SECURITY TEST AND EVALUATION TOOLS:
AN APPROACH TO OPERATING SYSTEM SECURITY ANALYSIS

Dennis Hollingworth

Steve Glaseman

Marsha Hopwood

September 1974

P-5298

The Rand Paper Series

Papers are issued by The Rand Corporation as a service to its professional staff. Their purpose is to facilitate the exchange of ideas among those who share the author's research interests; Papers are not reports prepared in fulfillment of Rand's contracts or grants. Views expressed in a Paper are the author's own, and are not necessarily shared by Rand or its research sponsors.

The Rand Corporation
Santa Monica, California 90406

SECURITY TEST AND EVALUATION TOOLS:
AN APPROACH TO OPERATING SYSTEM SECURITY ANALYSIS

Dennis Hollingworth

Steve Glaseman

Marsha Hopwood

ABSTRACT

As a result of studies of the security characteristics of selected large operating systems, it has become increasingly evident that any complex operating system requires testing and evaluation in order to validate the functional characteristics of the system and verify claims of improved security safeguards. Furthermore, over the next decade, it is likely that new systems will be subject to continuous testing and evaluation in much the same fashion, and for the same purposes, as are existing systems. As yet, the techniques employed in determining the security characteristics of system software are presently quite primitive, based primarily upon the notion of penetration testing - manually examining system source materials for security vulnerabilities. This suggests the development and refinement of tools and techniques of operating system security analysis. Some of the more desirable characteristics of such tools are explored in this document, and several example tools are described.

PROBLEM DESCRIPTION

The development of a multilevel secure processing environment is of prime importance to current and future operating concepts for military computer systems. Multilevel secure processing is a required operational capability if command and control components are to successfully meet mission objectives while providing requisite protection against compromise, need-to-know violation, denial of access, and data contamination. Lack of such a capability adversely affects the long-range goal of providing full integration of data and systems to promote operational efficiency, flexibility, timeliness, completeness, accuracy, and responsiveness. Significantly, Section 2-4 of the WWMCCS RFP specifically addresses the requirement for a multilevel secure processing capability. It states:

> The requirements of the WWMCCS installations for processing of sensitive and classified data give rise to certain characteristics which must be present in vendor-proposed hardware and nonfunctional software: the control of access to data within the ADP system from consoles, the protection of files in common on-line storage devices, the maintenance of system integrity, the control of output data, the protection against transmission of classified data over unsecure circuits, and the provision for accountability for specified unauthorized program operations.

That the intent of this RFP with respect to data security has not been fully satisfied has been clearly demonstrated by the results of a test and evaluation of the vendor-supplied software in which Rand was involved. Indeed, there appears to be a significant gap between the desired multilevel secure operational capability and the system and data security afforded by the vendor product. Numerous serious security vulnerabilities in the vendor software have been detected and in many cases dramatically demonstrated. Furthermore, at least some of these vulnerabilities were the result of vendor modifications and system enhancements introduced in subsequent system releases and updates. This suggests that vendor-supplied modifications and enhancements as well as major releases of the WWMCCS GCOS III system software will have to be audited for security vulnerabilities. It further suggests that the requirement for system test and evaluation will likely be a continuing one as new software releases and modifications result from the vendor's efforts to satisfy current and future software requirements.

The design and implementation deficiencies identified in the WWMCCS system are typical of those found in other operating systems. Serious system weaknesses have been identified in TENEX, MULTICS, IBM/MVT, and other major systems with minimal investment of resources. Thus, it has become evident that any complex operating system requires testing and evaluation in order to

validate the functional characteristics of the system and verify claims of improved security safeguards. Furthermore, over the next decade, it is likely that new systems will be subject to continuous testing and evaluation in much the same fashion, and for the same purpose, as are existing systems.

The techniques employed in determining the security characteristics of system software are presently quite primitive, based primarily upon the notion of penetration testing. Penetration testing involves (1) manual inspection of system documentation, (2) generation of penetration hypotheses based upon suspected security weaknesses, (3) hypothesis testing via the development of example penetration routines, (4) manual evaluation of the results, and (5) repetition of the preceding four steps as necessary to investigate unsuspected anomalies of system interaction and refine the penetration techniques.

