Robust Declassification

Steve Zdancewic Andrew C. Myers
Cornell University Computer Science Department
Upson Hall Ithaca, NY 14853
{zdance, andru}@cs.cornell.edu
Revised June 20, 2001

Abstract

Security properties based on information flow, such as noninterference, provide strong guarantees that confidentiality is maintained. However, programs often need to leak some amount of confidential information in order to serve their intended purpose, and thus violate noninterference. Real systems that control information flow often include mechanisms for downgrading or declassifying information; however, declassification can easily result in the unexpected release of confidential information.

This paper introduces a formal model of information flow in systems that include intentional information leaks and shows how to characterize what information leaks. Further, we define a notion of robustness *for systems that include information leaks introduced by declassification. Robust systems have the property that an attacker is unable to exploit declassification channels to obtain more confidential information than was intended to be released. We show that all systems satisfying a noninterference-like property are robust; for other systems, robustness involves a nontrivial interaction between confidentiality and integrity properties. We expect this model to provide new tools for the characterization of information flow properties in the presence of intentional information leaks.*

1 Introduction

Information flow control has for some time offered the promise of a higher-level approach to maintaining the confidentiality and integrity of data. Policies for the flow of information, such as noninterference [8], have the advantage over access control policies in that they can conveniently express precise, system-wide restrictions on the flow of sensitive data. The use of information flow controls has been only partially successful, however. Enforcement mechanisms have often been overly restrictive, preventing useful systems from being built. An even greater difficulty is that real systems often do leak some amount of confidential information, by intention. For example, even a program that checks passwords leaks a small amount of information (in an information-theoretic sense) about the actual passwords, when queried with an incorrect password.

To accommodate programs that leak information by design, information flow controls often include some notion of *declassifying* information (*downgrading* the sensitivity labels on the data). Because the use of declassification may violate information flow policies, its invocation is limited to appropriately trusted subjects. One difficulty with the addition of a declassification mechanism is deciding when the declassification is appropriate. Once a channel is added to the system along which sensitivity labels are downgraded, there is the potential for the channel to be abused to release sensitive information other than that intended.

For example, consider a subroutine that checks passwords. If a user has access to another subroutine that allows the user's own password to be modified freely, this pair of routines can be used to launder sensitive data one bit at a time, as follows. A sensitive boolean value is encoded in the password that the user assigns himself; this value is then laundered by checking whether the user's password is one of the encodings. Thus, the declassification needed in order to reduce the sensitivity labels on the password checker's result—so that it can function as intended—can be exploited to leak other information as well.

In some systems for information flow control, such as the decentralized label model [16], labels can be assigned to these subroutines to prevent this exploitation of declassification. However, the underlying problem still exists: how

This research was supported by DARPA Contract F30602-99-1-0533, monitored by USAF Rome Laboratory. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

to determine when declassification is not being exploited. In this paper, we explore this issue, developing a formal model for identifying what information is actually leaked by programs that contain intentional information leaks, under various assumptions about the abilities possessed by attackers who are attempting to steal confidential data.

We consider two kinds of attackers: First, there are passive attackers who are able to imperfectly observe the state of a computational system as it evolves: some aspects of the system state are observable, and others are not. Given such a system, we can characterize what information passive attackers may be able to learn through observation alone. Second, we consider active attackers who are able not only to observe the behavior of the system but also to modify it. Our formal model is sufficiently general that it can capture both changes to the data used by the program and also changes to the execution of the program. Active attackers are of interest because we wish to build intrusion-tolerant systems. By modeling active attackers formally, we can determine what confidentiality guarantees can be offered in a partially compromised system, and relate the degree of system intrusion to bounds on the information leaked.

The major contribution of this paper is the definition of when a computational system is *robust* with respect to an active attacker. Given a system that contains some intentional flows of confidential information, the system is robust with respect to a class of active attackers if these attackers can learn no more about the confidential information through active attacks than they can through passive observation. Equivalently, a system is robust if the intentional information leaks that it contains cannot be exploited through active attack to learn more than was intended. In accordance with this intuition, we are able to prove that a system containing no information leaks is also robust. By giving examples of robust and nonrobust systems, we demonstrate that robustness is an useful, nontrivial property of computational systems that results from an interaction between the confidentiality and integrity properties of the system.

The rest of the paper is structured as follows. In Section 2, we introduce a formal model for the computational system. We define a simple security property that captures possibilistic information flow within the system, and formally describe a passive attacker. Section 3 illustrates the system model using the password-laundering example. In Section 4, the model of an active attacker is developed; robust declassification is then defined and some of its more interesting properties are shown to hold. In Section 5, we conclude with some discussion about the related work, the benefits of these models, and possible future applications.

2 System Model

A **system** $S = \langle \Sigma, \mapsto \rangle$ consists of a set of states, Σ, and a transition relation $\mapsto \subseteq \Sigma \times \Sigma$. We use $\sigma, \sigma', \sigma_i$, *etc.*, to range over the elements of Σ, and we write $\sigma \mapsto \sigma'$ if the pair $\langle \sigma, \sigma' \rangle$ is in the relation $\mapsto$. We further assume that the relation $\mapsto$ is reflexive: for each $\sigma \in \Sigma$ we have $\sigma \mapsto \sigma$. If $S_1 = \langle \Sigma, \mapsto_1 \rangle$ and $S_2 = \langle \Sigma, \mapsto_2 \rangle$ are systems over the same set of states, we write $S_1 \cup S_2$ for the system $\langle \Sigma, \mapsto_1 \cup \mapsto_2 \rangle$.

