$, we have to obtain the value of $X_{1,\{p_1,p_3,p_4\}}$, and we therefore construct the graph shown in Fig 3.8 through depth-first search starting from $X_{1,\{p_1,p_3,p_4\}}$, ignoring tgs from call nodes. The initial value of each $X_{n,C}$ is $\{C\}$ if $IS(n) =$ *return* and is $\emptyset$ otherwise.

2. Propagate the values of return nodes to other nodes as follows:

   2-1) Let L be a list of edges and initialize it to the list consisting of every edge entering a return node.

   2-2) Extract one edge e from L. If e is a tg from $X_{n,C}$ to $X_{n',C'}$, then add the value of $X_{n',C'}$ to $X_{n,C}$. Moreover, if the value of $X_{n,C}$ changes as a result, then add every edge entering $X_{n,C}$ to L. If e is a cg from $X_{n,C}$ to $X_{m,C''}$, then expand the graph by adding a tg from $X_{n,C}$ to $X_{n',C\cap(C'\cup P_A)}$, where P_A is the accept permissions of n, for each n'

such that $n \xrightarrow{TG} n'$ and each $C' \in X_{m,C''}$. Moreover, starting from $X_{n',C\cap(C'\cup P_A)}$, expand the graph in the same way as Step 1 if $X_{n',C\cap(C'\cup P_A)}$ has never been in the graph. Whenever a new edge entering either an already existing node or a return node is added to the graph, add the edge to L.

2-3) Repeat Step 2-2 until L becomes empty.

For the above example, we finally obtain the graph in Fig 3.9 and the result that $X_{1,\{p_1,p_3,p_4\}} = \{\{p_1,p_4\},\{p_1,p_3,p_4\}\}$ through this algorithm, while the algorithm in 3.5.3 answers $X^*_{1,\{p_1,p_3,p_4\}} = 2^{\{p_1,p_4\}} \cup \{\{p_1,p_3,p_4\}\}$ since $X^*_{4,\{p_1,p_2,p_4\}} = 2^{\{p_1,p_2,p_4\}}$.

3.6. Experiments

To examine the efficiency of the optimization described in the previous section to practical HBAC programs, we implemented a verification tool and measured verification time on the following two examples.

- Chinese wall policy
  We extend the program π_4 in Example 3 to a program $\pi_c(k)$ with $k+1$ methods $\{client, service_1, \ldots, service_k\}$ by replacing serviceA and serviceB with k copies of serviceA. The set of static permissions of *client* is $\{p_1, p_2, \ldots, p_k\}$ and the one of $service_i$ $(1 \le i \le k)$ is $\{p_i\}$. We specify a verification property ψ for $\pi_c(k)$ as

  $$(N_1 \cup N_c)^* + (N_2 \cup N_c)^* + \cdots + (N_k \cup N_c)^*$$

  where N_c is the set of the nodes of method *client* and N_i is the one of $service_i$. An HBAC program π satisfies ψ if and only if there is no trace of π containing nodes of two or more distinct service methods.

- On-line banking system
  As mentioned in Section 3, we can convert every SI program into an equivalent HBAC program. We define $\pi_o(k)$ as an HBAC program that is obtained

