Formal Specification and Verification of Control Software for Cryptographic Equipment

D. Richard Kuhn and James F. Dray

National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, Md. 20899

ABSTRACT

This paper describes the application of formal specification and verification methods to two microprocessor-based cryptographic devices: a "smart token" system that controls access to a network of workstations, and a message authentication device implementing the ANSI X9.9 message authentication standard. Formal specification and verification were found to be practical, cost-effective tools for detecting potential security weaknesses, and helped to significantly strengthen the security of the access control system.

1. Introduction

Microprocessor-based systems are increasingly being used to provide improved security. The improvements in security are often accomplished at the cost of increased complexity, as when a smart card microprocessor replaces a simple password system for network access control. Formal methods are recognized as an effective means of assuring the security of systems, and have been used in several military security applications over the past 15 years [Neumann *et al.*, 1974; Tagney *et al.*, 1977; Feiertag *et al.*, 1977; Neuman *et al.*, 1980; Young *et al.*, 1986; Levin *et al.*, 1989]. This paper reports on the application of formal methods to two civilian security-critical systems: the NIST Token-Based Access Control System (TBACS), a "smart token"[1] system that controls access to a network of workstations, and a message authentication device implementing the ANSI X9.9 message authentication standard [ANSI, 1986]. A state-based specification was prepared for the smart token system. The message authentication device specification used the notation of the Vienna Development Method.

The projects were undertaken primarily as exercises in preparation for a larger project that is planned, but the results surpassed the initial goal of gaining familiarity with verification tools. It is noteworthy that no funding was available for formal methods work in either case. A verification tool, Unisys' Formal Development Methodology (FDM) [Eggert *et al.*, 1988], was obtained at no cost and the formal methods work was done as time permitted. Even with limited time available, we found the effort worthwhile. In the smart token access control system, several inconsistencies were found that led to improved security. In addition, a subtle error was discovered that could have compromised the security of TBACS, had it been released. A breakdown of hours and resources used in the access control system verification is given in section 2.8. The most interesting result of this work, beyond the increased assurance for TBACS security, is that it gives additional evidence that formal methods can be successfully applied to "real world" problems. Formal methods are rarely used today and are often rejected out-of-hand as being too difficult or expensive. Our experience has convinced us that, at least for small projects, or for small portions of large systems, formal methods are a practical and cost-effective adjunct to traditional software engineering methods.

2. The Token Based Access Control System

2.1. System Description

The Token Based Access Control System (TBACS) was developed as an experimental system to replace traditional password based systems. Based on the TBACS proof-of-concept, a Smart card based Access Control System (SACS) that incorporates the TBACS design and

[1] Strictly speaking, a *smart token* is different from a *smart card*, although the two terms are often used interchangably. Both are hand-carried devices containing microprocessors and memory, but there is an ISO standard for smart cards. A smart token is typically larger than a smart card.

code is now under development. TBACS uses a portable device called a smart token to control access to the resources of networked computer systems. The TBACS smart token performs cryptographic authentication to identify the user and up to 100 computers which the user wishes to access.

The system configuration for TBACS consists of a number of workstations and host computers interconnected by a communications network. Each workstation on the network is connected to a reader/writer device, which provides the electrical interface between the TBACS token and the workstation. When the user inserts a token into the reader/writer, a program running on the workstation manages the authentication process by issuing a sequence of commands to the token and receiving the token's responses to these commands.

2.1.1. Hardware

The smart token consists of a plastic carrier containing a microprocessor and non-volatile memory. The carrier has the same major dimensions as a standard credit card, with six recessed metallic contacts along one edge. The reader/writer connects to the workstation through a standard asynchronous serial communications port, eliminating the need for a custom communications interface.

2.1.2. Software

The TBACS token responds to a set of 17 commands (see Table 1), which are implemented in firmware stored in the token's non-volatile memory. The firmware code is approximately 2,600 lines of C. The sequence in which these commands are executed is controlled by a set of flags which are checked at the first step of each command. If the flags are not set correctly, the given command will not be executed and the token will return an error code.

