Application Level Security Using an Object-Oriented Graphical User Interface

Terry Rooker
D25
Naval Surface Warfare Center Dahlgren Division
Dahlgren, Virginia 22407

Abstract

The Trusted Computer Security Evaluation Criteria has become a defacto standard for security features in trusted systems. Unfortunately, the TCSEC was formulated at a time when computing was done in centralized facilities with low level access (i.e. operating system access) to the computer. Present computer use is much different. Users access applications, and only rely on the operating system to support the application. In this style of computing the application is more important for security, yet the TCSEC places all the responsibility in the operating system. In this paper we outline some of the changes required to move the focus for security from the operating system to the applciation. Since much of the application relies on the user interface, some of this change must also address the user interface. By emphasizing the application and interface security can be easier and more consistent across applications, and different computer systems.

In the United States, the primary statement of computer security has been the DOD Trusted Computer Security Evaluation Criteria [7], better known as the Orange Book or TCSEC. Much objection to the TCSEC stems from its focus on a single security policy; the DOD hierarchical system for protection of classified information [1–4]. These complaints are valid but there is a more fundamental problem with the TCSEC. That problem stems from the time that the TCSEC was originally developed in the mid to late 1970's. Our current view of computing is different from that time, and even if the TCSEC was freed from the strictly hierarchical policy it would still not address current trends in computer use.

We propose an alternative to the whole style of protections provided in the TCSEC, a different view of security. In the TCSEC the protections are located in the operating system. In the new view protections are distributed between the operating system, and the application. The new view is necessary because the trend away from systems level operations to applications. The system proposed here exploits these trends. It relies upon the abstractions provided by modern applications and object oriented graphical user interfaces to enhance the security that is provided by the operating system.

1 Problem Statement

The conceptual foundation of the TCSEC lies in the mid to late 1970's. The style of computing then was typically a mainframe computer with many terminal connections. Specialized support personnel would actually operate the computer and run jobs. Today, users have a cpu sitting on their desk (in some type of enclosure with the necessary ancillaries). They use applications which are loaded directly onto their local cpu and then run. These two views must be reconciled.

The emphasis was on the operating system and systems level programs. Even casual computer users had to have a fairly good understanding of the operating system. In addition, there were not many commercially available utilities or even application programs. In this environment the users required low level access to the system to perform their job. If a certain utility had not been written by someone else then the user required the tools to write his/her own. These tools were compilers, assemblers, and command languages (such a VMS COM jobs or UNIX Scripts). The computer security problem was magnified, since the low level access required low level security protections. These low level tools even aggravated computer security concerns such as convert channels since the tools provided access to such a variety of system resources that the number of possible covert channels was greatly increased.

In the current view, a user has a box on her desk. She has a variety of software applications that she uses to do her job. In many cases such users neither desire access to, nor care to have access to low level system primitives. Any necessary access to low level operations is done for the user by the application.

Such application-oriented operations conflicts with the TCSEC. The TCSEC treats the system in its entirety. Everything from the user interface to the lowest level of instruction must be considered when evaluating/certifying a system. Traditionally, it has been felt that focusing the protections at the lowest level possible will make it easier to understand the protections. For a monolithic system that is completely specified at its inception, this belief is true. The problem arises when you desire to use a different application. Since most applications perform low level operations for the user, they run afoul of the low level protections unless they or the system were designed for that operation. Consequently, it is difficult if not impossible to build a TCSEC style trusted system that allows simple execution of applications.

This is not a failing of the TCSEC. It is simply a problem that arises because we use computers differently now. The question should be whether only low level protections are necessary in the new style of computer use. The problem is aggravated since most applications are written with disregard for security. Granted some applications have simple password protections, but in general the only security provided in any measure is meant to protect against unauthorized duplication. Some researchers are beginning to understand that the security protections can be located in other parts of the system, such as in the application [6]. This understanding stems from the new style of computer operation. Unfortunately, such security distribution is at odds with the TCSEC.

2 A Solution

Ideally, a new view of computer security should exploit the new style of computing. This new style in some cases helps, for example covert channels would be less of a threat since fewer system resources would be accessible. In other cases the new style introduces new problems, for example composing the application security features with the operating system (and possible different operating systems). Defining this composition will be a major problem in achieving any sensible assurance.

