

INFORMATION FLOW

in questa lezione abbiamo definito un'analisi statica, la taint analysis che con un meccanismo di analisi simbolica in avanti identificava i flussi all'interno del comportamento di un programma dove delle sorgenti di informazione corrotte, quindi non affidabile, controllate dall'attaccante potevano avere un effetto sul comportamento del programma. La strategia dell'analisi è quella di essere sempre più precisa di tracciare come i dati non corrotti fluiscono all'interno delle possibili esecuzioni del programma e abbiamo visto diverse tecniche che avevano a che vedere sulla modalità di rendere l'analisi più precisa dal fatto che si voglia mettere delle path conditions per poter inserire dei vincoli relativamente ai cammini di esecuzione fino alle tecniche di single state assignment che permettevano di avere delle forme di programmi più facilmente manipolabili, ecc... L'idea dell'analisi era quindi come affrontare la propagazione dei dati corrotti, in modo tale da avere un insieme di vincoli e determinare la soluzione di questo insieme di vincoli che definivano le dipendenze tra i dati che venivano utilizzati all'interno del programma e quando si trovava una soluzione che mostrava che ci fosse un potenziale uso di un dato corrotto allora si era scoperta una vulnerabilità. Abbiamo detto che questo tipo di analisi riesce a trattare quelle situazioni in cui abbiamo un comportamento che dipende da dati che possono essere controllati dall'attaccante, quindi riesce a affrontare le problematiche tipo ad esempio SQL injection e abbiamo anche fatto vedere dove ci sono degli aspetti che hanno a che vedere col flusso implicito di dati, cioè di come l'informazione ma non il dato fluisce all'interno del programma che si possono trattare al prezzo di rendere l'analisi molto più complicata e molto più articolata. Abbiamo anche detto che l'aspetto diciamo di dilemma, quando uno affronta questi aspetti, è rendere l'analisi più precisa in modo da rendere l'analisi scalabile e quindi dato che una analisi poco precisa è tipicamente dell'ordine di $O(n)$ dunque una worklist abbastanza semplice un'analisi più precisa comincia a essere $O(n^3)$, quindi bisogna affrontare questi aspetti quando si sviluppa il backend del compilatore.

Where we were ... taint analysis

- Track information flow through a program at static time
- **Identify sources of taint**
 - Untrusted input (controlled by the attacker)
- **Identify Propagation of taint**
 - Tracks flow that manipulates data in order to determine whether the result is tainted or not
- *If the analysis finds that tainted data could be used where untainted data is expected, there could be a potential security vulnerability*

Where we were ...

- The analysis tracks sensitive “tainted” information through the application by starting at a pre-defined **source** (e.g. an API method) and then following the data flow until it reaches a given **sink** (e.g. a method writing the information to a socket), giving precise information about which data may be leaked where.



Allowed flow as a
lattice **untainted** < **tainted**

FLOW AS A LATTICE

Una delle caratteristiche della taint analisi è quella di dire il flusso di un dato potenzialmente corrotto segue un'informazione data da un ordinamento parziale e infatti noi abbiamo utilizzato questo ordinamento parziale, cioè che untainted è minore di tainted è un ordinamento parziale in realtà, come vedremo oggi, quello che noi ci interessa avere è avere un reticolo completo. Un reticolo completo è un insieme in cui è definita una relazione d'ordine quindi abbiamo una nozione di ordinamento, quindi una relazione che è riflessiva, antisimmetrica e transitiva e ha però altre due caratteristiche: altre due caratteristiche sono che ha un elemento minimo e un elemento massimo e per ogni coppia di elementi che appartengono all'insieme si riesce a determinare l'estremo inferiore e l'estremo superiore e questo è cosa vuol dire essere un reticolo. Un reticolo è la struttura che emerge naturalmente quando uno vuole avere delle informazioni che descrivono delle proprietà che ad un certo punto uno vuole avere all'interno del flusso di esecuzione del programma. L'esempio che noi abbiamo visto era untainted che è minore di tainted e l'idea che la relazione di ordinamento è il meccanismo che viene usato per misurare il livello di sicurezza della proprietà che stiamo considerando

Information flow

- Protecting confidentiality is preventing **private information** from leaking through computation.
- Access control mechanisms do not help with this kind of leak, since they only control information release, not its propagation once released.

allora quello che adesso noi vogliamo affrontare in questa lezione è un aspetto duale rispetto alla propagazione dei dati corrotti, cioè vogliamo affrontare il problema di quando un programma fa dei calcoli vogliamo evitare che il programma nell'effettuare calcoli butti fuori nell'ambiente ostile, quindi butti fuori all' attaccante dell'informazione sensibile, segreta o informazione che deve essere protetto. La taint analysis affronta un problema di integrità dei dati, cioè vuole evitare che ci sia una computazione in cui dati non integri abbiano un gioco. L' information flow invece vuole affrontare un problema duale della confidenzialità , vuole evitare che dati che devono rimanere segreti vengono sbattuti fuori, quindi questo è l'aspetto che differenzia. Sono problemi simili, ma uno affronta un aspetto dell'integrità l'altro un aspetto di confidenzialità in entrambi i casi, tuttavia il controllo degli accessi, seppur fatto anche a grana fine, come viene fatto nei linguaggi di programmazione non dà il controllo sufficiente perché dice come io accedo, quali sono i diritti che ho di accesso ma non fa vedere come fluisce l'informazione una volta che un' entità ha avuto accesso.



Readings

- Chapter 5 Erik Pool Notes (available on teams)
- Notes on Information Flow (available on teams)
- Andrei Sabelfeld, Andrew C. Myers: Language-based information-flow security. IEEE J. Sel. Areas Commun. 21(1): 5-19 (2003)

allora il problema di esaminare come fluisce il flusso all'interno dei linguaggi all'interno dei sistemi, il flusso ovviamente di informazione non è un problema che abbiamo affrontato ora in questo secolo, per via della sicurezza, un problema noto e che è stato studiato con attenzione già nel secolo scorso, allora in modo particolare, Lampson nel '71 ha scritto questo lavoro dove sostanzialmente fa un'analisi di come fluisce informazione, di come, in modo particolare ci possano essere dei leaks di informazioni dovute a dei medium di comunicazione che sono nascosti e in modo particolare introduce l'idea di covert channel e fa vedere come si può confinare il flusso dell'informazione all'interno di un sistema per evitare che ci sia del leakage.

Lampson (1971)

- “A Note on the confinement problem”
- Lampson explored the problem of information leak and introduces the category of **covert channels** as those that are not intended for information transfer at all.
- Paper available on Teams

un altro lavoro significativo è il lavoro di Bell-Lapadula del '76 era il periodo in cui si stavano sviluppando i moderni sistemi operativi, l'idea dice di questo lavoro di fatto sta già introducendo un reticolo, ma non lo dice esplicitamente, dice che all'interno di un sistema uno potrebbe identificare quattro livelli sicurezza che public, confidential, secret, top secret, e di fatto introduce due politiche, quelle che si chiamano NO READ UP vuol dire che se io sono a livello basso pubblico, non posso leggere delle informazioni top secret ma in realtà introduce anche un'altra proprietà, quelle si chiamano NO WRITE DOWN cioè quella duale, cioè quello dei livelli più alti non posso nemmeno scrivere nei livelli più bassi, allora questa è stata un po' controversa, infatti, ad esempio, in modo particolare il flusso di informazione dall'alto verso il basso può fluire attraverso quelli che si chiamano i soggetti trusted

Bell-Lapadula (1976)

- “Secure Computer System: Unified exposition and multics interpretation.”
- They introduced a computing system with different security levels: **public, confidential, secret, and top secret.**
- The rule is that no **read up** and (more controversially) no **write down** can happen on the more secret level.
 - information can flow from a public level to the a confidential level, but not the other way around.
- The transfer of information from a a high low level may happen via the concept of trusted subjects. Trusted Subjects are not restricted by the write down property.
- Paper available on Teams

a questo punto andiamo agli articoli di denning e gli articoli di denning l' affronta in modo formalmente precisa, cioè quello che dice è che la struttura matematica dei reticoli, ovvero di quelle strutture algebriche che abbiamo brevemente descritto in precedenza e il modo di descrivere e di modellare flusso di informazioni all'interno di programmi e del modo di caratterizzare le politiche che regolano il flusso all'interno del programma a questo punto formalizza proprio l'idea che l' approccio di bell lapadula è dato da quel lattice che vedete, in realtà è una catena, vede che public minor uguale di confidential che è minore uguale di secret che è minore uguale di top secret che è esattamente il modo in cui abbiamo introdotto in precedenza e Denning fa anche vedere come possiamo associare anche dell informazione relativa ai dati che vengono trasmessi sui canali di informazione, in modo particolare, si vede che siamo in periodo di guerra fredda, si vede bene che siamo nel periodo in cui i dati sugli armamenti nucleari sono dati particolarmente significativi e vi ricorda a tutti che ARPANET che da strutture di interconnessione che ha preceduto internet come conosciamo noi in questo periodo era stata introdotta esattamente per avere un controllo remoto sui lanci dei missili intercontinentali che avevano al loro interno testate nucleari in un altro articolo Denning e la moglie fanno vedere come l'aspetto che ora noi stiamo studiando ed essenziale in questi giorni, quindi hanno avuto una visione del futuro molto molto chiara e che questo aspetto di avere un modello del flusso di informazione fosse essenziale per la certificazione di proprietà di sicurezza dei compilatori, ovvero quello che stiamo facendo oggi l'esigenza che oggi è significativa

Denning (1976)

