It's My Privilege: Controlling Downgrading in DC-Labels [⋆]

Lucas Waye[1], Pablo Buiras[2], Dan King[1],
Stephen Chong[1], and Alejandro Russo[2⋆⋆]

[1] Harvard University, Cambridge MA, USA
`{lwaye,danking,chong}@seas.harvard.edu`
[2] Chalmers University of Technology, Gothenburg, Sweden
`{buiras,russo}@chalmers.se`

Abstract. Disjunction Category Labels (DC-labels) are an expressive label format used to classify the sensitivity of data in information-flow control systems. DC-labels use capability-like *privileges* to downgrade information. Inappropriate use of privileges can compromise security, but DC-labels provide no mechanism to ensure appropriate use. We extend DC-labels with the novel notions of *bounded privileges* and *robust privileges*. Bounded privileges specify and enforce upper and lower bounds on the labels of data that may be downgraded. Bounded privileges are simple and intuitive, yet can express a rich set of desirable security policies. Robust privileges can be used only in downgrading operations that are *robust*, i.e., the code exercising privileges cannot be abused to release or certify more information than intended. Surprisingly, robust downgrades can be expressed in DC-labels as downgrading operations using a weakened privilege. We provide *sound and complete* run-time security checks to ensure downgrading operations are robust. We illustrate the applicability of bounded and robust privileges in a case study as well as by identifying a vulnerability in an existing DC-label-based application.

1 Introduction

Information-flow control (IFC) systems track the flow of information by associating *labels* with data. Disjunction Category Labels (DC-labels) are a practical and expressive label format that can capture the security concerns of principals. IFC systems and DC-labels can provide strong, expressive, and practical information security guarantees, preventing exploitation of, for example, cross-site scripting and code injection vulnerabilities [9, 10, 19, 23, 26].

IFC systems often need to *downgrade* information: *declassification* downgrades confidentiality, and *endorsement* downgrades integrity. Downgrading of

[⋆] This work is supported in part by the National Science Foundation under Grants 1054172 and 1421770, DARPA CRASH under contract #N66001-10-2-4088, the Swedish research agencies VR and STINT, and the Barbro Osher Pro Suecia foundation.

[⋆⋆] Work done while visiting Stanford

DC-labels occurs via operations that require unforgeable capability-like tokens known as *privileges*. Unfortunately, DC-labels offer no methodology to protect developers from the *discretionary* (i.e., unrestricted) exercise of privileges—even a minor mistake in handling privileges can compromise the whole system's security. For example, we found a one-line vulnerability in an existing DC-label application written by experts that enabled confidential information to be inappropriately released, thus violating the application's intended security properties.

To address this, we introduce *restricted privileges*: privileges that are limited in their ability to declassify and endorse information. By declaratively restricting the use of privileges, developers can reason about the security properties of the system, regardless of the code that may possess or use the restricted privileges. Thus, the developer's local declaration of restrictions enables the enforcement of global information security guarantees.

We present two kinds of restricted privileges: *bounded privileges* and *robust privileges*. A bounded privilege imposes upper and lower bounds on the DC-labels of data that is declassified or endorsed using that privilege. Robust privileges avoid the accidental or malicious use of privileges to declassify or endorse more information than intended, achieving a property known as *robustness* [16, 25].

Bounded Privileges. A bounded privilege wraps an unrestricted privilege with two *immutable* labels that indicate upper and lower bounds for downgrading. DC-labels form a lattice structure (described in Section 2), and thus a bounded privilege restricts where in the lattice downgrading may occur. A bounded privilege also has a *mode*, indicating whether the bounded privilege may be used for declassification, endorsement, or both declassification and endorsement.

In terms of confidentiality, the upper bound limits the confidentiality of information that can be declassified using the privilege, and the lower bound limits the confidentiality of the information after declassification. For example, suppose principal `fb.com` passes a bounded privilege to `gogl.com`. If the lower bound of the bounded privilege is the label "`gogl.com`" then the privilege can be used to declassify information only from `fb.com` to `gogl.com`. Even if `gogl.com` passes the bounded privilege to another domain, say `evil.com`, the bounded privilege cannot be used to declassify information from `fb.com` to `evil.com`.

In terms of integrity, the upper bound of a bounded privilege indicates the least trustworthy level of information the privilege can be used to endorse, and the lower bound limits the integrity of the information after endorsement. For example, by setting the upper bound appropriately, `fb.com` can create a bounded privilege that can be used to endorse data only from `gogl.com`, and cannot be used to endorse other data, say from `evil.com`.

Robust Privileges. The security of a system might be at risk if an attacker is able to influence the decision to declassify or endorse information, or can influence what information is declassified. For example, consider a routine that receives a secret pair `(username,password)` and uses a privilege to declassify the first component of the pair. If an attacker (from another system component) can influence the pair to be `(password,username)` and trigger the declassification, the password will be leaked.

Robust declassification [25] and *qualified robustness* [16] are end-to-end semantic security guarantees that ensure that attackers are unable to inappropriately influence what information is revealed to them. These security conditions can be enforced by restricting declassification and endorsement operations. A robust privilege wraps a privilege and ensures that it is used only in declassification and endorsement operations that satisfy appropriate robustness checks.

This paper makes the following contributions: (i) We introduce bounded and robust privileges to limit the exercise of privileges for declassification and endorsement. (ii) We present a semantic characterization of how bounded privileges and robust privileges restrict declassification and endorsement operations. (iii) We define run-time security checks for bounded privileges and robust privileges that soundly and completely enforce the semantic characterization of restricted downgrading operations. The run-time checking for robust downgrading is effectively a weakening of the underlying unrestricted privilege: a surprisingly simple characterization of robustness. (iv) We illustrate the applicability of bounded and robust privileges via a case study. Moreover, use of restricted privileges identified a vulnerability in an existing DC-label-based application.

This paper is organized as follows. Section 2 introduces the DC-label model. Section 3 characterizes downgrading operations that use restricted privileges, and Section 4 provides the corresponding enforcement. Section 5 describes security properties in the presence of multiple restricted privileges. Case studies are given in Section 6. Section 7 examines related work and Section 8 concludes.

2 Background

We briefly define three concepts fundamental to our presentation: the DC-label model, privileges, and floating label systems.

