Language-based security

- Data leakage
- Security policy
- Information flow in programs
- Examples of illegal flow of information

Data leakage

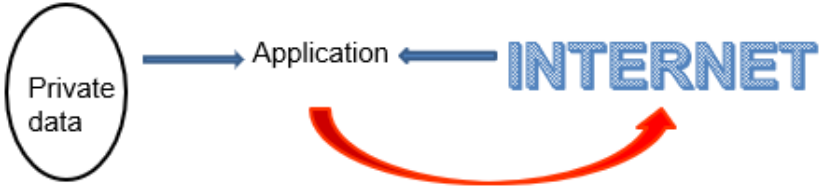
GENERAL DATA PROTECTION REGULATION(GDPR) - UE 2016/679

Regulation of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data

- explicit (private data made publicly available)
- interference between private and public data

Data leakage

Limit of Firewall and Access control mechanisms



Application authorized to access private data
Application authorized to access internet

Control on the information sent on the internet!!!!!

Certificate that the application does not send data that may reveal any private information

Certification of applications for secure information flow

Colluding apps

The Independent (British online newspaper)

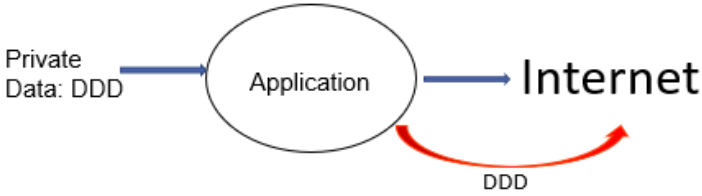
Taken from: http://www.independent.co.uk/life-style/gadgets-and-tech/news/android-app-steal-users-data-colluding-each-other-research-cartel-information-a7663976.html

The team reports that the types of app fall into two major categories / Justin Sullivan/Getty Images

The biggest security risks can come from some of the least capable apps

"Android apps are mining smartphone users data by secretly colluding with each other, according to a new study. Pairs of apps can trade information, a capability that can lead to serious consequences in terms of security."

Data leakage



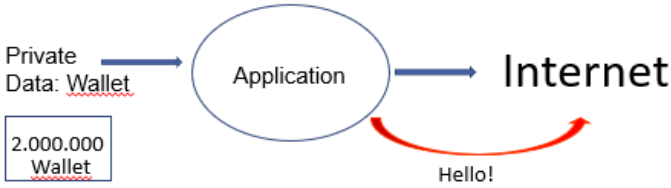
Secure information flow is violated

Application authorized to access private data
Application authorized to access Internet

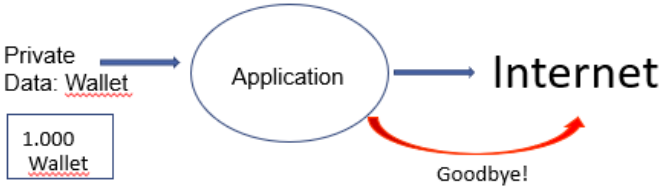
Explicit information flow

Control on the information sent on Internet!!!!!

Data leakage



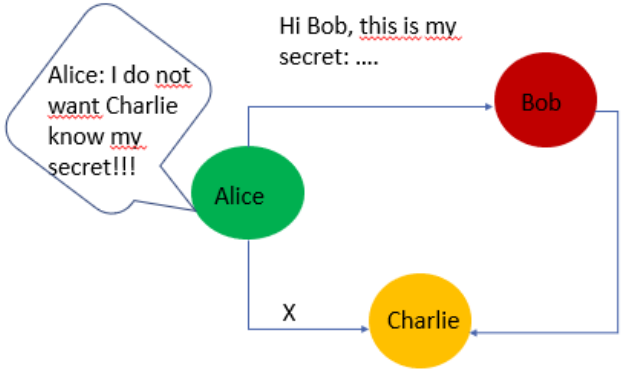
Secure information flow is violated



if Wallet > 1.000.000 then
«Hello!»
else «Goodbye!»

