Formal Analysis of Security Policies

Alessia Ciraudo

Security Workshop

University of Catania

Development of an Access Control System

1) Authentication

2) Security Policies

3) Security Mechanisms

Security Policy

"A set of norms regulating the modalities – obligation, permission, interdiction – for a set of agents on some action "

Inconsistencies

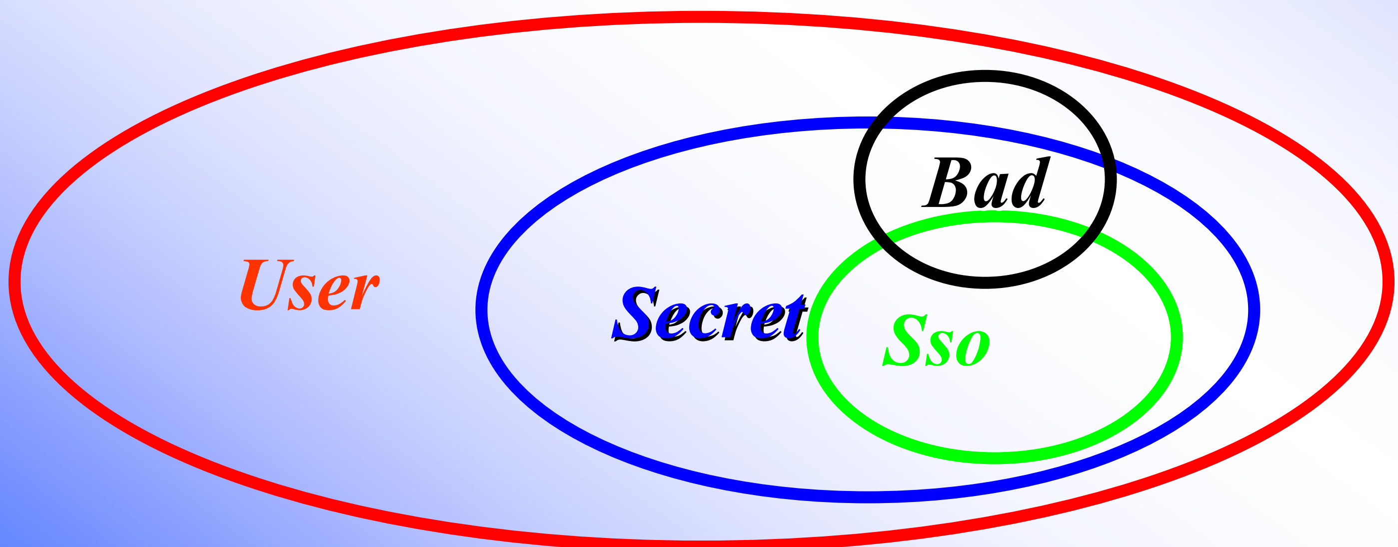
- **Contradiction:**

  "forbidden smoke" and "obligatory smoke"!

- **Dilemma:**

  "forbidden smoke" and "forbidden no smoke"!

Roles



Example Policy

11 Norms

4 Roles

```
N1: if play(a,User) and public(f)
    then Perm(Read(a,f))

N2: if play(a,User) and public(f) and owner(f,a)
    then Perm(Write(a,f))

N3: if play(a,User)
    then Forb(Downgrade(a,f))

N4: if play(a,User) and password(a,p) and old(p)
    then Obl(Change_Psswd(a))
```

Example Policy

```
N5: if play(a,Secret) and not(public(f))
    then Perm(Read(a,f))

N6: if play(a,Secret) and not(public(f))
        and owner(f,a)
    then Perm(Write(a,f))

N7: if play(a,Sso)

    then Perm(Downgrade(a,f))
```

Example Policy

```
N8:  if play(a,Bad)
     then Forb(Read(a,f))

N9:  if play(a,Bad)
     then Forb(Write(a,f))

N10: if play(a,Bad)
     then Forb(Downgrade(a,f))

N11: if play(a,Bad)
     then Forb(Change_Psswd(a))
```

