A Novel Model of Security Policies and Requirements

Preetam Mukherjee[1] and Chandan Mazumdar[2]
[1]*Centre for Distributed Computing, Jadavpur University, Kolkata, India*
[2]*Department of Computer Sc. & Engg., Jadavpur University, Kolkata, India*

Keywords: Security Requirements, Security Policies, Security Violations, Process Algebra.

Abstract: The responsibility of controlling, monitoring, analyzing or enforcing security of a system becomes complex due to the interplay among different security policies and requirements. Many of the security requirements have overlap among themselves and they are not exhaustive in nature. For that reason, maintaining security requirements and designing optimal security controls are difficult, and involve wastage of valuable resources. Finding out a set of mutually exclusive and exhaustive security requirements and canonical policies will indeed ease the security management job. From this motivation, in this paper we try to find out a set of mutually exclusive and exhaustive security requirements. To do this, a small set of low-level security policy descriptions are proposed using Process Algebraic notions, by which all kinds of high level security policies can be represented. Non-compliance to this new set of security policies gives rise to a set of security violations. These security violations are mutually exclusive and exhaustive, so all the other security violations can be described by this basic set of security violations. From these security violations, a set of security requirements is determined. To preserve the security for any system it is necessary and sufficient to maintain these requirements.

1 INTRODUCTION

For securing an Information Technology (IT) system, each IT entity needs to comply with the security policies imparted on them by the administration. Policies can be of different types, including access control policies, information flow control policies, obligation policies etc. When an entity does not fulfil a security policy, the non-compliance to the policy is termed *security violation by that entity*.

Security policies for a system are developed from the point of view of maintaining security requirements. In literature, different security requirements have been defined including confidentiality, integrity, availability, etc. These requirements are defined from different points of view and some of them are semantically overlapping. As such it is difficult to ascertain that all security issues relevant to a system are covered by the selected set of security requirements. Several security models are proposed in literature to find out inter relationships of these security requirements. But no model in the literature, to the best of our knowledge, has proposed the basic set of non-overlapping security requirements by which all the other requirements of security can be addressed. This basic set of security requirements is indeed required to precisely monitor, analyze, enforce or control security. This motivates us to gather and define a set of mutually exclusive and exhaustive security requirements for an IT entity.

In this paper, we have taken the notion of process and action from the process algebraic framework to model an IT entity and its actions (Milner, 1989). Based on the above notions a novel method of describing low-level Security Policies is presented. The proposed policies are dynamic in nature, in the sense that they may change as and when security requirements change. We have defined the possible violations to these policies. The security violations defined here form a mutually exclusive and exhaustive set. From these security violations we can determine the basic set of security requirements which are also mutually exclusive and exhaustive.

It is expected that, this basic set of security requirements will make analysis of security easier including in risk analyses, in threat analyses, in attack analyses, in monitoring security etc. Also, in the enforcement problem of security policy, this study will give a concrete basis regarding what to enforce to maintain security. Optimized set of

Mukherjee P. and Mazumdar C..
A Novel Model of Security Policies and Requirements.
DOI: 10.5220/0005239400730082
In *Proceedings of the 1st International Conference on Information Systems Security and Privacy* (ICISSP-2015), pages 73-82
ISBN: 978-989-758-081-9

controls can be designed as well, if a well-formulated set of security requirement can be obtained.

The rest of this paper is organized as follows. In section 2, we propose a model for defining an entity and its actions using the notions of process algebra. In section 3, a novel method of describing low-level security policies is proposed and the methodology of compliance with the policy is also given. Section 4 introduces the set of security requirements which are mutually exclusive and exhaustive. Analyses of different security violations are done from the view of hand-shake communication in section 5, with the identification of conjugate pairs of security requirements. Section 6 discusses some related research. Finally we conclude the paper by summarizing the findings.

2 PROCESS AND ACTION

An entity of an IT system can perform various internal and external actions. If watched externally, one can only observe the entity to carry out interactions with its environment. These interactions are nothing but information exchanges, and, can be considered to be instantaneous. The entity, performing these interactions, can be represented as a process, which can give out information and/or receive information (Milner, 1989).

