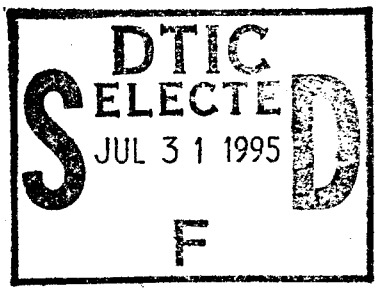
RL-TR-95-92
Final Technical Report
June 1995

RESEARCH ADVANCES IN HANDLING ADAPTIVE SECURITY



Odyssey Research Associates, Inc. (ORA)

Geoffrey R. Hird, Daryl McCullough, Stephen Brackin, and Doug Long

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-95-92 has been reviewed and is approved for publication.

APPROVED: John C. Faust

JOHN C. FAUST
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1995	Final May 94 - Oct 94

4. TITLE AND SUBTITLE

RESEARCH ADVANCES IN HANDLING ADAPTIVE SECURITY

5. FUNDING NUMBERS

C - F30602-94-C-0111
PE - 33140F
PR - 7820
TA - 04
WU - PB

6. AUTHOR(S)

Geoffrey R. Hird, Daryl McCullough, Stephen Brackin, and Doug Long

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Odyssey Research Associates, Inc. (ORA)
301 Dates Drive
Ithaca NY 14850-1326

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory (C3AB)
525 Brooks Rd
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-95-92

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: John C. Faust/C3AB/(315) 330-3241

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Static computer security policies may sometimes be inadequate for two reasons: (1) the high-level objectives of the security policy, and the approach to enforcing that policy, may change over time; and (2) the computer system itself may change its structure or configuration. The goal of this project was to study dynamic security that takes into account these two kinds of changes. The report gives the results of our study of these issues. We address the fundamental conflict between functionality and security that arises when the security policy must change dynamically. We suggest mechanisms for implementing dynamic security policies, and methods for analyzing the consequences (dynamic lattices). We introduce "task-based" dynamic policies. We present a foundational model of need-to-know. For systems that must adapt and change their configurations dynamically, we identify a way of decomposing an adaptive system that provides a systematic way of analyzing its security and ensuring that security is maintained after and during adaptations. We describe a method for performing security risk analysis of an adaptive system. We sketch a way of providing tool support for the risk analysis.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Computer security, Multilevel security (mls), Adaptive security, Security policies, Adaptive security policies, (see reverse)	64
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

14. (Cont'd)

Need-to-konw, Dynamic security lattices, Task-based access control policy and risk assessment

Contents

Chapter 1

Overview

The Problem

In certain circumstances, such as a crisis, a system may need to change its security policy dynamically. This includes two different aspects of a security policy: access control, and what mechanisms enforce security constraints.

There are problems associated with each aspect. For the first, suppose (in a crisis, say) we suddenly need to let a person see more than they are cleared to see. With respect to the usual static sort of policy, allowing the person access is a violation. We want to be able to allow this sort of change, however, so we need a wider framework in which to place such scenarios. Moreover, when we do change the policy in such a way, we are very concerned with *how* we do it — we want to do it so as to minimize the undesirable consequences. In Task 1, we address these two concerns.

The second adaptive security aspect deals with the enforcement of security. Suppose a system switches to "crisis" mode of operation in which weaker or different security enforcement is used. It now becomes a problem to ensure that all needed mechanisms are in place. The problem here is to analyze and control the system despite its complexity. Task 2 addresses this problem.

Fundamental to any secure system is risk analysis. In Task 3 we address the problem of risk analysis for adaptive systems as studied in Task 2.

Achievements

The subject of this project is a new area, at least to the literature. The aims of the project were to take a broad view of the area, survey the basic issues, and outline practical solutions to the various problems. We met these aims, and furthermore made some foundational advances in a new area (need-to-know).

On Task 1, we discuss the fundamental security vs. functionality trade-offs that must be made in changing conditions. We suggest mechanisms for implementing dynamic security policies. We suggest methods for analyzing the consequences of such policies (dynamic lattices). We identify other key areas (recovery, auditing). Also on Task 1, we determined that the basic motivation behind the accesses granted in an adaptive security policy is the tasks that need to be performed. Accordingly, we introduce task-based security policies (related to, but different from, role-based access control). This sort of policy provides an organized presentation of what needs to be be done, taking into account the security requirements of a system.

In addition, we produced new foundational work on Task 1. Much theoretical (and practical) work has been done in the past on how to define and implement security policies, and especially on how to analyze the behavior of systems with respect to their policies. This work takes the policy as a given. The other side of the coin — *how* to choose a policy, and what a policy means in relation to all other possible policies — has not been addressed from a modelling and analysis perspective. We take the first step in this direction by presenting a model and theory of need-to-know. This addresses the fundamental problem that arises when determining the "who can see what" part of a security policy: how to make the trade-off between functionality and secrecy.

On Task 2, we identified a way of decomposing an adaptive system that enables a systematic way of analyzing its security and ensuring that security is maintained in the presence of adaptations.

On Task 3, we developed a method for performing risk analysis of an adpative system. This method is an extension of the methods used for system profiling by the NSA System Profiling Group. We sketch a way of providing tool support for this risk analysis using RDD-100 as an exemplar.

Chapter 2

Task 1: Changing Access Authorizations

2.1 Introduction

In this chapter we discuss the access control aspect of adaptive security policies. We survey the fundamental issues and present concrete methods for resolving the conflict between security and functionality, and we present methods for establishing suitable security policies in an adaptive environment.

Section 2.2 is an introduction to the basic problems for adaptive security policies, and to mechanisms that address these. Section 2.3 presents a task-based security policy that is well suited to capture the security requirements of adaptive systems. In section 2.4 we present a model and theory of need-to-know that forms the theoretical foundation for the formulation of adaptive security policies.

2.2 Adaptive Security: Resolving Security vs. Functionality Conflicts

To formulate a good security policy requires balancing several competing goals. On the one hand, workers must be provided with enough information to do their jobs, but on the other hand, providing too much information

increases the chance that the information will be misused by malicious or untrustworthy individuals. Tools must provide workers with the functionality they need, yet there must be sufficient barriers to prevent misuse.

We make a note here on terminology. In this document, the word "disclosure" is used to refer to the result of any act or situation whereby information becomes available to a user or process (this includes deliberate release to an authorized person and undesired exposure to an unauthorized person).

There are two broad reasons that sensitive information may be disclosed:

1. Disclosure of sensitive information may be intentional.

2. Disclosure of sensitive information may be a side effect.

Disclosure is intentional when an activity requires that sensitive information be given to users. In the best of circumstances, such disclosure is not a security compromise because of the mandatory access control rules that require that information be disclosed only to a person who has the proper clearance, which indicates that he can be trusted with the information. However, in an emergency, there may be a sudden need to supply critical information to users that goes beyond their normal clearances.

Disclosure is a side effect when an activity is partially visible to people who are not directly involved in the activity. This kind of disclosure is often called a *covert channel*, although in some cases there is nothing "covert" about it. For example, there is no way to build a skyscraper or launch a rocket in complete secrecy, since these activities are visible to anyone. For an example closer to the computer world, it is impossible to commandeer the full resources of a computer system in times of emergency without revealing this fact to all users.

The dividing line between these two kinds of security/functionality conflicts is not always hard and fast. It often depends on a number of factors: whether disclosure can be prevented, the reasons for granting clearances, how fine-grained is the classification of data one uses, and others. For example, consider a computer system that contains both *secret* and *unclassified* data. If the system has "leaks" that cannot be plugged without impeding functionality, then there are two options:

1. If the system is considered to be a multilevel *secret/unclassified*, then *unclassified* users could obtain *secret* information as a side-effect.

2. One can consider all the information on the system to be *secret*. In this case, it will be necessary to give every user who needs to use the system a *secret* clearance, even those who would not normally be cleared to that level. This case would be an example of intentional disclosure.

Task 1 addresses the access control aspect of adaptive security policies, and so in the remainder of this chapter, we consider the issue of intentional disclosure of sensitive information. Side effects (covert channels) are included in the analysis of Task 3 (chapter 4).

2.2.1 Intentional Disclosure

Before discussing possible policies for dealing with intentional disclosure of sensitive information, we need to look more closely at the criteria that must be used for judging whether releasing information is justified. To start off, we need to ask why one would ever intentionally release information to someone who is not cleared for it. The answer, "Because exceptional circumstances require it", while correct, is not the complete story. If there is a possible circumstance under which a user is to be given *top secret* information, why not simply give the user *top secret* clearance to begin with? The biggest argument against this solution comes from the principle of least privilege. The trust that is placed in a person by granting them a clearance is only relative, not absolute. There have been a number of incidents in recent years in which people trusted with *top secret* information have betrayed that trust. To be on the safe side, standard military practice is to allow a user to have a clearance only if there is an imminent need for it. Allowing situation-dependent disclosure of information is a more fine-grained application of the principle of least privilege.

There are two possible approaches to granting a user temporary, situation-dependent access to sensitive information within the context of traditional multilevel security. For simplicity, we will assume that the information is contained in a file.

1. The security clearance of the user can be temporarily raised.

2. The security classification of the file can be temporarily lowered.

Both of these solutions run into problems. If the level of the user is raised so that he can legally see the file, then he will also be able to see other

information that he has no business seeing. On the other hand, if the level of the file is lowered, then it will also become visible to other users who have no business seeing it. Both situations violate the principle of least privilege.

Adding New Levels

A third alternative, which goes beyond traditional multilevel security, and is in keeping with the principle of least privilege, is to change the levels of *both* the user and the file. Let the original level of the user and the file be l_u and l_f, respectively, and let the new levels be l'_u and l'_f, so that $l'_u \geq l'_f$. These new levels will have to be added to the security lattice in such a way that the user is not given access to any unnecessary information, and so that the file is not given to any unnecessary users. This means that for every level l in the original lattice, it must be that $l \leq l'_u$ only if $l \leq l_u$ and $l \geq l'_f$ only if $l \geq l_f$.

One simple choice is to use a new category, `crisis`. (More generally, one might introduce a collection of new categories for different sets of users and files.) The level of the user could be raised by giving him this new category (without changing his hierarchical security level). The level of the information could be changed to *unclassified* with a single category `crisis`. This choice would make it impossible for anyone other than that user to see the file. If it is important that every user who was formerly able to see the file is still able to see it, this can be accomplished in one of two ways:

1. The category `crisis` could be added to every level in the lattice that was originally greater than or equal to l_f.

2. A copy of the file could be made, so that one copy is kept at the original level l_f, and the second is placed at the newly created level l'_f.

The first solution is equivalent to creating a new kind of level, a *disjunction* level. For example, instead of making the level of the file be *unclassified* with category `crisis`, one could instead make the new level be $l_f \cup$ `crisis`, meaning that it could be read by anyone who had level greater than or equal to l_f, or who had category `crisis`. (An object with such a disjunction level can be written only by someone who is able to write *both* objects of level l_f *and* objects with category `crisis`.) Such disjunction levels have been used in the past with ORGCON (ORGanization CONtrol) policies.

Problems with New Levels

Adding new levels to the security lattice on the fly as described above is a workable solution if the file can be made read-only. However, if the file needs to be constantly updated, then there is a real problem. By lowering the level of the file so that it can be read by additional users, one also restricts who can write the file, according to mandatory access control rules. If the file is made *unclassified* with category `crisis`, then it can be written to only by processes with a level less than or equal to this. In general, this restriction would prevent the file from being updated. (Since the original level of the file was l_f, it is likely that it is updated by processes of level l_f.)

