Design of a message processing system for a multilevel secure environment*

by STANELY R. AMES, JR.
The MITRE Corporation
Bedford, Massachusetts

and

DONALD R. OESTREICHER
USC Information Science Institute
Marina Del Rey, California

INTRODUCTION

A primary requirement for a message system in an operational military environment is that it be secure enough to process messages at multiple levels of classification. But for the system to be accepted, the operator interface must be “usable.” Usability relates to three things: features provided, ease of entering commands, and overall performance. Certainly an interface that does not perform well in these respects, or is difficult to learn or use, will not be used.

The Department of Defense Advanced Research Projects Agency (DARPA) and the Navy are sponsors of an experiment to evaluate the operational use of a computer-aided message handling system at PACOM Headquarters in Hawaii. The experiment will evaluate the operational and organizational impact of the automated service on a community that now uses a largely manual system. The purpose of this paper is to document the security design of SIGMA,** the system that will be used in the experiment.

In the following section, a description of the SIGMA message processing system is given. The third section provides background and discusses the kernel approach to multilevel security. In the fourth section we describe several security problems encountered in the design. The fifth section presents the design of the SIGMA message service. The additional features that the kernel must provide to support SIGMA efficiently are documented in the sixth section. Finally, a summary is provided to highlight the paper’s main points.

* This research was supported by the Defense Advanced Research Projects Agency under ARPA Order Nos. 3338, 2223 Contract Nos. F19628-78-C-0001, DAHC15-72-C-0508, MITRE Project No. 8010, ISI Project Nos. 3D90 and 3P10. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency or the United States Government.

** The SIGMA message service is one of three services developed during this experiment. The other two are the HERMES system built by Bolt, Beranek, and Newman, Inc. and DMS built by Massachusetts Institute of Technology.

THE SIGMA MESSAGE PROCESSING SYSTEM

Information Science Institute of the University of Southern California developed SIGMA specifically to meet the message handling needs of a military command. SIGMA is a secure interactive message handling system providing computer-aided message handling services for the receipt, filing, retrieval, creation, and coordination of military (AUTODIN) messages. We consider it secure in that it presents an interface to the user constrained to abide by the DoD security policy. It is an interactive system since all user-system communications occur via an on-line terminal with a CRT display. Finally, it is a message handling system because it supports the typical message processing functions needed by any formal organizational operation.

SIGMA supports the full cycle for processing incoming and outgoing messages in a military operation. It provides flexible filing capabilities for on-line storage of all messages. Easy access to messages and files is provided by selective search and retrieval functions. Incoming AUTODIN messages move through the system by informal forwarding or by formal action assignment. Outgoing messages are processed by a set of functions supporting message creation, editing, coordination, release, and post-transmission comeback copies.

SIGMA operates on a DEC PDP-10 computer with the TENEX operating system. AUTODIN messages enter through the local AUTODIN message exchange. SIGMA distributes these messages to all pertinent addressees on the system where each user can access them through his SIGMA display terminal.

THE KERNEL APPROACH TO MULTILEVEL SECURITY

The need to process multiple levels of classified data led the Air Force Electronic Systems Division to sponsor several research and development efforts to build an operating
The SIGMA design has two goals: to produce a certifiably secure service, and to present the user with an agreeable user interface. In many situations these goals are at cross-purposes. Our general approach has been to present the user with a true picture of what is happening, maintain
Overview of the design

When using SIGMA, a user is actually interacting with a collection of up to five processes (see Figure 1). These are the trusted process, an unclassified control process, and one process for each classified level that the user/terminal is cleared to operate. Each process (except the trusted process) can write data only at its own level and can read data at its level or lower.

SIGMA attempts to be as helpful to the user as it can, organizing the user's session and cleaning up the user's state (current context) as necessary. The service also attempts to understand the user's current context and conform its behavior to the situation. For this reason the context information must be available to all user processes; thus, it must be unclassified.

The user's state in SIGMA is divided into two parts. The first part contains the current list of objects being accessed and functions being performed by the user. This portion of the state is maintained at the unclassified level by the unclassified control process. The second part contains the current list of message entries (from the open message file) in which the user has expressed an interest. The entry list information is potentially classified at the level of the file and is thus maintained at this classification level by the appropriate classified process.

This dichotomy of state is reflected directly into the security design. Commands are divided into those which access the unclassified state (unclassified commands) and those which access the entry list (classified commands). The latter group includes both commands that use the entry list for input and those that allow the user to enter classified information as part of the command.

Multilevel terminal

We designed the terminal, used by SIGMA, to enable the user to interact with data at more than one security level at a time. The screen of this "multilevel terminal" is divided into "windows" (see Figure 2), each of which is logically an independent terminal. Each window scrolls independently and may have a different security level. Windows are further divided into domains that have various attributes (e.g., enterable, editable, underlined, etc.). The domain's security level is the same as that of the window.