Such an approach is somewhat incomplete with respect to the identification of the system's vulnerabilities and is highly dependent upon the investigator's grasp and understanding of system interactions. Furthermore, the essentially manual techniques employed are quite expensive, and are wasteful of resources - including the human resource. The inefficiency shows up most dramatically in terms of the time an investigator must spend examining, in minute detail, system documentation. Moreover, in order to identify weaknesses, it is necessary for the

investigator to fully understand how a given system component functions and how it interacts with the rest of the operating system. Thus, the quality of the evaluation is constrained by both the time available and the breadth and depth of understanding the investigator has of the system.

In order to improve the effectiveness of the test and evaluation activity, it is important to make better use of the investigator's time by allowing him to focus his efforts more appropriately, extend his awareness and understanding of complex system interactions, and eliminate the necessity for repetitive identification of instances of the same type of security system flaw. To this end, one would like to employ tools and techniques of software system analysis which amplify the investigator's efforts and, to whatever extent possible, automate the process of software test and evaluation. As yet, few such tools of any sophistication exist; those that do exist are not particularly suitable for and were not intended for operating system security evaluation. It would appear that a great deal more can be done in the area of tool development to facilitate operating system analysis. Furthermore, the area appears to be one which can offer tangible, demonstrable results in the near term with respect to increasing our capability for exploring and understanding operating system security characteristics.

RESEARCH AND DEVELOPMENT REQUIREMENTS

During Fiscal Year '74, Rand researchers participated in a effort to determine the security features of the system software for the World Wide Military Command and Control System. Rand's involvement consisted primarily of investigating the security characteristics of that part of the system which serves as an interface between the batch user and the operating system - the Master Mode Entry service routines. In analyzing this code Rand researchers found that their expected effectiveness was diminished by several contributing factors which were concomitants of the evaluative technique:

- o the difficulty of isolating and focusing on only security relevant activity in the analysis of a given service module;

- o the difficulty in tracking parameters through and between modules as they are saved in temporary storage locations, manipulated in registers, used to affect control switch settings, used in the generation of other parameters, etc.;

- o the difficulty in properly investigating the interactions of related modules which do not communicate in any formal fashion, but instead share a

common communications area;

- o the difficulty of fully validating complex algorithms with respect to all possible inputs;

- o the degradation of the effectiveness of the investigator as his sensitivity to program anomalies is dulled by the monotony of the search task.

These difficulties plus an awareness of the fact that much of our time was spent in activities which were repetitive and relatively straightforward in nature lead us to suggest the development of tools for operating system analysis which make better use of that relatively scarce human resource (individuals knowledgeable in the security characteristics of an operating system) and which facilitate the process of security test and evaluation.

In large part, the process of diagnosing the security vulnerabilities of an operating system is suggestive of the program debugging task, the difference being primarily one of scale. In the former case, a relatively small block of code is under investigation; in the latter case as much as one to two hundred thousand instructions must be considered. To the extent that the tasks are similiar in objective and that the operating system comprises a collection of functionally independent modules, however, the same techniques may apply to both areas.

Some tools already exist which might prove to be particularly useful. Companies such as TRW, General Research Corporation, and Logicon have already done a great deal of work in the area of verification, validation, and certification of software packages. However, most of the tools which have been developed are proprietary and have not been made available to the government. Furthermore, they are often specific to a particular language or system. To the extent that these tools or the techniques they incorporate are available, it would be desirable to incorporate them in more generalized tools oriented toward the analysis of operating system software.

There is a wide variety of tools which might be developed reflecting a broad spectrum of capabilities and operational characteristics. Based upon our previous experiences with operating system utility routines, our knowledge of techniques of evaluation employed in the analysis of small software packages such as utilized in missile guidance and targeting systems, and our ideas about what facilities might be particularly useful in the evaluation of operating system security characteristics, we would expect tools which might be developed to exhibit some combination of the following functional capabilities:

1. controlled program execution,

2. control/data flow mapping,

3. automated module exercising,

4. heuristic module analysis.

