Versatile Integrity and Security Environment (VISE) for Computer Systems

Charles G. Limoges, Ruth R. Nelson*, John H. Heimann, David S. Becker

Information Systems Security Directorate, GTE Government Systems Corporation
77 A Street, Needham, Massachusetts 02194

Abstract

We have developed a model of security and integrity for computer systems, the Versatile Integrity and Security Environment (VISE), which describes the basic functionality of these systems and which addresses data confidentiality and correctness of processing. It has yielded significant insight into secure application-oriented systems and is applicable to both Department of Defense and commercial requirements. Based on our model, we have developed an implementation framework which provides for enforcement of flexible access controls that are easily defined and tailored to meet a variety of requirements. This paper examines the integrity problem, discusses the general ideas behind VISE, illustrates the requirements of the framework, and demonstrates how the framework meets a general need for secure, high-integrity processing systems.

1.0 Integrity and security

A "secure" computer system's security services, e.g., confidentiality and integrity, interact with each other, and are not independent. Integrity is an absolute necessity for providing confidentiality. Secure systems which focus solely on confidentiality for security may not, in actuality, be able to assure this service, and hence, may not even be secure by their own definition.

1.1 Integrity

Much effort on the part of others has been put into studying integrity as a component of computer security, looking at several of its qualities. Most of this work focused on integrity as an inherent property of data, which must be preserved by protecting the data from unauthorized change. We take a different perspective.

Data integrity in a computer system depends on how the data is handled by the software, not solely on its content. Change is inherent in computer systems - processing data involves changing the data or using it to generate new data. Therefore, integrity in a processing environment depends on the correctness of the changes that occur to data, and not on prevention of change. Since data in a computer system is changed by users through software programs, integrity depends upon the correctness of software and the correct relationship between the software and the data on which it operates.

It is well known that proving correctness of software is a difficult if not impossible problem. This knowledge led to the current multilevel security (MLS) approach, based on the reference monitor concept. In this approach, only a small amount of the software requires high assurance of correct functionality; this "trusted computing base" protects the confidentiality of the data in the system from compromise by untrusted users or untrusted software. Unfortunately, systems built on this principle, which embodies a specific, limited definition of security, do not meet all users' needs.[1,2] In many, if not all, systems, significant portions of the application software must operate correctly and must violate the security definition implicit in the MLS models.

The VISE approach also relies on a small amount of trusted software, but the purpose of this enforcement mechanism is to protect the application functionality as well as the data.

We specifically define integrity as freedom from corruption or tampering which adversely affects a system's functions. For a computer system, this definition implies "correctness of processing" among the users, programs, and data elements of the system. This definition follows from examining various types of applications, where high integrity is regarded as important, and extracting integrity requirements that are common to them all—the essential requirement being correctness of processing.

* Now at Information System Security, 48 Hardy Avenue, Watertown, Massachusetts 02172

High integrity applications include secure systems, command and control systems, and communications systems. Examples of secure systems include accounting systems, cryptographic key management systems, and electronic funds transfer systems. An example of a command and control system is a weapons launch control system, and an example of a communications system is an electronic mail system.

The common requirement for all of these systems, regardless of the specific application, is that system functions must be performed correctly and exactly as expected. This is the essence of integrity. All systems, whether they perform accounting, manage keys, transfer funds, launch weapons, or send mail, must maintain correctness of processing. Considering the reliance of these systems on software, integrity is something that is desired of every program designed. However, since even non-malicious human users and programmers are fallible, some measure of control is needed to maintain integrity within the computer system itself.

1.2 The integrity problem

Increased awareness of malicious computer software has focused more attention on the problem of integrity. An understanding of the problem stems from these questions: Can users be sure that they're working with accurate data? Has a user's program been maliciously or erroneously modified? Do users really know what their programs are doing? The root of the problem is that, in computer systems, users do not directly change data - programs do it for them. Users may only assume that their data is being handled in an expected manner by the programs they run.

To illustrate the integrity problem, assume a user is running an editor program to create a message. The user types:

```
From:      Bush
To:        Schwarzkopf
Class:     TOP SECRET
Subject:   Desert Storm
Message:   Cease fire
```

The editor program, which has been maliciously modified to alter text, writes:

```
From:      Bush
To:        Schwarzkopf
Class:     TOP SECRET
Subject:   Desert Storm
Message:   Take Baghdad
```

As observed here, a loss of integrity can be very costly, potentially much more damaging than unauthorized disclosure of information.