Label Lattice DC-labels [21] are pairs of confidentiality and integrity policies. Confidentiality polices describe who may learn information. Integrity polices describe who takes responsibility or vouches for information. Both confidentiality and integrity policies are positive propositional formulas in conjunctive normal form, where propositional constants represent *principals*. Let CNF denote the set of all positive propositional formulas in conjunctive normal form; we use the term *formula* to range over CNF. We assume that operations on formulas always reduce their results to conjunctive normal form.

Both confidentiality policies and integrity policies form lattices—see Figures 1 and 2.

$$\begin{aligned} C_1 \sqsubseteq^{\mathrm{C}} C_2 &\iff C_2 \Rightarrow C_1 \\ C_1 \sqcup^{\mathrm{C}} C_2 &\iff C_1 \wedge C_2 \\ C_1 \sqcap^{\mathrm{C}} C_2 &\iff C_1 \vee C_2 \\ \bot^{\mathrm{C}} \equiv \mathit{True} & \qquad \top^{\mathrm{C}} \equiv \mathit{False} \end{aligned}$$

Fig. 1: Confidentiality Lattice

$$\begin{aligned} I_1 \sqsubseteq^{\mathrm{I}} I_2 &\iff I_1 \Rightarrow I_2 \\ I_1 \sqcup^{\mathrm{I}} I_2 &\iff I_1 \vee I_2 \\ I_1 \sqcap^{\mathrm{I}} I_2 &\iff I_1 \wedge I_2 \\ \bot^{\mathrm{I}} \equiv \mathit{False} & \qquad \top^{\mathrm{I}} \equiv \mathit{True} \end{aligned}$$

Fig. 2: Integrity Lattice

We interpret $C_1 \sqsubseteq^{\text{C}} C_2$ as: C_2 is at least as confidential as C_1. For instance, $\mathsf{Alice} \lor \mathsf{Bob} \sqsubseteq^{\text{C}} \mathsf{Alice}$, which means that data readable by either Alice or Bob is less confidential than data readable only by Alice. Conjunctions of principals represent the multiple interest of principals to protect the data. Conversely, disjunctions of principals represent groups wherein any member may learn the information. The integrity lattice is dually defined [3]; we interpret $I_1 \sqsubseteq^{\text{I}} I_2$ as: I_1 is at least as trustworthy as I_2. For example, $\mathsf{Alice} \land \mathsf{Bob} \sqsubseteq^{\text{I}} \mathsf{Alice}$, which indicates that data vouched for by $\mathsf{Alice} \land \mathsf{Bob}$ is more trustworthy than data vouched for only by Alice. In this case, conjunctions of principals represent groups whose members are independently responsible for the information. For example, data with integrity $\mathsf{Alice} \land \mathsf{Bob}$ means that Alice is completely responsible for the data, and so is Bob. Conversely, disjunctions of principals represent groups that collectively take responsibility for the information, however, no principal takes sole responsibility. For example, data with integrity $\mathsf{Alice} \lor \mathsf{Bob}$ means that Alice and Bob collectively are responsible for the data, i.e., both may have contributed to, or influenced the computation of the data.

Formally, a DC-label is a pair of a confidentiality policy C and an integrity policy I, written $\langle C, I \rangle$. DC-labels form a product lattice given in Figure 3. The $\sqsubseteq$ relation is called the *can-flow-to* relation because it describes information flows that respect confidentiality and integrity policies. We write $\text{C}(\cdot)$ and $\text{I}(\cdot)$ for the projection of confidentiality and integrity components, respectively.

$$
\begin{array}{rcl}
\langle C_1, I_1 \rangle \sqsubseteq \langle C_2, I_2 \rangle & \iff & C_1 \sqsubseteq^{\text{C}} C_2 \text{ and } I_1 \sqsubseteq^{\text{I}} I_2 \\
\langle C_1, I_1 \rangle \sqcup \langle C_2, I_2 \rangle & \equiv & \langle C_1 \sqcup^{\text{C}} C_2, I_1 \sqcup^{\text{I}} I_2 \rangle \\
\langle C_1, I_1 \rangle \sqcap \langle C_2, I_2 \rangle & \equiv & \langle C_1 \sqcap^{\text{C}} C_2, I_1 \sqcap^{\text{I}} I_2 \rangle \\
\text{C}(\langle C, I \rangle) \equiv C & & \text{I}(\langle C, I \rangle) \equiv I
\end{array}
$$

Fig. 3: Security lattice for DC-labels

Downgrading In the DC-label model, one security policy *downgrades* to another security policy if they do not satisfy the can-flow-to relation. Consider the pair of security labels in Figure 4. The first security label enforces the policy that data is vouched for by $\mathsf{Charlie}$. The second security label enforces the policy that data is vouched for by $\mathsf{Charlie}$ and Alice, therefore a secure system cannot permit data to flow from the sources protected by the first policy to sinks protected by the second policy. This downgrade is an *endorsement*, since it downgrades only integrity, i.e., it makes a value more trustworthy. Dually, a *declassification* downgrades only confidentiality, i.e., it makes a value less confidential. Consider the pair of security labels in Figure 5: The first security label enforces the policy that data is confidential to $\mathsf{Alice} \land \mathsf{Bob}$. The second security label enforces that data is confidential to Bob. Permitting data to flow from a source protected by the first policy to a sink protected by the second policy violates the confidentiality expectations of the source.

$$\langle \mathsf{Alice}, \mathsf{Charlie} \rangle \not\sqsubseteq \langle \mathsf{Alice}, \mathsf{Charlie} \land \mathsf{Alice} \rangle$$

Fig. 4: Downgrading integrity

$$\langle \mathsf{Alice} \land \mathsf{Bob}, \mathsf{Charlie} \rangle \not\sqsubseteq \langle \mathsf{Bob}, \mathsf{Charlie} \rangle$$

