Decentralized Robustness

Stephen Chong Andrew C. Myers
Department of Computer Science,
Cornell University
E-mail: {schong,andru}@cs.cornell.edu

Abstract

Robustness links confidentiality and integrity properties of a computing system and has been identified as a useful property for characterizing and enforcing security. Previous characterizations of robustness have been with respect to a single idealized attacker; this paper shows how to define robustness for systems with mutual distrust. Further, we demonstrate that the decentralized label model (DLM) can be extended to support fine-grained reasoning about robustness in such systems. The DLM is a natural choice for capturing robustness requirements because decentralized labels are explicitly expressed in terms of principals that can be used to characterize the power of attackers across both the confidentiality and integrity axes. New rules are proposed for statically checking robustness and qualified robustness using an extended DLM; the resulting type system is shown to soundly enforce robustness. Finally, sound approximations are developed for checking programs with bounded but unknown label parameters, which is useful for security-typed languages. In sum, the paper shows how to use robustness to gain assurance about secure information flow and information release in systems with complex security requirements.

1 Introduction

To describe information security in realistic applications, it is necessary to go beyond rigid security properties such as noninterference [7]. Real applications release information by intention, so validating their security requires some assurance that information release is intentional and that its implications are understood by the programmer and the user. The problem of expressing and enforcing policies for information release is crucial for putting language-based information security into practice [15, 16].

One useful security property that has been identified for this purpose is *robustness*, which intuitively says that an attacker should not be able to affect the security of information flow. Robustness has been applied to the control of information release in previous work on *robust declassification* [19, 18, 12]. Robust declassification ensures that an entity who can influence the behavior of a system (for example, by providing or modifying data or code), is unable to observe more information than an entity who cannot influence system behavior. Robustness has also been applied to the erasure of information [3].

Previous work defines robustness with respect to a single attacker, but real computing systems serve the needs of multiple principals who in general do not trust each other. This paper considers the problem of enforcing robustness in such decentralized systems, where security assurance requires convincing every principal that information flow is robust. This is challenging because the identity and power of the attacker depend on whose viewpoint is considered.

Mutually distrusting principals need the tools to express and enforce information security requirements. The decentralized label model (DLM) [11] provides the required expressive power because it expresses information security policies in terms of principals, and because individual principals can express and retain ownership over information security policies regarding confidentiality and integrity.

Rules intended for checking robust declassification have earlier been proposed for a simplified version of the DLM [18], but those rules do not enforce robustness as defined here. Relative to that work, this paper makes two important contributions. First, the rules defined here are shown to enforce a semantic security property of robustness. Second, this paper extends the DLM with support for fine-grained integrity policies. The resulting richer policy language enables a more precise characterization of trust, integrity, and the power of the attacker.

The paper proceeds as follows. Sections 2 and 3 review robustness and the decentralized label model, includ-

This work was supported by the Department of the Navy, Office of Naval Research, under ONR Grant N00014-01-1-0968. This work was also supported by the National Science Foundation under Grant Nos. 0133302 and 0430161, and by an Alfred P. Sloan Research Fellowship. Any opinions, findings, conclusions, or recommendations here are those of the authors and do not necessarily reflect the views of any funding agencies.

ing some minor extensions. Section 4 shows that robustness can be naturally expressed in the DLM framework, because attackers can be treated as principals in that framework. Constraints for checking robustness in that framework are derived. Section 5 shows that these constraints can be added to a security type system for a simple imperative language, with the result that any well-typed program enforces robustness against any attacker. In Section 6, qualified robustness [12] is shown to be enforced by constraining the uses of endorsement to boost integrity. The described robustness rules have been implemented in the Jif programming language [10, 13]; Section 7 discusses some additional issues that arise there, such as how to correctly handle labels unknown at compile time. Section 8 discusses further related work, and Section 9 concludes.

2 Robustness

We define robustness with respect to an abstract notion of system; a more concrete instantiation is given in Section 5. A system has *configurations*, and an execution of a system is a sequence of configurations, called a *trace*. A configuration may consist of memory, code, data, or other elements; the specifics of a configuration are system-specific. Every element of a configuration is associated with a security level, drawn from a set of security levels $\mathcal{L}$. For example, the security level associated with a memory location in a configuration could represent the security level of information that is allowed to be stored in that memory location.

Each security level $\ell \in \mathcal{L}$ is a pair of a confidentiality policy and an integrity policy; we write $C(\ell)$ for the confidentiality policy of ℓ, and $I(\ell)$ for the integrity policy. We assume there is a binary relation $\sqsubseteq_{\mathcal{L}}$ on $\mathcal{L}$ that indicates the relative restrictiveness of security levels: for security levels ℓ and ℓ', if $\ell \sqsubseteq_{\mathcal{L}} \ell'$, then ℓ requires confidentiality lower than (or equal to) that of ℓ', and higher (or equal) integrity.

The definition of robustness assumes that there is an *attacker*, an entity that is able to modify the behavior of the system in limited ways. An attacker is characterized by its *power*, its ability to modify system behavior. The power of an attacker A is a pair of security levels: $\langle R_A, W_A \rangle$. Security level R_A is an upper bound on the security level of elements in a configuration that A can read, and W_A is a lower bound on the security level of configuration elements that A can influence.

An *attack* by attacker A is a modification to some or all elements of a configuration with a security level bounded below by W_A. That is, an attack by A can modify any element in a configuration whose associated security level indicates that A is able to influence it. An attack a applied to a configuration s is denoted $s[a]$. The form of attacks is system-specific, but might include modifying the contents of a memory location, or replacing code in the system.

After attacking a system, the attacker observes the subsequent execution of the system. The observational ability of the attacker is system-specific, but is characterized by the security level R_A. We assume there is a relation over traces that indicates when two traces are indistinguishable to an attacker. We lift the indistinguishability relation over traces to two indistinguishability relations over configurations: *weak indistinguishability* and *strong indistinguishability*.

Let $Tr(s)$ denote the set of traces that start from the configuration s. Two configurations s and s' are weakly indistinguishable, written $s \approx_A s'$, if for every terminating trace in $Tr(s)$, there is a terminating trace in $Tr(s')$ such that the two traces are indistinguishable to the attacker A. Configurations s and s' are strongly indistinguishable, written $s \mathrel{\approx\!\!\!=}_A s'$, if s is weakly indistinguishable from s' and all traces in both $Tr(s)$ and $Tr(s')$ are terminating.

Having defined systems, attackers, and attacks, we can now present the definition of robustness.

Definition 1 (Robustness) *A system has* robustness *with respect to attacks by attacker A with power $\langle R_A, W_A \rangle$ if for all configurations s and s', and all attacks a and a' by attacker A, if $s[a] \mathrel{\approx\!\!\!=}_A s'[a]$, then $s[a'] \approx_A s'[a']$.*

Robustness captures the idea that the observations of an attacker should be independent of what attacks the attacker can make. In particular, an attacker should be unable to force the system to declassify information, or to influence what information is declassified by the system. (The latter is known as a *laundering attack*.)

By requiring strong indistinguishability in the premise of the condition, the robustness condition ignores inept attacks that cause a system to diverge and thus to present the attacker with fewer observations. See [12] for more discussion of this technical issue.

3 Decentralized label model

The robustness security condition ensures the inability of a single entity, the attacker, to influence the declassification of information. However, in a system containing several mutually distrusting entities, there is no single distinguished attacker. Indeed, from the perspective of any one entity, every other entity may be a potential attacker.

The *decentralized label model* (DLM) [11] provides a means to consider any entity as a potential attacker. It is a framework in which mutually distrusting principals can express information-flow security policies for confidentiality and integrity. A *principal* is any entity with security concerns, such as a user, a process, or a user group. A principal may delegate its authority to other principals: if principal p delegates its authority to principal q, then q is said to *act for* p, written $q \succeq p$. The *acts-for* relation is reflexive and transitive; it is similar to the *speaks-for* relation [9], and can be used to encode groups and roles.