1.3 Integrity threats

Loss of integrity can be either intentional or accidental. Intentional threats are due to humans, possibly program developers or system users, authorized or not. Mechanisms for loss include viruses, Trojan horses, logic bombs, and malicious modification of programs or data (as in the above example). Accidental loss may be due to erroneous modification of programs or data, or on hardware or software failure.

1.4 Integrity as a part of security

Integrity is a fundamental requirement for total system security. If a program executing in some environment is to do so securely, with respect to handling data and communicating with users, then that environment must ensure that essential security services are provided. The environment must reliably identify the user with whom it is communicating (authentication), and it must ensure that data it handles is not communicated to an unauthorized user (data confidentiality). Additionally, the environment must ensure that the program is performing only its expected functions, and that both the program's functions and the data it operates on are not maliciously altered (integrity). Overall system security cannot be assured without integrity of critical system functions, such as authentication and access control mechanisms.

Traditional views of computer security do not specifically address integrity. Although they identify the need for protecting data from unauthorized disclosure (data confidentiality), they are not concerned with the effects a malicious program might have on other programs or unclassified system data. An analysis of the DoD Trusted Computer System Evaluation Criteria[3], commonly referred to as the "Orange Book," reveals where the integrity problem comes into play.

1.5 Traditional "Orange Book" security

General concerns about computer security led the DoD to develop requirements for assuring data confidentiality. These requirements are specified in the Orange Book, a reference for traditional computer security. Requirements are grouped into hierarchical classes which reflect different levels of protection to be provided by a "trusted" system - "trusted" to provide data confidentiality. These requirements range from providing discretionary (user defined) access control lists for files, to providing mandatory (system defined) sensitivity labels for all data.

The Orange Book is based on the Bell-LaPadula model of confidentiality[4], although other similar data flow security models also fit within the criteria. This model considers the elements of a computer system to be either "subjects" or "objects." Subjects -- users, or processes executing on behalf of users -- act upon objects -- programs, or elements of data. In the Bell-LaPadula model, every subject and object is associated with a particular confidentiality level, i.e., classification. Orange Book systems enforce the Bell-LaPadula rules of "simple security" and the "*-property." The simple security rule ensures that subjects can only read objects at or below their level of confidentiality. The *-property ensures that subjects can only write to objects at or above their level of confidentiality. Enforcement of these access rules maintains data confidentiality, preventing data from high-level subjects and objects from leaking to lower level subjects and objects (Figure 1).

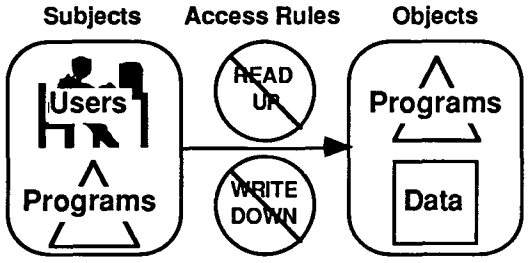

Figure 1. Bell-LaPadula Model

In its view of a computer system, the Orange Book considers two kinds of software - the Trusted Computing Base (TCB), and everything else. The TCB is "trusted" to enforce access rules and to protect itself from modification. The TCB also may include "trusted processes." These are programs that are allowed to violate the Bell-LaPadula rules. Like the TCB, trusted processes are protected from modification, and are trusted to enforce the system security policy even if they break the Bell-LaPadula rules. Everything else, which may include many useful application programs, is assumed to be potentially hostile, or "untrusted," by the TCB. The TCB restricts these programs' access to data and prevents them from modifying the TCB itself, but does not protect the programs themselves from modification.

1.6 The Orange Book is incomplete

The solution outlined in the Orange Book is primarily concerned with data confidentiality, and does not address integrity. Users are not distinguished from the programs they run - a subject can be either a user or a program running on his or her behalf. The useful work of a system, and possibly its critical mission function, is performed by untrusted application programs with no guarantee of integrity. Thus, the security of an Orange Book system can be undermined by its lack of integrity protection.

Consider the previous "Desert Storm" example where a malicious editor changed a message. Since the editor has not changed the classification of the message, the system is still functioning in a perfectly trusted manner according to the Orange Book rules. Data confidentiality has been maintained.