Implicit information flow

Data leakage



Secure information flow is violated

Data leakage

Can be studied by defining a security policy and by using the theory of information flow in programs.

Information flow in programs

Modular programming

Information flow occurs through

- simple variables, input/output files
- array, structures, objects
- pointers, references
- objects allocated in dynamic memory
- global variables
- function calls, parameters by value/ parameters by reference, return

Multilevel security policy

Multilevel Security policy: a security policy that allows the classification of data and users based on a set of hierarchical security levels.

Example:

$\mathcal{S} = \{Public, Private\}$

Private level is higher than Public level.

Multilevel security policy

Definition: A multilevel security policy $\mathcal{L}$ is a pair that consists of (i) a set of security levels $\mathcal{S}$ and (ii) an ordering relation $\sqsubseteq$ between the levels.
Moreover, every pair of elements in $\mathcal{S}$ has both:
a greatest lower bound (glb, $\sqcap$) and a least upper bound (lub, $\sqcup$).

$$\mathcal{L} = (\mathcal{S}, \sqsubseteq)$$

The relation $\sqsubseteq$ is reflexive and transitive. Moreover, $\sqsubseteq$ is antisymmetric.

$(\mathcal{S}, \sqsubseteq)$ is a **lattice** of security levels.

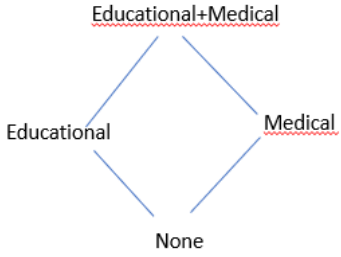
Example:
$\mathcal{L} = (\mathcal{S}, \sqsubseteq)$
$\mathcal{S} = \{\mathit{Public}, \mathit{Private}\}$
$\sqsubseteq$ defined as follows: $\mathit{Public} \sqsubseteq \mathit{Private}$.

Multilevel security policy

Example: *Educational* and *Medical* are sensitive classes of information of a user.

$\mathcal{S} = \{None, Educational, Medical, Educational + Medical\}$, with $\sqsubseteq$ defined as:
$None \sqsubseteq Educational$; $None \sqsubseteq Medical$;
$Medical \sqsubseteq Educational + Medical$; $Educational \sqsubseteq Educational + Medical$

least upper bound ($\sqcup$): $Educational \sqcup Medical = Educational + Medical$

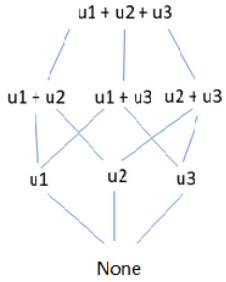


Multilevel security policy

Let u_i represents sensitive information of user i.

$\mathcal{S} = \{None, u1, u2, u3, u1 + u2, u1 + u3, u2 + u3, u1 + u2 + u3\}$ with
$None \sqsubseteq ui$;
$u_i \sqsubseteq ui + u_j, j \neq i$;
$u_i + u_j, j \neq i \sqsubseteq u1 + u2 + u3$

least upper bound ($\sqcup$): $u1 \sqcup u2 = u1 + u2$



We assume a security policy $\mathcal{L} = (\mathcal{S}, \sqsubseteq)$ such that:

- $\mathcal{S} = \{l, h\}$
- $\sqsubseteq$ defined as: $l \sqsubseteq h$

Input and output of a program are assigned either low level of security (l) or high level of security (h)

Secure Information Flow property: the low output do not reveal information on the high level input. Low output are not assigned high level data.

Non-interference property

Non-interference property: the security domain private is non-interfering with domain public if no input by private can influence subsequent outputs that can be seen by public.

A program has the non-interference property if and only if any sequence of low inputs will produce the same low outputs, regardless of what the high level inputs are.

The program responds in exactly the same manner on low outputs whether or not high sensitive data are changed.

The low user will not be able to acquire any information about data of the high user.

