Robust Declassification

Steve Zdancewic
Andrew Myers
CSFW ’01

CS711, 12 Nov 03
Stephen Chong

Declassification

- Real systems intentionally leak (declassify) confidential information
  - Purchase of information
  - Aggregated data
  - Encryption
  - Security protocols
    - Commit-reveal, challenge-verify, …
  - E.g. Password checker

Password Example

```
// passwd is the password
//      h is secret and shouldn’t be revealed
//  guess is the user’s guess
//      t is time (0 before guess checked;
//                 1 after)
//      r is the result (1 if passwd == guess)
```

context (a.k.a. pc)

```
if (passwd == guess) {
  r := 1;
}
else {
  r := 0;
}
t := t + 1;
```

assignment to r in a context that depends on passwd

Password Example

```
// passwd is the password
//      h is secret and shouldn’t be revealed
//  guess is the user’s guess
//      r is the result (1 if passwd == guess)
//      t is time (0 before guess checked;
//                 1 after)
```

context (a.k.a. pc)

```
if (declassify(passwd == guess)) {
  r := 1;
}
else {
  r := 0;
}
t := t + 1;
```

Program does not satisfy noninterference!
In general, declassification violates noninterference.

Life in a World Without Noninterference

- What useful info flow security properties can we describe that permit declassification?
  - Statically check authority (as in Jif, PKI)
  - Intransitive noninterference
  - Quantify information declassified
  - Robust declassification
  - Other notions?
- Still trying to *define* suitable security properties
  - Proving/guaranteeing properties another issue

Robust Declassification: Definitions

- **System** $S=(\Sigma, \alpha)$
  - Σ: set of states
  - $\alpha \subseteq \Sigma\times\Sigma$: transition relation
- **Trace** τ
  - A finite sequence of states
  - $\sigma_0\sigma_1\sigma_2\ldots\sigma_{n-1}$
  - Equivalent up to stuttering
    - $\sigma_0\sigma_1\sigma_1\sigma_1\sigma_2\sigma_2 \equiv \sigma_0\sigma_0\sigma_0\sigma_1\sigma_2 \equiv \sigma_0\sigma_1\sigma_2$
- **View** $\approx$
  - An equivalence relation on Σ
  - What observations can be made on a state
  - E.g. Low-equivalence: low security locations can be observed, high security locations cannot.

Password example

$$\begin{tikzcd}[column sep=small]
 & & (p, h, g, r, t) & & \\
\text{passwd} \arrow[urr] & \text{secret} \arrow[ur] & \text{guess} \arrow[u] & \text{result} \arrow[ul] & \text{time} \arrow[ull]
\end{tikzcd}$$

Password example

Transitions:

$$\begin{tikzcd}[column sep=small]
(0, h, 0, r, 0) \arrow[r] \arrow[d] & (0, h, 1, r, 0) \arrow[d] & (1, h, 0, r, 0) \arrow[r] \arrow[d] & (1, h, 1, r, 0) \arrow[d] \\
(0, h, 0, 1, 1) & (0, h, 1, 0, 1) & (1, h, 0, 0, 1) & (1, h, 1, 1, 1)
\end{tikzcd}$$

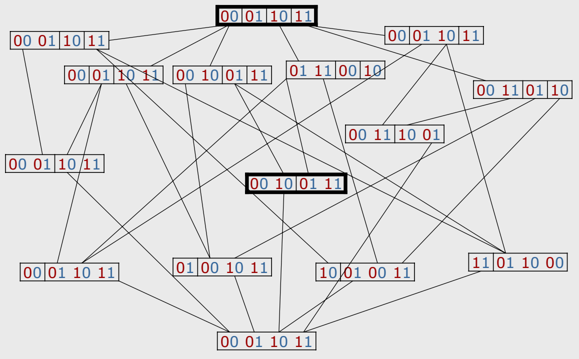
A view $\approx_L$ (for low equivalence):

$(0, h, 0, r, 0) \approx_L (1, h, 0, r, 0)$
$(0, h, 1, r, 0) \not\approx_L (1, h, 0, r, 0)$

Lattice of Views

- I(Σ): the set of all views of the system
  - Forms a lattice:
    - $\approx_A \;|_1\; \approx_B \;\Leftrightarrow\; \forall \sigma_1,\sigma_2.\; \sigma_1 \approx_B \sigma_2 \Rightarrow \sigma_1 \approx_A \sigma_2$
- Example
  - Consider states with 2 locations, each location having value 0 or 1.
  - 4 possible states: 00, 01, 10, 11