In this work, an IT entity is defined as a Process. Actions by a process are of three types: input action, output action and empty action or no-action (φ). In process algebra, output actions ($\bar{a}$) are denoted by an overbar to distinguish them from input actions (a). (In this paper 'a' is used to denote general action if not mentioned otherwise). An input or output action (a) is represented by a triple (e, f, k), where, e is a process which is performing the action, $e \in E$, E: set of all the processes. f is another process in e's environment with which the process e is performing the action, $f \in E$ and $f \neq e$. And k is information exchanged by the action, $k \in K$, K: set of all the information units possible to be transmitted.

$$a = (e, f, k) \in (E \times E \times K) \quad (1)$$

3 SECURITY POLICIES AND RULES

A security policy is an imperative statement that defines a choice in the behavior of a system (Damianou et al., 2001). Security policies are of different types, access control policies, information flow control policies, obligation policies etc. Access control policies specify the legal and illegal accesses to information by some entity; information flow control policies specify legal and illegal propagation of information; and obligation policies specify the mandatory actions to execute.

But from the process algebraic notion of system modeling, all the entities, including the passive entities, are taken as processes. As shown in section 2, an entity or process can perform actions only. If we can control actions of entities in a system of processes, the access and information flow can also be controlled, rendering the system secure. In this paper, low-level policies are used to control the actions performed by entities/processes.

These low-level policies decide, whether a particular action by a process is allowable, or if a process is obliged to perform an action. These policies are dynamic in nature as they may change as per requirement, with time. A Security policy can be expressed by a set of rules. A rule can be of three types: **Permission –** execution of an action is granted, **Prohibition –** execution of an action is not granted, or **Obligation** – execution of an action is mandatory (Cuppens et al., 2005).

Policy specification can be done in three ways, **Closed Policy:** Only permissions are specified; an action is granted only if the permission to that action is explicitly specified. **Open Policy:** Only prohibitions are specified; an action is granted only if the prohibition to that action is not explicitly specified. **Hybrid Policy:** Both permissions and prohibitions are specified; an action is allowed depending on the way the specified permission and prohibition to that action are handled or prioritized and how unspecified actions are handled (Jajodia et al., 2001). Obligation can be introduced in any of these policy specifications as required.

Policy Conflict happens when different policies are imparting mutually dissimilar rules on the same action at the same time. In the closed policy, permissions and (if required) obligations are specified. Obligations are actually permissions to the actions, which must be executed. So there is no possibility of conflict in the case of closed policy. For open policies prohibitions and (if required) obligations are specified. In open policies, there may be conflict between obligation and prohibition rules. For the case of Hybrid policies, policy conflict can happen between permission and prohibition, and/or obligation and prohibition. Among all these policy specifications, if there is a constraint over the parallel execution of actions, then two or more

obligations can be in conflict with each other (Essaouini et al., 2013).

Resolution of policy conflicts can be done in different ways; two of them are**,**

No Conflict is Allowed: under this scheme of resolution, conflicts will be considered as inconsistency.

Giving Priority among Rules: under this scheme of resolution of policy conflict, rules are given an order of precedence. Four types of precedence rules are possible,

- Prohibition is given priority among all other rules;
- Obligation is given the highest priority, then Prohibition and then Permission;
- Permission is given the highest priority, then Prohibition and then Obligation;
- Prohibition is given the lowest priority among all other rules.

3.1 Permitted Actions

A process can perform various actions at a point in time, from a set PA of possible actions including input, output and empty (φ) actions, but one action at a time. But not all these actions are permitted by the applied policies. In subsection 3.1.4, a methodology have been given to find out the effective set of permitted actions (pA) for a process at a particular point in time under a set of security policies/rules and priorities among these rules.

Example: A server 'S' has two enlisted clients (A and B). S is attached with Local Area Network (LAN). S can perform various actions with attached LAN. Using the representation in Equation(1) in this example, e is server S and f is LAN, information exchanged k will vary, and depending on that, actions will change accordingly.

Type of possible actions include, input actions where k = req_A or req_B (Request from client A or B respectively); output actions where k = rep_A or rep_B (Reply to client A or B respectively) and no-action (φ) where k = null.

Therefore, set of all possible actions (PA) by server S are, {req_A, req_B, rep_A, rep_B, φ}.

Scenario 1: In scenario 1, server S has received req_A, 4 time units ago (with the assumption that one action required one time unit for its performance) and after that S has performed no-action (φ) for 4 times.

Scenario 2: In scenario 2, server S has performed rep_A.

This example will be used throughout the paper for explanation.

3.1.1 Prohibitions

Definition: If execution of an action is marked as illegal then that action becomes prohibited action or prohibition.