A **trace** τ of a system S is any finite sequence $\sigma_0 \mapsto \sigma_1 \mapsto \sigma_2 \mapsto \ldots \mapsto \sigma_{n-1}$ where $n \geq 1$. If τ is a trace, we write $\tau(i)$ for the i^{th} state in the trace. For any state σ in Σ, we use the notation $Trc_\sigma(S)$ for the set of traces starting at σ. The set of all traces of S is denoted by $Trc(S)$:

$$Trc(S) \stackrel{def}{=} \bigcup_{\sigma \in \Sigma} Trc_\sigma(S)$$

The set of traces for the system S is completely determined by its relation $\mapsto$.

Following previous work on state-based models of computation [2], we write $\tau \equiv \tau'$ if the trace τ is stutter-equivalent to τ'. In what follows, we consider traces equal up to stuttering, but we will be explicit about using $\equiv$-equivalence where it makes the exposition clearer. We also extend the use of $\equiv$ to sets of traces. If T and T' are sets of traces, then $T \equiv T'$ whenever they contain the same traces modulo $\equiv$, formally:

$$(\forall \tau \in T.\, \exists \tau' \in T'.\, \tau \equiv \tau') \wedge (\forall \tau' \in T'.\, \exists \tau \in T.\, \tau \equiv \tau')$$

Note that identifying traces up to stutter-equivalence is compatible with our assumption that the relation $\mapsto$ is reflexive because extra "null" transitions $\sigma \mapsto \sigma$ can be eliminated from the trace. If $\tau \in Trc(S)$ and $\tau \equiv \tau'$ then $\tau' \in Trc(S)$.

We also use the notation $\equiv$ to mean stuttering equivalence of sequences from an arbitrary set X; that is if $x, y \in X^*$ we write $x \equiv y$ whenever x and y are stuttering equivalent.

2.1 Views of a System

A **view** of the system S is an equivalence relation $\approx$ on Σ. An equivalence relation corresponds to an ability to distinguish different states of the system S; the more distinctions made by the relation $\approx$, the more information is known about S. Views correspond to security domains or clearance levels because they describe a portion of the state accessible to an observer.

For example, consider the set of states Σ consisting of pairs $\langle h, l \rangle$, where h ranges over some high-security data and l ranges over low-security data. An observer with low-security access (only permitted to see the l component) can see that the states $\langle \texttt{attack at dawn}, 3\rangle$ and $\langle \texttt{do not attack}, 4\rangle$ are different (because $3 \neq 4$), but will be unable to distinguish the states $\langle \texttt{attack at dawn}, 3\rangle$ and $\langle \texttt{do not attack}, 3\rangle$. Thus, with respect to this view ($\approx$):

$$\langle \texttt{attack at dawn}, 3\rangle \approx \langle \texttt{do not attack}, 3\rangle$$
$$\langle \texttt{attack at dawn}, 3\rangle \not\approx \langle \texttt{do not attack}, 4\rangle$$

The universal relation, which we write $\approx_\bot$, relates every state to every other state. It corresponds to having no knowledge of the state of the system. Conversely, the identity relation, given by $\approx_\top$, corresponds to perfect information about the state of the system: any two states can be distinguished. If $\approx$ is a view of S, we write $[\sigma]_\approx$ for the equivalence class of the state σ with respect to $\approx$.

Let $\mathcal{I}(\Sigma)$ be the set of all views of the system. This set forms a complete lattice in which the equivalence relation $\approx_A$ is less than the equivalence relation $\approx_B$ (written $\approx_A \sqsubseteq_\mathcal{I} \approx_B$) whenever $\approx_B \subseteq \approx_A$ as sets. Under this ordering, $\approx_\top$ is the top of the lattice and $\approx_\bot$ is the bottom element. The lattice join operation, $\sqcup_\mathcal{I}$, is given by intersecting the relations, and the meet operation, $\sqcap_\mathcal{I}$, is the transitive closure of the union of the two relations. We write $\approx_A \sqsubset_\mathcal{I} \approx_B$ whenever $\approx_A \sqsubseteq_\mathcal{I} \approx_B$ and $\approx_A \neq \approx_B$.

Higher elements in the lattice represent more information about the state of the system, lower elements represent less information. Two elements may be incomparable, and, in general, $\mathcal{I}(\Sigma)$ is not distributive. See Landauer and Redmond [9] for a more detailed description of this lattice and its relation to unwinding conditions for noninterference.

2.2 The Security Property $\mathcal{SP}(\approx)$

The ordering $\sqsubseteq_\mathcal{I}$ yields a way of comparing how much information is declassified by a system S relative to some initial information about the system. The view relation $\approx$ describes a **passive attacker**, a principal able to observe the system and deduce information about the state. Systems that preserve a view $\approx$ are said to satisfy the security predicate $\mathcal{SP}(\approx)$; intuitively a system satisfies $\mathcal{SP}(\approx)$ if an observer with information given by $\approx$ cannot learn anything by watching the system run. We now formalize this intuition.

Given a trace $\tau \in \mathit{Trc}(S)$, the $\approx$-view of τ, written $\tau/\!\approx$, is simply the sequence of equivalence classes of states in τ:

$$\forall i \in \{0 \ldots \mathit{len}(\tau)\}.\ (\tau/\!\approx)(i) = [\tau(i)]_\approx$$

The intuition behind $\tau/\!\approx$ is that a passive attacker who is able to distinguish states only up to $\approx$ will see the trace τ generated by the system as the sequence of equivalence classes. An **observation** of system S with respect to starting state σ and view $\approx$, written $\mathit{Obs}_\sigma(S,\approx)$ is given by:

$$\mathit{Obs}_\sigma(S,\approx) \stackrel{def}{=} \{\tau/\!\approx \ \mid \tau \in \mathit{Trc}_\sigma(S)\}$$

$\mathit{Obs}_\sigma(S,\approx)$ is the set of all possible sequences of equivalence classes under $\approx$ that might be observed by watching the system whenever it starts in state σ.