Fig. 5: Downgrading confidentiality

$$
\begin{aligned}
\langle C_1, I_1\rangle \sqsubseteq_p \langle C_2, I_2\rangle &\iff C_1 \sqsubseteq^{\mathrm{C}}_p C_2 \text{ and } I_1 \sqsubseteq^{\mathrm{I}}_p I_2 \\
\text{where}\quad C_1 \sqsubseteq^{\mathrm{C}}_p C_2 &\iff C_1 \sqsubseteq^{\mathrm{C}} C_2 \sqcup^{\mathrm{C}} p \\
I_1 \sqsubseteq^{\mathrm{I}}_p I_2 &\iff I_1 \sqcap^{\mathrm{I}} p \sqsubseteq^{\mathrm{I}} I_2
\end{aligned}
$$

Fig. 6: Relation can-flow-to-with-privilege-p

Privileges Practical systems must permit some downgrading. The DC-label model controls downgrading with *privileges*, where every principal has an associated privilege, and a principal's privilege enables downgrading. More precisely, given principal p, the *can-flow-to-with-privilege-p* relationship, written $\sqsubseteq_p$, describes the information flows permitted with p's privilege—see Figure 6. Observe that both downgrading examples from the previous section are now permitted by the can-flow-to-with-privilege relationship for the principal Alice, i.e., $\langle \mathsf{Alice}, \mathsf{Charlie}\rangle \sqsubseteq_{\mathsf{Alice}} \langle \mathsf{Alice}, \mathsf{Charlie} \wedge \mathsf{Alice}\rangle$ and $\langle \mathsf{Alice} \wedge \mathsf{Bob}, \mathsf{Charlie}\rangle \sqsubseteq_{\mathsf{Alice}} \langle \mathsf{Bob}, \mathsf{Charlie}\rangle$.

Floating label systems DC-labels are usually part of *floating label systems* like LIO [22], Hails [9], and COWL [23]. Such systems associate a *current label*, L_{pc}, with every computational task—this label plays a role similar to the *program counter* (PC) in more traditional language-based IFC approaches [19]. The current label denotes the fact that a computation depends only on data with labels bounded above by L_{pc}. When a task with current label L_{pc} observes information with label L_A, the current label after observation, L'_{pc}, must "float" above both the previous current label and the observed information's label, i.e., $L'_{pc} = L_{pc} \sqcup L_A$. Importantly, and to respect the security lattice, the current label restricts the subsequent writes to communication channels. Specifically, a task with current label L_{pc} is prevented from writing to channels protected by policy L_A if $L_{pc} \not\sqsubseteq L_A$.

Floating-label systems typically use some run-time representation of principals' privilege, and downgrading operations require the run-time representation of a principal p's privilege to be presented in order to use the can-flow-to-with-privilege-p relation, $\sqsubseteq_p$. Thus, the run-time representation of a principal's privilege acts like a capability to downgrade that principal's information. We write $p^{♛}$ for the run-time representation of the privilege of principal p, and refer to this value as a *raw privilege* (to contrast it with the restricted privileges that we introduce in this paper).

3 Security Definitions

If a system contains $p^{♛}$, then downgrading of data with policies involving p depends entirely on how $p^{♛}$ is used in the system. Reasoning about what downgrading occurs may require reasoning about global properties of the system. Indeed, we found a vulnerability in a Hails example application [9] of a web-based rock-paper-scissors game where use of a raw privilege was localized to one component, but arbitrary data could be passed to this component to be downgraded. This motivates our work to restrict privileges, and enable local reasoning about downgrading that may occur in a system.

A *restricted privilege* is a raw privilege "wrapped" with limitations on its use. These limitations enable sound reasoning about the downgrading that may be performed using the restricted privilege, even if arbitrary code uses the restricted privilege. Thus, local reasoning that ensures $p^{\text{♛}}$ is always appropriately restricted provides global guarantees about the downgrading that can occur with respect to policies involving p.

We present two kinds of restricted privileges, *bounded privileges* and *robust privileges*, which provide simple declarative limitations on the use of raw privileges.

Bounded Privileges A bounded privilege wraps a raw privilege with *downgrading bounds* and a downgrading mode. A downgrading bound is a pair of security lattice labels L_{high} and L_{low} that provide upper and lower bounds on downgrading, and the mode indicates whether the bounded privilege can be used to both declassify and endorse, only to declassify, or only to endorse.

Definition 1 (Downgrading bounds). *An operation that downgrades from security policy L_{from} to security policy L_{to} in a computational context with current label L_{pc}* satisfies downgrading bounds *L_{high} and L_{low} if and only if $(L_{from} \sqcup L_{pc}) \sqsubseteq L_{high}$ and $L_{low} \sqsubseteq (L_{to} \sqcup L_{pc})$*

Definition 2 (Bounded privileges). *A* bounded privilege with bounds *L_{high}* and *L_{low}* and mode *m* on privilege *$p^{\text{♛}}$, written ${}^{m}[p^{\text{♛}}]^{L_{high}}_{L_{low}}$, can be used only for downgrading operations with privilege $p^{\text{♛}}$ that satisfy downgrading bounds L_{high} and L_{low}. Mode m is one of* de*,* d*, or* e*. Declassification operations are permitted only if the mode is* de *or* d*; endorsement operations are permitted only if the mode is* de *or* e*.*

Figure 7 shows a visualization of bounded downgrading. The security lattice on the left is overlaid with a visualization of where bounded downgrading can occur (shaded) with respect to bounds L_{high} and L_{low}. The security lattice on the right shows an example of what labeled values can be declassified (shaded) with a bounded declassification privilege with bounds $L_{high} = \langle \mathsf{A} \wedge \mathsf{B}, \mathsf{A} \vee \mathsf{B} \rangle$ and $L_{low} = \langle \mathsf{A} \vee \mathsf{B}, \mathsf{A} \wedge \mathsf{B} \rangle$.

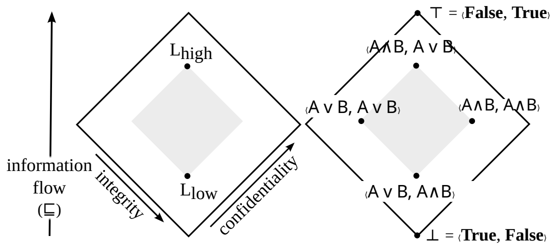

Fig. 7: Bounded Downgrading

In essence, the confidentiality lattice has collapsed $\mathrm{C}(L_{high})$ and $\mathrm{C}(L_{low})$ and all points in between: information that has confidentiality up to $\mathrm{C}(L_{high})$ may be declassified to confidentiality $\mathrm{C}(L_{low})$—all other points in the confidentiality lattice are not affected. Guarantees for endorsement with respect to bounded privileges are similar, but for integrity instead of confidentiality.