These prohibitions can be divided into two types:

- History based Prohibitions;
- Stand-alone Prohibitions;

History based Prohibitions: The actions which are prohibited at a certain point in time, depending on the history of actions by the process, are known as history based prohibitions.

$$(hA; a_x) \not\models P_i : a \in HQ_i \quad (2)$$

a is the prohibited action and hA is the history of actions by the process till the concerned point in time. Execution of 'a' is denoted by a_x. (hA; a_x) means execution of a after hA. P_i is the policy statement. HQ_i is the set of history based prohibitions at the certain point in time, for a certain process and for a certain policy P_i.

History based prohibitions can be derived from policies like,

- Separation of duty policy;

Performing a duty earlier, prohibits a process (may be human being) from performing another duty which is in conflict from the point of view of separation of duty policy.

- Non-generation of receipt by beneficiary before receiving money;

If in its history of actions the beneficiary process has not received money, then by this policy, beneficiary is prohibited to generate the receipt.

Stand-alone Prohibitions: The actions which are prohibited at a particular point in time, but the prohibitions do not depend on the history of actions are known as stand-alone prohibitions.

$$a_x \not\models P_i : a \in SQ_i \quad (3)$$

SQ_i is the set of stand-alone prohibitions at a certain point in time for a certain process for a certain policy P_i.

Stand-alone prohibitions can be derived from policies like,

- A process is always barred to execute an action;

An employee process is always prohibited to leak internal information of company.

- A process is barred to execute an action at present time;

A software process is prohibited to send page request to a social networking site within office

hours.

- A process is barred to execute an action from the present place;

A process (person) is prohibited to make a call while inside a bank ATM.

In a hybrid policy specification, the set of prohibitions Q for a particular process, for all the policy generated prohibitions (if there are any) at the concerned point in time, is:

$$Q = (\cup HQ_i) \cup (\cup SQ_i) \quad (4)$$

In the given example suppose, server S has two policies from which Prohibitions can be derived,

- **1.** Can not perform req_X when last action in history (not considering no-actions) is req_Y. (where, X may or may not be equal to Y)
- **2.** Can not perform rep_X when last action in history (not considering no-actions) is not req_X.

Both of policies 1 and 2 will give history based prohibitions. When these policies are applied to scenario 1 as given earlier, then following sets of prohibitions will come up,

- From the first policy, req_A and req_B are prohibited to be performed by the server in the given scenario. So, $HQ_1 = \{req_A, req_B\}$
- From the second policy, rep_B is prohibited. So, $HQ_2 = \{rep_B\}$.

Therefore, in the given scenario 1 under policies 1 and 2, $Q = \{HQ_1 \cup HQ_2\}$

Or, $Q = \{req_A, req_B, rep_B\}$

When policies 1 and 2 are applied to scenario 2 as given before, then following sets of prohibitions will come up,

- From the first policy, nothing is prohibited. So, $HQ_1 = \{\ \}$
- From the second policy, rep_A and rep_B are prohibited. So, $HQ_2 = \{rep_A, rep_B\}$.

Therefore, in the given scenario 2 under policies 1 and 2, $Q = \{HQ_1 \cup HQ_2\}$

Or, $Q = \{rep_A, rep_B\}$

3.1.2 Permissions

Definition: If execution of an action is marked as legal, then that action becomes permitted action or permission.

These permissions can be divided into two types:

- History based Permissions;
- Stand-alone Permissions;

History based Permissions: The actions which are permitted at a particular point in time, depending on the history of actions by the process, are known as history based permissions.

$$(hA; a_x) \vDash P_i : a \in HP_i \quad (5)$$

a is the permitted action, hA is the history of actions by the process till the concerned point in time and P_i is the policy statement. HP_i is the set of history based permissions at a certain point in time, for a certain process and for a certain policy P_i.

History based permissions can be derived from policies like,

- Things permitted to do after getting some license;

A process (person) is permitted to run a car by himself after getting driving license.

Stand-alone Permissions: The actions which are permitted at a particular point in time, but the permissions do not depend on the history of actions are known as stand-alone permissions.

$$a_x \vDash P_i : a \in SP_i \quad (6)$$

SP_i is the set of stand-alone permissions at a certain point in time for a certain process for a certain policy P_i.

Stand-alone permissions can be derived from policies like,

- A process is granted to execute an action at present time;

A software process is permitted to send page request to a social networking site after office hours.

- A process is granted to execute an action from the present place;

A process (person) is permitted to make a call while in its home.