Principals express their security concerns with *labels*. A label is a pair of a confidentiality policy and an integrity policy. Labels are associated with information, and a system that enforces labels ensures that the policies of a label are enforced on the appropriate information. Confidentiality policies are formed from conjunctions and disjunctions of *reader policies*, and integrity policies are formed from conjunctions and disjunctions of *writer policies*. Each reader policy and writer policy has an owning principal; a policy owned by a principal p is a statement of p's beliefs or requirements about the security of information.

3.1 Confidentiality policies

A *reader policy* allows the owner of the policy to specify which principals the owner permits to read a given piece of information. A reader policy is written $o \rightarrow r_1, \ldots, r_n$, where the principal o is the owner of the policy, and the principals $r_1, \ldots, r_n$ are the specified readers. A reader policy $o \rightarrow r_1, \ldots, r_n$ says that o permits a principal q to read information only if q can act for the owner of the policy or for any of the specified readers r_i. As a formal semantics for reader policies, we define the function $\mathsf{readers}(p, c)$ to be the set of principals that principal p believes should be allowed to read information according to reader policy c:

$$\mathsf{readers}(p, o \rightarrow r_1, \ldots, r_n) \triangleq \{q \mid \text{if } o \succeq p \text{ then } (q \succeq o \text{ or } \exists i \in 1..n.\ q \succeq r_i)\}$$

A principal p believes that a reader policy c should restrict the readers of information only if the owner of the policy can act for p. The parameterization on p is important in the presence of mutual distrust, because it allows the significance of the policy to be expressed for every principal independently. If principal o owns a policy that restricts the readers of information, it does not necessarily mean that another principal p also believes those restrictions should apply. Thus, if o does not act for p, then $\mathsf{readers}(p, o \rightarrow r_1, \ldots, r_n)$ is the set of all principals; in other words, p does not credit the policy with any significance. While this semantics is expressed differently, it is consistent with the original DLM semantics [11].

Conjunction and disjunction. Greater expressiveness can be achieved by taking conjunctions and disjunctions of reader policies. We define *confidentiality policies* to be the smallest set containing all reader policies and closed under the binary operators $\sqcup$ and $\sqcap$. That is, if c and d are confidentiality policies, then both $c \sqcap d$ and $c \sqcup d$ are too.

The operator $\sqcup$ is conjunction for confidentiality policies: $c \sqcup d$ is the policy that enforces both c and d. The policy $c \sqcup d$ permits a principal to read information only if both c and d allow it. Thus, $c \sqcup d$ is at least as restrictive as both c and d. The operator $\sqcap$ is disjunction for confidentiality policies: $c \sqcap d$ allows a principal to read information if either c or d allows it. Thus, $c \sqcap d$ is no more restrictive than either c or d.

We extend $\mathsf{readers}(p, c)$ for confidentiality policies. Since $c \sqcup d$ enforces both c and d, the reader sets for c and d are intersected; for $c \sqcap d$ the reader sets are combined.

$$\mathsf{readers}(p, c \sqcup d) \triangleq \mathsf{readers}(p, c) \cap \mathsf{readers}(p, d)$$
$$\mathsf{readers}(p, c \sqcap d) \triangleq \mathsf{readers}(p, c) \cup \mathsf{readers}(p, d)$$

Ordering confidentiality policies. Using the $\mathsf{readers}(\cdot, \cdot)$ function, we can define a "no more restrictive than" relation $\sqsubseteq_C$ on confidentiality policies. For two confidentiality policies c and d, we have $c \sqsubseteq_C d$ if and only if for all principals p, $\mathsf{readers}(p, c) \supseteq \mathsf{readers}(p, d)$. If $c \sqsubseteq_C d$ then every principal p believes that c permits at least as many readers as d does. The confidentiality policy c is thus of lower (or equal) confidentiality than d, and so information labeled c can be used in at least as many places as information labeled d: policy c is no more restrictive than policy d.

The relation $\sqsubseteq_C$ forms a pre-order over confidentiality policies, and the equivalence classes form a lattice. The operators $\sqcup$ and $\sqcap$ are the join and meet operators of this lattice. The least restrictive confidentiality policy is the reader policy $\bot \rightarrow \bot$, where $\bot$ is a principal that all principals can act for, since all principals believe that information labeled $\bot \rightarrow \bot$ is allowed to be read by any principal. The most restrictive expressible confidentiality policy is $\top \rightarrow \top$, where $\top$ is a principal that can act for all principals; information labeled $\top \rightarrow \top$ is allowed to be read only by principal $\top$.

Previous presentations of the DLM have considered only conjunctions of reader policies, resulting in a join semilattice structure. This work adds disjunctions of confidentiality policies, producing a lattice structure that is exploited in Sections 5 and 6, where the meet operation is used to express constraints that enforce robustness in the DLM.

3.2 Integrity policies

Integrity and confidentiality are well-known duals, and we define integrity policies dually to confidentiality policies. The set of *integrity policies* is formed by closing *writer policies* under conjunction and disjunction.

A *writer policy* $o \leftarrow w_1, \ldots, w_n$ allows the owner to specify which principals may have influenced ("written") the value of a given piece of information. The policy $o \leftarrow w_1, \ldots, w_n$ means that according to the owner o, a principal q could have influenced the value of the information only if q can act for the owner o or one of the specified writers $w_1, \ldots, w_n$. Writer policies describe the integrity of information in terms of its provenance.

We define the function $\mathsf{writers}(p, c)$ to be the set of principals that principal p believes may have influenced infor-

mation according to writer policy c. Like reader policies, a principal p believes that writer policy $o \leftarrow w_1, \ldots, w_n$ describes the writers of information only if o can act for p.

$$\mathsf{writers}(p, o \leftarrow w_1, \ldots, w_n) \triangleq \{q \mid \text{if } o \succeq p \text{ then } (q \succeq o \text{ or } \exists i \in 1..n.\ q \succeq w_i)\}$$

Dually to confidentiality policies, we denote disjunction for integrity policies with the operator $\sqcup$, and conjunction with $\sqcap$. The integrity policy $c \sqcap d$ is the conjunction of c and d, meaning that a principal p could have influenced information labeled $c \sqcap d$ only if both c and d agree that p could have influenced it. The writer sets for c and d are thus intersected to produce the writer set for $c \sqcap d$. The integrity policy $c \sqcup d$ is the disjunction of c and d; the writer set for $c \sqcup d$ is thus the union of the writer sets for c and d.

$$\mathsf{writers}(p, c \sqcap d) \triangleq \mathsf{writers}(p, c) \cap \mathsf{writers}(p, d)$$
$$\mathsf{writers}(p, c \sqcup d) \triangleq \mathsf{writers}(p, c) \cup \mathsf{writers}(p, d)$$

The "no more restrictive than" relation $\sqsubseteq_I$ on integrity policies is defined dually to the relation $\sqsubseteq_C$: for two integrity policies c and d, we have $c \sqsubseteq_I d$ if and only if for all principals p, $\mathsf{writers}(p, c) \subseteq \mathsf{writers}(p, d)$. Intuitively, information with a smaller writer set has higher integrity than information with a larger writer set, since fewer principals may have influenced the value of the former; the higher the integrity of information, the fewer restrictions on where that information may be used.

The relation $\sqsubseteq_I$ forms a pre-order over integrity policies, and the equivalence classes form a lattice, with join and meet operators $\sqcup$ and $\sqcap$ respectively. The most restrictive integrity policy is $\bot \leftarrow \bot$, since all principals believe that any principal may have influenced the information. The policy $\top \leftarrow \top$ is the least restrictive expressible integrity policy, as all principals believe that only principal $\top$ (who can act for all other principals) has influenced the information.