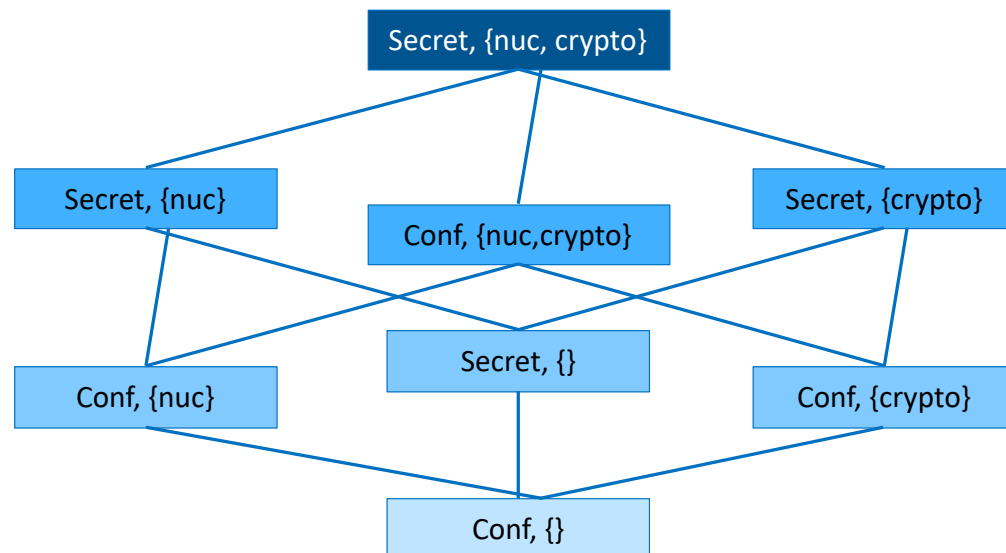
- “A lattice model of secure information flow.”
- This paper recognises that the mathematical structure of **lattices** characterizes the relation among policies.
- The Bell-Lapadula approach is based on the lattice
 - **public \sqsubseteq confidential \sqsubseteq secret \sqsubseteq top-secret**
- The subject of a secret data can be incorporated within the lattice
 - **(secret, {nuclear}) \sqsubseteq (secret; {nuclear; terror})**
- Paper available on Teams

SECURITY POLICIES AS LATTICES

il punto è che uno vuole modellare politiche di sicurezza di information flow in termini di reticolo, in modo particolare, il reticolo semplice è quello che abbiamo un livello basso e un livello alto l'ordine è che il basso è più piccolo del livello alto, ma potremmo avere dei reticoli più articolati.



SECURITY POLICIES AS LATTICES



ad esempio questo è un altro lattice vedete, avete un livello di confidenzialità e poi avete i segreti, poi avete ad esempio che il livello di confidenzialità può avere delle informazioni su dati nucleari, eccetera, sino al livello più alto del segreto che prendi dati nucleari, poi si possono scambiare delle informazioni cifrate, quindi avete un modo di rappresentare e i colori diversi che si va ad un livello più chiaro a quello più scuro, definiscono in modo implicito la nozione di ordinamento. Quindi vedete si ha davvero un controllo del flusso di informazioni, di un modello del flusso informazioni in base a reticolo.

Quindi il reticolo di riferimento è quello che abbiamo utilizzato durante la taint analysis, untaint minore di taint questo cosa vuol dire? vuol dire che nel caso dell'integrità gli input sono pericolosi perché uno vuole identificare dove l'attaccante può provocare dei danni corrompendo dei dati e come fluiscono questi dati corrotti all'interno nel caso della confidenzialità, invece, uno usa un problema duale che non è tanto che cosa io ho in ingresso ma che cosa fluisce fuori, cioè fare in modo che l'informazione che viene sbattuta fuori al controllo dell'attaccante non contenga dati riservati, quindi sono due problemi vicini ma duale

Discussion: integrity vs confidentiality

- Integrity of information is affected by flow.
- Trusted information is of higher integrity, so the flow of information is always from trusted information to untrusted information
 - The lattice **untainted** \leq **tainted**
- .This depicts the duality between trusted and confidentiality.
- **Inputs** are dangerous for **integrity**,
- **outputs** are dangerous for **confidentiality**

Examples (confidentiality)

String hi; // security level **secret**

String lo; // security label **public**

Which program fragments (may) cause problems if *hi* has to be kept confidential?

1. hi = lo;
2. lo = hi;
3. lo = "1234";
4. hi = "1234";
5. println(lo);
6. println(hi);
7. readln(lo);
8. readln(hi);

supponiamo di avere un livello di sicurezza 2, un reticolo di sicurezza due livelli, abbiamo la stringa hi segreta e la stringa lo public quindi vuol dire che lo è minore di hi e a questo punto ci domandiamo quali sono i problemi di confidenzialità (e non di integrità) che sono causati da questi 4 assegnamenti e 4 operazioni primitive? Stiamo cercando di comprendere a questo punto il problema della confidenzialità, ovvero se c'è un flusso di informazione avendo in mente che ci sono due livelli pubblico e segreto, ovvero che fuori non vada quindi alla fine dell'esecuzione di questo frammento di programma, in questo caso questi assegnamenti non vada fuori dell'informazione che deve essere mantenuto segreto.

Examples (confidentiality)

String hi; // security level **secret**

String lo; // security label **public**

Which program fragments (may) cause problems if *hi has to be kept confidential*?

1. **hi = lo;** OK
2. **lo = hi;** KO
3. **lo = "1234";** OK
4. **hi = "1234";** ??
5. **println(lo);** OK
6. **println(hi);** KO
7. **readln(lo);** OK
8. **readln(hi);** ??

hi uguale al lo è disco verde perché può fluire informazione dal livello pubblico a livello segreto quindi vuol dire che se io eventualmente metto dell'informazione in lo è un informazione che ha quanto meno lo stesso livello di aspettative di sicurezza di quella più bassa, infatti public è minore o uguale di secret. Se invece io faccio l'assegnamento di hi a lo questo dal punto di vista del flusso di informazione non va bene perché l'informazione pubblica poi potrebbe essere utilizzata e l'attaccante può vederla e quindi vuol dire che in questo modo prende dell'informazione che doveva rimanere confidenziale per questo il disco rosso ko. lo uguale alla stringa "1234" sicuramente l'informazione di un dato primitivo ha disco verde e non è significativa. Questionabile, nel senso che dipende e per questo c'è i due punti interrogativi se hi uguale "1234" è questionable perché dipende dal contesto di uso cioè abbiamo un flusso di informazioni a livello alto e non siamo in grado in questo modo di vedere se questo flusso di informazione potrebbe essere ad esempio un flusso implicito in questo momento non abbiamo tutte le informazioni che ci permettono di discriminare o meno se c'è disco rosso o disco verde. Se facciamo una stampa di un informazione di livello basso, ovviamente diamo verde se invece vogliamo stampare informazioni di livello alto, stiamo stampando informazione confidenziale siamo rossi, se leggiamo dal basso siamo verdi, se facciamo una readline dell'alto anche lì punto interrogativo dipende in che contesto siamo, perché se siamo in un contesto che siamo controllati dall'attaccante col cavolo che gli diamo il permesso di leggere informazione che è di alto livello.

Examples (integrity)

String hi; // security level **trusted** (**untainted**)

String lo; // security label **untrusted** (**tainted**)

Which program fragments (may) cause problems if integrity *hi* is important?

1. **hi = lo;**
2. **lo = hi;**
3. **lo = "1234";**
4. **hi = "1234";**

facciamo la stessa operazione con il discorso dell'integrità, quindi untaint versus taint dove hi deve essere untaint e lo deve essere taint anche qui l'abbiamo già visto di fatto nell'esempio della taint analysis che se abbiamo un passaggio da lo ad hi è ko perché stiamo dando informazione modificata se abbiamo lo uguale hi invece va bene perché abbiamo delle informazioni che untaint minore di taint e le altre informazioni sulla funzione invece che prendono dei valori primitivi abbiamo detto che i valori primitivi non possono esseri corrotti e quindi il flusso va corretto

Examples (integrity)

String hi; // security level **trusted** (**untainted**)

String lo; // security label **untrusted** (**tainted**)

Which program fragments (may) cause problems if integrity *hi* is important?

1. **hi = lo;** **tainted** < **untainted** **KO**
2. **lo = hi;** **untainted** < **tainted** **OK**
3. **lo = "1234";** **untainted** < **tainted** **OK**
4. **hi = "1234";** **untainted** <= **untainted** **OK**

Però andiamo a ragionare un po di più nel dettaglio sulla confidenzialità qui la domanda è la stessa, quali frammento di programma può causare problemi per la confidenzialità? Solo che il frammento di programma è decisamente leggermente più complicato

More on confidentiality

```
int hi; // security level secret  
int lo; // security level public
```

Which program fragments (may) cause problems for confidentiality?

1. `if (hi > 0) { lo = 99; }`
2. `if (lo > 0) { hi = 66; }`
3. `if (hi > 0) { print(lo); }`
4. `if (lo > 0) { print(hi); }`

More on confidentiality

- **int hi;** // security level **secret**
- **int lo;** // security level **public**
- *Which program fragments (may) cause problems for confidentiality?*
- **1. if (hi > 0) { lo = 99; } KO**
- **2. if (lo > 0) { hi = 66; } OK**
- **3. if (hi > 0) { print(lo);} KO**
- **4. if (lo > 0) { print(hi);} KO**

More on confidentiality

- `int hi;` // security level **secret**
- `int lo;` // security level **public**

- Which program fragments (may) cause problems for confidentiality because of **indirect flow**?