Doh! Stupid lattice…



Observations

- **Observations** of S wrt σ_0 and $\approx$
  - All sequences of equivalence classes of traces of S starting from σ_0
  - $Obs(S,\approx,\sigma_0) =$
    $\{[\sigma_0]_\approx[\sigma_1]_\approx\ldots[\sigma_{n-1}]_\approx \mid \sigma_0\sigma_1\ldots\sigma_{n-1}$ is a trace of $S\}$

Password Observations

- Traces
  - τ_0: (0, 0, 0, 1, 0) (0, 0, 0, 1, 1)
  - τ_1: (0, 0, 0, 1, 0) (0, 0, 1, 1, 0) (0, 0, 1, 0, 1)
  - τ_2: (0, 0, 0, 1, 0) (0, 0, 1, 1, 0) (0, 0, 0, 1, 0) (0, 0, 0, 1, 1)
  - …
- $Obs(S,\approx_L,(0, 0, 0, 1, 0)) = \{$
  (*, *, 0, 1, 0) (*, *, 0, 1, 1),
  (*, *, 0, 1, 0) (*, *, 1, 1, 0) (*, *, 1, 0, 1),
  (*, *, 0, 1, 0) (*, *, 1, 1, 0) (*, *, 0, 1, 0) (*, *, 0, 1, 1),
  …}

Observational Equivalence

- $Obs(S,\approx,\cdot):\Sigma \rightarrow$Observations induces another equivalence relation $S[\approx]$
  - $(\sigma,\sigma') \in S[\approx]$
    $\Leftrightarrow$ $Obs(S,\approx,\sigma) = Obs(S,\approx,\sigma')$
    $\Leftrightarrow$ "σ,σ' are observationally equivalent"
- Password example
  - (0, 0, 0, 1, 0) and (0, 1, 0, 1, 0) are obs. equivalent
  - (0, 0, 0, 1, 0) and (1, 0, 0, 1, 0) are not obs. Equivalent
- $S[\approx]$ gives at least as much info as $\approx$
  - I.e. $\approx \mid_I S[\approx]$

≈-Secure System

- S is **≈-secure**
  iff "all ≈-equiv. states are obs. equiv."
  iff $\forall \sigma,\sigma'.\ \sigma \approx \sigma' \Rightarrow (\sigma,\sigma') \in S[\approx]$
  iff $S[\approx] \mid_I \approx$
  iff $S[\approx] =_I \approx$
- Intuition: a passive attacker with view $\approx$ cannot learn anything new about the initial state by watching the system execute.
  - Essentially noninterference
  - Initial state contains all "important" information

A Limit to Information

- Recall: $S[\approx]$ is an equivalence relation on Σ, with $\approx \mid_I S[\approx]$
  - $S^0[\approx] = \approx$
  - $S^{n+1}[\approx] = S[S^n[\approx]]$
  - $S^\omega[\approx] = \int_{n\,\in\omega} S^n[\approx]$
- Intuition: $S^\omega[\approx]$ is the lowest view that can see all of the information that S will declassify
  - For any system S and view $\approx$, S is $S^\omega[\approx]$-secure

Active Attackers

- Assume we have an attacker with view $\approx_A$, and a system S that intentionally declassifies information
  - S is not $\approx_A$-secure
- Could an active attacker make S reveal more information than S meant to?
  - i.e. laundering attacks

Active Attackers

- **Active attackers**
  - Can add transitions α_{Att} to S
    - i.e. $(\Sigma, \alpha \cup \alpha_{Att})$
  - "Fairness": α_{Att} is limited to transitions that don't themselves declassify data, i.e. must be laundering attacks.
  - An $\approx_A$**-attack** is a system $Att = (\Sigma, \alpha_{Att})$ such that Att is $\approx_A$-secure
    - Write $Att \cup S$ for $(\Sigma, \alpha \cup \alpha_{Att})$
- What sort of attacks does this correspond to?
  - Attacker injecting code in the system that satisfies noninterference
  - Randomly flipping bits in the machine, e.g. passing a magnet over it

Robustness (at last)

- A system $S=(\Sigma, \alpha)$ is **robust** with respect to a class B of $\approx_A$-attacks if
  $\forall\ Att = (\Sigma, \alpha_{Att}) \in \mathrm{B}.\ (S \cup Att)[\approx_A] \mid_I S[\approx_A]$
- Intuition: Watching the attacked system reveals no more information than watching the original system