Assuming a hybrid policy specification, set of permissions R for a particular process, for all the policy generated permissions (if there are any) at the concerned point in time, is:

$$R = (\cup HP_i) \cup (\cup SP_i) \quad (7)$$

In the given example suppose, server S has three policies from which Permissions can be derived,

- **3.** S can perform input req_X.
- **4.** S can perform output rep_X.
- **5.** S can perform φ.

All three policies will give Stand-alone Permissions. When policies 3, 4 and 5 are applied to scenario 1 as given earlier, then following sets of permissions will come up,

- From the policy 3, req_A and req_B are permitted. So, $SP_3 = \{req_A, req_B\}$
- From the policy 4, rep_A and rep_B are permitted. So, $SP_4 = \{rep_A, rep_B\}$.

- From the policy 5, φ is permitted. So, $SP_5 = \{\varphi\}$.

Therefore, in the given scenario under policies 3, 4 and 5, $R = \{SP_3 \cup SP_4 \cup SP_5\}$

Or, $R = \{req_A, req_B, rep_A, rep_B, \varphi\}$

When policies 3, 4 and 5 are applied to scenario 2 as given earlier, then same sets of permissions will come up.

Therefore, in the given scenario under policies 3, 4 and 5, $R = \{req_A, req_B, rep_A, rep_B, \varphi\}$.

3.1.3 Obligations

Definition: If among a set of actions, execution of any one action is compulsory, then that set of action is known as obligatory actions or obligations. [As per assumption only one action is possible to be executed by a process at a time.]

These obligations can be divided into two types:

- Dead-lined Obligations;
- Point Obligations;

Dead-lined Obligations: An action, which has a deadline for its execution, can be executed at any time within the deadline. If at a particular point in time, without executing any action from a particular set of dead-lined actions, it is impossible to meet the deadline for all the dead-lined actions, then dead-lined actions in that particular set are called Dead-lined obligations.

$$(\forall a\ (\neg a_x)) \nvDash P : a \in DO \tag{8}$$

DO is the set of dead-lined obligations at a certain point in time for a certain process for the set of all applicable policies P.

Dead-lined Obligations can be derived from policies like:

- Within 7:00 PM delivery of pizza at two places separated by 5 minutes journey;

The pizza delivery process (delivery-man) is obligated to reach any one of the two places latest by 6:55 PM.

- Updating antivirus once a month;

Point Obligation: An action which must be executed only at a particular point in time is termed as point obligation.

$$\neg a_x \nvDash P : a = PO \tag{9}$$

PO is the point obligation at a certain point in time for a certain process for the set of all applicable policies P.

If, for just meeting the deadline of one action at the concerned point in time, that action itself have to be executed; then that action is also known as point obligation. So point obligation can be seen as a special case of dead-lined obligations.

Point Obligations can be derived from policies like,

- Time out;

In a software process, if a reply is not received within 5 seconds, then it is obligatory to stop the process.

- Monitor alarm;

Whenever an antivirus software process detects a virus definition it is obligatory for it to generate an alert.

Set of obligations for a process at the present moment is: (null set is represented by < > in this paper)

$$\begin{aligned} O &= PO, \quad \text{when } PO \neq <> \text{ and } DO = <> \\ O &= DO, \quad \text{when } PO = <> \text{ and } DO \neq <> \end{aligned} \tag{10}$$

If PO and DO both are non-null then, there will be conflict among the obligation policies. If there are two or more point obligations or, two or more sets of dead-lined obligations then also, there is conflict in obligation policies.

In the given example suppose, server S has two policies from which Obligations can be derived,

- **6.** Server S is obliged to perform output action rep_Y, within 5 time units of performing the input action of req_X (where, X may or may not equal to Y).
- **7.** Server S is obliged to perform req_X after performing rep_Y (where, X may or may not equal to Y).

These policies will generate Dead-lined obligation. When policies 6 and 7 are applied to scenario 1 as given earlier, then following set of obligations will come up,

- From the policies 6 and 7, one of rep_A or rep_B is obligated. So, $DO = \{rep_A, rep_B\}$

Therefore, in the given scenario 1 under policy 6 and 7, O = DO

Or, $O = \{rep_A, rep_B\}$

When policies 6 and 7 are applied to scenario 2 as given earlier, then following set of obligations will come up,

- From the policies 6 and 7, one of req_A or req_B is obligated. So, $DO = \{req_A, req_B\}$

Therefore, in the given scenario 2 under policy 6 and 7, O = DO

Or, $O = \{req_A, req_B\}$

3.1.4 Effective Set of Permitted Actions

Effective set of permitted actions are a set of actions which can be performed at a particular point in time

by a certain process fulfilling all the security policies.

