https://doi.org/10.26552/com.C.2009.3.5-10

Jaroslav Janacek *

A SECURITY MODEL FOR AN OPERATING SYSTEM FOR SECURITY-CRITICAL APPLICATIONS IN SMALL OFFICE AND HOME ENVIRONMENT

Personal computers are often used in small office and home environment for different purposes ranging from general web browsing and e mail processing to processing data that are sensitive regarding their confidentiality and/or integrity. Common operating systems do not provide sufficient protection. We present a security model combining the well known benefits of mandatory access control in classified information processing systems with the typical home and small office computer use. We use a simple two-dimensional data classification scheme and present a security model with provable properties that significantly reduces the risks of confidentiality and/or integrity protection violation.

1. Introduction

During the past few years we have been observing a significant increase in the number of people who use computers to perform tasks where security is important. Typical such applications (denoted *security-critical* applications in this paper) that are of interest to general public and can be expected to be used on home and office computers, include Internet banking, e-government applications, electronic signature creation and verification applications. Organizations use information systems to store and process confidential business data and personal data. Unauthorized access to information stored or processed by all of the mentioned applications (and many others) can often cause a substantial loss to the affected person or organization. It is usually the user's responsibility to protect the sensitive data. However, common users are not information security experts and can only follow some guidelines given to them. Even that is usually possible only if the guidelines are simple enough.

While a larger organization can dedicate some computers to security-critical applications and protect them against unauthorized access, modification or software installation, it can hardly be expected in home or small office environments. In such environments, the same computer is usually used for many different purposes – such as web browsing, e mail processing, running programs from untrustworthy sources, etc. – alongside running security-critical applications. We will discuss the typical examples of the applications used in these environments and consider the data they use in terms of security requirements in the next section.

We will concentrate on two security aspects – *confidentiality* and *integrity*. The goal of confidentiality protection is to prevent unauthorized subjects from obtaining the protected information. It includes the protection of stored information as well as the protection of information being transferred (e.g. by means of a computer network). The goal of integrity protection is to prevent unauthorized modifications of the protected information from taking place, or, if they cannot be prevented, to provide means of detection of such modifications.

Both confidentiality and integrity can be protected using different security mechanisms. We will concentrate on *access control* in this paper. Other popular security mechanisms used to protect confidentiality and integrity include cryptography – encryption, message authentication codes and digital signatures.

Applications processes can directly manipulate data in their memory. In order to access other data, processes have to use the services of an operating system. This enables the operating system to control access to the data and to enforce a security policy based on various attributes associated with the application process and with the data object, and thus makes the operating system a good place where to deal with security.

Currently common operating systems used in the target environment, such as the Professional edition of Microsoft Windows XP, Vista or various distributions of Linux, provide access control features to protect the data stored in files within the most often used native filesystems. The access control provided by the mentioned operating systems is based on the following principles:

- Every process – *subject* performs its actions on behalf of a user.
- Every filesystem object (i.e. a file, a directory, ...) is *owned* by a user (or a group of users) – its *owner*.
- The owner of an object can specify a discretionary access control list for the object. Each entry in the access control list specifies the permitted *operations* for processes running on behalf of an identified user (or a group of users).

* **Jaroslav Janacek**
Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia,
E-mail: janacek@dcs.fmph.uniba.sk

The set of operations depends on the operating system – Linux uses three basic operations (read, write, execute/use directory) while Windows uses more finer-grained operations. The common properties of this type of access control are:

- it can be used to protect data against access and/or modification by processes of a different user,
- but it cannot prevent a process from accessing/modifying data owned by the user that the process runs on behalf of.

If a process can communicate with an external system (e.g. the Internet), it can transmit the data from any file readable by the user the process runs on behalf of. It can also modify any file writeable by the user. The behaviour of certain types of applications can be significantly influenced, as will be discussed later, by external entities. This opens a way for external attackers to gain unauthorized access to the user's data.

We present a security model designed to protect confidentiality and integrity of data classified in terms of the confidentiality and integrity protection requirements in the home and small office environments.

