To appear in the Proceedings of the 1997 IEEE Symposium on
Security and Privacy, May 4–7, 1997, Oakland, CA.

Secure Software Architectures[*]

Mark Moriconi, Xiaolei Qian, R. A. Riemenschneider, Li Gong[†]

Computer Science Laboratory
SRI International
Menlo Park, California 94025

Abstract

The computer industry is increasingly dependent on open architectural standards for their competitive success. This paper describes a new approach to secure system design in which the various representations of the architecture of a software system are described formally and the desired security properties of the system are proven to hold at the architectural level. The main ideas are illustrated by means of the X/Open Distributed Transaction Processing reference architecture, which is formalized and extended for secure access control as defined by the Bell-LaPadula model. The extension allows vendors to develop individual components independently and with minimal concern about security. Two important observations were gleaned on the implications of incorporating security into software architectures.

Keywords: *secure systems, software architecture, X/Open DTP, formal methods, access control*

1. Introduction

In recent years, there has been a growing demand for vendor-neutral, open systems solutions. These solutions take the form of "open software architectures" that represent a family of systems. Open architectures are critical tools for competitive success in the information technology industry [12]. They enable vendors to substantially reduce time-to-market and development costs, since they do not have to provide an integrated solution in the context of a bewildering array of graphical user interfaces, operating systems, and connectivity schemes. Moreover, open architectures enable consumers to have confidence that future purchases will easily integrate with existing systems.

Several consortia have been created to develop open software architectures. For example, the Object Management Group (OMG), which consists of over 600 companies, has developed a widely accepted architecture, called the Common Object Request Broker (CORBA), that allows applications to communicate seamlessly in a heterogeneous, distributed environment. Interoperation architectures also have been developed by Sun Microsystems (Tooltalk), Xerox PARC (ILU), Open Software Foundation (DCE), and Microsoft (OLE), among others.

An example of an important open application architecture is the X/Open Distributed Transaction Processing (DTP) reference architecture. X/Open DTP is intended to standardize the interactions and communications between the components of the 3-tiered client/server model. Figure 1 illustrates a representative 3-tiered model in which presentation aspects are handled by "thin" clients, business logic is incorporated in the transaction processing monitor (e.g., BEA's Tuxedo™), and data/resource management is the responsibility of data servers (e.g., Oracle 7, IBM/DB2, and Sybase 11).

The X/Open DTP reference architecture allows multiple application programs to share heterogeneous resources provided by multiple resource managers, and allows their work to be coordinated into global transactions. If the X/Open DTP interfaces (APIs) are adhered to, the components of a particular DTP system (namely, every application, transaction manager, resource manager, and resource) will be portable, interchangeable, and interoperable.

Tremendous leverage can be gained by incorporating security directly into architectural standards. This opportunity has been recognized, for example, by OMG in its attempts to extend CORBA for secure object access and communication. An extension of X/Open DTP for secure transaction processing would enable vendors to develop single- or multilevel products with little or no concern about the security of the overall transaction processing system in which they will

[*]This research was supported by the Advanced Research Projects Agency under Rome Laboratory contracts F30602–93–C–0245 and F30602–95–C–0277.

[†]Presently at JavaSoft, 10201 N. De Anza Blvd., Cupertino, California 95014

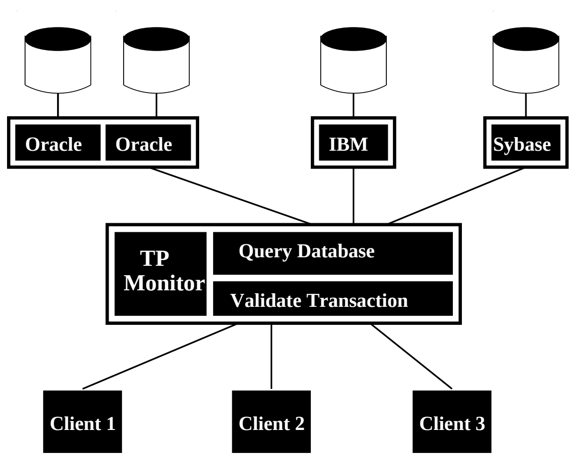

Figure 1. 3-Tiered Client-Server Architecture

be used. Furthermore, such an extension would allow users and application vendors to concentrate on the selection of components, knowing with confidence that the overall DTP system will be secure.

The benefits of a secure architectural standard can be realized only if there is high and justifiable confidence that any instance of it satisfies the intended security property. An erroneous assumption that every instance of an X/Open DTP architecture properly controls access to protected information could have serious legal and business consequences. In this paper, we propose a formal approach to secure architectures that involves three steps:

1. Formalization of the system architecture in terms of common architectural abstractions.
2. Refinement of the system architecture into specialized architectures, each suitable for implementation under different assumptions about the security of the system components.[1]
3. Rigorous proof that every implementation that conforms to the system architecture, or one of its specializations, satisfies the intended security policy.