3.3 Labels

A label is a pair of a confidentiality policy and an integrity policy. We write a label $\{c; d\}$, where c is a confidentiality policy, and d is an integrity policy. The confidentiality projection of $\{c; d\}$, written $C(\{c; d\})$, is c, and the integrity projection $I(\{c; d\})$ is d. We extend the $\mathsf{readers}(\cdot, \cdot)$ and $\mathsf{writers}(\cdot, \cdot)$ functions appropriately:

$$\mathsf{readers}(p, \{c; d\}) \triangleq \mathsf{readers}(p, c)$$
$$\mathsf{writers}(p, \{c; d\}) \triangleq \mathsf{writers}(p, d)$$

Example. Consider the following label.

$$\{Alice \rightarrow Bob, Chuck\ ;\ Alice \leftarrow Chuck\ \sqcup\ Bob \leftarrow Chuck, Dave\}$$

The confidentiality policy of this label is a single reader policy, and the integrity policy is the disjunction of two writer policies. The reader policy is owned by Alice, and permits any principal that can act for Bob, Chuck, or Alice to read information. No other principal specifies a reader policy, so principals for whom Alice cannot act for allow all principals to read the information; principals that Alice can act for adhere to Alice's restrictions, and permit only principals that can act for Bob, Chuck, or Alice to read information. Of the two writer policies, one is owned by Alice and the other by Bob. Alice believes that only Chuck or Alice could have influenced the information, while Bob believes only principals that can act for any of Chuck, Dave, or Bob could have influenced the information. Principals that neither Alice nor Bob can act for implicitly believe that the information may have been influenced by any principal at all, and is thus completely untrustworthy. A principal that both Alice and Bob can act for believes that principals that can act for Alice, Bob, Chuck, or Dave may have influenced the information.

Ordering labels. We define the "no more restrictive than" relation $\sqsubseteq$ on labels using the relations $\sqsubseteq_C$ and $\sqsubseteq_I$. In particular, $\{c; d\} \sqsubseteq \{c'; d'\}$ if and only if $c \sqsubseteq_C c'$ and $d \sqsubseteq_I d'$. For labels L_1 and L_2, $L_1 \sqsubseteq L_2$ holds if there are the same or more restrictions on uses of information labeled with L_2 as there are on information labeled with L_1.

The relation $\sqsubseteq$ forms a pre-order, whose equivalence classes form a lattice. We use $\sqcup$ and $\sqcap$ for the join and meet operations over this lattice,

$$L_1 \sqcup L_2 \triangleq \{C(L_1) \sqcup C(L_2)\ ;\ I(L_1) \sqcup I(L_2)\}$$
$$L_1 \sqcap L_2 \triangleq \{C(L_1) \sqcap C(L_2)\ ;\ I(L_1) \sqcap I(L_2)\}$$

For the rest of the paper, we assume the set of security levels $\mathcal{L}$ is the set of decentralized labels, and the relation $\sqsubseteq_{\mathcal{L}}$, used in Section 2, is this relation $\sqsubseteq$ on DLM labels.

4 Robustness in the DLM

In the DLM, the security condition of robustness can be generalized to consider attacks launched by an arbitrary principal. To motive this, we first present an example of a simple system with mutually distrusting principals. We then present the definition of *robustness against all attackers*, and derive label constraints that ensure a declassification is robust against all attackers.

4.1 Example

Consider a simple Vickrey auction, shown in Figure 1. There are two bidders, Alice and Bob, abbreviated A and B respectively. There are ten consecutive auctions, indexed by the variable i, each auction for a different item. In each

```
int{⊥→⊥; A←au ⊓ B←au} winner[10];
int{⊥→⊥; A←au ⊓ B←au} i;
for (i = 1..10) {
  int{A→au; A←au ⊓ B←au} bidA = getAliceBid(i);
  int{B→au; A←au ⊓ B←au} bidB = getBobBid(i);

  // end of auction i
  int{⊥→⊥; A←au ⊓ B←au} openA =
        declassify(bidA, {⊥→⊥; A←au ⊓ B←au});
  int{⊥→⊥; A←au ⊓ B←au} openB =
        declassify(bidB, {⊥→⊥; A←au ⊓ B←au});

  // compute winner
  winner[i] = computeWinner(openA, openB);

  // process payment of winning bid
  ...
}
```

Figure 1. Vickrey auction example.

auction, both bidders submit a secret bid; after all bids for the ith auction have been submitted, the secret bids are declassified, and the winner computed. We model each bidder as a principal, and have an auctioneer principal au. We assume there are no *acts-for* relationships between these principals. Every variable in the program is annotated with a security label from the DLM, which is enforced on information stored in the variable.

Consider the auction program from Alice's perspective. In each auction, Alice submits a bid, stored in variable bidA, with the label $\{A \rightarrow au; A \leftarrow au \sqcap B \leftarrow au\}$ enforced on it. Thus, Alice specifies that her bid should be readable only by the auctioneer, and both Alice and Bob are prepared to accept the bid as high integrity, influenced only by the auctioneer (due to his ability to control when the ith auction commences). After Bob has submitted his bid, Alice's bid is declassified to $\{\bot \rightarrow \bot; A \leftarrow au \sqcap B \leftarrow au\}$, allowing the bid to be read by all principals, and stored in openA.

Alice may be concerned with attempts by Bob to corrupt the auction. For example, could Bob corrupt the control flow so that Alice's bid is declassified before Bob has submitted his bid, permitting Bob to always win with the minimal winning bid? Or could Bob alter the value stored in bidA, and fool the system into releasing sensitive information of Alice's, such as her credit card number, or her bid for auction $i + 1$?

Alice would like assurance that the program is robust against attacks by Bob. However, Bob also needs assurance that the program is robust against attacks by Alice. And both principals may be concerned with the auctioneer's ability to corrupt the auction. Even in this simple example there are several potential attackers, and it is necessary to reason about robustness against all possible attackers.

4.2 Robustness against all attackers

The power of an attacker A is defined by the pair of labels $\langle R_A, W_A \rangle$, which bound the information that A can observe and influence. In the setting of Myers, Sabelfeld and Zdancewic [12], there is no *a priori* relationship between R_A and W_A, making it difficult to characterize an arbitrary attacker's power, and therefore difficult to prove robustness against all possible attackers.

However, in the DLM the power of an attacker A can be expressed in terms of the attacker's identity, because all entities are represented by principals. Moreover, we can express the power of an attacker as perceived by a particular principal: for principals p and q, the security levels $R_{p\rightarrow q}$ and $W_{p\leftarrow q}$ are bounds on the labels of information that p believes q can read and write:

Definition 2 *The label $R_{p\rightarrow q}$ is the least upper bound on labels of information that principal p believes principal q can read:*

$$L \sqsubseteq R_{p\rightarrow q} \text{ if and only if } q \in \mathsf{readers}(p, L)$$

The label $W_{p\leftarrow q}$ is the greatest lower bound on labels of information that principal p believes principal q can influence:

$$W_{p\leftarrow q} \sqsubseteq L \text{ if and only if } q \in \mathsf{writers}(p, L)$$

The labels $R_{p\rightarrow q}$ and $W_{p\leftarrow q}$ cannot be expressed as conjunctions and disjunctions of reader and writer policies. We can, however, characterize their reader and writer sets.

$$\begin{aligned}
\mathsf{readers}(r, R_{p\rightarrow q}) &\triangleq \{q' \mid q' \succeq q \text{ and } r \succeq p\} \\
\mathsf{writers}(r, R_{p\rightarrow q}) &\triangleq \{q' \mid q' \text{ is a principal}\} \\
\mathsf{readers}(r, W_{p\leftarrow q}) &\triangleq \{q' \mid q' \text{ is a principal}\} \\
\mathsf{writers}(r, W_{p\leftarrow q}) &\triangleq \{q' \mid q' \succeq q \text{ and } p \succeq r\}
\end{aligned}$$

We extend the labels of the DLM to include the labels $R_{p\rightarrow q}$ and $W_{p\leftarrow q}$ for all principals p and q. The definition of the label relation $\sqsubseteq$ is extended in the obvious way, using the definitions for $R_{p\rightarrow q}$ and $W_{p\leftarrow q}$ given above. The key property, that $\sqsubseteq$ forms a pre-order whose equivalence classes are a lattice, continues to hold.