2. Security needs of the typical data in the small office and home environment

We will consider the typical classes of applications used in the target environment in this section, and we will specify the security needs, in terms of confidentiality and integrity, of the data they process.

2.1. Examples of applications

General Internet access

One of the typical classes of applications is the class of the applications used to access Internet resources. It includes web-browsers, e-mail clients, and various other communication systems (e.g. ICQ, IRC, Voice over IP, video-conferencing systems, ...). The common feature of these applications is that they communicate with external systems that cannot be trusted to protect the confidentiality of the data transferred to them, nor can they be trusted not to send *malicious data* back to our system. By the term *malicious data* we mean data that have been prepared in a way to abuse a weakness of the application receiving them. Applications often contain programming errors (bugs) that can be abused by providing specially prepared data. These errors can often be abused to execute arbitrary code as if it were a part of the application, i.e. with the same privileges as the application itself. Therefore, even if the application itself is believed not do anything unwanted, its behaviour may be changed, by processing malicious data, in an unpredictable way.

It has to be assumed that the applications of this class:

- export any data available to them to the Internet,
- import (potentially) malicious data from the Internet, and therefore, their output has to be considered (potentially) malicious too,
- may become malicious by processing the malicious input.

Malicious applications

A special class of applications is the class of malicious applications. These are applications that have been intentionally programmed to perform malicious activities. The typical examples are computer viruses, worms, Trojan horses and other kinds of so called malware. They can be downloaded from the Internet by the user, received as an attachment of an e-mail message, or a vulnerable application may be turned into a malicious one by processing malicious data. The user is usually unaware of the fact that a particular application is malicious.

It has to be assumed that the malicious applications do anything not prevented by the operating system or the environment of the computer (e.g. a network firewall).

Local applications

The class of local applications contains the applications that are used to process data stored in a local filesystem. These applications generally do not need network access to perform their tasks. The typical examples are text processors, spreadsheets, presentation software, graphic editors, ...

Local applications are used to process data with varying requirements regarding the confidentiality and integrity protection. If they process malicious data, they may become malicious due to programming errors.

Sensitive web access

Web browsers are often used to access remote services that process data requiring confidentiality and/or integrity protection. A typical example is an Internet-banking system. It provides access to financial information; it allows the user to submit transaction orders to the bank, etc. It also processes authentication data (e.g. passwords). All such data may be considered confidential by the user, and therefore, are to be adequately protected. The confidentiality and the integrity of the data during their transmission is usually protected by means of cryptography. Cryptography is usually also used to provide authentication of the remote system. But the data is also to be protected while stored in the memory or in a file on the local computer. Consider an instance of a web browser used for general Internet access. It may have processed some malicious data, and therefore, it may have become a malicious application exporting everything to an attacker. If the instance of the web browser is later used to access an Internet banking system, all the confidential information may leak.

Digital signature and data encryption/decryption

Digital signature creation applications, as well as data decryption applications need access to a private key. Digital signature verification applications, as well as data encryption applications need access to a public key.

The private key is a very sensitive piece of information the confidentiality of which has to be protected. The integrity of the private key has to be protected as well because its modification can lead not only to the loss of ability to create correct digital signatures or to decrypt data, but also to the leak of information that is sufficient to compute the corresponding private key in certain cases.

The public key requires no confidentiality protection, but it does require integrity protection. If attackers were able to modify the public key used to verify a digital signature, they would be able to create a digitally signed document that would pass the signature verification process. In the case of encryption, if the public key were modified by an attacker, the attacker would be able to decrypt the encrypted data instead of the intended receiver.

The encrypted output of a data encryption application may be transmitted via communication channels that do not provide confidentiality protection even if the confidentiality of the original data is to be protected. The output of a data decryption application may also require confidentiality protection.

2.2. Data classification scheme

We can conclude, from the previous subsection, that the need of confidentiality protection and the need of integrity protection are independent on each other. Some data need integrity protection while they can be disclosed to the public, some data need both, some need none. We will, therefore, use a two-dimensional classification scheme for the data consisting of the *confidentiality* level and the *integrity* level.