The only obvious solutions to this problem are:

1. Continue to allow the original processes to update the file, with no changes to their security levels. This means allowing write-downs.

2. Reclassifying all the processes that write to the file so that their new levels are less than or equal to the new level of the file.

The first approach is equivalent to raising the level of the user to l_f, and then using discretionary access control to prevent him from reading any other files. The second approach is perhaps safer, but is significantly more complicated, since it involves changing the level not only of a single file, but an entire interrelated collection of files and processes.

Dynamic Lattice Approach

In adaptive security, circumstances may force disclosure of information in a way that violates static mandatory access control. In a sense, these violations collapse many of the distinctions between levels. This situation can lead to an effective lattice of security levels that is coarser than the original lattice. For example, consider a multilevel system with three security levels, *unclassified*, *confidential*, and *secret*. If it becomes necessary to make a nonstatic *secret* file readable to *confidential* users, then it will, in general, be impossible to maintain the distinction between *secret* information and *confidential* information. However, as long as both *secret* and *confidential* information are kept away from *unclassified* users and files, it may still be possible to maintain multilevel security with a reduced security lattice consisting of two levels,

unclassified, and *secret/confidential*. After the crisis is over, it may be possible to manually separate *secret* and *confidential* information to reestablish the original security lattice.

Recovery from information flow leaks is more complicated when it comes to users, since there is no way to "erase" information that a user has learned during a crisis. Security for users must involve procedural enforcement; for example, debriefing interviews, and auditing.

The dynamic security lattice approach is developed in section 2.2.5.

A Non-MLS Approach

Another approach is to abandon using MLS security for enforcing the principle of least privilege, and instead use role-based access control with situation-dependent access. According to this strategy, users of a system must be cleared by mandatory security to the highest level on the system. The principle of least privilege is enforced by making each user's access to files dependent on both (1) the role the user is playing and (2) the current situation (whether or not there is some kind of crisis brewing). This kind of non-MLS security policy was advocated by Boebert[BK85] and Clark-Wilson[CW87]. The benefit of such an approach is that access restrictions are designed from the start to be in accordance with the information needs of the user. However, non-mandatory security policies suffer from the fact that there is no good way to check for covert channels. (It is not even clear what the definition of a covert channel would be for such policies.)

An non-MLS approach is developed in section 2.3, where we describe a type of policy, called "task-based", that meets the needs of adaptive security. We also discuss how to handle associated problems.

2.2.2 Intentional Interference

Another issue that is related to intentional disclosure is intentional interference. According to the usual mandatory access control rules, a high-level process is not allowed to interfere with low-level files or processes. (For example, by writing to low-level files or setting low-level variables.) However, there may be some cases where it becomes necessary to allow high-level processes to influence low-level behavior. For example, *top secret* information may reveal that a bomb has been planted in a building. If the building is

then evacuated (by a *top secret* process setting off an alarm), the result will, strictly speaking, be a security violation. Intentional interference is (from the point of view of information flow) equivalent to downgrading, but it differs with respect to who is responsible for actions. In the case of downgrading, information is made available to low-level processes, which then make decisions based on it. In the case of intentional interference, the actions are directly taken by high-level processes.

2.2.3 Mechanisms for Situation-Dependent Disclosure

There are several possible triggering mechanisms that can be used to invoke situation-dependent disclosure:

1. The trigger may be an external signal.
2. The trigger may be the system entering a particular state.
3. Individual users may be authorized to judge whether the situation calls for disclosure.

The last case is the most flexible and also the simplest to implement. However, since the decision to enter the crisis mode is made "off-line", there are no automated checks that the decision was made appropriately. For each of these mechanisms, there is a possible opportunity for manipulation of the system by a malicious user. This threat is alleviated somewhat by careful auditing of all security-relevant events (see section 2.2.4). Preventing the manipulation of the system is usually considered not a security concern but an integrity concern. However, as can be seen in this case, integrity violations can lead to security violations. (Strictly speaking, in such a case, there would be no violation of the automated security policy — the ASP, as discussed by Sterne in [Ste91]). This policy would say that information disclosure can occur in particular circumstances, and those circumstances will actually occur. However, manipulation of the system would be considered a violation of the security policy objective (the SPO of Sterne). It should also be a violation of the operational security policy (the OSP of Sterne), since this policy includes informal rules about how users are supposed to come to their decisions about security-relevant actions.)

Formalizing integrity constraints is notoriously more difficult than formalizing security constraints. This is because the definition of integrity depends on the intended functionality of the system, while security can be formulated in terms of information flow without reference to functionality. (Functionality considerations may go into the choices of level assignments to users, processes, and information, but these choices are all simply parameters to the security theory.) A common way to specify integrity is in terms of "normal forms" for data, which is really not sufficient to capture the distinction between valid and malicious manipulation of the data.

2.2.4 Other Issues for Adaptive Security

Recovery

When information leaks during a crisis, steps must be taken afterwards to recover as well as possible. There are several issues involved in such recovery:

1. Determining what information leaked.
2. Determining the reasons the information leaked.
   (a) Who is responsible.
   (b) What was the situation.
3. Determining how far the information has leaked.
   (a) To which users.
   (b) Into which data containers.
   (c) Out of which output channels.
4. Sanitization/recovery after crisis.
   (a) What to do about people?
5. Containing the leaks.

Auditing

There are two different ways to enforce information flow policies. The first way is to use access control, which prevents a user from performing actions that violate the policy. A second method is to allow the user to perform such actions, but make sure that such actions are audited. The approach using auditing is slightly less secure, since there is no immediate enforcement, but it is more flexible, since it allows the user to use his judgement as to when functionality overrides security concerns. Auditing is thus vital for adaptive security when a user is allowed to trigger disclosure — the third possibility listed in section 2.2.3. To the extent possible, enough auditing information must be collected to ensure that if the user makes irresponsible judgements, he must answer for his actions.

In a general system, there may be information flow polices that are enforced using both of these methods. There must then be an overall policy determining when it is acceptable to use auditing for enforcement.

2.2.5 Dynamic Security Lattices

One way to model the conflicts between security and functionality is using a dynamic security lattice. This approach defines, at any given time, an *effective* security lattice where the partial ordering is determined by the transitive closure of the information flows that are currently enabled. In other words, if it is possible to modify object B based on information in object A, then in the dynamic security lattice, the level of B is greater than or equal to the level of A. The method used here is a simplification of the approach taken by Lee Badger in [Bad90].

The primary advantage of using a dynamic security lattice is that it provides a rough estimate of where sensitive information may have leaked following a change in security constraints. In order that dynamic lattices not violate the constraints of mandatory security, it is imperative that the current security lattice is always compatible with the official mandatory security lattice in the following ways:

1. If a user is forbidden under any circumstances to learn information, then the user's level can never become greater than or equal to the level of the information.

2. When information is exported from the system, it is important to label the information with a level that is greater than or equal to the level of the sources of the information (unless there is an official downgrading policy).

Typically, these constraints imply that all users must be cleared for all information that they could ever, in any circumstance, receive. The dynamic security lattices in this case are a way of enforcing a need-to-know policy.

A Model of Dynamic Lattices

To understand the impact of dynamic lattices, it is helpful to try to formalize the issues. We assume a computer system (operating system plus applications software) can be described by the following parameters:

- A set E of possible information-containing entities. This set will include system variables, and files, as well as users.
- A set A of possible *actions.*
- For each action, there is a set $R(a)$ of entities that are read by the action, and a set $W(a)$ of entities that are written by the action.
- At any time, there is a set P of permitted actions.
- The effective level lattice is defined by the requirement that for any two entities u and v: $l(u) \leq l(v)$ holds if for some sequence of entities e_j and actions a_j in P:
  - $e_0 = u$
  - e_j is in $R(a_j)$
  - e_{j+1} is in $W(a_j)$
  - $e_N = v$

To create a policy for a system, it is necessary to specify the circumstances under which an action will be enabled, and a set of constraints (policy situation-dependent) on the flow relations. In some circumstances, it may be necessary to accompany the granting of new information flow rights

(enabling new actions) with the rescinding of other information flow rights in order to prevent the transitive closure of the flow rights from connecting two entities that should never be connected. For example, it may be that the security policy prevents information from ever flowing from A to C, although in certain circumstances, information can flow from B to C, and in other circumstances, information can flow from A to B. To enforce such a policy, it must be that information flow from B to C is severed whenever information flow from A to B is enabled.

While the analysis described above can be done off-line to determine where information will flow in what circumstances, there is a method developed by Sutherland, Perlo, and Varadarajan in [SPV89] that allows for "run-time" enforcement of changes to data sensitivity. Their approach is to label each data object and message with a specification, not only of its current sensitivity, but also of the circumstances under which that sensitivity may change. As long as the labels are preserved, this approach permits run-time determination of where information of various sensitivity has flowed, and automates the time-dependence of enforcement. However, the implementation requires additional machinery above and beyond what is normally available on secure systems.

2.3 A Task-Based Access Control Policy

One approach to the management of security in adaptive systems is to use a *task-based* access control policy, which makes explicit the relationship between an agent's privileges and the tasks he must perform. The usual term for this kind of policy is "role-based access control", but for adaptive systems the term "task" is more appropriate because it emphasizes what needs to be done rather than what we call the people who will do it.

In role-based systems, a system administrator usually groups related responsibilities together and calls that a role. However, in creating roles in such a way, one is constrained by the implicit assumption that related responsibilities should be concentrated in one individual. This assumption might be wrong or inconvenient for several reasons:

1. As was discussed by Clark and Wilson[CW87], it may be dangerous to give one individual unconstrained privilege in one area. If several people

must work together to play what might be conceptually considered a single role, then they can keep each other honest.

2. In a crisis, it may be important to distribute privileges so that no person becomes a "single point of failure".

3. Because a task is typically more constrained in time and scope than a role is, it may be easier to determine the access rights that are needed to perform a particular task than it would be to determine the access rights to fulfill a particular role.

4. Task-based access control provides more flexibility than role-based access control. One can in principle give an agent the authority to do a task only once, in a particular situation. In contrast, a role implies ongoing privileges.

In creating a task-based access control policy for an adaptive system, one needs to determine the following parameters:

1. A collection of *scenarios* in which the system will have to operate. Each scenario would be a class of situations (we describe such classes in section 2.4).

2. The collection of *tasks* that may need to be performed for each scenario.

3. The collection of *actions* that may need to be taken to accomplish each task.

4. The collection of *objects* that are involved in each action, together with the type of access involved. (Does the action require reading the object or writing it, or both?)

2.3.1 Completeness of the Parameters

These parameters describing the system need to be checked for completeness. In the case of the set of scenarios, completeness involves either showing that the collection of scenarios covers all possible situations in which the system can be used, or else making explicit the circumstances that are covered by the analysis. It is also necessary to show that the set of tasks are complete

for each scenario, the set of actions are complete for each task, and the set of objects are complete for each action.