The function that maps σ to $\mathit{Obs}_\sigma(S,\approx)$ induces another equivalence relation, written $S[\approx]$, on Σ. This relation can be thought of as the information that might be learned by watching S through the view $\approx$: two states are equivalent only if the possible traces leading from these states are indistinguishable under $\approx$. To say that a system S induces this **observational equivalence** $S[\approx]$ with respect to $\approx$, we define:

$$\begin{gathered}\forall \sigma, \sigma' \in \Sigma.\ \langle \sigma, \sigma'\rangle \in S[\approx] \\ \Leftrightarrow \\ \mathit{Obs}_\sigma(S,\approx) \equiv \mathit{Obs}_{\sigma'}(S,\approx)\end{gathered}$$

We characterize our security predicate, $SP(\approx)$, in terms of the information lattice $\mathcal{I}(\Sigma)$ by simply requiring that the induced observational equivalence corresponds to no more information than was originally known.

Definition 2.1 ($\approx$-Secure System) *A system S is secure with respect to passive attacker $\approx$ if and only if all $\approx$-equivalent states are observationally equivalent. Formally: $S[\approx] \sqsubseteq_\mathcal{I} \approx$. Whenever S satisfies this property we write:*

$$S \models \mathcal{SP}(\approx)$$

This predicate tries to capture the idea that there is no (possibilistic) information flow to an observer with view $\approx$. Any two $\approx$-equivalent states σ and σ' must generate equivalent observations when the system is run. Unfolding the definition of $S \models \mathcal{SP}(\approx)$ yields the equivalent statement:

$$\begin{gathered}S \models \mathcal{SP}(\approx) \\ \Leftrightarrow \\ \forall \sigma, \sigma' \in \Sigma.\ \sigma \approx \sigma' \Rightarrow \mathit{Obs}_\sigma(S,\approx) \equiv \mathit{Obs}_{\sigma'}(S,\approx)\end{gathered}$$

Or, in terms of the traces of the system:

$$\begin{gathered}S \models \mathcal{SP}(\approx) \\ \Leftrightarrow \\ \forall \sigma, \sigma' \in \Sigma. \sigma \approx \sigma' \Rightarrow \forall \tau \in \mathit{Trc}_\sigma(S). \\ \exists \tau' \in \mathit{Trc}_{\sigma'}(S).\ (\tau/\!\approx)\ \equiv\ (\tau'/\!\approx)\end{gathered}$$

We now make a few observations about our security predicate and its interaction with our notion of view.

First, note that an observer can only gain information by watching the system run; information is not lost or destroyed by watching the system.

Proposition 2.1 *For any system S and view $\approx$ it is the case that $\approx \sqsubseteq_\mathcal{I} S[\approx]$.*

One consequence of this monotonicity property is that whenever $S \models \mathcal{SP}(\approx)$ holds, the views of two states coincide with their observations: $\approx \; = \; S[\approx]$.

Next, note that for every system S both $S \models \mathcal{SP}(\approx_\bot)$ and $S \models \mathcal{SP}(\approx_\top)$ hold, but for different reasons. In the former case, no interesting observations can be made about the system and consequently there are no channels through which information could flow. In the latter case, the observer already has complete information about the system state, and so could not learn anything by watching it run.

This last statement may be somewhat surprising, because a $\approx_\top$-observer may learn what nondeterministic choices are made in a particular trace of the system. In our model of information flow all of the "interesting" information is found in the initial state of the system—which is unknown to the passive attacker—and that the actual transitions are "uninteresting."[1] The transition relation $\mapsto$ is already known to the observer.

We have chosen this model because it is simple, fairly general, and it suffices to describe our ideas about robust declassification. By comparison, event and state-event based models [22, 7, 8, 11, 12] take the dual position that only the transitions of the system are of interest (they correspond to augmenting our relation $\mapsto$ to include *labels*, the events observed from outside the system).

To some extent, the difference between state-based systems and labeled-transition systems is only a matter of modeling: each approach can simulate the other with appropriate encodings [5]. For example, the state can keep track of the event (label) of the most recent transition, or even the entire history of the computation. State-based approaches have been advocated in the past [4], although our definition of security differs from traditional noninterference in that purge functions are not used. The combination of taking states modulo $\approx$-equivalence and traces up to stutter-equivalence, yields essentially the same result.

Equivalence relations over states appear in all of these formulations in the guise of unwinding relations [8, 20, 13, 12] and the closely related notion of simulation relations [10]. The difference between unwinding relations and views is that rather than starting with an event system and trying to find a consistent unwinding relation as a means of establishing a security property, we start with a view of the system and determine how the view is altered by information leaks inherent in the system. We intend that the definitions of attack and robust declassification developed in what follows be applicable to richer system models, but we leave to future work such generalization.

[1] In the terminology of Mantel's Assembly Kit [11], all high-security events, *i.e.* those transitions in the set $\mapsto \cap \approx$, are adaptable.

