Information flow control using Decentralized Labels

Kumar Avijit
Recap

• Last lecture:
  • Non-interference as a security property
  • Simple model of security levels: HI, LO
  • A type system that ensures non-interference.

• This lecture:
  • Non-interference too conservative.
  • “Controlled” information leaks necessary
Outline

• Motivation
• Decentralized Label Model
• Declassification and Endorsement
• Type system
• What more is to be desired
Motivation

Bank Server

Enter your password
Motivation

Wrong password!

Enter your password

Bank Server
Motivation

Too many attempts!

Bank Server
public class GetPassword{
    private String secretpw;
    private int tries;
    private int balance;

    void makeGuess(String pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp("Too many attempts!");
            }
            else{
                tries ++;
                if(pw == secretpw){
                    finishapp("Your balance is “ + balance);
                }
                else
                    print("Wrong password");
            }
        } else{
            tries ++;
            if(pw == secretpw){
                finishapp("Your balance is “ + balance);
            }
            else
                print("Wrong password");
    }
} .....
public class GetPassword{
    private String secretpw;
    private int tries;
    private int balance;

    void makeGuess(String pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp("Too many attempts!");
            }
            else{
                tries ++;
                if(pw == secretpw){
                    finishapp("Your balance is "+balance);
                }
                else
                    print("Wrong password");
            }
        }
    }
}.....
Bank application

```java
public class GetPassword{
    private String secretpw;
    private int tries;
    private int balance;

    void makeGuess(String pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp("Too many attempts!");
            } else{
                tries ++;
                if(pw == secretpw){
                    finishapp("Your balance is "+ balance);
                } else
                    print("Wrong password");
            }
        }
        } ..... }
```
A note on the adversary model

- The adversary plays by the rules of the language.
- He/she cannot peak into the memory.
- The language is implemented correctly.
What can be observed by attacker?

- The value of $x$
  - directly reveals the value of $y$.
- that the value of $x$ has (not) changed.
- reveals information contained in the control flow of the program.
What can be observed by attacker?

• The value of $x$
  • directly reveals the value of $y$.
• that the value of $x$ has (not) changed.
• reveals information contained in the control flow of the program.

```java
boolean b;
int x = 0;
...
if (b) {
    x = y + 1;
}
```
Other possible observations

• How much time did it take for the value of \( x \) to change!

• could give a clue about the number of iterations of a loop.

• Patterns in power drawn by the processor.

• ....
• Motivation

• **Decentralized Label Model**

• **Declassification and Endorsement**

• **Type system**

• **What more is to be desired**
Decentralized label model

• Each location (variable) has a set of owners who can read/write the value.
• Each owner can express his policy about allowing others to read/write the location.
• Each owner’s policy is expressed independently of others’ policies.
• The set of policies for a location is termed as its label.
Labels and information

• The label of a value is the same as the label of the variable/location holding the value.

• The type system ensures that the labels of a location and the value being stored there match.

• direct information channels
Simple flows due to control transfer

- Each statement and expression tagged with a “program-counter label” (pc)
- The pc approximates which variables can be leaked by knowing that control came to that particular statement/expression.
Simple flows due to control transfer

• Each statement and expression tagged with a “program-counter label” ($pc$)

• The $pc$ approximates which variables can be leaked by knowing that control came to that particular statement/expression.

```c
if (b) {
    x = y + 1;
}
```
Decentralized label model

- Labels speak directly about the principals.
- More granularity than HI and LO classes.
- Notion of execution on behalf of a principal.
  - this is actually ascertained by runtime checks.
Confidentiality policies

\[ \{ o_1 \rightarrow r_1, r_2 \} \]

Owner \quad Allowed readers
Confidentiality policies

\{ o_1 \rightarrow r_1, r_2; o_2 \rightarrow r_2, r_3 \}

Owner \quad Allowed readers
Confidentiality policies

\[ \{ o_1 \rightarrow r_1, r_2; o_2 \rightarrow r_2, r_3 \} \]

Owner  \hspace{2cm} Allowed readers

Access is allowed only

to \( r_2 \)
Variable names in policies

• Allow a label to contain variable names.