Of course, it may not be possible to make a formal and rigorous demonstration of completeness because of the inherent complexity and fuzziness of real-world objectives. However, it is possible to improve the chances that nothing was overlooked by using a fault-tree analysis. A fault-tree analysis organizes the set of situations into a tree of possibilities. At each node of the tree is attached a question about the situation, which has a finite number of possible answers. For example: "Is the system in war mode?" has two possible answers, "yes" and "no". Underneath a node is a number of branches corresponding to each possible answer to the question. Each inherits all the answers from higher nodes of the tree. If the questions asked at each node are not formal, then there is no way to prove rigorously that the tree is complete, but it can be tested for completeness by a "devil's advocate". The advocate should be someone other than the people who constructed the tree. The advocate tests the completeness of the tree by constructing a number of hypothetical situations, and then for each situation, answering the questions appropriately until he or she finds a leaf node that corresponds to the situation. The system designers must then give an argument that the scenarios, tasks, actions, and objects are all sufficient to cover this situation.

2.3.2 Information Flow Considerations

After demonstrating as far as possible the completeness of the set of parameters used to describe the system, it is necessary to perform an information flow analysis on the system. Since the set of actions performed is dependent on the scenario, it may be necessary to consider each scenario separately for information flow. The information flow analysis for each scenario can then be performed according to the rules given in section 2.2.1.

2.3.3 Need-to-Know Considerations

It is not enough simply to consider the information flows that are directly involved in each action (this is discussed further in section 2.4). It is also necessary to take into account what kind of information is needed for an agent to be able to determine what action to take. Therefore, for each scenario and each task, at least a rough description must be made of the criteria for taking

actions. For each criterion, it is necessary to determine what information an agent needs to know in order to decide whether that criterion is met. At this point, it is helpful to decide for each scenario a way to partition tasks among agents. Completeness of the set of actions must be rechecked to make sure that each agent receives the information he needs to perform his tasks.

2.3.4 Trustworthiness Considerations

After partitioning tasks among agents, and determining the information flows in the system for each scenario, it is necessary to ask whether any agents will be given a dangerous amount of information (as defined later in section 2.4) in any scenarios. If so, then if possible, the tasks must be repartitioned. If it is not possible to repartition the tasks to eliminate dangerous concentrations of information, then other steps must be taken to decrease the risk of malicious agents. Among these steps are: (1) adding more auditing to increase the chance that malicious agents will be caught, and (2) increasing the qualifications for the involved agents, which means choosing agents that have more extensive background checks and evidence of trustworthiness.

2.3.5 Multilevel Considerations

For multilevel systems, other checks must be made to be sure that the information flows created to satisfy need-to-know constraints — the "internal" notion of security — also respect externally-imposed sensitivity levels — the "external" notion of security. There are two such external constraints that must be respected: (1) an agent must not receive labeled sensitive information unless he is cleared for it, and (2) information may flow from labeled external inputs to labeled external outputs only if the label of the output is greater than or equal to the label of the input. These multilevel considerations further restrict the possible clearances of system agents and the possible information flows that the system can permit. A mismatch between external and internal notions of security can occur in several ways:

1. The necessary clearances of agents may not be the same. This is easily resolved by requiring that agents meet both internal and external clearance requirements. However, this resolution may run into practical

considerations, since there is a limited supply of high-clearance users, and there may be other factors restricting what agents are available.

2. System functionality may inherently involve flows that violate the external security constraints. If this is the case, it is a very serious situation that has no easy remedy. In some circumstances, it may be enough to interpose a human reviewer for data that leaves the system that may be "contaminated" with data of higher or incomparable sensitivity. The reviewer would have to take into account the internal data flows determined for the system in order to estimate the amount of sensitive information that could have leaked into an output. If the use of a human reviewer is impractical, either because timeliness prohibits a lengthy review process, or because it is too difficult for a human to judge the seriousness of the leaks, then it may be necessary to redesign the system or its interface with other systems. In some circumstances, it may help to encrypt the outputs so that they can legitimately be sent out at a lower security level than the sensitivity of the data it contains.

3. External security may be insufficient to enforce internal security requirements. The internal trustworthiness analysis determines that agents should be prevented from learning certain information, depending on the tasks the agents must perform and the clearances of the agents. However, if the external label for this information is *unclassified*, then it is impossible to prevent the agents from learning the information through external means. This is another very serious circumstance. There are several responses to such a situation. First, it may be possible to introduce new categories that are respected by external systems that will make sure that the sensitive information does not make it to agents that might misuse it. Another possibility is that the external information can be encrypted, so that external access control is not needed. A final possibility is just that a higher level of trust must be demanded of the agents, so that agents with higher degree of trustworthiness must be used.

2.3.6 Automated Enforcement Mechanisms

Some systems, such as LOCK [FHOT89], provide support for role-dependent access control, and these can be adapted to enforce task-based access con-

trol. In more standard systems, it is possible to use mandatory access control mechanisms (MAC) to enforce information-flow restrictions and to use discretionary access control to restrict users further so that they can perform only actions that are relevant to their tasks. One problem with using MAC to enforce a nonstandard policy is that it is necessary at all times to keep track of the externally defined sensitivities of data. If the set of external levels is small enough, and the set of possible internal levels is rich enough (for example, if the number of category is large enough), then it is possible to encode both internal and external security labels in the same label field. This approach, of course, assumes the compatibility of the internal and external notions of sensitivity. If the two notions clash, then special provisions must be made ahead of time, as described above.

2.4 A Simple Model for Need to Know

A security policy for sensitive information is an attempt to balance people's needs for information to do their jobs against the risks of misusing that information. The approach taken in the military has been to assign a clearance (or access rights) to each authorized agent that reflects what type of sensitive information that agent can see. An agent's clearance is based on two considerations:

1. The agent's *need to know.*

2. The agent's *trustworthiness.*

Roughly speaking, an agent's trustworthiness has been indicated by the agent's hierarchical level (either *unclassified*, *confidential*, *secret*, or *top-secret*), while his or her need-to-know has been indicated by a combination of his or her assigned security categories, working groups, or file-by-file discretionary access rights.

In the traditional view of security, an agent's trustworthiness is not assumed to change dynamically, so any changes to his or her access rights is due either to changes in need-to-know, or to a change to the balance between need-to-know and security.

2.4.1 What Is Need-To-Know?

In military security, access to information (as well as other kinds of access) is governed by the *principle of least privilege.* Following this principle, an agent must be given the least information necessary to do his or her job. Before information is given out to the agent, it must be established that the agent has the appropriate *need-to-know.*

In most work on theoretical security, the criteria for deciding need-to-know are left outside of the theory, because it is considered too complicated and too subjective to submit to rigorous analysis. But it is not difficult to come up with a simple model of need-to-know that nonetheless illustrates many of the issues involved in situation-dependent security.

Situations and Actions

Our model of need-to-know starts with the case of a single agent with a job to perform. At any moment, the agent is in some *situation* and must make a decision about what *action* to perform. There is one or more actions that the agent could perform that would be considered *acceptable* for the situation, while other actions would not be acceptable for that situation. The agent's responsibility is to choose some acceptable action and perform it. For simplicity, we ignore for now the complications arising from multiple agents, and we also ignore the fuzziness inherent in knowing what is an acceptable action for a given situation.

Knowledge and Sufficient Knowledge

The agent's job is complicated by the fact that he does not know completely what the situation is. All he knows is some collection of facts about the situation: the facts that he is allowed to learn, based on his clearance. There may in general be many possible situations consistent with his facts. In order to do his job, he must (if possible) find an action that would be acceptable for any situation consistent with his knowledge.

In terms of this simple model, we can say that an agent's knowledge at any time is *sufficient* for his task if there is some action that would be acceptable for any situation consistent with his knowledge.

Formalizing Need-to-Know

We formalize the notions that we have introduced so far by using the following primitives:

- $\mathcal{S}$: the set of possible *situations.* In the following, the word "situation" refers to a complete specification of the relevant facts about the domain that the agent must make judgements about. Both the state of the computer system the agent is working with and facts about the outside world may be facets of the total situation.

- $\vdash$: the *satisfaction* relation. If Φ is any statement about a situation, then the expression $s \vdash \Phi$ will be said to hold if Φ is a true statement about situation s.

- $\mathcal{K}$: the set of possible *knowledge states* of the agent. We will leave the precise nature of these knowledge states unspecified for the sake of generality. The simplest model of an agent's knowledge at a given time would be simply the data he has received up to that moment. However, such a model neglects the inferences that the agent might draw from the data, some of which may not be logical deductions but instead are likely interpretations of the data. The use of a more general notion of knowledge opens up the possibility that the agent's "knowledge" can be mistaken.

- B_k : the belief predicate for an agent with knowledge state k. If Φ is any statement about a situation, then we will say that $B_k(\Phi)$ holds if an agent with knowledge state B_k would believe that Φ holds.

- $\mathcal{A}$: the set of possible *actions.*

- $\mathcal{R}$: the set of possible *results* of actions.

- $\rightsquigarrow$: the *consequence relation* connecting situations, actions and results. The expression $s \vdash a \rightsquigarrow r$ will be said to hold if in situation s, action a "leads to" result r.

  For simplicity, we will ignore how results relate to changes in the situation, and instead assume that the notion of a good or bad result takes

into account such changes of situation. Also, we are ignoring nondeterminism in the transition relation, assuming that an action in a specific situation leads to a specific result.

- *acceptable* : a predicate on results indicating that a result is desirable for the current situation. We will use the notation $s \vdash \mathit{acceptable}(r)$ to indicate that result r is acceptable for situation s.

Some Definitions In terms of the primitive notions introduced above, we can define some of the terms needed to discuss need-to-know and situation-dependent security.

Let us first extend the notion of acceptability to actions as well as results as follows:

Definition: $\mathit{acceptable}(a) \equiv \exists r \in \mathcal{R} \ : \ a \rightsquigarrow r \& \mathit{acceptable}(r)$

In other words, an action is acceptable if it leads to an acceptable result. As with acceptable results, an action is only acceptable with respect to a situation. We will write $s \vdash \mathit{acceptable}(a)$ to indicate that action a is acceptable in situation s.

In the next two definitions, we can classify knowledge states according to how accurately and how completely they reflect the situation.

Definition: $k \in \mathcal{K}$ is *accurate* for situation s if for all situational statements Φ, $s \vdash B_k(\Phi) \rightarrow \Phi$.

Knowledge state k is accurate for a situation if everything believed by an agent with that knowledge is true of the situation. If a knowledge state is inaccurate, that means that it contains partially false knowledge.

Definition: $k \in \mathcal{K}$ is *conservative* in situation s if for all actions a, $s \vdash B_k(\mathit{acceptable}(a)) \rightarrow \mathit{acceptable}(a)$.

Introducing the idea of conservative knowledge takes into account that it is not actually necessary for an agent's beliefs to be completely accurate, as long as they always err on the side of safety — that is, as long as the inaccuracy does not lead him to take an unacceptable action. For example, if a person trying to repair a lamp believes that touching any bare wire

will electrocute him, then such a belief may be inaccurate—it may be the case that some bare wires, such as the ground wire, are perfectly harmless. However, the inaccurate belief errs on the side of safety, and therefore is a conservative belief.

Another important property of knowledge states is *decisiveness.* Obviously, if the agent does not know enough to decide what action to take, then he is unable to do his job, even if his knowledge is perfectly accurate (that is, he does not have any false knowledge). An agent's knowledge is said to be *decisive* if it allows him to figure out some action to take that is acceptable. Decisiveness is formalized as follows:

> Definition: $k \in \mathcal{K}$ is *decisive* if for some action a, $B_k(\mathit{acceptable}(a))$ holds.