We will use three confidentiality levels:
- 0 – public data,
- 1 – normal data (C-normal),
- 2 – sensitive data (C-sensitive).

The public data require no confidentiality protection. They may be freely transmitted via communication channels and/or to remote systems that provide no confidentiality protection. An example of public data is the data downloaded from public Internet.

The normal data are to be protected by means of discretionary access control against unauthorized reading by other users than the owner of the data.

The C sensitive data are the data that their owner (a user) wishes to remain unreadable to the others regardless of the software the user uses, and even if the users makes some mistakes (such as setting wrong access rights for discretionary access control). Examples of C sensitive data are private and secret keys, passwords for Internet banking, etc.

We will also use three integrity levels:
- 0 – potentially malicious data,
- 1 – normal data (I-normal),
- 2 – sensitive data (I-sensitive).

The requirement of the integrity protection of data is tightly coupled to the trustworthiness of the data. The trustworthiness of data can be thought of as a metric of how reliable the data are. If some data can be modified by anyone, they cannot be trusted not to contain wrong or malicious information. If some data are to be relied on, their integrity has to be protected.

The potentially malicious data require no integrity protection, and can neither be trusted to contain valid information, nor can be trusted not to contain malicious content.

The normal data are to be protected by means of discretionary access control against unauthorized modification by other users than the owner of the data.

The I sensitive data are the data that their owner wishes to remain unmodified by the others regardless of the software the user uses, and even if the user makes some mistakes. The I sensitive data are to be modifiable only under special conditions upon their owner's request. A special category of I sensitive data is the category of the shared system files such as the programs, the libraries, various system-wide configuration files, the user database, ... Some of these files may be modifiable by the designated system administrator, some of them should be even more restricted.

3. Security model

A common approach to ensuring the confidentiality and/or the integrity of information in systems that deal with data classified into several confidentiality/integrity levels, is to define an information flow policy, and then to enforce the policy. In order to enforce an information flow policy, subjects are divided into two categories – trusted and untrusted. A trusted subject is a subject that is trusted to enforce the information flow policy (with exceptions) by itself; an untrusted subject is a subject that is not trusted to enforce the policy by itself, and therefore the policy has to be enforced on the subject's operations by the system.

A typical information flow policy protecting confidentiality (e.g. one based on Bell-LaPadula model [1]) states that a subject operating at a confidentiality level C_S may only read from an object with a confidentiality level C_O if $C_S \geq C_O$, and may only write to an object with a confidentiality level C_O if $C_S \leq C_O$. If a subject is to be able to read from a more confidential object, and to write to a less confidential object, it has to be a trusted subject.

A typical information flow policy protecting integrity (e.g. one based on Biba model [2]) states that a subject operating at an integrity level I_S may only read from an object with an integrity level I_O if $I_S \leq I_O$, and may only write to an object with an integrity level I_O if $I_S \geq I_O$. Only a trusted subject can read from an object with a lower integrity level, and write to an object with a higher integrity level.

The problem with the division of subjects into the two categories is that it would lead to the need of too many trusted subjects in the home and office environment. Considering the examples given in the previous section, many of the identified applications would have to be trusted.
We will divide the subjects into three categories:
- untrusted subjects,
- partially trusted subjects, and
- trusted subjects.

An *untrusted* subject is a subject that is not trusted to enforce the information flow policy. It is assumed to perform any operations on any objects unless it is prevented from doing so by the operating system.

A *trusted* subject is a subject that is trusted to enforce the information flow policy by itself. A trusted subject may be used to perform tasks that require violation of the policy under conditions that are verified by the trusted subject. A trusted subject can, therefore, be used to implement an exception to the policy.

A *partially trusted* subject is a subject that is trusted to enforce the information flow policy regarding a specific set of objects, but not trusted to enforce the information flow policy regarding any other objects. In other words, a trusted subject is

- trusted not to transfer information from a defined set of objects (designated inputs) at a higher confidentiality level to a defined set of objects (designated outputs) at a lower confidentiality level in a way other than the intended one, and
- trusted not to transfer information from a defined set of objects (designated inputs) at a lower integrity level to a defined set of objects (designated outputs) at a higher integrity level in a way other than the intended one, but
- not trusted not to transfer information between any other objects.