Controlled Program Execution

Controlled program execution is a standard approach to diagnosing and eliminating the errors in computer programs, i.e., debugging a program, and is characteristic of the function of dynamic debugging tools. Indeed, the objective of such tools is to accomodate the controlled execution of program elements by facilitating the run-time introduction of execution break-points and permitting user-directed alteration of control flow independent of the normal execution logic of the program. Further, they generally permit the run-time interrogation of program variables and constants, alteration of program variables, automatic display of control flags, and the like. Such a capability allows the investigator to dynamically study the execution characteristics of discrete program elements and, consequently, lessens the need for exhaustive static analysis of program execution logic, program algorithms, etc.

Capability comparable to that found in program debuggers, but redirected and enhanced as necessary to bridge the gap between debugging discrete programs and debugging an operating system, would be particularly useful for the operating system test and

evaluation activity. Specialized tools of this nature already exist in some operating systems (e.g., BBN's TENEX and DEC's TOPS-10 system) and have been demonstrated to be invaluable in eliminating the errors in those systems. [1-4]

Control/Data Flow Mapping

Determination of the security characteristics of a system module from the source code representation of that module requires that the investigator be cognizant of and understand the various control paths through the module as well as those conditions which cause a given control path to be executed. Furthermore, it is often necessary for the investigator to track parameters through modules as they are saved in and retrieved from temporary storage locations, manipulated in registers, passed to and from other modules, and the like. Such requirements suggest the need for readily available software tools which largely automate the development of control transfer maps and parameter usage maps and permit their logical integration and representation in the same diagram. To some extent such tools already exist, although they are often proprietary and in general unavailable for use by the government. [5-6]

Tools which provide such information can be generally useful in the manual analysis of system modules. Furthermore, such a capability is a prerequisite for the development of more advanced analytic techniques.

Functional enhancement of such tools might facilitate the production of module interaction profiles which illustrate both the explicit and implicit interactions between modules resulting from the transfer of control, the passing of parameters, and the shared use of information in tables and communication areas.

Automated Module Exercising

In evaluating the security characteristics of an operating system one is often more concerned about unusual execution conditions and their effect on other system components than the standard execution paths that have, for the most part, been well exercised and debugged through the running of the operating system itself. Thus, one is interested in determining that all significant control paths have been exercised under all parameter combinations. A useful approach found in industry and used for similiar purpose is exemplified by automated module exercisers - utilized, for example, in the validation of missile guidance packages. Such tools are employed to automatically exercise a specified program module according to some user-determined parameter constraints, thereby eliminating much of the monotony associated with manually directing program execution activity. Unlike software debugging tools, the investigator's involvement is required only when an execution environment is established, when an execution anomaly is detected, and when the exercise is complete.

Module exercisers provide information about which, if any, instructions or groups of instructions have not been exercised for a given set of parameters, and may additionally provide statistical information about the frequency of execution of selected statements. Such information is particularly useful in determining that all instructions have been executed and, thus, to some extent tested, and in determining that a module will not behave abnormally under some atypical parameter set or unusual run-time conditions.

Tools of this nature already exist. [7-9] General Research Corporation, for example, has one for FORTRAN programs. Others have been developed for particular assembly languages. However, it is desirable to investigate the generality of such tools with the objective of making them available for use with a variety of languages and systems.

Heuristic Module Analysis

The diagnosis of more complex or heretofore unrecognized security vulnerabilities may require a more heuristic approach to module analysis. Such an approach, however, presupposes a more focused module representation than is provided by a module source listing because a source listing: (1) contains much information which is extraneous to the security characteristics of the module and merely distractive in nature, (2) presents only the physical

organization of the module and does not clearly indicate the logical organization of the module, and (3) is not optimally structured for computer-assisted analysis. Thus, a requisite functional capability is the computer-assisted production of machine-readable representations of the security-relevant activities of system modules.