2.3 Multilevel Security, Confidentiality, and Integrity

So far, our definition of information flow security has been motivated from the point of view of protecting the confidentiality of data with respect to one view of the system, $\approx$. For a system with multilevel confidentiality concerns, we take a lattice of security domains $\mathcal{L}_C$ and assume that there is a lattice-homomorphism *lvl* from $\mathcal{L}_C$ into $\mathcal{I}(\Sigma)$. This homomorphism maps a domain $\ell \in \mathcal{L}_C$ to a corresponding view relation $\approx_\ell \in \mathcal{I}(\Sigma)$. Note that because we require the map $lvl : \ell \longmapsto \approx_\ell$ to be a homomorphism, $\mathcal{L}_C$ must contain top and bottom security clearances that are sent to the "omniscient" and "null" views of the system, respectively. Write $lvl(\mathcal{L}_C)$ for the image of $\mathcal{L}_C$ under *lvl*.

The definition of $\mathcal{SP}(-)$ can also be used to indicate when computation depends on low-integrity data. Thus, we may specify integrity constraints about a system by simply giving another lattice of integrity levels, $\mathcal{L}_I$, and corresponding equivalence relations, $\leftrightarrow_\iota$ for $\iota \in \mathcal{L}_I$. Although integrity relations are treated by the formalism in the same way as the confidentiality relations, their meaning is different. Confidentiality equivalence says that two states are equivalent from the observer's point of view, whereas integrity equivalence says that two states are equivalent from the point of view of a user who relies on the state. Two states are equivalent if the differences between them are unimportant. If the system satisfies the security property $\mathcal{SP}(\leftrightarrow_\iota)$, the "important" aspects of its behavior are unaffected by "unimportant" differences between the states. Because confidentiality and integrity are expressed in terms of observational equivalence, the same security property enforces both.

As an example of how the lattice structure of $\mathcal{I}(\Sigma)$ can be used to reason about a multilevel security system, consider the problem of trying to determine which principal's information has been leaked by the system. We assume that the declassifications in the system occur under some principal's authority. Clearly, someone with top-level clearance (someone who knows everything about the system) could have leaked the information. A more interesting question to ask is: What is the *lowest* security domain that could have authorized the declassification?

It is possible to assign responsibility for the declassification based on the security clearances in $\mathcal{L}_C$. We construct the set of security domains whose available information about Σ, together with the information represented by $\approx$, can explain the observed behavior in $Obs(S, \approx)$. This is the following set:

$$D = \{\approx_\ell \mid S[\approx] \sqsubseteq_{\mathcal{I}} (\approx_\ell \sqcup_{\mathcal{I}} \approx)\}$$

The join $(\approx_\ell \sqcup_{\mathcal{I}} \approx)$ represents the sum of information available to security domain ℓ and the information known to the viewer of the system. When the join is higher in $\mathcal{I}(\Sigma)$ than $S[\approx]$, the principal whose view is $\approx_\ell$ has access to enough information to cause the apparent declassification.

If the lattice $\mathcal{L}_C$ is distributive, we can pinpoint the least security domain that could have been responsible for the declassification by simply taking the greatest lower bound on the members of D, namely $\approx_D = glb\{\approx_\ell \in D\}$. By distributivity, $\approx_D$ is guaranteed to be an element of D itself. It is the smallest level of information that, together with $\approx$ is sufficient to explain the $\approx$-view of the system. If $\mathcal{L}_C$ is not distributive, any one of the $\sqsubseteq_\mathcal{I}$-minimal elements of D could have declassified information sufficient to cause the evident information flow.

3 An Example

To illustrate the model, let us consider the example of the attack discussed in the introduction, in which a password system is used to launder confidential information.

To model that scenario, we assume that the state of the system consists of a 5-tuple $\langle t, h, p, q, r\rangle$. The component $t \in \{0, 1\}$ is the time—0 indicates that the password checker has not run yet, and 1 indicates that the password checker has completed. In more realistic examples, this simple notion of time could be replaced with the program counter of a computer, but this suffices for our discussion. The component h is a bit representing some high security data that should not be leaked to external users of the system. For simplicity, we assume that there is only one user password in the database, and its value is a bit given by the component p. The external user submits a query, q, which will be compared against p by the password checker. If p and q match, the password checker toggles the value of the boolean r, which stores the result of the query. If p and q are not the same, the password checker leaves the value of r unchanged.

The execution of the password checker can be given by the transition relation below:

$$\begin{array}{lcll}
\langle t,h,p,q,r\rangle & \mapsto & \langle t,h,p,q,r\rangle & \\
\langle 0,h,p,p,0\rangle & \mapsto & \langle 1,h,p,p,1\rangle & (p = q, \text{toggle } r) \\
\langle 0,h,p,p,1\rangle & \mapsto & \langle 1,h,p,p,0\rangle & (p = q, \text{toggle } r) \\
\langle 0,h,p,q,0\rangle & \mapsto & \langle 1,h,p,q,0\rangle & (p \neq q, \text{leave } r) \\
\langle 0,h,p,q,1\rangle & \mapsto & \langle 1,h,p,q,1\rangle & (p \neq q, \text{leave } r)
\end{array}$$

An external user of the system is only able to directly see the value of the query submitted to the password checker, the result that the password checker returns, and that the password checker has completed its computation (time has passed). This leads to an equivalence relation, $\approx$, given by:

$$\begin{gathered}
\langle t,h,p,q,r\rangle \approx \langle t',h',p',q',r'\rangle \\
\Leftrightarrow \\
(t = t') \wedge (q = q') \wedge (r = r')
\end{gathered}$$

Let S be the password checking system just described. The external user of the system can learn some information about the password p, namely whether it matches the query they submitted, by watching the system run. Thus the system S induces an observational equivalence $S[\approx]$ which is strictly higher in the information lattice $\mathcal{I}(\Sigma)$:

$$\begin{gathered}
\langle t,h,p,q,r\rangle \; S[\approx] \; \langle t',h',p',q',r'\rangle \\
\Leftrightarrow \\
(t = t') \wedge (q = q') \wedge (r = r') \wedge (t = 0 \Rightarrow (p = p'))
\end{gathered}$$

