Vulnerability Analysis through a Graph-based Protection System

Mohammad Ebrahim Rafiei[1], Rasool Jalili[2], Hamid Mousavi[1].

Network Security Center, Department of Computer Engineering,
Sharif University of Technology, Tehran, Iran*

Summary
Vulnerability analysis *is the process of specifying, designing, and implementing a computer system without vulnerabilities, discovering unknown vulnerabilities, and detecting vulnerabilities' possible exploits. Some approaches to achieve such a process, integrate the concept of vulnerability into an access control model, and use ad hoc ideas to analyze them. Such approaches usually suffer from problems including weak modeling abilities and separation of authorized and unauthorized rules. To overcome such problems, we propose VGBPS as a new graph-based protection system with the main focus on vulnerabilities. Dealing with access rights, vulnerabilities, attributes, and relations similarly and using edge patterns to define rich types of rules, VGBPS adds the concept of vulnerability into a general access control model in a way that no extra effort is needed to handle vulnerabilities. In VGBPS, vulnerability analysis can be done by answering the* safety problem*. Considering safety problem more thoroughly, it is proven that safety problem, in the general form of VGBPS, is an NP-Complete problem. However, we introduce some simplified cases of the model, such as monotonically increasing systems and systems containing only* permanent *rules, in which the safety problem can be answered in polynomial time*

Keywords:
Vulnerability Analysis, Protection System, Safety Problem, NP-Completeness.

1. Introduction.

Vulnerabilities are those failures in software which may allow unauthorized access to attackers [1]. It is almost impossible to implement a software component with no bug or failure. Some approaches are required to deal with specifying, designing, and implementing a computer system without vulnerabilities, discovering unknown vulnerabilities, and detecting their possible exploits. This is usually referred to as vulnerability analysis. Most of the previous researches in the area of vulnerability analysis either focused on classification of vulnerabilities [2][3][4] or provided a model for vulnerability analysis [5][6][7][8].
In [5], Ramakrishnan and Seker proposed a model based on Prolog and attempted to use new model checking approaches to identifying vulnerabilities. Proposing a new analyzing tool, Farmer and Spafford tried to analyze vulnerabilities and their interactions in UNIX [6]. In [9], Amman and Ritchey analyzed some network vulnerabilities using a model checking approach. Graph-based techniques have also been used to design protection systems [5][7][8][9][10][11]. Polynomial time solutions to safety sub-problems were also proposed in some of them [7][10].

The approaches partially suffer from 1) disability to decrease access rights, 2) limited categories of rules, 3) state explosion in model checker based ones, 4) separation of authorized and unauthorized rules, as well as some other deficiencies mentioned in the literature which are out of our interest. In this paper, we focus mainly on separation of authorized and unauthorized rules. An access right or privilege can be added to (or deleted from) a model through two categories of commands; those indicated by the access control model policy (which we call them *authorized rules*) and those derived from a vulnerability (to which we refer as *unauthorized rules*). Based on the authors' knowledge, none of the existing models integrate these two categories together, while many exploit scenarios contain the both categories. Supporting such integration is one of our main contributions in this paper.

Another important shortcoming in many of the existing models is the lack of rules checking for the *absence of rights* as pre-conditions. It is also impossible to

* This research was partially supported by ITRC, # 500/8478

Manuscript received December 5, 2006.
Manuscript revised December 25, 2006.

define specific attributes or vulnerabilities for an entity in many of the models.

Considering the tradeoff between expressiveness of a model and the complexity of the safety problem in it, designing a simple and general model is not an easy task. HRU is one of the first general models in which the safety problem is challenged [12]. According to [13], *safety problem* is the question of: "Given an initial configuration of a protection system, whether subject *s* can obtain access right *r* on an object *o* or not." Other models such as SPM [14] and TAM [15] were also proposed. However, the safety problem is undecidable in all of the models.

Investigations to solve the safety problem resulted in the appearance of some models such as Take-Grant (TG) [16]. TG was initially proposed by Jones *et al.* and was extended by the other researchers [10][17]. Frank and Bishop in [17] introduced the notion of *cost* in TG in order to find the shortest path toward the creation of an access right. Shahriari *et al.* in [10] extended TG and added some rules standing for the vulnerabilities exploitations. They used their model to analyze some network vulnerabilities.

In this paper, we propose Vulnerability Graph-Based Protection System (VGBPS) which integrates concept of vulnerabilities into a general access control model. Vulnerability analysis in the model is achieved using the safety problem, considering both authorized and unauthorized rules. In other words, integration of the concept of vulnerabilities into the model helps investigation of their effects on leakage of access rights, through answering the safety problem. The model is aimed to deal with as much features of the real-world systems as possible, whilst the safety problem is solvable in a reasonable time. To provide the features in VGBPS, we use the same structure for access rights, vulnerabilities, attributes, and relations as well as the concept of *edge patterns*. The generality and flexibility of the definition of rules also help to provide the features. We prove that the safety problem in VGBPS is an NP-Complete problem. However, we show how the model can be simplified to some cases in which the safety problem is polynomially solvable. As an example, it is shown how to solve the safety problem in polynomial time for monotonically increasing systems.