The application view of a computer has several advantages for system security. Probably the most important is that only the information necessary for the task at hand is made available to the user. In essence the application provides a context that the system designer can use to restrict features. For example in a query database at any given screen there would only be certain reasonable types of information necessary. To provide a specific example consider an online employment database. When accessing the personal information for the applicant, only personal history and employment information are relevant. The employment history can be used to screen for possible employment against a job database. At this point, the personal history may not (and possibly legally could not) be relevant. When a possible match is found against the job listing, information about the employer would be relevant. Other users might require access to unemployment insurance records, which should be unavailable to users only screening for employment.

The application provides much of this protection. Notice that at any time, access to the operating system was not required by the user. Even if the user must start another application, access to the operating system is not necessary. Either a master application can be used to coordinate different applications, or the applications themselves can call other applications. Not all of the protection would reside in the application. The operating system would be responsible for some of the protections. Ideally, a subset of the security features would be identified and the operating system would provide those features across all applications. The applications would be responsible for the remainder if they were applicable to the type of application.

In this view of security the information displayed for the user is important, which leads to the user interface. Much of what the user sees is a function of the user interface, and not the individual application. This is particularly true of graphical user interfaces (GUIs). A GUI is a screen display where the physical display is divided into windows, familiar examples being the X-window system, and Microsoft's Windows. Not only are individual processes and applications displayed as separate windows, but much of the user interface is displayed graphically, such as buttons and bars for window commands, and icons for files or applications. Much of the security features discussed in this paper are also relevant to non-graphical user interfaces (i.e. command line interpreters, and even ascii-based menus). The problem is that GUIs are in such demand that it is better to focus the discussion on their features, bearing in mind that the solution does not require a GUI.

The problem with GUIs is the number of levels of software between the user and the kernel. The abstract layers of software are: application, window, window manager, system software, operating system, and kernel. Not all GUIs have these layers, and some might even have more. With this number of layers it is little wonder that assurance of such a system is suspect. Some method is required to reduce the complexity.

Object orientation offers some hope of reducing the problem. Object orientation in general promises to deliver many of the advantages of software engineering. In particular object orientation offers encapsulation, inheritance, and abstraction. A truly integrated object oriented GUI (OOGUI) might offer even more advantages.

Currently, the only OOGUI commercially available is the NeXTstep environment [10], so I will use it as an example. The interface is structured in an Objective-C (the object oriented language used for implementation, a mix of Smalltalk and C) class hierarchy. All classes are subclasses of the Object class which allows all objects to share certain attributes. This sharing is the inheritance. Any class that is defined lower in the hierarchy than an arbitrary class inherits all of the functions and data structures of that class. Those subclasses are then free to modify or overwrite the inherited functions and data structures.

NeXTstep minimizes the GUI complexity issue by eliminating several of the layers. For NeXTstep the layers are: application, NeXTstep, Mach kernel. For compatibility the "traditional" Unix system calls are implemented as calls to the kernel, but they are not used by the NeXTstep environment. All code in a NeXTstep application is part of the class hierarchy, therefore every function must have a place in the hierarchy. Either a function will be located in a default location or it will be inserted into the hierarchy by the programmer, possibly inheriting other functions from superclasses.

The inheritance can help the security issues. The security functions can be collected into separate classes, preferably grouped by some functional classification, i.e. access control object, auditing object, etc. In addition, the methods (messages) in these classes could not be overwritten. Then all subclasses would inherit the security properties for the appropriate activities. All interaction with the services provided by the operating system would also automatically get the security functions.

With security objects there would be three levels of security abstractions; operating system level protections, OOGUI level protections, and application level abstractions. When implemented these security abstractions would actually reside in the security objects. The purpose of these objects would then be three-fold: to keep the applications from directly accessing system level resources, provide all security functions for the GUI, and to provide a foundation for application developers to include security in their applications.