The commands are grouped into three general classes: security officer (SO) commands, user/workstation authentication commands, and user/remote host authentication commands. The SO commands provide for the initialization of new tokens by loading host IDs, cryptographic keys, and PINs. The token is ready to be issued to the user after the SO has completed this initialization process. The remaining commands implement the authentications required by TBACS to control the login process.

Table 1. TBACS Commands

Command	Verified
Reset	no
Enter SO PIN	yes
Authenticate SO	yes
Enter User PIN	yes
Load Key	yes
Authenticate Token	yes
Generate Challenge	yes
Authenticate User	yes
Change Token PIN	yes
Workstation Verify and Respond	yes
Output ID Table	no
Host Verify and Respond	yes
Read Zone	no
Write Zone	no
Append Zone	no
Call DES	no
Test	no

2.2. Authentication Processes

For a user to gain access to computing resources on a network using TBACS, a series of authentications between the smart token, the user, and various host computers must be performed. TBACS selectively controls access to all computers on the network, including the user's local workstation. By taking advantage of the processing capabilities of the smart token, the login process can proceed transparently to the user while providing a high level of authentication. The DES algorithm, operating firmware, and critical data are stored internally on the smart token, providing a higher level of security than systems which use tokens only as data storage devices.

2.2.1. User/Token Authentications

When a user begins the login process on a workstation, he or she should have some means of determining the identity of the token. A program called the "login manager" is executed on the workstation when the user initiates a login, and is responsible for mediating the required series of authentications between the user, the token, and the workstation. First, the user must prove his or her identity to the token. The next step performed by the login manager is to request the token identification number from the token and display it on the user's screen for visual verification. The user can choose to either continue the login process or abort by simply pressing a key. The login manager prompts the user for his or her PIN/password, which is then encrypted and transmitted to the token along with the user ID. The token decrypts the user PIN and uses it as the key to encrypt the user ID. The result is then compared to the value stored on the token, and if these values match the token accepts the identity of the user. From this point on, TBACS uses the token to represent the user's identity for the remaining authentications.

2.2.2. Three-Way Handshake Protocol

Once the previous steps have been completed, the token and the workstation must authenticate to each other. This is accomplished through a three-way handshake protocol which allows each party to prove that it posesses the same cryptographic key as the other party, without having to physically exchange keys [NIST, 1988]. This protocol works as follows:

1 Party A generates a 64-bit random number and transmits it to party B.

2 Party B encrypts the random number using its secret key, generates a second random number, and transmits both values to party A.

3 Party A decrypts the first number and verifies the result. Party A then encrypts the second random number and transmits it to party B.

4 Party B decrypts and verifies the second random number. At this point, each party is satisfied that the other party posesses the same secret key.

2.2.3. User/Workstation Authentications

After the user and token authenticate to each other, the token must authenticate to the workstation. To perform the authentications between the workstation and the token, the login manager requests a random number from the token. The three-way handshake then proceeds with the token acting as party A and the workstation as party B. If this handshake is completed successfully, the login manager terminates and the user is logged in to the system.

2.2.4. User/Remote Host Authentications

At some point during a session, the user may decide to connect to a remote host via the network. The user activates an rlogin manager, which requests a table of the allowed TBACS hosts for this user from the token and displays this table in a menu format. After the user selects the desired remote host from this menu, the rlogin manager connects to an rlogin server on the remote host. At this point, the local rlogin manager acts primarily as a communications path between the token and the remote rlogin server. The token is provided with the host ID, which it uses to select the proper key for subsequent cryptographic operations. The steps of the three-way handshake are repeated between the token and the rlogin server on the remote host, and finally the rlogin server terminates and the standard rlogin process connects the user to the remote host.

2.3. Token Deactivation

In addition to sequence control, the TBACS token is capable of deactivating itself after three failed login attempts or when the token expiration date is reached. Deactivation is accomplished by deleting the internal token identification number, after which none of the authentication steps required for user login will execute. A token is reactivated when a security officer installs a new token identification number.

