

Enforcing Robust Declassification

Authors: A. Myers, A. Sabelfeld, S. Zdancewic

Presented by Dilsun Kaynar

November 12, 2007, Languages and Logics for Security, CMU

Motivation

- Real systems leak some information as a way of their correct functioning. Some declassification mechanism is useful to have.
 - Then, an attacker can use this mechanism to leak more information than intended.
 - **Robust declassification:** a system may release information but gives attackers no control over whether information is released or what is released
-

Contributions

- Formalization of robustness in a language-based setting
 - A relaxed notion of robustness (qualified robustness) that gives untrusted code and data limited ability to affect information release
 - Enforcement of these robustness notions through a type system
-

Robustness

- ❑ Attackers should not be able to affect what information is released and whether information is released.
 - ❑ Regardless of what attack is launched, the same information should be released.
 - ❑ A passive attacker who merely observes execution learns no more than an active attacker who can both change execution and observe it.
-

Motivating example

**if lowX then lowY := declassify(highZ)
else skip**

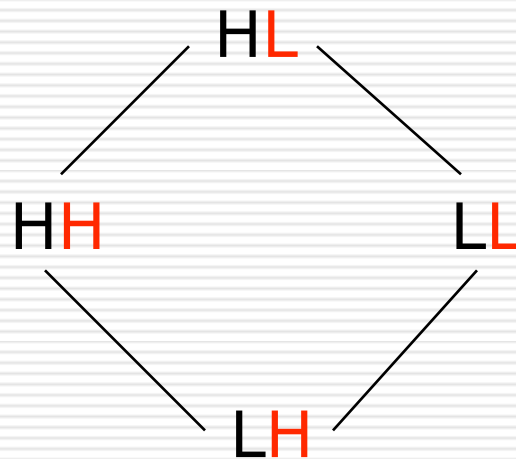
Motivating example

```
if lowX then lowY := declassify(highZ)
else skip
```

- Robustness is contingent on the power an attacker has to affect the execution of the system.
 - Attach integrity labels to manipulated information.
 - Low-integrity variables may be changed by the attacker
Low-integrity code may be replaced by the attacker with different code.
-

Security lattice

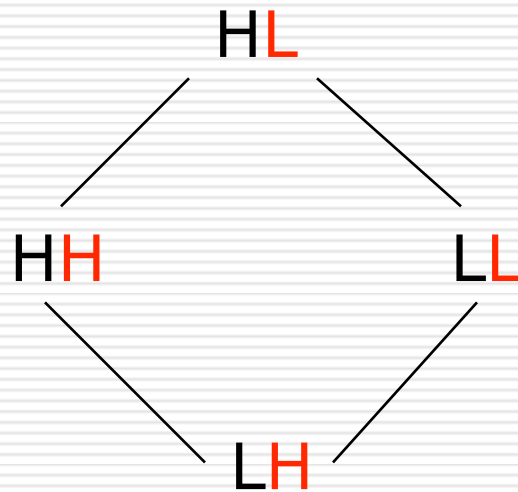
Product of confidentiality and integrity lattices:



Usage of data at the top of the lattice is most restricted
data at the bottom may be used arbitrarily

