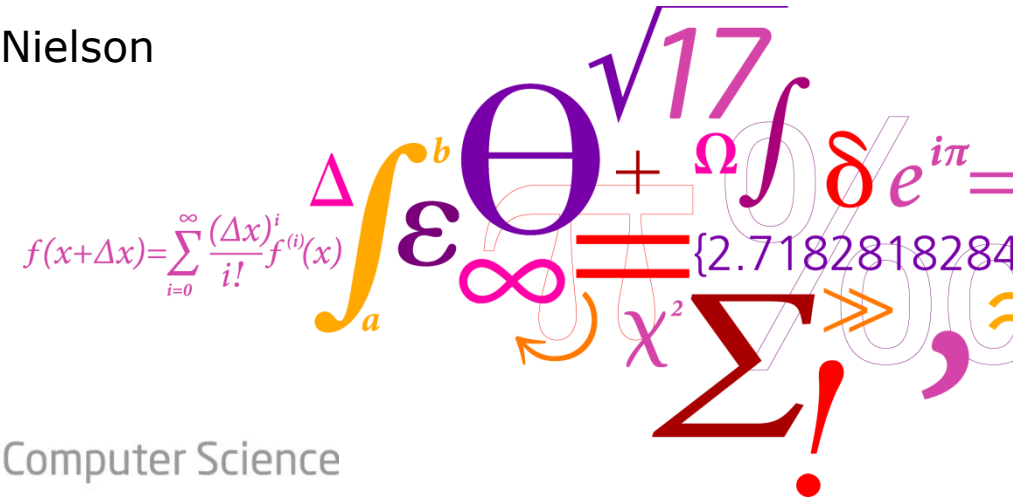
02295 Advanced Topics in Computer Science

Information Flow Control in C

using the Decentralized Label Model

Andrew Habib
Supervised by Professor: Flemming Nielson



DTU Compute
Department of Applied Mathematics and Computer Science

Outline

- Motivation
- Summary of DLM
- DLM applicability to C
- DLM applied to Subset of C
- Proposal for DLM with C pointers
- Future work
- Questions

Motivation

- Security(confidentiality and integrity) through access control models
- Finer-grain control over information dissemination
- Information flow control models
- DLM has been successfully applied to Java (JiF)
- C is widely used, including safety-critical systems
- Industry is into it: Airbus

Summary of DLM

- Environments with decentralized authority and mutual distrust
- Users have full control over information propagation
- Static analysis technique
  - Label data with owner(s)
  - Each owner authorizes his set of allowed reader(s)
  - Labels can change (by add/remove owners/readers) according to strict rules
  - Labels are (mostly) statically checked in similar fashion to normal types by the compiler

Summary of DLM
Principals and Labels

- **Principal**
  - User or authority entity/group
  - Organized in hierarchy called principal hierarchy (***acts for***)
- **Label**
  - Similar to normal types
  - ***L = {o1:r1; o2:r1,r2}***
  - ***owners(L) = {o1,o2}***
  - ***readers(L,o1)={r1}***

Summary of DLM

Relabeling Rules: $L_1 \rightarrow L_2$

- **Restriction**
  - Adds owners, removes readers or both

$$L_1 \sqsubseteq L_2 \Leftrightarrow \begin{array}{r} owners(L_1) \subseteq owners(L_2) \\ \forall O \in owners(L_1) : readers(L_1, O) \supseteq readers(L_2, O) \end{array}$$

- **Declassification:**
  - Adds readers to owner ***O***, removes owner ***O*** itself
  - Only if the process ***acts for*** owner ***O***
  - ***declassify(exp, L)***

Summary of DLM
Joining Labels: $L_1 \sqcup L_2$

- Values are computed by combining other values
- Labels of those values are joined (combined) to obtain label of the result

$$owners(L_1 \sqcup L_2) = owners(L_2) \cup owners(L_2)$$
$$readers(L_1 \sqcup L_2, O) = readers(L_1, O) \cap readers(L_2, O)$$

- $L_1 \sqcup L_2$ is the least restrictive label of both
- This definition is precise if

$$O \notin owners(L) \Rightarrow readers(L, O) = set\ of\ all\ principals$$

Summary of DLM
Lattice of Labels

- Partial order: restriction operator $\sqsubseteq$
- Least element: $\bot$
- Greatest element: $\top$
- Leas upper bound: join operator $\sqcup$
- Greatest lower bound: meet operator $\sqcap$

Summary of DLM
More...

- Block labels prevent implicit information flow
  - All variables v_i observed to reach a specific program point are joined together to form the block label: $\underline{B} = \bigsqcup_i \underline{v_i}$
- Implicit labels to functions and labels polymorphism
- Generated system of constraints is solved simultaneously
  - Solution found: adherence to flow policy
  - No solution / contradiction: violation of flow policy

DLM Applicability to C

- C pointers
- Weakly-typed C
- Inline assembly
- Pointers to functions

DLM Applied to Subset of C

```
1  int{C ⊔ d} decrypt(C,d,n){
2      int Msg=1;                          // ⊥ ⊑ Msg
3      for(i=0, i<d, i++)                  // L1 = i ⊔ d
4          Msg=Msg*C%n;                    // Msg ⊔ C ⊔ n ⊔ {rol} ⊔ L1 ⊑ Msg
5      Msg=Msg%n;                          // Msg ⊔ n ⊑ Msg
6      if_acts_for(decrypt,rol)
7          return declassify(Msg);         // Msg ⊑ Ld ⊔ {rol}, Ld ⊑ C ⊔ d
8      return 0;                           // ⊥ ⊑ C ⊔ d
9  }
```

$$L_1, \underline{Msg}, \underline{i} = \underline{C} \sqcup \underline{d} \sqcup \{rol\}$$
$$L_d = \underline{C} \sqcup \underline{d}$$

Proposal for DLM with C pointers

- Keep pointers functionality as is
- Do not spoil the advantages of static analysis techniques
- Introduce no/minor changes/additions to the original DLM

Proposal for DLM with C pointers
Variable label and address label

- Variable label becomes $L = \{L_a, L_b\}$
  - L_a is the normal label of the variable
  - L_b is the label of the variable address

- Moreover, we choose to enforce

$$owners(L_a) \subseteq owners(L_b)$$

Proposal for DLM with C pointers
Pointer dereference: which variable?

- At compile time, actual value of a pointer dereference is unknown
- Use results of **alias analysis** to limit the possibilities
- Label of a pointer dereference expression should be at least as restrictive as the most restrictive label of all variables in the alias analysis results

y=*z incurs constraint $\bigsqcup_i \underline{x_i} \sqsubseteq \underline{y}$

Where x_i ranges over alias analysis result of the pointer expression ***z**

Proposal for DLM with C pointers
Pointer dereference: who can do it?

- Pointer dereference is a sensitive operation
- Use results of **alias analysis** to allow a pointer dereference only when all possible owner(s) of variables in the alias analysis results are in the effective authority

$$effective\ authority \supseteq \bigcup_i owners(\underline{x_i})$$

where x_i ranges over alias analysis result of the pointer expression

Future Work

- Alias analysis might be invalidated at run-time
- Develop type system for C inline assembly
- Enforcing strong typing on C

Please refer to the paper for list of references

Acknowledgement

- The original idea: **Can we apply DLM to C language** is courtesy of professors Hanne and Flemming Nielson

- Guidance and supervision with insightful comments from professor Flemming Nielson

Thank you,

Questions