Our approach is made difficult by the dominance of informal architectures[2] and security theories that are difficult to connect to implementations. Fortunately, we can overcome these difficulties by combining recent results from the software research community [1, 2, 5, 9, 10] with well-known results from the security community.

To illustrate our approach, we present excerpts from our formalization of a secure version of the X/Open DTP standard — called SDTP. It consists mostly of structural information involving component interfaces and the connections among them. It also contains causal dependencies between certain interfaces and connections, but they are not required to demonstrate secure access control. The access control property was chosen because it is well understood and because it is important in the commercial and the military sectors.

The development of a particular secure architecture should take into account two important observations, both of which are illustrated in the subsequent discussion of SDTP.

1. The interplay between architectural and security concerns can dramatically affect the practicality of a solution. For example, the obvious way to add security to X/Open DTP has the consequence that vendors must build multilevel components — even though they are not security experts and may want to target larger, single-level markets.
2. Rather than modeling a system as a single architecture, it should be modeled as multiple, but related, architectures — each making different assumptions about the security of the system components and the protocols.

This paper is organized as follows. The next section discusses how our work differs from previous work in security. Sections 3 and 4 describe the X/Open DTP standard and how it can be formalized in terms of architectural abstractions. Section 5 presents three useful SDTP architectures and proves that they satisfy an access control policy based on the Bell-LaPadula model. Covert channels are not considered in this model. Section 6 proves that the three SDTP architectures are related in a manner that preserves secure access. In particular, we show that all three are "conservative refinements" of a more general architectural model of SDTP. The abstract model is suitable as a common abstraction, but it is not concrete enough to serve as a standard architecture. The final section summarizes our results and discusses future work.

2. Related Work in Security

There are two main challenges in developing secure (software or hardware) systems. One is to obtain a thorough understanding of the desired security properties and the relevant (functional and non-functional) properties of the target system. This normally calls for a formal model of the system together with security proofs, often given in some mathematical or logical language. The second challenge is to assure that the actual system implementation realizes the more abstract formal model.

The research community has spent considerable effort on the first of these challenges. The security properties

[1]Refinement also can be used to accomodate different computing and networking environments, but that is not the focus of this paper.

[2]The X/Open DTP standard consists of approximately 500 pages of diagrams, C interfaces, and English explanations.

considered include non-interference, information-flow, and composability, with system models built using traces, CSP, and other formal languages [3, 6, 7, 8]. These behavioral models are far removed from the actual systems, making it extremely hard, if not impossible, to be convinced that an implementation satisfies the security properties proven about the model.

On the other hand, commercially-available systems, such as OSF's DCE 1.1, include a wide range of security functionalities, such as authentication and access control. But these architectures cannot be linked formally to a solid theory of security and, given the overall complexity of these architectures, it is difficult to distill whether or not they provide the desired security properties. The same observation applies to uses of the "reference monitor" concept that was popular in the 1980s. Systems designed around this concept could not reliably be connected to implementations.

A middle ground between abstract mathematical modeling and system-level analysis can be seen in connection with the Rampart system [15]. Rampart is a distributed system for which an attempt was made to prove that it satisfies certain security and fault-tolerance properties. However, the proof is with regard to an abstract algorithmic model, which again suffers from the difficulty in relating it to the actual implementation.

A successful effort to connect security requirements for a secure gateway to its implementation is described in [13, 14]. A convincing, but not formal, argument is made that the connection preserves security. The argument depends on the use of architectural elements that are very similar to those used in the implementation. Our architecture descriptions [9, 10, 11] are more general in three important ways. First, an architecture hierarchy can include both horizontal and vertical decomposition (change in representation), whereas the gateway development involves only horizontal. Second, formal mappings between architectures are used to bridge the gap in vertical hierarchies. Third, our architecture definition language makes it easy to define generic architectures. These three capabilities are useful in defining practical architecture standards, such as X/Open DTP, and are useful in architecting any large system. By formalizing mappings and correctness arguments, we can prove that lower levels in a vertical hierarchy — ultimately, the implementation — preserve security properties proved at higher levels.

3. The X/Open DTP Standard

Distributed transaction processing captures the core activities in distributed information systems, namely the interaction between applications and resources. Transactions form the basis of such interaction, which are multiple application programs running concurrently and accessing shared data resources. Distributed transaction processing is ubiquitous in military and commercial applications, many of which have stringent security requirements.

The X/Open DTP standard architecture is described in a series of publications of X/Open Ltd. [16, 17, 18, 19]. An executable prototype of the DTP standard appears in Luckham et al. [5]. The standard describes a particular set of component interfaces, and sequences of interactions between the components. Components may be various application programs, resource managers (such as databases, file systems, or mailers) and transaction managers. The purpose is to define a standard communication architecture through which multiple application programs may share resources concurrently by organizing their activities into transactions which appear to be atomic. Transactions, which may execute concurrently, can contain many operations on resources. *Atomicity* means that after a transaction executes, either all or none of its operations take effect.

