What is a Secret
- and -
What does that have to do with Computer Security?

Ruth Nelson

Information System Security
48 Hardy Avenue, Watertown, MA 02172

Abstract

This paper questions some of the basic assumptions of computer security in the context of keeping secrets, and it finds some major discrepancies. It then proposes a new paradigm for functional security in computer systems.

The first conclusion of the paper is that secrecy and security cannot be expressed both algorithmically and accurately. The second conclusion of the paper is that functional security models, which look at the application software as well as the data, can be very useful. Use of more realistic models involves a more complex definition of secure systems, but it may reduce the conflict between security and function and may result in more effective secure systems.

Introduction

The question "What is a secret?," is not often asked in the computer security community. We think that we know what secrets are and what security is. The questions this paper raises are whether our assumptions about secrets and security are true and, more importantly, whether they lead to useful conceptual and formal models of security. When we use these assumptions without questioning them, and develop models and security criteria for trusted computer systems, we may be misleading ourselves and the user community into a false comfort. On the other hand, we need models and criteria to build and evaluate systems. This paper examines some assumptions embedded in current security models and some possibly fundamental limitations of an algorithmic approach to security. The goal of the effort is not to discard the current paradigm altogether, but rather to understand its scope and its limitations and to explore whether some other approaches may be more useful.

The paper reaches two conclusions. The first, and more important, is drawn from examination of the characteristics of secrets and systems. It is that secrecy and security cannot be expressed both algorithmically and accurately. Any precise model we construct of a secure system is an abstraction that captures part of the requirements but misses the heart of the matter. This fact is probably responsible for much of the conflict between the security profession and the system users and designers. Our models are not and cannot be "correct," because of the nature of the problem. The model is not the reality.

This does not mean, however, that precise security models are not useful. They point out aspects of the system that need attention, they show security problems and indicate countermeasures, and they encourage careful system design with security as a real system goal. Security models are important for secure systems.

The second conclusion of the paper is that a functional security model that looks at the application software as well as the data can capture required behavior that is outside of the information flow models. This makes the functional model extremely useful and appropriate, especially for those systems that require large amounts of information flow from "high" to "low." It is true that the definition of security in the functional model is more complex (and therefore harder to enforce with high assurance) than the "no information flow" definition. However, enforcement of any algorithmic model cannot assure that the system keeps secrets, and system operational requirements generally conflict with the data flow constraints imposed by current models. Use of more realistic models may reduce the conflict between security and function and may result in more effective secure systems.

What is a secret? And how does it behave?

One "common sense" definition of a secret is some information that is purposely being kept from some person or persons. It is interesting to investigate the behavior and characteristics of secrets; this can lead to doubts about secrets being easily defined objects. It can also lead to some understanding of the limitations of any automated or unautomated approach to keeping secrets, for secrets cannot in general be kept absolutely. In the broader security community, this is fairly well understood. The goal is to keep things secret for some amount of time, or to make secrets hard to obtain, or to slow down the leakage of secrets. In computer security, however, we ignore considerations of this kind, and we try to design systems that have high assurance of not leaking secrets at all. This leads us to some very difficult technical problems, but may not be leading us to greater system security.

Another interesting question is what piece of information contains or communicates a secret. The relationship between information and secrecy is complicated, as the following examples suggest.

1. If we cut a secret in half, is it still a secret? Suppose that a secret recipe calls for 6 cups of sugar. Is 6 the secret? Cups? Sugar? That may depend on what the observer already knows and what he can guess from context. If he knows nothing then none of the pieces may be a useful secret, but the whole is. The U.S. Industrial Security Manual calls for independent portion marking of classified documents, but it is known that this is inadequate for some documents that are classified only in aggregate.

2. If we move a secret out of context, is it still a secret? In the example above, it is clear that "6" is not a secret in general. In the context of the secret recipe, it may be.

3. If we collect enough non-secret information and process it correctly, we may have a secret. One example of this is intelligence signal processing. The radiation containing the signals can be freely detected and is not kept secret, but the sender hopes that the information-containing signals themselves stay hidden in the surrounding noise.

4. Some observers may already know something about a secret or have a good guess at it; in that case, a large secret can be communicated with very little information flow. If an observer knows that an invasion is planned by the Pentagon, but does not know when it will take place, he may be able to learn the date from simple activity observation around the Pentagon (the famous Pizza Truck example).

5. Secrets can be communicated by very condensed codes, if the parties have agreed on these ahead of time. The famous "one if by land, two if by sea" is an example of very effective secret information transfer.

6. In encrypted communications, we can communicate large amounts of data with no secrecy leak, because there is another secret protecting the flow. However, if the observer learns our encryption key and algorithm, a relatively small amount of information, then he can learn all the information we have sent encrypted with that key.