2.0 Versatile Integrity and Security Environment (VISE) model

As shown in the previous section, the security outlined in the Orange Book is inadequate. Application programs and operational data, the most valuable parts of a system, must be protected not only from unauthorized disclosure, but also from unauthorized change. Correctness of processing must also be maintained. In terms of security, the relationship between data and program is just as important as the relationship between user and data. A corrupted program can corrupt data. A program allowed more access than needed can leak data. User interaction with programs, and program interactions with data, must be strictly controlled. An integrity solution must enforce these controls.

The solution we have developed, the Versatile Integrity and Security Environment (VISE) Model, is based upon these goals.

2.1 Contributions from other models

Prior to developing the VISE Model, we examined existing security and integrity models to determine whether or not they could meet our needs for data confidentiality and functional integrity. The models reviewed include Clark and Wilson [5], Harkness-Pitelli [6], Bell-LaPadula, Goguen and Meseguer [7], Lipner [8], and Biba [9]. We also reviewed security and integrity concepts developed under the National Security Agency/Secure Computing Corporation Logical Co-processing Kernel (LOCK) project [10]. We concluded that, although these models provide valuable insight to security and integrity solutions, none completely meets our goals for a high integrity solution. However, many of the ideas presented in these models are useful and applicable to a comprehensive integrity solution, and these ideas have been incorporated into the VISE Model.

The Clark and Wilson model explicitly addresses application integrity. It defines integrity in terms of self-consistency of data and separation of duties among users. Self-consistency of data implies that the correctness of certain data, such as accounting data, may always be

verified by "balancing the books," much like balancing checkbook credits and debits. Separation of duty protects functional integrity by dividing a critical process into sub-processes, and allowing only certain users to perform each part. For example, in a funds transfer system, one user might only be allowed to enter funds transfers, another user might only be allowed to approve the transfers, and still another user might only be allowed to execute the transfers. Thus, the chance that one or more users could collude maliciously to undermine a critical process is greatly reduced. These concepts are clearly useful for applications such as accounting, but the model does not provide guidance on there use in general applications.

The Clark and Wilson model includes the most important concept which we have used in VISE. This is the notion of the triple (user, program, data) instead of the pair (user, data). The departure from the subject-object paradigm allows the particular functionality of the system (represented by the programs) to be controlled and managed separately from the human users and the data acted upon by the programs. The triples are far less abstract than the subject-object pairs and can reflect the actual operations of a computer system in a much more natural manner. The complication of managing the triples is more than balanced by the simplicity, accuracy and power of their expression. Access decisions in the Clark and Wilson model and in VISE are expressed in terms of a user being allowed to use a particular program to access particular data.

The work of Harkness and Pitelli identifies the concept of command authorization as a component of integrity: "that all commands capable of changing data be executed only when issued by authorized users." They also demonstrated the direct relationship of command authorization to Clark and Wilson's separation of duty and Goguen and Meseguer's non-interference, both identified as requirements for high integrity systems.

The Bell-LaPadula security model provides the basic idea of data confidentiality, which is the capacity to control a user's ability to access data - a required component of system security. However, the Bell-LaPadula requirements of no "read up" and no "write down" incorporate a particular security policy. These rules are neither necessary nor sufficient for all secure systems, and do not provide integrity. For example, a database management program might extract unclassified records from a classified database on behalf of an uncleared user. Such a program must operate as an exception to the Bell-LaPadula model. In the VISE model, we wanted to incorporate treatment of these programs, and not treat them as exceptions.

The Goguen and Meseguer security model is based on the idea of "non-interference" - that commands issued by one user do not, under certain conditions, appear to affect a system as viewed by another user. Non-interference is desirable for a high-integrity system. A multi-level security policy, for example, may be represented by the requirement that users operating at a given security level are non-interfering with users operating at lower security levels. Uncleared users should have no indication whatever of a cleared user's classified activities. This model has been shown to be essentially equivalent to the Bell-LaPadula model, and adds nothing new for integrity protection.

The Lipner security model requires controlled "promotion" of programs from development to production status and calls for the enforcement of rigid configuration controls on the operational environment. These requirements are indeed critical to the development of a high-integrity system. Programs should only be developed by trusted developers, and should be thoroughly inspected and tested in an isolated environment before production use.