To keep the user apprised of the level of information he is viewing and entering, we added two sets of lights to the terminal. Each set consists of four lights (one for each security level); one and only one light of each set is on at any time. The first set is mounted on the keyboard; it specifies the classification of the window in which the cursor currently resides. If the user wishes to know the level of any particular piece of information on the screen, he may move his cursor to the information. The second set of lights is mounted next to the screen and specifies the maximum level of information displayed on the screen.

Form of command input

We designed command input so that it could be done through a separate window that is normally at the unclassified level in order to keep the majority of the user's state information at unclassified. Certain commands, such as the
"find text string" command, have potentially classified arguments. For these commands the security level of the command window is raised to the level of the object that the command is affecting before the user enters the parameters.

Strict enforcement of the security model eliminates any possibility of a security compromise: a write-down path through the system that could be used to release information of a higher security level to a lower security level. However, even with the enforcement of the model, there are several situations in a message system where the user, by following instructions given by the system, can inadvertently compromise small amounts of information.

Consider the following example: A user asks for a list of all his messages with a subject having word "x" in them. To perform this operation, the user must be at the security level of the file that he is looking at—greater than or equal to the security level of all the messages within that file. The enforcement of the *-property forces the result of this examination to be at the level at which the examination was performed—the security level of the file. Should the user then decide to perform any modification to a message returned by this examination that has a security level lower than the file security level, the *-property would require him to: issue commands at the security level of the message that he desired to modify, and tell the system the unique identification of the message told to him by the classified process. (The unique identification is required here because the system is unable to pass the desired identifier "down" due to the enforcement of the *-property.) However, this transmission through the user of the message-id from the higher process to the lower process is, itself, a violation of the *-property. Although it is conceivable that a maliciously written program could use this *-property violation to compromise information, we assume that the user serves as an effective filter in this write-down path (both in "bandwidth" and in checking for reasonableness), thereby precluding any reasonable software means of making use of this path.

Because of the hardships that strict enforcement of the *-property imposes on the user and because of the existence of *-property violations, a case can be made to ease the user interface in situations where this type of violation exists. The improvement takes the form of allowing SIGMA, in violation of the *-property rule of the security model, but with user concurrence, to write-down the unique identifier of the message that the user wants to modify. We limit the bandwidth of this type of *-property violation, so that it is no larger than the path that otherwise occurs through actions of the user, by allowing only a specific amount of fixed-format information to be transmitted to a lower security level and then only if the user has depressed an appropriate function key that is linked directly to the security kernel. Allowing the system to transmit this information greatly simplifies the user interface.
Trusted process functions

Certain functions need special capabilities to operate (such as the passing of message identifiers) but are relatively message-system dependent and thus are not included in the security kernel. We group these functions together in a "trusted process" that has the ability to transfer information in a controlled fashion in violation of the *-property.

The trusted process in SIGMA performs four functions: change classification; message release; command completion signals; and entry list transmission.

Change classification

SIGMA allows users to change the classification of text that they are allowed to access. When this happens, the trusted process clears the screen and presents the text in a simple fashion (19 lines at a time) for confirmation. When the user has confirmed the entire object, the trusted process logs this action and passes the text to a process at the new security level for refiling.

Message release

In the military formal messages are released with the commander's signature. Therefore, we consider the act of releasing a message a security event. To control message release, we require that the trusted process insure that the user requesting the release is authorized to release and that this user is making the request from an authorized terminal.

An additional security consideration with message release is that some AUTODIN terminals (ours in particular) treat the message header as unclassified. If SIGMA this header is created in the same window as the message text. Therefore, releasing a message implicitly lowers the classification of the header information. During message release, the trusted process requires the user to confirm that all of the header information is actually unclassified. The trusted process logs this action before releasing the message.

Command completion signals

We have based the SIGMA design on the concept of an unclassified process that receives the majority of the commands, determines the proper security level needed to execute these commands, and then activates a process at that security level to perform the execution. The disadvantage of this approach is that, should an error occur between the unclassified control process and the classified operational process, the classified process cannot ask for clarification. Thus error recovery is difficult. This problem is referred to as the open loop problem.

Presently we believe that the best solution to the open loop problem is to allow the trusted process to close the loop when an error of this type is encountered, provided the user has depressed a function key since the last such request. Closing the loop improves recovery but has an impact on security, since a *-property violation exists when the loop is closed. As in command input, requiring a user action between successive writedowns restricts the bandwidth of this operation.

Entry lists

When a user process needs to write down a list of message identifiers, "entry lists," it passes this list to the trusted process for user confirmation directly. The trusted process checks the format and bounds of the entry numbers, and asks the user to directly confirm the number of entries being processed at each security level. Thus, the user has the ability to directly monitor (and control) the bandwidth of the writedown channel. This separate step is reasonable even from the user interface side, for if the number is too high or too low, the user can see that he specified his request incorrectly.