7. Sometimes the information content of binary data is easy to extract because the data representation is an easily guessed standard. A good example of this is ASCII text stored in a computer. If an observer gets access to the text data, she can probably know the information it represents, or at least much of it. Sometimes the data representation is less easily guessed, making the information harder to extract. Examples of this are data base data, hard to interpret without knowing the schema, and binary data constructed and used by a particular application program.

Whatever the definition of a secret is, it seems clear that if no information is passed from the holder of a secret to the observer who desires the secret, then no secrets are passed either. However, if the observer has access to any information at all, then it is extremely difficult to know whether she has the secret, knows the secret with certainty, or has enough clues to make useful guesses about the secret.

Automating security

The field of computer security is based on the premise that it is possible to specify and construct processing algorithms that capture security requirements and allow them to be met by an automated system. The attempt is to adapt the security rules developed for manual systems and paper by making them both precise and general. Even in a manual environment, real security cannot be made precise, and this has been understood from the beginning. The goal for computer security was and is to construct an approximation to security that is precise, algorithmically enforceable and "safe." This implies some overconstraint, but this is necessary in a system that cannot use human wisdom and discretion.

The formalization efforts in the 1970s were spurred by the increased use of computing, the complex nature of computer systems (and their propensity for failure and erratic behavior), and the clear impossibility of analyzing all code in every system to insure that security was preserved.

The Reference Monitor concept proposed by Anderson in 1972[1] suggests use of a small amount of highly trustworthy software (the Reference Monitor) which constrains the operation of the system so that only authorized access can occur. The Bell and LaPadula model[2] and others developed since then, have guided the design and evaluation of reference monitor-based systems for multilevel security. The TCSEC[3] generalized the Reference Monitor to include other security relevant functions of the system, keeping the idea that a small part of the system could be trusted to prevent all other users and software from unauthorized access.

The reference monitor concept and the computer security research of the last twenty years assume that security can be defined in an application-independent manner. This implies that security can be defined syntactically rather than semantically, that understanding of the meaning or even the form of information is unnecessary for assuring secure automatic system operation. Since semantic interpretation is very difficult if not impossible to implement algorithmically, the assumption of content-independent security is crucial for any general approach to automating security.

These two assumptions: that it is important to localize security functions to a relatively small portion of the system and that automated security must rely on a structural abstraction rather than a semantic understanding of secrecy, have been crucially important in the development of secure systems. However, the current Reference Monitor paradigm makes some additional assumptions that act as further constraints on system functionality in the name of security. These additional constraints often forbid necessary functions of the system, thus leading to a strong conflict between security and functionality. An interesting question is whether there can be other models of automating security that are less restrictive than the current models and which capture more of the desired functionality of the system.

The subject-object paradigm

The current models of computer security are all based on control of access to data by human users or software operating on the users' behalf. The subject-object paradigm divides the world into active entities (subjects), which access passive entities (objects). The actions of subjects are to access objects; more specific functionality is not modeled. Access types include read, write and execute, implying a consideration of active access. However, the special characteristics of execute access are lost and are often modeled as equivalent to read access.

This model captures the concept of access control for the data resources of the system, but does not consider access to specific processing functionality. The assumption is that security can be modeled in terms of access to data. Behavior of "untrusted" code is assumed to be non-security-relevant, possibly hostile and subject to contamination by Trojan Horses deliberately trying to leak data.

Data access is constrained by the system, but the software itself is given no special protection. Code resident in the system is treated like other data, and is assigned a label representing its sensitivity. When software is operating, it runs "at" the current security level of the user. The system has no particular knowledge of the functions performed by the software or the types of data it is designed to use, nor does it provide any special configuration control to prevent the software from being changed, either by the user or without his knowledge.

In addition, the subject-object paradigm assumes that human users and the software acting on their behalf have similar characteristics. In a multilevel secure environment, for example, users have clearances and can have read access to data up to the level of their clearance. The software running on behalf of a user takes on the privileges of the user. Software runs "at a level," with read access at or below that level and write access at or above that level.

Mandatory security and information flow constraints

The current computer security paradigm embodies constraints on information flow from "high" to "low" as essential for confidentiality. The "mandatory security" (MAC) policies and models all assume that information flow represents potential loss of confidentiality and that control of this flow represents security. However, in all practical systems, some information flow is necessary to make the system work. The solution to date is to declare that this flow must be mediated by trusted processes, which can determine the difference between information flow that violates confidentiality constraints and flow that is safe. The goal is, however, to minimize the flow of information and to get as close as possible to the ideal of no flow in the interest of high security.