From our assumed scenario of hybrid policies with prohibitions given the highest priority among all rules, the effective set of permitted actions can be found out as follows, (with further assumption of default prohibition on unspecified actions)

$$\begin{aligned} pA &= O \setminus Q \quad \text{if } O \neq <> \\ pA &= R \setminus Q \quad \text{if } O = <> \end{aligned} \tag{11}$$

For other cases, where permission or obligations get priorities, pA can be found out with some simple changes in Equation(11).

In the given example, if all the policies from 1 to 7 are applied in the given scenario 1, the permitted set of actions pA is given by the Equation(11) as,

$pA = O \setminus Q$ as, $O \neq <>$.

Or, $pA = \{rep_A, rep_B\} \setminus \{req_A, req_B, rep_B\}$

Or, $pA = \{rep_A\}$.

In the given example, if all the policies from 1 to 7 are applied in the given scenario 2, the permitted set of actions pA is given by the Equation(11) as,

$pA = O \setminus Q$ as, $O \neq <>$.

Or, $pA = \{req_A, req_B\} \setminus \{rep_A, rep_B\}$

Or, $pA = \{req_A, req_B\}$.

3.2 Security Policy Compliance

To comply with all the policies imposed on a process at a particular point in time, the process needs to maintain the following;

$$a \in pA : a_x \vDash P \tag{12}$$

a_x being the execution of action a. P is set of all the applied policies.

4 SECURITY VIOLATIONS

If the executed action by a process is not in the set of the effective permissions (pA), then it causes security violation. Therefore, non-compliance to security policies infers security violation. V is the violation statement.

$$b_x : (b \notin pA) \Rightarrow V \tag{13}$$

In more details, violation can happen in the following ways,

$$\begin{gathered} (b \neq \varphi) \wedge (\varphi \in pA) : (b_x \nvDash P) \Rightarrow V \\ (b \neq \varphi) \wedge (\varphi \notin pA) : (b_x \nvDash P) \Rightarrow V \\ (b \neq \varphi) \wedge (\varphi \notin pA) : \forall a\, (((a \in pA) \wedge ((\forall a\, (\neg a_x)) \nvDash P)) \Rightarrow V) \\ (b = \varphi) : \forall a\, (((a \in pA) \wedge ((\forall a\, (\neg ax)) \nvDash P)) \Rightarrow V) \end{gathered} \tag{14}$$

If the executed action (b) by a process is not in the set of the effective permissions (pA), then violation can happen in the ways shown above. If the executed action is not no-action (φ) and no-action belongs to pA then, violation will happen only for executing b, which does not conform to the security policy. If no-action (φ) does not belong to pA then, violation will also happen for non-execution of all the actions in pA, anyone of which should be executed to fulfil the policy. If executed action b is the no-action (φ) then violation will only happen for non-execution of all the actions belonging to pA.

All these violations given above are behavioural integrity violations by the process (Biba, 1977) (Mayfield et al., 1991) or violations of safety properties (Alpern and Schneider, 1985).

Based on the type of violating action, the following 4 types of security violations can be conceived of.

4.1 Availability Violation

If a process is not complying with security policy by not performing any of the permitted actions (not including no-action), then non performance of output action ($\bar{o}$), which belongs to the set of permitted actions, from the point of view of environment, is reflected as availability violation (AV).

$$\begin{gathered} b_x : (b \notin pA) \wedge (\varphi \notin pA) \wedge (a \in pA): \\ ((\bar{o} \in pA) \wedge ((\forall a\, (\neg a_x)) \nvDash P)) \Rightarrow AV \end{gathered} \tag{15}$$

If action b is executed which does not belong to pA and no-action (φ) also does not belong to pA, then non-execution of output action $\bar{o}$, which belongs to pA, causes availability violation, by not performing the action expected by the environment.

4.2 Confidentiality Violation

If a process is not complying with security policy by performing output action ($\bar{o}$), from the point of view of environment this is reflected as confidentiality violation (CV).

$$\begin{gathered} \bar{o}_x : \bar{o} \notin pA: \\ (\bar{o}_x \nvDash P) \Rightarrow CV \end{gathered} \tag{16}$$

If output action $\bar{o}$ is executed which does not belong to pA, then execution of that output action reveals information to environment and causes confidentiality violation.