• A label \( \{a, o:r\} \) specifies following policies:
  • \( o:r \), and
  • the policies associated with the variable \( a \)

• A \( pc \) can now be treated as a label.
Ordering confidentiality labels

- A label determines which principals can read the data.
- \( L_1 \sqsubseteq L_2 \) if \( L_1 \) is at most as restrictive as \( L_2 \)
  - i.e. if \( L_2 \) permits \( r \) to read, then \( L_1 \) too permits \( r \).
Ordering labels

\[ \{alice \rightarrow bob, charlie\} \sqsubseteq \{alice \rightarrow bob\} \]

\[ \{alice \rightarrow mary\} \sqsubseteq \{alice \rightarrow mary; bob \rightarrow mary\} \]
public class GetPassword{
    private String{bank ➔ bank} secretpw;
    private int{bank ➔ client} tries;
    private int{bank ➔ bank} balance;

    void makeGuess{bank ➔ client}(String{bank ➔ client} pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp(“Too many attempts!”);
            }
            else{
                tries ++;
                if(pw == secretpw){
                    finishapp(“Your balance is ”+ balance);
                }
                else
                    print(“Wrong password”);
            }
        }
    }
}
An integrity policy is a guarantee of quality by owner $o$, that only $w_1$ and $w_2$ will be able to affect the value.
Integrity policies

An integrity policy is a guarantee of quality by owner $o$, that only $w_1$ and $w_2$ will be able to affect the value.
## Integrity labels

<table>
<thead>
<tr>
<th>$x =$</th>
<th>$v$</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>${o \leftarrow w}$</td>
<td>${o \leftarrow w, z}$</td>
<td>$\times$</td>
</tr>
<tr>
<td>${o \leftarrow w, z}$</td>
<td>${o \leftarrow w}$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>${o \leftarrow w}$</td>
<td>${o \leftarrow w; o' \leftarrow z}$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>
public class GetPassword{
    private String secretpw;
    private int tries;
    private int balance;

    void makeGuess(String pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp("Too many attempts!");
            }
            else{
                tries ++;
                if(pw == secretpw){
                    finishapp("Your balance is "+ balance);
                }
                else
                    print("Wrong password");
            }
        }
    }
}
Using integrity labels

```java
public class GetPassword{
    private String secretpw;
    private int tries;
    private int balance;

    void makeGuess(String pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp("Too many attempts!");
            }
            else{
                tries ++;
                if(pw == secretpw){
                    finishapp("Your balance is " + balance);
                }
                else
                    print("Wrong password");
            }
        }
    }
} ....
```

Should be typed under `pc = {pw}`
public class GetPassword{
    private String \{bank \rightarrow \text{bank}; \text{bank} \leftarrow \text{bank}\} secretpw;
    private int \{bank \rightarrow \text{client}; \text{bank} \leftarrow \text{bank}\} tries;
    private int \{bank \rightarrow \text{bank}; \text{bank} \leftarrow \text{bank}\} balance;

    void makeGuess(\{bank \rightarrow \text{client}\}(String \{bank \rightarrow \text{client}; \text{bank} \leftarrow \text{client}\) pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp("Too many attempts!");
            }
            else{
                tries ++;
                if(pw == secretpw){
                    finishapp("Your balance is "+ balance);
                }
                else
                    print("Wrong password");
            }
        }
    }
} .....
• Motivation
• Decentralized Label Model
• Declassification and Endorsement
• Type system
• What more is to be desired
Declassification

- Reflects intensional release of information.
- Instructs the compiler to downgrade the security policy associated with a variable/value within a region of code.
- This is done by syntactic relabeling of the variable/value being declassified.
- Since this is purely a compiler directive, does not have any operational significance.
How to declassify?

- A code running as authority p can only declassify policies where p is the owner.
- Declassification by one principal does not violate other principals’ policies.
# Decentralized declassification

<table>
<thead>
<tr>
<th>me → alice; you → bob</th>
<th>{}</th>
</tr>
</thead>
<tbody>
<tr>
<td>me → charlie; you → bob</td>
<td>{}</td>
</tr>
<tr>
<td>me → alice, bob; you → bob</td>
<td>{bob}</td>
</tr>
</tbody>
</table>
Using declassification

```java
public class GetPassword where authority(bank){
    ....
    void makeGuess(bank → client)(String(bank → client; bank ← client) pw){
        if(pw != NULL){
            if(tries >= MAXTRIES){
                finishapp("Too many attempts!");
            }
        } else{
            tries ++;
            if(pw == secretpw){
                declassify( {bank → bank} to {bank → client} ){
                    finishapp("Your balance is " + balance);
                }
            } else
                declassify( {bank → bank} to {bank → client} ){
                    print("Wrong password");
                }
        }
    }
}
```
public class GetPassword where authority(bank) {
    ....
    void makeGuess(bank → client)(String(bank → client; bank ← client) pw) {
        if (pw != NULL) {
            if (tries >= MAXTRIES) {
                finishapp("Too many attempts!");
            } else {
                tries ++;
                if (pw == secretpw) {
                    declassify( {bank → bank} to {bank → client} ){
                        finishapp("Your balance is " + balance);
                    } } 
            } else 
                declassify( {bank → bank} to {bank → client} ){
                    print("Wrong password");
                } 
        } 
    } 
}
Endorsement: Upgrading integrity