Mandatory security policy restricts data flow from a higher to a lower security environment and also across compartment boundaries. Untrusted processes (subjects) running at a higher level are not permitted to write data (objects) at a lower level, nor are they allowed to cause effects that can be seen by a lower level user. Any necessary data flows from high to low are done by trusted processes and are not covered by the model. These flows are examined on a case by case basis.

The abstraction assumes that information flow from "high" to "low" is equivalent to unauthorized disclosure of secrets and that this flow must be avoided in a secure system. It seems obviously true that preventing information flow prevents disclosure of secrets. The model, therefore, does represent a "safe" formulation of confidentiality. The problem is the conflict with required system functions, which can be more or less severe, depending on the use of the system. In all practical systems information flow from high to low is unavoidable, and, in some systems, it is the main required activity.

Mission-oriented systems

The current security models have been helpful in making a precise definition of automated security, and they do capture some of the aspects of secure computing. In the time-sharing environment for which they were devised, the approximation of zero data flow from high to low and the identification of user with software work. However, they do not apply nearly so well to a growing class of systems, which we have called "mission-oriented."[4]. In these systems, the computation is directed towards a single mission objective. The application software is typically static, having been developed for the mission and installed in an operational configuration. Software development is typically not done on the operational machines but in a separate development environment. Processing is not done on behalf of human users; it is a response to conditions such as input signals, time of day, communications with other machines, etc. Users do not have much interaction with the computer system, and their interaction is quite stylized compared to a time sharing user at a terminal. The system is designed to produce output for users, and the users are not authorized either to see the system's internal data or to modify its operation.

As a designer of secure dedicated systems at GTE, I participated in analysis of security and functional requirements for a number of these mission-oriented systems. One of the issues was whether and how "trusted" technology could help. Our conclusion for most of our work was that we needed a reliable operating system environment but no mandatory security controls. The information flow and labeling constraints imposed by these controls would not add to the security of our systems and would interfere with needed functionality.

Leakage problem

The assumptions in the current MLS model are violated when there is a required data flow out of the system to a "lower" environment. The data flow constraints are necessary because of the lack of control over untrusted functionality and the possibility of Trojan Horse software. If the assumptions of the model are violated (and they always are in real systems), then Trojan Horses can cause compromise of confidentiality, no matter how strong the Trusted Computing Base may be. Here is an example:

Suppose that an A1 trusted system is connected to a network via an end-to-end encryption device that encrypts user data but leaves the network addresses exposed. The device works securely and is trusted. It receives individual packets and address information from the application and sends out encrypted packets to the requested address, if the communication is authorized. The untrusted communications application is running at Top Secret and is allowed to communicate with two other applications at two other network addresses. These are also running at Top Secret. This untrusted application then has available to it a two-symbol alphabet -- the two addresses. If it contains a Trojan Horse programmed to use this alphabet, it can send one bit of Top Secret data with each packet by ordering the packets to its two correspondents so that the sequence of addresses encodes the data. (If the application can send to more than two addresses, then its symbol alphabet is larger and it can send its information more quickly.) The important thing to notice about this example is that the trusted software is all acting correctly, but it cannot prevent the leakage of information out of the system. The model works only in the absence of required or permitted information flow.

Functional security model

A truly secure and effective system would allow information to flow as needed for system functionality without allowing unauthorized release of secrets. This kind of security requires that there be an algorithmic way of telling whether particular data actually conveys a secret. This is difficult if not impossible, as the examination in the beginning of this paper indicates. The security approach that considers processing function as well as information flow provides for the required information flow, but controls the functions that cause this flow to happen. This is still a syntactic model of security, but it is significantly more flexible than a model that considers system activity only in terms of data access.

An abstraction of security that controls access to particular processing functions as well as to data captures security and functional requirements more realistically than the current data access reference monitor models. The new paradigm proposed in this paper does this. It is an outgrowth of the model described by Clark and Wilson[5] in 1987 for commercial security applications. The model was further developed at GTE and shown to apply to multilevel security and other DoD applications[6,7]. This paper describes some of the previous work and refines the approach. It also considers some fundamental issues of secrecy and its automation and concludes that the "historical" view of what is secret, as embodied in both the current paradigms and the new one, are the best we can do in an algorithmic environment.

The new paradigm proposed in this paper considers the resources of the system to be both passive data and active processing services (performed by software programs). It looks at access in terms of a triple: human users, who may request access to data or processing service; programs, which access and produce data; and data. Only the data resource is passive. The idea of security embodied in this model differs significantly from that in the reference model, subject-object concept. The system is secure if it performs its functions as programmed. The security policy is not part of the model and may be different for each system.