Inductive Approach

- **Trace**: list of admissible norms induced by policy
- **Model of Policy**: set of all possible trace of norms that the policy admits
  - Mechanized with the proof assistant: PVS or Isabelle
- Properties of the model proved with the correspondent inductive principle

Inductive Definition of Policy

"Set of all possible trace of norms that the policy admits"

Base case

```
[] ∈ Policy
```

Inductive case

```
trace ∈ Policy ⇒ nm # trace ∈ Policy
```

Types

```
typedecl Agent

typedecl File

typedecl Psswd


datatype Role = User | Sso | Secret | Bad
```

Functions

```
consts
  play     :: "[Agent, Role]    ⇒ bool"
  owner    :: "[File, Agent]    ⇒ bool"
  password :: "[Agent, Psswd]   ⇒ bool"
  public   :: "File             ⇒ bool"
  old      :: "Psswd            ⇒ bool"
```

Constraints on Roles

```
axioms
 Secret_User [simp] : "play a Secret → play a User"
 Sso_Secret  [simp] : "play a Sso → play a Secret"
 Bad_User    [simp] : "play a Bad → play a User"

lemma Transitivity_Sso_User [simp] :
  " ∀ (a::Agent). play a Sso → play a User"
```

Operations

```
datatype operation =
    Read          Agent File
  | Write         Agent File
  | Change_Psswd  Agent
  | Downgrade     Agent File
  | Not_op        operation   ("¬o")

axioms
  Not_op_idemp [simp] : "¬o (¬o oper) = oper"
```

Norms

```
datatype norma =
    Obl          operation
  | Perm         operation
  | Forb         operation
  | Waived       operation
  | Not_norma norma          ("¬n")
```

Axioms for Norms

```
axioms

 Not_norma_idemp [simp]: "¬n (¬n nm) = nm"

 Perm_Obl [simp]: "Perm oper = ¬n (Obl (¬o oper))"

 Forb_Obl [simp]: "Forb oper   = Obl (¬o oper)"
```

Mechanization with Isabelle

```
types trace = “norma list”

consts Policy :: "trace set"

inductive "Policy"

intros

Empty : “[] ∈ Policy”

Norma_1 : “[|tr1 ∈ Policy; play a User;
         public f|]
            ⇒ Perm (Read a f) # tr1 ∈ Policy”
```

Inconsistencies

Contradiction

$$(\text{Obligatory}(op) \wedge \neg\, \text{Obligatory}(op))$$
$$\vee$$
$$(\text{Obligatory}(\neg\, op) \wedge \neg\, \text{Obligatory}(\neg\, op)$$

Dilemma

$$\text{Obligatory}(op) \wedge \text{Obligatory}(\neg\, op)$$
$$\vee$$
$$(\neg\, \text{Obligatory}(op) \wedge \neg\, \text{Obligatory}\ (\neg\, op))$$

Contradiction in Isabelle

```
consts Contradiction :: “norma ⇒ norma“

axioms Contradiction_1 [simp] :
“Contradiction (Obl oper) = ¬n (Obl oper)“

axioms Contradiction_2 [simp] :
“Contradiction (¬n (Obl oper)) = Obl oper”
```

Dilemma in Isabelle

```
consts Dilemma :: "norma ⇒ norma"

axioms Dilemma_1 [simp] :

"Dilemma (Obl oper) = Obl (¬o oper)"
```

Absence of Contradictions and of Dilemmas

```
theorem No_Contradiction :
  “[|nm ∈ set tr; tr ∈ Policy|] ⇒
   Contradiction nm ∉ set tr“
```

```
theorem No_Dilemma :
  “[|nm ∈ set tr;tr ∈ Policy|] ⇒
   Dilemma nm ∉ set tr”
```

Proof - step 1

```
tr ∈ Policy ⇒
        nm ∈ set tr → Contradiction nm → set tr
```

```
erule
Policy.induct
```

12 subgoal!!!

