4-20-2006

On Optimizing Compatible Security Policies in Wireless Networks

Scott C-H Huang
Computer Science Department, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong, huang@cs.umn.edu

Kia Makki
Telecommunications and Information Technology Institute, Florida International University, Miami, FL

Nikki Pissinou
Telecommunications and Information Technology Institute, Florida International University, Miami, FL

Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2006, Article ID 23728, Pages 1–7
DOI 10.1155/WCN/2006/23728

On Optimizing Compatible Security Policies in Wireless Networks

Scott C.-H. Huang,[1] Kia Makki,[2] and Niki Pissinou[2]

[1] *Computer Science Department, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong*
[2] *Telecommunications and Information Technology Institute, Florida International University, 10555 W Flagler Street, EC 2910, Miami, FL 33174, USA*

Received 29 September 2005; Revised 19 January 2006; Accepted 1 February 2006

This paper deals with finding the maximum number of security policies without conflicts. By doing so we can remove security loophole that causes security violation. We present the problem of maximum compatible security policy and its relationship to the problem of maximum acyclic subgraph, which is proved to be NP-hard. Then we present a polynomial-time approximation algorithm and show that our result has approximation ratio $1 + 1/k$ for any integer k with complexity $O(N^{k+1})$.

1. INTRODUCTION

Security is the basis of any system, and it can be described in general terms such as confidentiality, integrity, and availability, but what we do precisely mean by security for a particular system varies from system to system and possibly depends on the situation. The security policies in the army, in a financial institution, at a university, and in a big corporation are significantly different. Each of which has their own needs that should be reflected in the design of their security infrastructures. In the army, every document is classified into different confidential levels. Only authorized persons have the right to view a document. Also, certain sensitive tasks can only be executed by authorized personnel. In a financial institution, each customer has his/her data file, in which some are confidential and some are not. As a whole, there must be a system to deal with these requirements. Security policies should also be designed at different layers. This all depends on the actual data for which security mechanisms are designed. Confidentiality and authenticity of communications are usually classified at the application layer while authenticity of routing packets should be at the network layer. Even for confidentiality, there are end-to-end encryption and link-based encryption, which totally depend on system needs. Since the requirement and design techniques vary quite widely from layer to layer, we cannot regard them as a whole and not differentiate the needs for each layer when designing a system's security infrastructure.

A security policy is a description of the security goals for a system and how a system should behave in order to meet these goals. It may concern access control, information flow, and availability. Security policy bridges the gap between static implementations and the broad and diverse security requirements of user communities. Security policy becomes more complicated in heterogeneous environments. When two or more entities share a security association, they must reach agreement on a governing policy. An example policy is that the request for a purchase and its approval must be by different users. Since the security goals for a system affect how the security mechanisms on the system are configured, it is important for the security policy to be stated clearly. The term security policy may mean different things to different communities. For example, access control policy defines who has access to what and under what circumstances. Other forms of security policy specify under what conditions credentials are accepted, or how a firewall is configured. In its broadest definition, security policy is the specification of security-relevant system behavior.

In a large enterprise, it is often necessary to manage a large set of diverse objects across various organization boundaries. For example, access to the shared resources like printer, scanner, and so forth needs to be carefully managed. In such a scenario, conflicts often arise when large sets of objects across diverse boundaries are being managed. In general, finding whether there is a security policy conflict is an NP-complete problem. If there exists a conflict, then these

security policies must be reconciled by removing some part of the policies. Since the security goals for a system affect how the security mechanisms on the system are configured, it is important for the security policy to be stated clearly. Additional benefit is obtained if the policy can be directly used to configure the security mechanisms or to formally reason about the effect of the policy. A common security need is to restrict access to the resources on a system. This is reflected in a constraint security policy, which states constraints on the system.