The Clark and Wilson model recognized that the functionality of a computer system depends on its application software, and that correct functionality depends on correct software accessing the appropriate data for that program. It also recognized the need for functional limitation of access, that is, for allowing users to use some software in the system and not other software. Their model went beyond subject-object to the triple of user, program, data. The Clark and Wilson work was intended to capture the way that operational business data processing is done. The application software for commercial systems is also usually developed on separate development machines, is installed and becomes operational through controlled procedures and is not permitted to be changed or updated in an uncontrolled fashion. In addition, auditing is commonly done in business computing so that transactions are checked and books balanced.

The Clark and Wilson model incorporates the business computing paradigm. Only some of the software is configuration and access controlled. Auditing and balancing functions are included in the model. At GTE, we recognized the similarities between the Clark and Wilson paradigm and the DoD mission-oriented systems we were developing. We extended the model and removed the constraints of business application.

In our version, all application software in the system is registered and installed. The enforcement mechanism, a kind of reference monitor that manages active as well as passive resources, controls access on the basis of users, programs and data. Programs and users are known to the system individually; data is known by its type and characteristics. Each program has appropriate input and output data types, and the relationships between users' privileges, input data and output data can all be expressed and enforced.

A partially formalized model was developed for this concept. Programs are active, acting as functions that transform and create data. The security policy is not embedded in the model so that each system can be tailored to its requirements. It is possible to define levels of classification and privilege and to constrain data flow from high to low. The data flow models are a kind of projection of this more functional model of security. Instead of having software be either "trusted" and examined or untrusted and unprotected, the system protects and constrains all software. Some programs are "trusted" to produce output at less than the levels of some of their inputs, but they are constrained as to what types of data they can use and produce. All programs, trusted or not, are protected from unauthorized change. Data can only be produced or changed by a program that is authorized and registered for that type of data. Some data access is constrained by the privileges of the user running a program, but programs can also be allowed to have internal data that their users cannot access more directly.

The reference monitor concept was developed with the idea that only a small portion of the software in the system could be thoroughly examined and verified correct. It is still true that verification of correctness to any high degree is difficult if not impossible for large programs. However, it is also true that high quality software can be developed. In the functional security model, the security constraints and security functionality of each program are explicit and visible. Each program can be developed with an appropriate amount of assurance for its function. Once it is developed and installed, the system will preserve its quality and prevent it from being corrupted.

Security as history

It is clearly difficult to determine what secrets are and whether a particular information flow constitutes a security leak. Automated, application-independent, general security models cannot hope to capture the semantics of secrecy. It is noteworthy that both the "traditional" models and the functional model proposed in this paper take a historical view of data sensitivity. The sensitivity of data depends on its derivation -- how it was produced or changed, or how it may have been produced or changed.

In the Bell and LaPadula model, the presumed data sensitivity level depends on the access matrix. The *-property requires that data output by a program (or data to which the subject had write access) be classified at least at the level of the most sensitive input (subject had read access). This is a kind of high-water-mark history of the data. For untrusted subjects, all interesting history is captured by the current label, determined by the access matrix at the time the subject is active.

In the functional security model, the history is more complex, because it includes the processing. If the system is operating correctly, then any data object in the system either was in the system at the initial time, or it was created by some sequence of legitimate program executions.

Conclusions

Security is a subtle concept and hard to capture in an algorithmic way. The current reference monitor model of security focuses on data and on access to data. It was designed for the time-sharing, general purpose computing environment and applies reasonably well. It does not allow control of authorized information flow from high to low and so may not reveal or prevent leakage of data in a system with required flows. The functional security model has a less precise but more realistic idea of security and correct operation. Its enforcement mechanism protects both software functionality and data. The functional model includes the current security model in that systems can be configured to enforce access on the basis of subjects and objects and a lattice of security levels, but it can also provide controlled, authorized data flows without allowing secrecy leaks. All of the models are syntactic and historical and avoid the problem of what a secret actually is.

References:

[1] J. P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, vol. I, AD-758 206, ESD/AFSC, Hanscom AFB, Bedford, MA, October 1972.
[2] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: A Mathematical Model," ESD-TR-73-278 Volume 2, MITRE, November 1973.
[3] Department of Defense Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD, National Computer Security Center, December 1985.
[4] C. Limoges, R. Nelson, J. Brunell, J. Heimann, "Security for Mission-oriented Systems," MILCOM '92, San Diego, CA, September 1992.
[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] C. Limoges, R. Nelson, J. Heimann, D. Becker, "Versatile Integrity and Security Environment (VISE) for Computer Systems, New Security Paradigms Workshop II, Little Compton, RI, August 1994.
[7] Internetwork Security Research, Contract Number MDA904-89-C-6030, 1989-1993