Example 1 (Policy: Only Bob controls Alice's privilege). Principal Alice allows Bob to declassify her data provided that Bob vouches for the data and the decision to declassify. In other words, information labeled with Alice can be declassified only after endorsement by Bob. This property can be captured by a bounded privilege with mode d and bounds: $L_{high} = \langle \top^{\mathrm{C}}, \mathsf{Bob}\rangle, L_{low} = \langle \bot^{\mathrm{C}}, \mathsf{Bob}\rangle$. If the privilege is used to declassify information that is not endorsed by Bob or in a context where the current label is not endorsed by Bob, then the declassification fails. In general, data must be vouched for by Bob (e.g., by using $\mathsf{Bob}^{♛}$ or another restricted privilege) before the bounded privilege for Alice can be used. For example, if a computational task has a current label $L_{pc} = \langle \mathsf{Alice}, \mathsf{Bob} \vee \mathsf{Charlie}\rangle$, the current label must be endorsed by Bob first. By endorsing the current label, Bob effectively vouches for any influence $\mathsf{Charlie}$ may have had on the computational task.

Example 2 (Policy: "A close source said..."). The bounded privilege ${}^{\mathsf{d}}[\mathsf{Alice}^{♛}]^{\langle \top^{\mathrm{C}}, \top^{\mathrm{I}}\rangle}_{\langle \bot^{\mathrm{C}}, \top^{\mathrm{I}}\rangle}$ requires that the integrity of data being declassified is $\top^{\mathrm{I}}$, i.e., data that no principal takes responsibility for. Alice may wish to impose this restriction on declassification involving data confidential to her to ensure that she has plausible deniability regarding the source of the data released. That is, the bounded privilege can not be used to declassify data for which Alice is explicitly responsible.

Robust Privileges *Robustness* [16, 25] is a semantic security condition that limits downgrading based on which principals might benefit from the downgrading, and which principals have influenced the data to downgrade and the decision to downgrade.

Consider a declassification of information from a source protected by label L_{from} to a sink protected by label L_{to}. A formula A (representing a principal or party of principals) will benefit from the declassification if A cannot read from the source, but can read the sink, i.e., $\mathrm{C}(L_{from}) \not\sqsubseteq^{\mathrm{C}} A$ and $\mathrm{C}(L_{to}) \sqsubseteq^{\mathrm{C}} A$. A robust declassification does not permit any principal that benefits from it to influence either the decision to declassify or the data to declassify. A influences the decision to declassify if $A \sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{pc})$, and A influences the data to declassify if $A \sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{from})$.

Definition 3 (Robust declassification). *A robust declassification using privilege $p^{♛}$ from a source protected by L_{from} to a sink protected by L_{to}, in a computational context with current label L_{pc} is a declassification (i.e., $\mathrm{C}(L_{from}) \sqsubseteq^{\mathrm{C}}_{p} \mathrm{C}(L_{to})$) where $\forall A \in \mathit{CNF}.\, \mathrm{C}(L_{to}) \sqsubseteq^{\mathrm{C}} A \wedge \mathrm{C}(L_{from}) \not\sqsubseteq^{\mathrm{C}} A \Rightarrow A \not\sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{pc}) \wedge A \not\sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{from})$.*

For endorsement, a principal benifits if it may be held responsible for information from the source but is not held responsible for information from the sink. In other words, A benefits from an endorsement if A gets absolved of responsibility for a value, i.e., $A \sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{from}) \wedge A \not\sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{to})$. Robust endorsement does not permit principals that benefit from it to influence the decision to endorse.

Definition 4 (Robust endorsement). *A* robust endorsement *using privilege $p^{♛}$ from a source protected by L_{from} to a sink protected by L_{to}, in a computational context with current label L_{pc} is an endorsement (i.e., $\text{I}(L_{from}) \sqsubseteq^{\text{I}}_{p} \text{I}(L_{to})$) where $\forall A \in CNF. A \sqsubseteq^{\text{I}} \text{I}(L_{from}) \wedge A \not\sqsubseteq^{\text{I}} \text{I}(L_{to}) \Rightarrow A \not\sqsubseteq^{\text{I}} \text{I}(L_{pc})$.*

A *robust privilege* is a privilege that can only be used for robust downgrading operations.

Definition 5 (Robust privilege). *A* robust privilege with mode *m* on privilege *$p^{♛}$, written $rbst^{m}\{p^{♛}\}$, restricts downgrading operations where it is used to those that are robust for $p^{♛}$. Mode m is one of* de*,* d*, or* e*. Declassification operations are permitted only if the mode is* de *or* d*; endorsement operations are permitted only if the mode is* de *or* e*.*

The definitions of robust declassification and endorsements both quantify over all formulas A in the (possibly infinite) set CNF. In Section 4, we consider how to implement efficient checks that do not use universal quantification.

Figure 8 shows a visualization of where robust declassification is allowed for a given robust privilege. The security lattice on the left is overlaid with a visualization of where a value with label L_{from} can be declassified to (shaded line) using a robust declassification privilege. (Note that the current label L_{pc} is not included in the diagram for brevity.) I represents the boolean formula for the integrity of the labeled value. L_{low} is one of the lowest points where L_{from} can be declassified to while still being a robust declassification, i.e., $L_{low} \sqsubseteq L_{to}$. That is, the integrity of the label of the value for declassification (together with the integrity of the current label of the process) is used as a lower bound for declassification. Intuitively, those who influence a declassification should not learn from it. In the right hand side of Figure 8, the shaded line indicates to where a robust privilege may declassify the labeled value $\langle \mathsf{A} \wedge \mathsf{B}, \mathsf{A} \rangle$. The declassification is robust if A is not able to learn from the declassification. As a result, the value could not be declassified to $\langle \mathsf{A} \vee \mathsf{B}, \mathsf{A} \rangle$ as A would learn from a declassification that it influenced. In contrast, it is robust to declassify it to $\langle \mathsf{B}, \mathsf{A} \rangle$.