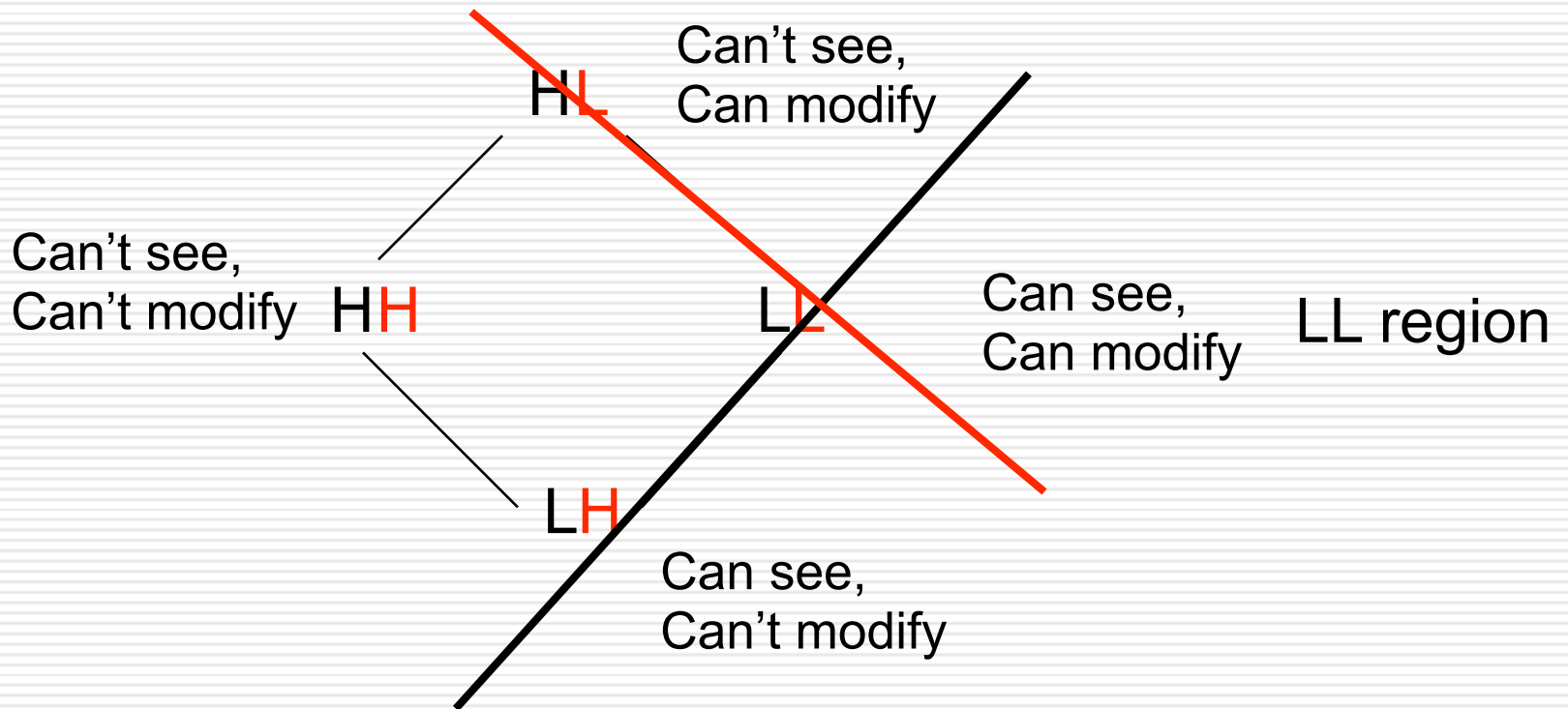
Attacker model

The power of an attacker is described by a lattice element



Attacker model

The power of an attacker is described by a lattice element



Language syntax

$e ::= \text{val} \mid v \mid e1 \text{ op } e2 \mid \text{declassify}(e, l)$

$c ::= \text{skip} \mid v := e \mid c1; c2$
 $\quad \mid \text{if } e \text{ then } c1 \text{ else } c2$
 $\quad \mid \text{while } e \text{ do } c$

l ranges over security levels

Security policy

- Security lattice L
- Security environment $\Gamma : \text{Var} \rightarrow L$
- A flow from variable v_1 to v_2 is allowed only if $\Gamma(v_1) \leq \Gamma(v_2)$
 - For example, from $v_1: LH$ to $v_2: LL$

Operational semantics

- Standard small step semantics
- Assignments annotate labels with variable names
- $\text{declassify}(e, l)$ has the same operational meaning as e
- Assume expression evaluation always terminates, while commands may diverge

Indistinguishability notions

Indistinguishability notions

□ Trace: $Tr(\langle M, c \rangle) = [M, M', M'', \dots]$

Indistinguishability notions

- Trace: $Tr(\langle M, c \rangle) = [M, M', M'', \dots]$
- Two memories $M1$ and $M2$ are **l -indistinguishable**, if for all v . $\Gamma(v) \leq l$ implies $M1(v) = M2(v)$.

Indistinguishability notions

- Trace: $Tr(\langle M, c \rangle) = [M, M', M'', \dots]$
 - Two memories $M1$ and $M2$ are **l-indistinguishable**, if for all v . $\Gamma(v) \leq l$ implies $M1(v) = M2(v)$.
 - $\langle M1, c1 \rangle$ and $\langle M2, c2 \rangle$ are **weakly indistinguishable upto l**, if whenever both traces terminate then
 - $M1$ and $M2$ are l-indistinguishable
 - The subsequences of memories resulting from l-observable assignments in $c1$ and $c2$ are l-indistinguishable.
 - $\langle M1, c1 \rangle$ and $\langle M2, c2 \rangle$ are **strongly indistinguishable upto l**, if
 - Both traces terminate
 - $\langle M1, c1 \rangle$ and $\langle M2, c2 \rangle$ are **weakly indistinguishable upto l**.
-

Noninterference

- A command satisfies noninterference under Γ if
 - for all $l, M1, M2$.
if $M1$ and $M2$ are l -indistinguishable
then $\langle M1, c \rangle$ and $\langle M2, c \rangle$ are **weakly indistinguishable upto l** .