The Biba integrity model was found not applicable to our integrity solution. Biba postulates a hierarchy of data integrity levels, analogous to the Bell-LaPadula levels of confidentiality, where untrusted processing can only decrease the integrity of data. We found no consistent relationship of this integrity hierarchy to real applications. We also found that the Biba model is inconsistent with functions which intuitively "increase" data integrity, such as intelligence fusion, which correlates related information from multiple sources to increase the accuracy and validity of the data.

The LOCK program developed the ideas of type enforcement and "assured pipelines." Type enforcement supports access control and integrity through the use of labels and levels on subjects and objects which go beyond classification markings. Assured pipelines, which may be created using type enforcement, direct the output from one function to be input to another. Type enforcement is used in VISE to assure that programs operate on suitable data and produce suitably typed data. VISE combines this type enforcement with the Clark and Wilson triple, not with the subject-object pair.

In general, the Bell-LaPadula, Goguen and Meseguer, Lipner, and Biba models did not meet our needs because they do not address our definition of integrity. We needed more in our model, including control over user access to data and control over what a potentially corrupt programs may do. We particularly wanted to distinguish between users and programs. We did not want to make the assumption, as done in some models, that programs inherit privileges from the user running them. We believe that users and programs must be identified and handled separately, and that programs' actions on data should be restricted independent of user privileges, thus making it impossible for a potentially malicious program to exploit

the full capabilities of a highly-privileged user without that user's knowledge.

2.2 The VISE model

For the purpose of discussing the VISE Model, it is important to define exactly what we mean by the terms user, program, data, security policy, and system security administrator. A user is a human who is actively communicating with a computer system. Programs are individually identifiable collections of executable instructions, e.g., executable code files. Data are individually identifiable information elements, e.g., files, records, etc. A security policy is a set of criteria that dictates the security rules for a specific environment. A security administrator is a user responsible for ensuring adherence to a security policy. For programs and data in particular, although their representation in actual systems can vary, the use of these general terms will still be consistent in any application of the VISE Model (Figure 2).

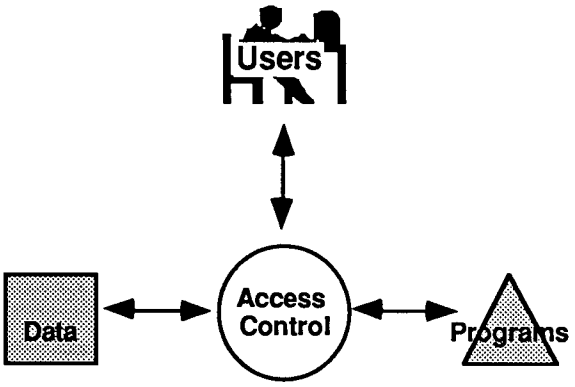

Figure 2. VISE Model

Our model of integrity provides control of a computer system's processing environment and the interactions between its elements - users, programs, and data. Controlling these interactions, restricting what programs a user may run and limiting what data a program or user may read and write, assures data confidentiality and integrity. In the model, we refer to restricting what programs a user may run as functional limitation of access.

VISE supports the concepts of separation of duty, command authorization, and assured pipelines. VISE also provides for the controlled promotion of development code to a production environment. These concepts are combined with our observation that users and programs are different and have distinct privileges, and that a general integrity solution should be versatile enough to address many applications.

VISE accomplishes all of this through configuration control of system elements, and enforcement of flexible access control rules. Configuration controls are placed upon the users, programs, and data elements of a system in order to identify individual elements and to protect these elements from unauthorized use. Interactions between elements are regulated by the VISE enforcement mechanism, which implements the security administrator-defined access control scheme. A key characteristic of the VISE model is that it enforces access control decisions based on the triple of user, program, and data, not the subject-object pair.

2.3 Operational concept of model

We have stated that the VISE Model is based on the concept of functional limitation of access, the concept that users and programs may have different privileges, and the concept that programs running on behalf of a user do not take on the privileges of the user. To see the implications of this in a real computer system, consider the following example. A user is attempting to edit a file. The user should only be allowed to edit the file if the following conditions are met:

- The user is allowed to run the editor program.
- The editor program is allowed to use the file as input and output on behalf of the user.

The significance of the concepts described above is illustrated here in this example. By limiting functional access, the user can run the editor program only if he or she is authorized to do so. By applying access control based on the triple of user, program, and data, the user and the editor program are treated separately in making the access control decision. Instead of combining the privileges of user and program together as a single "subject," the privileges of the user and the editor program are evaluated independently to determine whether to allow access to the file (data).