In this paper, we do not deal with the problem of conflict resolution (or conflict reconciliation). Previous works about security policy conflict resolution (Dunlop et al. [1]) or logic regarding security policies (Abadi [2], Jajodia et al. [3]) all have the goal to determine a security conflict and resolve it. Our work is motivated by optimizing security policies without conflicts to achieve maximal benefit instead of trying to locate conflicts, on which none of the previous works have addressed. In the discussion about security policies, we particularly focus on the part of access control in networks. We seek to maximize the subset of compatible security policies in a system. If there exists some security loophole in a system, by using our algorithm we can find the maximum compatible policies and remove other ones that cause conflict. First we will present the problem statement and prove this problem is NP-hard. Then we will present approximation algorithm and show that our result has approximation ratio $1 + 1/k$ for any integer k with complexity $O(N^{k+1})$.

2. PROBLEM FORMULATION

We present the standard logic used by Abadi as a basis of our problem statement. Jajodia et al. also proposed similar results and this was, in fact, earlier than Abadi's. Martin Abadi viewed access control as a description of ternary relation *may-access*. In addition to the functions Abadi introduced, we also consider two more functions *may-not-access* and *forbids*. *May-not-access*, as its name suggests, represents that some principal does not have the right to do something. Namely, *may-not-access*(p, o, r) stands for "p" not having the right "r" on "o." The function *forbids* represents that a principal will forbid another one to do something. Here, this function is a bit different, as only certain authorized principals, called the *deauthorizer*, are allowed to use the *forbid* function. Also, u forbids v means that if u does not have access to something, it will result in v not having access to it either. u itself cannot make such kind of policy, and this kind of policy must be made by some deauthorizer other than both u and v. As for who the deauthorizers are, it totally depends on the security policy.

We basically use Abadi's terminology as a basis, and modify it to best suit our scenario. The building blocks of our notation are as follows.

(1) Access control verifier "*may-access*": *may-access*(p, o, r) represents that principal "p" has the right "r" on object "o." Here the right "r" can be thought of as the right "read" or "write" on a file "toad.txt." The example *may-access(Alice, toad.txt, read)* tells us that Alice has the "read" authorization on file "toad.txt."

(2) Authorization function "*says*": we also consider the transfer or recommendation of authorization. In other words, we allow certain users to authorize others if they have authorization on something themselves. This is represented by the function *says*. For example, p *says may-access*(q, o, r) basically means that "p" hands the right "r" over to "q" or "p" authorizes "q" to obtain the right "r."

(3) Negated access control verifier "*may-not-access*": this is actually the negated access control verifier *may-access*(p, o, r). In other words,

$$\textit{may-not-access}(p, o, r) = \neg(\textit{may-access}(p, o, r)). \quad (1)$$

(4) Deauthorization function "*forbids*": different to *says*, u *forbids* v means that there is a chain of deassociation from u to v. If *may-access*(u, o, r) is false, then *may-access*(v, o, r) will be false as well. Note that such a policy cannot be made by either u or v. Instead, it can only be made by legal deauthorizers.

2.1. Logical deduction rules

With these two functions *may-access* and *says*, we can perform logical deductions as follows.

(1) First authorization rule:

$$\begin{aligned}&(\textit{may-access}(p, o, r))\\ &\quad \wedge (p\ \textit{says may-access}(q, o, r))\\ &\quad \Longrightarrow \textit{may-access}(q, o, r).\end{aligned} \quad (2)$$

Conceptually, this means if "p" has access to "o," then p can grant q the same access rights, too. In other words, the right "r" can be transferred by a user that has this right.

(2) Second authorization rule:

$$p\ \textit{says}(s1 \Longrightarrow s2) \Longrightarrow (p\ \textit{says}\ s1 \Longrightarrow p\ \textit{says}\ s2). \quad (3)$$

This means if something that has an implication such as having right a will result in having right b, this implied right will be passed on in the case of authorization, too. Take the file access rights, for example. If "p" says that a file allowed to be modified by a user will automatically have the right to be read by that user too, then if p authorizes some other user to have the right of modifying a file, it is implied that p also authorizes that user to read the file too. This rule is supplementary, but it allows us to do nested or compound authorization.