With an appropriate machine-readable module representation the investigator could interactively explore the module(s) for control paths which satisfied selected constraints on security-relevant actions inasmuch as he perceived such paths to represent potential security vulnerabilities. Output from such activity might consist of a list of all control paths through the module(s) which satisfied the investigator-imposed path-constraints and, for each such path, a list of all ancillary conditions which must also be satisfied. Such information would permit the investigator to study more complex security interactions within and between modules and determine whether suspect interactions actually constituted security flaws by virtue of the ancillary conditions which must also be true. Further refinement might permit semi-automated analysis of the operating system representation for previously identified generic vulnerabilities. At the direction of the investigator, the machine-readable operating system representation might be scanned for instances of security vulnerabilities described algorithmically in a library of error patterns. Diagnosed instances of suspected errors would be noted

for subsequent analysis by the investigator. Such capability would free the investigator for the study of new error types and the analysis of automatically diagnosed errors, in large part alleviating the drudgery associated with operating system security analysis and optimizing the use of the human resource.

EXAMPLE TOOLS

The following two examples reflect our initial thoughts on the form of some specific tools which might be developed. They are included to indicate how the functional capabilities described above might be employed in particular tools designed for operating system security analysis. They do not in any way constitute an exhaustive set of examples.

The first example represents a tool which might be produced in the near term with something on the order of 2 to 3 man-years of effort. The effort would capitalize on existing methodology in the area of program debugging to produce a tool which would contribute to the investigation of certain types of security vulnerabilities such as those resulting from nonexistent or inadequate parameter checking.

The second example represents a more significant developmental effort but would offer potentially greater payoff. Difficult problems would have to be solved with regard to isolating, identifying, and characterizing security vulnerabilities in terms of module logic and execution activity. It would be necessary to develop an appropriate intermediate representation format for system software which to some extent reflected program semantics, a translator which facilitated the translation of the system source representation into the intermediate representation format, and pattern matching programs which searched the intermediate representation for instances of security vulnerabilities. Some preliminary work which employs similar techniques has been done in dealing with programming language semantics and planning in computer problem solving. Some transfer may be possible. In any case, the developmental effort could be undertaken in such fashion that it yielded near term payoff in terms of partially-automated analytic capability.

Example #1

Interactive Aid to Test and Evaluation

Description

The Interactive Aid to Test and Evaluation (IATE) is

visualized as a software tool designed to achieve the following specific objectives:

1. Greatly reduce the amount of time the investigator must devote to manual examination of system documentation.

2. Aid the investigator in identifying the areas of highest vulnerability within the operating system.

3. Ease the procedural characteristics of the test and evaluation activity by providing a simple and well-defined user/tool interface.

The IATE would consist of three functionally distinct components. These components would allow the specification, parameterization, and monitored interactive execution of a partial reproduction of the operating system under consideration. Such partial reproductions would consist of selected operating system modules retrieved from system libraries, modified appropriately as required by the IATE, and linked together in an executable form that mirrors the linking of the chosen modules within the actual operating system. In this fashion the IATE is expected to facilitate the examination of module interactions at two levels of analysis:

1. The preliminary search level, which aims at locating general areas of possible weakness in module interaction.

2. The level of detailed examination of selected module interactions aimed at testing specific hypotheses related to system vulnerability.

The functional components are broadly outlined below:

Specification. An interactive program, based on currently available software technology, would display to a user at a CRT console a list of the major subsystems included in the operating system. The user, having selected a subsystem, is then shown a list of the system modules comprising the subsystem, and may select from the list a group of modules whose interactions are of interest. The modules selected are retrieved from source libraries, and scanned for linkage information -- including requirements for other system modules not specifically belonging to the subsystem (e.g., a system service routine). It is the user's option to include these ancillary modules along with those previously selected, or, with the assistance of the IATE, generate "black box" routines in the appropriate linkage positions. (Black boxes are pseudo-modules which simulate the performance of the module they represent by

time delays, returned parameters, etc.) At this point a block diagram showing selected system modules, black boxes, and connective paths is generated and displayed on a CRT screen or other display medium.