4.3 Completeness Violation

If a process is not complying with security policy by

not performing any of the permitted actions (not including no-action), then non performance of input action (i), which belongs to the set of permitted actions, from the point of view of environment, is reflected as completeness violation (CoV).

$$b_x : (b \notin pA) \wedge (\varphi \notin pA) \wedge (a \in pA) : ((i \in pA) \wedge ((\forall a (\neg a_x)) \not\models P)) \Rightarrow CoV \quad (17)$$

If action b is executed which does not belong to pA and no-action (φ) also does not belong to pA, then non-execution of input action i, which belongs to pA, means non-acceptance of desired changes in the entity, causes completeness violation.

4.4 Accuracy Violation

If a process is not complying with security policy by performing input action (i), from the point of view of environment this is reflected as accuracy violation (AcV).

$$i_x : i \notin pA : (i_x \not\models P) \Rightarrow AcV \quad (18)$$

If input action i is executed which does not belong to pA, then execution of that input action causes undesired changes in the entity, causing accuracy violation.

In the given example suppose the server S performs req_B in scenario 1.

Then, as the server is not complying with security policies by performing input action req_B not in pA, so Accuracy violation has occurred.

And, the server is not complying with security policies by not performing any of the actions in pA (which does not include φ). So for non-execution of rep_A Availability violation has occurred.

In the given example suppose the server S performs rep_B, in the given scenario 2.

Then, as the server is not complying with security policies by performing output action rep_B not in pA, so Confidentiality violation has occurred.

And, the server is not complying with security policies by not performing any of the actions in pA (which does not include φ). So for non-execution of req_A Completeness violation has occurred and for non-execution of req_B another Completeness violation has occurred.

4.5 Violation Quadrant

These four violations can be represented in the form of quadrant structure, as shown in Figure 1,

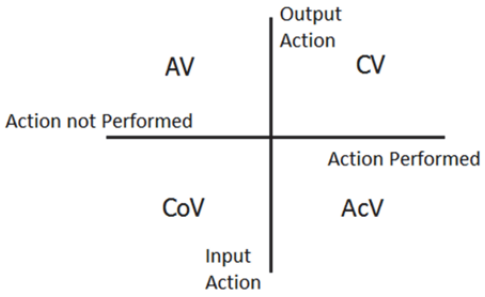

Figure 1: Violation Quadrant.

The actions of a process may be within the effective set of permitted actions or not, because of the application of the security policies as shown earlier. By not complying with security policies a process creates security violation. Security violations covered in subsection 4.1 to 4.4, are the only possible violations and they form an exhaustive set. Since, these violations are semantically non-overlapping they are mutually exclusive.

Corresponding set of mutually exclusive and exhaustive Security Requirements are termed as follows,

- Availability requirement;
- Confidentiality requirement;
- Completeness requirement;
- Accuracy requirement.

Maintenance of the above security requirements is expected to be the necessary and sufficient condition to make a system of processes secure.

5 HAND-SHAKE COMMUNICATION

In the process algebraic framework proposed by Milner (Milner, 1989), communication between two processes is simultaneous actions by both the involved processes. This is known as handshake communication. From this notion we have found out relationship among the security requirements proposed in the subsection 4.5.

We have represented Handshake communication as the ensemble of two points,

1> One process sends information or performs an output action ($\bar{a}$) and the recipient process in its environment has not received that information or has not performed the corresponding input action (a) – is not possible. Thus the following is true:

$$\neg((\bar{a}) \wedge \neg(a)) \text{ or, } (\bar{a}) \rightarrow (a) \quad (19)$$

2> One process performs a receive action or input action (a) and the sender process in its environment has not performed the corresponding output action ($\bar{a}$) – is not possible. Thus the following is true:

$$\begin{array}{c} \text{or, } \neg((a) \wedge \neg(\bar{a})) \\ \text{or, } (a) \rightarrow (\bar{a}) \end{array} \quad (20)$$

Therefore, hand-shake communication can be expressed as,

$$(\bar{a}) \leftrightarrow (a) \quad (21)$$

5.1 Induced Security Violations

Interactions of processes have to be mutual every time as per hand-shake communication. Due to this mutuality of interactions security violation of one process can induce security violation on another process; this phenomenon brings out different existing security aspects for processes as shown below.