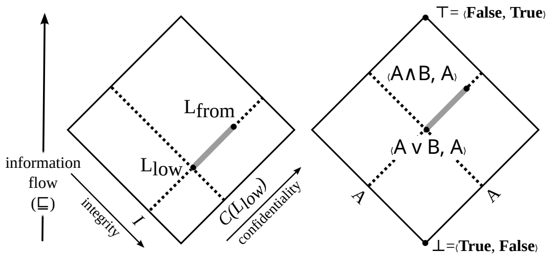

Fig. 8: Robust Declassification

4 Enforcement for robust privileges

In this section we describe enforcement mechanisms for restricted privileges that satisfy their semantic characterizations described in Section 3. We have implemented these mechanisms in LIO and use them in our case study (see Section 6).

When a bounded privilege (Definition 2) is used at run time, it is simple to check that the downgrading operation satisfies the appropriate bounds, since the labels relevant to the downgrading (L_{from}, L_{to}, and L_{pc}) are all available at run time, and the label ordering relation can be easily checked dynamically.

Robust privileges (Definition 5) impose restrictions on downgrading operations which quantify over formulas A. However, attempting to explicitly check each possible formula A at run time is not feasible. We can however, derive simple and efficient run-time checks that are sound and complete with respect to their semantic characterizations. These checks are inspired by Chong and Meyers [6], who provide run-time checks for robustness that are sound but not complete.

The following theorem shows that the semantic characterization of robust declassification (Definition 3) is equivalent to two confidentiality-policy comparisons involving only L_{from}, L_{to}, and L_{pc}.

Theorem 1 (Robust declassification check). *A declassification using privilege $p^{\text{♛}}$ from a source protected by L_{from} to a sink protected by L_{to} in a computational context with current label L_{pc} is robust if and only if* $\mathrm{C}(L_{from}) \sqsubseteq_p^{\mathrm{C}} \mathrm{C}(L_{to})$, $\mathrm{C}(L_{from}) \sqsubseteq^{\mathrm{C}} \mathrm{C}(L_{to}) \sqcup^{\mathrm{C}} \mathrm{I}(L_{pc})$, *and* $\mathrm{C}(L_{from}) \sqsubseteq^{\mathrm{C}} \mathrm{C}(L_{to}) \sqcup^{\mathrm{C}} \mathrm{I}(L_{from})$.

The run-time check ensures that if there is any formula A that benefits from the declassification ($\mathrm{C}(L_{from}) \not\sqsubseteq^{\mathrm{C}} A$ and $\mathrm{C}(L_{to}) \sqsubseteq^{\mathrm{C}} A$) then $A \not\sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{pc})$ (or, equivalently, $\mathrm{I}(L_{pc}) \not\sqsubseteq^{\mathrm{C}} A$), and similarly that $A \not\sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{from})$. Thus, the run-time check converts a comparison of integrity policies to a comparison of integrity policies that does not involve A.

The next theorem describes a simple run-time check for robust endorsement.

Theorem 2 (Robust endorsement check). *An endorsement using privilege $p^{\text{♛}}$ from a source protected by L_{from} to a sink protected by L_{to} in a computational context with current label L_{pc} is robust (Definition 4) if and only if* $\mathrm{I}(L_{from}) \sqsubseteq_p^{\mathrm{I}} \mathrm{I}(L_{to})$, *and* $\mathrm{I}(L_{pc}) \sqcap^{\mathrm{I}} \mathrm{I}(L_{from}) \sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{to})$.

The run-time check that all formulas A that may be responsible for either the current label ($A \sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{pc})$) or the data itself ($A \sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{from})$) should also be responsible for the data after endorsement ($A \sqsubseteq^{\mathrm{I}} \mathrm{I}(L_{to})$). Proofs of Theorems 1 and 2 are omitted due to space limitations.

Alternative formulation In DC-labels, privileges can be arbitrary formulas, which can be stronger or weaker than privileges for individual principals. For example, a privilege for $\mathsf{A} \wedge \mathsf{B}$ can downgrade more information than a privilege for A or B alone, whereas a privilege for $\mathsf{A} \vee \mathsf{B}$ can downgrade less information than a privilege for A or B alone. Leveraging this feature, we show how robust downgrading can be seen (and enforced) as normal downgrading operations that use a weakened privilege. That is, the privilege used in a downgrading operation is weakened so as to permit all and only robust downgrading operations.

The next corollaries follow from Theorems 1 and 2 and the definition for the *can-flow-to-with-privilege-p* relation.

Corollary 1. *A declassification using raw privilege $p^{♛}$ from a source protected by L_{from} to a sink protected by L_{to} in a computational context with current label L_{pc} is robust (Definition 3) if and only if* $\mathrm{C}(L_{from}) \sqsubseteq^{\mathrm{C}}_{p \,\vee\, \mathrm{I}(L_{from}) \,\vee\, \mathrm{I}(L_{pc})} \mathrm{C}(L_{to})$.

This indicates that robust declassification can be achieved by simply weakening privilege $p^{♛}$ with the integrity labels of the current label and the data to be released, i.e., $p \vee \mathrm{I}(L_{from}) \vee \mathrm{I}(L_{pc})$. Robust endorsement has a similar corollary.

Corollary 2. *An endorsement using raw privilege $p^{♛}$ from a source protected by L_{from} to a sink protected by L_{to} in a computational context with current label L_{pc} is robust (Definition 3) if and only if* $\mathrm{I}(L_{from}) \sqsubseteq^{\mathrm{I}}_{p \,\vee\, \mathrm{I}(L_{pc})} \mathrm{I}(L_{to})$.

The proof of Corollary 1 is omitted due to space limitations; the proof of Corollary 2 is similar.

The current implementation of DC-labels [21] provides the ability to infer appropriate L_{to} labels of downgrading operations given a privilege p. By expressing the runtime checks for robust downgrading operations as a standard downgrading operation with a weakened privilege, we can take advantage of this feature and automatically infer a suitable L_{to} label if one exists. This reduces the burden on the programmer.

