Enforcing Noninterference by Running One Version of the Program per Level

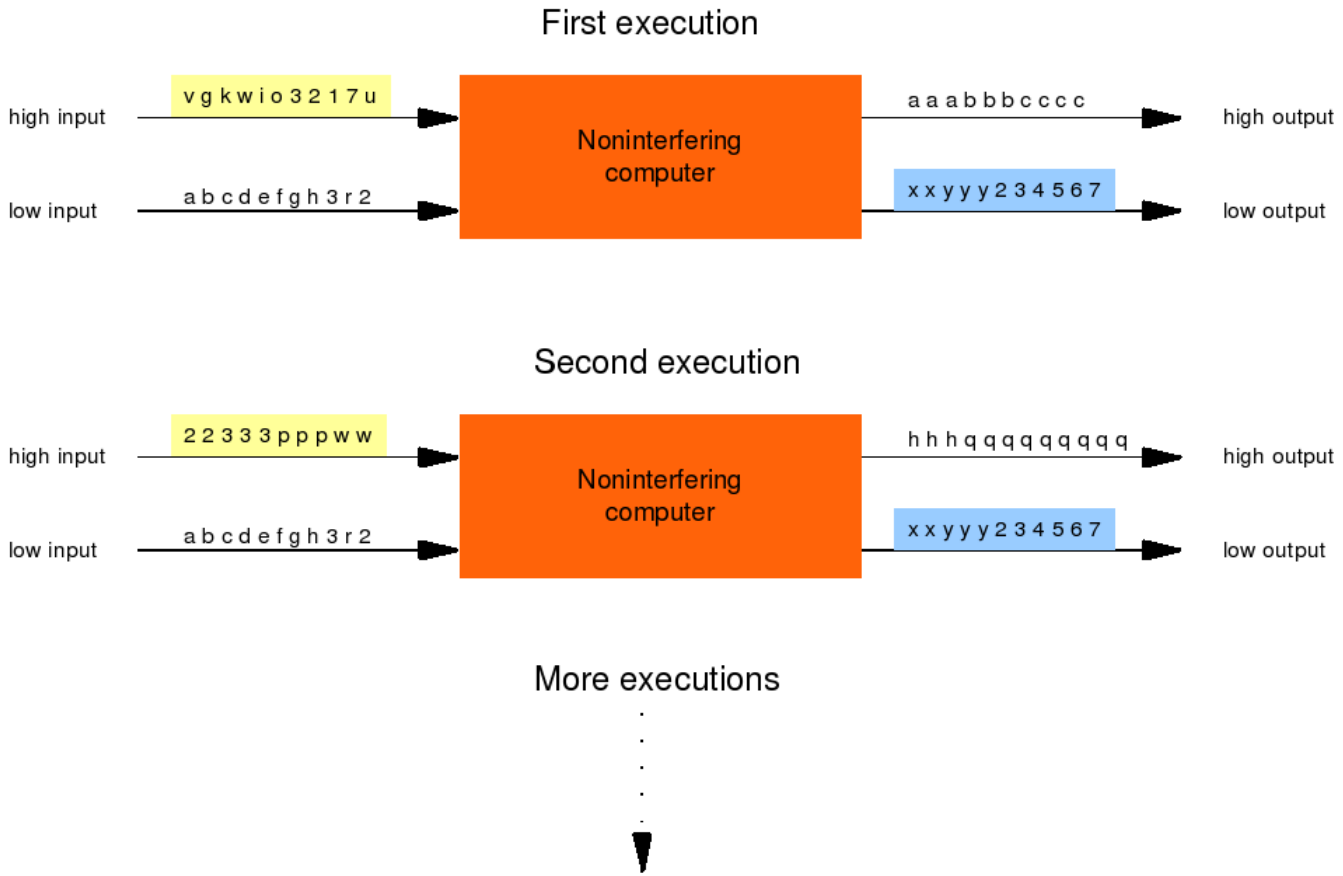
Maximiliano Cristiá
Universidad Nacional de Rosario
Flowgate Security Consulting
Argentina

Formal Methods in Security – ReSeCo Workshop
November, 2007

Introduction

- Noninterference is a *very* abstract formalization of the confidentiality problem of computer systems.
- It was proposed by Goguen and Meseguer in 1982.
- In simple terms it says: a group of users H is noninterfering w.r.t. to another group of users L, if the activity of H over the system cannot be sensed by L.
  - In other words, as far as the observations made by members of L, users belonging to H don't exist.