The sets of designated inputs and outputs regarding confidentiality are distinct from the sets regarding integrity. Any of the sets may be empty. A partially trusted subject, like a trusted one, can be used to implement an exception to the policy, because it can violate the policy (and it is trusted to do it only in an intended way).

The most important difference between trusted and partially trusted subjects is in the level of trust. While trusted subjects are completely trusted to behave correctly, partially trusted subjects are only trusted not to abuse the possibility of the information flow violating the policy between a defined set of input objects and a defined set of output objects.

The presented version of the model is limited to two basic operations: read and write. It can be conservatively extended to support other operations on objects, as well as operations on subjects, however.

3.1. Information flow policy objectives

We will first specify the policy objectives in an informal way, and then we will define the policy formally.

In accordance with the classification of objects, the information flow policy has the following objectives:

1. Prevent reading of C-sensitive objects by subjects of other users than the owner of the object.
2. Prevent modification of I-sensitive objects by subjects of other users than the owner of the object.
3. Prevent information passing from objects with a higher confidentiality level to objects with a lower confidentiality level by untrusted subjects with the exception stated below.
4. Allow the user to explicitly allow a subject to read a C-normal object on per request basis. The user's approval in such case must be obtained via a mechanism independent on the subject. The idea of this objective is to allow the user to perform operations such as submitting a C-normal document to a remote system, that is not trusted to process C-normal data in general and is considered a public object with respect to our classification scheme, without the need to reclassify the document first (and, therefore, to expose its content to any subject). Because this approach is very prone to the user's mistakes, it should be limited to C-normal objects and not applicable to C-sensitive objects.
5. Prevent information passing from objects with a lower integrity level to objects with a higher integrity level by untrusted subjects.
6. Allow the user to specify the maximal integrity level for each subject and prevent the subject from writing to objects with a higher integrity level.
   The idea of this objective is to prevent modification of objects with a high integrity level unless required by the user.
7. Allow the user to define four sets of special input and output objects (two sets for confidentiality protection and two sets for integrity protection) and two special confidentiality levels (for reading and writing respectively) and two special integrity levels associated with the sets for each partially trusted subject, and apply the same rules to partially trusted subjects with the following exceptions:
   a) Allow a partially trusted subject to transfer information from an object O_{in} with a confidentiality level c_{in} to an object O_{out} with a confidentiality level $c_{out} < c_{in}$ if the object O_{in} is in the input set for confidentiality protection, the object O_{out} is in the output set for confidentiality protection, cin is at most the special confidentiality level for reading, and c_{out} is at least the special confidentiality level for writing.
   b) Allow a partially trusted subject to transfer information from an object O_{in} with an integrity level i_{in} to an object O_{out} with an integrity level $i_{out} > i_{in}$ if the object O_{in} is in the input set for integrity protection, the object O_{out} is in the output set for integrity protection, i_{in} is at least the special integrity level for reading, and iout is at most the special integrity level for writing.
8. Allow the user to define the maximal confidentiality level for reading ${C^{max}}_S$, the minimal confidentiality level for writing ${C^{min}}_S$, the minimal integrity level for reading ${I^{min}}_S$ and the maximal integrity level for writing ${I^{max}}_S$ for each trusted subject S, and allow the trusted subject S to read from an object O with a confidentiality level C_O and an integrity level I_O only if $C_O \leq {C^{max}}_S$ and $I_O \geq {I^{min}}_S$, and allow the trusted subject S to write to the object O only if $C_O \geq {C^{min}}_S$ and $I_O \leq {I^{max}}_S$.

The trusted subjects are, therefore, able to transfer information between any objects within some limits.