One process performed availability violation by not performing output action ($\bar{a}$), induces, another process, to perform completeness violation by not performing the corresponding input action (a).

$$\begin{array}{c} (\text{AV at sender}) \wedge ((\bar{a}) \leftrightarrow (a)) \\ \rightarrow \ (\text{CoV at receiver}) \end{array} \quad (22)$$

If the receiving process performs completeness violation by not performing input action (a) then, by the laws of hand-shake communication the sending process performs availability violation by not performing output action ($\bar{a}$).

$$\begin{array}{c} (\text{CoV at receiver}) \wedge ((\bar{a}) \leftrightarrow (a)) \\ \rightarrow \ (\text{AV at sender}) \end{array} \quad (23)$$

Similarly confidentiality violation done at sender process will induce accuracy violation at the receiver process and vice versa.

$$\begin{array}{c} (\text{CV at sender}) \wedge ((\bar{a}) \leftrightarrow (a)) \\ \rightarrow \ (\text{AcV at receiver}) \\ (\text{AcV at receiver}) \wedge ((\bar{a}) \leftrightarrow (a)) \\ \rightarrow \ (\text{CV at sender}) \end{array} \quad (24)$$

Accuracy Violation at receiver violating the sender process' Confidentiality is actually the depiction of covert channel, which is coming from the interrelation of these violations.

After the above analysis we can deduce that, Availability – Completeness and Confidentiality – Accuracy form two conjugate pairs of security requirements. When a particular requirement is violated in one process then there must be another process where the paired requirement gets violated, due to hand-shake communication.

In the given example the server S and the attached LAN are two processes performing Handshake communication.

Then, as in scenario 1 the server S performed Availability violation by not performing output action rep_A, induces the attached LAN to perform Completeness violation by not performing the corresponding input action rep_A (by LAN).

6 RELATED WORK

In (Alpern and Schneider, 1985) authors show every property can be described as the intersection of safety and liveness properties. (McLean, 1994) disproves this argument and shows only property of traces can be described by Alpern-Schneider framework. In this work a partial ordering of possibilistic security properties is done. Separability, Non-inference, Generalized Noninference, Generalized Noninterference are partially ordered by their relative strength from the notion of proposed selective interleaving function. Zakinthinos and Lee have extended the partial ordering with some other properties like, Perfect security property (PSP) and Output Non-deducibility (Zakinthinos and Lee, 1998).

In (McCullough, 1987) author has compared different multi level security requirements like, Simple security property & *-security property by Bell Lapadula, Non-interference property by Goguen-Meseguer and Deducibility secure property by Sutherland; and found large similarities in them.

In (Clarkson and Schneider, 2010(a)) authors have quantified the integrity by using information theoretic approach. The work has proposed two security violations, contamination and suppression, and their measuring techniques. Different information flow security properties like, confidentiality, integrity and availability has been compared from the point of view of Biba's duality. Authors have also formulated security policies with the conception of hyperproperties in (Clarkson and Schneider, 2010(b)). Where, every hyperproperty falls at the intersection of safety hyperproperty and liveness hyperproperty.

But none of the above works have tried to find out a set of mutually exclusive and exhaustive security requirements, which was the motivation behind writing the present paper.

There is rich literature dealing with enforcement of security, (Schneider, 2000), (Ligatti et al., 2005),

(Khoury and Tawbi, 2012), (Basin et. al, 2013). How these enforcement mechanisms can be applied to this work and finding out the set of enforceable actions will set a possible future direction of our research.

7 CONCLUSIONS

In this paper we have derived the set of mutually exclusive and exhaustive security requirements. To find that out basic notion of process algebra has been taken. We proposed novel method for describing low level security policies, dealing with the legality of actions by a process. Basic security rules of the policies like Permission, Prohibition and Obligation are explored. Some examples of formulation of high level security policies in terms of low level policies are given in the appendix. The mechanism to get the effective set of permitted actions complying with all the applicable security policies at a point in time is mentioned in this paper. Non-compliance to these low-level policies is taken as security violation. We have tried to find out all the possible security violations. The requirements corresponding to these security violations are expected to be necessary and sufficient for a system of processes to maintain security. This paper also tried to find the dependency of one security requirement on another and have found the conjugate pairs among them. The approach has been illustrated by a running example of interactions between a server and the attached network.

Our future work includes how this study can be applied to different cases of security analyses, like in risk, threat or attack analysis. With mutually exclusive and exhaustive set of security requirements in hand, it is expected to get a better formal view of security analysis. How to monitor or enforce security requirements is another possible area of research. Designing the set of security controls for a system with optimal usage of resources is yet another future challenge. It seems, security policy generation may be done in more formal way, by using this set of mutually exclusive and exhaustive security requirements, which needs further attention.