3.0 VISE framework

The VISE framework is a set of requirements for implementing the VISE model on an actual computer system. The framework also divides the task of configuring a VISE-protected system into logical phases to ensure protection of users, programs, and data. Constructing this framework has proven to be extremely useful, in that it aids us in investigating the issues involved with assuring integrity for actual systems, and it also serves as a mechanism whereby the VISE model itself can be evaluated. We view the VISE framework as one possible implementation of the VISE model.

3.1 Overview of VISE framework

The VISE Framework allows security administrators to configure an actual computer system, employing the VISE Model, to meet their specific needs for data confidentiality and integrity. Programs, system users, and elements of data are identified to VISE and protected. Programs are write-protected. Users are tagged with distinguishing characteristics, such as job title or department number. Data is also tagged with corresponding characteristics.

The access control policy for the system is represented to VISE by security administrators in the form of access control rules. These rules compare specific characteristics of users and data elements to make access control decisions. For the access control policy of the previous example, where the user attempted to edit a file, rules would allow only certain users to run the editor, and would restrict the files that a particular user could access. Other rules would restrict the editor program itself to accessing only text files (not system files or executable code files).

The VISE Framework assures integrity though the functions of registration, policy representation, and enforcement (Figure 3).

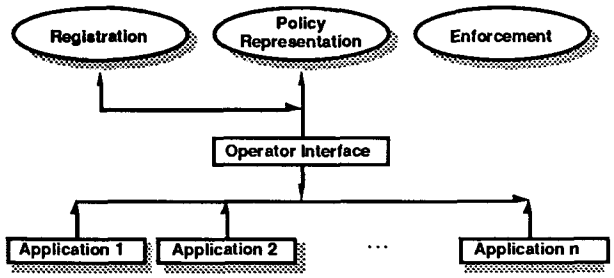

Figure 3. VISE Functions

Registration provides configuration control, policy representation implements the site-specific access control policy, and enforcement makes and upholds all access control decisions (evaluates access control rules). Each of these functions will be discussed in more detail in the sections that follow.

3.2 Registration

All users and programs on a system must be introduced to VISE though a registration process. Registration supports access control, and thus integrity, by maintaining configuration control of the users of the computer system and the programs that are to be recognized on the computer system.

3.2.1 User registration: User registration uniquely identifies users and links them to specific user characteristics. For example, a user might be linked to a characteristic called "Job Title" with a value of "Engineer." This same user might also be linked to a characteristic called "Dept. Number" with a value of "100." Use of these characteristics, called attributes, in making access control decisions is discussed in the section on policy representation. As an example, the table below summarizes a possible assignment of attribute values during user registration for user Jones:

User	Job Title	Dept. Number
Jones	Engineer	100

Users must be registered in order to log-in to and use a VISE-protected computer system.

3.2.2 Program registration: Programs are installed on a VISE-protected system though program registration. Programs must be registered in order for them to be executed by users or other programs. Each program, once registered on the system, is write-protected to preclude corruption by accidental or malicious modification.

3.3 Policy representation

As previously stated, a security policy is a set of rules defining the security services required in a specific environment. An example is the U.S. Government's policy for handling classified information. This policy states that any person having access to classified material must have a proper security clearance and a valid need to know. Another policy might be used by a bulletin-board service (BBS) that provides advertising services to subscribers and to the general public. The administrators of the BBS may allow any user to read advertisements, but may allow only subscriber users to post advertisements. Note that access control policies may vary for different organizations, or even for different sites within an organization, depending on the specific needs of each.

Policy representation allows a system-specific access control policy to be expressed to VISE so that it may be enforced on an actual computer system. Through policy representation, a security administrator defines the attributes, types, and rules which tailor VISE to meet an organization's access control needs.

3.3.1 Attributes: As discussed above, attributes represent characteristics of users (User Attributes) or data (Data Attributes). An attribute is defined by its name, a

set of values, and an ordering scheme. Attribute comparison between user and data elements is the basis for access control decisions.

For example, to represent the user characteristic of Job Title discussed earlier, a security administrator would define a user attribute of "Job Title." This attribute might include hierarchical values of "Engineer," "Manager," and "Dept. Head." Other attributes, such as "Dept. Number," may not have hierarchical values, and would have an ordering scheme of "independent."