High-level language. Let x, y be variables

```
y := x;   explicit flow
```

variable y is assigned the value of x, there is an explicit flow from x to y

```
if   (x = 0)        implicit flow
     then y:=1;
     else y:=0;
```

there is an implicit flow from variable x to y, since y is assigned different values depending on the value of the condition of the control instruction (variable x)

In both cases observing the final value of y reveals information on the value of x.

A conditional instruction in a program causes the beginning of an implicit flow. The implicit flow begins when the conditional instruction starts (we say that we have an opened implicit flow); all the instructions in the scope of the if depend on the condition of the if.

If a function call is executed in the scope of a conditional instruction, the function is executed under the implicit flow.

```
if   (y < 0)
     then f();
```

Function f() is invoked depending on the value of variable y.

Instructions of f() are executed under the implicit flow of the condition of the if statement.

Abstract interpretation of the operational semantics for secure information flow in programs

- instrumented semantics that add the the security level to data and traces the information flow (enhanced semantics)
- abstract semantics that abstract from real value and execute the program on security level. consider only taking only the security level of data
- correctness of the abstraction

Enhanced Operational semantics

Given a program $P = \langle c, H, L \rangle$ and an initial memory $m \in \mathcal{M}^{\epsilon}_{Var(c)}$

$E(P, m)$ is the transition system defined by $\longrightarrow^{\epsilon}$ starting from the initial state $\langle c, m \rangle$.

We enrich the standard operational semantics, in such a way that a violation of security can be discovered.

Enhanced operational semantics

The enhanced semantics is an instrumented semantics which:

- Handles values (k, σ) annotated with a security level ($k = 0, 1, 2 \cdots$ and $\sigma \in \mathcal{S}$).
- Executes instructions under a security environment $\sigma \in \mathcal{S}$.
- C(P, M): enhanced transition system for $P = \langle c, H, L \rangle$, with $M(x) = (k, \sigma)$, for variable x.

Enhanced operational semantics

annotated value (k, σ)
during the execution, σ indicates the least upper bound of the security levels of the information flows, both explicit and implicit, on which k depends.

execution environment σ: $(e)^{\sigma}$ and $(c)^{\sigma}$
during the execution, σ represents the least upper bound of the security levels of the open implicit flows. σ is (possibly) upgraded when a branching instruction begins and is (possibly) downgraded when all branches join.

Enhanced Operational semantics

$exp ::= const \mid var \mid exp\ op\ exp$
$com ::= var := exp \mid \texttt{if}\ exp\ \texttt{then}\ com\ \texttt{else}\ com \mid$
$\texttt{while}\ exp\ \texttt{do}\ com \mid com\ ;\ com \mid \texttt{halt}$

Let $(\mathcal{S}, \sqsubseteq)$, with $\mathcal{S} = \{l, h\}$, be a lattice of security levels, ordered by $l \sqsubseteq h$, where $\sqcup$ denotes the least upper bound between levels.

A program P is a triple $\langle c, H, L \rangle$
$c \in com$
H are the high variables of P
L are the low variables of P
$H \cup L = Var(c)$ and $H \cap L = \emptyset$

Enhanced Operational semantics

$$\textbf{Expr}_{const} \quad \frac{}{\langle k^{\sigma}, M\rangle \longrightarrow_{expr} (k, \sigma)}$$

$$\textbf{Expr}_{var} \quad \frac{M(x) = (k, \tau)}{\langle x^{\sigma}, M\rangle \longrightarrow_{expr} (k, \sigma \sqcup \tau)}$$

$$\textbf{Expr}_{op} \quad \frac{\langle e_1^{\sigma}, M\rangle \longrightarrow_{expr} (k_1, \tau_1) \;\; \langle e_2^{\sigma}, M\rangle \longrightarrow_{expr} (k_2, \tau_2)}{\langle (e_1 \; op \; e_2)^{\sigma}, M\rangle \longrightarrow_{expr} (k_1 \;\; op \; k_2, \tau_1 \sqcup \tau_2)}$$

$$\textbf{Ass} \quad \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} v}{\langle (x{:=}e)^{\sigma}, M\rangle \longrightarrow M[v/x]}$$

Rules description

- The rules compute the security level of the value of an expression dynamically using both the security level of the operands and the security level of the environment.