The X/Open description is informal, consisting of English text together with interfaces given in the C programming language. The important features of the standard are (1) the interfaces, (2) the architecture (the ways of connecting components that satisfy the standard), and (3) the protocols (calling sequences) for using the interfaces. The calling sequences are described in terms of a single thread of control and C function calls. Many different systems with various applications and resources may satisfy this standard. Those that do should be easier to combine, thus promoting the goal of "open" systems.

A version of the X/Open architecture, shown in Figure 2, consists of three types of components — one application program (AP), one transaction manager (TM), and one or more resource managers (RMs). The boxes indicate the component interfaces, and the lines indicate the communication between them. The label TX indicates a complex connection and protocol defining communication between any application module and any transaction manager. The TX connection contains connections between functions for initiating and finalizing transactions. Communication is always initiated by the application. A series of calls back and forth continues until communication is completed. Similar complex connections exist between the application and every resource (the AR connection), and between the transaction manager and every resource manager (the XA connection). The XA connection involves the well-known two-phase commit protocol for ensuring atomicity. Much of this activity can be concurrent, and many transactions may take place at once.

4. Formal Model of X/Open DTP

We first show how to define the X/Open DTP interfaces, components, and wiring using an architecture definition lan-

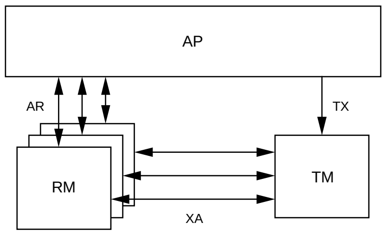

Figure 2. X/Open DTP Reference Architecture

guage called SADL [11].[3] Then, we describe how architectures written in SADL can be translated systematically into logic. All of the proofs in this paper can be fully formalized in terms of the logical theories that result from the translation.

An interface is defined as a type so that it can be associated with more than one component. For example, we define the interface for transaction managers as follows.

```
tm: TYPE <=
  MODULE
  EXPORTING ALL
  BEGIN
    register: AX_Register_Procedure
      [id: X_Id, rmid: INT, flags: INT
         -> ret: INT]
    unregister: AX_Unregister_Procedure
      [rmid: INT, flags: INT
         -> ret: INT]
    begin: TX_Begin_Procedure
      [ -> ret: INT]
    close: TX_Close_Procedure
      [ -> ret: INT]
    commit: TX_Commit_Procedure
      [ -> ret: INT]
    information: TX_Info_Procedure
      [info: TX_Info -> ret: INT]
    open: TX_Open_Procedure
      [ -> ret: INT]
    rollback: TX_Rollback_Procedure
      [ -> ret: INT]
  END
```

This declaration says that every object of type `tm` is a module containing a definition of the procedures `register`, `unregister`, `begin`, and so on, used in the X/Open specification of the TX interface. For example, `begin` is declared to be a `TX_Begin_Procedure`, which guarantees that it has an appropriate "semantics" relative to its role in the protocol. `TX_Begin_Procedure` is declared elsewhere to be a subtype of `Procedure`.

As an example of a more complex interface, consider the definition of the application interfaces, which must allow for an arbitrary number of resources.

```
   ap: TYPE <=
      MODULE
         [<< ap_in1(i): r_type(i)
                     |(i: NAT) i < n >>
            -> << ap_out1(i): q_type(i)
                     |(i: NAT) i < n >>]
```

[3] SADL is an acronym for Structural Architecture Definition Language.

This declaration says that an object of type `ap` is a module with `n` input ports (`n` is the number of resource managers in the system being specified) and `n` output ports. The type of the `i`-th input port, `ap_in1(i)`, is `r_type(i)`, where `r_type` has been declared to be a function from non-negative integers less than `n` to subtypes of the type `ar_resources` of data sent from the RMs to the AP. Similarly, the type of the `i`-th output port, `ap_out1(i)`, is `q_type(i)`, where `q_type` has been declared to be a function from non-negative integers less than `n` to subtypes of the type `ar_requests` of data sent from the AP to the RMs.

The declaration of a resource manager, denoted by the `rm` type, contains procedure calls, as in the `tm` declaration, and ports, as in the `ap` declaration. Since an instance of DTP can contain an arbitrary number of resources, the type `rms` is used to denote the union of an arbitrary number of resource managers.

To define a component, we simply declare an instance of an interface type. The declarations

```
  the_ap:  ap
  the_rms: rms
  the_tm:  tm
```

define the three components in X/Open DTP.

The interfaces of these components are wired together by constraints associated with the AR, XA, and TX connections. Here is a representative constraint on the TX connection.

```
  tx_1: ASSERTION =
    Called_From(the_tm.begin, the_ap)
```

says that the procedure `begin` of the TX protocol declared in `the_tm` is called by `the_ap`. The XA interface definition consists of twelve assertions that are similar, except that they are general so as to apply to all RMs. An example is

```
  xa_1: ASSERTION =
    (FORALL y: rm)
      Called_From(the_tm.register, y)
```

The AR interface specification is more abstract, since it is described in terms of a general dataflow connection which need not be implemented as a procedure call. The assertion

```
  ar_1: ASSERTION =
    (FORALL i: NAT | i < n)
     (EXISTS c: Channel[q_type(i)])
       Connects(c, the_ap.ap_out1(i),
         the_rms.rm_in1(i))
```

says that, for every RM index `i`, there is a dataflow channel `c` which carries the appropriate subtype of `ar_requests` from the `i`-th output port of `the_ap` to the input port of the `i`-th RM in the collection of all RMs, `the_rms`. A similar assertion characterizes the flow from the RMs to the AP.