Figure 2 depicts the points $R_{p\rightarrow q}$ and $W_{p\leftarrow q}$ in the product lattice of confidentiality and integrity. Their confidentiality and integrity levels divide the set of all labels into four subsets characterized by the power of the attacker to either read or write information with those labels.

Having precisely described an arbitrary attacker's power, we can now define robustness against all attackers.

Definition 3 (Robustness against all attackers) *A system has* robustness against all attackers *if for all principals p and q, the system has robustness with respect to attacker q with power $\langle R_{p\rightarrow q}, W_{p\leftarrow q} \rangle$.*

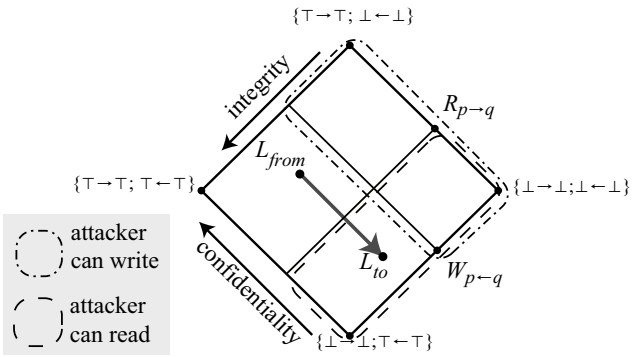

Figure 2. Robust declassification in a confidentiality–integrity product lattice.

If a system is robust against all attackers, then every principal p believes that the system is robust against attacks by any principal q.

4.3 Constraints for checking robustness

As will be seen in Section 5, the key to enforcing robustness is to ensure that if a declassification reveals information to attacker A, then A is unable to influence either the decision to declassify, or the data to be declassified. This requirement has a very natural expression in the DLM.

Suppose L_{from} is the label of the information to be declassified, L_{to} is the label of the declassified information, and pc is the *program counter label*, an upper bound on the labels of information contributing to the decision to declassify. If, from the perspective of a principal p, the declassification reveals information to a principal q, then $q \in \mathsf{readers}(p, L_{to}) - \mathsf{readers}(p, L_{from})$; if this is the case, then we require that q cannot influence either the decision to declassify ($q \notin \mathsf{writers}(p, pc)$), or the data to be declassified ($q \notin \mathsf{writers}(p, L_{from})$).

Figure 2 shows part of this requirement graphically: if the declassification from L_{from} to L_{to} crosses the line defined by $R_{p\to q}$ (i.e., $q \in \mathsf{readers}(p, L_{to}) - \mathsf{readers}(p, L_{from})$) then L_{from} should not be above the line defined by $W_{p\leftarrow q}$ (i.e., $q \notin \mathsf{writers}(p, L_{from})$).

Since we would like this requirement to hold from every principal's perspective, and for all principals that the declassification may reveal information to, the following statement should hold at every declassification:

$$\forall p.\ \forall q \in \mathsf{readers}(p, L_{to}).\ q \in \mathsf{readers}(p, L_{from}) \text{ or } (q \notin \mathsf{writers}(p, pc) \text{ and } q \notin \mathsf{writers}(p, L_{from})) \quad (1)$$

Unfortunately, it is difficult to prove directly that this statement is true: membership of the sets $\mathsf{writers}(p, pc)$ and $\mathsf{writers}(p, L_{from})$ depends upon the *acts-for* relation $\succeq$, and we may have only partial knowledge of the *acts-for* relation that will be in effect at run time [2]. Demonstrating that a principal q is not a member of $\mathsf{writers}(p, pc)$ or $\mathsf{writers}(p, L_{from})$ is impossible.

However, the following two label constraints suffice to entail condition (1).

$$L_{from} \sqsubseteq L_{to} \sqcup \mathsf{writersToReaders}(pc) \quad (2)$$

$$L_{from} \sqsubseteq L_{to} \sqcup \mathsf{writersToReaders}(L_{from}) \quad (3)$$

These label constraints can be verified syntactically, with only partial knowledge of the *acts-for* relation [11]. The label constraints make use of operator $\mathsf{writersToReaders}(L)$, which converts the writers of label L into readers of label $\mathsf{writersToReaders}(L)$.

$$\begin{aligned} \mathsf{writersToReaders}(L) &\triangleq \{\mathsf{wtr}(I(L)); \top \leftarrow \top\} \\ \mathsf{wtr}(c \sqcup d) &\triangleq \mathsf{wtr}(c) \sqcap \mathsf{wtr}(d) \\ \mathsf{wtr}(c \sqcap d) &\triangleq \mathsf{wtr}(c) \sqcup \mathsf{wtr}(d) \\ \mathsf{wtr}(o \leftarrow w_1, \ldots, w_n) &\triangleq o \rightarrow w_1, \ldots, w_n \end{aligned}$$

We do not define $\mathsf{writersToReaders}(\cdot)$ for the labels $R_{p\to q}$ or $W_{p\leftarrow q}$. Although suitable definitions could be given, we ensure that $R_{p\to q}$ and $W_{p\leftarrow q}$ never appear in label constraints (2) or (3).

The key property of $\mathsf{writersToReaders}(\cdot)$ is that if principal p believes q is a writer of label L, then p believes q is a reader of $\mathsf{writersToReaders}(L)$:

Property 1 *For all labels L, and all principals p and q, if $q \in \mathsf{writers}(p, L)$, then $q \in \mathsf{readers}(p, \mathsf{writersToReaders}(L))$.*

Proof: By induction on the structure of the integrity policy $I(L)$, exploiting the duality between confidentiality and integrity policies. □

The following lemma shows that if constraints (2) and (3) hold, then condition (1) holds, that is, every principal p believes that if the declassification reveals information to principal q, then q could not have influenced the decision to declassify or the information to be declassified.

Lemma 1 *If $L_{from} \sqsubseteq L_{to} \sqcup \mathsf{writersToReaders}(pc)$ and $L_{from} \sqsubseteq L_{to} \sqcup \mathsf{writersToReaders}(L_{from})$ then $\forall p.\ \forall q \in \mathsf{readers}(p, L_{to}).\ q \in \mathsf{readers}(p, L_{from})$ or $(q \notin \mathsf{writers}(p, pc)$ and $q \notin \mathsf{writers}(p, L_{from}))$.*

Proof: Assume $L_{from} \sqsubseteq L_{to} \sqcup \mathsf{writersToReaders}(pc)$ and $L_{from} \sqsubseteq L_{to} \sqcup \mathsf{writersToReaders}(L_{from})$. Let p be a principal, and let $q \in \mathsf{readers}(p, L_{to})$. If $q \in \mathsf{readers}(p, L_{from})$ then we are done. Suppose $q \notin \mathsf{readers}(p, L_{from})$. From the definition of $\sqsubseteq$, we have $\mathsf{readers}(p, L_{from}) \supseteq \mathsf{readers}(p, L_{to}) \cap \mathsf{readers}(p, \mathsf{writersToReaders}(pc))$. If $q \in \mathsf{writers}(p, pc)$ then by Property 1 we have

$e ::= val \mid v \mid e_1 \text{ op } e_2 \mid \mathsf{declassify}(e, \ell)$

$c ::= \mathsf{skip} \mid v := e \mid c_1; c_2 \mid \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ e\ \mathsf{do}\ c$

Figure 3. Language grammar

$q \in \mathsf{readers}(p, \mathsf{writersToReaders}(pc))$. But then $q \in \mathsf{readers}(p, L_{to}) \cap \mathsf{readers}(p, \mathsf{writersToReaders}(pc))$, and so we have $q \in \mathsf{readers}(p, L_{from})$, a contradiction. So $q \notin \mathsf{writers}(p, pc)$. By a similar argument, $q \notin \mathsf{writers}(p, L_{from})$. □

Consider the declassification of Alice's bid in the auction example of Section 4.1. The label of Alice's bid is $\{A \to au; A \leftarrow au \sqcap B \leftarrow au\}$, and it is declassified to the label $\{\bot \to \bot; A \leftarrow au \sqcap B \leftarrow au\}$. The program counter at the declassification depends only on the variable i, so the pc label is $\{\bot \to \bot; A \leftarrow au \sqcap B \leftarrow au\}$. Instantiating label constraints (2) and (3) for these labels results in the following constraint:

$$\{A \to au; A \leftarrow au \sqcap B \leftarrow au\} \sqsubseteq \{\bot \to \bot \sqcup A \to au \sqcup B \to au; A \leftarrow au \sqcap B \leftarrow au\}$$

The integrity policies of both of these labels are identical, and the reader policy of the left hand side ($A \to au$) is contained in a join on the right hand side, so the constraint is satisfied. This implies that every principal believes that any principal who gains the ability to read Alice's bid is unable to influence either the value declassified, or the decision to declassify that value. Thus, Alice believes that if the auctioneer is trusted, then the declassification will never reveal anything other than Alice's bid, and will not occur other than at the appropriate time.