Noninterference



Military Security

- The military were the first in worrying in how to keep a secret inside of a multi-user, general-purpose computer system. cómputo multiusuario.
- They label each document and person with an *access class*, and they authorize a person to read a document if and only if the access class of the former is higher than or equal to the access class of the last.
  - To simplify the exposition we can think that an access class is a natural number; in this case it is called *security level*.

Military Security (cont.)

- The idea is to implement that security policy, *efficiently*, in a general-purpose, multi-user computer system; it's not as easy as it might look like.
  - Efficiently: preserve backward compatibility of applications, transparency for programmers and users, and usability.
- Noninterference it's a very good model for the military security policy since we can group users according to their security level and ask noninterference between the higher groups and the lower groups.
- Today it is accepted that noninterference is a solution the the military security problem.

Approaches to Noninterference

- Today noninterference is investigated according to two approaches:
  - Static. Static code analysis, type systems, certifying compilers.
  - Dynamic. An underlying module controls the execution of each process to ensure noninterference.
- The static approach is the dominant research line.
- We think that the static approach cannot be applied, efficiently, to general-purpose operating system like UNIX or Windows; this is the reason for which we work on the dynamic approach.

Variable Labeling

- One of the possible solutions is to label each process' variable with the security label of the information that it stores in each moment.
- To do that it is applied some flavor of the information flow model proposed by Denning (1976).
- However, it is not possible to implement it, efficiently, in low level programming languages.
- The problem rises with pointers.

Counterexample

`x` stores a secret value between 1 and 5. The following program can be used to reveal that value or it is seldom usable, depending on the labeling schema. `L` means not secret, `H` means secret, standard output works at `L`.

```
a := malloc(5); b : = a;
read(x);
for i := 0 to x - 1 do
  *(a + i) := 1;
endfor
for j := 0 to 4 do
  print(*(b + j));
endfor
```

Labeling only variables:

$(a[L], b[L], x[L], 0[L], 0[L], 0[L], 0[L], 0[L])$
$(a[L], b[L], x[H], 0[L], 0[L], 0[L], 0[L], 0[L])$
$(a[L], b[L], x[H], 1[H], 1[H], 1[H], 0[L], 0[L])$

Labeling pointers:

$(a[L], b[L], x[L], 0[L], 0[L], 0[L], 0[L], 0[L])$
$(a[H], b[H], x[H], 0[L], 0[L], 0[L], 0[L], 0[L])$
$(a[H], b[H], x[H], 1[H], 1[H], 1[H], 0[L], 0[L])$

A Dynamic Model Based on Virtualization

- The previous program shows that is not possible to control, efficiently, the flow of information within a process that needs to access both secret and public values.
- Then, the model we propose is bases on forking a process when it needs to access information at a higher security level.
- In this way, each process access data *up to* a particular level.
- Besides, all the forked process share the input depending on its security level.

Virtualization?

We say that the model is based on virtualization because the system behaves as if there were one computer for each security level.

For instance, if a user issues the command `vi secret_file public_file`, the system will create two processes for `vi`: one accesses both files but it can only write in `secret_file`, while the other accesses only the public file but this process can modify it.

Besides, when the user types in something, if the standard input is deemed public, both processes will receive that.

Then, it looks like if each process executes in a different computer, each one processing information up to a certain level, but both are connected to the same I/O.