2.4. Key Management

When a user first enrolls on a TBACS computer system, the user must contact the appropriate security officer for that computer. The SO initializes a blank token by loading the following: the security officer's ID, encrypted under the security officer's PIN; the user's ID, encrypted under an initial user PIN; a token identification number; and the token expiration date.

The SO next generates a DES key which is loaded onto the token. The random number generation capability of the security officer's token can be used to generate these keys. The token encrypts this key using the user's PIN and stores it in the key table along with the computer's host identification number. The host computer can generate this key from the user's PIN and the host master key as required during future login processes. As an alternative, the DES key could be stored in the computer's key database indexed by the user's identity. After receiving the token from the SO, the user may change the token identification number and the user PIN by entering the current values.

The user may now enroll on another TBACS computer by contacting that computer's SO, who generates another DES key which is stored on the token and the host computer as previously described. The

TBACS token is designed so that only the SO who first initialized the token can delete token keys. Other security officers can only append keys to the token key table.[2]

In order to activate the token during a login, the user must supply the correct user PIN. Once activated, the token can be used to authenticate the user to the user's workstation and then to other host computers by means of the three-way handshake previously described.

2.5. Development

TBACS is a small but reasonably complex embedded system containing custom hardware. It was developed at NIST primarily as a proof-of-concept for the Smart card based Access Control System. Initially, a software simulation of TBACS was written to serve as a prototype. Experimentation with the prototype resulted in several design changes that were later incorporated into TBACS. The prototype also served as a specification for TBACS functions. Because of hardware requirements, most of the simulation code could not be used in the TBACS implementation. SACS, however, does incorporate almost all of the TBACS code. For this reason, the formal specification was based on the design as reflected in the TBACS code.

The formal specification and verification were done after the TBACS hardware and software had been implemented because, as noted earlier, formal verification was not initially part of the development plan. Fortunately however, we were able to complete the verification before the implementation of the Smart card based Access Control System, allowing a problem detected in the formal verification to be corrected in the SACS implementation.

2.6. Security Policy

Generally accepted practice for developing trusted systems requires the statement of a security policy that describes the security properties of the system [NSA, 1985; Tavilla, 1986; Bell, 1988]. A formal model defining the meaning of the security policy in terms of mathematical logic can then be constructed. Confidence is gained in the security of the system by showing that it implements the requirements of the model. When a formal top-level specification of the system is prepared, its consistency with the model can be shown by rigorous mathematical argument. Proofs of lower level specifications and of the code may be formal or informal, depending on the complexity of the system and the resources available. Showing the consistency of the model with the policy statement is necessarily an informal process.

A formal model must be oriented toward a particular class of systems [Nessett, 1986]. For example, a model prepared for an operating system is not appropriate for expressing the security requirements for a network. Significant work has been done on the definition of formal models for multi-level secure operating systems [Bell and LaPadula, 1976; Feiertag *et al.*, 1977], and for trusted networks [Gove, 1985; Freeman *et al.*, 1988;]. Integrity models, such as those of Biba [1977], Lipner [1982], and Clark and Wilson [1987] are more directly related to TBACS verification requirements, but even these are not completely appropriate, so we developed a model that is particular to the requirements of TBACS.

Figure 1 summarizes the rules of operation that were originally defined as the security policy for TBACS, detailed in Dray *et al.* [1989] and Smid *et al.* [1989]. The original security policy was developed informally. The formal specification effort was started later. Initially we derived mathematical statements of the assertions given in Figure 1. However, it was not immediately clear that the conjunction of these assertions would guarantee the security of TBACS. For a greater degree of assurance, a more rigorously developed model of the security policy was required. The goal of this model development was to prepare a formal statement, P, of the security policy at a sufficiently abstract level that its security would be clear. Detailed assertions, $A_1, \ldots, A_n$, such as those in Figure 1, could then be stated and the model of TBACS functions shown correct with respect to these detailed assertions provided that $A_1 \& A_2 \& \ldots \& A_n \Rightarrow P$. This model and its derivation are documented in the next section.