(3) Deauthorization rule:

$$\begin{aligned}&(p\ \textit{forbids may-access}(q, o, r))\\ &\quad \wedge (\textit{may-not-access}(p, o, r))\\ &\quad \Longrightarrow \textit{may-not-access}(q, o, r).\end{aligned} \quad (4)$$

Table 1: Security policy example

(1) *may-access*$(u, o) = true$
(2) *u says may-access*$(v, o) = true$
(3) *u says may-access*$(w, o) = true$
(4) *may-not-access*$(x, o) = true$ (*or may-access*$(x, o) = false$)
(5) *x forbids may-access*$(y, o) = true$

2.2. Security policy graph

Now we introduce the use of *security policy graph* to transform the problem of security policies into a problem of graph theory. To simplify the construction, we only consider the graph representing four functions: *may-access*, *says*, *may-not-access*, and *forbids*. Also, to further simplify our discussion, we do not differentiate different rights, as we can actually represent them as different objects (i.e., to differentiate the rights "read," "write" on a file, we can actually regard them as two separate objects "read-file" and "write-file"). The construction of the basic security policy graph $G(V, E)$ is as follows: V ="the set of all users, object o and $\neg o$," $e = (u, v) \in E$ if and only if one of the following is true:

(1) u is a user, v is an object, and *may-access*$(u, v) = true$;
(2) u is an object, v is a user, and *may-access*$(v, u) = false$;
(3) u, v are both users and v *says may-access*$(u, o) = true$;
(4) u, v are both users, and u *forbids may-access*$(v, o) = true$;
(5) u, v are both objects, and $v = \neg u$.

Consider the following example of security policy shown in Table 1, whose corresponding security policy graph will be Figure 1. There are two objects o and $\neg o$, and five users u, v, w, x, y. An arc is added from u to o because of policy 1 on the table and rule 1. Two arcs from v to u and from w to u are added according to policy 2, 3 and rule 3. The arc from $\neg o$ to x is added according to policy 4 and rule 2, while the arc from x to y is added according to policy 5 and rule 4. Finally, an arc is added from o to $\neg o$ according to rule 5.

2.3. Properties of security policy graphs

Lemma 1. *Let u be a user and o be an object. Then u has access to o if and only if there is a path from u to o.*

Proof. u has access to $o \Leftrightarrow$ *may-access*$(u, o) = true \Leftrightarrow \exists u_1, u_2, \ldots, u_k$ such that

$$\begin{aligned} &\textit{may-access}(u_1, o) \wedge (u_1 \textit{says may-access}(u_2, o)) \\ &\quad \wedge \cdots \wedge (u_k \textit{ says may-access}(u, o)) = \textit{true} \end{aligned} \tag{5}$$

(according to the first authorization rule) $\Leftrightarrow \exists u_1, u_2, \ldots, u_k$ such that $(u_1, o), (u, u_k) \in E$ and $(u_{i+1}, u_i) \in E\ \forall 1 \leq i < k \Leftrightarrow$ there is a path from u to o (on which $u_k, u_{k-1}, \ldots, u_1$ are intermediate nodes). □

Lemma 2. *Let u be a user and ¬o be the negation of an object. Then u has no access to o if and only if there is a path from ¬o to u.*

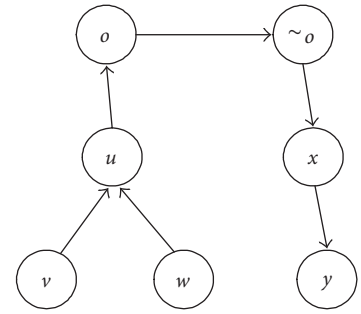

Figure 1: Security policy graph.