SADL specifications can be translated systematically into logic, allowing us to regard architectures as logical theories. As an illustration, consider the declaration

```
begin: TX_Begin_Procedure
  [ -> ret: INT ]
```

that appears in the `tm` type. It is translated into

$$\forall x\,[\,\mathrm{TM}(x) \longrightarrow \exists y\,\mathrm{TX\_Begin\_Procedure}(y,x)\,]$$

$$\forall x\,\forall y\,[\,\mathrm{TM}(x) \wedge \mathrm{TX\_Begin\_Procedure}(y,x) \longrightarrow \exists z\,\mathrm{Return\_Parameter}(z,y)\,]$$

$$\forall x\,\forall y\,\forall z\,\forall w\,[\,\mathrm{TM}(x) \wedge \mathrm{TX\_Begin\_Procedure}(y,x) \wedge \mathrm{Return\_Parameter}(z,y) \wedge \mathrm{Integer}(w) \longrightarrow \mathrm{May\_Hold\_Value\_On\_Return}(z,w)\,]$$

This says that every transaction manager has a begin procedure, that every begin procedure has a return parameter, and that the value of a begin procedure's return parameter can be an integer.

Among the assertions that define the TX interface is

```
Called_From(the_tm.begin, the_ap)
```

which can be translated to

$$\forall x\,[\,\mathrm{TX\_Begin\_Procedure}(x,\mathrm{the\_tm}) \longrightarrow \exists y\,[\,\mathrm{Call\_Site}(y,x) \wedge \mathrm{Location}(y,\mathrm{the\_ap})\,]\,]$$

This says that every TX begin procedure is called from a site located in the application.

Translation to logic is essential for complicated arguments, such as the relative correctness proofs of Section 6, and when fully formalized arguments are required. But often, simpler results can be directly established by reasoning in terms of SADL specifications. The next section, which proves the security of alternative DTP architectures, illustrates this kind of rigorous informal reasoning.

5. Secure DTP Architectures

Suppose that we want to enforce the multilevel security (MLS) policy in the DTP architecture. A standard model of the MLS policy is the Bell-LaPadula model [4]. Given a set of subjects each with an attached clearance level, and a set of objects each with an attached classification level, the model ensures that information does not flow downward in a security lattice by imposing the following requirements.

- **The Simple Security Property**. A subject is allowed a read access to an object only if the former's clearance level is identical to or higher than the latter's classification level in the lattice.

- **The $*$-Property**. A subject is allowed a write access to an object only if the former's clearance level is identical to or lower than the latter's classification level in the lattice.

In terms of the DTP, a subject might be a user invoking an application or a transaction manager, and an object could be any data item in a resource. We say a component in DTP is MLS, if input and output of that component are properly labeled with security levels, and the component enforces the MLS policy internally. Similarly, the DTP architecture is MLS, if it enforces the MLS policy.[4]

Notice that the MLS policy regulates the communication between the applications and the resources, which is application-specific and not part of the DTP standard. It is not obvious how the MLS policy can be enforced while still maintaining plug-and-play.

A naive approach is illustrated in Figure 3, where each (non-MLS) application or resource is wrapped by an MLS wrapper, which together enforce the MLS policy at the interface. This approach suffers from at least three problems.

- High-assurance technology does not yet exist that allows a non-MLS component (application or resource) to be wrapped to enforce the MLS policy, especially when the component contains untrusted code.

- Even if an MLS wrapper technology did exist, the enforcement of the MLS policy by a wrapper is likely dependent on the application-specific interface of the underlying component, which destroys the plug-and-play benefit of a standard architecture.

- The alternative is to require that vendors provide MLS components, which would have significant drawbacks: reduced time-to-market, stiff performance penalties, and increased development costs.

Given the fact that most of the COTS and legacy components available today are not MLS, our strategy is therefore to develop not one but several secure DTP standards, each geared toward a particular configuration of applications and resources. In the remainder of this section, we consider three possible configurations.

The simplest configuration is when the application and all the resources are single-level and are at the same level, as shown in Figure 4. The DTP standard does not need any extension to enforce the MAC policy, since there cannot

[4]We concern ourselves only with the two properties of the Bell-LaPadula model, not with issues such as object reuse or covert channels.