Now suppose that the owner of the password alters p based on the value of the high-security data h before the password checker is run. Because we've assumed that both the high-security data and the password are represented as bits, the simplest variant of such an attack is to copy the high security data into the password. This attack corresponds to adding some transitions[2]to the system above:

$$\langle 0,h,p,q,r\rangle \quad \mapsto_A \quad \langle 0,h,h,q,r\rangle$$

Now, as expected, the observational equivalence induced on the attacked system S' is not the same as the one induced by the original system S. We have:

$$\begin{gathered}
\langle t,h,p,q,r\rangle \; S'[\approx] \; \langle t',h',p',q',r'\rangle \\
\Leftrightarrow \\
(t = t') \wedge (q = q') \wedge (r = r') \wedge \\
(t = 0 \Rightarrow p = p' \vee h = h' \vee p = h' \vee h = p')
\end{gathered}$$

Stating the equivalence relations in this way, it is easy to see that the external observer can possibly learn the value of h by watching the system S' run. The external observer can distinguish any two states based on the run of the system σ and σ' just when σ is not related to σ' via $S'[\approx]$. Negating the right hand side of the equivalence above yields:

$$\begin{gathered}
(t \neq t') \vee (q \neq q') \vee (r \neq r') \vee \\
(t = 0 \wedge p \neq p' \wedge h \neq h' \wedge p \neq h' \wedge h \neq p')
\end{gathered}$$

This says that the external observer can see when time has passed, when q changes, when r changes, or when $t = 0$ and $p = h$, $p' = h'$ and $p \neq p$. Some information about h has been leaked.

As this example shows, the equivalence relations induced by a system may be quite complex.[3] Note that the attack just described doesn't leak all of the information about h because when $h = p$, copying it into the password doesn't lead to any new behavior in the system (with respect to observations through view $\approx$). A more savvy attacker might

[2]We use the subscript A to indicate that these are transitions introduced by an attacker.

[3]In this setting, because there are only two possible values for p, h, *etc.*, more information is leaked than when more values are possible. The reason is that $p \neq q$ and $h \neq q$ implies that $p = h$, which, in general is not true. We have made use of this kind of reasoning to simplify the description of the equivalence relations.

also toggle r whenever he copied h into p, thus indicating that p does in fact contain h. This smarter attack adds these transitions:

$$\begin{array}{lll} \langle 0,h,p,q,0\rangle & \mapsto_A & \langle 0,h,h,q,1\rangle \\ \langle 0,h,p,q,1\rangle & \mapsto_A & \langle 0,h,h,q,0\rangle \end{array}$$

The equivalence relation induced by S' now is given by:

$$\begin{array}{c} \langle t,h,p,q,r\rangle\ S[\approx]\ \langle t',h',p',q',r'\rangle \\ \Leftrightarrow \\ (t=t')\wedge(q=q')\wedge(r=r')\wedge \\ (t=0\Rightarrow(p=p')\vee(h=h')) \end{array}$$

Reading off the negation, we see that an attacker can distinguish states whenever $t = 0$ and $h \neq h'$ and $p \neq p'$, that is, it is possible for the observer to learn the complete information about the initial state of the system.

Clearly this simple password system is not secure with respect to an attacker who has the ability to both alter one piece of high-security data (the password) based on another (h) and communicate that this change has been done (toggle r). On the other hand, if the attacker may only toggle r no additional information is leaked. In what follows, we develop a methodology for characterizing systems in terms of their robustness against different kinds of attacks.

4 Robust Declassification

This section examines declassification in a system, specifies a class of attackers that is interesting from the information-flow perspective, and defines robustness for systems with respect to this class of attackers.

Having defined information flow in terms of the lattice of information, $\mathcal{I}(\Sigma)$, we are now in a position to consider declassification of data. The starting point for our notion of declassification is that any system that leaks information—any system that does not satisfy $\mathcal{SP}(\approx)$—can be thought of as containing declassifications. A passive attacker may be able to learn some information by observing the system but, by assumption, that information leakage is allowed by the security policy.

We first define active attackers: principals that may alter the system in an attempt to learn secret information.

4.1 Active Attacks

What constitutes a valid attack on the system? We would like to model ways that an attack can affect the confidentiality properties of the system. Typical assumptions about the attacker in an information-flow setting are that the attacker can make (perhaps limited) observations of the system and draw inference from those observations—passive attacks. Another common means of specifying attackers is to require that they are programs running concurrently with the system (for example, in process calculi such as CSP [21] or the Spi calculus [1]) or perhaps more limited processes (for example, restricted to polynomial-time probabilistic computation).

Our concern is that an attacker will be able to exploit the information learned via declassification, or simply the fact that a declassification occurs, to cause a system to divulge more information than permitted by the security policy.

In our model attackers are able to change the behavior of the executing system. For example, in a system that is a single-computer program, the attacker might overwrite memory locations or registers of the machine. As in Section 3, we model these changes as an *attack transition relation* $\mapsto_A$ that performs the change to the state. The power of the attacker can also be captured simply by the attacker's view $\approx_A$, because any attack must be secure with respect to $\approx_A$:

Definition 4.1 ($\approx_A$-Attack)
An ***$\approx_A$-attack*** *is a system $A = \langle\Sigma, \mapsto_A\rangle$ such that $A \models \mathcal{SP}(\approx_A)$.*

Note that the requirement that $A \models \mathcal{SP}(\approx_A)$ is essentially the fair environment assumption: The attacker must not know the secret already (or be able to learn it from means other than the system in question). We use $\mathcal{A}(\approx_A)$ to mean the set of all attacks with respect to the view $\approx_A$.