The Model – The Grammar

The formal model introduced in this talk is very abstract and igt is only meant to show its most important features.

$$
\begin{aligned}
&Expr ::= \mathbb{N} \mid VAR \mid *VAR \mid \&VAR \mid Expr \boxplus Expr \\
&BasicSentence ::= \\
&\qquad \mathsf{skip} \mid var := Expr \mid *var := Expr \mid \mathsf{syscall}(\mathbb{N}, arg_1, \ldots, arg_n) \\
&ConditionalSentence ::= \\
&\qquad \mathsf{if}\ Expr\ \mathsf{then}\ Program\ \mathsf{fi} \mid \mathsf{while}\ Expr\ \mathsf{do}\ Program\ \mathsf{done} \\
&BasicAndConditional ::= BasicSentence \mid ConditionalSentence \\
&Program ::= BasicAndConditional \mid Program\ ;\ Program
\end{aligned}
$$

The Model – Two System Calls

Name	**System call**	**Meaning**
read(dev, var)	syscall($0, dev, var$)	Reads var from input device dev
write($dev, expr$)	syscall($1, dev, expr$)	Writes $expr$ to output device dev

The Model – The Semantics

$LS : Memory \rightarrow Program \rightarrow Memory$ where $Memory : VAR \rightarrow \mathbb{N}$, $addr : \mathbb{N} \rightarrow VAR$, $\overrightarrow{x, M} = addr \circ M(x)$.

$$LS(M, \text{skip}) = M \qquad \text{(LS-skip)}$$

$$LS(M, x := e) = M \oplus \{x \mapsto eval(M, e)\} \qquad \text{(LS- :=)}$$

$$LS(M, *x := e) = M \oplus \{\overrightarrow{x, M} \mapsto eval(M, e)\} \qquad \text{(LS-*)}$$

$$LS(M, \text{if } e \text{ then } P \text{ fi}) = \mathbf{if}\ eval(M, e)\ \mathbf{then}\ LS(M, P)\ \mathbf{else}\ M \qquad \text{(LS-if)}$$

$$LS(M, \text{while } e \text{ do } P \text{ done}) = \mathbf{if}\ eval(M, e)\ \mathbf{then}\ LS(M, P \text{ ; while } e \text{ do } P \text{ done})\ \mathbf{else}\ M \qquad \text{(LS-while)}$$

$$LS(M, P_1 \text{ ; } P_2) = LS(LS(M, P_1), P_2) \qquad \text{(LS- ;)}$$

The Model – Expression Evaluation

$M \in Memory$, $n \in \mathbb{N}$, $x \in VAR$, $e_1, e_2 \in Expr$

$$eval(M, n) = n \qquad \text{(eval-}\mathbb{N}\text{)}$$

$$eval(M, x) = M(x) \qquad \text{(eval-}VAR\text{)}$$

$$eval(M, *x) = M(\overrightarrow{x, M}) \qquad \text{(eval-}*\text{)}$$

$$eval(M, \&x) = addr^{-1}(x) \qquad \text{(eval-\&)}$$

$$eval(M, e_1 \boxplus e_2) = eval(M, e_1) \boxplus eval(M, e_2) \qquad \text{(eval-}\boxplus\text{)}$$

The Model – The Controlling Machine

$$SM : SState \times Env \times Memory \times Program \rightarrow SState \times Env \times Memory$$

where $SState$ and Env are defined by:

$$DEV ::= il \mid ih \mid ol \mid oh$$
$$LEVEL ::= L \mid H$$
$$SState \triangleq [m : Memory, dl : DEV \rightarrow LEVEL]$$
$$Env == DEV \rightarrow \text{ seq } \mathbb{N}$$

The intention behind these definitions is that the secret data will be stored in in SM while m will store public data.

The Model – Rules for SM

They are defined inductively following the grammar of the language.

$$SM(S, E, M, \mathsf{skip}) = (S, E, M) \qquad \text{(SM-skip)}$$

$$SM(S, E, M, x := expr) = \\ ([m \leftarrow LS(S.m, x := expr), dl \leftarrow S.dl], E, LS(M, x := expr)) \qquad \text{(SM- :=)}$$

$$SM(S, E, M, *x := expr) = \\ ([m \leftarrow LS(S.m, *x := expr), dl \leftarrow S.dl], E, LS(M, *x := expr)) \qquad \text{(SM-*)}$$