KERNEL REQUIREMENTS

In order to be able to support the SIGMA architecture, the security kernel must provide certain additional features not found in kernels designed to date, including a terminal multiplexor for the multilevel terminal, a variety of object sizes, the ability to support large numbers of processes, an efficient inter-process communication facility, and a policy that can support "trusted" processes.

Multilevel terminal certification

Since the terminal supports the simultaneous display and editing of data at different classifications, we must demonstrate that the terminal (1) maintains the proper levels for all information it contains (possibly 20,000 characters) and (2) marks all information returned to the computer with the proper security level. It is the terminal's responsibility to assure that no information entered in a window by either the user (doing local editing) or the application computer is transferred to any other window. While at first pass the certification of the terminal may seem trivial, one must consider that the terminal code is currently produced for a single INTEL 8080 and occupies 32K bytes of PROM. Eventually multiple 8080's, application of Denning's flow analysis, or the introduction of a kernel in the terminal will be necessary to guarantee separation of the windows.

Terminal multiplexor

A significant problem is the method for attaching the multilevel terminal to a secure system. We have identified two

* If the entry list has only one element, then the appropriate function key is sufficient and the further "confirm 1 entry" step is omitted. This operation allows the user to point to a single entry or mention it by number or context (CURRENT, NEXT) for a display, reply, file, etc., without being required to do more than use the proper function key to enter or confirm the command.
alternatives: each window could have a unique connection to the system or the kernel could multiplex all information to a terminal over the same communication line. We have chosen the multiplexing approach in order to minimize the number of terminal lines.

Communication between the system and the terminal is in the form of NOTICES and DISPATCHES. The terminal multiplexor must assure that each NOTICE received from the terminal is directed to a user process whose security level is the same as the security level of the window. The multiplexor must also forward Function Key Notices to the trusted process to provide for the capability for a controlled write-down of message identifiers.

The terminal multiplexor must insure the correctness of all DISPATCHES to the terminal. With the exception of “window allocation” DISPATCHES, the terminal multiplexor need only check the window identifier and length to assure that the user process is communicating with a window to which it has access. All requests for terminal window allocations and deallocations must be done by the unclassified user control process. This process provides the terminal multiplexor with the security level for all newly created windows. The unclassified process can, at a later time, request a change in a window classification by notifying the multiplexor. If this new security level is lower than the current window security level, the multiplexor must erase the information currently in the window.

Process structure

The design of the kernel’s process structure will have significant implications for the performance of SIGMA. On traditional timesharing systems, such as TENEX, process creation is expensive, and process swapping is lengthy. In order for SIGMA to operate efficiently the kernel must be able to (1) support large numbers of processes; (2) allow for fast process creation and deletion; and (3) swap processes with little overhead. Large numbers of processes are required because SIGMA requires several active processes per user. If SIGMA is extended to handle compartmented intelligence, fast process creation and deletion would be required. Finally, because large numbers of processes are doing small amounts of processing, process swapping occurs often. To expect a kernel to provide this type of support may require significant hardware support.

Interprocess communication

Equally important to the efficient operation of SIGMA are the speed and types of interprocess communication provided for by the kernel. SIGMA will require the kernel to support both preemptive (interrupt-like) and non-preemptive (message-like) types of interprocess communication. In addition, the latter mechanisms must support small messages for passing message-ids and large messages for transmitting entire command strings.

File system

The file system is often one of the most complex portions of the kernel, a quality which can cause unnecessary overhead. For example, SIGMA does not require the kernel to provide a file organization such as a directory hierarchy. A "flat file" system is entirely adequate and can be used more efficiently. The only special requirements for SIGMA are that the kernel should support both small files (512 bytes) and large files (10K bytes).

System integrity controls

To support SIGMA the security kernel must provide a mechanism that implements the notion of least privilege. This mechanism has been given the name "System Integrity." SIGMA uses three separate privileges: a secure write-down privilege used by the trusted process to reclassify text; a release privilege used to restrict the releasing of messages to a select group; and a system security officer privilege used to initiate and set the security level of new users.

The primary rule that the system integrity controls must obey is: to modify an object or execute a kernel call a subject’s system integrity level must be greater than or equal to the system integrity level of the object or kernel call. There are no rules on reading or executing programs (programs in execution use the system integrity level of the process that they are executing under). We must therefore demonstrate that each subsystem with a system integrity level greater than system low (non-kernel, no privileges) does not execute any programs other than ones that we know execute properly.

CONCLUSION

The design of SIGMA demonstrates that it is possible to build a secure message processing system based on the kernel approach using the DoD security model. We have shown the refinements to the security policy that are needed to achieve a usable interface and have documented the features that a security kernel must provide to support a secure message processing system efficiently. The techniques used in designing SIGMA should be directly applicable to other transaction-oriented or data base management systems.

REFERENCES