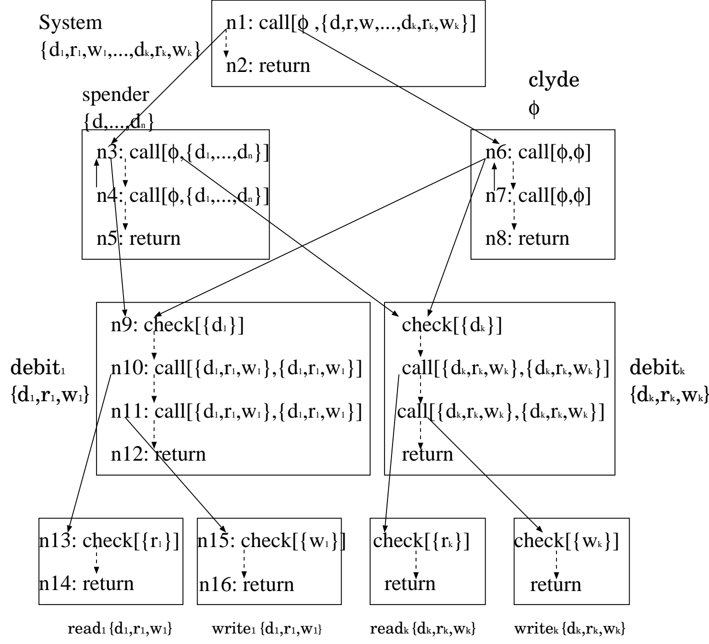


Figure 3.10. On-line banking system

from a sample SI program in [30], which models a part of an integrated on-line banking system with k banks (Fig 3.10). Each bank serves its clients with a method for withdrawing money. Method *spender* is an agent of a reliable user that has static permissions $\{d_1, \ldots, d_k\}$, and *clyde* is an agent of an unreliable user without any permission. They can access method $debit_i$ $(1 \leq i \leq k)$ that is a service provider of the i-th bank. Each $debit_i$ checks whether a user has permission d_i, and performs privileged calls on $read_i$ and $write_i$. A verification property ψ is given as the same as in [30]. That is, $\psi = \overline{N_{clyde}}^* N_{clyde} \overline{N_{rw}}^*$, where N_{clyde} is the set of nodes in method *clyde* and N_{rw} is the union of the sets of nodes of every $read_i$ and every $write_i$. An HBAC program π satisfies ψ if and only if the control never reaches $read_i$ or $write_i$ after it once reaches a node in *clyde*.

Table 3.2 summarizes the results of the experiments. G is the cfg generated in Step 1 of Algorithm 1. M is a regular grammar such that $\psi = \overline{L(M)}$. Fig 3.11

Table 3.2. Verification profiles of sample programs

		$\pi_c(k)$						$\pi_o(k)$			
k		5	10	20	40	60	80	5	10	15	20
the number of permissions		5	10	20	40	60	80	15	30	45	60
the number of the rules of G	base †	1613									
	1‡	128	353	1103	3803	8103		1870			
	1+2§	81	211	621	2041	4261	7281	184	1316	33200	
	1+2+3 ¶	81	211	621	2041	4261	7281	142	277	412	547
$\|M\|$		124	389	1369	5129	11289	19849	212	392	572	752
computation time $^\|$ (sec)	base	0.047									
	1	0.000	0.016	0.187	2.83	16.9		0.031			
	1+2	0.000	0.005	0.136	2.72	19.3	63.2	0.005	0.021	1.365	
	1+2+3	0.005	0.016	0.146	2.72	16.5	64.1	0.000	0.000	0.011	0.010
verification result		true	true	true	true	true	true	true	true	true	true

†base is the algorithm 1 modified based on lemma 2 (section 5.1).
‡optimization 1 described in section 5.2.
§optimization 1 and 2 described in section 5.3.
¶optimization 1,2 and 3 described in section 5.4.
$^\|$Java VM build 1.5_06,von Windows XP Pentium4, 2GHz, 1GB RAM.

shows the computation time needed to verify $\pi_c(k)$ and $\pi_o(k)$. Without the optimization, we could not verify $\pi_o(k)$ for any $k \geq 1$ and $\pi_c(k)$ for $k \geq 10$, because the number of the rules of G was exponential to k and the amount of memory was not sufficient to store G. With the full optimization, both the number of the rules of G and the computation time were reduced to polynomial to k for $\pi_c(k)$ and $\pi_o(k)$. Especially, the computation time for $\pi_o(k)$ was linear to k. As in [30], we estimate that the number of permissions used in an ordinary network application is at most several tens, and the results suggest that the proposed verification method is feasible for practical programs.