Fair attacks

- A command is a fair attack at some level l in LL region, if it is formed with respect to the following grammar:

$a ::= \text{skip}$

| $v := e$ for all variables x in e , $\Gamma(x) = l = \Gamma(v)$

| $a1; a2$

| **if** b **then** $a1$ **else** $a2$ for all variables x in b , $\Gamma(x) = l$

| **while** b **do** a for all variables x in e , $\Gamma(x) = l$

- No declassify
-

High-integrity contexts

- Attacker-controlled low-integrity computation may be interspersed with high-integrity code.

- High-integrity contexts
 - $c[\bullet]^+ := \dots \mid [\bullet]$

Formalizing robustness

- Command $c[\bullet]^+$ has robustness with respect to fair attacks at level A if
 - For all $M1, M2, seq(a), seq(a')$.
If $\langle M1, c[seq(a)] \rangle$ and $\langle M2, c[seq(a)] \rangle$ are **strongly indistinguishable upto** $(C(A), Top)$ then
 $\langle M1, c[seq(a')] \rangle$ and $\langle M2, c[seq(a')] \rangle$ are **weakly indistinguishable upto** $(C(A), Top)$.

A non-robust context

□ $x_{LL} := 1; [\bullet];$ **while** $x_{LL} > 0$ **do** skip;
if $x_{LL} = 0$ **then** $y_{LH} := \text{declassify}(z_{HH}, LH)$
else skip

Consider filling the hole with

$a = x_{LL} := -1$ and $a' = x_{LL} := 0$

Robust contexts

Robust contexts

□ $[\bullet] ; x_{LH} := \mathbf{declassify}(y_{HH}, LH)$

Robust contexts

- $[\bullet] ; x_{LH} := \text{declassify}(y_{HH}, LH)$
- $[\bullet] ; \text{if } x_{LH} \text{ then } y_{LH} := \text{declassify}(z_{HH}, LH)$

Robust contexts

- $[\bullet] ; x_{LH} := \mathbf{declassify}(y_{HH}, LH)$
- $[\bullet] ; \mathbf{if } x_{LH} \mathbf{ then } y_{LH} := \mathbf{declassify}(z_{HH}, LH)$
 $\mathbf{else skip}$

Robust contexts

- $[\bullet] ; x_{LH} := \mathbf{declassify}(y_{HH}, LH)$
- $[\bullet] ; \mathbf{if } x_{LH} \mathbf{ then } y_{LH} := \mathbf{declassify}(z_{HH}, LH)$
 $\mathbf{else skip}$
- Remark: If x_{LH} was x_{LL} , and y_{LH} was y_{LL} , then these would be non-robust.

Security type system

- $\Gamma, pc \vdash e: l$
 - $\Gamma, pc \vdash c$
 - pc controls what information has affected control flow up to the current program point
 - If high confidentiality then attacker might learn secrets from the fact that execution reached that point
 - If low-integrity, then attacker might be able to affect whether control reaches that point
-

Typing rules

(val) $\Gamma \vdash \text{val} : l$

(var)
$$\frac{\Gamma(v) = l}{\Gamma \vdash v : l}$$

(op)
$$\frac{\Gamma \vdash e1 : l \quad \Gamma \vdash e2 : l}{\Gamma \vdash e1 \text{ op } e2 : l}$$

(sub)
$$\frac{\Gamma \vdash e : l \quad l \leq l'}{\Gamma \vdash e : l'}$$

Typing rules

(**skip**) $\Gamma, pc \vdash \text{skip}: l$ (**asgn**) $\frac{\Gamma, pc \vdash e: l \quad l \vee pc \leq \Gamma(v)}{\Gamma, pc \vdash v := e}$

(**seq**) $\frac{\Gamma, pc \vdash c1 \quad \Gamma, pc \vdash c2}{\Gamma, pc \vdash c1 ; c2}$

(**if**) $\frac{\Gamma, pc \vdash e: l \quad \Gamma, l \vee pc \vdash c1 \quad \Gamma, l \vee pc \vdash c2}{\Gamma, pc \vdash \text{if } e \text{ then } c1 \text{ else } c2}$

Typing rules

$$\text{(while)} \frac{\Gamma, \rho c \vdash e: l \quad \Gamma, l \vee \rho c \vdash c}{\Gamma, \rho c \vdash \text{while } e \text{ do } c}$$

$$\text{(sub)} \frac{\Gamma, \rho c \vdash c \quad \rho c' \leq \rho c}{\Gamma, \rho c' \vdash c}$$