5 Interaction among restricted privileges

We can extend restricted privileges to allow them to be composed, i.e., by allowing bounded privileges and robust privileges to wrap around other restricted privileges, as well as raw privileges. The guarantee provided by the composition of restricted privileges is the intersection of their individual guarantees. For example, a bounded privilege composed with another bounded privilege will require that downgrading operations satisfy the bounds of both privileges. A bounded privilege composed with a robust privilege (and vice-versa) requires the downgrading both to be robust and satisfy the downgrading bounds. Robust privileges are idempotent: a robust privilege composed with a robust privilege will simply require all downgrade operations to be robust.

Privileges might also interact because a system has multiple privileges available. Unlike composed privileges (which further restrict possible information flows), multiple privileges enable additional information flows. In the remainder of the section, we discuss the guarantees that result from the use of multiple restricted privileges. In the accompanying figures, bounded privileges are depicted as a shaded rectangle corresponding to their bounds. Robust declassification privileges are depicted as a pair of dashed lines: one line represents the integrity of the source and the other line represents the lower bound to which data may be declassified. Labels are depicted as points along with their names.

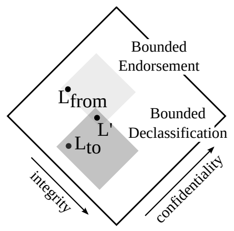

Fig. 9: Multiple bounds.

Bounded declassification and bounded endorsement Figure 9 depicts two bounded privileges, one for declassification and one for endorsement, as well as a label, L_{from} that is outside the bounds of the declassification privilege. Because the bounds of the privileges overlap, data can transitively flow from L_{from} to L_{to}. The endorsement privilege enables data from L_{from} to be endorsed to L'. The bounded declassification privilege can then declassify data from L' to L_{to}.

Bounded declassification and robust declassification Figure 10 depicts two declassification privileges, one robust and one bounded, and a label that is outside the bounds of the bounded declassification privilege. Neither privilege alone permits a flow from L_{from} to L_{to}. However, when used together, the robust declassification privilege permits declassification of data from L_{from} to L' and the bounded declassification permits a flow from L' to L_{to}, completing a flow from L_{from} to L_{to}.

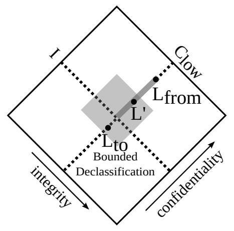

Fig. 10: Bounded and robust declassification.

Endorsement and robust declassification In a system with unrestricted endorsement, robust declassification provides almost no protection against attackers influencing what they learn. Intuitively, the endorsement of data by p can make the data trustworthy enough to make a subsequent declassification robust. Consider a declassification of a value from label $L_{from} = \langle \mathsf{A} \wedge \mathsf{B}, \mathsf{A} \rangle$ to $L = \langle \mathsf{A}, \mathsf{A} \rangle$ using the robust privilege $rbst^{\mathsf{d}}\{\mathsf{B}^{♛}\}$. This declassification is not robust: principal A, who benefits from this declassification, may be held responsible for the value, i.e., A may have decided what gets declassified. However, an unrestricted endorsement privilege $\mathsf{B}^{♛}$ could be used to endorse the value—effectively endorsing any possible influence by A. In other words, $\langle \mathsf{A} \wedge \mathsf{B}, \mathsf{A} \rangle$ can be endorsed to $\langle \mathsf{A} \wedge \mathsf{B}, \mathsf{B} \rangle$, and a subsequent declassification from $\langle \mathsf{A} \wedge \mathsf{B}, \mathsf{B} \rangle$ to $\langle \mathsf{A}, \mathsf{B} \rangle$ is robust.

Bounded endorsement effectively limits the aforementioned deletrious effects of unrestricted endorsement to the bounded area of the lattice, Figure 11 depicts this situation. Besides mitigating the effects of unrestricted endorsement, bounded endorsement is useful to relax robust declassification so that it succeeds for principals collaborating in achieving a common goal—see, for example, Section 6.

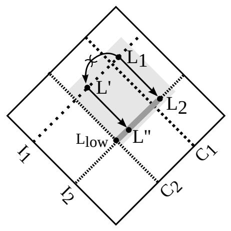

Fig. 11: Bounded endorsement and robust declassification.

Bounded and robust declassification Figure 10 shows the guarantees when a robust declassification-only privilege (i.e., $rbst^{\mathsf{d}}\{p^{♛}\}$) and a bounded declassification-only privilege (i.e., ${}^{\mathsf{d}}[p^{♛}]^{L_{high}}_{L_{low}}$) for the same principal are both available in the system. Intuitively, p's information can be declassified from $L_{from}$to L' using the robust privilege. The information can then be declassified again to L_{to} using the bounded privilege, even though L_{from} is below the threshold imposed by robust declassification (i.e., the lowest possible label that robust declassification could declassify label L_{from}). Thus, the presence of a bounded declassification-only privilege can bypass the guarantees provided by robust declassification.

Several bounded privileges Multiple robust privileges for the same principal do not add any additional complexity, as all robust privileges are equivalent (up to their modes). Bounded privileges, however, may differ on the bounds they impose. The presence of multiple bounded privileges in a system for principal p collapses the label lattice for principal p in complex ways. For instance, the left diagram of Figure 9 illustrates an example where there is a bounded endorsement-only privilege and a bounded declassification-only privilege with different bounds. It may be possible for a value labeled L_{from} to be relabeled to L_{to} via an endorsement to L' followed by a declassification. Thus, labels between L_{from} and L' and between L' and L_{to} are effectively collapsed, since the bounded privileges allow a value with any of these labels to be relabeled to any other of these labels. More generally, as more overlapping bounded privileges exist for a given principal, data can be downgraded in more possible ways.

6 Case studies

6.1 Calendar Case Study

We have extended LIO [22] with support for bounded privileges and robust privileges, and used them to develop a Calendar application to explore and illustrate the utility of restricted privileges. The application allows users to view their appointments, and schedule appointments with each other. DC-label principals are the calendar users. A user's appointments are confidential to that user.

We consider a setting where principals belong to groups and a principal is willing to disclose her availability to all and only members of her groups. For example, if Bob wants to schedule an appointment with Alice at time t, the application will check Alice's calendar and inform Bob whether Alice is available at that time. This operation, which declassifies Alice's availability at time t to Bob, should succeed only if Alice and Bob are in the same group.