ACKNOWLEDGEMENTS

This research was partially supported by grants allocated by the Department of Electronics and Information Technology, Govt. of India. We wish to thank anonymous reviewers, there comments come extremely helpful to articulate the ideas and shape the illustrations.

REFERENCES

Milner, R., 1989. *Communication and Concurrency*. Prentice-Hall International.

Damianou, N., Dulay, N., Lupu, E., Sloman, M., 2001. the Ponder Policy Specification Language. in *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*. Springer-Verlag.

Cuppens, F., Cuppens-Boulahia, N., Sans, T., 2005. Nomad: A Security Model with Non Atomic Actions and Deadlines. in *the Computer Security Foundations Workshop (CSFW)*.

Jajodia, S., Samarati, P., Sapino, M. L., Subrahmanian, V. S., 2001. Flexible Support for Multiple Access Control Policies. in *ACM Transactions on Database Systems (TODS), V.26 N.2, P.214-260*.

Mayfield, T., Roskos, J. E., Welke, S. R., Boone, J. M., Mcdonald, C. W., 1991. Integrity in Automated Information Systems. *C Technical Report 79-91, Library No. S-237,254 (IDA PAPER P-2316)*.

Biba, K. J., 1977. Integrity Considerations for Secure Computer Systems. *Mitre TR-3153*, Mitre Corporation, Bedford, MA.

Alpern, B., Schneider, F. B., 1985. Defining Liveness. in *Information Processing Letters, 21(4):181-185*.

Mclean, J., 1994. a Generai Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. in *Proceedings of the 1994 IEEE Symposium on Security and Privacy, Pages 79-93*. IEEE Press.

Zakinthinos, A., Lee, E. S., 1998. A General Theory of Security Properties and Secure Composition. in *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy.* IEEE Press.

Mccullough, D., 1987. Specifications for Multi-Level Security and a Hook-up Property. in *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy.* IEEE Press.

Clarkson, M. R., Schneider, F. B., 2010(a). Quantification of Integrity. in *Proc. 23nd IEEE Computer Security Foundations Symposium (CSF '10), Pp. 28–43.*

Clarkson, M. R., Schneider, F. B., 2010(B). Hyperproperties. *Journal of Computer Security, 18(6):1157–1210.*

Schneider, F. B., 2000. Enforceable Security Policies. *ACM Trans. on Information and System Security. 3, 1.*

Ligatti, J., Bauer, L., Walker, D., 2005. Edit Automata: Enforcement Mechanisms for Run-Time Security Policies. *International Journal of Information Security 4(1-2), 2–16.*

Khoury, R., Tawbi, N., 2012. Which Security Policies Are Enforceable by Runtime Monitors? a Survey. *Computer Science Review 6(1), 27–45*.

Basin, D., Jugé, V., Klaedtke, F., Z˘alinescu, E., 2013. Enforceable Security Policies Revisited. *ACM Trans. on Information and System Security*. 16, 1.

Essaouini, N.,Cuppens, F., Cuppens-Boulahia, N., Kalam, a.a.E., 2013. Conflict Management in Obligation with Deadline Policies. in *Proceedings of the Eighth International Conference on Availability, Reliability and Security. (IEEE Computer Society).*

APPENDIX

EXAMPLES OF POLICY FORMULATION

Using the low level policy representation formulated in section 3, high level policies can be explained. In this section, this point is illustrated by some examples.

Example of **Access control policy** rule:

- A has read access to B

Let, in this example A and B be two processes, performing handshake communication. In that scenario, this Access Control Policy rule is mapped to the set of low level policies as follows,

- A has permission of sending “Read Request” via output action to B.
- A has the permission of receiving “Content” via input action from B.

On the other hand,

- B has the permission of receiving “Read Request” via input action from A.
- B has the permission of sending “Content” via output action to A.

Some other policies like, **Principle of Least Privilege** states,

- A subject should get the minimal privileges required, to accomplish a given task, by the authority.

In the low level policy set this policy can be mapped as,

- The authority is prohibited to output the grant of extra privilege, than required, to a subject.
- A subject is prohibited to input the grant of extra privilege, than required, from the authority.

On the other hand **Information flow control policies** which are violating safety properties can be explained by this basic set of rules as shown earlier.