Finally, we can define sufficiency of a knowledge state for a given situation:

> Definition: $k \in \mathcal{K}$ is *sufficient* for situation s if k is decisive and k is conservative for situation s.

By this definition, knowledge is sufficient for a situation if (1) it allows the agent to decide on at least one action, and (2) whatever action the agent decides to take is actually acceptable for the current situation.

Necessary Knowledge and Need-To-Know We have said when an agent's knowledge is sufficient for him to perform his job, but we have not said when it is necessary, which is the basis for need-to-know restrictions. We will say that a particular fact (situational statement) is *necessary* for the agent to know if, whenever he has sufficient knowledge to perform his job, he will be able to deduce that fact. In other words:

> Definition: Φ is *necessary* for situation s if for all $k \in \mathcal{K}$, $\mathit{sufficient}(k) \rightarrow B_k(\Phi)$.

Naively, one might think that applying the principle of least privilege, that an agent should only be given information if it is necessary for his or her job. We can formalize this as follows:

Definition: $k \in \mathcal{K}$ conforms to the principle of least privilege for situation s, or is *necessary*, if for all situational statements Φ, $s \vdash B_k(\Phi) \rightarrow necessary(\Phi)$.

Unfortunately, this interpretation of the principle of least privilege often cannot be used in practice, for the simple reason that *necessary* knowledge may not be *sufficient.* In other words, it could be that for some situation s:

The principle of least information fails for s if $\forall k \in \mathcal{K} : s \vdash necessary(k) \rightarrow \neg sufficient(k)$

In other words, in some situations *you need to know something that you do not need to know.* This could be illustrated with a simple example. Consider a scenario in which an agent must choose between two possible buttons, a red button and a green button. There are four possible situations: A, B, C, or D. In situation A, only the red button is acceptable, in situation B, only the green button is acceptable, and in situation C, either button is acceptable. In situation D, the agent should not push either button.

Now consider the agent's knowledge in situation C. The agent could be told any one of the following four statements:

1. The red button is acceptable.
2. The green button is acceptable.
3. Both buttons are acceptable.
4. At least one of the two buttons is acceptable.

The first three statements fail to conform to the principle of least privilege. It is not *necessary* for the agent to know that the red button is acceptable, since he could take an acceptable action (namely, push the green button) without knowing that the red button is acceptable, as well. Similarly, it is not necessary for him to know that the green button is acceptable, and it is certainly not necessary for him to know that both buttons are acceptable.

The last statement in the list above contains only information that the agent needs to know. But it is not sufficient, since it does not allow him to make a decision: he doesn't know whether to push the green button or the

red button. If the agent is to be given sufficient information to do his job, he must be given some information that is not necessary.

We can amend the naive formulation of the principle of least privilege by interpreting it as the principle of *minimal* privilege, with the following definitions:

> Definition: For $k_1, k_2 \in \mathcal{K}$ we say that $k_1 \leq k_2$ if for all situational statements Φ, $B_{k_1}(\Phi) \rightarrow B_{k_2}(\Phi)$

This definition says that knowledge state k_1 is less than or equal to knowledge state k_2 if every statement known by an agent with knowledge k_1 would also be known by an agent with knowledge state k_2. In terms of this partial ordering on knowledge states, we can define what is minimally sufficient knowledge for a given situation.

> Definition: $k \in \mathcal{K}$ conforms to the principle of minimal privilege for situation s, or is *minimally sufficient*, if $s \vdash \mathit{sufficient}(k)$ and for all $k' \in \mathcal{K}$, if $s \vdash \mathit{sufficient}(k')$, then $k \leq k'$.

The principle that an agent should be given a minimally sufficient amount of information, has one problem: there may be several different knowledge states that are both minimally sufficient. Therefore, the principle does not uniquely specify what information should be given to the agent. In order to address this issue, it is necessary to re-examine the reasons why agents should be given a minimal amount of information, in the first place.

2.4.2 Temptation and Trust

The reason that one tries to minimize the amount of information given to an agent is the risk that the agent might misuse the information (or else, inadvertently pass the information along to someone else who might misuse it). But what, exactly, does it mean to *misuse* information?

This is a complicated question, but once again we will try to illustrate some of the major issues by constructing a simple model. The issue of misuse of information arises when one considers the possibility that an agent may be motivated by goals other than doing his job well. These other motivations need not be as nefarious as treason or greed. An agent could also be

motivated by simple curiosity, or the thrill of getting away with something (as hackers often are).

It is obviously impossible to construct a rigorous model of human motivations, so instead we will just postulate their existence. We formalize it in terms of the temptations, risk and plausibility.

> For each $k \in \mathcal{K}$ let $tempting_k$ is a predicate on the set of results $\mathcal{R}$. We say that $tempting_k(r)$ holds if it the agent with knowledge k might be tempted to try to achieve result r.

> Let *undesirable* be another predicate on the set of results $\mathcal{R}$. We say that $undesirable(r)$ holds if the agent will try to avoid result r. For example, punishment is an undesirable result.

> For each $k \in \mathcal{K}$, let P_k be the *plausibility* predicate for an agent with knowledge k. We say that $P_k(\Phi)$ holds if an agent with knowledge k would believe that statement Φ is plausible. This is much weaker than believing it is true.

We can define risky actions in terms of plausibility and undesirable results:

$$\text{Definition: } risky_k(a) \equiv P_k(\exists r \in \mathcal{R} \ : \ a \rightsquigarrow r \& undesirable_k(r))$$

This definition says that an action is *risky* for an agent if it is plausible (not necessarily certain) that it will lead to an undesirable result. In a more sophisticated model, whether an action is considered risky or not would be relative to the temptation: if an action is very tempting, then it would take a large risk to deter an agent from taking the action, while a lesser temptation could be deterred by a smaller risk. However, in our simplified model, we will only consider the extreme case of risks that the agent will avoid in all circumstances.

Now, we can extend the notion of a temptation to an action as follows:

$$\text{Definition: } tempting_k(a) \equiv P_k(\exists r \in \mathcal{R} \ : \ a \rightsquigarrow r \& tempting_k(r))$$

This definition says that an action is tempting if it is plausible that it might lead to a tempting result. Why, in the definition of temptations, is it important that plausibility, rather than belief, be used? The reason is that

people do not have to be *assured* of a reward in order to take an action; it is enough that that there be a plausible chance for a reward. For example, if a thief is walking through a building, looking for valuables, he will try a door on the plausible assumption that it might be unlocked, and that there might be something valuable behind it. He might have no reason to *believe* that the door is unlocked, but he could still decide that it is worth a shot.

In these terms we can say that a knowledge state is *dangerous* if there is some action that is tempting, but not risky.

$$\text{Definition: } dangerous(k) \equiv \exists a \in \mathcal{A} : tempting_k(a) \& \neg risky_k(a)$$

This definition calls a knowledge state dangerous if the agent with that knowledge state would be tempted to do some action motivated by something other than the job he has to perform. This definition doesn't consider the harm that an agent might do. In some cases, one might allow an agent to be tempted, in the hopes that the agent could be caught. For such a trap to work, it would have to be the case that the agent's beliefs about the plausibility of his getting caught would have to be inaccurate. One way to accomplish that is to have alarms that are set off by suspicious behavior, and keep the existence of these alarms secret.

If there is no reason to suspect that there is a malicious agent involved, then the goal of system security designers should be to try to avoid putting agents into temptation. In other words, try to keep the agents from having a dangerous amount of knowledge. We are now in a position to give a definition of the trustworthiness of an agent:

> Definition: An agent is *untrustworthy* in situation s if for all knowledge states $k \in \mathcal{K}$, $s \vdash sufficient(k) \rightarrow dangerous(k)$.

In other words, an agent is untrustworthy in a given situation if it is impossible to give the agent sufficient information for him to do his job without also making some actions tempting.

2.4.3 Summary and Conclusions

Although we are using a very simplified model of an agent's behavior, there are a few important conclusions that can be drawn from it. Below is a summary of the model and its conclusions.

1. An agent has sufficient knowledge to do his job if he is able to determine an acceptable action to take.

2. It is not necessary for an agent's knowledge to be accurate, as long as it is conservative—that is, as long as it errs on the side of safety.

3. An agent needs to know a fact in a given situation if it is impossible to give the agent sufficient knowledge without letting him know the fact.

4. In general, there is no unique smallest amount of information that can be given to an agent that is sufficient for him to do his job.

5. Trustworthiness is situation-dependent. It is not an all-or-nothing thing. An agent may be completely trustworthy in normal circumstances, but will be tempted to misbehave in some situations. For example, if suddenly there would be a large financial reward for taking a certain action.

6. Certainty of the result is not necessary for an action to be tempting for an agent. It is not enough to prevent an agent from *knowing* sensitive information for certain. A plausible guess that an agent can make will sometimes be as damaging as a real leak of information. The most extreme case is of course passwords: if an agent believes some passwords to be plausible, he might try them out. To avoid the temptation of trying out passwords, password guidelines should be announced so that everyone knows that easily guessable passwords are not being used.

7. Certainty of the result is not necessary for an action to be considered risky for an agent.

8. Increasing the risk associated with misbehavior (for example, by auditing) can decrease the temptation and increase the trustworthiness of the agent.

Future Work on Need-To-Know

There is much work that needs to be done in order for a theory of need-to-know to have practical benefits in the security of adaptive systems. The biggest problem is that so many of the basic elements of the theory are

unknowable in general, such as what results an agent would find tempting and how much of a risk an agent would be willing to take. For the work to be applied to actual systems, guidelines must be developed for making informed estimates for these unknown factors. The following list describes some of the work that must be made to make the theory more realistic and more usable.

- Classification of types of risks, temptations, and uses of knowledge.

  There are different uses that a malicious agent can find for knowledge. Some knowledge, such as passwords, are keys that give an agent more power. Other knowledge is a commodity that can be sold (for example, to a foreign government), and is not used directly by the agent. Finally, knowledge can increase the certainty about the results of actions, thus making some actions more tempting and others less tempting. In order to have a more complete understanding of security, it would be helpful to have a classification of the different uses of knowledge and the most effective measures (auditing, or access control) that can be used to prevent its misuse.

- Extending the model to allow for risks to be relative to the temptation.

- Develop the connections with

  - Lattice-based security.
  - Information flow.
  - The aggregation problem.

- Develop methodology/tool for

  - Analyzing need-to-know, risks, and temptations.
  - Partitioning information to minimize temptations.
  - Creating roles that require sufficient knowledge that is not dangerous.
  - Identifying situations where trust breaks down. There are several aspects of this that might be considered:
    * Analysis before fielding to identify situations that might lead to a breakdown of trust. The security policy and enforcement mechanisms should take such an analysis into account.

* Automated detection of dangerous situations, so that an alarm can be given.
* The use of traps for malicious agents. To the extent that it can be determined ahead of time which situations are most likely to tempt an agent to betray his trust, it may be possible to set up a trap to catch such agents.

Chapter 3

Task 2: Maintaining Security in Adaptive Systems

3.1 Introduction

When switching between configuration modes that involve some change in security enforcement, there is the problem of ensuring that all of the mechanisms to support security are in place. The aim of this task is to outline a procedure that helps designers or developers, or evaluators, to check, systematically, that everything is indeed in place.