5 Enforcing robustness

In this section, we consider enforcing robustness against all attackers in the setting of a simple imperative language. After introducing the language, we refine the definitions of robustness and robustness against all attackers, and present a type system for enforcing robustness against all attackers.

The language grammar is presented in Figure 3; it is identical to that of Myers, Sabelfeld and Zdancewic [12]. The language and its semantics are standard, with the exception of the explicit declassification operator, $\mathsf{declassify}(e, \ell)$, which declassifies expression e to security level ℓ, and is operationally equivalent to the expression e. The language includes literal values $val \in Val$, and variables $v \in Var$. A variable context $\Gamma : Var \to \mathcal{L}$ maps each variable to a security level that is an upper bound (with respect to $\sqsubseteq_{\mathcal{L}}$) on the security level of information that can be stored in the variable. The range of Γ is restricted to labels of pairs of confidentiality and integrity policies—$R_{p \to q}$ and $W_{p \leftarrow q}$ are not permitted as security levels of variables. The function $Vars(e)$ returns the set of variables that occur in the expression e.

This language captures the key aspects of language-based declassification, while being simple enough to permit straightforward proofs. In Section 7 we apply the type system to Jif, a more complex and practical language.

5.1 Defining robustness

In order to give a meaningful definition of robustness (and robustness against all attackers) in this language-based setting, we must first define what attacks can be made by an attacker A with power $\langle R_A, W_A \rangle$. Following Myers, Sabelfeld and Zdancewic, we define an attack by A to be a command a that will be inserted into a program. The attack a is not arbitrary code, but is restricted to a subset of the language, to model "fair" attacks. The allowed attacks are defined by the following grammar, where variable v can be read or updated only if the security label $\Gamma(v)$ permits these accesses by the attacker.

$a ::= \mathsf{skip} \mid v := e \mid a_1; a_2 \mid \mathsf{if}\ e\ \mathsf{then}\ a_1\ \mathsf{else}\ a_2 \mid \mathsf{while}\ e\ \mathsf{do}\ a$

The allowed attacks do not include declassifications, because if the attacker can declassify confidential information directly, the game is already over.

Attacks may be inserted into the program at points where the attacker is able to influence the execution of code. For example, in a distributed system, the attacker may be able to insert attacks on a server that is under the attacker's control. Myers, Sabelfeld and Zdancewic assume that program points at which an attacker may insert attacks are explicitly marked by *code holes* (•). There may be multiple holes in a command, represented as a vector of holes $\vec{\bullet}$; the holes in a program $c[\vec{\bullet}]$ will be replaced with a vector of attacks $\vec{a}$ to obtain a complete (hole-free) program, written $c[\vec{a}]$. The grammar of commands with holes $c[\vec{\bullet}]$ extends the command grammar from Figure 3:

$$c[\vec{\bullet}] ::= \ldots \mid [\bullet]$$

We can now refine Definition 1, the definition of robustness, for this language-based setting. A configuration is a pair $\langle M, c \rangle$ of memory M and command c. A memory is a function from $Vars$ to Val. Configurations $\langle M, c \rangle$ and $\langle M', c' \rangle$ are indistinguishable to A (written $\langle M, c \rangle =_A \langle M', c' \rangle$) if for all variables $v \in Var$, if $\Gamma(v) \sqsubseteq_{\mathcal{L}} R_A$ then $M(v) = M'(v)$. Traces are indistinguishable to A if the sequence of configurations are equivalent (according to $=_A$) up to stuttering. The definition of trace indistinguishability is enough to define weak indistinguishability ($\approx_A$) and strong indistinguishability ($\cong_A$).

Definition 4 (Robustness) *Command $c[\vec{\bullet}]$ has* robustness *with respect to attacks by attacker A with power $\langle R_A, W_A \rangle$*

if for all memories M_1 and M_2, and all attacks $\vec{a}$ and $\vec{a'}$ by attacker A, if $\langle M_1, c[\vec{a}]\rangle \cong_A \langle M_2, c[\vec{a}]\rangle$, then $\langle M_1, c[\vec{a'}]\rangle \approx_A \langle M_2, c[\vec{a'}]\rangle$.

This refinement of robustness assumes that the code holes where an attacker may insert code are explicitly given; however, in general, the location of code holes depends upon which attacker we are considering. Since we are concerned with the possibility of many attackers, we need to reason about the security of code into which different attackers may insert code at different locations.

To indicate where code holes may be inserted for a given attacker A, we assume the existence of a *hole insertion relation* $\lhd_A$. Let $c \lhd_A d[\vec{\bullet}]$ denote that the command with holes $d[\vec{\bullet}]$ can be obtained by inserting code holes into command c at program points where attacker A is able to insert code. The actual form of the hole-insertion relation depends on the system. For example, in the context of automatic program partitioning [20] (in which a program is automatically partitioned into code segments executed on different servers), an attacker may be able to insert code into any segment that is placed on a server controlled by the attacker.

For the purposes of this paper, we require only that the hole insertion relation $\lhd_A$ does not allow holes to be inserted into high-confidentiality contexts. That is, an attacker may not insert code at a program point whose execution depends upon information with a security level not bounded above by R_A. In the context of automatic program partitioning, program points in a high-confidentiality context correspond to code segments whose very execution would insecurely reveal sensitive information to the attacker; such code segments are never placed on a server where the attacker could insert attacks. More formally, we define the property of *safe hole insertion* as follows.

Definition 5 (Safe hole insertion) *A hole insertion relation $\lhd_A$ is* safe *if whenever $c \lhd_A d[\vec{\bullet}]$, then for all holes in $d[\vec{\bullet}]$, if the hole is a subcommand of a command* $\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$ *or a subcommand of a command* $\mathsf{while}\ e\ \mathsf{do}\ c_1$*, then for all variables $v \in Vars(e)$, we have $\Gamma(v) \sqsubseteq R_A$.*

We can now refine Definition 3, the definition of robustness against all attackers, for the specific language-based setting presented here.

Definition 6 (Robustness against all attackers) *Command c has* robustness against all attackers *if for all principals p and q, and all commands with holes $d[\vec{\bullet}]$ such that $c \lhd_q d[\vec{\bullet}]$, command $d[\vec{\bullet}]$ has robustness with respect to attacker q with power $\langle R_{p\to q}, W_{p\leftarrow q}\rangle$.*

5.2 Enforcing robustness

Myers, Sabelfeld and Zdancewic present a type system, parameterized on a single attacker A, that enforces robustness against attacks by A. However, due to the number of possible attackers, it is infeasible to show directly that a program is well-typed in their type system instantiated on each and every possible attacker A. Instead, we ensure that a program that is well-typed in our type system is also well-typed with respect to Myers, Sabelfeld and Zdancewic's type system instantiated on any possible attacker. This implies that our type system enforces robustness against all attackers.

Figure 4 presents Myers, Sabelfeld and Zdancewic's type system, adapted slightly for our purposes. The judgment $\Gamma, pc \vdash_A c$ indicates that command c is well typed under variable context Γ and program counter label pc; the judgment $\Gamma \vdash_A e : \ell$ indicates that the expression e is well typed under variable context Γ, and ℓ is an upper bound on the security level of the information that e depends upon. The attacker A appears in the typing rules for $v := \mathsf{declassify}(e, \ell)$ and command holes $[\bullet]$. All other typing rules are standard for an imperative security-typed language.