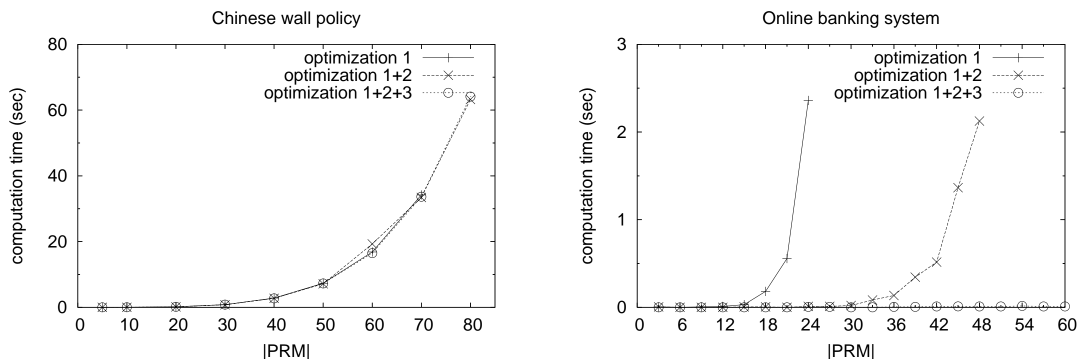


Figure 3.11. Verification time for $\pi_c(k)$ and $\pi_o(k)$

3.7. Conclusions of chapter 3

In this chapter, we presented a new model for dynamic access control based on execution history, called HBAC programs. The expressive power of HBAC programs was examined and the verification problem for HBAC programs was shown to be solvable. Although the complexity of the problem is EXPTIME-complete in general, our verification tool could verify sample programs within a reasonable time.

Our program model has close relation to a class of infinite state systems called *pushdown systems* (abbreviated as PDS). Indeed, behavior of an HBAC program can be modeled by a PDS with an exponential number of stack symbols. Decidability and complexity of LTL and CTL* model checking [8] for PDS are extensively studied in [12, 13]. Verification results conducted on a model checker for PDS is reported in [14]. Although the verification problem and the model checking algorithm in this chapter are based on finite traces, we can extend the algorithm to infinite traces (and thus LTL) using ω-context-free grammars [9], and the time complexity of the algorithm is slightly better than the one needed when we apply the algorithm in [12] to a PDS that models a given HBAC program. Namely, the former is proportional to $|Q_M|^3$ where $|Q_M|$ is the number of the states of a Büchi automaton M representing the negation of a verification property, while the latter is proportional to $|Q_M|^2|\Delta_M|$ where $|\Delta_M|$ is the number of the transitions of M. Note that the optimization described in section 5 can be

applied not only when using our algorithm but also when we use a model checker for PDS to verify an HBAC program.

Chapter 4

Conclusion

In this thesis, we formally defined two problems of security system, one is IDS partition deployment problem, the other is security verification problem of HBAC, and describe the algorithms to solve them. then analyse the computational complexity of these algorithms.

In chapter 2, We have defined a generalized IDS partition deployment problem which computes the deployment of IDSs, the set of messages which should been monitored by each IDS and a partition of an attack scenario. Furthermore, we simplified this problem and designed an efficient algorithm which computes the optimal solution of the problem. The complexities of the related problems are summarized in table 2.1. For any state transition IDS, the IDS partition deployment problem (simplified) can be solvable in P, and the probe number minimization IDS partition deployment problem is NP-complete, while nontrivial upperbound and lowerbound of the complexity of the generalized IDS-PDP are unknown at present. As future works, we will design an algorithm which calculates the optimal solution, or analyze the lowerbound of the complexity of the generalized IDS partition deployment problem. Moreover, we will consider applying IDS partition deployment problems to a larger class of IDS(e.g., [26], [19]).

In chapter 3, we presented a new model for dynamic access control based on execution history, called HBAC programs. The expressive power of HBAC

programs was examined and the verification problem for HBAC programs was shown to be solvable. Although the complexity of the problem is EXPTIME-complete in general, our verification tool could verify sample programs within a reasonable time.