We first remark that we are talking about functional adaptation (configuration) that has major impact on security enforcement. This need not necessarily involve a change to the security policy in the sense of who is cleared to see what, though these adaptations are included here. Adaptations involving clearances were the subject of Task 1, where their special problems were analyzed in detail.

We will begin by identifying more precisely the type of adaptation that is under study. After that, we will explain why a problem arises, and what makes this situation different from the usual problems with dynamically reconfigurable systems that do not involve adaptive security. Finally, we will elucidate our approach to handling this problem.

3.2 The Kind of Adaptation Involved

We will illustrate the kind of adaptation at issue here with examples. Suppose that we have a system with four modes: `normal`, `extraordinary`, `training`, and `maintenance`. We can think of this as a system involving ground bases, aircraft, and C^3I.

Consider a processor that handles both high and low sensitivity requests. Suppose that in `normal` mode of operation, a scheduler uses a fixed time-slicing algorithm and thereby avoids any covert timing channel. In a crisis, we are prepared to tolerate a greater rate of possible information leakage (it may be also that certain high data in a crisis does not have much value for long). So suppose further that when the system makes the transition from `normal` to `extraordinary`, the scheduler algorithm changes to a more efficient algorithm driven by criticality, where the high requests are the most critical. Thus in this new mode, we have a potential covert timing channel.

Consider next an encryption component that uses long encryption keys when the system is in `normal` mode of operation. When switched to `extraordinary` mode, the encryption is performed with shorter keys, providing less security.

Suppose again that highly sensitive material is stored on disk and available in `normal` or `extraordinary` mode. When changing to `training` (or `maintenance`) mode this sensitive data must be removed and less sensitive data must be substituted for the original data. An example of this would be the case where locations of friendly and enemy positions are in a database on the disk, and we wish this knowledge to be secret, but we wish to train people on functionally similar but fictitious data.

A final example of security-relevant reconfiguration that occurs as a result of a mode change, concerns the re-routing of messages. Sensitive messages that would be sent over secure channels in `normal` mode may be sent encrypted, or even unencrypted, over insecure channels in `extraordinary` mode. Circumstances that could warrant this reconfiguration be generated from among the following: high time criticality, overloading of secure facilities, unavailability of secure facilities, etc.

We have just described four cases of security adaptations that occur along with mode changes. We will explain what sets this sort of adaptation apart from the more standard sort of dynamic reconfiguration. After that, we address the main concern of this chapter – what to do about the security

problems.

Systems without adaptive security are currently built that have various dynamic reconfigurations incorporated, like re-assignment of processing in the case of processor failure (fault tolerance), or message re-routing in response to load, to achieve maximum throughput. But in the usual case, the security enforcement on information, users, and processes does not change. The designers must ensure that the proper, constant, security is enforced across all configurations. Examples of what must be ensured are: that encryption is performed for designated message and channel types; that MAC and DAC are checked; that the security mechanisms like the form of MAC and DAC are consistently interpreted and enforced in disparate systems.

It is the responsibility of the designers of these systems to impose system-wide methods of doing things so that when the system is reconfigured, security needs are met. The uniformity of the requirements (a message of such-and-such a type must always be sent over a channel of such-and-such a type) is of some assistance in designing these systems, and motivates the solutions. Providing these services is certainly an important and difficult issue, but it is an issue that is not specific to adaptive security.

What is our responsibility, is to present a procedure that will meet new problems that are consequences of the adaptive nature of the security — the sorts of problems that arose in the examples above. We will deal with this in the next section.

We note however that in some circumstances there is some overlap between the adaptive security problems and the conventional ones. For example, if fault-tolerance causes a disk used for secret data to be re-assigned to unclassified data then while no inherently security-related mode change has occurred, there are adaptive security consequences. We will discuss how to treat these cases in section 3.3.2.

3.3 Maintaining Security in Adaptive Systems

Our aim here is to describe a systematic procedure that will help designers or evaluators to ensure that all the security measures have been taken for an adaptive system. We identify two key goals in handling the adaptive security

problem:

1. Ensure that after a transition, the system is in a consistent, legal state (from the security point of view) for the new configuration mode. This amounts to meeting an *initial condition* for a mode.

2. Ensure that the transition itself does not create any security violations.

Consider the first of these in the context of our some of our examples above.

When changing back from `extraordinary` mode, in the case where short encryption keys have been used, to `normal` mode, care must be taken to reset all the mechanisms – e.g., if there is a table of pre-selected short keys, it must be emptied and refilled with longer keys.

If a disk used for low-sensitivity data is to be re-assigned to high-sensitivity data, all rights to interact with the disk must be re-set correctly.

When making the change that allows sensitive messages to be re-routed across insecure channels, various tables have to be re-written to perform this. This must be done at all appropriate points in the network, and done carefully so that just the designated insecure channels are used.

Finally, suppose in a crisis (not one of our four examples here, though if added it would presumably be reachable only from `extraordinary` mode) we reset the access control mechanisms to allow a user access to information he is not normally allowed to see. Whatever solution (see chapter 2) is used, there are potentially many entries in the authentication databases and/or files that must be set (including audit logging).

Now let us look at item 2 above. The transition action between modes may result in a legal state, in the above sense, yet in itself cause a security breach.

The archetypal example of this is failure to sanitize. We gave an example above where the system changes from `normal` mode to `training` mode and a disk has to be sanitized. Suppose that the disk was originally labeled SECRET, and is re-labeled UNCLASSIFIED. Suppose now that the disk was not sanitized, so that UNCLASSIFIED users can see SECRET information left on the disk. The system is in a properly configured state, and there is nothing in the state of the system that can indicate that the data on the disk is SECRET — put loosely, the system cannot "know" that the data bits on

the disk constitute SECRET data. The breach of security occurred during the transition, in placing the label UNCLASSIFIED on an object containing SECRET data.

We see that both items together cover what is needed to ensure secure functioning in the face of adaptivity. In the following subsection we will outline how to apply this method.

3.3.1 Systematic Analysis of the Security of Adaptive Systems

The first step in the analysis is to identify the various configuration modes. For example, we identified four in our example above. The modes need not necessarily have names that are recognized on, or incorporated into, the system, though it would be useful if that were the case. We also identify the possible transitions between modes. There will not necessarily be a transition between any two modes.

We pause to remark on the number of modes involved. As discussed above, these are not all possible functional configurations that a system could be in. The number of modes that involve security adaptations of the kind we are concerned with, will be small enough that examination of all of them is feasible.

Much of the system will remain unchanged when in the various modes, and this common part will be analyzed once. Then, for each mode, we address the two items listed above.

The first item — the list of things that have to be achieved after a transition to a mode — is associated with the mode in question. Note that this list will be associated with the mode, rather than the particular transition into the mode (there may be more than one). This is because the list depends on the mode (what its proper states are), and not how the mode was entered. It is true that achieving those goals will be more or less work depending on which mode the system came from (in some cases nothing may need to be done), but breaking the analysis down further will complicate the picture, while not usually providing much benefit.

The second item — the list of things that have to be done to ensure that the transition itself does not generate any security violations — is associated with particular transitions. This list will include things like the requirement

that a disk be sanitized. We remark that not all of these procedures can necessarily be executed automatically on a computerized system. It may be, for example, that on transition from `normal` to `maintenance` mode, the disk in our example must be physically removed and subjected to sanitization outside the computer system. The full instructions for this procedure will be detailed in the analysis component associated with the transition.

The analyst uses this framework to cover systematically the special security needs of an adaptive system. In chapter 4, we will describe how a system is modeled according to this method of decomposition. We will use as an example the design tool RDD-100 and show how to attach the information described here to a model of the system being studied.

3.3.2 Other Adaptations That Have Security Consequences

At the end of section 3.2 we mentioned that certain other functional adaptations that do not inherently have anything to do with security may have adaptive security consequences. An example of this would be unpredictable fault-tolerance reconfiguration that requires a disk previously dedicated to SECRET material to be re-assigned to UNCLASSIFIED material. The issues discussed above arise also in this case. To handle this, we do not introduce any new modes into the above model, but we add a further transition from a mode to itself (for any mode in which this can occur). We attach the same sort of information to this transition, that we would to any other. To uncover any of the exceptional adaptations of this kind, the analyst must check all of the ordinary adaptations to ensure that all potential security problems are handled by pre-defined mechanisms. Some kind of broad classification scheme will be needed to avoid an excessive number of cases. Any problem found will be handled by adding a further transition as described.

3.3.3 Large Scale Systems

We briefly mention another setting that poses adaptive security problems. This setting involves systems that are outside the range of systems that this project is concerned with. When systems (e.g. a WAN) become so large that no centralized component has a full description of the system or

even knows the extent of the system, special techniques have to be used to keep the system in some sort of acceptable configuration. This applies to security also – heterogeneous components with different security polcies may be connected, and moreover, security policies will evolve with time. This problem will become more important in the future, and is an important research topic.

Chapter 4

Task 3: Risk Assessment

4.1 Introduction

In this chapter we describe how to assess risk for adaptive systems. There are currently several methodologies for risk analysis of (not specifically adaptive) systems. Our approach is to take a recognized methodology and enhance it to encompass adaptive systems. We will use the decomposition described in chapter 3 to handle the complexity of the multiple operational (security) modes of an adaptive system. Moreover, this decomposition provides a structure that surfaces the new risks to which adaptive systems are vulnerable, as described in Chapter 3.

We will begin by briefly describing two major risk analysis methodologies, one quantitative and the other not. The non-quantitative methodology will be used in our work here. We later suggest linking this analysis with the quantitative approach as a sensible future goal.

4.2 Background to Risk Assessment

Current results on risk assessment primarily consist of management guidelines for controlling risk in the absence of quantitative means for measuring it. There are software tools for enforcing managerial policy, though, and for making quantitative estimates of risk. There are two different approaches to risk management that we will describe here, one derived from CSC-STD-003-85 [Nat85] (the "Yellow Book"), and the other incorporated into the System

Security Profiles [Fin94] of the NSA System Profile Group.

4.2.1 Yellow-Book Style Risk Analysis

The Yellow Book requires computing a *risk index* for an intended system, assigning a *security mode* to the intended system operation, and then using a trusted computing base for the intended system whose Trusted Computer System Evaluation Criteria ("Orange Book") [Dep85] division and class ratings meet minimum standards that the Yellow Book sets as functions of the risk index and security mode. The process for doing this has been changed and simplified slightly by later publications such as [Dep88]. The risk index is generally the intended system's *data sensitivity rating* minus its *user clearance rating*. See [Nat85] for more details.

Landwehr and Lubbes [LL85] of the Naval Research Laboratory developed extensions to the Yellow Book risk assessment process, creating an analysis method that incorporates further relevant information — "processing capability" factors (see [LL85] for details).

ANSSR

The Analysis of Networked Systems Security Risks (ANSSR) tool [BCK90, Bod92, BC93] incorporates and automates basic Yellow Book, processing capability, and networking considerations analyses. It also incorporates consideration of particular attack scenarios, with considerations such as a potential penetrator's potential gain and risk of apprehension.

ANSSR offers three types of analyses, requiring progressively more detail about the system being analyzed: Yellow Book; Extended Yellow Book; and Scenario Analysis. This last type of analysis requires the analyst to enter event-sequence scenarios (e.g., user goes bad, user logs in, etc.). The analyst makes numerical estimates such as the sensitivity of data exposed by an event, or the effort required by an attacker of a given capability, etc. ANSSR calculates likelihoods of events in relation to system states. ANSSR also makes estimates about the expected losses in a time period (normally a year), based on its model.