The key idea of enforcing robustness is to ensure that if a declassification may reveal information to an attacker A, then both the data to be declassified and the decision to declassify information cannot have been influenced by A. The typing rule for declassification is where this restriction is enforced: the constraints $W_A \not\sqsubseteq_{\mathcal{L}} pc$ and $W_A \not\sqsubseteq_{\mathcal{L}} \ell'$ ensure that both the decision to declassify and the information to be declassified are high-integrity with respect to the attacker's power.

We have adapted Myers, Sabelfeld and Zdancewic's type system by using two different typing rules for declassifications. The first is for declassifications that reveal information to the attacker A; that is, information is declassified from security level ℓ' (where A cannot read information) to security level ℓ (where A can read information). The second rule is for declassifications that do not reveal information to attacker A, either because the attacker could already read information at level ℓ', or because after declassification the attacker is still unable to read the information at its new level ℓ. Only the first rule needs to enforce the robustness conditions; Myers, Sabelfeld and Zdancewic's original system did not contain the second rule, requiring suitably high integrity for all declassifications, even if they do not reveal information to the attacker. We modify their declassification typing rule in anticipation of enforcing robustness against all attackers.

The rule for command holes restricts holes from occurring in high-confidentiality contexts, which ensures that an attacker is unable to observe sensitive information through implicit flows [6].

Theorem 1 *If $\Gamma, pc \vdash_A c$ then command c has robustness with respect to attacker A.*

Proof: Similar to Myers, Sabelfeld and Zdancewic's, adapted for the modified declassification typing rules. □

$$\frac{}{\Gamma \vdash_A val : \ell} \qquad \frac{\Gamma(v) = \ell}{\Gamma \vdash_A v : \ell} \qquad \frac{\Gamma \vdash_A e : \ell \quad \Gamma \vdash_A e' : \ell}{\Gamma \vdash_A e \text{ op } e' : \ell} \qquad \frac{\Gamma \vdash_A e : \ell \quad \ell \sqsubseteq_{\mathcal{L}} \ell'}{\Gamma \vdash_A e : \ell'} \qquad \frac{}{\Gamma, pc \vdash_A \mathsf{skip}} \qquad \frac{\Gamma \vdash_A e : \ell \quad \ell \sqcup pc \sqsubseteq_{\mathcal{L}} \Gamma(v)}{\Gamma, pc \vdash_A v := e}$$

$$\frac{\Gamma, pc \vdash_A c_1 \quad \Gamma, pc \vdash_A c_2}{\Gamma, pc \vdash_A c_1; c_2} \qquad \frac{\Gamma \vdash_A e : \ell \quad \Gamma, \ell \sqcup pc \vdash_A c_1 \quad \Gamma, \ell \sqcup pc \vdash_A c_2}{\Gamma, pc \vdash_A \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2} \qquad \frac{\Gamma \vdash_A e : \ell \quad \Gamma, \ell \sqcup pc \vdash_A c}{\Gamma, pc \vdash_A \mathsf{while}\ e\ \mathsf{do}\ c} \qquad \frac{\Gamma, pc \vdash_A c \quad pc' \sqsubseteq_{\mathcal{L}} pc}{\Gamma, pc' \vdash_A c} \qquad \frac{pc \sqsubseteq_{\mathcal{L}} R_A}{\Gamma, pc \vdash_A [\bullet]}$$

$$\frac{\Gamma \vdash_A e : \ell' \quad \ell \sqcup pc \sqsubseteq_{\mathcal{L}} \Gamma(v) \quad I(\ell') \sqsubseteq_{\mathcal{L}} I(\ell) \quad \ell \sqsubseteq_{\mathcal{L}} R_A \quad \ell' \not\sqsubseteq_{\mathcal{L}} R_A \quad W_A \not\sqsubseteq_{\mathcal{L}} pc \quad W_A \not\sqsubseteq_{\mathcal{L}} \ell'}{\Gamma, pc \vdash_A v := \mathsf{declassify}(e, \ell)} \qquad \frac{\Gamma \vdash_A e : \ell' \quad \ell \sqcup pc \sqsubseteq_{\mathcal{L}} \Gamma(v) \quad I(\ell') \sqsubseteq_{\mathcal{L}} I(\ell) \quad \ell \not\sqsubseteq_{\mathcal{L}} R_A \text{ or } \ell' \sqsubseteq_{\mathcal{L}} R_A}{\Gamma, pc \vdash_A v := \mathsf{declassify}(e, \ell)}$$

Figure 4. Typing rules for $\Gamma \vdash_A e : \ell$ and $\Gamma, pc \vdash_A c$

$$\frac{\Gamma \vdash e : \ell' \qquad \ell \sqcup pc \sqsubseteq \Gamma(v) \quad \ell' \sqsubseteq \ell \sqcup \mathsf{writersToReaders}(pc) \quad \ell' \sqsubseteq \ell \sqcup \mathsf{writersToReaders}(\ell')}{\Gamma, pc \vdash v := \mathsf{declassify}(e, \ell)}$$

Figure 5. Declassification rule for $\Gamma, pc \vdash c$

To enforce robustness against all attackers, we derive a type system using constraints (2) and (3) given in Section 4.3. This type system ensures that for all principals p and q, a well-typed program is robust against attacks by q with power $\langle R_{p\to q}, W_{p\leftarrow q}\rangle$. We prove this by showing that a well-typed program is also well-typed in Myers, Sabelfeld and Zdancewic's type system instantiated on the attacker q with power $\langle R_{p\to q}, W_{p\leftarrow q}\rangle$.

The new typing judgments are $\Gamma, pc \vdash c$ (command c is well typed under variable context Γ and program counter label pc) and $\Gamma \vdash e : \ell$ (expression e is well typed under variable context Γ, and ℓ is an upper bound on the security level of the information that e depends upon). All inference rules for $\Gamma \vdash_A e : \ell$ are also inference rules for $\Gamma \vdash e : \ell$. All inference rules for $\Gamma, pc \vdash_A c$ are also inference rules for $\Gamma, pc \vdash c$, with the exception of the rules for declassification and command holes. There is no need for a rule for command holes, as we are only concerned with complete programs; holes are introduced through hole insertion relations $\lhd_A$. The new declassification rule, which replaces both previous declassification rules, is shown in Figure 5. The inference rules for the new typing judgments contain no negated label ordering relations ($\not\sqsubseteq$), which is consistent with having only partial knowledge of the *acts-for* relation in effect at run time.

Theorem 2 *If $\Gamma, pc \vdash c$ then command c has robustness against all attackers.*

Proof: Let $\Gamma, pc \vdash c$, let p and q be principals, and let $d[\vec{\bullet}]$ be a command with holes such that $c \lhd_q d[\vec{\bullet}]$. We show that $\Gamma, pc \vdash_A d[\vec{\bullet}]$, where $A = q$, and thus by Theorem 1, $d[\vec{\bullet}]$ has robustness with respect to attacker q with power $\langle R_{p\to q}, W_{p\leftarrow q}\rangle$.

We first show $\Gamma, pc \vdash_A c$, by induction on $\Gamma, pc \vdash c$. The only interesting case is for declassification. If the declassification doesn't reveal information to A (i.e., either $\ell \not\sqsubseteq_{\mathcal{L}} R_A$ or $\ell' \sqsubseteq_{\mathcal{L}} R_A$), then the declassification type-checks with the second declassification rule. If it does reveal information to A, then by definition of $R_{p\to q}$ and $W_{p\leftarrow q}$ and Lemma 1 we have $W_{p\leftarrow q} \not\sqsubseteq pc$ and $W_{p\leftarrow q} \not\sqsubseteq \ell'$ as required.