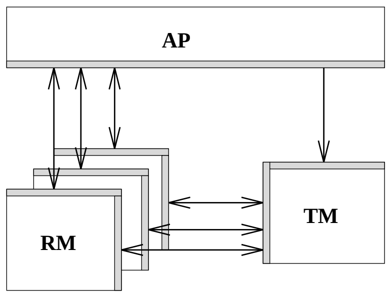

Figure 3. Secure DTP with MLS Wrappers

be any downward (or cross-level) information flow in the security lattice.

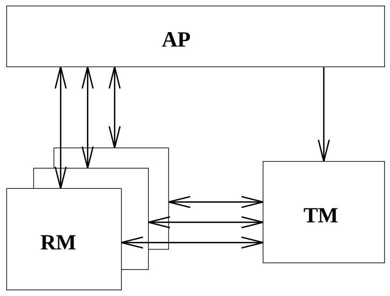

Figure 4. Secure DTP with Same-Level AP and RMs

A second possible configuration is when the application and all the resources are single-level but perhaps at different levels, as shown in Figure 5. To enforce the MAC policy, the AP and the RMs cannot directly communicate with each other. Instead, communication has to be filtered by an MLS filter, which regulates every access by the AP to the RMs to ensure the simple security property and the $*$-property of the Bell-LaPadula model. The TM is required to be MLS since it interacts with components at different levels.

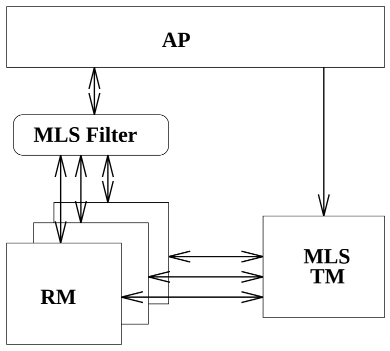

Figure 5. Secure DTP with Single-Level AP and RMs

For this scenario, the DTP standard needs to be extended as follows. The begin and register procedure calls in the TM interface are extended with a label parameter to indicate the level of the RM or AP caller.

```
tm: TYPE <=
  MODULE
  EXPORTING ALL
  BEGIN
    register: AX_Register_Procedure
      [id: X_Id, rmid: INT,
            flags: INT,
            lb: LABEL
         -> ret: INT]
    begin: TX_Begin_Procedure
      [lb: LABEL -> ret: INT]
    ...
  END
```

The security of the resulting DTP architecture with respect to the MLS policy can be shown as follows. Suppose that the AP is low and a particular RM r is high. Let us consider the simple security property of the Bell-LaPadula model. The only way that AP can read data in r is through either the TM or the filter. Since communication is secure, any data from r to the TM/filter will be properly labeled high. Since the TM/filter is MLS, it cannot pass high data to the low AP, meaning that AP cannot read data from r. Similarly we can argue that the $*$-property of the Bell-LaPadula model holds.

The most complicated configuration, shown in Figure 6, is when the application and the resources are multilevel, in contrast to the wrapped components in Figure 3. The AP and the RMs can directly communicate with each other, since they are capable of handling data at different levels. Again, the TM should be MLS as well. Conceptually every multilevel connection can be viewed as consisting of multiple single-level connections, one for each level.

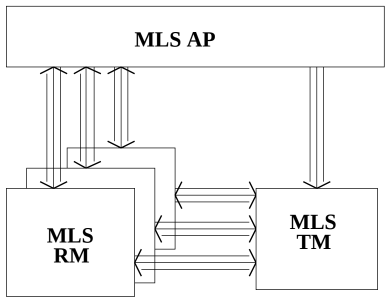

Figure 6. Secure DTP with Multilevel AP and RMs

The DTP standard needs to be extended as follows. Every AP and RM interface is extended with a label-range parameter to indicate the range of levels of the AP and RM respectively. Like before, the register and begin procedure calls in the TM interface are extended with a label parameter to indicate the level of the RM or AP caller. In addition, there is a constraint requiring that there is a register procedure call for every level in every RM's range. A similar constraint is needed for the begin procedure call.

```
rm: TYPE <=
  MODULE
  EXPORTING ALL
  BEGIN
    low:  LABEL
    high: LABEL
    label_order: ASSERTION =
      low <= high
    ...
  END

tm: TYPE <=
  MODULE
  EXPORTING ALL
  BEGIN
    register(lb: LABEL):
      AX_Register_Procedure
        [id: X_Id, rmid: INT,
              flags: INT
           -> ret: INT]
    ...
  END

the_tm: tm

rms_label: ASSERTION =
  (FORALL x: rm)(FORALL y: LABEL)
    [x.low <= y AND y <= x.high
       => Called_From
            (the_tm.register(y), x)]
```

It is straightforward to prove the security of this architecture. Since every connection is single-level, and every component is MLS, a component cannot send data through a connection if the level of the data is different from that of the connection. Since communication is secure, any data received via a connection will have the same level as that of the connection. Given that every component properly enforces the two properties of the Bell-LaPadula model, so does the resulting architecture.

6. Relating the SDTP Architectures

We can relate the four SDTP architectures by defining a general model for SDTP and then proving that each of the four SDTP architectures are correct specializations of it.