The rest of the paper is organized as follows. An overview of VGBPS and its formal description are presented in sections 2 and 3. The safety problem in VGBPS is discussed in section 4. In sections 5 and 6, two main applications of the model are proposed. Some polynomially solvable sub-problems of the safety problem are discussed in section 7. In section 8, we prove the NP-completeness of the general problem. Potential of the model is discussed in section 9. Finally, section 10 concludes the paper.

2. An overview of VGBPS

Most vulnerability analysis models considered access rights and vulnerabilities from different perspectives. The idea of combining vulnerabilities into an access control system was initially introduced in [10]. Such a new category of models attempts to define vulnerabilities and their consequences as some access propagation rules appended to the conventional access control rules. As a result, the models can easily discover and analyze potential attacks though both conventional access control rules as well as vulnerabilities related rules. However, inclusion of vulnerabilities into the access control model is not sufficient. The extended model should also be as general and simple as possible. It should be general in the sense that different aspects of security systems to be covered. Simplicity of the model is essential due to the requirement of solving the safety problem in a reasonable time. Satisfying all the mentioned objectives, we propose Vulnerability Graph-Based Protection System (); which is a graph-based protection system provisioning vulnerabilities.

In order to support simplicity and expressiveness, several access control systems utilize graph as the basis of their models. VGBPS makes use of the graph as its basic modeling concept to represent all the system entities and their characteristics. Characteristics of entities include their attributes and vulnerabilities plus access rights and inter-entity relations. VGBPS models these characteristics using a simple labeling technique on the modeling graph edges. Access rights in VGBPS have the same meaning as in access control systems. Vulnerabilities in VGBPS indicate the association of some known vulnerabilities to some entities. Similar to attributes in ABAC [18], properties of entities such as age, type, and location are modeled as attributes in VGBPS. By relation, in VGBPS, we model interactions (such as the parent/child relationship) between pairs of entities. The structure represents the static view of the model which provides the possibility of modeling more complex systems and dealing similarly with the four abovementioned characteristics.

To signify the dynamic view of VGBPS, it is required to define rules (commands). Each rule has some pre-conditions and some post-conditions. Rules are formed as four sets of edge patterns. Informally, each edge pattern is a pattern indicating a set of edges in the model. Two of the edge patterns' sets specify pre-conditions, introducing those edges whose existence/non-existence is compulsory in the modeling graph. The other two sets denote post-conditions, indicating edges to be added to and deleted from the modeling graph.

The proposed model aims to be a general and simple framework covering both vulnerability and access control concepts simultaneously. It primarily provides a basis for analyzing the effects of vulnerabilities more easily. In the next sections, after providing a formal description of the model, the way of expressing some other access control system through VGBPS is presented. The main contribution of this paper is to express and model vulnerabilities and their effects using a general access control model as well as demonstrating the safety problem complexity in different cases.

3. VGBPS in Detail

A formal specification of VGBPS is provided in theis section. VGBPS is defined as a tuple (G, R), where G is the current modeling graph and R is the set of rules indicating how G can be changed. The definition does not include the safety problem, which is covered later in this paper. The following subsections discuss the modeling graph G and the rule set R respectively.

3.1. The Modeling Graph G:

Let V_{all} be all entities (vertices or nodes) in the system and E_{all} be all potential edges. G(V, E) is the modeling graph where $V \subseteq V_{all}$ and $E \subseteq E_{all}$. For the sake of simplicity, we define edges as a triple (*v, u, l*) in G, where *v* and *u* are source and destination vertices (nodes) and *l* denotes the edge's associated label. The label set L consists of four sets L_{vul}, L_{attr}, L_{rgt}, and L_{rel}, to demonstrate vulnerabilities, attributes, access rights, and relations respectively. In other words:

$$v,\ u \in V_{all},\ l \in L,\ L = L_{vul} \bigcup L_{attr} \bigcup L_{rgt} \bigcup L_{rel}$$

Using our definition of edges; vulnerabilities, attributes, access rights, and relations can be dealt with similarly. To depict that a node *v* has *read* access over *u,* we can use the edge (*v, u, read*). To demonstrate a vulnerability *vul* in node *a*, the loop edge (*a, a, vul*) can be used. To assign an attribute *attr* to the node *a*, the loop edge (*a, a, attr*) may be employed. Having a relation *rel* between *a* and *b*, the edge (*a, b, rel*) can be used. Based on the nature of vulnerabilities and attributes, L_{vul} and L_{attr} should be used only in loop edges. Accordingly, the following condition should be held:

$$\forall\ E = (v_0, v_1, l) \mid l \in L_{vul} \bigcup L_{attr} \rightarrow v_0 = v_1$$

Having such a graph, we mainly focused on edges to demonstrate all characteristics under study. In this graph, nodes are created initially, and they will be remained unchanged. There is no need to remove a node; instead we can remove all its connected edges. We also assume that there is no need to create new nodes. One can simulate create command for nodes by considering some extra nodes in the modeling graph with no edge and simply use one of them instead of creating a new node. However, this resulted in the creation of a limited number of nodes.