  For example, an integer constant k results in the value (k, σ), where σ is the security level of the environment under which k is evaluated.

Enhanced Operational semantics

$$\textbf{If}_{true} \quad \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} (true, \tau)}{\langle (\texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2)^{\sigma}, M\rangle \longrightarrow \langle c_1^{\tau}, Impl(M, Mod(c_1; c_2), \tau)\rangle}$$

$$\textbf{If}_{false} \quad \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} (false, \tau)}{\langle (\texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2)^{\sigma}, M\rangle \longrightarrow \langle c_2^{\tau}, Impl(M, Mod(c_1; c_2), \tau)\rangle}$$

$Mod(c)$ finds the set of variables *mod*ified in the sequence of instructions c i.e. those which are on the left of an assignment

Impl upgrades the security level of the values of the previous variables to take into account an *impl*icit flow.

Rules description

- Assume that the condition of an `if` command results in a value (k, τ).

  The branch c_1 or c_2, selected according to k (*true* or *false*), is executed in the memory $Impl(M, Mod(c_1; c_2), \tau)$ under the environment τ.

  In particular, if $\tau = h$, the value of every variable assigned in at least one of the two branches is upgraded to h and the selected branch is executed in a high environment.

  When the conditional command terminates, the security environment is reset to the one holding before the execution of the command

$$\textbf{While}_{true}\quad \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} (true, \tau)}{\langle (\texttt{while}\ e\ \texttt{do}\ c)^{\sigma}, M\rangle \longrightarrow \langle (c;\texttt{while}\ e\ \texttt{do}\ c)^{\tau}, \mathit{Impl}(M, \mathit{Mod}(c), \tau)\rangle}$$

$$\textbf{While}_{false}\quad \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} (false, \tau)}{\langle (\texttt{while}\ e\ \texttt{do}\ c)^{\tau}, M\rangle \longrightarrow \langle \mathit{Impl}(M, \mathit{Mod}(c), \tau)\rangle}$$

- The `while` command is handled similarly to the conditional command.

$$\textbf{halt}\quad \frac{}{\langle \texttt{halt}^{\sigma}, M\rangle \longrightarrow \langle M\rangle}$$

$$\textbf{Seq}_1\quad \frac{\langle c_1^{\sigma}, M\rangle \longrightarrow \langle M'\rangle}{\langle c_1^{\sigma}; w, M\rangle \longrightarrow \langle w, M'\rangle}$$

$$\textbf{Seq}_2\quad \frac{\langle c_1^{\sigma}, M\rangle \longrightarrow \langle w', M'\rangle}{\langle c_1^{\sigma}; w, M\rangle \longrightarrow \langle w'; w, M'\rangle}$$

- The **w** command is the continuation of the program.

Enhanced Operational semantics

Given a program $P = \langle c, H, L \rangle$ and an initial enhanced memory $M \in \mathcal{M}_{Var(c)}$, the rules define a transition system $C(P, M)$, which is the enhanced semantics of the program.

We assume that the program starts with a low security environment.

The initial state of $C(P, M)$ is: $\langle c^l, M \rangle$.

Secure memory. We introduce the definition of a memory safe for a program:

given a program $P = \langle c, H, L \rangle$, an enhanced memory $M \in \mathcal{M}_{Var(c)}$ is *secure* for P if and only if each low variable of P holds a low value in M.