6.1. Correctness Criterion

The traditional interpretation of relative correctness is not strong enough for our purposes because it only requires that the concrete architecture extend the abstract architecture. Hence, a low-level architecture can introduce properties that contradict the higher-level architecture, which can lead to a violation of security properties in the reference architecture. For example, consider the proof in Section 5 that the SDTP architecture in Figure 5 is secure. The proof critically depends on the fact that the only way that AP and RMs communicate is through either the MLS TM or the MLS filter. The low-level architecture can extend this by including a new non-MLS flow from AP to an RM. This extended low-level architecture by definition "implements" the high-level architecture, but the extension clearly violates multilevel security requirements.

We have developed a new correctness criterion for architectures in which a concrete architecture implements *exactly* the abstract architecture, no more and no less [10]. In other words, a security hole that does not exist, with respect to a given security policy, in the abstract architecture will not come into existence in any concrete architecture. To use our earlier example, under our new correctness criteria, it is possible to ensure that no new data flow can be introduced in the low-level architecture. That is, if flow from B to C exists in the low-level architecture, then this flow must already exist in the high-level architecture, either as a direct or indirect flow. This flow thus implies that the high-level design is not secure and should be modified. Conversely, if we can prove that the high-level architecture is secure, then we can rest assured that the mapped low-level architecture is also secure.

To formalize this notion, we first need to make explicit an important *completeness assumption* about architectures. In particular, we assume that an architecture contains *all* components, interfaces, and connections intended to be true of the architecture *at its level of detail*. If a fact is not explicit in the architecture, or deducible from it, we assume that it is not intended to be true of the architecture. In general, an architecture (whether static or dynamic) can contain an unbounded number of facts.

Formally, we need to prove that two architectures, represented as theories, are correct with respect to an interpretation mapping between them and the completeness assumption. Let Θ and Θ' be theories associated with an abstract and a concrete architecture, respectively. Let I be an interpretation mapping from the language of Θ to the language of Θ'. For every sentence F, mapping I is a *theory interpretation* provided

$$\text{if} \;\; F \in \Theta \;\; \text{then} \;\; I(F) \in \Theta'$$

This is the usual definition of correctness.

Since a given architecture is assumed to be complete with respect to its level of detail, we additionally require that the concrete architecture add no new facts about the abstract architecture. To prove this, we must additionally show that

$$\text{if} \;\; F \notin \Theta \;\; \text{then} \;\; I(F) \notin \Theta'$$

in which case, I is a *faithful interpretation*. This says that, if a sentence is not in the abstract theory, its image cannot be in the concrete theory. (Observe that Θ' is a conservative extension of Θ provided the identity map faithfully interprets Θ in Θ'.) Theory Θ' *can* contain sentences not in $I[\Theta]$ — i.e., new facts about the architecture not included in Θ — but these facts must not have any consequences expressible in the language of Θ.

6.2. The General Model

Let the AP, TM, and RMs be represented as functional components with labeled input and output ports connected by secure channels. More specifically, we require that

- Data have associated *security levels*, which are collections of authentication data and credentials that characterize the data's security standing. Thus, a datum can be thought of as a value together with a security label that determines its level.
- The ports where the data is supplied and received have associated *clearance levels*.
- The channels that carry data also have associated *clearance levels*.

The connections in the general model that represent the AR, XA, and TX interfaces are subject to three constraints.

1. An input port can receive data only if its clearance level dominates the data's security level.
2. An output port can supply data only if its clearance level is dominated by the data's security level.
3. A channel can carry data only if its clearance level dominates the data's security level.

If these constraints are satisfied, we may refer to ports as *secure ports* and to channels as *secure channels*.

6.3. Example Proof Relating Two Architectures

It can be proven that the four SDTP architectures correctly implement the general model. In this section, we focus on the most complicated proof, which involves Figure 4. Specifically, we prove that a combination of writing and reading data that is mediated by the MLS filter correctly implements secure dataflow.

This requires showing that a mapping I from the language of secure dataflow to the language of mediated reading and writing is a faithful interpretation of the theory of secure dataflow, Θ, in the theory of mediated reading and writing, Θ'. The interesting predicates in the language of secure dataflow are

- $\mathsf{Secure\_Channel}(\mathsf{x})$, which means that x is a secure channel,
- $\mathsf{Connects}(\mathsf{x},\mathsf{y},\mathsf{z})$, which means that secure channel x connects secure output port y to z,
- $\mathsf{Can\_Carry}(\mathsf{x},\mathsf{y})$, which means that secure channel x can carry datum y (i.e., that the datum contains an appropriate type value and has a security level no higher than the clearance level of the channel),
- $\mathsf{Clearance\_Level}(\mathsf{x},\mathsf{y})$, which means that either the security label x is greater than or equal to the clearance level of y, when y is either an output port or channel, or the security label x is less than or equal to the clearance level of y, when y is either an input port, and
- $\mathsf{Security\_Level}(\mathsf{x},\mathsf{y})$, which means that either the security label x is less than or equal to the security level of the datum y,

together with a lattice ordering $\leq$ of security labels.