3.2. Commands in VGBPS

Edge pattern plays an important role in VGBPS rule definition. An *edge pattern* is a triple (*a*, *b*, *t*), where *a* and *b* belongs to the set of defined phrases called *PV* (Pattern Variables), and $t \in L$. We refer to the set of all possible edge patterns as EP. The most important concern regarding edge patterns is to identify edges matching an edge pattern.

Definition 1. We say edge *e*(*v*, *u*, *l*) *matches* edge pattern *ep*(*a*, *b*, *t*); if and only if *l* and *t* be identical and if *a* = *b* then *v* = *u*. In this case, we say *a* and *b* respectively match *v* and *u* or vice versa. Formally:

$$match\text{: } EP \times E_{all} \rightarrow \{\text{true, false}\}$$
$$match((a, b, t), (v, u, l)) = \text{true} \Leftrightarrow t = l \wedge (a = b \rightarrow v = u)$$

For example, the edge pattern *ep*(*a*, *b*, *r*) matches all edges labeled *r*, or *ep*(*a*, *a*, *o*) matches all loop edges labeled *o*. An edge pattern is not individually useful in rule definition; a set of edge patterns is required to be matched with a set of edges.

Definition 2. Suppose that EPS is a subset of EP, (that is EPS is a set of edge patterns). EPS matches with $E_m \subseteq E_{all}$, if and only if:

$$|E_m| = |EPS| \wedge$$
$$\forall ep \in EPS \mid (\exists e \in E_m \mid match(ep,\ e)) \wedge$$
$$\forall ep_1, ep_2 \in EPS, e_1, e_2 \in E_m \mid match\ (ep_1, e_1)$$
$$\wedge\ match\ (ep_2, e_2) \rightarrow$$
$$((ep_1.a = ep_2.a \rightarrow e_1.a = e_2.a) \wedge$$
$$(ep_1.b = ep_2.b \rightarrow e_1.b = e_2.b) \wedge$$
$$(ep_1.a = ep_2.b \rightarrow e_1.a = e_2.b))$$

where *e.a* and *e.b* are the first and the second items of tuple *e*. We call E_m a setmatch of EPS. The definition implies that if any $a \in PV$ matched with a vertex *v* in one of the edge-match, it can not match with any other vertex.

Definition 3. Rule set R is the set of rules that each of them is a tuple (EP_e, EP_n, EP_a, EP_d) where EP_e, EP_n, EP_a, EP_d are all subsets of EP.

Informally, EP_e and EP_n are two sets of edge patterns indicating which edges should exist/not exist in order that the rule can be applied, and EP_a and EP_d respectively represent new edges which will be added to and the edges which will be removed from graph G after the rule application.

Definition 4. Rule *r*(EP_e, EP_n, EP_a, EP_d) is *applicable* if there exists a setmatch E_m for EPS = $EP_e \bigcup EP_n \bigcup EP_a \bigcup EP_d$ where

$$matchededge(EP_e, E_m, EPS) \subseteq E\ \wedge$$
$$matchededge(EP_n, E_m, EPS) \subseteq E_{all} \setminus E$$

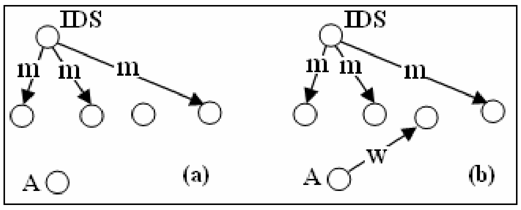

Fig. 1. A simple IDS model: a) before applying the IDS rule. b) after applying the IDS rule.

where *matchededge*(EP_x, E_x, EPS_x) returns the related edges to edge patterns in EP_x from found setmatch E_x for EPS_x. Rule *r* is also *effective* if:

$$matchededge(EP_a, E_m, EPS) \not\subseteq E \vee$$
$$matchededge(EP_d, E_m, EPS) \not\subseteq E_{all} \setminus E$$

Formally, applying an applicable rule $r(EP_e, EP_n, EP_a, EP_d) \in R$ in protection system PS(G, R) means to update current modeling graph G to G' as follows:

$$V_{G'} = V_G,$$
$$E_{G'} = (E_G \bigcup matchededge(EP_a, E_m, EPS)) \setminus matchededge(EP_d, E_m, EPS)$$

where E_m is a setmatch for $EPS = EP_e \bigcup EP_n \bigcup EP_a \bigcup EP_d$. The application of rule *r* on modeling graph G which generate new modeling graph G' is also shown with $G \mapsto_r G'$. We can show this using $PS(G, R) \mapsto_r PS(G', R)$, but we do not usually use this since the rule set R is unchangeable.