3.2. Information flow policy formal definition

Let C_O, I_O denote the confidentiality and integrity levels associated with an object O, U_O denote the owner of O, and L_O denote a label assigned to O. The labels will be used to specify the sets of special input and output objects mentioned in the objective 7 above. Let CR_S, CW_S denote the highest confidentiality level the subject S can normally read from and the lowest confidentiality level it can normally write to; let CRL_S denote the highest confidentiality level the subject S can read from if the respective object is a member of the special input set for confidentiality protection; let CWL_S denote the lowest confidentiality level the subject S can write to if the respective object is a member of the special output set for confidentiality protection; let $CRLS_S$ and $CWLS_S$ denote the sets of labels defining the special input and output sets of the subject S for confidentiality protection (an object O is a member of the special input set if the label L_O is in $CRLS_S$). Let IR_S, IW_S, IRL_S, IWL_S denote the lowest integrity level for reading, the highest integrity level for writing, the lowest integrity level l for reading from the special input objects, and the highest integrity level for writing to the special objects of the subject S, and let $IRLS_S$ and $IWLS_S$ denote the sets of labels defining the special input and output sets of the subject S for integrity protection. Let U_S denote the user the subject S is running on behalf of. Let $IRUS_S$ and $CWUS_S$ denote the sets of users that are trusted by S to maintain trustworthy integrity and confidentiality levels respectively on objects they own. $IRUS_S$ would typically include the special system user identifiers used as owners of system files that need to be relied on by applications (e.g. system shared libraries, system programs, ...). $CWUS_S$ would typically include the special system user identifiers used as owners of confidential files that have to be writeable by processes accepting input from users (e.g. the owner of the password database).

Using the notations above a subject S can read from an object O only if **read**(S, O) is true, and S can write to O only if **write**(S, O) is true:

$$\begin{aligned}\textbf{read}(S, O) = {} & [CR_S \geq C_O \vee (CRL_S \geq C_O \wedge L_O \in CRLS_S) \vee \\ & (C_O \leq 1 \wedge \textbf{UserApprovedRead}(S,O))] \wedge \\ & [IR_S \leq I_O \vee (IRL_S \leq I_O \wedge L_O \in IRLS_S)] \wedge \\ & [U_S = U_O \vee C_O \leq 1] \wedge \\ & [U_S = U_O \vee U_O \in IRUS_S \vee IR_S \leq 1]\end{aligned}$$

$$\begin{aligned}\textbf{write}(S, O) = {} & [CW_S \leq C_O \vee (CWL_S \leq C_O \wedge L_O \in CWLS_S)] \wedge \\ & [IW_S \geq I_O \vee (IWL_S \geq I_O \wedge L_O \in IWLS_S)] \wedge \\ & [U_S = U_O \vee I_O \leq 1] \wedge \\ & [U_S = U_S \vee U_O \ \ CWUS_S \vee CW_S \leq 1]\end{aligned}$$

The **UserApprovedRead**(S, O) is the function returning true if the user has approved the exception specified in the objective 4 in the previous subsection.

Each untrusted subject S must obey the following conditions:

- $CW_S = CWL_S \geq CR_S = CRL_S$
- $IW_S = IWL_S \leq IR_S = IRL_S$
- $CWLS_S = CRLS_S = IWLS_S = IRLS_S = \varnothing$

Each partially trusted subject S must obey the following conditions:

- $CW_S \geq CR_S$
- $CW_S \geq CRL_S$
- $CWL_S \geq CR_S$
- $IW_S \leq IR_S$
- $IW_S \leq IRL_S$
- $IWL_S \leq IR_S$

The structure of the **read** and **write** predicates is as follows. The subexpressions in the first pair of square brackets deal with the objectives 3, 4, 7a, and 8, i.e. with preventing the unintended flow of information from a more confidential object to a less confidential one. The subexpressions in the second pair of square brackets deal with the objectives 5, 6, 7b, and 8, i.e. with preventing the unintended flow of information from an object with a lower integrity level to an object with a higher integrity level. The subexpressions in the third pair of square brackets deal with the objectives 1 and 2. The subexpression in the fourth pair of square brackets of **write** prevents a subject running on behalf of a user U_1 from passing information from a C-sensitive object owned by U_1 to a C-sensitive object owned by another user U_2, where it could be read from by subjects of U_2, thus violating the objective 1. The subexpression in the fourth pair of square brackets of **read** has a similar role for integrity protection – it prevents reading of I-sensitive data prepared by another user. The exceptions using the sets of trusted users are to support reading of I-sensitive system files and writing to C-sensitive system objects.