Typing rules

(declassify)

$$\Gamma, pc \vdash e: l' \quad | \quad \forall pc \leq \Gamma(v)$$

$$l(l) = l(l') \quad pc, l' \text{ in HI}$$

$$\Gamma, pc \vdash v := \mathbf{declassify}(e, l)$$

Only high integrity data may be declassified and that declassification might only occur at a high integrity program point (pc)

A-attacks

- A command a is an A-attack under Γ if Γ , $(\text{Bot}, l(A)) \vdash a$ and `declassify` doesn't occur in a .
- Suppose $A = LL$. Are the following attacks?
 - $x_{LL} := y_{LL}$
 - **while** x_{HL} **do** skip
 - **skip**
 - $x_{HH} := y_{LH}$

Properties of A-attacks

- An A-attack under Γ
 - Does not have assignments to high-integrity variables
 - Satisfies noninterference

Properties of the type system

- If $\Gamma, pc \vdash c$ and declassify doesn't occur in c , then c satisfies noninterference.
- If $\Gamma, pc \vdash c$ and then for all integrity levels l , we have
 - for all $M1, M2$. if $M1$ and $M2$ are (Top, l) -indistinguishable, then $\langle M1, c \rangle$ and $\langle M2, c \rangle$ are weakly indistinguishable upto (Top, l) .

Key theorem: Robustness

- It is important that holes not be placed at high-confidentiality program points:
 - $C(pc)$ in LC
 $\Gamma, pc \vdash \bullet$
 - If $\Gamma, pc \vdash c[\cdot]^+$ then $c[\cdot]^+$ satisfies robust declassification.
-

Key idea

- Since declassification is not allowed in a low-integrity context, the type system ensures that changes in the low-integrity values may not reflect on low-confidentiality behavior.

Password example

$\Gamma, pc \vdash \text{match}(\text{pwdI}, \text{salt}, \text{pwd}, \text{hashR}, \text{matchR}) =$

$\text{hashR} := \mathbf{declassify}(\text{hash}(\text{pwd}, \text{salt}), (C_salt, I));$

$\text{matchR} := (\text{pwdI} == \text{hashR})$

$LH \vdash \text{match}(\text{pwdI}, \text{salt}, \text{pwd}, \text{hashR}, \text{matchR}): LH * LH * HH * LH * LH$

Attack: $[\bullet]$; $\text{match}(\text{hash}(x_{HH}, 0), 0, y_{LL}, \text{hashR}, \text{matchR});$

if matchR **then** $z_{LL} := 1$ **else** $z_{LL} := 0$

Consider $M1(x_{HH}) = 2$ and $M2(x_{HH}) = 3$, and attacks $y_{LL} := 0$ and $y_{LL} := 2$

Endorsement

- Sometimes it makes sense to give untrusted code the ability to affect what information is released by the program
 - $[\bullet]$; **if** $x_{LL} = 1$ **then** $z_{LH} := \text{declassify}(y1_{HH}, LH)$
else $z_{LH} := \text{declassify}(y2_{HH}, LH)$
 - Consider attacks $x_{LL} := 1$ and $x_{LL} := 2$
 - This code is considered reasonably secure
-

Qualified robustness

- Untrusted code is given a limited ability to affect information release.

 - This ability is marked explicitly in the code with **endorse**(e, l): upgrades the integrity of the result
 - Qualify robustness property to make it insensitive to how these expressions evaluate

 - $\langle M, \text{endorse}(e, l) \rangle \rightarrow \text{val}$
 - Val is nondeterministically chosen
-

Formalizing qualified robustness

- Two trace sets $T1$ and $T2$ are indistinguishable upto l if for all $t1$ in $T1$, there exists some $t2$ in $T2$ such that $t1$ and $t2$ are weakly indistinguishable upto l and for all $t2$ in $T2$, symmetric
- Lift robustness definition to sets of traces

Enforcing qualified robustness

$$\Gamma, pc \vdash e: l' \quad | \quad \forall pc \leq \Gamma(v)$$
$$\mathbf{C}(l) = \mathbf{C}(l') \quad l' \text{ in } \mathbf{LI} \quad | \quad l \text{ in } \mathbf{HI}$$

$$\Gamma, pc \vdash v := \mathbf{endorse}(e, l)$$

Has no impact to confidentiality in the absence of declassify

Qualified robustness example

□ LH |- update(pwdI, salt, oldP, newP, hashR, matchR)
= oldH := **endorse**(oldP, LH);
newH:= **endorse**(newP, LH);
match(pwdI, salt, oldH, hashR,matchR);
if matchR **then** pwdI = hash(newH, salt)
else skip

LH |- update(pwdI,salt,oldP,newP,hr,mr):
LH * LH * HL * HL * LH * LH

Discussion points

- Attacker model
- “Scrambling” semantics for endorsement
- Password example