Proof. u has no access to $o \Leftrightarrow$ *may-not-access*$(u, o) = true \Leftrightarrow \exists u_1, u_2, \ldots, u_k$ such that

$$\begin{aligned} &\textit{may-not-access}(u_1, o) \\ &\quad \wedge (u_1 \textit{ forbids may-access}(u_2, o)) \\ &\quad \wedge \cdots \wedge (u_k \textit{ forbids may-access}(u, o)) = \textit{true} \end{aligned} \tag{6}$$

(according to the deauthorization rule) $\Leftrightarrow \exists u_1, u_2, \ldots, u_k$ such that $(\neg o, u_1), (u_k, u) \in E$ and $(u_i, u_{i+1}) \in E\ \forall 1 \leq i < k \Leftrightarrow$ there is a path from $\neg o$ to u (on which $u_1, u_2, \ldots, u_k$ are intermediate nodes). □

Definition 1. A set of security policies is said to have a security conflict if there exists at least one user u and an object o such that *may-access*(u, o) = *may-not-access* $(u, o) = true$.

Theorem 1. *There is a security conflict if and only if the corresponding security policy graph has a (directed) cycle that contains the edge from an object to its negation.*

Proof (forward direction). If there is a security conflict, then by definition there exists a user u such that *may-access*(u, o) = *may-not-access*$(u, o) = true$. Because *may-access*$(u, o) = true$, by Lemma 1, there is a path from u to o with intermediate nodes $u_j, u_{j-1}, \ldots, u_1$. On the other hand, because *may-not-access*$(u, o) = true$, there is a path from $\neg o$ to u with intermediate nodes $u'_1, u'_2, \ldots, u'_k$. $\{u, u_j, u_{j-1}, \ldots, u_1, o, \neg o, u'_1, u'_2, \ldots, u'_k\}$ thus forms a cycle that contains $(o, \neg o)$. Backward direction: if there is a cycle containing $(o, \neg o)$, pick a user v on the cycle. There must be a path from either v to o or from o to v because it is a cycle. However, there cannot be any outgoing edge from o to anything other than $\neg o$, so this path must be from v to o. By Lemma 1, *may-access*$(v, o) = true$. Similarly, there must be a path from $\neg o$ to v because $\neg o$ cannot have any incoming edge except from o. By Lemma 2, *may-not-access*$(v, o) = true$. □

3. THE MAXIMUM COMPATIBLE SECURITY POLICY PROBLEM

Our main motivation is to find the maximum subset of compatible security policies (i.e., in which there is no conflict).

Theorem 1 gives us a necessary and sufficient condition for whether or not a conflict exists in a set of security policies. In light of this theorem, finding the maximum subset of compatible security policies is equivalent to finding the maximum acyclic subgraph (with certain property) in its security policy graph. In general, the maximum acyclic subgraph problem is NP-hard, but the maximum compatible security policy problem is a special case of it. In this section, we introduce the *maximum acyclic subgraph* problem and show that it is NP-hard by reducing 3-SAT to it. First we define the maximum acyclic subgraph problem as follows.

Maximum acyclic subgraph problem

Given a directed graph $G = (V, E)$, find a subset $E' \subset E$ of maximum cardinality such that $G = (V, E')$ is acyclic.

3-SAT (maximization version)

Given a formula $F = \{C_1, C_2, \ldots, C_m\}$ of clauses on a finite set U of variables such that $|C_i| = 3$ for $1 \leq i \leq m$, find a subcollection S (of F) of maximum cardinality such that there is a truth assignment for S.

3-SAT $\leq_P$ max-acyclic-subgraph

(This reduction is based on Newman [4], though quite different from it.) Given a 3-SAT formula F with n variables and m clauses, we construct a corresponding multigraph G using the following rules.