The rule definition is more general than the one proposed in [12] especially because of the existence of set E_n. For example, consider a system running an Intrusion Detection System (IDS) to monitor activities of some entities. Also, assume that the attacker *A* can gain write access to a given object if it is not being monitored by IDS. As shown in Fig. 1, the system can be represented in our model by a new vertex in the modeling graph in place of the IDS alongside with its associated edges, labeled m, to the nodes which it is monitoring. Adding the following rule can show how an attacker may reach her goal:

$$r(\varnothing, \{(b, o, m)\}, \{(a, o, w)\}, \varnothing)$$

The new rule implies that if the IDS (*b*) is not monitoring (*m*) a specific object, called *o*, then the attacker (*a*) can gain write (*w*) access to *o*. As you can see, VGBPS can model this kind of exploits very easily, since many of current model needs extra efforts to model it and some of them are not even capable of modeling it. Note that too many of such exploits can be addressed in the real world.

4. Safety Problem in VGBPS

Before defining the safety problem, we express the concept of witness and the predicate can●share [19].

Definition 5. Having a protection system PS(G_0, R), a *witness* is a sequence of rules, $r_1, r_2 \dots r_n$, ($r_i \in R$, $1 \le i \le n$) which the first one is applicable to the current modeling graph G_0 and r_{i+1} is applicable to the resulted modeling graph after application of first *i* rules in the sequence. That is $G_0 \mapsto_{r_1} G_1 \mapsto_{r_2} G_2 \dots \mapsto_{r_n} G_n$.

Definition 6. Let *l* be a label ($l \in L$) and, *v* and *u* be two distinct vertices in the protection system PS(G, R). The predicate can●share(*v, u, l*) is true in PS if and only if there is a witness whose application to modeling graph G generates a new graph containing the edge (*u*, *v*, *l*).

Definition 7. Having the protection system PS(G, R) and a set of can●share predicates P, the *Safety Problem* is the problem of finding a witness *w* whose application to graph G violates (makes true) at least one of the predicates included in P.

While there are other predicates which can be included in P, they are not required to be considered explicitly. This is due to the possibility of answering them based on our answer to the basic predicate can●share. As an example, we show how to answer the can●revoke predicate using can●share. The predicate can●revoke can be defined as follows.

Definition 8. Let *v* and *u* be two distinct vertices in the modeling graph G_0 of the protection system PS(G_0, R) and there is an edge between *v* and *u* labeled $r \in L$. The predicate can●revoke(*v*, *u*, *r*) is true in PS if and only if there is a witness whose application to G_0 generates a new graph containing no edge from *v* to *u* labeled ***r*.**

To answer this predicate, we can add following rule to the set of rules R:

$$newrule(\varnothing, \{(a, b, r)\}, \{(a, b, r')\}, \varnothing)$$

Provided that label *r′* has not been used in any other rules or edges, can●share(*v, u, r′*) will not be satisfied without using the new rule, which involves non-existence of an edge labeled *r* between *v* and *u*. Accordingly, the predicates can●revoke(*v*, *u*, *r*) and can●share(*v*, *u*, *r′*) can be used interchangeably. Despite all of these, one may add its own defined predicates to the security policy P. Obviously in special cases of the model (for example when EP_n of all rules is empty), can●share may not be able to describe the whole policy and it may be obligatory to add some new predicate to P.

5. Simulating other Protection Systems

To illustrate the expressiveness of VGBPS, it can be shown that many of the existing models can be simulated using our protection system. As an example, we show how to simulate Take-Grant model [16][19] using our protection system. Let $L_{attr} = \{sub, obj\}$, $L_{rgt} = \{t, g, r, w\}$, $L_{rel} = L_{vul} = \varnothing$, and $PV = \{a, b, c\}$; PS(G, R) simulates the known Take-Grant model, where G is almost the same as the Take-Grant graph model. The only difference is that in G we use loop edges for nodes' attributes (subject or object). The rule set R includes the following rules:

- $take_r$ $(\{(a, b, t), (a, a, sub), (b, c, r)\}, \varnothing, \{(a, c, r)\}, \varnothing)$
- $grant_r$ $(\{(b, a, g), (b, b, sub), (b, c, r)\}, \varnothing, \{(a, c, r)\}, \varnothing)$
- $remove_r$ $(\{(a, b, r), (a, a, sub)\}, \varnothing, \varnothing, \{(a, b, r)\})$

Similar to $take_r$, we can also define $take_t$, $take_w$, and $take_g$. Obviously, more complex models can also be simulated similarly.