An example

Let be given the program $P1\langle c, H, L\rangle$ and the enhanced memory M with $M(x) = (1, l)$ and $M(y) = (2, h)$.

$$P1 = \langle \texttt{if y = 0 then x := 0 else x := 1}, \{\texttt{y}\}, \{\texttt{x}\}\rangle$$

The memory in the final state of the transition system is not safe for $P1$, because the security level of $\texttt{x}$ is h:

$$\langle (\texttt{if y = 0 then x := 0 else x := 1})^{l}, [x : (1, l), y : (2, h)]\rangle$$

$$\downarrow$$

$$\langle (\texttt{x := 1})^{h}, [x : (1, h), y : (2, h)]\rangle$$

$$\downarrow$$

$$\langle [\mathbf{x} : (\mathbf{1}, \mathbf{h}), y : (2, h)]\rangle$$

An example

Let be given the program $P2\langle c, H, L\rangle$ and the enhanced memory M with $M(\mathtt{x}) = (1, l)$, $M(\mathtt{y}) = (2, h)$ and $M(\mathtt{z}) = (0, h)$.

$P2 = \langle \mathtt{if\ x} = 1\ \mathtt{then\ y} := \mathtt{x\ else\ z} := 1;\ \ \mathtt{x} := \mathtt{y}, \{\mathtt{y}, \mathtt{z}\}, \{\mathtt{x}\}\rangle$

The assignment `y:=x` assigns a low value to `y`. Thus the assignment `x:=y` assigns a low value to `x`. The memory in the final state is secure for $P2$.

An example

The transition system is the following:

$$\langle (\texttt{if x} = 1 \texttt{ then y} := \texttt{x else z} := 1)^l; (\texttt{x} := \texttt{y})^l, [x : (1, l), y : (2, h), z : (0, h)] \rangle$$

$$\downarrow$$

$$\langle (\texttt{y} := \texttt{x})^l; (\texttt{x} := \texttt{y})^l, [x : (1, l), y : (2, h), z : (0, h)] \rangle$$

$$\downarrow$$

$$\langle (\texttt{x} := \texttt{y})^l, [x : (1, l), y : (1, l), z : (0, h)] \rangle$$

$$\downarrow$$

$$\langle [\mathbf{x} : (\mathbf{1}, \mathbf{l}), y : (1, l), z : (0, h)] \rangle$$

An example

Note that if the value of x in the initial memory is equal to 0 (instead of 1 as in the example above) the memory in the final state is not secure for $P2$ (in the final state $\mathtt{x}$ holds a high value)

$$\langle (\mathtt{if\ x = 1\ then\ y := x\ else\ z := 1})^{l};\ (\mathtt{x := y})^{l}, [x : (0, l), y : (2, h), z : (0, h)] \rangle$$

$$\downarrow$$

$$\langle (\mathtt{z := 1})^{l}; (\mathtt{x := y})^{l}, [x : (0, l), y : (2, h), z : (0, h)] \rangle$$

$$\downarrow$$

$$\langle (\mathtt{x := y})^{l}, [x : (0, l), y : (2, h), z : (1, l)] \rangle$$

$$\downarrow$$

$$\langle [\mathbf{x} : (\mathbf{2}, \mathbf{h}), y : (2, h), z : (0, h)] \rangle$$

Abstract Operational semantics

The enhanced transition system could be infinite, because there are infinitely many memories.

The enhanced operational semantics cannot be used as a static analysis tool.

The purpose of abstract interpretation (or abstract semantics) is to correctly approximate the enhanced semantics of all executions in a finite way.

Abstract Operational semantics

- The first step in the construction of the abstract semantics is the definition of the abstract domains.
- The nodes of the abstract transition system contain abstractions of states.
- In particular, in our abstract semantics each enhanced value, composed of a pair of a value and a security level, is approximated by considering only its security level.
- As a consequence, when dealing with conditional or iterative commands, the abstract transition system has multiple execution paths due to the loss of precision of abstract data.