An example of a data attribute would be "File Type." This attribute could be used to identify data elements that are used by different programs, and might be defined with values like "Text File," "Drawing," or "Spreadsheet." Data attributes can also represent characteristics of data that are related to characteristics of users, and these relationships can be useful when making comparisons. A data attribute "Data Dept. No." could be defined with the same range as the user attribute "Dept. Number." This data attribute could be used to identify data created by each individual department. The example attributes of "Job Title," "Dept. Number," and "File Type" are listed in the table below:

User attributes are linked to specific users during user registration. Data attributes are assigned to specific elements of data during data attribute assignment, which is discussed in the section on enforcement.

Attribute	Order	Value #1	Value #2	Value #3
Job Title	Hierarchical	Engineer	Manager	Dept. Head
Dept. Number	Independent	100	200	300
File Type	Independent	Text File	Drawing	Spread-sheet

3.3.2 Types: Types are compositions of attributes, with restrictions on the values that each attribute can take. A type consists of a name and a logical expression. For instance, to represent a restriction that a person's Job Title must be greater than or equal to "Manager" and a person's Dept. Number must be equal to "100," the security administrator could define a type called "Dept. 100 - Administrative," using the expression below:

Type	Expression
Dept. 100 - Administrative	Job Title >= "Engineer" and Dept. Number = "100"

Types are useful as shorthand for representing restrictions on one or more attributes. When used in an access control rule (described in the next section), the type is evaluated to yield a Boolean result of true or false. Using the example, a user can be determined to either be, or not be, of the type Dept. 100 - Administrative.

3.3.3 Rules: Rules express the system's access control policy to VISE in a form that can be both easily understood by human security administrators and used to enforce access control on a computer system.

Three kinds of rules may be defined: user-to-program rules, program-to-data rules, and user-to-data rules. Each rule type restricts interactions between two system elements. Before any access is allowed, all applicable rules are evaluated by the VISE enforcement mechanism (described in a later section).

Each rule is an expression, similar to the expressions used to define types. Expressions can use the logical "and," "or," and "not," as well as the comparison operators, "=," ">," and "<."

These rules are flexible. They may be changed by the security administrators whenever a change in the organization's access control policy is desired. This flexibility allows VISE to be configured to match other security and integrity models, including that of Bell-LaPadula where appropriate.

User-to-program rules: User-to-program rules restrict users' ability to execute programs. Depending upon the security policy, security administrators may want to limit which users may run which programs. To meet this need, both "general" and "specific" user-to-program rules can be written by the security administrator. General user-to-program rules automatically apply to every program on the system, and must be met before any program is executed. Specific user-to-program rules, on the other hand, apply to only one program, and must only be met if a user attempts to run that program. For example, if there were an editor program that only members of department 100 were allowed to use, then this restriction could be represented as a user-to-program rule, as in the following:

Kind of Rule	Specific/ General	Specific Name	Allow Access if:
User/ Program	Specific	EDITOR.EXE	Dept. Number = "100"

Program-to-data rules: Program-to-data rules restrict programs' access to data by limiting what data may be read as input and written as output. By defining these rules for each registered program, security administrators can ensure that programs can only operate on data for which they were intended. For example, if an application required that text file messages created by users must be

reviewed and approved before being processed by an e-mail program, then program-to-data rules could be used to represent this restriction. For the "Reviewer" program, the example program-to-data rules, shown below, would only allow the reviewer program to read files of the type "Text File," and would only allow the reviewer program to write files of the type "Reviewed."

Kind of Rule	Input/ Output	Allow Access if:
Program/Data	Input	File Type = "Text File"
Program/Data	Output	File Type = "Reviewed"

Program-to-data output rules tell the VISE enforcement mechanism how to automatically assign attributes to data created by a program. This "attribute assignment" is described in Section 3.4.3.

A security administrator can define as many or as few program-to-data rules for a registered program as are needed to bring the program in line with the site's access control policy. These restrictions can also be changed as a program proves itself in testing, gradually giving it more access to operational data. New and untested programs can start out with very restrictive program-to-data rules, which can be loosened as confidence is gained in the programs' operation.