$$SM(S, E, M, P_1 \ ; \ P_2) = SM(SM(S, E, M, P_1), P_2) \qquad \text{(SM- ;)}$$

$$SM(S, E, M, \mathsf{read}(il, x)) =$$
$$([m \leftarrow S.m \oplus \{x \mapsto head \circ E(il)\}, dl \leftarrow S.dl], \qquad (\text{SM-read}(il))$$
$$E \oplus \{il \mapsto tail \circ E(il)\}, M \oplus \{x \mapsto head \circ E(il)\})$$

$$SM(S, E, M, \mathsf{read}(ih, x)) =$$
$$(S, E \oplus \{ih \mapsto tail \circ E(ih)\}, M \oplus \{x \mapsto head \circ E(ih)\})$$
$$(\text{SM-read}(ih))$$

$$SM(S, E, M, \mathsf{write}(ol, e)) =$$
$$(S, E \oplus \{ol \mapsto \langle eval(S.m, e)\rangle \frown E(ol)\}, M) \qquad (\text{SM-write}(ol))$$

$$SM(S, E, M, \mathsf{write}(oh, e)) =$$
$$(S, E \oplus \{oh \mapsto \langle eval(M, e)\rangle \frown E(oh)\}, M) \qquad (\text{SM-write}(oh))$$

$$SM(S, E, M, \text{if } e \text{ then } P \text{ fi}) =$$

$$\begin{cases} SM(S, E, M, P) & \textbf{if } eval(S.m, e) \wedge eval(M, e) \\ (SM(S, E, M, P).1, \\ \quad SM(S, E, M, P).2, M) & \textbf{if } eval(S.m, e) \wedge \neg eval(M, e) \\ (S, E', SM(S, E, M, P).3) & \textbf{if } \neg eval(S.m, e) \wedge eval(M, e) \\ (S, E, M) & \textbf{if } \neg eval(S.m, e) \wedge \neg eval(M, e) \end{cases}$$

$$\text{where } E' = E \oplus \{ih \mapsto SM(S, E, M, P).2(ih), \\ oh \mapsto SM(S, E, M, P).2(oh)\}$$

(SM-if)

Let $\mathcal{P}While$ be P ; while e do P done

$SM(S, E, M,$ while e do P done$) =$

$$\begin{cases} SM(S, E, M, \mathcal{P}While) & \textbf{if } eval(S.m, e) \wedge eval(M, e) \\ (SM(S, E, M, \mathcal{P}While).1, \\ \quad SM(S, E, M, \mathcal{P}While).2, M) & \textbf{if } eval(S.m, e) \wedge \neg eval(M, e) \\ (S, E', SM(S, E, M, \mathcal{P}While).3) & \textbf{if } \neg eval(S.m, e) \wedge eval(M, e) \\ (S, E, M) & \textbf{if } \neg eval(S.m, e) \wedge \neg eval(M, e) \end{cases}$$

where $E' = E \oplus \{ih \mapsto SM(S, E, M, \mathcal{P}While).2(ih),$
$oh \mapsto SM(S, E, M, \mathcal{P}While).2(oh)\}$

(SM-while)

A Noninterference Theorem

The model verifies the following theorem.

Theorem. $Noninterference$

$\forall S \in SState; E_1, E_2 \in Env; M_1, M_2 \in Memory; P \in Program \bullet$
P terminates
$\wedge S.dl = \{il \mapsto L, ol \mapsto L, ih \mapsto H, oh \mapsto H\}$
$\wedge E_1(il) = E_2(il)$
$\wedge E_1(ol) = E_2(ol)$
$\wedge SM(S, E_1, M_1, P) = (S_1', E_1', M_1')$
$\wedge SM(S, E_2, M_2, P) = (S_2', E_2', M_2')$
$\Longrightarrow E_1'(ol) = E_2'(ol)$

Conclusions an Future Work

The model is being implemented by Pablo Mata as his undergraduate research thesis (for LCC-FCEIA-UNR, Argentina) over Linux 2.6 as a security model using Linux Security Modules. Pablo will give a talk showing the current status of the implementation.

We plan to refine the model to specify the Linux's kernel interface.