- **1. if (hi > 0) { lo = 99; } KO**
- **2. if (lo > 0) { hi = 66; } OK**
- **3. if (hi > 0) { print(lo); } KO**
- **4. if (lo > 0) { print(hi); } KO**

andiamo a esaminarli uno per uno allora se vedete qui abbiamo un ifthenelse e gli diamo disco rosso se $hi > 0$ allora lo prende 99 perché gli diamo il rosso? Possiamo dedurre informazioni su hi andando a vedere lo perché è chiaro che l' attaccante può controllare soltanto il basso livello, quindi può soltanto vedere il valore di lo, quindi andando a discriminare il valore di lo ad esempio, diventato 99 o è diverso dal 99 sa se hi è maggiore di zero o meno quindi ha un flusso di informazione sul valore alto e per questo non gli diamo disco verde. Il secondo perché è disco verde? Perché il problema qui della guardia non si pone? Perché è su un dato a low security Il terzo? Lo stesso problema del primo. Il quarto? Stiamo accedendo direttamente ad hi, quindi sono dei flussi di informazione non immediati, ma sono tutti i flussi di informazione che devono essere considerati.

Direct Flow vs Indirect Flow

direct aka explicit flows by “direct” assignment
or leak

```
lo=hi;
```

```
println(hi);
```

Abbiamo dei flussi espliciti dovuti all' assegnamenti diretti, oppure di quelli impliciti, cioè di quelli impliciti, sono in modo particolare l'operazione che viene fatta con l'ifthenelse che viene messa in verde che noi possiamo dedurre dell'informazione sul valore di informazione di livello alto andando a operare sull'informazione di livello basso, cioè modo in particolare l'esempio che qui è in verde vi fa vedere come io posso andare a vedere l'informazione sulla positività o meno di un valore di livello alto, semplicemente controllando informazioni pubbliche e quindi questo è indiretto.

Direct Flow vs Indirect Flow

indirect aka implicit flows (by indirect “influence”)

```
if (hi>0){lo=99;}
```

Implicit flows can be partial, ie leak some but not all info

the example above only leaks the sign of `hi`, not its value.

Qui l' attaccante può solo vedere i valori di livello basso non può controllo perché sono pubblici, non può controllare i valori di livello low allora andando a vedere quindi assumendo che lo abbia un valore iniziale che l' attaccante conosce perché controlla i livelli pubblici allora andando a vedere la differenza tra il valore iniziale e il valore finale sa se è stato eseguito quella assegnamento e quindi se il valore finale è 99 e assumendo ovviamente che il valore iniziale diverso da 99 allora a questo punto sa che il valore della variabile `hi` è positivo, altrimenti è negativo. In questo senso si riesce a derivare il segno della variabile di alto livello.

More on confidentiality

int hi; // security level **secret**

int lo; // security level **public**

Which program fragments (may) cause problems for confidentiality?

while (hi>99) do {...};

while (lo>99) do {...};

a[hi] = 23; // where a is **high/secret**

a[hi] = 23; // where a is **low/public**

a[lo] = 23; // where a is **high/secret**

a[lo] = 23; // where a is **low/public**

More on confidentiality

```
int hi; // security level secret
```

```
int lo; // security level public
```

Which program fragments (may) cause problems for confidentiality?

```
while (hi>99) do {...}; KO
```

```
// timing or termination may reveal if hi > 99
```

```
while (lo>99) do {...}; OK
```

Perché diamo disco rosso al while? Perché è un ko? Perché in un qualche modo e l'abbiamo già determinato, una specie di cover channel, cioè abbiamo un flusso di informazione che ci permette di comprendere se il valore della variabile di alto livello è maggiore di 99 e quando è che lo conosciamo? Non andando a vedere il valore della variabile di alto livello, ma andando a scoprire se è terminato o non terminato il ciclo, quindi misurando, in quel senso c'è scritto timing, e andando a vedere se il ciclo è terminato, noi possiamo determinare una informazione sul valore della variabile di alto livello, la stessa cosa invece non vale quando la guardia è controllata da una variabile pubblica di basso livello, questo è lo stesso pattern del ciclo dove però il controllo della guardia è fatto dal lo.

More on confidentiality

int hi; // security level **secret**

int lo; // security level **public**

Which program fragments (may) cause problems for confidentiality?

a[hi] = 23; // where a is high/secret KO

// exception may reveal if hi is negative

a[hi] = 23; // where a is low/public KO

// contents of a may reveal value of hi and, again,

// exception may reveal if hi is negative



More on confidentiality

int hi; // security level **secret**

int lo; // security level **public**

Which program fragments (may) cause problems for confidentiality?

a[lo] = 23; // where a is high/secret **KO**

// exception may reveal the length of a, which may be secret

a[lo] = 23; // where a is low/public **OK**

assumiamo che a sia sempre di alto livello e supponiamo di fare un'operazione di accesso non con un indice di alto livello ma con indice di basso livello, anche in questo caso il fatto di avere un'eccezione ci può fare tirare fuori delle informazioni che volevamo avere privata, invece se a è di basso livello e l'indice di basso livello abbiamo disco verde, vedete non c'è flusso di informazioni che creerà problemi.

Covert Channels

More subtle forms of indirect information flows can arise via
hidden channel aka **covert channels** aka **side channels**

(non)termination

```
while (hi>99) do {....};  
if (hi=99) then {"loop"} else {"terminate"}
```

execution time

```
for (i=0; i<hi; i++) {...};  
if (hi=1234) then {...} else {...}
```

exceptions

```
a[i] = 23
```

may reveal length of a (if i is known),
or leak info about i (if length of a is known),
or reveal if a is null..

More on cover channels

- Apart from timing & terminations, there are many more covert channels:
 - noise
 - power consumption
 - :

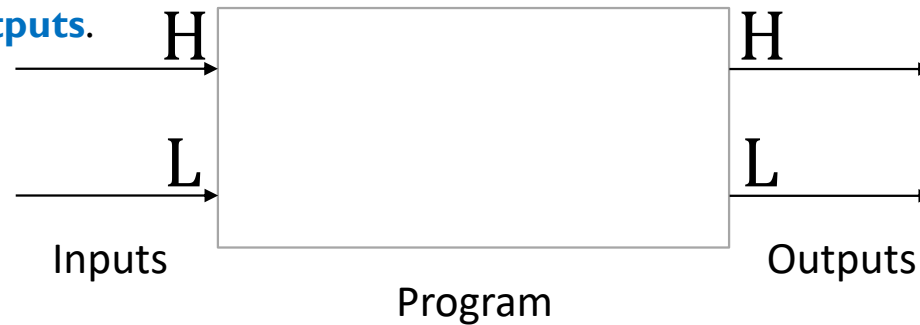
allora abbiamo visto sostanzialmente tre tipologie di covert channel, cioè tre tipologie di canali che non sono previsti all'interno del programma in termini di canali legali di comunicazione, ma che possono influire su il flusso di informazione. Il primo è il discorso della terminazione non terminazione quando abbiamo un ciclo, una guardia che dipende da un valore di alto livello beh, a seconda che terminiamo non terminiamo la guardia, possiamo dedurre dell'informazione sul valore di alto livello la stessa cosa si può fare sugli ifthenelse. Il tempo di esecuzione può rivelare se la si riesce a misurare, poi vedremo degli esempi maggiormente significativi più avanti, può rivelare delle informazioni di alto livello se non si fa attenzione. Le eccezioni l'abbiamo appena detto, cioè può rivelare il valore dell'indice a seconda della struttura dell'array, se pubblico, di basso livello, di alto livello così via...



A BIT OF THEORY

L'uso da parte degli attaccanti di covert channel, voi l'avete visto in altri contesti, ad esempio la power consumption che viene utilizzata per fare delle analisi sulle smart card o altri sistemi per andare a vedere quanto tempo viene utilizzato e utilizzando queste informazioni uno riesce a dedurre delle informazioni private. Allora questo era solo per fare degli esempi, adesso vi devo annoiare nel senso buono del termine, con un po' di teoria che c'è dietro. Qual è l'idea? Ancora una volta noi dobbiamo fare qual è il modello del nostro comportamento.

deduce information flows
from inputs to outputs.



H high Level
L Low level

OBSERVING PROGRAMS

Quindi il modello che noi abbiamo è un modello osservazionale di programmi, ma un modello osservazionale di programmi dove i programmi hanno delle caratteristiche particolari, in modo particolare i programmi in questo momento supponiamo di avere un reticolo di informazione a due livelli lo verso hi, ma possiamo generalizzare questa nozione a reticoli più articolati e reticoli che portano dati nelle etichette di informazione, noi osserviamo i programmi, quindi che cosa vuol dire? vuol dire che abbiamo informazioni di input che possono essere divise in high e low, abbiamo informazioni di output che possono essere suddivise in high e low e vogliamo vedere il comportamento del programma in termini di input/output sui livelli alti e su livelli bassi. Quindi il nostro programma è una specie di black box, in realtà è grigia, non è proprio una black box, perché abbiamo un po di informazioni su i livelli dei dati che fluiscono all'interno del programma.

$x := y \bmod 2$ y flows to x

$x := y * 0$ no flow

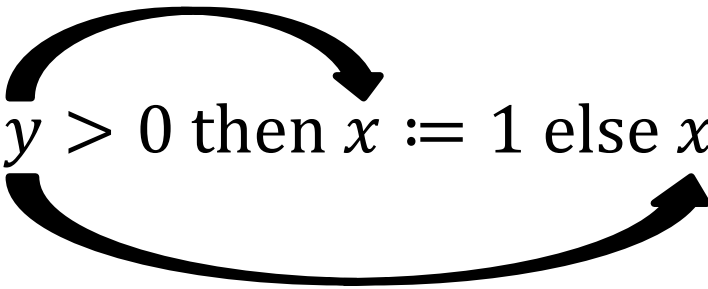
$z := y + 2; x := z$

$z := y + 2; x := z - y$

Flow is not
always
transitive!