Besides restricting unproven programs, program-to-data rules are also useful for creating assured pipelines. Consider a message release system application, alluded to earlier, that requires messages to be reviewed by a supervisory authority before being e-mailed from a site. This application would involve the editing of message files, the review of these message files for approval, and finally, the mailing of these approved messages from the site. With the appropriate attributes defined, user-to-program rules could be defined to funnel the output of one program into the other, creating an assured pipeline. The editor could be restricted to only reading and writing message files, the reviewer could be restricted to only reading message files and only writing approved messages, and the mailer program could be restricted to only reading approved messages. Since other programs can be prevented from reading and writing such files through program-to-data rules, complete control of process information flow can be assured.

User-to-data rules: User-to-data rules restrict a users' ability to read data. As with user-to-program rules, both "general" and "specific" user-to-data rules can be written by the security administrator. General user-to-data rules automatically apply to all data on the system. Specific user-to-data rules, on the other hand, apply to only one element of data, and must only be met if a user attempts to read that element of data.

For example, there might be a requirement that each department's files should only be read by people in that particular department. This restriction could be represented as a general user-to-data rule. First, the security administrator would need to define a data attribute of "Data Dept. Number," which would match the user attribute, Dept. Number. The security administrator would then insert an expression indicating that the user's "Dept. Number" must equal the "Data Dept. Number" for the data being read. This general user-to-data rule is shown in the table below.

Restrictions on other data may require a specific user-to-data rule. For example, if there were a personnel data file that only the department head and the managers of department 100 were allowed to read, then this restriction could be represented as a user-to-data rule. This rule would be a "specific" rule, applying to just the personnel file (PERSN.DAT), and would include an expression relating the attribute Job Title to a value of "Manager," as shown below:

Kind of Rule	Specific/ General	Specific Name	Allow Access if:
User/ Data	General	—	Dept. Number = Data Dept. Number
User/ Data	Specific	PERSN.DAT	Job Title >= "Manager"

3.4 Enforcement

Decisions granting or denying access to any program or element of data within a VISE-protected computer system are made by the VISE enforcement process. Enforcement evaluates the rules defined during policy representation using the attribute values assigned during registration. The enforcement process is performed automatically for all program execution and data access attempts.

The VISE enforcement mechanism is linked tightly to the operating system (OS). It is never bypassed. The enforcement function receives access requests from the OS, processes these requests, and sends responses back to the operating system to grant or deny access. The requests, described in the sections that follow, include program execution requests and data access requests.

3.4.1 Program execution request: Program execution requests are sent by the OS whenever a user attempts to run a program, or a program attempts to run another program. Enforcement validates each request by

evaluating all user-to-program rules using the specific attribute values of the user.

For example, assume that user Jones attempts to run the editor program (EDITOR.EXE), and that Jones has the user attribute values in Section 3.2.1. Assume the user-to-program rules of Section 3.3.3. For enforcement to validate the program access request, user Jones value for the attribute Dept. Number, "100," is evaluated in the user-to-program rule. Since Jones' Dept. Number equals "100," the user-to-program rule is met. Enforcement returns a response to the OS granting access to EDITOR.EXE, and the program is run for user Jones.

3.4.2 Data access request: Data access requests are sent by the OS whenever a program attempts to read or write a data element. Enforcement validates each request by evaluating all program-to-data rules and user-to-data rules using the specific attribute values of the user and the data element involved.

Continuing with the example of user Jones running the EDITOR.EXE program, assume that Jones attempts to edit the file PERSN.DAT. Assume the user attribute values for Jones from the above example, and the user-to-data rules of Section 3.3.3.

Assume for this example that program EDITOR.EXE was registered with the following program-to-data rules:

Kind of Rule	Input/ Output	Allow Access if:
Program/Data	Input	File Type = "Text File"
Program/Data	Output	File Type = "Text File"

Also assume that the data PERSN.DAT has the following data attribute values:

User	Data Dept. Number	File Type
PERSN.DAT	100	Text File

For enforcement to validate the data access request, the data element's (PERSN.DAT) value for the attribute is evaluated in the "input" program-to-data rule (shown above). Since the File Type of PERSN.DAT equals "Text File," the input program-to-data rule is met. The EDITOR.EXE program is allowed to use PERSN.DAT as input.

Next, user Jones' value for the attribute Dept. Number, "100," is evaluated in the first user-to-data rule shown above. This value is compared with the Data Dept. Number of the file PERSN.DAT, which also happens to be "100." Since the values are equal, the first rule is met.