Each user A has a robust declassification privilege $rbst^{\mathsf{d}}\{\mathsf{A}^{♕}\}$, and, for each group G that A belongs to, a bounded endorsement privilege ${}^{\mathsf{e}}[\mathsf{A}^{♕}]^{\langle \top^{\mathrm{C}}, G\rangle}_{\langle \bot^{\mathrm{C}}, \bot^{\mathrm{I}}\rangle}$, where G is the disjunction of all users in the group. These are the only privileges available in the system for user A, and thus all endorsements must be bounded appropriately, and all declassifications must be robust.

Joint scheduling between A and B works as follows:

1. User B sends a scheduling request for time t labeled $\langle \mathsf{B}, \mathsf{B}\rangle$ to user A.
2. User A computes her availability for time t. Because the context that computes the availability reads data labeled $\langle \mathsf{A}, \mathsf{A}\rangle$ and $\langle \mathsf{B}, \mathsf{A}\rangle$, the label of the availability result is $\langle \mathsf{A} \wedge \mathsf{B}, \mathsf{A} \vee \mathsf{B}\rangle$.

3. If A and B are both in some group G, then A uses her bounded privilege to endorse the availability result to $\langle \mathsf{A} \wedge \mathsf{B}, \mathsf{A} \rangle$, since she is prepared to take sole responsibility for the availability result. Since both A and B are in the same group, the endorsement satisfies the bounds (i.e., $\mathsf{A} \vee \mathsf{B} \sqsubseteq^{\mathrm{I}} G$). If there is no group for which both A and B are members, then A has no bounded endorsement privilege for which the bounds will be satisfied.
4. User A uses her robust privilege to declassify the availability result to $\langle \mathsf{B}, \mathsf{A} \rangle$. The declassification is robust.
5. User A sends the declassified value to B.

Because all downgrading in the system relevant to user A must use A's restricted privileges, we obtain strong system-wide guarantees, even if A's restricted privileges manage to escape from the scheduling component, and even if if B sends malicious scheduling requests. Section 5 (Figure 11) discusses in more detail the system-wide guarantees that hold when both a bounded endorsement privilege and a robust declassification privilege are available.

6.2 Restricted Privileges in Existing Applications

Using our restricted privileges, we found a security vulnerability in an application written using Haskell Automatic Information Labeling System (Hails) [9]. Hails is a web framework built on LIO that extends the traditional Model-View-Controller paradigm to Model-Policy-View-Controller. The policy module specifies all models and describes the labels for data fetched from the database. When data is stored in the database, Hails checks labels against the policy module to ensure appropriate data integrity. The policy module has access to a privilege that can declassify all models. As a design pattern, policy modules export functions that perform declassification for untrusted applications using the privilege; untrusted applications never have direct access to the privilege.

Rock-Paper-Scissors[3] is a Hails application that contains a security vulnerability due to misuse of the policy privilege, despite being written by security experts who developed Hails.

The policy module includes a function to get the outcome of a match given a particular move by a player. This function can be exploited to reveal the opponent's move before the player has actually committed to a move by submitting it to the database. As a result, a player can always win a match by exploiting this function to determine which move will win, and then committing to that winning move. When we replaced the policy module's raw privilege with a robust privilege, the robust declassification check signalled a potential security vulnerability. To fix the vulnerability, we added code that checks whether a player had committed to a move (i.e., the move is in the database), and, if so, endorses the submitted move. This endorsement allows the robust declassification check to succeed. Endorsing only when the player has committed to his move fixes the security vulnerability.

[3] `https://github.com/scslab/hails/tree/master/examples/hails-rock`

7 Related Work

Declassification can be characterized into different dimensions: *who*, *what*, *where*, and *when* [20]. Our work can be considering as restricting *where* in the security lattice downgrading may occur (bounded downgrading) and *who* may influence downgrading (robustness). Almeida Matos and Boudol [1] introduce a construct **flow** $p \prec q$ **in** c to indicate *where* additional information flows are allowed within a lexical scope. Intransitive noninterference [11, 12, 18] posits a non-transitive information flow ordering which describes *what* downgrading operations are permitted. Mantel and Sands [11] combine intransitive noninterference with language techniques that use declassification annotations to explicitly identify non-transitive information flows. In our bounded declassification mechanism, violating the normal ordering of security levels is tied to a runtime value, and not lexically scoped or marked by annotations.

In Jif [13], declassifications may explicitly state where in the security lattice the declassification occurs. By contrast, our bounded mechanisms declare this restriction on the run-time value that authorizes downgrading. Jif uses a form of access control to restrict which code may downgrade information, coined *selective declassification* by Pottier and Conchon [17]. Specifically, a downgrading operation that may compromise the security of principal p may only occur in code that has been (statically or dynamically) authorized by p. Similarly, the authority to declassify or endorse information in Asbestos [7], HiStar [26], Flume [10], and COWL [23] must come from the creator of the exercised privileges. By contrast, LIO associates the authority to declassify or endorse a principal's information with a run-time value. This capability-like approach to authorizing downgrading enables our local declarative approach to restrict downgrading. Birgisson et al. [4] use capabilities to restrict the ability to read and write memory locations, but do not consider the use of capabilities to restrict downgrading.

Zdancewic and Myers [25] introduce the semantic security condition of *robust declassification*, and Myers et al. [16] enforce robust declassification with a security type system [19, 24], and introduce *qualified robustness*, which extends the concept to reason about endorsement. Askarov and Myers [2] subsequently present a semantic framework for downgrading, and present a crisper version of qualified robustness. Chong and Myers [6] extend the notion of robust declassification to the Decentralized Label Model [14, 15]. The run-time checks used in this work to enforce robustness are analogous to the run-time checks Chong and Myers introduce for the DLM. In other work, Chong and Myers [5] note that the semantic security condition for robust declassification applies to information flow of confidential information generally, including, for example, information erasure, and is more general than just declassification. If the only privilege for p available in the system is a robust privilege with mode mode d then the system will be robust for p. If the privilege for that mode is de (i.e., robust declassification operations and robust endorsement operations are possible), then the end-to-end security guarantee is *qualified robustness* [2, 16]. A system satisfies qualified robustness if the only way an attacker can influence what information is released to it is via robust endorsement operations.