6. Using the Model for Vulnerability Analysis

As already stated, the model facilitates an easier vulnerability analysis by dealing with vulnerabilities similar to access rights. For example, consider the SQL-injection vulnerability, in which an application places input directly into a SQL statement without filtering out its dangerous characters properly. This may grant the users some unprivileged access rights. Fig. 2 shows how we can model this vulnerability in its simplest case. Briefly, the *SQL_inj* attack says that if an application *App*, having access to some part (tb_i) of a database (*DB*,) has *SQL_inj* vulnerability, then any user (*A*) having execute access on *App* may gain the access to that part (tb_i). This attack can be shown as the following rule:

r_{SQL_inj} $(\{(a, app, exec), (app, app, SQL\_inj), (app, tb, access), (tb, c, b), (c, c, DB)\}, \varnothing, \{(a, tb, access)\}, \varnothing)$

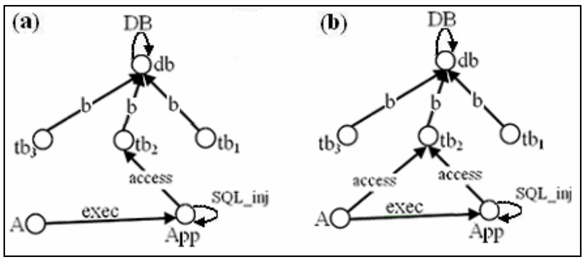

Fig. 2. a) The system before applying the rule r_{SQL_inj} and b) The system after application of the rule. The edge (*tb, db, b*) indicates that node *tb* belongs to *db*, and (*db, db, DB*) indicates that node *db* is a database.

As the example shows, modeling vulnerabilities and also their possible effects do not need any new consideration in the protection system. This way, many of tasks related to vulnerability analysis can be done only by answering the safety problem. For example, to find out exploits of SQL-injection, it is enough to find out which access rights can be added to the model using r_{SQL_inj} beside other existing rules of the system and this can easily be done by answering some can●share predicates.

7. Polynomially Solvable Sub-Problems

In order to demonstrate the flexibility of VGBPS and its capability to model special systems with less cost, we provide some instances of the model in which safety problem can be solved in polynomial time. Let's call the rules which can decrease edges from the modeling graph in the protection system as *decreasing rules*. Initially, we can divide decreasing rules into two main classes; the rules which do not add new edges to the model, and those which add new edges in addition to removing some other edges. Later we will show that we can eliminate the former rules in some cases.

Definition 9. *Monotonically decreasing* rules are rules which may decrease edges, that is, $EP_a = \varnothing$.

Most of DoS attacks [20] and also DDoS attacks are clear examples of monotonically decreasing rules, which may cause some services to be inaccessible, without adding new access rights (edges) to the system. Similarly, we can define monotonically increasing rules. We call these two types of rules monotonic rules.

Definition 10. *Simple rules* are the rules in which $EP_n = \varnothing$.

In other words, applying a simple rule just involves checking existence of some edges. Application of simple rules take O(1) time, as the sets EP_e, EP_a, and EP_d can contain O(1) number of edge patterns. Recall that in its general form, application of a rule may require the graph not to include some edges. Most of the previously proposed graph-based models are restricted to simple rules. Whereas, there are some kinds of exploits in which we have to be sure that some edges do not exist in the modeling graph such as the IDS exploit example described in section 3.

Theorem 1. In a protection system PS(G, R), the predicate can●share can be answered in polynomial time, if R contains only rules that are both simple and monotonically increasing.

Proof. An algorithm like the one proposed by Frank and Bishop in [17] can compute the closure of the protection system as shown in Fig. 3. The closure is informally a graph resulted form initial modeling graph G after applying all possible rules. The proof of polynomial time complexity of this algorithm is similar to theirs as well. Thus, to answer can●share(*v*, *u*, *r*), we can check whether the edge (*v*, *u*, *r*) exists in the closure or not.

Corollary 1. In a protection system PS(G, R), the predicate can●share can be answered in polynomial time, if R contains only those rules which are both simple and monotonic.

Monotonically decreasing rules can not cause new edges to be added to the modeling graph, since the rules are *simple*. Thus, we can simply eliminate all monotonically decreasing rules and use the closure-based algorithm described above to answer the predicate can●share in polynomial time.