Proof – step 2

```
[|tr1 ∈ Policy; play a User; public f;
   nm ∈ set tr1 → Contradiction nm ∉ set tr1|]
⇒ nm ∈ set (Perm (Read a f) # tr1) →
  Contradiction nm ∉ set (Perm(Read a f) # tr1)
```

simp del: “Perm_Obl”

Proof – step 3

```
[| tr1 ∈ Policy; play a User; public f;
  nm ∈ set tr1 → Contradiction nm ∉ set tr1 |]
⇒ (nm = Perm (Read a f) →
  Contradiction (Perm (Read a f)) ≠ Perm (Read a f) ∧
  Contradiction (Perm (Read a f)) ∉ set tr1)
  ∧ (nm ∈ set tr1→ Contradiction nm ≠ Perm(Read a f))
```

```
subgoal_tac
"Contradiction(Perm(Read a f))
 ∉ set tr1"
```

Proof – step 4

```
[| tr1 ∈ Policy; play a User; public f;
   nm ∈ set tr1→ Contradiction nm ∉ set tr1 |]
⇒ Contradiction (Perm (Read a f)) ∉ set tr1
```

```
erule
Policy.induct
```

Proof – step 5

```
[| tr1 ∈ Policy; play a User; public f;
   nm ∈ set tr1 → Contradiction nm ∉ set tr1;
   Contradiction (Perm (Read a f)) ∉ set tr1|]
⇒ (nm = Perm(Read a f) →
   Contradiction(Perm(Read a f)) ≠ Perm(Read a f) ∧
   Contradiction(Perm(Read a f)) ∉ set tr1)
∧ (nm ∈ set tr1→ Contradiction nm ≠ Perm(Read a f))
```

rule conjI

Proof – step 6

```
[| tr1 ∈ Policy; play a User; public f;
   nm ∈ set tr1 → Contradiction nm ∉ set tr1;
   Contradiction(Perm(Read a f)) ∉ set tr1|]
⇒ nm = Perm(Read a f) →
  Contradiction(Perm(Read a f)) ≠ Perm(Read a f)
  ∧ Contradiction(Perm(Read a f)) ∉ set tr1
```

simp

Proof – step 7

```
[| tr1 ∈ Policy; play a User; public f;
   nm ∈ set tr1 → Contradiction nm ∉ set tr1;
   Contradiction (Perm (Read a f)) ∉ set tr1|]
⇒ nm ∈ set tr1 → Contradiction nm ≠ Perm(Read a f)
```

```
erule
Policy.induct
```

Proof – step 8

auto

```
⋀ a f tr8 tr1.[| play a Bad; tr1 ∈ Policy;
  public f; ¬n (Obl (¬o (Read a f))) ∉ set tr8;
  ¬n (Obl (¬o (Read a f))) ∉ set tr1|]
  ⇒ False
```

Policy Inconsistencies

6 Contradictions:

N7 – N3: "A system security officer is both permitted and forbidden to downgrade a public file"

N8 – N1: "A bad user is both forbidden and permitted to read a public file"

N8 – N5: "A bad user is both forbidden and permitted to read a not public file"

N9 – N2: "A bad user is both forbidden and permitted to write on a public file he owns"

N9 – N6: "A bad user is both forbidden and permitted to write on a not public file he owns"

N10 – N7: "A bad user is both forbidden and permitted to downgrade a file"

1 Dilemma:

N11 – N4 : "A bad user is both forbidden and obliged to change his password"

Conclusions

- Developed the first inductive approach to prove security policy correctness
- Mechanized the approach with the proof assistant *Isabelle*
- Verified presence of many inconsistence in the example policy: proof script of 500 lines

Next steps…

- To simplify proof demonstration strategy
- Search of alternative formalization, if possible without trace
- Application to widest study case
- Extension to union of more policy

Thanks!!!