Attacking the Password Program

- Add attack transitions:
  (p, h, g, r, 0) α_{Att} (h, h, g, r, 0)
  - Note: $Att = (\Sigma, \alpha_{Att})$ is $\approx_L$-secure
- Password program is *not* robust against Att, since
  - $((0, 1, 0, 0, 0), (0, 0, 0, 0, 0)) \notin (S \cup Att)[\approx_L]$
    but
  - $((0, 1, 0, 0, 0), (0, 0, 0, 0, 0)) \in S[\approx_L]$
  - i.e. $(S \cup Att)[\approx_L] \mid_{\mathrm{I}}\!/ S[\approx_L]$

$\approx_A$-security and Robustness

- If S is $\approx_A$-secure, then S is robust to all $\approx_A$-attacks
  - i.e. If a system doesn’t do any declassification, an attacker cannot launder any data.

Dude, Where’s my Language?

- Use language-level constructs/analysis to rule out attacks that the system would not be robust against
  - High integrity for the data to declassify
  - High integrity for the decision to declassify
- But…
  - Vulnerable to attacks outside language abstraction
  - What is the interaction with endorse, the dual of `declassify`?

Language level attacks

- High integrity for data to declassify

```
if (declassify(passwd == guess)) {
  r := 1;
}
else {
  r := 0;
}
t := t + 1;
```

Language level attacks

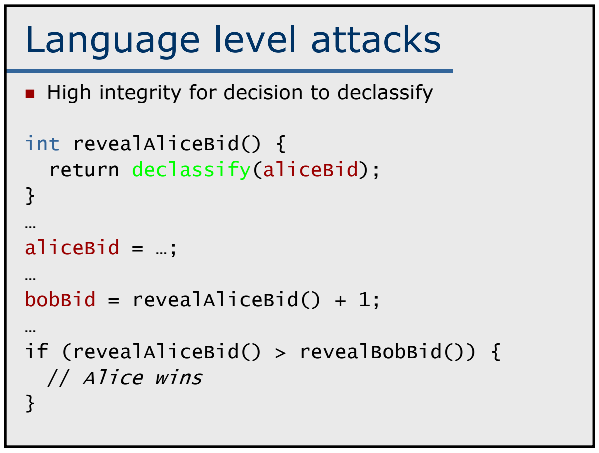
- High integrity for data to declassify

```
passwd = h;
if (declassify(passwd == guess)) {
  r := 1;
}
else {
  r := 0;
}
t := t + 1;
```

Language level attacks

- High integrity for decision to declassify

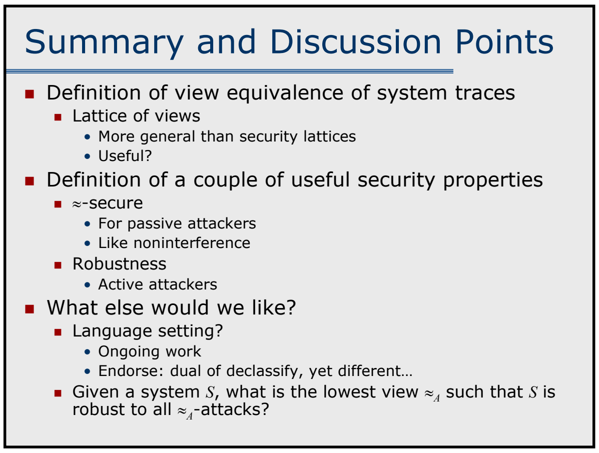
```
int revealAliceBid() {
  return declassify(aliceBid);
}
…
aliceBid = …;
…
bobBid = …;
…
if (revealAliceBid() > revealBobBid()) {
  // Alice wins
}
```



Language level attacks

- High integrity for decision to declassify

```
int revealAliceBid() {
  return declassify(aliceBid);
}
…
aliceBid = …;
…
bobBid = revealAliceBid() + 1;
…
if (revealAliceBid() > revealBobBid()) {
  // Alice wins
}
```



Summary and Discussion Points

- Definition of view equivalence of system traces
  - Lattice of views
    - More general than security lattices
    - Useful?
- Definition of a couple of useful security properties
  - $\approx$-secure
    - For passive attackers
    - Like noninterference
  - Robustness
    - Active attackers
- What else would we like?
  - Language setting?
    - Ongoing work
    - Endorse: dual of declassify, yet different…
  - Given a system S, what is the lowest view $\approx_A$ such that S is robust to all $\approx_A$-attacks?