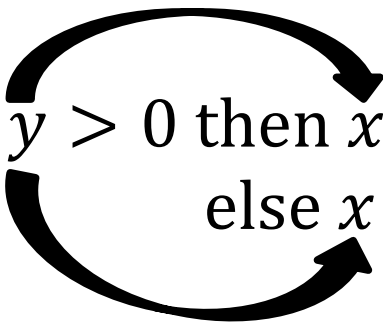
OBSERVING FLOWS

Vogliamo osservare il flusso del comportamento allora sicuramente l'assegnamento, come già visto quando abbiamo fatto la taint analysis, dice che c'è un flusso, in modo particolare, ad esempio, l'assegnamento da x a y modulo 2 dice che c'è un flusso di informazione tra y e x. Però non è vero che tutti gli assegnamenti definiscono un flusso d'informazione, in modo particolare se io faccio l'assegnamento $x = y * 0$ non c'è flusso tra y e x, tutt'altro, perché dipende dall'operazione primitiva, quindi il flusso dipende non solo dal fatto che ho un assegnamento ma dalle operazioni che operano sui dati, primitive o operazione definita da programma. Se noi mettiamo in sequenza due assegnamenti, in modo particolare vedete qui abbiamo l'assegnamento $z = y + 2$ quindi vuol dire che mettiamo un flusso tra y e z, poi dopo successivamente a x associamo il valore z, quindi a questo punto transitivamente uno si aspetta che y influenza x quindi abbiamo c'è un flusso tra y e z, c'è un flusso da z e x, per la transitività diciamo che c'è un flusso tra y e x. Ma è sempre questo il caso? Andiamo a vedere l'esempio sotto che invece vi fa vedere che non sempre il flusso è transitivo perché abbiamo il solito assegnamento a z di $y + 2$ poi dopo abbiamo un assegnamento di x a z che è quello che abbiamo prima modificato dando le informazioni su y, meno y, quindi a questo punto y non è coinvolto nel valore finale di x e quindi a questo punto questo è un controesempio alla transitività del flusso di informazione.



if $y > 0$ then $x := 1$ else $x := 2$


if $y > 0$ then $x := 0$ else $x := 0$



if $y > 0$ then $x := 1; x := 0$
else $x := 2; x := 0$


OBSERVING FLOWS

Andiamo a fare il controllo di una guardia a questo punto la guardia, a seconda che sia maggiore o minore uguale a zero, influisce su l'assegnamento di x uguale a uno e sull'assegnamento di x uguale a 2, quindi vuol dire che abbiamo un flusso che dipende dalla guardia ma anche in questo caso ci sono delle eccezioni. Nell'ultimo esempio, invece, un esempio che mette insieme le due cose che abbiamo visto poc anzi bene, in questo caso sicuramente abbiamo un flusso di informazione perché facciamo delle operazioni comunque sui valori della variabile x



while $y > 0$ do $x := x + 1; y := y - 1$ end

?



while $y > 0$ do C end; $x := 1$



OBSERVING
FLOWS

Sicuramente abbiamo un flusso di informazione tra la variabile del while quando è maggiore di zero e quando eseguiamo le istruzioni all'interno del corpo del while, ma sicuramente abbiamo un flusso indiretto dove lì c'è un punto interrogativo, sempre su l' assegnamento a x uguale 1 quando usciamo dal while perché c'è un punto interrogativo? Perché questo flusso l'abbiamo non sempre cioè intuitivamente uno immagina lo abbia sempre, ma dipende dalla terminazione di c. Cioè, abbiamo il flusso di informazione che è regolato dalla guardia del while soltanto diretto, ovviamente soltanto quando termina proprio per questo prima abbiamo introdotto il cover channel di terminazione perché abbiamo un flusso indiretto su x uguale a 1 quando termina negli altri casi, no.

Information Flow Policies

An IF policy specifies restrictions on the associated data, and on all its derived data.

**Example: IF policy for *confidentiality*:
Value v and all its derived values are
allowed to be read at most by user A**

v is allowed to flow only to A

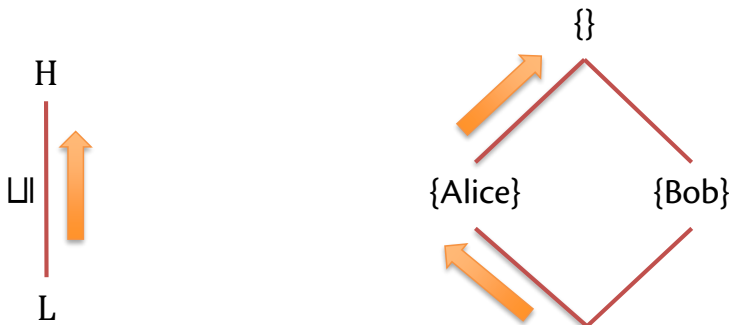
allora, a questo punto, noi quello che vogliamo fare vogliamo scrivere delle politiche di sicurezza che regolano il flusso di informazione, sia diretto che indiretto, per salvaguardare la confidenzialità dei dati che affluiscono all'interno dei programmi.

le politiche e quindi riassumiamo quello che abbiamo detto sino a questo punto, indubbiamente sono caratterizzate da un reticolo che è una struttura di un insieme equipaggiata con una relazione di ordinamento e l'operazione di ordinamento quando io ho due etichette due label del reticolo che dicono L e L' che dicono se L minore di L' vuol dire che L' è restrittivo, almeno restrittivo quanto L, quindi cosa vuol dire? Vuol dire che gli stessi vincoli di L può avere ulteriori proprietà e per questo motivo il flusso da L e da L' è permesso qui sotto ci sono vedete due livelli, due esempi di lattice, uno con valore lo e hi che abbiamo visto in questo momento e l'altro un reticolo dove abbiamo il minimo, poi che alice e bob sono allo stesso livello ma non sono confrontabili tra di loro, sono confrontabili soltanto a livello più alto, vuol dire che le proprietà di alice non sono necessariamente uguale alle proprietà di bob, ma c'è un super user che può avere le proprietà di entrambi di alici e bob.

They form a *lattice* $\langle L, \sqsubseteq \rangle$ with join operation \sqcup .

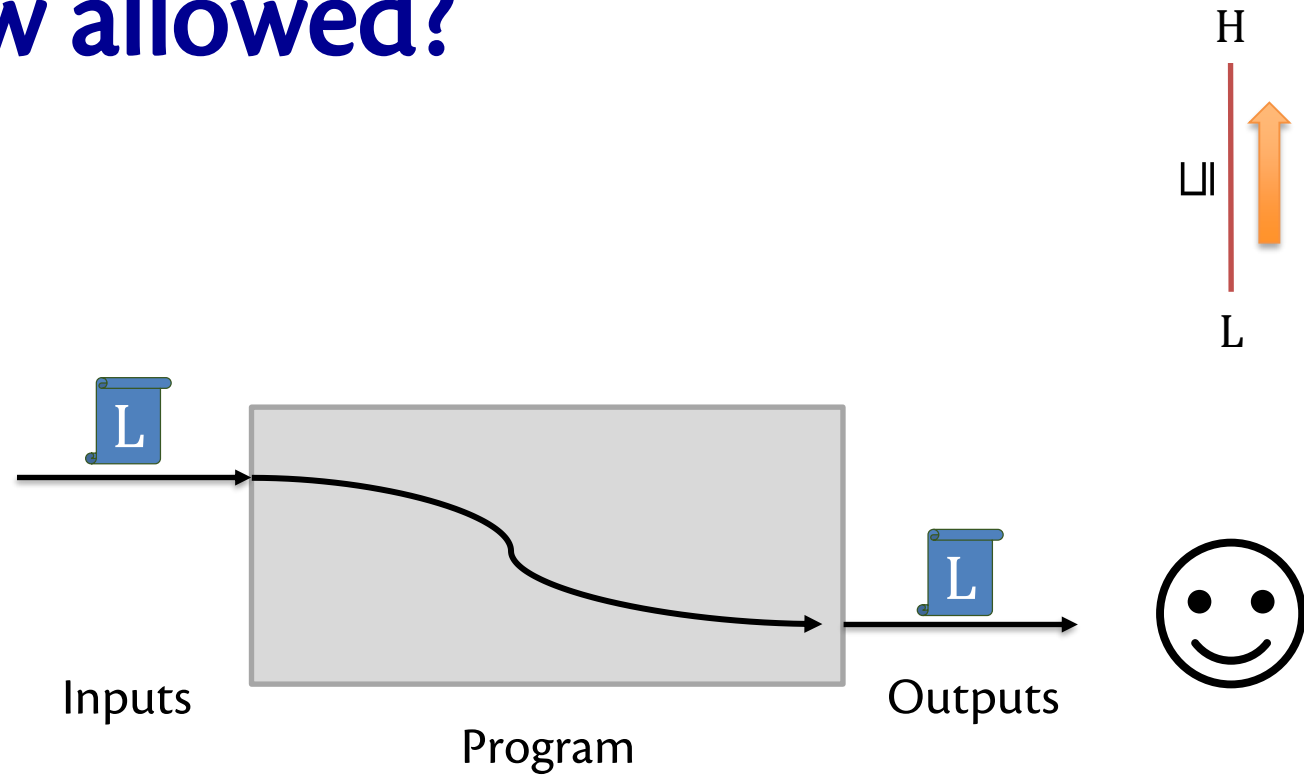
For $\ell, \ell' \in L$, if $\ell \sqsubseteq \ell'$, then:

ℓ' is *at least as restrictive as* ℓ , and thus,
information flow from ℓ to ℓ' is allowed.