Corollary 2. In a similar way it can be shown that if all defined rules were simple, then monotonically decreasing rules could be removed from the protection system. But this time, since some decreasing rules still may exist in the model, we can not use the same closure-based algorithm to answer the safety problem.

```
1. Let list F be initiated by the set of all
     edge of the modeling graph PS.G
2. while (!isEmpty(F))
3.   e = head (F)
4.   Check for all applicable monotonically
     increasing rules which e is involved in
5.   foreach (resulting edge like f)
6.     add f to PS.G
7.       if (f has never been in F before)
8.         Add f to F
9.     Delete e from F
10.return PS
```

Fig. 3. Gen_Closure_4MIR*: A polynomial time algorithm which answers can●share predicate when the protection system contains only simple and monotonically increasing rules.*

Definition 11. An edge in a modeling graph is permanent, if and only if its associated access right will never be removed because of deletion of any other edges.

The definition implies that no matter whether the conditions which have caused adding a permanent edge still hold or not, it will continue to exist in the graph permanently, unless it will be deleted by a rule. As an example of this type of access right, suppose the attacker A wants to use the passwords stored in a file f on a host in its attack scenario. As soon as A achieves read access to f, it has reached its goal. Even encrypting the password file f later on, (which removes the read access of A,) will not hide the achieved information from A, because A has already read what it wanted. Thus, the read access is permanent. Therefore, the only way to remove a permanent edge is to delete it directly by a rule; in this example, changing the password as an instant.

We will refer to edges which are not permanent as impermanent edges**.** As an example, consider that an attacker wants to use a service which needs authentication. Suppose the attacker has acquired the information of an account for the service. The attacker can use the service as long as the promised account has not been disabled. Therefore, the attacker's access to the service is impermanent. ■

Definition 12. A rule is permanent if it generates only permanent edges; otherwise it is impermanent.

In the case of impermanent rules, if the preconditions of a previously applied rule dose not hold any more, in addition to removing the edges generated by that rule, those edges which has removed because of that rule should be added again to the modeling graph. Next theorem deals with an interesting property of the systems which use only permanent and simple rules:

Theorem 2. Let PS(G, R) be a protection system in which all initial edges are permanent and only the permanent and simple rules are allowed. In such a system, the predicate can●share can be answered in polynomial time.

Proof. The main idea is to construct the closure of the modeling graph using a conflict graph. The conflict graph has one vertex in association with every possible edge in the modeling graph and is initially empty. We say that two edges in the closure have conflict (according to the conflict graph) if and only if there is a directed path between their related vertices in the conflict graph. The rule rl(EP_e, ∅, EP_a, EP_d) is applicable according to the conflict graph if and only if the setmatch found for EP_e contains non-conflicting edges according to the conflict graph.

Initially the closure is identical to the graph G. In each step of the algorithm, we will apply an applicable rule according to the conflict graph and update two graphs (closure and conflict graphs) as follows; Let rl(EP_e, ∅, EP_a, EP_d) be the selected rule. To apply *rl* the edges produced by EP_a will be added to the closure and all their conflicts will be removed from the conflict graph (that is, for each edge *e* produced by EP_a, we will remove all the edges outgoing from *e*'s related vertex in the conflict graph). The edges included in EP_d , if there is any, will not be removed from the closure; instead we will add directed edges from EP_d's associated vertices in the conflict graph to those of EP_a.

This step is repeated until there is no applicable rule according to the conflict graph. Thus, if there is an edge from vertex *v* to vertex *u* labeled *r* in the computed closure, the answer to the predicate can●share(v, u, r) will be yes. Obviously, the algorithm is polynomial because in each step at least one edge will be added to the closure. Note that there is no need to consider monotonically decreasing rules according to corollary 2. Since the closure contains polynomial number of edges and each step of the algorithm needs polynomial time, the time complexity of the algorithm will be polynomial too.

8. NP-Completeness Results

To show the NP-Completeness of the general problem, we use reduction from 3-SAT [21]. 3-SAT is a special case of Boolean satisfiability (SAT) problem. SAT is the first established NP-complete problem [22] and all typical NP-Complete problems can be directly reduced to it.

Basically, SAT problem is either to find a satisfying truth assignment of all variables or to prove there is no satisfying assignment for a given Boolean formula (usually in Conjunctive Normal Form (CNF).) 3-SAT is a special case of SAT in which each clause has exactly three literals.

Theorem 3. The safety problem in the protection system PS = (G, R) is NP-complete if the initial edges and rules are not necessarily permanent.

Proof. The safety problem can be interpreted as finding a witness which makes one or more predicates in P true. Therefore, it is clear that if we have a witness, it is possible to verify whether the witness violates security policy or not in polynomial time. That is, the problem is in NP. Note that since a witness includes only different rules it size is polynomial. To prove NP-Completeness, we use reduction from 3-SAT. Let φ be an instance of 3-SAT problem. We construct protection system PS(G, R) such that φ is satisfiable if and only if there exists a witness in PS which violates at least one of the predicates included in P.