Given an attack A and a system S, both specified in terms of the same set of states Σ, the attack on S by A is just the union of the systems: $S \cup A$. This means of composition is justified by our possibilistic interpretation of information flow: the attacker will learn more information if it is possible for a trace in the new system to distinguish one state from another.

4.2 Robust Systems

Given a system S and an attacker's view of the system $\approx_A$, we would like a way to characterize classes of attacks drawn from the set $\mathcal{A}(\approx_A)$. The first such characterization, on which all our other classifications are based, is robustness:

Definition 4.2 (Robust Declassification)
A system $S = \langle\Sigma, \mapsto\rangle$ is ***robust*** *with respect to the class $\mathcal{B} \subseteq \mathcal{A}(\approx_A)$ of attacks if for all attacks $A = \langle\Sigma, \mapsto_A\rangle$ in $\mathcal{B}$, it is the case that $(S \cup A)[\approx_A] \sqsubseteq_{\mathcal{I}} S[\approx_A]$. To indicate that S is robust in this way, we write:*

$$S \models \mathcal{R}(\mathcal{B})$$

This says formally that observing the attacked system $S \cup A$ reveals no more information than watching the original system S.

By identifying interesting subsets of attacks from which the system is immune, we can better understand its information flow properties. Conversely, if we can be reasonably sure that the the only attacks on the system are ones for which the system is robust, we believe the system is secure. As with any formalization of attacks, we aren't guaranteed anything about attacks that fall outside our model. Also, we can never hope to prevent all attacks against every system. We see our results as tools for mapping the landscape of attacks, information flow systems, and their interaction with declassification.

The first interesting lesson we learn from this formalization is that all systems that are secure with respect to $\approx_A$ are robust to all attacks from that view. Intuitively, whenever running the system reveals no information to the attacker, there is no way for an attacker to boost their information of the system by modifying its behavior.

Theorem 4.1
If $S \models \mathcal{SP}(\approx_A)$ then $S \models \mathcal{R}(\mathcal{A}(\approx_A))$.

Proof: Let A be an attack in $\mathcal{A}(\approx_A)$. Then, by definition of an attacker, we have $A \models \mathcal{SP}(\approx_A)$. From Proposition 2.1 and the definition of $\mathcal{SP}(\approx_A)$ it follows that $S[\approx_A] = \approx_A$, and hence $S \models \mathcal{SP}(S[\approx_A])$ and also $A \models \mathcal{SP}(S[\approx_A])$. From Lemma A.1 (its proof is in the Appendix) it follows that $(S \cup A) \models \mathcal{SP}(S[\approx_A])$, from which we obtain $(S \cup A)[\approx_A] \sqsubseteq_{\mathcal{I}} S[\approx_A]$ as required. $\square$

This result justifies to some extent the use of $\mathcal{SP}(\approx_A)$ as a strong notion of security—not only does it guarantee information flow properties of the system S with respect to $\approx_A$, it also says that S is not susceptible to any attacks by such an observer either.

Clearly there are other sets of attackers for which any system is robust. For example, let $\mathcal{B}$ be the set of attacks such that attack transition relation $\mapsto_A$ is contained in the view $S[\approx_A]$. Then any system S (even one that does not satisfy $\mathcal{SP}(\approx_A)$) is robust with respect to $\mathcal{B}$. The proof is a simple inductive argument. However, this is a particularly limited class of attackers that are unable to alter any part of the state they are able to observe, and so it is not particularly useful.

In order to formulate a more useful class of attackers for which the system is robust, we describe the relation between information learned by certain attackers and the security properties of a system that is not secure with respect to $\approx_A$. We first construct the iterated observation of a system, $S^n[\approx_A]$, which can be thought of as the least view refining $\approx_A$ for which S is secure. The definition of iterated observation is the following:

$$\begin{array}{rcl} S^0[\approx_A] & \stackrel{def}{=} & \approx_A \\ S^{n+1}[\approx_A] & \stackrel{def}{=} & S[S^n[\approx_A]] \\ S^\omega[\approx_A] & \stackrel{def}{=} & \bigsqcup_{n\in\omega} S^n[\approx_A] \end{array}$$

The least fixed-point specified by the last definition exists because $\mathcal{I}(\Sigma)$ is a complete lattice and Proposition 2.1 implies that the iterated observation forms the ordered chain:

$$\approx_A \sqsubseteq_{\mathcal{I}} S[\approx_A] \sqsubseteq_{\mathcal{I}} S^2[\approx_A] \sqsubseteq_{\mathcal{I}} S^3[\approx_A] \sqsubseteq_{\mathcal{I}} \dots$$

As we desired, any system is secure with respect to its ω-iterated view:

Proposition 4.1
Any system, S, and view $\approx_A$ satisfy $S \models \mathcal{SP}(S^\omega[\approx_A])$.

The following proposition states that the observational equivalence generated by a system operates monotonically on equivalence relations.

Proposition 4.2
If $\approx_A \sqsubseteq_{\mathcal{I}} \approx_B$ then for any system S, $S[\approx_A] \sqsubseteq_{\mathcal{I}} S[\approx_B]$.