Our program model has close relation to a class of infinite state systems called *pushdown systems* (abbreviated as PDS). Indeed, behavior of an HBAC program can be modeled by a PDS with an exponential number of stack symbols. Decidability and complexity of LTL and CTL* model checking [8] for PDS are extensively studied in [12, 13]. Verification results conducted on a model checker for PDS is reported in [14]. Although the verification problem and the model checking algorithm in this paper are based on finite traces, we can extend the algorithm to infinite traces (and thus LTL) using ω-context-free grammars [9], and the time complexity of the algorithm is slightly better than the one needed when we apply the algorithm in [12] to a PDS that models a given HBAC program. Namely, the former is proportional to $|Q_M|^3$ where $|Q_M|$ is the number of the states of a Büchi automaton M representing the negation of a verification property, while the latter is proportional to $|Q_M|^2|\Delta_M|$ where $|\Delta_M|$ is the number of the transitions of M. Note that the optimization described in section 5 can be applied not only when using our algorithm but also when we use a model checker for PDS to verify an HBAC program. Comparing the expressive power of various subclasses of security automata [15, 36] with that of HBAC programs is interesting future work. At present we have the following conjecture.

Conjecture 8 *The expressive power of HBAC programs is strictly greater than finite state security automata.*

A fomal proof of Conjecture 8 is future work.

References

[1] M. Abadi and C. Fournet, "Access control based on execution history," Network & Distributed System Security Symposium., pp.107–121, 2003.

[2] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel and E. Stoner, "State of the practice of intrusion detection technologies," Technical Report, Carnige Mellon University, CMU/SEI-99-TR-028, ESC-99-028, 2000.

[3] A. Banerjee and D. A. Naumann, "History-based access control and secure information flow," CASSIS04, LNCS 3362, pp.27–48, 2004.

[4] M. Bartoletti, P. Degano and G. L. Ferrari, "History-based access control with local policies," 8th FOSSACS, LNCS 3441, pp.316–332, 2005.

[5] M. Bartoletti, P. Degano and G. L. Ferrari, "Enforcing secure service composition," IEEE 18th CSFW, pp.211-223, 2005.

[6] D. F. C. Brewer and M. J. Nash, "The Chinese wall security policy," IEEE Security & Privacy, pp.206–214, 1989.

[7] A. K. Chandra, D. C. Kozen and L. J. Stockmeyer, "Alternation," Journal of the ACM, 28, pp.114–133, 1981.

[8] E. M. Clarke, Jr., O. Grumberg and D. Peled, *Model Checking,* MIT Press, 2000.

[9] R. S. Cohen and A. Y. Gold, "Theory of ω-languages. I: Characterizations of ω-context-free languages," Journal. of Computer & System Science, 15, pp.169–184, 1977.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, MIT Press, 2003.

[11] Ú. Erlingsson and F. B. Schneider, "IRM Enforcement of Java Stack Inspection," IEEE Security & Privacy, pp.246–255, 2000.

[12] J. Esparza, D. Hansel, P. Rossmanith and S. Schwoon, "Efficient algorithms for model-checking pushdown systems," CAV2000, LNCS 1855, pp.232–247, 2000.

[13] J. Esparza, A. Kučera and S. Schwoon, "Model-checking LTL with regular variations for pushdown systems," TACS01, LNCS 2215, pp.316–339, 2001.

[14] J. Esparza and S. Schwoon, "A BDD-based model checker for recursive programs", CAV2001, LNCS 2102, pp.324–336, 2001.

[15] P. W. Fong, "Access control by tracking shallow execution history," IEEE Security & Privacy, pp.43–55, 2004.

[16] M. R. Garey and D. S. Johnson, *Computers and intractability: A Guide to the Theory of NP-Completeness*, Springer, 1979.