Corresponding to each clause C_i in φ ,G has a node C_i. There are three vertices y_i, x_i and $\bar{x}_i$ in G, corresponding to each literal x_i in 3-SAT problem. Graph G also contains two other vertices T and T'. T has a specific type called m which is shown by a loop labeled m on it. For each literal x_i, we add directed edges from y_i to both x_i and $\bar{x}_i$ labeled r. If the i^{th} clause contains x_j ($\bar{x}_j$), we will put a directed edge from C_i to x_j ($\bar{x}_j$) labeled s and true, and another directed edge to $\bar{x}_j$ (x_j) labeled false. For each i, there is an edge from y_i to T labeled notok. This means that yet we do not know whether x_i has consistent values in all of its occurrences. The initial graph which is constructed from a sample 3-SAT problem is shown in Fig. 4.

The rule set R contains 5 rules:

1. If the clause c contains the literal x (a variable or its negation), we will assign the true value to x and false to its negation:

   r_1 ({(c, x, s), (y, x, r), (y, x', r), (c, x, false)}, ∅, {(c, x, true), (c, x', false)}, {(c, x, false), (c, x', true)})

2. If the clause c contains literals x_1 and x_2 in which both have the true value, we can assign the false value to one of them and true value to its negation:

   r_2 ({(c, x_1, s), (y, x_1, r), (y, x'_1, r), (c, x_1, true), (c, x_2, s), (c, x_2, true)}, ∅, {(c, x_1, false), (c, x'_1, true)}, {(c, x_1, true), (c, x'_1, false)})

3. If the literal x has got true value in all clauses which contain x, a new edge can be added to the modeling graph G from y to T labeled ok, where y is the vertex having an edge to x labeled r:

   r_3 ({(y, x, r)}, {(a, x, false)}, {(y, T, ok)}, {(y, T, notok)})

4. Similar to the previous rule for x's false value:

   r_4 ({(y, x, r)}, {(a, x, true)}, {(y, T, ok)}, {(y, T, notok)})

5. If all literal get consistent values in their related clauses, a new edge can be added from T to T' labeled ok which shows that φ is satisfiable (m is the type of node T):

   r_5 ({(T, T, m)}, {(a, T, notok)}, {(T, T', ok)}, ∅)

It is not so hard to show that if there is a satisfying truth assignment for φ , there will be a witness in SP which violates the predicate can●share(T, T', ok) and vice versa.

9. Discussion

As already mentioned, VGBPS is an appropriate framework for vulnerability analysis. The rules (commands) in this model can basically be considered in two categories; ordinary access propagation rules just like rules in access control models and vulnerability related rules. The latter one checks for the existence of a vulnerability in its pre-condition. The way, we defined general rules, make it possible to define many other forms of rules too. Having both forms of rules, we can study the effect of vulnerabilities in access propagation much more easily. For example, we can simply use an edge created from a vulnerability rule as the pre-condition of an access propagation rule or vice versa. This is simply what we can not have in many other vulnerability analysis models.

Additionally, VGBPS considers the non-existence of a right as pre-condition of rules too, which many famous protection systems such as HRU, TAM, and TG do not support it. This type of rules is especially important in modeling SoD (Separation of Duty). We also consider the decreasing rules in VGBPS, which is necessary in modeling DoS and DDoS attacks. Most of vulnerability analysis techniques do not consider this concept.

Combining decreasing rules with increasing ones, we can model some complex events such as changing group of a user easily.

Adding concepts such as nodes attributes, their relations, and of course their vulnerabilities make it possible to express those behaviors of real systems which are important in vulnerability analysis with no trouble. This was not provided in early models like HRU and TG, but they are advancing in new models such as TAM and ABAC. Moreover, we tried to express this kind of features uniformly. This makes the vulnerability analysis easier, since we can deal with different concepts in the same way. Using the concept of edge-pattern also helps with this uniformity.

Because of dealing with the concept of vulnerabilities just like attributes and access rights, the task of vulnerability analysis will highly be depend on the safety problem. In other words, many of tasks in vulnerability analysis can be done only by answering the safety problem. Thus, we examine different variation of the safety problem deeply in this paper.

Another important application of VGBPS may be in vulnerability analysis based on other models. For example, one may want to add SQL injection vulnerability to TG and analyze it. Through the paper, we showed how to simulate other models with VGBPS. This way, we can just simulate TG with VGBPS and add the SQL injection to selected nodes in the new model and add its related rules. This way, the new model can easily be used in vulnerability analysis.

10. Conclusions and Future Work

In this paper, we proposed VGBPS which is a new graph-based protection system with the main focus on vulnerability analysis. Dealing with different concepts such as access rights, vulnerabilities, relations, and attributes similarly, as well as the edge-pattern concept and general type of rules are of the most important features of VGBPS. We also showed how VGBPS can simulate some other models, and how it can contribute in vulnerability analysis more easily.