Given that $\Gamma, pc \vdash_A c$ and $c \lhd_q d[\vec{\bullet}]$, we can use the hole safety of $\lhd_q$ to show that $\Gamma, pc \vdash_A d[\vec{\bullet}]$, since code holes can only be introduced in low-confidentiality contexts, and thus any hole type-checks. □

6 Qualified robustness

Robustness is concerned with systems in which the confidentiality of information can be downgraded. In some systems, it is also necessary to downgrade the integrity of information, also called *endorsement* of information. For instance, in the auction example of Section 4.1, Alice submits a bid via the function getAliceBid(), which returns a value with integrity $A \leftarrow au \sqcap B \leftarrow au$. Clearly Alice influences the value of the bid, yet Bob is prepared to treat the information as high integrity, since the actual value of Alice's bid is unimportant for the security properties of the auction. An endorsement is thus required in the implementation of the function getAliceBid().

Robustness does not hold when a system endorses data that is subsequently used to determine what information is declassified. For example, consider a system where a user pays a fee and then chooses one of two files to download. The system is prepared to endorse the user's choice, to treat is as high-integrity data, since the system is prepared to allow either file to be downloaded. However, the user's choice controls which file is declassified for download, violating robustness with respect to attacks by the user.

Myers, Sabelfeld and Zdancewic define a weaker security condition that may hold in the presence of endorsement: *qualified robustness* [12]. Qualified robustness allows an attacker limited ability to influence what information is de-

$$\frac{\ell \sqcup pc \sqsubseteq_{\mathcal{L}} \Gamma(v) \qquad C(\ell') \sqsubseteq_{\mathcal{L}} C(\ell) \quad \Gamma \vdash_A e : \ell' \quad W_A \not\sqsubseteq_{\mathcal{L}} \ell \quad W_A \sqsubseteq_{\mathcal{L}} \ell' \quad W_A \not\sqsubseteq_{\mathcal{L}} pc}{\Gamma, pc \vdash_A v := \mathsf{endorse}(e, \ell)}$$

$$\frac{\ell \sqcup pc \sqsubseteq_{\mathcal{L}} \Gamma(v) \qquad C(\ell') \sqsubseteq_{\mathcal{L}} C(\ell) \quad \Gamma \vdash_A e : \ell' \qquad W_A \sqsubseteq_{\mathcal{L}} \ell \text{ or } W_A \not\sqsubseteq_{\mathcal{L}} \ell'}{\Gamma, pc \vdash_A v := \mathsf{endorse}(e, \ell)}$$

$$\frac{\Gamma \vdash e : \ell' \quad \ell \sqcup pc \sqsubseteq \Gamma(v) \quad \ell' \sqcap \mathsf{writersOnly}(pc) \sqsubseteq \ell}{\Gamma, pc \vdash v := \mathsf{endorse}(e, \ell)}$$

Figure 6. Additional typing rules for endorse

classified. This is made explicit in the language by providing a new expression: $\mathsf{endorse}(e, \ell)$. Intuitively, a command has qualified robustness if the only control that an attacker exerts over information release is via explicit endorsement.

To formally define qualified robustness, Myers, Sabelfeld and Zdancewic define an alternate nondeterministic semantics for the language, in which the evaluation of an $\mathsf{endorse}(e, \ell)$ expression returns all possible values. This captures the idea that the system is prepared to treat the result of evaluating $\mathsf{endorse}(e, \ell)$ as high-integrity data, regardless of the actual value of the expression e.

The statement of qualified robustness is syntactically identical to that of robustness; it differs only in the new program semantics.

Definition 7 (Qualified robustness) *Command $c[\vec{\bullet}]$ has* qualified robustness *with respect to fair attacks by attacker A with power $\langle R_A, W_A \rangle$ if for all memories M_1 and M_2, and all attacks $\vec{a}$ and $\vec{a'}$ by attacker A, if $\langle M_1, c[\vec{a}] \rangle \approx\!\!\!\approx_A \langle M_2, c[\vec{a}] \rangle$, then $\langle M_1, c[\vec{a'}] \rangle \approx_A \langle M_2, c[\vec{a'}] \rangle$.*

Just as robustness can be generalized to qualified robustness to account for the downgrading of integrity of information, so robustness against all attackers can be generalized to qualified robustness against all attackers.

Definition 8 (Qualified robustness against all attackers) *Command c has* qualified robustness against all attackers *if for all principals p and q, and all commands with holes $d[\vec{\bullet}]$ such that $c \lhd_q d[\vec{\bullet}]$, command $d[\vec{\bullet}]$ has qualified robustness with respect to attacker q with power $\langle R_{p \to q}, W_{p \leftarrow q} \rangle$.*

6.1 Enforcing qualified robustness

The key idea for enforcing qualified robustness is similar to that for enforcing robustness: the decision to endorse information must be of high integrity. However, unlike robustness, the information being downgraded does not need to be high integrity. (Indeed, by definition, the information to be endorsed is low integrity.)

Myers, Sabelfeld and Zdancewic extend their type system with a rule for endorse statements, which we adapt and present in Figure 6. As with the two rules for declassification, endorsement has two rules: one for raising the integrity of data beyond the attacker's ability to modify it, and the other for leaving integrity unchanged from the attacker's perspective. Note that the first typing rule for endorsement requires the program counter label pc to be high integrity, ensuring that only high-integrity information influences the decision to endorse.

Theorem 3 *If $\Gamma, pc \vdash_A c$ then command c has qualified robustness with respect to attacker A.*

Proof: Similar to Myers, Sabelfeld and Zdancewic's, adapted for the modified endorsement typing rules. □

Qualified robustness against all attackers can be enforced through a suitable label constraint, which we derive analogously to the constraints for robustness against all attackers. If a principal p believes that the endorsement removes a principal q from the writer set, then q should be unable to influence the decision to endorse the data. That is, if $q \in \mathsf{writers}(p, L_{from}) - \mathsf{writers}(p, L_{to})$, then we require $q \notin \mathsf{writers}(p, pc)$. This requirement should hold for all principals p and q. From this requirement we can derive a sufficient label constraint:

$$L_{from} \sqcap \mathsf{writersOnly}(pc) \sqsubseteq L_{to} \tag{4}$$

The label operator $\mathsf{writersOnly}(\cdot)$ strips away the confidentiality policy of a label. More precisely, for any label L and we define $\mathsf{writersOnly}(L) \triangleq \{\top \to \top; I(L)\}$.

The following lemma shows that if label constraint (4) holds, then every principal believes that if the endorsement removes a principal q from the writer set, then q could not have influenced the decision to endorse.

Lemma 2 *If $L_{from} \sqcap \mathsf{writersOnly}(pc) \sqsubseteq L_{to}$ then $\forall p.\ \forall q \in \mathsf{writers}(p, L_{from}).\ q \in \mathsf{writers}(p, L_{to})$ or $q \notin \mathsf{writers}(p, pc)$.*

Proof: Similar to proof of Lemma 1. □

Label constraint (4) leads to a new typing rule for endorsement, given in Figure 6. Adding this rule to the type system ensures qualified robustness against all attackers.

Theorem 4 *If $\Gamma, pc \vdash c$ then command c has qualified robustness against all attackers.*

Proof: Similar to proof of Theorem 2. □

7 Jif implementation

The type system for enforcing robustness against all attackers is a practical one. Unlike the type system of Myers, Sabelfeld and Zdancewic, it is not parameterized on a particular attacker. We have adapted the type system for enforcing robustness against all attackers and implemented it in the Jif compiler [10, 13].

```
1  final label{⊥ → ⊥; ⊤ ← ⊤} lbl = ...;
2  int{∗lbl} i = ...;
3  if (lbl ⊑ {Alice → Bob; Alice ← Chuck}) {
4     int j = declassify(i, {⊥ → ⊥; Alice ← Chuck});
5  }
```

Figure 7. Example Jif code

Jif extends the Java [8] programming language with information flow control, using DLM labels to annotate variables and methods. The Jif compiler checks that the policies specified by label annotations in Jif programs are obeyed.

To implement robustness against all attackers, we first extended Jif with the modified DLM of Section 3 to allow the expression of the required label constraints. Although some Jif variants [20] incorporate (less expressive) integrity policies, the base Jif compiler did not. We implemented writer policies, and extended the label ordering within Jif to allow for meets of labels. These modifications did not require extensive changes to the Jif compiler.