Abstract Operational semantics

Let α the abstraction function. The abstract semantics:

- abstracts enhanced values into their security level: $\alpha(k, \sigma) = \sigma$
- uses the same rules of the enhanced semantics on the abstract domains. The transition relation of the abstract semantics is denoted by $\longrightarrow^{\natural}$.
- Both rules for *if* are always applied, since *true* and *false* are both abstracted to "$\cdot$"
- Both rules for *while* are always applied, since *true* and *false* are both abstracted to "$\cdot$"

Abstract Operational semantics

The abstract semantics:

- let A(P, $M^\sharp$) be abstract transition system for P
  - finite
  - multiple paths
  - each path of C(P, M) is correctly abstracted onto a path of A(P, $M^\sharp$)

Abstract transition system

$P2 = \langle \texttt{if x = 1 then y := x else z := 1;x := y}, \{\texttt{y}, \texttt{z}\}, \{\texttt{x}\}\rangle$
$M^\sharp(\texttt{x}) = (l)$, $M^\sharp(\texttt{y}) = (h)$ and $M^\sharp(\texttt{z}) = (h)$

$$\langle (\texttt{if x = 1 then y := x else z := 1})^l; (\texttt{x := y})^l, [x : (l), y : (h), z : (h)]\rangle$$

$$\downarrow^\sharp$$

$$\langle (\texttt{y := x})^l; (\texttt{x := y})^l, [x : (l), y : (h), z : (h)]\rangle \qquad \langle (\texttt{z := 1})^l; (\texttt{x := y})^l, [x : (l), y : (h), z : (h)]\rangle$$

$$\downarrow^\sharp \qquad\qquad \downarrow^\sharp$$

$$\langle (\texttt{x := y})^l, [x : (l), y : (l), z : (h)]\rangle \qquad \langle (\texttt{x := y})^l, [x : (l), y : (h), z : (l)]\rangle$$

$$\downarrow^\sharp \qquad\qquad \downarrow^\sharp$$

$$\langle [\mathbf{x} : (\mathbf{l}), y : (l), z : (h)]\rangle \qquad \langle [\mathbf{x} : (\mathbf{h}), y : (l), z : (l)]\rangle$$

Abstract Operational semantics

Let $P = \langle c, H, L \rangle$ and $M^\sharp \in \mathcal{M}^\sharp_{Var(c)}$ with
$M^\sharp(x) = l, \forall x \in L$ and $M^\sharp(x) = h, \forall x \in H$.

If for each final state $\langle M_{f_i}^\sharp \rangle$ of $A(P, M^\sharp)$, it holds that $M_{f_i}^\sharp(x) = l$, then Secure Information Flow property is satisfied.

The memory in each final state of the abstract transition system is secure for P.

For each program $P = \langle c, H, L \rangle$ and abstract memory $M^\natural \in \mathcal{M}^\natural_{Var(c)}$, $A(P, M^\natural)$ is finite.

To check if a program is secure, we build the abstract transition system and examine all final states.

For example, the abstract transition system of the program $P2$ has two final states: $\langle [\mathbf{x} : \mathbf{l}, y : l, z : h] \rangle$ and $\langle [\mathbf{x} : \mathbf{h}, y : h, z : l] \rangle$. Secure Information Flow property is not satisfied because $x \in L$ and $x = h$ in the second state.

Secure Information Flow (SIF). A program P has secure information flow if in each final state of A(P), each $x : \sigma$ holds a value $\tau \sqsubseteq \sigma$.

This approach is based on a finite-state transition system and thus has the advantage of being fully automatic.

Exercise

Apply the standard operational semantics, the enhanced and the abstract operational semantics to the following program

$P = \langle c1; c2; \cdots ; c5, \{x\}, \{y, z\}\rangle$
with $m(x) = 2$, $m(y) = 7$, $m(z) = 3$

```
c1:  z := 0;
c2:  while    (x > 0)
c3:           y:=y*10;
c4:           x:=x-1;
c5:  z:=y;
```

Does P satisfy SIF? Why?

While statement

In the repetition of the while, new abstract memories can be generated as possible input to the while statement.

$PowerSet(Mem)$, is the set of collections of abstract memories.