Finally, we give a bound on information leaked.[4]

Theorem 4.2
Let S be a system and let $\approx_A$ be a view in $\mathcal{I}(\Sigma)$. Let A be an $\approx_A$-attack such that $A \models \mathcal{SP}(S^\omega[\approx_A])$. Then

$$(S \cup A)[\approx_A] \sqsubseteq_{\mathcal{I}} S^\omega[\approx_A].$$

Proof: From Proposition 4.1 we have $S \models \mathcal{SP}(S^\omega[\approx_A])$, and, by using Lemma A.1, it follows that for any $A \in \mathcal{B}$ that $(S \cup A) \models \mathcal{SP}(S^\omega[\approx_A])$. Consequently,

$$(S \cup A)[S^\omega[\approx_A]] \sqsubseteq_{\mathcal{I}} S^\omega[\approx_A].$$

Propositions 2.1 and 4.2 show that

$$(S \cup A)[\approx_A] \sqsubseteq_{\mathcal{I}} (S \cup A)[S^\omega[\approx_A]]$$

and we obtain the required result by transitivity of $\sqsubseteq_{\mathcal{I}}$.
$\square$

How can we use this theorem to help understand the behavior of a system under attack? As we described in Section 2.3, the security property can capture both confidentiality and integrity aspects of a system. The equivalence relation $S[\approx_A]$ can be thought of as describing either the maximal amount of information that can be learned by watching the system, or, perhaps more intuitively, as an integrity

[4] In the proceedings version of this paper, Theorem 4.2 was claimed to be a generalization of Theorem 4.1, and was incorrect as stated. The version presented here is weaker in that it does not define a class of attacks against which S is robust unless $S^\omega[\approx_A] = S[\approx_A]$.

property of the system. Two states related by $S[\approx_A]$ are, in some sense, unimportant to the behavior of S as observed by the attacker. Only attacks that force two such "unimportant" states to be "important"—by providing transitions that distinguish them—can cause additional information to be leaked by the system.

We can use Theorem 4.2 to characterize attacks on the password checking facility described in Section 3. It is easy to show that, for this particular system, $S^\omega[\approx_A] = S[\approx_A]$. It follows that any attack that satisfies $A \models \mathcal{SP}(S[\approx_A])$ cannot cause the system to leak information. The attacker that simply toggles r (at time 0) falls into this class, as does the one that changes q to a string not equal to p. The attack that copies h into p, on the other hand, sends the states $\langle 0, h_1, p, h_1, r\rangle$ and $\langle 0, h_2, p, h_1, r\rangle$ to the states $\langle 0, h_1, h_1, h_1, r\rangle$ and $\langle 0, h_2, h_2, h_1, r\rangle$, respectively. The first pair of states are $S^\omega[\approx_A]$-equivalent, whereas the second two are not. While Theorem 4.2 does not guarantee that such an attack will cause more information to be leaked, it does say that the attack lies outside those that the system is known to be robust against.

The bound on information flow given by Theorem 4.2 is not tight; it is possible to construct systems and attacks for which the estimated information flow given by $S^\omega[\approx_A]$ is strictly more than the actual information learned by $(S \cup A)[\approx_A]$. However, the $S^\omega[\approx_A]$ usefully bounds information flow for a variety of systems. Determining more precise bounds on what attackers can learn is a goal of future work.

5 Discussion and Conclusions

There has been a fair amount of prior work on controlled declassification or downgrading mechanisms, or the formal characterization of systems incorporating them. The simplest and most standard approach to declassification is to restrict its uses to those performed by a trusted subject. This approach does not address the question of whether an information channel is created. Many systems have incorporated a more limited form of declassification. Ferrari et. al [6] augment information flow controls in an object-oriented system with a form of dynamically-checked declassification called *waivers*. Myers and Liskov [15] define a form of *selective declassification* that can be checked at compile-time, based on the authority of the declassifying process. However, these efforts provide only limited characterization of the safety of the declassification process.

Intransitive noninterference policies [19, 17, 18] generalize noninterference to describe systems that contain restricted downgrading mechanisms. The work by Bevier *et al.* on *controlled interference* [3] is most similar to this work in allowing the specification of policies for information released to a set of *agents*. Their notion of agent largely agrees with the notion of a passive attacker defined here. None of this prior work addresses the issue of an active attacker. However, the results in this paper should also be applicable to specifying intransitive noninterference policies.

Our notion of attack is clearly connected with *refinement*. In particular, the original system S refines (has less nondeterminism than) the attacked system $(S \cup A)$. S is robust to the attack A if the refinement preserves the equivalences given by $S[\approx_A]$. Another important direction for future work is to consider attacks that can remove transitions from S, effectively causing some computation paths to become impossible.

This paper makes a number of contributions to the problem of systems containing intentional information leaks that presumably arise from controlled declassification. Using a purely state-based system model and definition of a noninterference-like information flow property, we precisely characterize the information that is released to an arbitrary observer (passive attacker) of the system, described as an equivalence relation $\approx_A$ over the states of the system. The possible executions of the system, defined by its nondeterministic transition relation, generate a refinement of the view equivalence relation, $S[\approx_A]$. The difference between these two equivalence relations captures the information released to an observer. The lattice of information (whose elements are views of the system) is a powerful tool for understanding the information flow behavior of the system.

The major contributions of this paper lie in the characterization of information flow in systems suffering some intrusion by an active attacker that is able to modify the state of the executing system. Making the reasonable assumption that the attacker cannot construct an attack that depends on the exploitation of information that it cannot observe directly, we obtain the expected property that an attacker cannot violate confidentiality if the system obeys the information flow security property. Importantly, for systems that contain intentional information leaks (do not obey the security property), we give a recipe for bounding the ability of a class of attackers to obtain information. From a description of the direct powers of observation of an attacker ($\approx_A$), the relation $S[\approx_A]$ is obtained, defining both a level of confidentiality that can be maintained, and a degree of integrity that must not be violated by an active attacker in order to preserve that confidentiality.