[17] L. Gong, M. Mueller, H. Prafullchandra and R. Schemers, "Going beyond the sandbox: An overview of the new security architecture in the Java™ development kit 1.2," USENIX Symp. on Internet Technologies and Systems, pp.103–112, 1997.

[18] K. W. Hamlen, G. Morrisett and F. B. Schneider, "Certified In-lined reference monitoring on .NET," Cornell University Computing and Information Science Technical Report, TR2005-2003, 2005.

[19] Y. Ho, D. Frinck, D. Tobin and Jr, Planning, "Petri nets and intrusion detection," 1998.
http//csrc.nist.gov/nissc/1998/proceedings/paperF5.pdf.

[20] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 2001.

[21] K. Ilgun, R. A. Kemmerer and P. A. Porras, "State transition analysis: a rule-based intrusion detection system," IEEE Transactions on Software Engineering, 21(3), pp.181–199, 1995.

[22] T. Jensen, D. Le Métayer and T. Thorn, "Verification of control flow based security properties," IEEE Security & Privacy, pp.89–103, 1999.

[23] S. Jha, O. Sheyner and J. Wing, "Two formal analyses of attack graphs," IEEE Symposium on Security and Privacy, pp.49–63, 2002.

[24] S. Kiyamura, Y. Takata and H. Seki, "A method of decomposing a labeled transition system into parallel processes," IPSJ Journal, 42(12), pp.2992–3003, 2001 (in Japanese).

[25] S. Kiyamura, Y. Takata and H. Seki, "Process decomposition via synchronization events and its application to counter-process decompostion," Proc. of the 5th International Conference on Parallel Processing and Applied Mathematics (PPAM 2003), LNCS 3019, pp.298–305.

[26] S. Kumar and E. H. Spafford, "An application of pattern matching in intrusion detection," Technical Report CSD-TR-94-013, Department of Computer Science, 1994.

[27] S. Kuninobu, Y. Takata, D. Taguchi, M. Nakae and H. Seki, "A specification language for distributed policy control," 4th ICICS, LNCS 2513, pp.386-398, 2002.

[28] R. Milner, *Communication and Concurrency*, Prentice Hall International Series in Computer Science, 1989.

[29] B. Mukherjee, L. T. Heberlein and K. N. Levitt, "Network intrusion detection," IEEE Network, pp.26–41, 1994.

[30] N. Nitta, Y. Takata and H. Seki, "An efficient security verification method for programs with stack inspection," 8th ACM Computer & Communications Security, pp.68–77, 2001.

[31] J. Pieprzyk, T. Hardjono and J. Seberry, *Fundamentals of Computer Security*, pp.459–497, Springer, 2003.

[32] P. A. Porras and P. G. Neumann, "EMERALD: event monitoring enabling responses to anomalous live disturbances," In 1997 National Information Systems Security Conference, 1997.

[33] R. W. Ritchey and P. Ammann, "Using Model checking to analyze network vulnerabilities," IEEE Symposium on Security and Privacy, pp.156–165, 2000.

[34] H. A. B. Saip and C. L. Lucchesi, "Matching algorithms for bipartite graphs," Relatorio Tecnico DCC-03, 1993.
http:citeseer.nj.nec.com/baiersaip93matching.html

[35] A. Schaad, J. Moffett and J. Jacob, "The role-based access control system of a European Bank: A case study and discussion," 6th ACM Symp. on Access Control Models and Technologies, pp.3–9, 2001.

[36] F. B. Schneider, "Enforceable security policies," ACM Transactions. on Information & System Security, 3(1), pp.30–50, 2000.

[37] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, pp.259–375, Spinger, 2003.

[38] G. Vigna and R. A. Kemmerer, "NetSTAT: a network-based intrusion detection system," Journal of Computer Security, 7(1), IOS Press, pp.37–71, 1999.

[39] D. Volpano and G. Smith, "A type-based approach to program security," TAPSOFT'97, LNCS 1214, pp.607–621, 1997.