Enforcement then evaluates the second user-to-data rule, using user Jones' value for Job Title, "Engineer." This value is checked to ensure that it is greater than or equal to the value "Manager." Since "Engineer" is less than "Manager," the second rule fails. Enforcement returns a response to the OS denying access to PERSN.DAT, and user Jones is not able to edit the file.

3.4.3 Data attribute assignment: Data elements, created or modified by programs during execution, are linked to corresponding data attribute values during data attribute assignment. This process is similar to user registration, where user attribute values are linked to particular users. However, data attribute assignment is automatically performed on program output, as part of the enforcement function. Program-to-data rules, enforced by VISE, dictate how data attribute values are allowed to change.

Data attribute assignment enforces "output" program-to-data rules by correctly labeling a program's output data. For example, a text editor program might be restricted by the following program-to-data rules:

Kind of Rule	Input/ Output	Allow Access if:
Program/Data	Input	File Type = "Text File"
Program/Data	Output	File Type = "Text File"
Program/Data	Output	Data Dept. Number = Dept. Number

If a new file is created, then data attribute assignment would link data attribute values according to the output program-to-data rules. The new file is labeled with a File Type of "Text File," and would also be labeled with the Dept. Number associated with the user running the editor program.

4.0 Conclusions and related work

The GTE investigation of integrity in computer systems has led us to a much clearer understanding of both integrity and security issues, particularly for mission-oriented systems. Integrity and security (confidentiality) are *not* independent, particularly in systems which must violate the constraints of the Bell and LaPadula or other data flow models to accomplish their missions. In these systems, and, we believe, in most systems, security-relevant processing cannot be totally restricted to a small subset of the software. It is useful and important to protect the quality of the application software, to control its use by human users and to constrain its operation by

limiting what data a particular program can access. Attention to these areas, even if it does not offer total assurance of security, can reduce risk. Real security compromises have resulted from allowing software to run with privilege (because it needs to) without protecting its quality or constraining its data access. Code and data are represented similarly in system memory, but compromise has resulted from execution of code which was introduced into the system as data.

4.1 Conclusions

A few of our key conclusions are:

- Integrity in a processing system depends upon the correctness of change, rather than protection of data from change.
- Data flow models capture the idea of security as control of access to data. It is important to consider control of functionality as well.
- Current models incorporate the policy of "no read up, no write down." This policy is useful in some, but not all systems. VISE allows the enforcement of a broader variety of policies, depending on the needs of the system and the mission.

4.2 Related work

A formal representation of the VISE model was created for the NCSC as part of GTE's Internetwork Security Research contract. This model is not intended as a complete security solution, but represents control of the program execution in a processing system. It is part of a collection of models for network security, which also includes an identification and authentication model and a model for secure movement of data from one end-system to another across a packet internetwork.

We also developed a VISE system demonstrator. This focused our attention on some practical problems such as suitable attributes for users and data, and the identification of programs in a system.

GTE is investigating functional security in distributed systems, extending some of the VISE concepts to process protection in a distributed environment.

5.0 References

[1] R. Nelson, "What is a Secret and What Does That Have to do with Computer Security?," presented at the New Security Paradigms Workshop II, Little Compton, RI, August 1994.

[2] C. Limoges, R. Nelson, J. Brunell, J. Heimann, "Security for Mission-oriented Systems," MILCOM '92, San Diego, CA, September 1992.

[3] Department of Defense Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD, National Computer Security Center, December 1985.

[4] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: A Mathematical Model," ESD-TR-73-278 Volume 2, MITRE, November 1973.

[5] D. Clark and D. Wilson, "A Comparison of Commercial and Military Computer Security Policies," Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, CA, April 1987.

[6] Pitelli, P., "Formalizing Integrity using Non-Interference," Proceedings of 11th National Computer Security Conference, Baltimore MD, October 1988.

[7] Goguen, J. A. and Meseguer, J., "Security Policies and Security Models," Proceedings of the 1982 IEEE Symposium on Security and Privacy, Oakland, CA, April 1982.

[8] Lipner, S. B., "Non-Discretionary Controls for Commercial Applications," Proceedings of the 1982 IEEE Symposium on Security and Privacy, Oakland, CA, April 1982.

[9] Biba, K. J., "Integrity Considerations for Secure Computer Systems," MITRE Technical Report TR-3153, April 1977

[10] Saydjari, O. S., et al., "LOCKing Computers Securely," LOCK - Selected Papers, 1985-1988, Secure Computing Technology Center.