[2] Anyone can in fact append keys to the key table. This somewhat suprising feature was determined to be a reasonable design tradeoff. Security officers maintain control over the keys for their systems, and a user must have a valid key to access a particular host. A user can append a key, but it will be of no use unless it is the correct one that is controlled by the security officer. An alternative to this design would be to have each security officer store an encrypted secret key on the token, but this would require the token to be initialized by up to 100 security officers, since it is not known in advance which hosts a user will eventually need access to. Another alternative would be to have a "master key" that could be used by any security officer. But such a key would add little security, since a key known to over 100 people would likely be leaked in a short time in a civilian environment, where there are no criminal penalties for disclosure of confidential information.

1: Token commands may only be executed in legal sequences: user authorization; token authorization; workstation authorization; remote host authorization.

2: A token deactivates itself when its expiration date is reached.

3: A user must enter correct ID and PIN to be authorized:

4: A token deactivates itself after three failed login attempts

5: A deactivated token will not permit login.

6: A token allows a user access only to hosts whose ID and key are stored on the token.

7: The user cannot open the token if fail limit exceeded or token expired.

8: The user cannot get SO privileges.

9: An SO must enter correct ID and PIN to be authorized:

10: Only an SO may initialize a blank token.

11: A PIN for a particular token may be changed only by an SO or by the owner of that token.

12: After an SO has initialized a token, only this SO can enter the user PIN.

13: After an SO has initialized a token, only this SO can reactivate the token after it has been deactivated.

14: After an SO has initialized a token, only this SO can delete a key from the token.

15: Only the SO can change the expiration date.

Figure 1. Security Assertions

2.7. Security Model

This section describes the derivation of the formal statement of security policy. In summary, TBACS security is defined as the conjunction of the following conditions:

1. *Access control:* Access to the network is granted only if the user posesses the correct PIN and a valid token. Ensuring this condition holds requires condition 2.
2. *Change control:* An invalid token cannot be made valid by the user, only by the security officer. Ensuring this condition holds requires condition 3.
3. *Privilege control:* A user cannot gain security officer privileges through manipulation of TBACS functions.[3]

2.7.1. Terms

The primitive terms shown in Table 2 are used. In the remainder of the paper, the symbols &, |, ¬, => represent *and, or, not, implies,* respectively. The notation x' indicates the value of variable x after a state transition. The universal quantifier is denoted by **A** and the existential quantifier by **E**.

2.7.2. Formal Statement of Model

2.7.2.1. Access Control

Access to the network is permitted only if the user possesses a PIN which encrypts the user ID to the value stored on the token, and the token is valid. That is,

(1) access => $E_{pin\_in}(id\_in) =$ user_pin & token_valid

where $E_K(I)$ represents the encryption of I with key K. Access is defined as authorization of remote host, workstation, token, or user. The token is valid when the token has not expired and is active, the failure limit has not been reached, and the workstation ID is in the token's host table. Substituting terms for these conditions into invariant (1) gives

(2)

remote_host_authd | ws_authd | token_authd | user_authd
=> $E_{pin\_in}(id\_in) =$ user_pin &
today < exp_date &
fail_log < 3 &
token_pin ≠ null &
ws_id ∈ host_ids

2.7.2.2. Change Control

Invariant (2) must be maintained across state transitions. If the user could change the variables that determine if the token is valid, an invalid token could be made valid illegitimately. Thus for each variable in the definition of token_valid, we must define the conditions under which its value can change:[4]

[3] Note that this refers only to user actions within the system, and does not deal with actions that are beyond the control of TBACS, such as the user observing the security officer's PIN being entered, which is a separate concern.

[4] Recall that no restriction is placed on the addition of keys to the host table, as explained in Key Management, Section 2.4. Security in this case is external to TBACS and relies on the security officers for the different hosts maintaining confidentiality of keys.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.