- Reflects intensional upgrading of integrity levels of result of a computation.
- Similar in spirit to declassification, except for the integrity labels.
public class GetPassword where authority(bank){
   ....
   void makeGuess(bank → client)(String (bank → client; bank ← client) pw){
      if(pw != NULL){
         if(tries >= MAXTRIES){
            finishapp("Too many attempts!");
         } else{
            endorse(pw, {bank ← client} to {bank ← bank}){
               tries ++;
            }
            if(pw == secretpw){
               declassify( {bank → bank} to {bank → client} ){
                  finishapp("Your balance is "+ balance);
               }
            }
         } else
            declassify( {bank → bank} to {bank → client} ){
               print("Wrong password");
            }
      }
   }
}
• Motivation
• Decentralized Label Model
• Declassification and Endorsement
• **Type system**
• What more is to be desired
Type system: Judgments

\[ \Gamma; pc \vdash e : l \quad \text{Expression } e \text{ has a label } l \]

\[ \Gamma; pc \vdash s \ ok \quad \text{Statement } s \text{ is well typed} \]
Type system: rules

\[ \begin{align*}
\Gamma(x) &= l \\
\Gamma; pc \vdash x : l
\end{align*} \]

\[ \begin{align*}
\Gamma; pc \vdash e : l \\
pc' \subseteq pc \\
l \subseteq l'
\end{align*} \]

\[ \Gamma; pc' \vdash e : l' \]

\[ \begin{align*}
\Gamma; pc \vdash e_1 : l \\
\Gamma; pc \vdash e_2 : l
\end{align*} \]

\[ \Gamma; pc \vdash e_1 + e_2 : l \]

\[ \begin{align*}
\Gamma; pc \vdash s_1 \ ok \\
\Gamma; pc \vdash s_2 \ ok
\end{align*} \]

\[ \Gamma; pc \vdash s_1; s_2 \ ok \]

\[ \begin{align*}
\Gamma; pc \vdash e : l \\
\Gamma; pc \sqcap l \vdash s_1 \ ok \\
\Gamma; pc \sqcap l \vdash s_2 \ ok
\end{align*} \]

\[ \Gamma; pc \vdash if \ e \ then s_1 \ else \ s_2 \ ok \]
Join of security policies

• Recall the ordering on labels and policies.

• A join of two labels can be thought of as the least upper bound wrt the ordering

• The join is at least as restrictive than either labels.

• Is the least restrictive such label.
Calculating joins

- Simply combine the policies from the two labels.

\[ \{ o_1 \rightarrow r_1, r_2 \} \sqcup \{ o_2 \rightarrow r_2, r_3 \} = \{ o_1 \rightarrow r_1, r_2; o_2 \rightarrow r_2, r_3 \} \]
Typing declassification

\[ [\Gamma; pc]^Q_P \vdash s \text{ ok} \quad \text{authority}(pc) = \text{owner}(P) \quad \text{owner}(P) = \text{owner}(Q) \]
\[ \Gamma; pc \vdash \textit{declassify}(P \text{ to } Q)\{s\} \text{ ok} \]
• Motivation
• Decentralized Label Model
• Declassification and Endorsement
• Type system

• What more is to be desired
Security guarantees

• In absence of declassification and endorsement, non-interference is respected.

• With declassification, a property called “robust declassification” is followed.
Limitations of DLM

• What is the meaning of the program types in presence of declassification and endorsement?

• Is it possible to safely install a third-party app upon just examining its type!
Meaning of types in JiF?

```java
sign(String{*->pub} msg){
    int{*->pub} hash = msg.hashCode();
    int{*->*; *<-*} skey = get_skey();
    String{*->pub} sig = RSAEnc[*](skey, hash);
    return msg+sig;
}
```
Meaning of types in JiF?

```java
sign(String{*->pub} msg){
    int{*->pub} hash = msg.hashCode();
    int{*->*; *<-*} skey = get_skey();
    String{*->pub} sig = RSAEnc[*](skey, hash);
    return msg+sig;
}
```

RSAEnc : int{p->p; p<-p} x int --> int @ p
Meaning of types in JiF?

```java
sign(String{*->pub} msg){
    int{*->pub} hash = msg.hashCode();
    int{*->*; *<-*} skey = get_skey();
    String{*->pub} sig = RSAEnc[*](skey, hash);
    return msg+sig;
}

RSAEnc(int{p->p; p<-p} k, int msg)
where authority(p){
    ....
    ....
    declassify({p->p} to {p->pub}){
        print(p + " 's key is " + k)
    }
    ....
}
```