(1) For each variable $x \in F$, we create 2 vertices x_1 and x_2. These two vertices will form the *variable gadget* for the variable x.
(2) For each clause $C_k \in F$, we create a directed 6-cycle and label each of 3 alternating edges with a distinct literal from the clause C_k. This will be the *clause gadget* for the clause C_k as shown in Figure 2. Each of the three literals corresponds to an arc in the 6-cycle, and the other 3 arcs are used to connect them.
(3) Each clause gadget is linked up to the variable gadgets as follows. (1) For an arc (i, j) corresponding to x (the positive form of a variable) in the clause gadget, we add a directed edge from vertex i to vertex x_1, an edge from x_2 to j, and a 2-cycle between x_1, x_2. (2) For an arc (i, j) corresponding to $\overline{x}$ (the negated form of a variable), we add a directed edge from vertex i to vertex x_2, an edge from x_1 to j, and a 2-cycle between x_1, x_2 (we allow multiple occurrences).
(4) Then in every 6-cycle we remove each arc that corresponds to each literal.

Note that this graph has 15 m edges in total: in each clause gadget, there are 6 edges from it to its 3 corresponding variable gadgets, 6 edges within its corresponding variable gadgets (because we allow multiple occurrences), and 3 edges within it. There are thus 15 m edges in total, as each clause gadget contributes to 15 edges and there are m clauses. Now we need to make a definition.

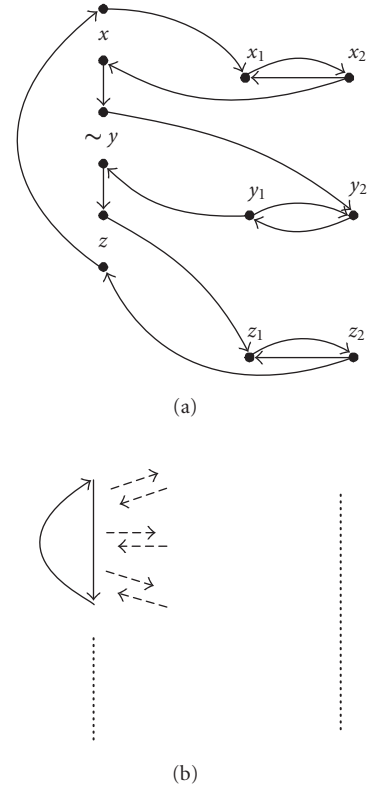

Figure 2: Clause and variable gadgets. Top part represents the clause $(x + \neg y + z)$. Note that the clause and variable gadgets are linked together.

Definition 2. A feedback arc set is a set of arcs that makes a graph acyclic when removed.

The *minimum feedback arc set* is a feedback arc set of minimum cardinality. It has the following properties.

Lemma 3. *A minimum feedback arc set is acyclic.*

Proof. An acyclic graph can be viewed as an ordering of the vertices such that all the arcs are in the forward direction, that is, for each arc (i, j), i comes before j in the ordering. Given a feedback arc set, consider such an ordering for the acyclic graph obtained upon deleting the feedback arc set. If the feedback arc set has any edges in the forward direction, then it is not minimum (such an edge can be added to the acyclic graph without creating any cycles). Thus the feedback arc set consists only of backward edges and hence is itself acyclic. □

Lemma 4. *The minimum feedback arc set either has all the edges from x_{i1} to x_{i2} and none of the edges from x_{i2} to x_{i1} or vice versa, for all i.*

Proof. If we include any edges from x_{i1} to x_{i2} and even one edge from x_{i2} to x_{i1} in the minimum feedback arc set, then it would not be acyclic, which is a contradiction to Lemma 3.

If we do not include all the edges from one of the sets in the minimum feedback arc set, then we will not have an edge from every cycle in the minimum feedback arc set, which is also a contradiction. □

Theorem 2. *The minimum feedback arc set for the graph G contains 3 m + u edges, where u is the minimum number of unsatisfied clauses of the formula F.*

Proof. This theorem will be proved as a consequence of two claims: (i) given an assignment for the variables in F that results in u unsatisfied clauses, we can construct a feedback arc set of size at most $3\,\mathrm{m}+u$; (ii) conversely, given a feedback arc set of size $3\,\mathrm{m}+u$, we can find an assignment for the variables of F such that no more than u clauses are satisfied.