Foley et al. incorporate bounds constraints on a system with relabeling operations on objects [8]. Our model performs relabeling based on the use of capability-like tokens rather than with respect to a particular subject. Bound restrictions can be placed per privilege rather than on all relabeling operations, so the guarantees of this work are more dependent on what sorts of privileges are available for use, but do not require changes to the trusted computing base.

The system HiStar [26] provides the notion of gates: entities designed to encapsulate privileges so that processes can safely switch their current label by exercising them through the gate. Gates have a clearance component which imposes an upper bound on the label that results from using it. Gates can be leveraged to restrict the use of privileges similar to upper bounds in bounded privileges. Similar to our approach, Flume [10] distinguishes privileges used for declassification (symbol $-$) and endorsement (symbol $+$).

8 Conclusion

Restricted privileges are a new mechanism to control declassification and endorsement in DC-labels that is simple and intuitive yet expresses a rich set of desirable policies. Bounded privileges impose upper and lower bounds on data that is declassified or endorsed. Robust privileges help prevent the accidental or malicious exercise of privileges to downgrade more information than intended, and can provide the end-to-end security guarantees of robustness and qualified robustness. We provide sound and complete efficient security checks for downgrading using restricted privileges. We note that robust downgrading operations can be viewed as privileged downgrading with a weakened privilege. We explore the guarantees provided by combining the use of bounded and robust privileges as well as their composition in a case study. This work establishes a basis for better design of IFC systems that use privileges for downgrading information.

References

1. Almeida Matos, A., Boudol, G.: On declassification and the non-disclosure policy. In: Proc. 18th IEEE Computer Security Foundations Workshop. pp. 226–240 (2005)
2. Askarov, A., Myers, A.: A semantic framework for declassification and endorsement. In: Proc. 19th European Symposium on Programming (2010)
3. Biba, K.J.: Integrity considerations for secure computer systems. ESD-TR-76-372 (1977)
4. Birgisson, A., Russo, A., Sabelfeld, A.: Capabilities for information flow. In: Proc. 6th Workshop on Programming Languages and Analysis for Security (2011)
5. Chong, S., Myers, A.C.: Language-based information erasure. In: Proc. 18th IEEE Computer Security Foundations Workshop. pp. 241–254 (Jun 2005)
6. Chong, S., Myers, A.C.: Decentralized robustness. In: Proc. 19th IEEE Workshop on Computer Security Foundations. pp. 242–256 (2006)
7. Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazières, D., Kaashoek, F., Morris, R.: Labels and event processes in the Asbestos operating system. In: Proc. 20th ACM Symposium on Operating Systems Principles (2005)

8. Foley, S., Gong, L., Qian, X.: A security model of dynamic labeling providing a tiered approach to verification. In: Proc. 1996 IEEE Symposium on Security and Privacy. pp. 142–158 (1996)
9. Giffin, D.B., Levy, A., Stefan, D., Terei, D., Mazières, D., Mitchell, J., Russo, A.: Hails: Protecting data privacy in untrusted web applications. In: Proc. Symposium on Operating Systems Design and Implementation (2012)
10. Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M.F., Kohler, E., Morris, R.: Information flow control for standard OS abstractions. In: Proc. 21st Symp. on Operating Systems Principles (October 2007)
11. Mantel, H., Sands, D.: Controlled Declassification based on Intransitive Noninterference. In: Proc. 2nd Asian Symposium on Programming Languages and Systems. vol. 3303, pp. 129–145 (Nov 2004)
12. van der Meyden, R.: What, indeed, is intransitive noninterference? In: Proc. 12th European Symposium On Research In Computer Security. vol. 4734, pp. 235–250 (Sep 2007)
13. Myers, A.C., Zheng, L., Zdancewic, S., Chong, S., Nystrom, N.: Jif: Java Information Flow (2001–), software release. `http://www.cs.cornell.edu/jif`
14. Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: Proc. 16th ACM Symposium on Operating System Principles. pp. 129–142. New York, NY, USA (1997)
15. Myers, A.C., Liskov, B.: Complete, safe information flow with decentralized labels. In: Proc. IEEE Symposium on Security and Privacy. pp. 186–197 (May 1998)
16. Myers, A.C., Sabelfeld, A., Zdancewic, S.: Enforcing robust declassification and qualified robustness. Journal of Computer Security 14(2), 157–196 (Apr 2006)
17. Pottier, F., Conchon, S.: Information flow inference for free. In: Proc. 5th ACM SIGPLAN International Conference on Functional Programming. pp. 46–57. New York, NY, USA (2000)
18. Roscoe, A.W., Goldsmith, M.H.: What is intransitive noninterference? In: Proc. 12th IEEE Computer Security Foundations Workshop (1999)
19. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (Jan 2003)
20. Sabelfeld, A., Sands, D.: Dimensions and principles of declassification. In: Proc. 18th IEEE Computer Security Foundations Workshop. pp. 255–269 (Jun 2005)
21. Stefan, D., Russo, A., Mazières, D., Mitchell, J.C.: Disjunction category labels. In: 16th Nordic Conference on Security IT Systems. vol. 7161, pp. 223–239 (Oct 2011)
22. Stefan, D., Russo, A., Mitchell, J.C., Mazières, D.: Flexible Dynamic Information Flow Control in Haskell. In: Proc. 4th ACM symposium on Haskell. pp. 95–106. New York, NY, USA (2011)
23. Stefan, D., Yang, E.Z., Marchenko, P., Russo, A., Herman, D., Karp, B., Mazières, D.: Protecting users by confining JavaScript with COWL. In: Proc. 11th Symposium on Operating Systems Design and Implementation (Oct 2014)
24. Volpano, D., Smith, G., Irvine, C.: A sound type system for secure flow analysis. Journal of Computer Security 4(3), 167–187 (1996)
25. Zdancewic, S., Myers, A.C.: Robust declassification. In: Proc. 14th IEEE Computer Security Foundations Workshop. pp. 15–23 (Jun 2001)
26. Zeldovich, N., Boyd-Wickizer, S., Kohler, E., Mazières, D.: Making information flow explicit in HiStar. In: Proc. 7th Symposium on Operating Systems Design and Implementation. pp. 263–278 (2006)