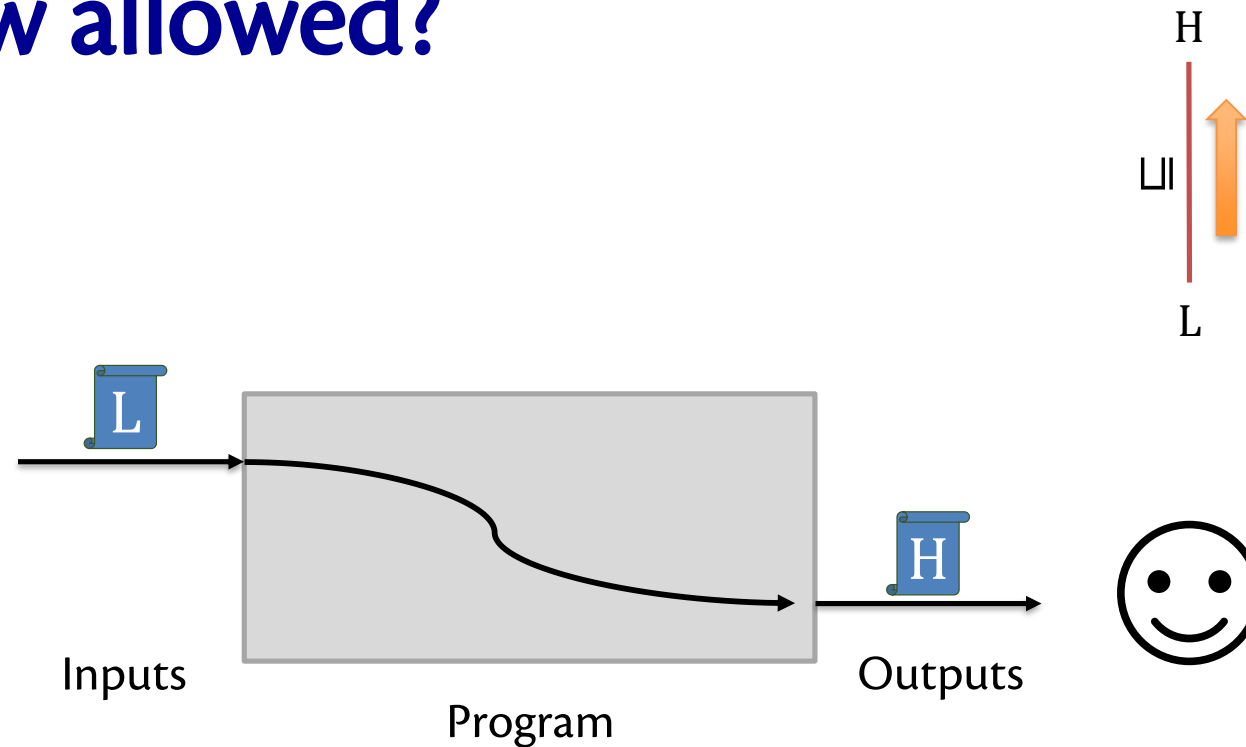
INFORMATION
FLOW POLICIES

Is a flow allowed?



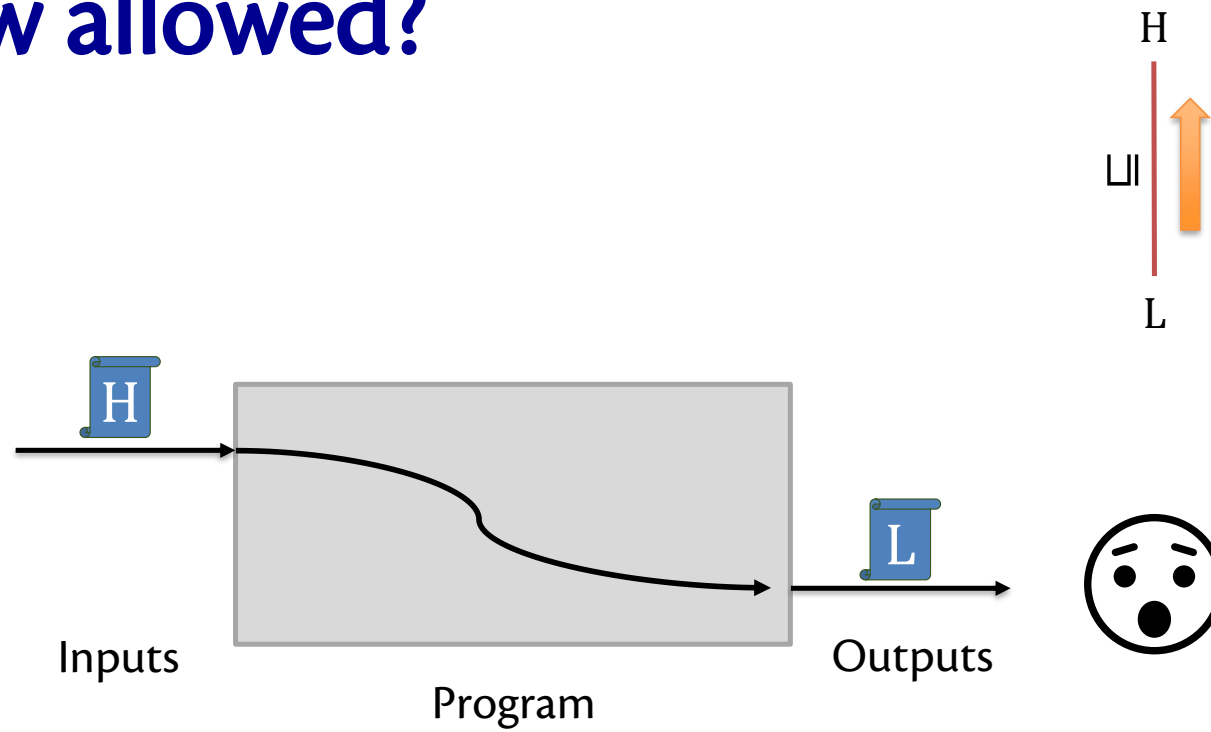
vedi spiegazione dopo

Is a flow allowed?

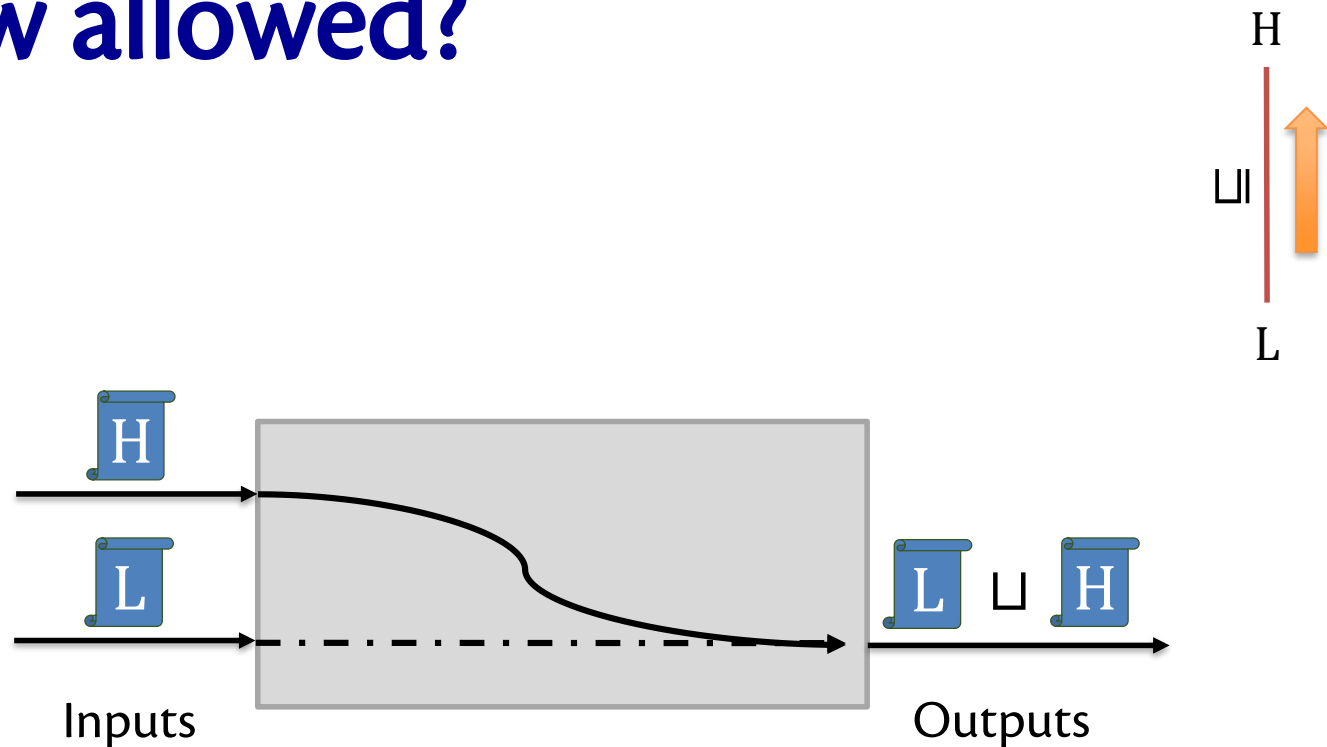


A questo punto uno deve andare a comprendere qual è ancora una volta il modello del comportamento in termini di un reticolo, in questo caso abbiamo il reticolo lo verso hi e in termini di flusso. Allora se noi abbiamo un flusso di informazioni all'interno del programma che mi porta da low a low va la faccina sorridente, ci dice che lo possiamo accettare come comportamento, non fa un informazione di flusso scorretto. Similmente, se abbiamo un flusso sempre nel solito reticolo che mi porta da low a high compatibile col flusso di informazione, per cui anche lì abbiamo la faccina corretta, è un flusso accettabile. La faccina invece cambia, l'emoticon cambia quando invece abbiamo nel nostro programma un flusso che trasmette dell'informazione, parte di essa o qualcosa di essa dal livello alto al livello basso, bene, li possiamo dire invece di dover porre particolare attenzione, perché? perché abbiamo un leakage di informazione che doveva rimanere segreta.