The analysis of the program is an assignment of abstract memories with each node in the program graph

$$Assign : Q \rightarrow PowerSet(Mem)$$

Standard semantics

$EP(P, m)$ $m = [x : 2, y : 7, z : 3]$

$$\langle (c1; c2; c3; c4; c5), [x : 2, y : 7, z : 3] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c2; c3; c4; c5), [x : 2, y : 7, z : 0] \rangle$$
$$\downarrow_{while_{true}}$$
$$\langle (c3; c4; (c2; c3; c4); c5), [x : 2, y : 7, z : 0] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c4; (c2; c3; c4); c5), [x : 2, y : 70, z : 0] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c2; c3; c4); c5), [x : 1, y : 70, z : 0] \rangle$$
$$\downarrow_{while_{true}}$$
$$\langle (c3; c4); (c2; c3; c4); c5), [x : 1, y : 70, z : 0] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c4; (c2; c3; c4); c5), [x : 1, y : 700, z : 0] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c2; c3; c4); c5), [x : 0, y : 700, z : 0] \rangle$$
$$\downarrow_{while_{false}}$$
$$\langle (c5), [x : 0, y : 700, z : 0] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (), [x : 0, y : 700, z : 700] \rangle$$

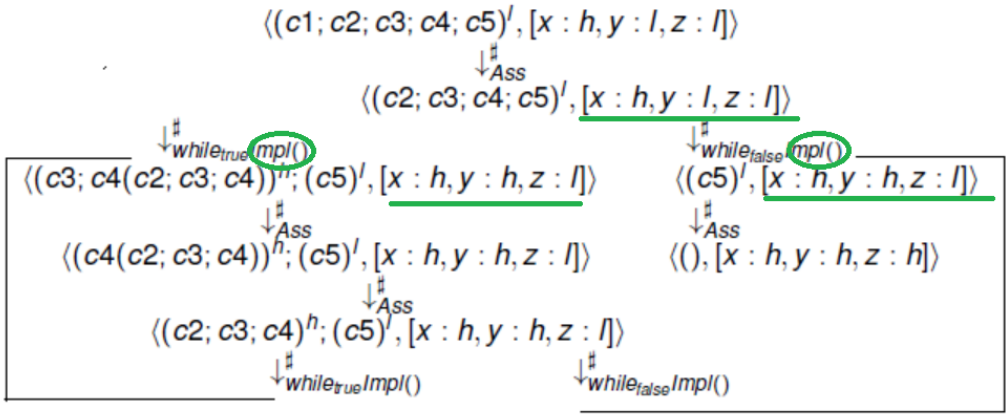
Concrete semantics

$CP(P, M)\ m = [x : (2, h), y : (7, l), z : (3, l)]$

$$\langle (c1; c2; c3; c4; c5)^l, [x : (2, h), y : (7, l), z : (3, l)] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c2; c3; c4; c5)^l, [x : (2, h), y : (7, l), z : (0, l)] \rangle$$
$$\downarrow_{while_{true}Impl()}$$
$$\langle (c3; c4; (c2; c3; c4))^h; (c5)^l, [x : (2, h), y : (7, h), z : (0, l)] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c4; (c2; c3; c4))^h; (c5)^l, [x : (2, h), y : (70, h), z : (0, l)] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c2; c3; c4)^h; (c5)^l, [x : (1, h), y : (70, h), z : (0, l)] \rangle$$
$$\downarrow_{while_{true}Impl()}$$
$$\langle (c3; c4; (c2; c3; c4))^h; (c5)^l, [x : (1, h), y : (70, h), z : (0, l)] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c4; (c2; c3; c4))^h; (c5)^l, [x : (1, h), y : (700, h), z : (0, l)] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (c2; c3; c4)^h; (c5)^l), [x : (0, h), y : (700, h), z : (0, l)] \rangle$$
$$\downarrow_{while_{false}Impl()}$$
$$\langle (c5)^l, [x : (0, h), y : (700, h), z : (0, l)] \rangle$$
$$\downarrow_{Ass}$$
$$\langle (), [x : (0, h), y : (700, h), z : (700, h)] \rangle$$