7.1 Implementing writersToReaders(·)

Jif contains a number of sophisticated type mechanisms to facilitate the creation, use, and reuse of expressive, modular, secure code [10]. These mechanisms include parameterized classes, labels for method arguments, dynamic labels [21] and dynamic principals [17]. Labels in Jif thus include class parameters, argument labels, and dynamic labels, as well as pairs of confidentiality and integrity policies. This complicates the implementation of the label constraints of Section 5.

In particular, writersToReaders(·) must be extended to account for the additional labels in Jif, such as dynamic labels. A dynamic label is a first-class value that is represented at run time. A dynamic label dyn(x) is equivalent to the value stored in a final variable named x, of type label; however, at compile time, it may not be known what the value of variable x will be at run time. Dynamic labels may be used to label other variables. For example, in the Jif code of Figure 7, the label of the variable i is dyn(lbl), the value that is held at run time in the dynamic label variable lbl.

In extending writersToReaders(·) to dynamic labels, we must ensure that Property 1 continues to hold: for all principals p and q, and labels L, if $q \in \mathsf{writers}(p, L)$, then $q \in \mathsf{readers}(p, \mathsf{writersToReaders}(L))$. Clearly, we could safely define writersToReaders(dyn(x)) to be $\{\bot \to \bot; \top \leftarrow \top\}$ for all possible x; since $\mathsf{readers}(p, \{\bot \to \bot; \top \leftarrow \top\})$ is the set of all principals, this would satisfy Property 1. While imprecise, this is the best we can do without additional information about the dynamic label.

However, in some situations, a given dynamic label has an upper bound. For example, at the declassification statement in line 4 of Figure 7, we know statically that {Alice → Bob; Alice ← Chuck} is an upper bound on the dynamic label lbl, because of the run-time label test on line 3.

If L is an upper bound for dyn(x), we have dyn(x) $\sqsubseteq L$, and so for any principal p, by definition of $\sqsubseteq$, we have $\mathsf{writers}(p, \mathsf{dyn}(x)) \subseteq \mathsf{writers}(p, L)$. Thus, if L consists only of reader and writer policies, then writersToReaders(L) is a conservative approximation for writersToReaders(dyn(x)), and if $q \in \mathsf{writers}(p, \mathsf{dyn}(x))$, then $q \in \mathsf{readers}(p, \mathsf{writersToReaders}(L))$.

We can extend this technique for approximating writersToReaders(·) to other labels, such as class parameters and polymorphic argument labels. Other than the implementation of writersToReaders(·), there were no major challenges in adapting and implementing the type system of Section 5 for Jif.

8 Related work

It has been recognized since the beginnings of work on information flow that noninterference is too rigid to describe the information security of real applications [5, 7]. There has been a great deal of work on mechanisms and security definitions that relax noninterference in various ways. Much of this work (particularly, that which is language-based) has been summarized in a survey paper [15]. More recently, Sabelfeld and Sands constructed a taxonomy of different methods for securely mediating information release [16]. According to their taxonomy, robustness operates on the "who" dimension of declassification controls, as it prevents untrusted entities from affecting how information is released to them. We compare the robustness mechanism in this paper to other mechanisms operating along the "who" dimension, which is not to disparage the other, largely orthogonal, dimensions of declassification.

Selective declassification (introduced in the DLM [11] and developed more abstractly by Pottier and Conchon [14]) is a more primitive "who" mechanism that imposes statically enforced access controls on declassification so only explicitly authorized information release can occur. Further extensions to the DLM have added capability mechanisms for controlling access to declassification and have integrated these mechanisms with public-key infrastructures [4, 17]. Banerjee and Naumann have also explored mediating information release with the Java access control mechanism [1].

Access controls are useful, but robustness offers additional assurance since it prevents the owner of information from authorizing information release that is not robust. Robustness not only prevents attackers from using declassification directly, it prevents them even from affecting declassification. Thus, robustness can be seen as an end-to-end extension of "who" mechanisms based on access control.

Zdancewic has earlier given typing rules intended to check robustness [18] (but not qualified robustness), for-

malizing rules used in work on secure program partitioning [20]. The rules differ from those given here because they do not take into account the label on the declassified value, and because they use a simpler integrity policy language. The checking rules in this paper are the first that provably enforce robustness for a rich policy language.

9 Conclusion

This work extends the semantic security condition of robustness to systems with mutually distrusting entities. Previous work on robustness considers attacks only by a single idealized attacker.

We use the decentralized label model to characterize the power of an arbitrary attacker, allowing the derivation of label constraints that ensure robustness against all possible attackers. We prove that a type system incorporating these label constraints enforces robustness against all attackers in the setting of a simple imperative language with explicit declassification. These constraints have been implemented in the Jif programming language, using sound approximations for the bounded but unknown labels that occur in Jif's type system.

We also extend qualified robustness, a security condition for systems that endorse information, and prove that a type system for a simple imperative language with explicit declassification and endorsement enforces qualified robustness against all attackers.

Robustness has been identified as a useful property for characterizing and enforcing security. This work shows how robustness applies to, and can be obtained in, systems with mutual distrust, providing new tools for reasoning about secure information flow and information release in systems with complex security requirements.

Acknowledgments. We thank Lantian Zheng and the anonymous reviewers for useful comments.

References

[1] A. Banerjee and D. A. Naumann. Using access control for secure information flow in a Java-like language. In *Proc. 16th IEEE Computer Security Foundations Workshop*, pages 155–169, June 2003.

[2] H. Chen and S. Chong. Owned policies for information security. In *Proc. 17th IEEE Computer Security Foundations Workshop*, June 2004.

[3] S. Chong and A. C. Myers. Language-based information erasure. In *Proc. 18th IEEE Computer Security Foundations Workshop*, pages 241–254, June 2005.

[4] T. Chothia, D. Duggan, and J. Vitek. Type-based distributed access control. In *Proc. 16th IEEE Computer Security Foundations Workshop*, pages 170–186, June 2003.

[5] E. S. Cohen. Information transmission in computational systems. *ACM SIGOPS Operating Systems Review*, 11(5):133–139, 1977.

[6] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Comm. of the ACM*, 20(7):504–513, July 1977.

[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. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification*. Addison Wesley, 2nd edition, 2000. ISBN 0-201-31008-2.

[9] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. In *Proc. 13th ACM Symp. on Operating System Principles (SOSP)*, pages 165–182, October 1991. *Operating System Review*, 253(5).

[10] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. 26th ACM Symp. on Principles of Programming Languages (POPL)*, pages 228–241, San Antonio, TX, Jan. 1999.

[11] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442, Oct. 2000.

[12] A. C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification. In *Proc. 17th IEEE Computer Security Foundations Workshop*, pages 172–186, June 2004.

[13] A. C. Myers, L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom. Jif: Java information flow. Software release, at http://www.cs.cornell.edu/jif, July 2001–.

[14] F. Pottier and S. Conchon. Information flow inference for free. In *Proc. 5nd ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 46–57, 2000.

[15] A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, Jan. 2003.

[16] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proc. 18th IEEE Computer Security Foundations Workshop*, pages 255–269, June 2005.

[17] S. Tse and S. Zdancewic. Run-time principals in information-flow type systems. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2004.

[18] S. Zdancewic. A type system for robust declassification. In *Proceedings of the Nineteenth Conference on the Mathematical Foundations of Programming Semantics*, Electronic Notes in Theoretical Computer Science, Mar. 2003.

[19] S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 15–23, June 2001.

[20] L. Zheng, S. Chong, A. C. Myers, and S. Zdancewic. Using replication and partitioning to build secure distributed systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 236–250, Oakland, California, May 2003.

[21] L. Zheng and A. C. Myers. Dynamic security labels and noninterference. In *Proc. 2nd Workshop on Formal Aspects in Security and Trust, IFIP TC1 WG1.7*. Springer, Aug. 2004.