3 Future Directions

This type of application level security is essentially a new paradigm in computer security. The immediate obstacle is the view influenced by existing security guidelines. This new view is proposed with as little bias as possible (towards any security policy). This presentation does not mean that it cannot support a DOD style hierarchy, rather this this paradigm supports a much broader range of options. Regarding the TCSEC, the main difficulty would be the lack of a single coherent trusted computing base. Again, the TCSEC view of the world is monolithic operating systems while some interpretations of the TCSEC [8,9] address distributed resources, they still rely on the concept of a single TCB. As distributed security is better understood, the security object will become more feasible. In particular, the security object appears well suited to a client server type architecture. The Trusted Mach project is the only system currently undergoing evaluation that supports anything close to this style of TCB [5]. T-Mach at least provides a model to work with, and it would be instructive to map the requirements of security objects to the services of T-Mach.

A related question is the OOGUI itself. Currently, NeXTstep is the only OOGUI available, although other vendors are working on similar products. Accessing a distributed TCB's services would require extensions to the OOGUI. The problem arises because of a difference in philosophy. The methods of the security objects could not be overwritten, which is counter to the idea of inheritance. It might even require a modification to the underlying system since normally the methods can be overwritten in subclasses. The issue is more than simply restricting the methods, but in the semantics of the methods. For example, since the security methods cannot be overwritten what should be the desired behavior if a security methods was overwritten by a subclass? An additional factor is that much of the behavior of this OOGUI is only determined at runtime. It is possible that the behavior might not be detected during compilation.

A similar problem exists with the security considerations. The security functions are not provided by a

monolithic TCB. The functionality is provided by several different abstractions. There is a major problem with analyzing the overall security of such a system. At the higher levels of assurance, a formal model is required. If the security functions cannot be sufficiently described the only recourse might be to model the entire system! This solution is obviously unsatisfactory. A better solution would be to provide a method for composing the security functions when they are provided by different parts of the system. This question is even more important since most current approaches to formal security models rely on first order logic, which is not suitable for extensible domains. As we better understand composible security functions, we can begin to understand assurance of security objects.

4 Conclusion

We no longer use computers as we did when the TCSEC was developed. This is a separate issue from the complaint about the TCSEC's focus on the DOD hierarchical security policy. Most users do not have and do not want system level access to the computer resources. They need to start applications and get work done. We must move some protections from the operating system to the applications. A trusted operating system could provide protections for these new applications, but it would be more difficult. It would be easier to place the protections in the application, since the application can protect those resources it knows it uses.

The problem then is to provide an environment that simplifies developing these applications. An object oriented graphical user interface provides one possible solution. Incorporating security objects into the inheritance hierarchy the security protections would automatically be available to application code. These security objects would also provide the interface to the operating system level protections.

Such a radical change in trusted systems has its difficulties. Not all the necessary tools are in place. Work is required on composing security features across different parts of the system, understanding distributed TCBs, and the effects of theses changes on the OOGUI. Such radical change also has promise. Security would not be as onerous as it is now. Security features across applications would be more consistent. The most important advantage would be that such security classes would be a major advance towards providing "plug and play" security in a distributed architecture.

References

[1] L. Chalmers, "An Analysis of the differences Between the Computer Security Practices in the Military and Private Sectors." Proceedings of the 1986 IEEE Symposium on Security and Privacy, 1986.

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

[3] P. Neumann, "Rainbows and Arrows: How the Security Criteria Address Computer Misuse." Proceedings of the 13th National Computer Security Conference, National Computer Security Center, 1990.

[4] D. Parker, "Restating the Foundation of Information Security." Proceedings of the 14th National Computer Security Conference, National Computer Security Center, 1991.

[5] E. Sebes, R. Freitag, "Trusted Distributed Computing Using Untrusted Network Software." Proceedings of the 14th National Computer Security Conference, National Computer Security Center, 1991.

[6] D. Sterne, M. Branstad, B. Hubbard, B. Mayer, and D. Wolcott, "An Analysis of Application specific Security Policies." Proceedings of the 14th National Computer Security Conference, National Computer Security Center, 1991.

[7] National Computer Security Center, "Trusted Computer System Evaluation Criteria," DOD 5200.28-STD, 1985.

[8] National Computer Security Center, "Trusted Database Management System Interpretation." NCSC-TG-021 Version-1, 1991.

[9] National Computer Security Center, "Trusted Network Interpretation." NCSC-TG-005 Version-1, 1987.

[10] NeXT Computers, Inc., NeXTstep Concepts, 1990.