Is a flow allowed?



Is a flow allowed?

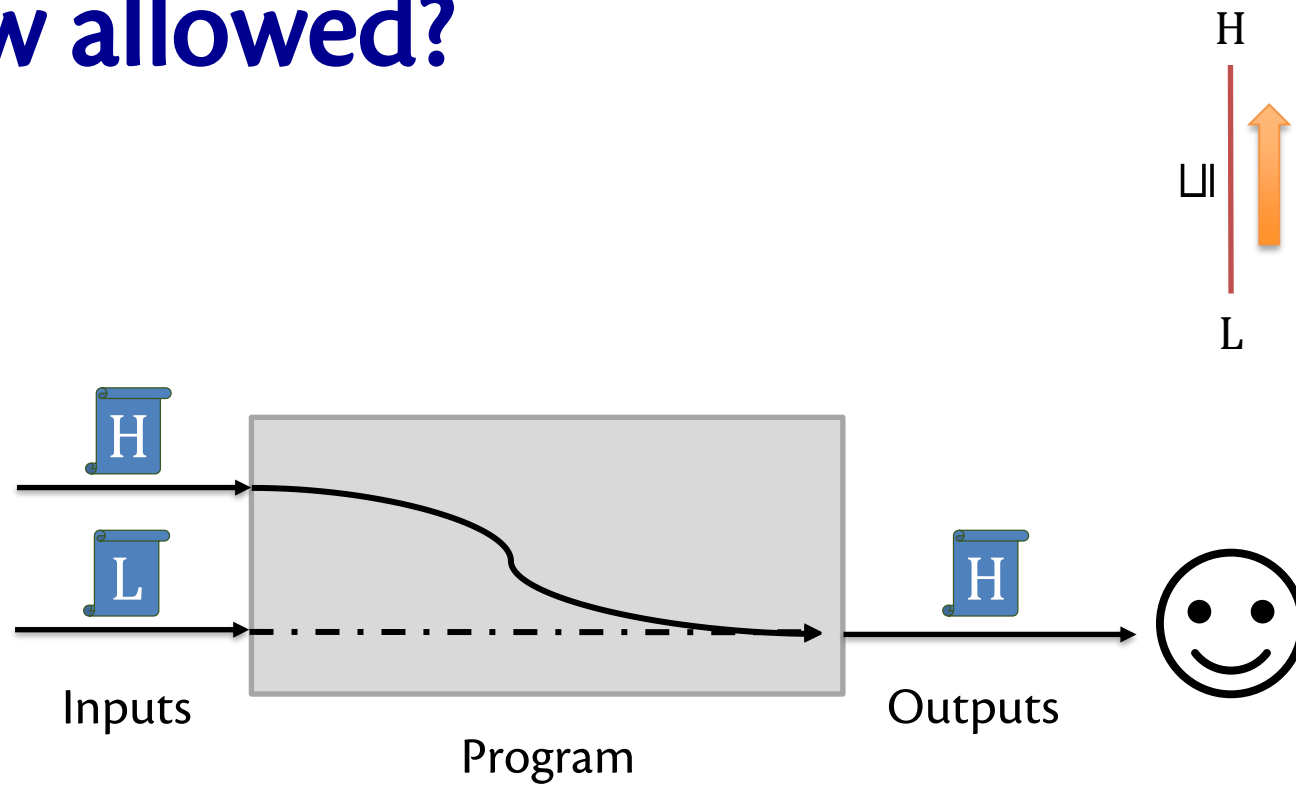


Ma se abbiamo due informazioni, una di livello alto e una di livello basso che fluiscono sull' output, allora, a questo punto, qual è il livello di sicurezza dell' output? C'è scritto che deve essere $L \sqcup H$ che cosa vuol dire? Vuol dire che il livello di sicurezza deve essere dato dall'estremo superiore dei livelli dei dati che fluiscono in ingresso, quindi in modo particolare, nel nostro caso, deve essere un'informazione di livello alto quindi a questo punto noi accettiamo che ho un flusso di informazione che dal basso arriva a livello alto, però proprio per il fatto che abbiamo un reticolo che mi dice che è $L \sqcup H$ mi dà H .

- For each ℓ and ℓ' , there should exist label $\ell \sqcup \ell'$, such that:
 - $\ell \sqsubseteq \ell \sqcup \ell'$, $\ell' \sqsubseteq \ell \sqcup \ell'$, and
 - if $\ell \sqsubseteq \ell''$ and $\ell' \sqsubseteq \ell''$, then $\ell \sqcup \ell' \sqsubseteq \ell''$.
- $\ell \sqcup \ell'$ is called the **join** of ℓ and ℓ' .
- Examples: $L \sqcup L = L$, $H \sqcup H = H$, $L \sqcup H = H$



Is a flow allowed?



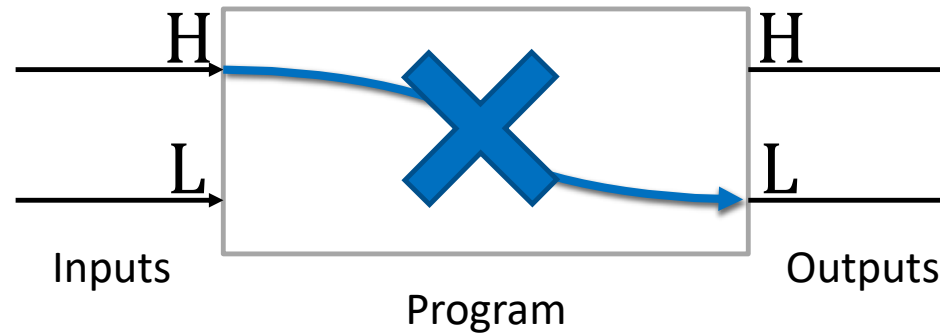
Is a flow allowed?

Given

- a lattice $\langle \{L, H\}, \sqsubseteq \rangle$ of labels,
- a program C , and
- labels on program inputs and outputs,

are all the flows from inputs to outputs that are caused by executing C allowed?

Dato un reticolo di etichette che definiscono il livello di sicurezza dell'informazione, dato un programma e definito le etichette di alto e di basso livello sul programma, allora quand'è che tutti i flussi che portano il programma dall'ingresso all'uscita sono flussi legali, cioè non violano la politica di sicurezza, definita da questo reticolo? Perché a questo punto abbiamo definito il modello del comportamento, adesso dobbiamo definire il modello che ci dice quand'è che i flussi sono corretti o meno.



Changes on H inputs should not cause changes on L outputs

NON INTERFERENCE: GOGUEN-MESEGUER 1982



Questa nozione è stata introdotta anche questa un po di tempo fa, nell 82, ancora una volta per dire che il problema della sicurezza in ambito della ricerca era un problema che era sentito già da diverso tempo ed è chiamata NON INTERFERENCE . Vedete lì c'è una bella x, cosa mi dice? riceviamo il comportamento del nostro programma come questa grey box, dove ho informazioni sugli ingressi suddiviso tra high e low, sull uscita suddivisa tra high e low e a questo punto, noi abbiamo che il programma soddisfa la non interference quando noi, modificando i dati in ingresso che sono di alto livello e lasciando inalterati quelli di basso livello non si modifica il comportamento di basso livello. che cosa vuol dire questo intuitivamente? Vi ricordate l' attaccante può influire sui valori di basso livello, non può influire sui valori di alto livello quindi vuol dire che non può alimentare il programma dando lui valori di alto livello. Osservare il comportamento del programma soddisfa la non interference esattamente nelle situazioni in cui noi alimentiamo valori noi che possiamo dare valori di alto livello, alimentiamo con valori di alto livello e andiamo a osservare il comportamento del programma se non cambia lasciando inalterato il valore di basso livello, se non cambia, vuol dire che il valore di basso livello non ha un flusso diretto/indiretto dei valori di alto livello e quindi vuol dire che il programma soddisfa la non interference

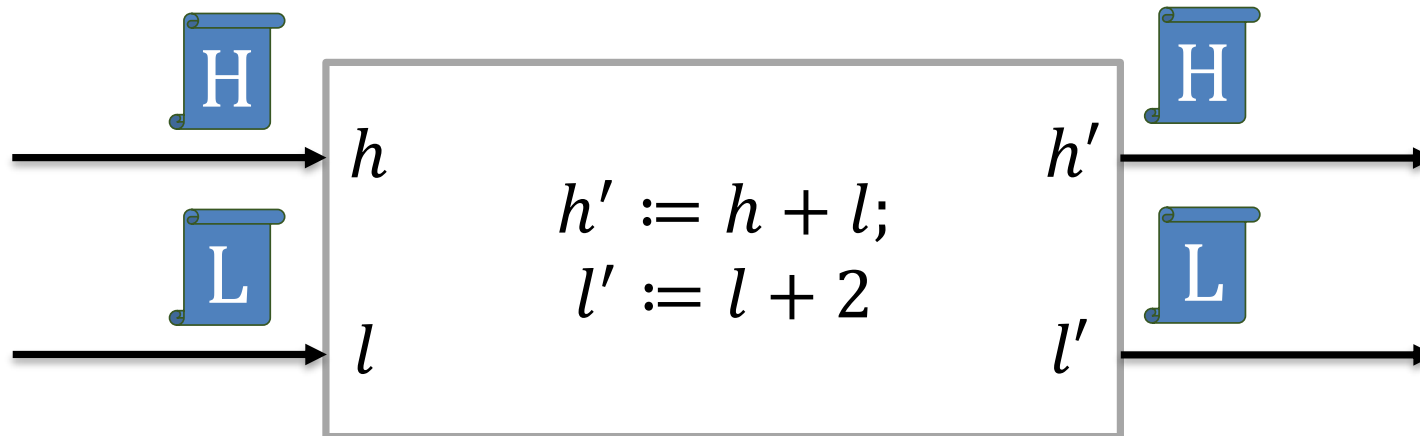
Non Interference

Paper available on Teams

Non interference for a program: Different H inputs, keeping L inputs fixed, should not cause different L outputs.