The predicates of interest in the language of MLS-mediated reading and writing are

- $\mathsf{MLS\_Filter}(\mathsf{x})$, which means that x is an MLS filter,
- $\mathsf{Secure\_Write}(\mathsf{x},\mathsf{y})$, which means that x is a call site that performs a secure write to y,
- $\mathsf{Secure\_Read}(\mathsf{x},\mathsf{y})$, which means that x is a call site that performs a secure read of y,
- $\mathsf{Filter\_Passes}(\mathsf{x},\mathsf{y})$, which means that MLS filter x can pass datum y (i.e., that the datum contains an appropriate type value and has a security level no higher than the clearance level of the secure read and no lower than the clearance level of the secure write),
- $\mathsf{Clearance\_Level}(\mathsf{x},\mathsf{y})$, which has the same meaning as in the secure dataflow language, and
- $\mathsf{Security\_Level}(\mathsf{x},\mathsf{y})$, which which also has the same meaning as in the secure dataflow language,

together with a lattice ordering of security labels which is also called $\leq$.

The relevant part of I can be defined as follows:

$$\begin{aligned}
&I(\mathsf{Secure\_Channel}(x)) \\
&\qquad = \mathsf{MLS\_Filter}(I(x)) \\
&I(\mathsf{Connects}(x, y, z)) \\
&\qquad = \mathsf{Secure\_Write}(I(y), I(x)) \\
&\qquad\qquad \land \mathsf{Secure\_Read}(I(z), I(x)) \\
&I(\mathsf{Can\_Carry}(x, y)) \\
&\qquad = \mathsf{Filter\_Passes}(I(x), I(y)) \\
&I(\mathsf{Clearance\_Level}(x, y)) \\
&\qquad = \mathsf{Clearance\_Level}(I(x), I(y)) \\
&I(\mathsf{Security\_Level}(x, y)) \\
&\qquad = \mathsf{Security\_Level}(I(x), I(y)) \\
&I(x \leq y) = (I(x) \leq I(y))
\end{aligned}$$

together with clauses that map each abstract level secure channel to a concrete level mediated read and write combination.

Note that I naturally determines a mapping I' from structures of the *concrete* language to structures of the *abstract* language. Given a structure $\mathbf{M}'$ of the concrete language, $I'(\mathbf{M}')$ is defined as follows. The universe of $I'(\mathbf{M}')$ is the same as $|\mathbf{M}'|$, the universe of $\mathbf{M}'$. If I maps atomic formula $P(x_1, x_2, \ldots, x_n)$ to concrete formula A, then the extension of P in $I'(\mathbf{M}')$ is the set of n-tuples from the universe of $\mathbf{M}'$ that satisfy A. That is, the extension of P in $I'(\mathbf{M}')$ is

$$\begin{aligned}
&\{\langle m_1, m_2, \ldots, m_n\rangle \in |\mathbf{M}'|^n: \\
&\quad \mathbf{M}' \models A[x_1/m_1, x_2/m_2, \ldots, x_n/m_n]\}
\end{aligned}$$

Showing that I is a theory interpretation is simply a matter of formally deriving the concrete level interpretation of each abstract level axiom from the concrete level axioms. For example, the image of the security constraint

$$\begin{aligned}
&\forall \mathsf{x}\, \forall \mathsf{y}\, \forall \mathsf{z}\, \forall \mathsf{w} \\
&\quad [\, \mathsf{Secure\_Channel}(\mathsf{x}) \\
&\qquad \land \mathsf{Clearance\_Level}(\mathsf{x}, \mathsf{y}) \\
&\qquad \land \mathsf{Datum}(\mathsf{z}) \\
&\qquad \land \mathsf{Security\_Level}(\mathsf{z}, \mathsf{w}) \\
&\qquad \land \mathsf{Can\_Carry}(\mathsf{x}, \mathsf{z}) \\
&\qquad\qquad\qquad \to \mathsf{z} \leq \mathsf{y}\,]
\end{aligned}$$

must be derived from the axioms describing the properties of the filter. This is straightforward.

To show that I is faithful, we will describe a mapping J from models of abstract level theory Θ to models of the concrete level theory Θ' such that, for every model $\mathbf{M}$ of Θ, $I'(J(\mathbf{M}))$ is isomorphic to $\mathbf{M}$. The faithfulness of I follows, by the model-theoretic result cited in [10].

As the definition of I suggests, the only difficulty in "inverting" the interpretation is handling the equation for Connects. But is it easy to see that letting the extension of Secure_Read in $J(\mathbf{M})$ be

$$\begin{aligned}
&\{\langle m_1, m_2\rangle \in |\mathbf{M}|^2\text{: for some } m_3 \text{ in } |\mathbf{M}|, \\
&\quad \mathbf{M} \models \mathsf{Connects}(x_1, x_2, x_3)[x_1/m_2, x_2/m_3, x_3/m_1]\}
\end{aligned}$$

and letting the extension of Secure_Write in $J(\mathbf{M})$ be

$$\begin{aligned}
&\{\langle m_1, m_2\rangle \in |\mathbf{M}|^2\text{: for some } m_3 \text{ in } |\mathbf{M}|, \\
&\quad \mathbf{M} \models \mathsf{Connects}(x_1, x_2, x_3)[x_1/m_2, x_2/m_1, x_3/m_3]\}
\end{aligned}$$

will yield the required structure.