First we observe that the graph G consists of lots of 12-cycles and many arcs within the variable gadgets (so the only possible cycles are those 12-cycles and the 2-cycles within the variable gadgets). Each 12-cycle contains 3 arcs within the clause, 2 arcs to and from each variable gadget (thus 6 arcs in total), and 2 arcs within each variable gadget (6 arcs in total too). The graph G thus has m 12-cycles as there are m clauses and each clause corresponds to a cycle. Now if we remove certain arcs within each variable gadget, then both the 12-cycles and the 2-cycles will be made acyclic.

(i) Given an assignment for the variables in F, we will show that we can find a feedback arc set including exactly 3 arcs from each satisfied clause and exactly 4 arcs from each unsatisfied clause. We construct the feedback set as follows: if x_i is set to TRUE, then we include all the arcs from x_{i1} to x_{i2}; if it is set to FALSE, we include all the arcs from x_{i2} to x_{i1}. In addition, we include one arc in the clause gadget corresponding to an unsatisfied clause. The resulting subset is a feedback set for the following reasons:

(1) including all arcs from x_{i1} to x_{i2} or from x_{i2} to x_{i1} will break all 2-cycles;
(2) in a satisfied clause, at least one literal will be true and the way we connect the clause gadget to it will break the 12-cycle;
(3) in an unsatisfied clause, including one more arc in the clause gadget will break the 12-cycle.

Thus, it is a feedback set having a total of $3m + u$ arcs.

(ii) Given a feedback arc set, we now show how to construct an assignment from it. First we delete edges from the feedback arc set until it is minimum. Then we assign each variable x_i in F a value depending on which set of edges with endpoints in $\{x_{i1}, x_{i2}\}$ is included in the feedback arc set. If all the edges from x_{i1} to x_{i2} are in the feedback arc set, the variable x_i is set to FALSE. Otherwise all the edges from x_{i2} to x_{i1} are in the feedback arc set, and then x_i is set to TRUE. Now we look at each clause and its corresponding variable gadgets. If the 12-cycle is broken because of at least one reversed arc in one variable gadget, then it must be a satisfied clause and exactly 3 arcs are added in the minimum feedback arc set. If, in one clause gadget, no reversed arc exists in any of the corresponding variable gadgets, then there must be another arc taken out and there must be exactly 4 arcs in the minimum feedback arc set. Therefore, if the feedback arc set has $3m + u$ arcs, the assignment leaves at most u clauses unsatisfied. □

Corollary 1. *The maximum acyclic subgraph for G is of size $11m+s$ where m is the number of clauses and s is the maximum number of satisfied clauses.*

4. APPROXIMATING MAX-COMPATIBLE SPP

In this section we are going to provide an efficient algorithm for approximating the maximum compatible security policy problem. Actually, it is not clear whether it is NP-hard or not because of its limitations (though we believe so). Our algorithm has approximation ratio $1 + 1/k$ for any given integer k. The computational complexity for our algorithm is $O(N^k)$. Our algorithm has three parts: (1) k-cycle removal, (2) marking of vertices, (3) arc removal.

4.1. Our algorithm

k-cycle removal

In this part, any cycle with degree less than or equal to k that contains the arc from an object to its negation will be removed. This can be done trivially for the following reason. To remove all j-cycles, we can generate all possible sequences of $(j-1)$ vertices and check whether they (along with the special arc) form a cycle or not. In a graph that has N vertices, such an attempt will take $O(N^j)$ time. Therefore, to remove all j-cycles, for all $j \le k$, it will take $O(N^k)$ to do so.

Marking of vertices

After executing the k-cycle removal part, we are sure that there are no cycles of order less than or equal to k. Now we mark all vertices as follows.

(1) Starting from the negated object. We mark it 0.
(2) If there is an arc that goes directly from the negated object to a vertex, we mark it 1.
(3) If there is an arc from a marked vertex to an unmarked one and the mark of that vertex is i, then we mark the other vertex $i + 1$.
(4) If there is an arc from a marked vertex (with mark i) to a marked one (which has been marked by some other vertex), then we compute its new mark $(i+1)$ and compare with its old mark. If the new mark is smaller than the old mark, then we remark that vertex with $i + 1$. Otherwise, do nothing.
(5) If all vertices have been marked already, we stop.