As the safety problem is widely used for vulnerability analysis in VGBPS, it gets more importance. Although we showed that the safety problem in its general form in VGBPS is NP-Complete, there are still cases in which the safety problem can be answered in polynomial time. We introduced ideas of simple, permanent, and monotonically decreasing rules alongside with some simplified sub-problems of the safety problem. We illustrated how the monotonically decreasing rules can be eliminated when a system contains only simple rules. We also proved that the safety problem can be answered polynomialy in a system which is restricted to rules which are both permanent and simple.

Investigating more polynomialy solvable problems can be considered as an interesting future work. Especially, relevant problems to impermanent rules are worthy of considering more thoroughly. Using the system to model vulnerabilities and trying to analyze them are of other future works. We are especially interested to compare the effects of vulnerabilities while being located in different nodes of the system or while interacting with each others.

References

[1] M. Bishop, "Computer Security: The Art and Science," Addison-Wesley, 2003.

[2] M. Bishop, "Vulnerabilities Analysis," Proceedings of Recent Advances in Intrusion Detection, pp. 125–136, 1999.

[3] T. Aslam, I. Krsul, and E. H. Spafford, "Use of A Taxonomy of Security Faults," Proceedings of the 19th National Information Systems Security Conference, pp. 551–560, 1996.

[4] R. Abbott, et al., "Security Analysis and Enhancements of Computer Operating Systems," Technical Report NBSIR 76–1041, ICET, National Bureau of Standards, Washington, DC 20234, 1976.

[5] C. Ramakrishnan and R. Sekar, "Model-based Vulnerability Analysis of Computer Systems," Proceedings of the 2nd International Workshop on Verification, Model Checking, and Abstract Interpretation, 1998.

[6] D. Farmer and E.H. Spafford, "The Cops Security Checker System", Technical Report CSDTR-993, Purdue University, 1991.

[7] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, Graph-based Network Vulnerability Analysis," Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, 2002.

[8] D. Zerkle and K. Levitt, "NetKuang - A Multi-host Conjuration Vulnerability Checker," Proceedings of 6th USENIX Security Symposium, San Jose, California, pp. 195-204, 1996.

[9] R.W. Ritchey and P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 156-165, Oakland, 2000.

[10] H.R. Shahriari, R. Sadoddin, R. Jalili, R. Zakeri, and A.R. Omidian, "Network Vulnerability Analysis through Vulnerability Take-Grant Model (VTG)," Proceedings of the 7th International Conference on Information and Communications Security (ICICS'05), 2005.

[11] C. Phillips and L. Swiler, "A Graph-based System for Network-Vulnerability Analysis," Proceedings of the New Security Paradigms Workshop, pp. 71-79, Charlottesville, VA, 1998.

[12] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman, "Protection in Operating Systems," Communications of the ACM, 19(8), pp. 461–471, August 1976.
[13] J.S. Shapiro, "The Practical Application of a Decidable Access Control Model", Technical Report SRL-2003-04, John Hopkins University, 2003.
[14] R. Sandhu, "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes," Journal of the ACM, 35 (2), pp. 404–432, 1988.
[15] R. Sandhu, "The Typed Access Matrix Model," IEEE Symposium on Security and Privacy, pages 122-136, 1992.
[16] A.K. Jones, R.J. Lipton, and L. Snyder, "A Linear Time Algorithm for Deciding Security," Proceedings of the 17th Annual FOCS Conference, pp. 33-41, Houston, 1976.
[17] J. Frank and M. Bishop, "Extending the Take-Grant Protection System," Technical Report, 1996.
[18] X. Zhang, Y Li, and D. Nalla, "An Attribute Based Access Control Matrix Model," Proceedings of the 2005 ACM symposium on Applied computing, 2005.
[19] M. Bishop, "Conspiracy and Information Flow in the Take-Grant Protection Model," Journal of Computer Security, vol. 4(4), pp. 331-360, 1996.
[20] A.R. Sharafat and M.S. Fallah, "A Framework for the Analysis of Denial of Service Attacks," The Computer Journal, Oxford University Press, Vol. 47, No. 2, pp 147-162, 2004.
[21] M.R. Garey, and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W.H. Freeman, New York, USA, 1979.
[22] S.A. Cook, "The Complexity of Theorem Proving Procedures," Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing, New York, pp. 151-158, 1971.

Mohammad Ebrahim Rafiei received his B.Sc. degree in Software Engineering from University of Tehran, Iran in 2004. He is currently an M.Sc. student in Sharif University of Technology, Tehran, Iran. His research interests are Access Control models and Vulnerability Analysis.

Rasool Jalili received his PhD in Computer Science from The University of Sydney, Australia in 1995. He joined the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, as an assistant professor. His research interests are Distributed Systems and Information Security.

Hamid Mousavi received his B.Sc. degree in Software Engineering from University of Tehran, Iran in 2004. He is currently an M.Sc. student in Sharif University of Technology, Tehran, Iran. His research interests are Security Models, Web Search Engines, and Internet Measurements.