Parameterization. An interactive program would allow the specification of execution parameters. Such parameters would supply, as a minimum, the following capabilities:

1. Specification of input parameters required by the system modules under investigation. Frequently the root of a system's vulnerability lies in the quality of the processing devoted to input parameters. The IATE would allow a user to easily specify or re-specify these parameters as the diagnostic session proceeds.

2. User-specified execution control points. The IATE would allow flagging of specific points within the constructed model which, when encountered during execution, would cause the execution monitor to temporarily suspend execution and return control to the user. At these points the user may change the values of parameters, insert and/or delete control points, or modify the status of the trace facility, described next.

3. An interactive module and parameter trace program would provide inspection and modification facilities for system parameters and/or variables, and would be accessible for modification at the control breaks encountered during execution.

User specification of any of these capabilities would be implemented by additions to the source code, made by the IATE, in the appropriate modules which reside in the user's work space.

Monitored Execution. This interactive execution monitor would perform two passes on the specified and parameterized modules:

1. Pass 1 would assemble and link the model and its associated "black boxes," control point flags, and trace facility code inserts, into an executable form.

2. Pass 2 would submit the user-specified input parameters to the model, and allow execution to begin. The model would execute as a subsystem under the control of the monitor. The monitor would regain control as dictated by imbedded trace operations and/or control point specifications. When the

monitor regained control, the user would be given an opportunity to examine parameter values and system variables. Based on such information, the user could elect to continue execution of the model to the next break point, or to reenter earlier phases, specify changes to the model, and re-initiate execution.

While freeing the investigator from the less productive aspects of diagnosis, test, and evaluation activities, the IATE would take good advantage of his skill and experience by supplying a simple, easy to use, and powerful modeling tool. The IATE is expected to increase investigator productivity, afford a means of gaining valuable experience on the systems to which it is applied, and allow more comprehensive evaluation of data security software features in a time frame competitive with that of current manual techniques.

Example #2

Interactive Security Topography Analysis Tool

Description

A significant part of the WWMCCS security analysis study was

devoted to the investigation and characterization of intra- and inter-module security interactions. To facilitate this effort, we developed a representation format we call "Focused Flow Diagrams." These diagrams incorporate as objects those activities in a given module perceived by the investigator as performing security-related functions. Also represented are all control paths connecting these "security-objects" thereby permitting the investigation of more complex security interactions resulting from the interrelationship of security-objects. Security-objects include operations such as: the validation of a user-supplied parameter, the enabling of program execution, the use of the user's address-space as a work area by system routines, the relinquishment of control pending completion of an outstanding I/O operation, the transfer of control to other modules, and the like. Such diagrams, we believe, permit the investigator to better understand and conceptualize some of the security-relevant interactions which take place within and between modules by allowing him to focus and concentrate his analytic capabilities on a narrow subset of the module's activities.

There are several deficiencies in the analytic technique as it is presently employed, foremost of which is that the generation and analysis of Focused Flow Diagrams is performed manually and demands considerable time and perseverance. Furthermore, even these focused diagrams may become relatively complex in structure, requiring substantial effort on the part of the investigator to

study certain types of interactions. Moreover, to some extent the generative and investigative activities are mutually-feeding, with the generative process determining which types of security-objects can be investigated, and the investigative process identifying other or additional security-objects which should be considered. However, the present method of generating FFDs does not afford the desired degree of flexibility and adaptability required by such interrelationship of the generation and analysis processes. This suggests the development of the Interactive Security Topography Analysis Tool (ISTAT), described herein, a tool designed to satisfy the evaluative requirements suggested by the use of FFDs, but in a more generalized and semi-automated fashion.

The ISTAT would consist of two functional components: a component designed to accomodate generating and editing focused module representations, and a component designed to facilitate analysis of such focused representations for investigator-determined module activities. The objective is to provide a semi-automated, flexible, rule-driven tool with which to construct, investigate, and manipulate functionally-focused representations of operating system activity to gain insight into security-relevant activities and identify instances of known as well as new types of security vulnerabilities.