Arc removal

Now we look at the mark of the object. Since there is no k-cycle containing the special arc after executing the first part, we know that its mark is at least k. Let its mark be l $(l \ge k)$. Now we look at the relation between an arc and its vertices.

For an arc $e = (v_1, v_2)$, there are only two cases:

(1) $m(v_2) = m(v_1) + 1$, where $m(v)$ stands for the mark of v;
(2) $m(v_2) \leq m(v_1)$.

Note that $m(v_2)$ cannot be greater than $m(v_1)+1$ because that way it would have been remarked $m(v_1)+1$ according to step 4 of the marking algorithm. Now we group all of the arcs into $S_1, S_2, \ldots, S_l$ as follows:

$$S_j := \{e \in E \mid e = (u, v),\ m(v) = m(u) + 1 = j\}. \quad (7)$$

Let $T = \{e \in V \mid e = (u, v),\ m(v) \leq m(u)\}$, then these S_j's have the following properties:

(1) $S_i \cap S_j = \emptyset$ if $i \neq j$;
(2) $S_i \cap T = \emptyset$ for all i;
(3) $(\bigcup_{i=1}^{l}) \cup T = E$.

Now we choose one S_i of the smallest cardinality and call it S^*. From the above properties, we know that $|S^*| \leq (1/l)|E|$. Now we remove S^* from E and the rest of the arcs cannot have any cycle containing the special arc.

Lemma 5. *No cycle containing the arc from the object to its negation can be in $E - S^*$.*

Proof. Suppose there exists a cycle $(v_1, v_2, \ldots, v_m)$ $(m \geq l)$ that contains the special arc. Consider their marks $m(v_1), m(v_2), \ldots$. We know that $m(v_{i+1}) = m(v_i) + 1$ or $m(v_{i+1}) \leq m(v_i)$ for all $1 \leq i < m$, so if we take a tour from v_1 to v_m and look at their marks, at each step the mark cannot increase by 2 or more. We also know that $S^* = S_p$ for some $1 \leq p \leq l$. Then, at some point, there must be some v_i such that $m(v_i) = p$. Since marks cannot increase by 2 or more, $m(v_{i-1})$ must be $p - 1$. Then $(v_{i-1}, v_i) \in S_p = S^*$, which should have been removed. There is a contradiction. □

4.2. Performance analysis

Approximation ratio

Since $|S^*| \leq (1/l)|E|$ and $l \geq k$, we know that $|S^*| \leq (1/k)|E|$. It follows that $|E - S^*| \geq (1 - 1/k)|E| \geq (1 - 1/k)|\,\mathrm{OPT}\,|$ (where $|\,\mathrm{OPT}\,|$ is the size of the optimal solution). Then $|E - S^*|k/k - 1 \geq |\,\mathrm{OPT}\,|$. Finally we get $|E - S^*|(1 + 1/k - 1) \geq |\,\mathrm{OPT}\,|$. Since k is a dummy variable, given any integer k, we can choose $k - 1$ in the first place. Therefore the approximation ratio is $1 + 1/k$ for any integer k.

Computational Complexity

We already know that the first part of our algorithm takes $O(N^{k+1})$ time (since we choose k as $k - 1$ now). Both the second and the third are involved in going through all the arcs once, so the time complexity is $O(|E|)$, where $|E|$ is the number of arcs in the graph. If a directed graph has N vertices, then the number of arcs is $2\binom{N}{2} = N(N - 1)$, which is also a polynomial of N. It follows that the time complexity of the second and the third part cannot exceed $O(N^2)$. Overall, the time complexity of our algorithm is of order $O(N^{k+1})$.