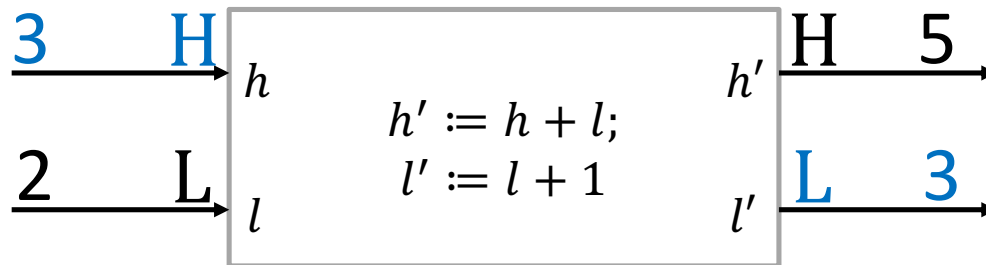
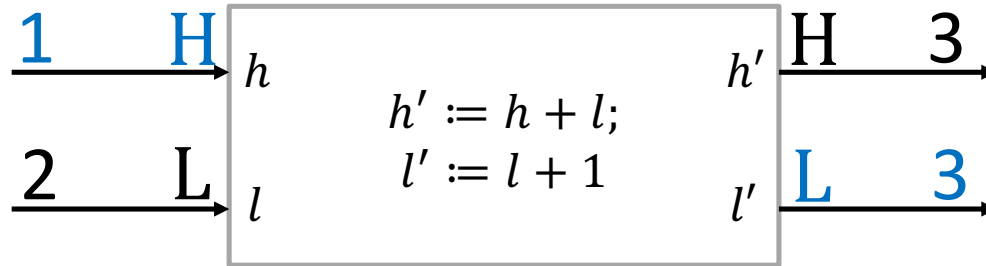
If a program C satisfies NI, then all flows from inputs to outputs are allowed.

Non interference per un programma è se input diversi di alto livello, lasciando inalterato quelli di basso livello, non cambiano i valori di basso livello.



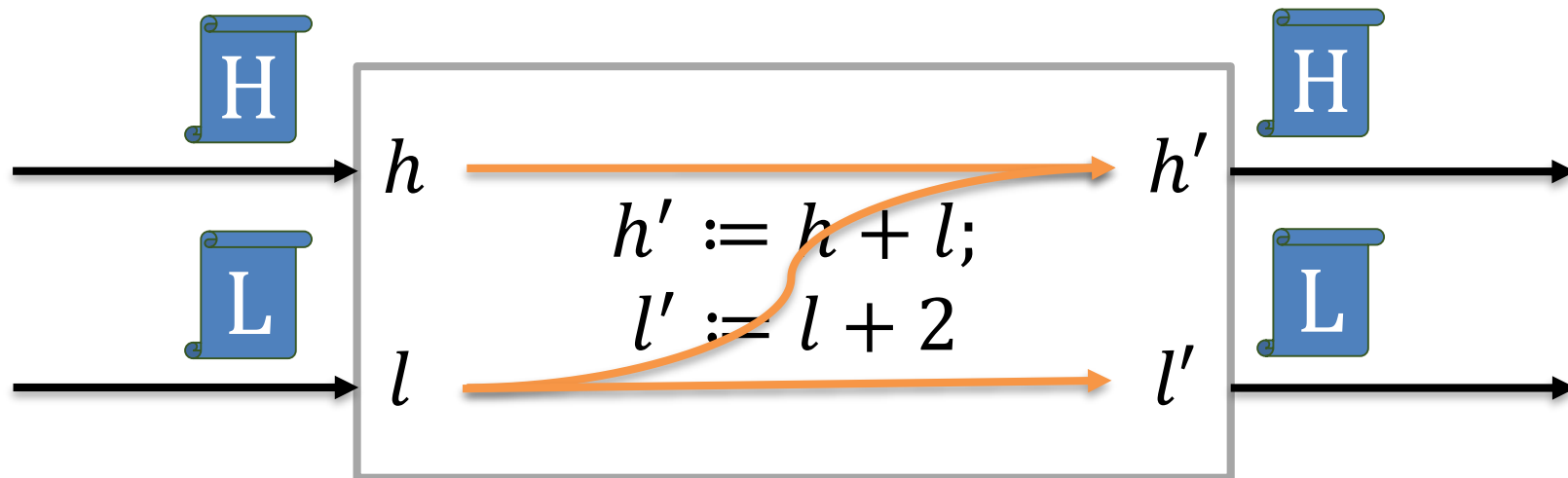
EXAMPLE

Supponiamo di avere questo programma, vedete abbiamo due variabili h e l di alto livello, h' e l' di basso livello e di fatto il programma è una sequenza che dice: $h' = h + l$ e $l' = l + 2$ Allora la domanda è soddisfa la non interference?



The program satisfies noninterference!

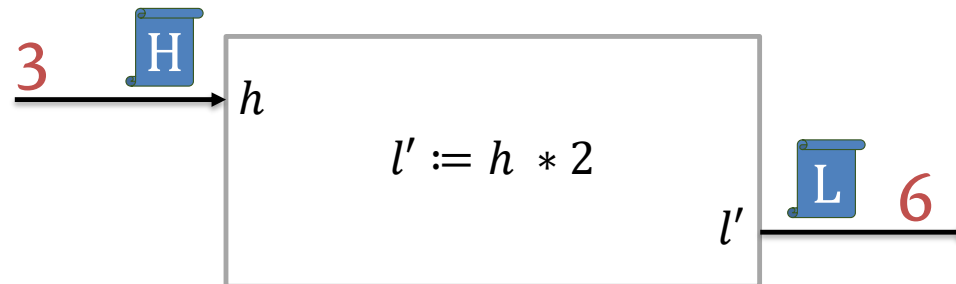
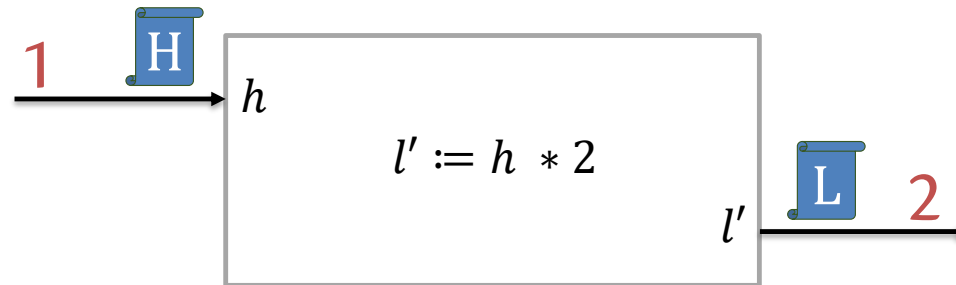
Facciamo un po' di esperimenti, allora supponiamo di alimentare questo programma con i valori 1 e 2 e andare a vedere l'uscita allora nel primo caso notato che quando noi facciamo questo tipo di esperimenti di gray box non è che vediamo dentro, vediamo proprio l'I/O la trasformazione è imputabile allora questo punto nel primo caso vedete, alimentando con h con 1 abbiamo il valore 3 come risultato ad alto livello e il valore di basso livello è $2+1 = 3$ Cambiamo il valore di alto livello, che è diventato 3 quindi a questo punto abbiamo che h' è uguale ad $h + l$ quindi $3+2$ uguale a 5, il valore di basso livello rimane inalterato, quindi vuol dire che non abbiamo un flusso di informazione che mi trasforma il valore di basso livello, tenendo conto delle informazioni di alto livello, per cui possiamo affermare abbastanza ragionevolmente che questo programma soddisfa la non interference.



THE PROGRAM HAS ALLOWED FLOWS!!

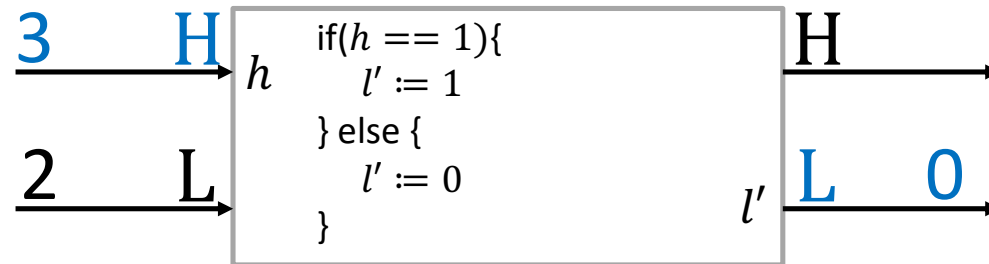
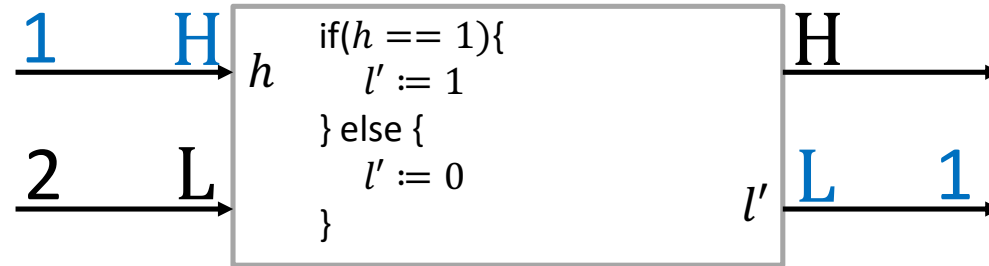


Se l' andiamo a vedere in termini del flusso interno a questo punto andiamo a vederlo quello, come dicevamo dalla visione non dell'attaccante ma del controllore del sistema, come grey box, quello che possiamo dire che è vero che c'abbiamo un flusso di informazione, abbiamo un flusso di informazione tra l ed h' però il flusso di informazione tra l ed h' è un flusso di informazione compatibile con il reticolo low verso high che abbiamo, quindi è un flusso di informazione legale. Questo proprio per caratterizzare bene la non interference.