4.2.2 System Profile Style Risk Analysis

The risk analysis in System Security Profiles makes no attempt to compute risk indexes, and does not involve quantitative estimates of risk. The analysis comprises the compilation of all known system weaknesses together with related security concerns, and this is presented in a highly structured way. The System Profile approach also covers risks other than those of disclosure from deliberate attack (the only risks considered in ANSSR) — e.g., the risk of loss of service from physical damage.

In more detail, the System Profile approach calls for listing all known system weaknesses, and for each weakness listing the following:

- which engineering tests establish the existence and properties of the weakness;
- all known vulnerabilities resulting from that weakness;
- all known threats exploiting that weakness;
- an assessment of whether the weakness is tolerable, with the reasons for this assessment; and
- recommendations on system operation (e.g., special restrictions) for dealing with the weakness.

The System Profile approach treats a vulnerability as arising from the combination of a weakness and a threat capable of exploiting that weakness. For each vulnerability, it calls for describing the vulnerability, assessing the risk posed by that vulnerability, and making recommendations on what, if anything, could be done to minimize the risk posed by that vulnerability.

The NSA System Profile Group has built RDD-100 (see [Asc93]) schema to provide tool support for this analysis. The parts of a System Profile description of risk are pieces of text that are implemented as attributes of elements in an RDD-100 database. RDD-100 creates element-relationship-attribute models, so the text fragments describing risk properties are attributes of elements linked to each other by various relationships. In the RDD-100 System Profile implementation, for example, an element of type **Weakness** is linked to an element of type **Vulnerability** by a *results in* relationship. See Fink [Fin94] for details.

The System Profile approach is simpler and more straightforward than the quantitative approach, and so is the appropriate starting point for the first development of our methods for handling adaptive systems. The description here was intended to give a general view of the System Profile approach. Further description of relevant details will be given in the next section, where we concentrate on extending the approach for our needs.

4.3 Security Profiles of Adaptive Systems

This section describes how to create a security profile of an adaptive system. This method is based on the system security profiles proposed by the System Profile group at NSA. Section 4.3.1 provides necessary background on system profiles as proposed by NSA. Section 4.3.2 describes how to extend this approach to describe an adaptive system in a way that focuses on and highlights the system adaptations. This section also describes how the security analysis is added to the profile. Finally, section 4.3.3 describes how RDD-100 can be used to support this process.

4.3.1 System Security Profiles

System security profiles highlight and document the essential security features of a system. Ideally, a profile of a system is produced in concert with the development of the system so that the profile is available for use by system integrators in making informed decisions about the suitability of or proper use of the system. The effort to produce a profile may be limited in comparison to a formal Orange Book evaluation, but a profile is expected to be available in a more timely manner.

The system profile group of the NSA has proposed a standard form for and a methodology for producing system profiles. Key elements of this proposal are described here; for more details see the draft report [Fin94].

A profile contains descriptions of the security policy for the system, a description of the system architecture, a description of the systems security services, assessments of these services, descriptions of the assurances that the system offers, descriptions of the support services the system requires, a system vulnerability assessment, and guidance as to how to operate the system securely. Of these, the system vulnerability assessment is considered

to be the most important.

A system profile is described in terms of elements and relationships between these elements. For example, the system architecture is described in terms of the system components; components are decomposed into subcomponents. The system vulnerability assessment is organized in terms of system weaknesses, threats to the system, and system vulnerabilities. A system weakness that is exploited by a system threat results in a system vulnerability. The system vulnerabilities are connected to the system description by attaching weaknesses to components. In such a case, a component is said to exhibit the weakness.

Each element of a system profile also has attributes associated with it. For example, the description attribute of a system weakness describes a problem with or a concern about the system. The description attribute of a system threat describes the capabilities, intentions, and attack methods of an adversary. A system vulnerability has three attributes of interest: a description, a recommendation, and a risk. The description attribute tells how to perform a particular attack and what the results will be. The recommendation attribute provides recommendations for fixes. The risk attribute describes the ease of exploitation of the vulnerability, opportunities that may exist to exploit the vulnerability, and the risk that an exploitation of the vulnerability will be detected.

4.3.2 Profiles of Adaptive Systems

We are concerned with adaptive systems in the sense of chapter 3. An adaptive system will be described in terms of its *modes* of operation and the *transitions* between these different modes.

An adaptive profile organizes information in three different ways. First, to understand the security of a mode, it is necessary to understand which parts of the system have been adapted for that mode and for each adapted piece how the piece operates in this mode. The first goal of our security analysis is to focus attention on the security of the system when it is operating in each mode. Second, to understand the security of transitions between modes, it is necessary to look at the many different changes in many different parts of the system that may occur. Another goal of our security analysis is to take a look at the effects of these transitions and the security issues that arise during a transition. Third, there may also be many parts of a system that

do not adapt. These parts of the system will be common to all modes and the security concerns for these parts are less likely to be effected by mode changes. The third goal of our security analysis is to organize the security analysis of common parts of the system centrally so that it does not need to be repeated for each mode.

The Generic Model

An adaptive profile starts with a generic model of the system, as illustrated in Figure 4.1. The purpose of the generic model is to provide a baseline description of the entire system. Parts of the system common to all modes can be described in full detail in the generic model. If the security analysis of the system identifies a weakness of one of the common components then this weakness can be attached to that component in the generic model; if this weakness can be exploited by a threat then the resulting system vulnerability must be reported. This part of the profile is the same as for system profiles. Components that are subject to adaptations can only be described in general terms in the generic model. General functionality, and possibly descriptions of the alternative ways of functioning, can be provided here, but details should be left to the descriptions of individual modes.

Modes of the System

In addition to the generic description of the system there is a separate description of each mode. Each description of a mode concentrates attention on those components that are adapted for that mode. More detailed descriptions and additional documentation are provided in the mode description for the adaptive components. This additional information will detail how a component works in this particular mode. This may require a more detailed description of the component than was provided in the generic model of the system — e.g., the component may be further decomposed into subcomponents. Components that are not adapted for the mode are described using the same descriptions as are used to describe the component in the generic model.

For our example system, three modes are illustrated in Figures 4.2, 4.3, and 4.4. In these figures, the components that are shaded dark gray are the components that are adapted for the mode. Components whose functional-

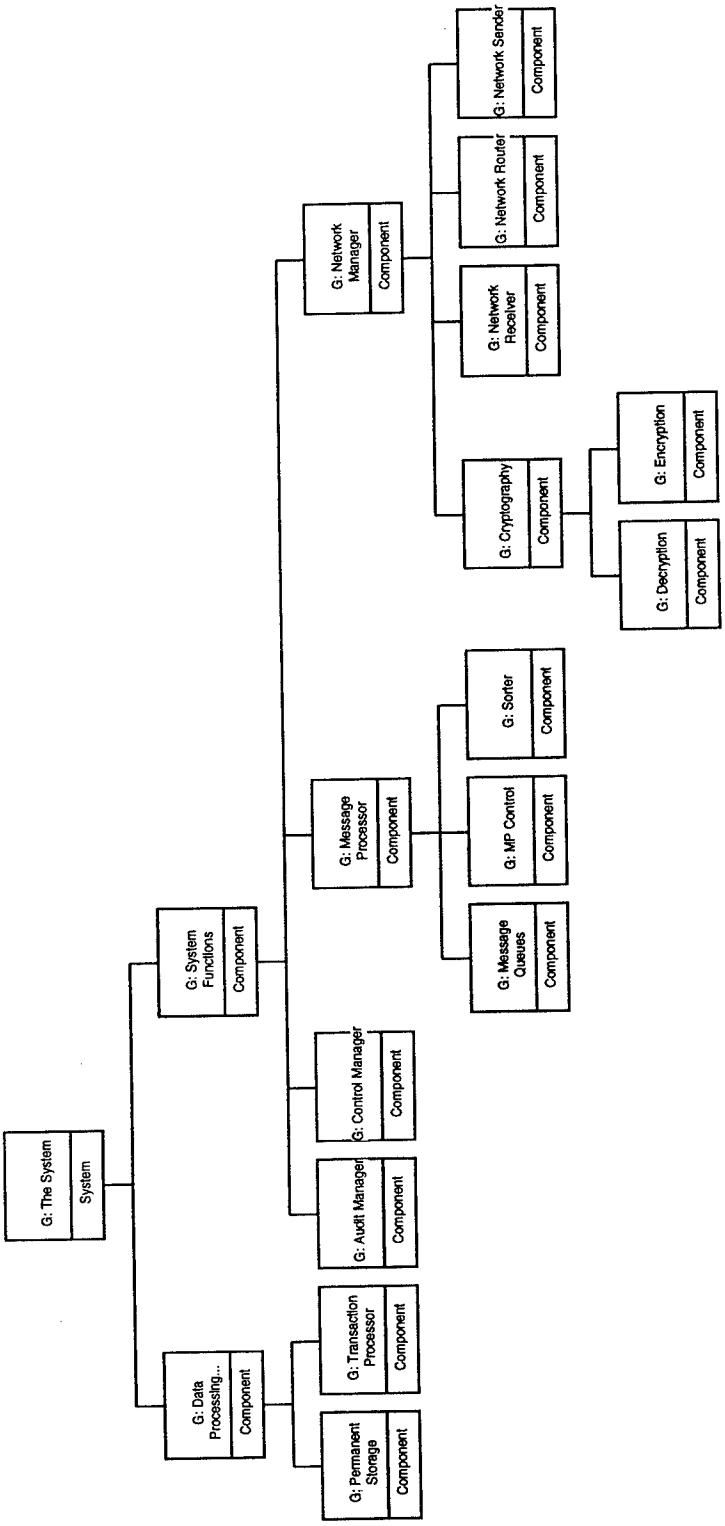


Figure 4.1: The Generic System

ity does not change in the mode are unshaded or shaded light gray; these components are the same as for the generic model. Different kinds of adaptations are possible for different components; these different adaptations are highlighted in the figures by outlining the different adaptations using different dashed lines. For example, first mode is called Normal Mode and is illustrated in Figure 4.2. In this mode there are four different adaptations, effecting six different components. In Extraordinary Mode there are four adaptations, effecting five different components and in Training Mode there is one adaptation, effecting only one component.

Once one has a description of a mode of a system, the security of the system operating in that mode can be analyzed. As for system profiles, this may result in the discovery of system weaknesses. These weaknesses may be exploited by system threats resulting in system vulnerabilities. The weaknesses, threats, and vulnerabilities that are specific to a mode are documented as part of the mode description. A weakness that is exhibited by an adaptive component is associated directly with that component in the description of the mode. This also associates any threat that can exploit this weakness and any resulting vulnerability indirectly with the adaptive component.

In reviewing an adaptive profile, it is highly desirable to be able to quickly identify which components may have weaknesses that are specific to a mode. In Figures 4.2, 4.3, and 4.4 this is easy to do for adaptive components since these components are shaded dark gray. However, it is possible that a weakness in an adaptive component will result in a weakness in a component at a higher level in the component tree, that is, weaknesses may propagate up the component tree. Thus it is possible that a non-adaptive component, whose descriptions are taken from the generic model, will have weaknesses that are specific to a mode. This kind of weakness is called a *derived weakness.* In Figures 4.2, 4.3, and 4.4, components with derived weaknesses are are highlighted by shading them light gray. This shading allows an analyst to quickly locate all components that have weaknesses that are specific to a mode.