We expect this model to provide new tools for the characterization of information flow properties in the presence of intentional information leaks and system intrusion. Because the model is state-based, it seems particularly applicable to language-based approaches to information flow control [14]. The connections to models of intransitive noninterference also deserve further exploration.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.

[2] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 14(4):1–60, October 1992.

[3] W. R. Bevier, R. M. Cohen, and W. D. Young. Connection policies and controlled interference. In *Proc. of the 8th IEEE Computer Security Foundations Workshop*, pages 167–176, 1995.

[4] W. R. Bevier and W. D. Young. A state-based approach to non-interference. In *Proc. of the 7th IEEE Computer Security Foundations Workshop*, pages 11–21, 1994.

[5] R. De Nicola and F. Vaandrager. Three logics for branching simulation. *Journal of the Association of Computing Machinery*, 42(2):458–487, 1995.

[6] E. Ferrari, P. Samarati, E. Bertino, and S. Jajodia. Providing flexibility in information flow control for object-oriented systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 130–140, Oakland, CA, USA, May 1997.

[7] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, Apr. 1982.

[8] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symposium on Security and Privacy*, pages 75–86, Apr. 1984.

[9] J. Landauer and T. Redmond. A lattice of information. In *Proc. of the 6th IEEE Computer Security Foundations Workshop*, pages 65–70. IEEE Computer Society Press, June 1993.

[10] N. Lynch and F. Vaandrager. Forward and backward simulations – Part I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995. Also, Technical Memo MIT/LCS/TM-486.b (with minor revisions), Laboratory for Computer Science, Massachusetts Institute of Technology.

[11] H. Mantel. Possibilistic definitions of security: An assembly kit. In *Proc. of the 13th IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, United Kingdom, 2000.

[12] H. Mantel. Unwinding possibilistic security properties. In *ESORICS 2000*, volume 1895 of *Lecture Notes in Computer Science*, pages 238–254. Springer-Verlag, 2000.

[13] J. K. Millen. Unwinding forward correctibility. In *Proc. of the 7th IEEE Computer Security Foundations Workshop*, pages 2–10, 1994.

[14] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. 26th ACM Symp. on Principles of Programming Languages (POPL)*, San Antonio, TX, USA, Jan. 1999.

[15] A. C. Myers and B. Liskov. Complete, safe information flow with decentralized labels. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1998.

[16] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4), 2000.

[17] S. Pinsky. Absorbing covers and intransitive noninterference. In *Proc. IEEE Symposium on Security and Privacy*, 1995.

[18] A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proc. of the 12th IEEE Computer Security Foundations Workshop*, 1999.

[19] J. Rushby. Noninterference, transitivity and channel-control security policies. Technical report, SRI, 1992.

[20] P. Ryan. A CSP formulation of non-interference and unwinding. *Cipher*, pages 19–30, 1991.

[21] S. Schneider. Security properties and CSP. In *Proc. IEEE Symposium on Security and Privacy*, 1996.

[22] A. Zakinthinos and E. S. Lee. A general theory of security properties and secure composition. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, 1997.

A Proofs

Lemma A.1

Let $S_1 = \langle \Sigma, \mapsto_1 \rangle$ and $S_2 = \langle \Sigma, \mapsto_2 \rangle$ be systems and suppose $\approx$ is an equivalence relation in $\mathcal{I}(\Sigma)$ then:

$$S_1 \models \mathcal{SP}(\approx) \wedge S_2 \models \mathcal{SP}(\approx) \quad \Rightarrow \quad S_1 \cup S_2 \models \mathcal{SP}(\approx)$$

Proof: Let σ_1 and σ_1' be two states such that $\sigma_1 \approx \sigma_1'$. Let τ_1 be a trace in $Trc_{\sigma_1}(S_1 \cup S_2)$. We must show that there exists a trace τ_1' in $Trc_{\sigma_1'}(S_1 \cup S_2)$ such that

$$(\tau_1/\approx) \;\equiv\; (\tau_1'/\approx).$$

We proceed by induction on the length of τ_1. In the case that τ_1 has length 1, $\tau_1/\approx \;\equiv\; [\sigma_1]_\approx$ and we may choose $\tau_1' = \sigma_1'$, which is equivalent to τ_1 modulo $\approx$ because $\sigma_1 \approx \sigma_1'$. If τ_1 starts with the transition $\sigma_1 \mapsto_x \sigma_2 \ldots$, then $\mapsto_x$ is either of the form $\mapsto_1$ or $\mapsto_2$. In either case, because S_1 and S_2 satisfy $\mathcal{SP}(\approx)$, we may construct a $\equiv$-equivalent trace $\sigma_1' \mapsto_x \sigma_2' \mapsto_x \ldots \mapsto_x \sigma_n'$ consisting of transitions from the system S_x and such that $\sigma_n' \approx \sigma_2$. We inductively construct the rest of the list starting from the states $\sigma_2 \approx \sigma_n'$: Let τ_2 be the suffix of τ_1 starting at σ_2. Then there exists a $\tau_2' \in Trc_{\sigma_n'}(\Sigma_1 \cup \Sigma_1)$ such that $\tau_2 \equiv \tau_2'$. Because stuttering equivalence is preserved by trace concatenation $\tau_1 = \sigma_1 \mapsto_x \tau_2 \equiv \sigma_1' \mapsto_x \ldots \mapsto_x \tau_2'$ as required.

$\square$