3.3. Security properties of the information flow policy

We have defined the information flow policy with its objectives in mind. It does not, however, provide a guarantee that it really meets the objectives. We will now state some basic security properties in the formal way. The theorems can be proven although the proofs are out of the scope of this paper.

First, we will formally define the flow of information within the system.

Definition 1: Let $\boldsymbol{S}$ be a set of subjects and $\boldsymbol{O}$ be a set of objects. Let O_0, $O_{out} \in O$ be any two objects. We say that a policy allows an information flow from O_0 to Oout within the system ($\boldsymbol{S}$, $\boldsymbol{O}$) and denote it as **flow**($\boldsymbol{S}$, $\boldsymbol{O}$, O_0, O_{out}) if there exists a finite sequence of pairs (S_1, O_1), (S_2, O_2), ..., (S_n, O_n) where $\forall\ i \in [1, ..., n]$: $O_i \in \boldsymbol{O} \wedge S_i \in \boldsymbol{S}$ such that

$$\forall\ i \in [1, ..., n]: \textbf{read}(S_i, O_{i-1}) \wedge \textbf{write}(S_i, O_i) \text{ and } O_n = O_{out}$$

where **read**(S, O) and **write**(S, O) are the functions of the policy determining whether the subject S can read from, or write to the object O.

Using the formal definition of flow, we can precisely define what we mean by an information leak – a violation of the confidentiality protection requirements.

Definition 2: Let $\boldsymbol{S}$ be a set of subjects and $\boldsymbol{O}$ be a set of objects. We say that our policy allows an information leak within the system ($\boldsymbol{S}$, $\boldsymbol{O}$) if

$$\exists\, O_a, O_b \in \boldsymbol{O}\colon C_{Oa} > C_{Ob} \wedge \mathbf{flow}(\boldsymbol{S}, \boldsymbol{O}, O_a, O_b)$$

We can also define the precise meaning of a violation of the integrity protection requirements – information spoiling.

Definition 3: Let $\boldsymbol{S}$ be a set of subjects and $\boldsymbol{O}$ be a set of objects. We say that our policy allows information spoiling within the system ($\boldsymbol{S}$, $\boldsymbol{O}$) if

$$\exists\, O_a, O_b \in \boldsymbol{O}\colon I_{Oa} < I_{Ob} \wedge \mathbf{flow}(\boldsymbol{S}, \boldsymbol{O}, O_a, O_b)$$

Having the definitions, we can state the basic security properties using the following theorems. The first two theorems deal with the case when there are only *untrusted* subjects. In such case, the information flow policy guarantees that no information from a more confidential object can end up in a less confidential object, and that no information from an object with a lower integrity level can influence any object with a higher integrity level.

Theorem 1: If $\boldsymbol{S}$ is a set of *untrusted* subjects and $\boldsymbol{O}$ is a set of objects, our policy does not allow any information leak within the system ($\boldsymbol{S}$, $\boldsymbol{O}$) unless approved by the user.

Theorem 2: If $\boldsymbol{S}$ is a set of *untrusted* subjects and $\boldsymbol{O}$ is a set of objects, our policy does not allow any information spoiling within the system ($\boldsymbol{S}$, $\boldsymbol{O}$).

Another two theorems deal with the case when there may be some *partially trusted* subjects as well. The first of these theorems says that in order to pass information from an object with a higher confidentiality level to an object with a lower confidentiality level using only untrusted and partially trusted subjects, each subject which passes information from an object with a higher confidentiality level to an object with a lower confidentiality level, must be a partially trusted subject and it must be passing the information from its specially labelled input to its specially labelled output. Assuming that no partially trusted subject passes information from its special inputs to its special outputs in an unintended way, any information leak allowed by the policy within a system without trusted subjects is intended.