A mode also has associated with it a set of initial conditions that describe conditions that must be met in order to enter the mode. Initial conditions are described more fully in chapter 3.

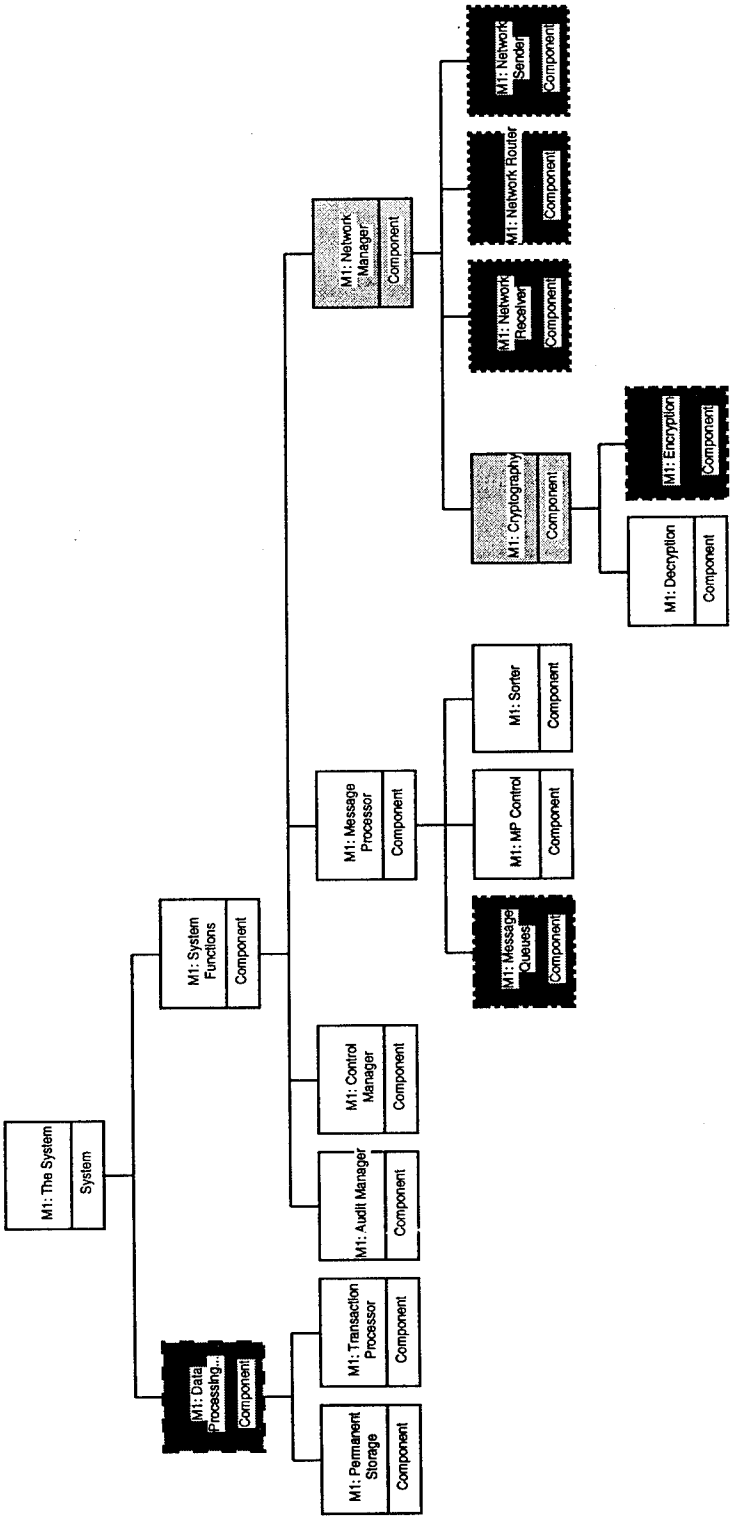


Figure 4.2: Normal Mode

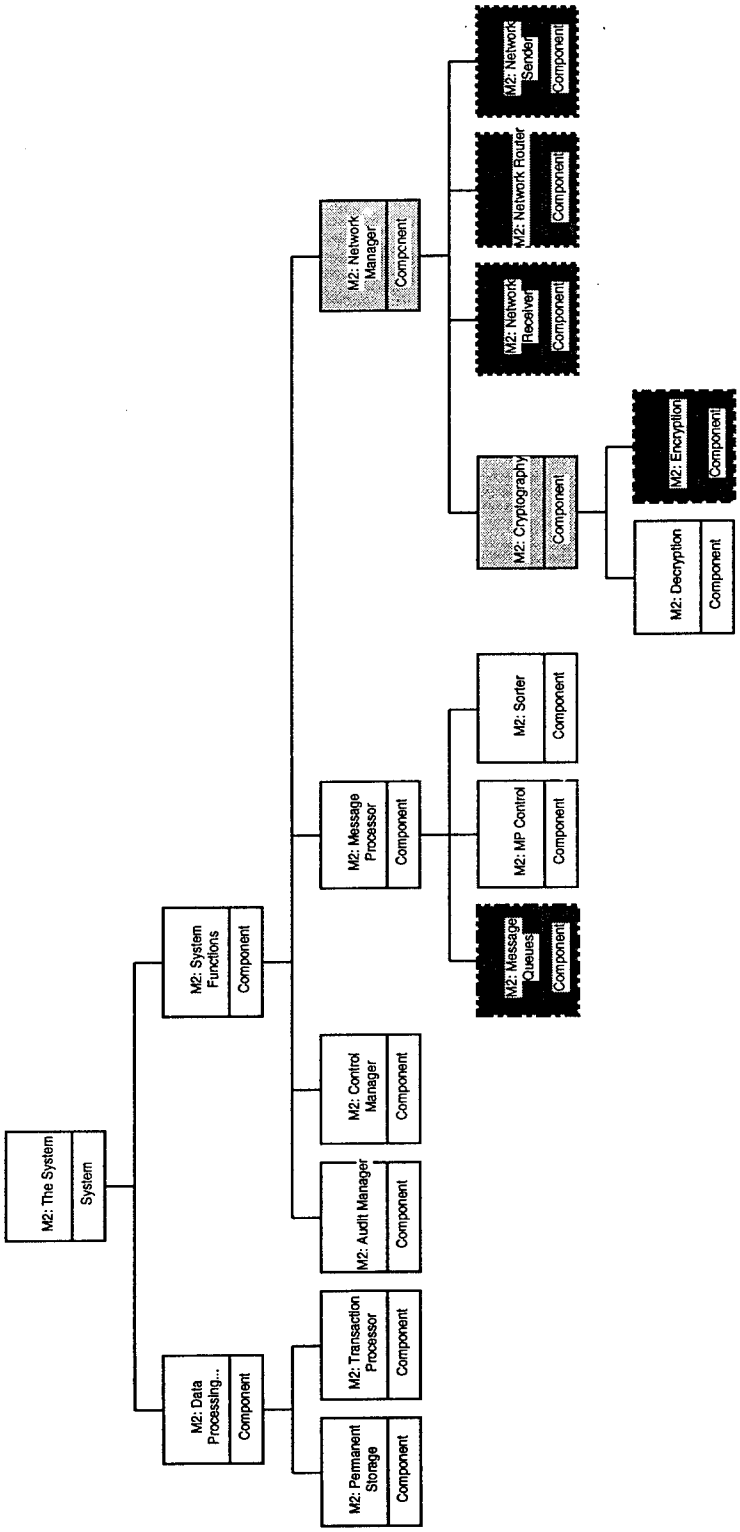


Figure 4.3: Extraordinary Mode

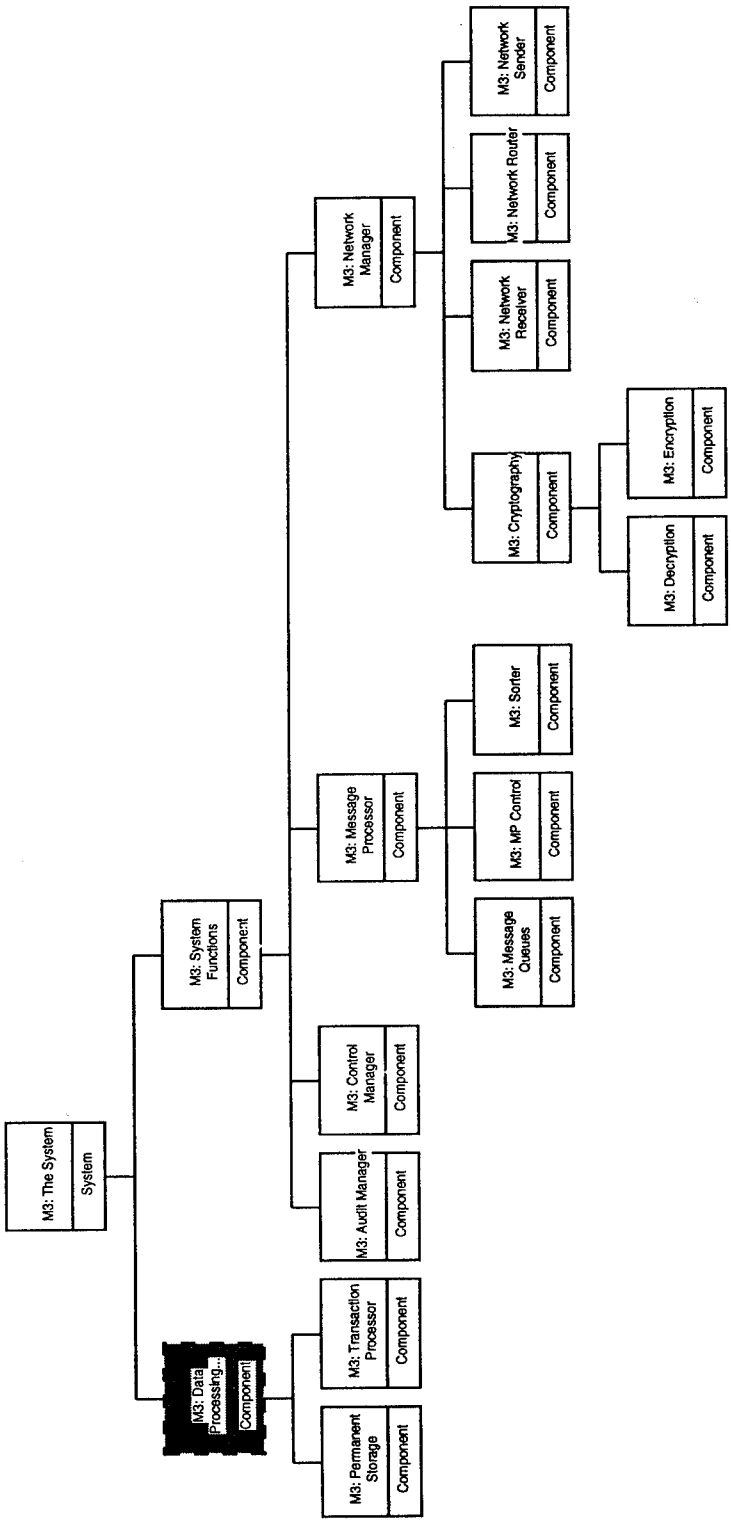


Figure 4.4: Training Mode

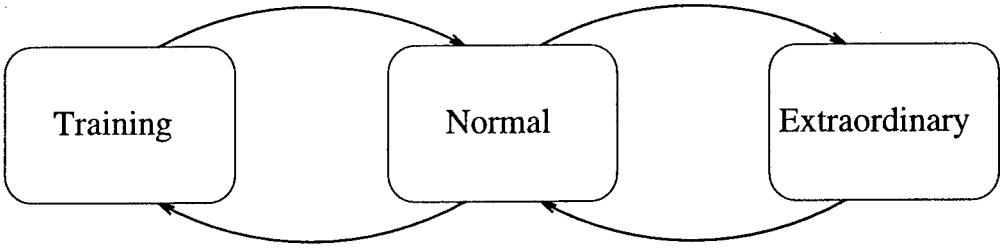


Figure 4.5: The modes and transitions of the example system

Transitions between Modes

A transition of an adaptive system is a change from one mode to another in response to a change in the external environment of the system. Transitions are built into the system; they are a part of the normal operation of an adaptive system. What is important about a transition is how that change in operation of the system affects the security of the system, that is, the security concerns of the system operating in one mode may be different from the security concerns of the system operating in another mode. Also, the transitions may be constrained so that it is only possible for the system to change from certain modes to certain other modes. It may not be possible for an adaptive system to switch between any arbitrary pair of modes.