5. RELATED WORK

Jajodia et al. [3] pointed out the problem that specification of security requirement may be quite complex in a large-scale system and proposed a logical language that deals with security policies. Dunlop et al. [1] and Abadi [2] used different graph-based approaches to locate and resolve a security conflict in a set of security policies. Schneider [5] addresses the questions for the class of enforcement mechanisms that work by monitoring execution steps of some target and terminating the target's execution if it is about to violate the security policy being enforced. Walker [6] talked about certified code for enforcing security properties. In his scheme, untrusted agent code carries annotations that allow a host to verify its trustworthiness. He used the host to check the annotations and proved that they imply the host's security policy. Hoagland et al. [7] also use directed graphs to represent security policies. They designed LaSCO, the language for security constraints on objects, to express many of the security policy situations and the composition of policies. Works [8–10] focused on access control policy (i.e., who has access to what and under what circumstances). Blaze et al. [11] specified under what conditions credentials are accepted. Bartal et al. [12] mentioned how a firewall is configured according to different security policies.

6. CONCLUSION AND FUTURE WORK

In this paper we have presented the maximum compatible security policy problem and its relationship to the maximum acyclic subgraph problem. We have proved that, in general, the maximum acyclic subgraph problem is NP-hard. We have also designed a polynomial time approximation algorithm and have shown that our result has approximation ratio $1 + 1/k$ for any integer k with complexity $O(N^{k+1})$. However, it is still not clear whether the maximum compatible security problem is NP-hard or not, nor is it clear whether there exists a better algorithm that can achieve a tighter bound. These will be interesting topics to dig in more.

REFERENCES

[1] N. Dunlop, J. Indulska, and K. Raymond, "Methods for conflict resolution in policy-based management systems," in *Proceedings of 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC '03)*, pp. 98–109, Brisbane, Queensland, Australia, September 2003.
[2] M. Abadi, "Logic in access control," in *Proceedings of 18th Annual IEEE Symposium on Logic in Computer Science*, pp. 228–233, Ottawa, Ontario, Canada, June 2003.
[3] S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A logical language for expressing authorizations," in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 31–42, Oakland, Calif, USA, May 1997.
[4] A. Newman, "Approximating the maximum acyclic subgraph," M.S. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass, USA, 2000.

[5] F B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000.
[6] D. Walker, "A type system for expressive security policies," in *Symposium on Principles of Programming Languages (POPL '00)*, pp. 254–267, Boston, Mass, USA, January 2000.
[7] J. A. Hoagland, R. Pandey, and K. N. Levitt, "Security policy specification using a graphical approach," Tech. Rep. CSE-98-3, University of California, Davis Department of Computer Science, Davis, Calif, USA, July 1998.
[8] D. E. Bell and L. J. LaPadula, "Secure computer systems: mathematical foundations and model," Tech. Rep. M74-244, MITRE Corporation, Bedford, Mass, USA, 1973.
[9] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.
[10] R. S. Sandhu and P. Samarati, "Access control: principles and practice," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994.
[11] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 164–173, Oakland, Calif, USA, May 1996.
[12] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: a novel firewall management toolkit," in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 17–31, Oakland, Calif, USA, May 1999.

Scott C.-H. Huang received his B.S. degree in mathematics from National Taiwan University in 1998, and his Ph.D. degree in computer science from University of Minnesota in 2004. He was a Postdoctoral Researcher at Florida International University from 2004 to 2005 and in 2005 he moved to City University of Hong Kong as a Research Fellow. His research area includes ad hoc and sensor networks, network security, and combinatorial optimization.

Kia Makki received his Ph.D. degree from University of California, Davis. He is currently a Chair Professor at Florida International University. His research area includes network security and multicasting, wireless networks, intrusion detection, adaptive routing and forwarding protocols, flow and congestion control, and information assurance.

Niki Pissinou received her Ph.D. degree from University of South California and she is currently a Professor and Director of IT2 at Florida International University. Her research area includes network centric middleware components, wireless information networks, distributed and wireless systems, and networked databases for newly emerging applications.