Theorem 3: Let $\boldsymbol{S}$ be a set of *untrusted* and/or *partially trusted* subjects and let $\boldsymbol{O}$ be a set of objects. Let O_0, $O_{out} \in \boldsymbol{O}$ be two objects such that $C_{O0} > CO_{out}$ and $\mathbf{flow}(\boldsymbol{S}, \boldsymbol{O}, O_0, O_{out})$. For every finite sequence of pairs (S_1, O_1), ..., (S_n, O_n) such that $\forall\, i \in [1, ..., n]$: $S_i \in \boldsymbol{S} \wedge O_i \in \boldsymbol{O} \wedge \mathbf{read}(S_i, O_{i\text{-}1}) \wedge \neg\, \mathbf{User\ ApprovedRead}(S_i, O_{i\text{-}1}) \wedge \mathbf{write}(S_i, O_i) \wedge O_n = O_{out}$, for each pair (S_j, O_j) such that $C_{Oj\text{-}1} > C_{Oj}$:

S_j is a partially trusted subject, and
$L_{Oj\text{-}1} \in CRLS_{Sj}$, and
$C_{Oj\text{-}1} \leq CRL_{Sj}$, and
$L_{Oj} \in CWLS_{Sj}$, and
$C_{Oj} \geq CWL_{Sj}$

The last theorem says that in order to pass information from an object with a lower integrity level to an object with a higher integrity level using only untrusted and partially trusted subjects, each subject which passes information from an object with a lower integrity level to an object with a higher integrity level, must be a partially trusted subject and it must be passing the information from its specially labelled input to its specially labelled output. Assuming that no partially trusted subject passes information from its special inputs to its special outputs in an unintended way, any information spoiling allowed by the policy within a system without trusted subjects is intended.

Theorem 4: Let $\boldsymbol{S}$ be a set of *untrusted* and/or *partially trusted* subjects and let $\boldsymbol{O}$ be a set of objects. Let O_0, $O_{out} \in \boldsymbol{O}$ be two objects such that $IO_0 < IO_{out}$ and $\mathbf{flow}(\boldsymbol{S}, \boldsymbol{O}, O_0, O_{out})$. For every finite sequence of pairs (S_1, O_1), ..., (S_n, O_n) such that $\forall\, i \in [1, ..., n]$: $S_i \in \boldsymbol{S} \wedge O_i \in \boldsymbol{O} \wedge \mathbf{read}(S_i, O_{i\text{-}1}) \wedge \mathbf{write}(S_i, O_i) \wedge O_n = O_{out}$, for each pair (S_j, O_j) such that $I_{Oj\text{-}1} < I_{Oj}$:

S_j is a partially trusted subject, and
$L_{Oj\text{-}1} \in IRLS_{Sj}$, and
$I_{Oj\text{-}1} \geq IRL_{Sj}$, and
$L_{Oj} \in IWLS_{Sj}$, and
$I_{Oj} \leq CWL_{Sj}$

4. Conclusions

The presented security model supplements the traditional discretionary access control by providing protection of confidentiality and integrity of sensitive data against malicious applications running on behalf of the owner of the data. When extended to cover other operations (e.g. creation and deletion of objects, object attributes' manipulation, interaction between subject, etc.) and implemented, it will allow a user to run security critical applications alongside potentially malicious applications while assuring the user that the malicious applications cannot interfere with the sensitive data, whether regarding the confidentiality or the integrity aspect. This would be a significant improvement of the current state in the security of common operating systems in home and small office environment.

References

[1] BELL D. E., LA PADULA L. J.: *Secure Computer System: Unified Exposition and Multics Interpretation,* Technical report, 1976.
[2] TIPTON H. F., KRAUSE M. (editors): *Information Security Management Handbook*, 5th edition, CRC Press LLC, 2004, ISBN 0-8493-1997-8.