An adaptive profile will list all the transitions that are possible in the system. Each transition will have a set of security concerns associated with it. A security concern details what is necessary to make sure that the security mechanisms required for the new mode are in place. Security concerns are described in detail in chapter 3.

In the example of Figure 4.1, there are four possible transitions: from Normal to Training, Training to Normal, Normal to Extraordinary, and Extraordinary to Training. There are no direct transitions between Training and Extraordinary modes. These transitions are illustrated in Figure 4.5.

4.3.3 Use of Tools: RDD-100

It is quite feasible to provide tool support for this analysis, using existing tools. As one approach to this, this section shows how to adapt the RDD-100 database schema provided by the System Profile Group of the NSA to support adaptive profiles. First we give an overview of the NSA schema and

then describe our extensions.

System Profile Data Base Schema

RDD-100 is a systems engineering tool that can be used to support profiling. For example, the system profile proposal described in section 4.3.1 uses RDD-100. RDD-100 offers a powerful entity-relationship database and sophisticated graphics that can be used to organize and display large amounts of information and the complex relationships between different pieces of this information.

The RDD-100 database organizes information in terms of instances of different element types and relationships between elements. For example, a system is represented in RDD-100 by instances of elements of type **System** and **Component**. The fact that a system is built from components is represented by the relationship *built from* between elements of type **System** and elements of type **Component**. An example of this can be seen in Figure 4.1 where the box labeled "G: The System" is an instance of type **System** and the boxes labeled "G: Data Processing..." and "G: System Functions" represent the two **Components** that "G: The System" is *built from*. Each element also has attributes that provide information about the element. For example, most elements have a DESCRIPTION attribute that is used to describe the element.

The NSA system profile extends the standard RDD-100 database by adding several element types; three of these types are **Weakness**, **Threat**, and **Vulnerability**. Elements of type **Weakness** can be related to elements of type **Component** by the relationship *exhibits*. Elements of type **Threat** can be related to elements of type **Weakness** by the relationship *exploits*. A **Weakness** is related to any resulting **Vulnerability** by the relationship *results in*.

The DESCRIPTION attribute of a **Weakness** describes the weakness exhibited by the component. The DESCRIPTION attribute of a **Threat** describes the threat to the system. A **Vulnerability** element contains the attributes DESCRIPTION, RECOMMENDATIONS, and SECURITY RISK that give details on how to perform the attack, recommendations for fixes, and describe the ease of exploitation and likelihood of detection, respectively.

Adaptive Profile Data Base Schema

The data base schema for system profiles is extended by adding elements that represent modes and transitions. A mode of a system is represented by an instance of the type **Mode**. The DESCRIPTION attribute of a **Mode** element is used to describe the mode and any information that applies to the mode as a whole. The adaptive components of the mode are grouped together according to the adaptation affecting the components. A single adaptation may affect many components and grouping these components together makes it easy to identify which components are affected by which adaptations. An element type, **Adaptation**, is introduced that is used to group related adaptive components together. The DESCRIPTION attribute of an **Adaptation** is used to describe, in general terms, the particular adaptation. Elements of type **Adaptation** and elements of type **Component** may be related by the relation *directly affects*. All the **Component**s targeted by this relation are affected in some way by this adaptation. A **Component** may be affected by more than one adaptation.

The Extraordinary Mode of Figure 4.3 is represented in the RDD-100 data base using an element named "M2: Extraordinary" of type **Mode**. The targets of the *directly effects* relation for this element are three elements of type **Adaptation** named "M2: Criticality Scheduling", "M2: Short Encryption Keys", and "M2: Alternate Routing". The "M2: Criticality Scheduling" **Adaptation** *directly affects* the "M2: Message Queues" **Component** and represents the scheduling of messages for processing based on their criticality. The "M2: Short Encryption Keys" **Adaptation** *directly effects* the "M2: Encryption" **Component** and represents the use of short encryption keys for the encryption of all outgoing messages. The "M2: Alternate Routing" **Adaptation** *directly effects* the "M2: Encryption", "M2: Network Receiver", "M2: Network Router", and "M2: Network Sender" **Component**s and represents the reconfiguration of the network to allow sending messages over insecure links if there is no secure path available to their destination. Note that the **Component** "M2: Encryption" is involved in two adaptations. In addition each of these **Component**s can have a **Weakness** attached to it if one is uncovered during the security analysis.

A **Mode** must also direct attention to those components that exhibit derived weaknesses. The relation *indirectly affects* relates the **Mode** with the **Component**s that are indirectly affected by the mode. In the example, the

"Extraordinary" **Mode** *indirectly affects* the **Components** "M2: Network Manager" and "M2: Cryptography". Each of these **Components** should have a **Weakness** attached to it.

Initial conditions are associated with a **Mode** using the relation *requires*. The targets of this relation are elements of type **InitialCondition**. Each element of this type describes an initial condition that must be satisfied upon entering the mode. For example, the "Training" **Mode** *requires* the "Disks are Sanitized" **InitialCondition** that requires that all sensitive information has been removed from all disks and the disks sanitized so that no residual sensitive information remains on them.

Transitions between **Modes** are represented by elements of type **Transition**. The DESCRIPTION attribute describes the transition. A **Transition** represents a transition from one **Mode** to another **Mode**; the relations *starts in* and *ends in* relate the **Transition** to the starting **Mode** and the ending **Mode**, respectively.

Each transition has associated with it a set of concerns. These concerns describe what is required to make sure that all the mechanisms are in place to meet the security needs of the next mode. Each of these concerns is described using an element of type **Concern**. A **Transition** is related to its set of **Concerns** using the relation *contains*.

The four transitions of the example are represented by four elements of type **Transition**: "Normal to Training", "Training to Normal", "Normal to Extraordinary", and "Extraordinary to Normal". For the "Normal to Training" **Transition**, the target of the *starts in* relation is "Normal" mode and the target of the *ends in* relation is "Training".

Chapter 5

Future Directions

We give here a brief list of suggested topics for future work on adaptive security.

- Implement operational aspects of dynamic security lattices
  - This would employ both existing and new access control mechanisms, and involve recovery and auditing procedures
- Build tools to support use of task-based policies and the need-to-know model.
  - Such a tool would assist in assigning, understanding, and managing the powers granted to users of a system.
- Extend the systems profiling style of qualitative analysis of risk with quantitative analysis
  - In particular, link the ANSSR quantitative analysis with the NSA-style profiles
- Develop RDD-100 or other tool support of risk analysis
- Get feedback on use of methodology
  - Assist on security design and analysis of a practical example of an adaptive security system

Bibliography

[Asc93] Ascent Logic Corporation. *RDD-100 User's Guide*, September 1993. P/N 100096.

[Bad90] L. Badger. Providing a flexible security override for trusted systems. In *Proceedings of the Computer Security Foundations Workshop III*, pages 115–123. IEEE, June 1990. Franconia, NH.

[BC93] Deborah J. Bodeau and Frederick N. Chase. Modeling constructs for describing a complex system of systems. In *Proceedings of the Ninth Annual Computer Security Applications Conference*, pages 140–148, Orlando, FL, December 1993. IEEE Computer Society Press.

[BCK90] Deborah J. Bodeau, Frederick N. Chase, and Sharon G. Kass. ANSSR: A tool for risk analysis of networked systems. In *Proceedings of the 13th National Computer Security Conference*, pages 687–696, Washington, DC, October 1990.

[BK85] W.E. Boebert and R.Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, pages 18–27, Gaithersburg, MD, October 1985.

[Bod92] Deborah J. Bodeau. A conceptual model for computer security risk analysis. In *Proceedings of the Eighth Annual Computer Security Applications Conference*, pages 56–63, San Antonio, TX, December 1992. IEEE Computer Society Press.

[CW87] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987*

IEEE Symposium on Security and Privacy, pages 184–194, Oakland, CA, April 1987. IEEE.

[Dep85] Department of Defense. *Trusted Computer System Evaluation Criteria*, December 1985. DoD-5200.28-STD.

[Dep88] Department of Defense. *Security Requirements for Automated Information Systems(AISs)*, March 1988. DoDD-5200.28.

[FHOT89] Todd Fine, J. Thomas Haigh, Richard C. O'Brien, and Dana L. Toups. Noninterference and unwinding for LOCK. In *Proceedings of Computer Security Foundations Workshop II*, pages 22–28. IEEE, June 1989. Franconia, NH.

[Fin94] James Fink. *(Draft) A Description of the System Security Profile Report Template.* National Security Agency, April 1994. Revised 4/29/94.

[LL85] C. E. Landwehr and H. O. Lubes. Determining security requirements for complex systems with the Orange Book. In *Proceedings of the 8th National Computer Security Conference*, pages 156–162, Gaithersburg, MD, October 1985.

[Nat85] National Computer Security Center. *Computer Security Requirements: Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, June 1985. CSC-STD-003-85.

[SPV89] Ian Sutherland, Stanley Perlo, and Rammohan Varadarajan. Deducibility security with dynamic level assignments. In *Proceedings of Computer Security Foundations Workshop II*, pages 3–8. IEEE, June 1989. Franconia, NH.

[Ste91] Daniel F. Sterne. On the buzzword "security policy". In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 219–230, Oakland, CA, April 1991. IEEE.

*U.S. GOVERNMENT PRINTING OFFICE: 1995-610-126-50236

Rome Laboratory

Customer Satisfaction Survey

RL-TR-_____

Please complete this survey, and mail to RL/IMPS, 26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and feedback regarding this technical report will allow Rome Laboratory to have a vehicle to continuously improve our methods of research, publication, and customer satisfaction. Your assistance is greatly appreciated.
Thank You

_____

_____

Organization Name:_____(Optional)

Organization POC: _____(Optional)

Address:_____

1. On a scale of 1 to 5 how would you rate the technology developed under this research?

5-Extremely Useful 1-Not Useful/Wasteful

Rating_____

Please use the space below to comment on your rating. Please suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes___ No_____

If yes, please identify the area(s), and comment on what aspects make them "stand out."

3. Do any specific areas of the report stand out as inferior?

Yes___ No___

If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.