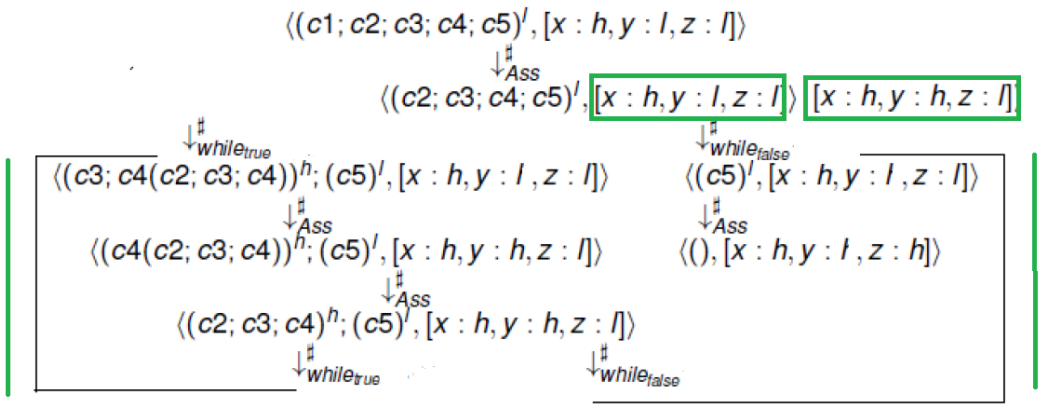
Abstract semantics

$AP(P, M^\sharp)$
$M^\sharp = [x : h, y : l, z : l]$



Abstract semantics

Funtion *Impl*() reduces the number of iterations of the analysis.
Assume *Impl*() is not applied. The memory input to while changes, the while statement is re-executed.
The levels in the old memory are $\sqsubseteq$ of the levels in the new memory.
Since the set of security level is finite, the number of memories that can be assigned to a state are finite.



Abstract semantics

A variant of the previous program.
The semantics is the same, but a new variable is introduced (w).
$P = \langle c1; c2; \cdots ; c5, \{w\}, \{x, y, z\}\rangle \quad m(x) = 2,\ m(y) = 7,\ m(z) = 3,\ m(w) = 1$

```
c1:  z := 0;
c2:  while   (x > 0)
c3:          y:=y*10;
c4:          x:=x-w;
c5:  z:=y;
```

While executed in a low environment (x is low). *Impl*() is applied.
In the body of the while x is assigned an expression that depends on a high value; the memory changes. While is executed starting on a new abstract memory.

Secure information flow

A variable can represent

- input/output file
- a port for network connections
- .....

PIN cloner

The code catches the private information of the userâs Personal Identification Number (PIN) without directly assigning it to the public variable clone. It achieves this by using a mask that reveals the value of each bit of the PIN.

$P = \langle Pincloner, H, L \rangle$ $H = \{PIN\}$ $L = \{clone\}$

```
input : PIN
output : clone
clone := 0x0000;
mask := 0x0001;
while(mask > 0)
   b :=PIN & mask;
   if (b ! = 0)
      clone:=clone || mask;
   mask := mask << 1;
```

PIN cloner

- Initially the mask has all the bits set to 0 except for the least significant bit, which is set to 1: this bit shifts one step to the left after each loop cycle and clones the value of one bit of the PIN during each cycle.
- In particular, the direct assignment of the PIN bits to the clone variable is avoided by using a variable b. This last variable is different from 0 if and only if the i-th bit of the PIN and of the mask are both equal to 1: the value of b can be used to set (or not set) the i-th bit of the clone variable.
- Once the program has gained the access to the userâs private data, the access control mechanism is not able to reveal the illicit flow

Other properties

SIF analyses the variables when the program terminates.

Other possibilities:

- check the variables at given program points
- check properties of execution paths
- .....

Correctness of the analysis

In the definition of the abstract semantics, we have applied *Abstract Interpretation*.

Abstract interpretation is a widely applied method for designing approximate semantics of programs.

P. Cousot, R. Cousot. Abstract interpretation frameworks. Journal of Logic and Computation, 2, 1992