7. Discussion

We have combined results from the software and security research communities to form a new methodology for the construction of secure architectures. The method involves the formalization of a system architecture with security mechanisms embedded directly in the architecture. More specifically, the mechanisms are intended to provide secure access control as defined by the Bell-LaPadula model (the simple security property and the $*$–property). A proof about the security of a system implementation is performed by reasoning about its architecture. If an architecture is secure, every valid instance of it is secure. Architecture instantiation is equivalent semantically to theory instantiation.

An important contribution of our work is the modeling of a system in terms of multiple secure architectures that are related by formal mappings. The architectures may represent both horizontal and vertical decompositions. Proper application of our modeling technique enables vendors to develop single- or multilevel products with little or no knowledge about the overall application. Similarly, customers can select single- or multilevel components, knowing with confidence that the security of the overall system will be intact. An example of this was seen in the SDTP development, where the classification of a component had a radical effect on the SDTP architecture. Our modeling technique also offers benefits to the system architect, who can initially develop a simple, abstract system architecture that is easy to reason about but is too restrictive for implementation purposes. We saw this in the abstract SDTP architecture, which requires that all components provide multilevel access control. The gap between it and the three more concrete system architectures, which are less restrictive with respect to security, was bridged by refinement mappings that were shown to preserve the security property of the abstract architecture.

It is important to mention that our approach to architecture modeling is application independent. In this paper, we concentrated on the X/Open DTP reference architecture to maximize the commercial relevance of our work. However, our approach is not tied to any particular application and should apply equally well in the development of other secure architectures.

Future work includes the development of standard refinement rules for implementing secure architectures in execution environments containing existing or emerging security standards. It also includes an extension of SADL to model behavior-related security properties using the RAPIDE architecture prototyping language. This would make it possible, for example, to reason about covert channels in terms of a system architecture.

References

[1] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of SIGSOFT'94: Foundations of Software Engineering*. ACM Press, December 1994.

[2] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4), April 1995.

[3] J. Goguen and J. Meseguer. Security Polices and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, California, April 1982.

[4] C. Landwehr. Formal Models for Computer Security. *ACM Computing Survey*, 13(3):247–278, September 1981.

[5] D. Luckham, L. Augustin, J. Kenney, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, April 1995.

[6] D. McCullough. A Hookup Theorem for Multilevel Security. *IEEE Transactions on Software Engineering*, 16(6):563–568, June 1990.

[7] J. McLean. The Specification and Modeling of Computer Security. *IEEE Computer*, 23(1):9–16, January 1990.

[8] J. McLean. A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 79–93, Oakland, California, May 1994.

[9] M. Moriconi and X. Qian. Correctness and composition of software architectures. In *Proceedings of ACM SIGSOFT'94: Symposium on Foundations of Software Engineering*, pages 164–174, New Orleans, Louisiana, December 1994.

[10] M. Moriconi, X. Qian, and R. Riemenschneider. Correct architecture refinement. *IEEE Transactions on Software Engineering*, 21(4):356–372, April 1995.

[11] M. Moriconi and R. A. Riemenschneider. Introduction to SADL 1.0: a language for specifying software architecture hierarchies. Technical report, Computer Science Laboratory, SRI International, 1996.

[12] C. Morris and C. Ferguson. How architecture wins technology wars. *Harvard Business Review*, pages 86–96, March–April 1993.

[13] R. B. Neely and J. W. Freeman. Structuring systems for formal verification. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 2–13, Oakland, CA, April 1985.

[14] R. B. Neely, J. W. Freeman, and M. D. Krenzin. Achieving understandable results in a formal design verification. In *Proceedings of the Computer Security Foundations Workshop II*, pages 115–124, Franconia, NH, June 1989.

[15] M. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, Fairfax, Virginia, November 1994.

[16] X/Open Company Ltd., Apex Plaza, Forbury Road, Reading, Berkshire RGI 1AX, U.K. *Distributed Transaction Processing: The XA Specification*, June 1991.

[17] X/Open Company Ltd., Apex Plaza, Forbury Road, Reading, Berkshire RGI 1AX, U.K. *Distributed Transaction Processing: The TX (Transaction Demarcation) Specification*, November 1992.

[18] X/Open Company Ltd., Apex Plaza, Forbury Road, Reading, Berkshire RGI 1AX, U.K. *Distributed Transaction Processing: The Peer-to-Peer Specification*, December 1992.

[19] X/Open Company Ltd., Apex Plaza, Forbury Road, Reading, Berkshire RGI 1AX, U.K. *Distributed Transaction Processing: Reference Model, Version 2*, November 1993.