Focused Representation Generating and Editing:

The production of a focused module representation is currently a rather laborious task essentially equivalent to flowcharting a module (but with a more focused objective) from a set of source listings. This suggests the development of a mechanism for generating and editing machine-readable focused module representations in a fashion which minimizes the demands on the investigator's time and resources. Three capabilities are indicated. The primary capability is that of semi-automatically generating a desired focused module representation from the module's source representation according to a set of specification rules entered by the analyst which identify the types of elements to be included in the representation, the way they are to be identified in the source representation, and the way they are to be represented in the focused module representation. After initialization is completed, such activity would proceed automatically until interaction with the investigator is required to resolve some difficulty in the source code analysis.

The second capability is that of manipulating and editing the focused module representation according to commands entered by the investigator. The third capability is that of displaying the machine-readable representation on

an external display medium to facilitate study of the focused module representation.

The internal representation would most likely be that of a specialized form of directed graph which carries along attribute and descriptive information for objects and control paths. This representation would be amenable to search and analysis by other tool elements and would facilitate the generation of graphical output.

Focused Module Representation Analysis:

A primary objective of developing focused module representations is to facilitate the investigation of more complex security interactions than can be practically examined by conventional means. Such investigation would, for example, contribute to the identification of security vulnerabilities and a set of "security-primitives" which can be used to describe those vulnerabilities. These primitives would reflect activity content rather than form, thus eliminating module idiosyncrasies which may be evident in security-objects, and would be useful in describing the majority of identified security-relevant operations and hence, security vulnerabilities which occur in operating systems. (Security-primitives might include such operations as: retrieval of a parameter from the user's address-space,

bounds checking the location of the parameter, bounds checking the value of the parameter, storing the parameter in the system's address-space, enabling I/O interruptions, halting user I/O activity, and other such actions.)

The focused representation analyzer would utilize as input both the machine-readable focused module representations created as described above and a set of rules describing a particular set of objects to be investigated and the way in which their characteristics are to be studied. For example, a path-constraint string might be defined by the investigator consisting of a string of security-objects separated by boolean operators and grouped by precedence relationship. The string describes to the analysis tool a possible path(s) through the module(s) which the investigator suspects yields a security flaw by virtue of the interaction of security-objects on the path. The analysis tool might search the encoded focused module representation, utilizing the constraints described in the string, to determine if the desired path existed. Output might consist of the set of all paths which satisfy the path-constraint string and all ancillary conditions which must additionally be satisfied for each specified path.

As an appropriate set of security-primitives is identified and a working set of error descriptions developed, a library

facility might be added to the focused representation analyzer. The library would include all known error patterns and the working set of focused module representations. The library could be periodically augmented with any additional error patterns identified through use of the analytic tool or other means. At the direction of the investigator, a specified set of encoded focused module representations would be analyzed for instances of such error patterns and all suspected occurrences reported.

REFERENCES

1. Myer, T., J. Barnaby, W. Plummer, "TENEX Executive Language," Bolt Beranek and Newman Inc., NIC-16874, April 1973

2. Leavitt, E., et. al., "TENEX User's Guide," Bolt Beranek and Newman Inc., January 1873

3. Bobrow, D., et. al., "TENEX, A Paged Time Sharing System for the PDP-10," Bolt Beranek and Newman Inc., BBN Report No. 2180, August 1971

4. "DECsystem10 Assembly Language Handbook," Digital Equipment Corporation, DEC-10-NRZA-D, 1972

5. Brown, J. R., "Practical Applications of Automated Software Tools," TRW Systems Group, TRW-SS-72-05, September 1972

6. Miller, E. F., M. R. Paige, "Automatic Generation Of Software Test Cases," General Research Corporation, May 1974

7. Brown, J. R., R. H. Hoffman, "Evaluating the Effectiveness of Software Verification - Practical Experience With an Automated Tool," TRW Systems Group, TRW-SS-72-08, December 1972

8. Rubey, Raymond J., B. Dulac, "Software Tools for Certifying Operational Flight Programs," Proc. National Space Navigation Meeting, 1967

9. Henderson, V., "Program Validation", Logicon Corp.