The program does not satisfy noninterference!

Andiamo a vedere questi due esempi che invece non soddisfano la non interference. Abbiamo un valore di alto livello h , abbiamo solo un'uscita di basso livello l' e quello che facciamo è che l' uguale ad $h * 2$ bene vedete che qui non c'è bisogno di tanto ingegno per comprendere che il risultato del basso livello è influenzato dal livello alto, quindi non soddisfa la non interference perché vedete cambiando il valore di alto livello abbiamo trovato il controesempio subito e quindi questo programma non soddisfa la non interference.



The program does not satisfy noninterference!

In questi altri esempi abbiamo il valore di un ifthenelse, abbiamo se h uguale a 1 che è di alto livello allora il risultato di l' è 1 altrimenti l' uguale a zero. Voi vedete che il risultato di h non cambia? in entrambi i casi, quindi quando alimentiamo il programma con uno e quando alimentiamo con tre su h non abbiamo alcun valore in uscita, ma su l abbiamo dei valori diversi, l' è in un caso uguale a 1 e nell'altro caso uguale a 0, cioè di fatto abbiamo un flusso di informazione perché riusciamo a determinare di di fatto se h uguale a 1 o se h diverso da 1 andando a vedere il valore del risultato l .

State equivalence

- Assume C to be a program
- Variables in the program C model inputs and outputs.
- Let M be a state (the memory) (mapping variables to values)
- **We say that $M1$ is equivalent to $M2$ iff they agree on values of variables tagged with L**
- **$M1(I) =_L M2(I)$ for all $I \in L$**
- **Note states $M1$ and $M2$ may not agree on values of variables tagged with H !!!**

Adesso continuiamo con la teoria, allora assumiamo che ci sia un programma e assumiamo di voler vedere le variabili di C supponiamo che le variabili di C modellano i comportamenti input output nello stile che abbiamo detto prima, supponiamo di avere una memoria intesa come una funzione che mappa i nomi degli identificatori in valori quindi la memoria è esattamente lo store, lo stato del programma. Allora diciamo che due memorie sono equivalenti se concordano sui valori di basso livello, cosa vuol dire? vuol dire, ho discriminato le variabili di alto e di basso livello, quindi ho un modo che mi dice questa variabile di alto livello, questa variabile di basso livello io dico che due memorie sono equivalenti quando per quanto riguarda i valori di variabili di basso livello, i contenuti della memoria sono identici, i contenuti dei valori di ogni memoria possono variare, però quelle di basso livello no, allora a questo punto ho quindi una nozione di equivalenza, cioè dico che due memorie sono equivalenti rispetto ai valori di basso livello e lo definisco in quel modo, quando questo vale per tutti i valori di basso livello, quindi sto definendo una nozione di equivalenza, quindi una relazione riflessiva, simmetrica e transitiva tra le memorie e metto nelle classi di equivalenza, quindi nelle partizioni delle memorie che sto così definendo tutti quelle memorie che non sono distinguibili per i valori di basso livello, cioè se sono l' attaccante, posso vedere soltanto l'informazione di basso livello nello stato del mio programma, non riesco a discriminare una memoria rispetto all'altra quindi che cosa vuol dire? vuol dire che questa è una nozione di equivalenza che tiene conto intuitivamente del fatto che l' attaccante può vedere soltanto il livello basso, quindi può ispezionare la memoria, ma può ispezionare la memoria soltanto sulle variabili di basso livello, non può accedere a quelle di alto livello, quindi da un'idea molto chiara anche del threat model, di cosa può fare l' attaccante.

Let C be a program

$C(M_i)$ are the observations produced by executing C to termination on initial state M_i :

- **final outputs, or**
- **intermediate and final outputs.**

Then, observations tagged with L should be the same:

- **$C(M_1) =_L C(M_2)$.**

Consideriamo un programma C e chiamiamo $C(M_i)$ le memorie quindi le osservazioni del comportamento che sono prodotte dall'esecuzione di C . Quindi vedo tante memorie che possono essere o il risultato finale, cioè quando il programma termina o tutti i risultati intermedi dell'esecuzione e il programma finale. Che cosa vuol dire questo intuitivamente? In questo modello l'attaccante cosa può fare? Può mandare in esecuzione il programma, ha il controllo sui valori di basso livello, può andare a esaminare il risultato finale dell'esecuzione del programma sui valori di basso livello oppure può andare a esaminare il valore delle variabili di alto livello bloccandone un certo punto l'esecuzione del programma, andando a vedere quello che c'è sempre nei valori di basso livello, questo quindi vuol dire l'attaccante ha un po' di potenza che permette soltanto di vedere i valori dell'alto livello. Allora a questo punto possiamo introdurre su questo insieme di comportamenti, su queste tracce di comportamento, una nozione di equivalenza. Diciamo che due programmi sono equivalenti quando, se io vado a vedere tutte le loro esecuzioni, anche quelle finali, sono state equivalent, che cosa vuol dire? Vuol dire che anche se vado a vedere solo quelle finali o anche tutti quelli intermedi, l'attaccante non è in grado di scoprire la differenza.

For a program C and a mapping from variables to labels in $\{L, H\}$:

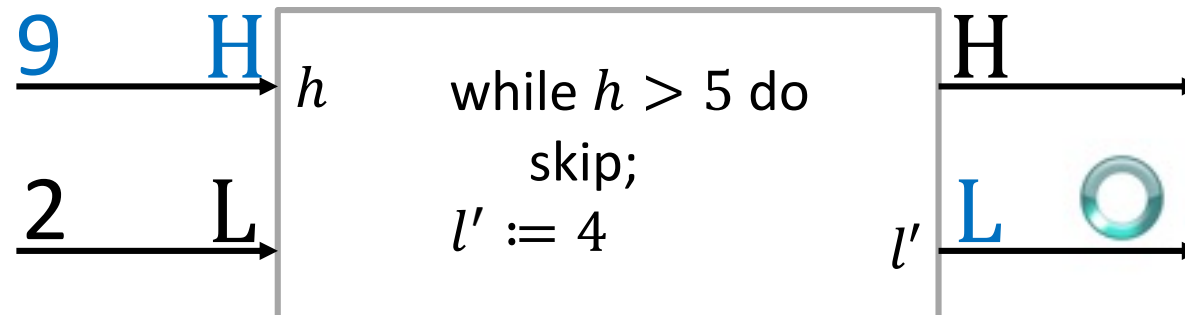
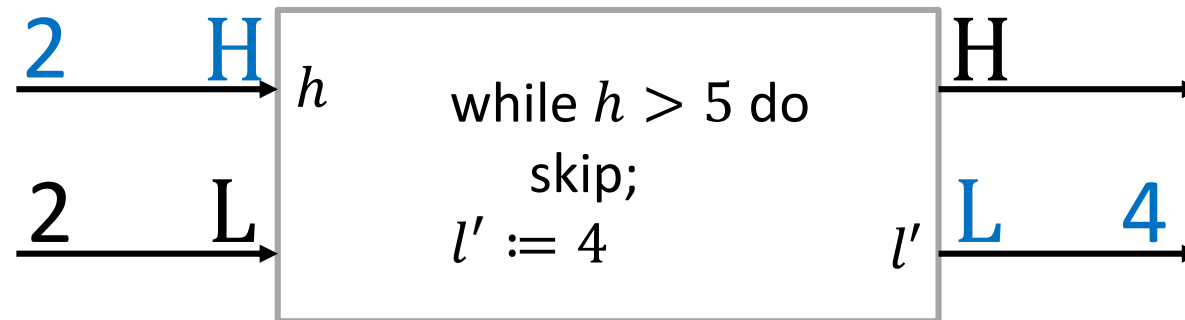
$$\forall M_1, M_2: \text{ if } M_1 =_L M_2, \text{ then } C(M_1) =_L C(M_2)$$

allora a questo punto vedete se io ho la stessa operazione, la parametrizzo, cioè questa nozione di coerenza dei comportamenti la parametrizzo con tutte le memorie, se dico che per ogni possibile memoria i due sono equivalenti, di fatto sto formalizzando la non interferenza, quindi ho dato in termini di nozione di equivalenza la non interferenza.

Threat model

- **The attacker could only observe outputs tagged with L.**
- **What if the attacker can also sense nontermination?**

Do una nazione di non interferenza, che è dipendente, gli inglesi dicono sensitive to termination, cioè dipende proprio dal fatto che io noto e come attaccante sto vedendo che il programma termina. Quindi a questo punto, vedendo questa nozione mi dice: quand'è che due programmi non sono distinguibili per non interferenza? Quando in qualunque stato di ingresso, se il programma termina allora l'attaccante non riesce a vedere differenze sui valori bassi, se questo è il caso, il programma soddisfa la proprietà di non interferenza e questo il modo di definirla esattamente in termini di memoria, quindi ho riprodotto in termini di una nozione di esecuzione e dello stato dell'esecuzione del programma, la nozione di interferenza. Adesso uno si domanda, ma se l'attaccante è anche in grado di avere un'idea di quello che succede su una non terminazione?



The previous definition considers only programs that terminate.

What if the termination behavior of a program depends on secret data?

Termination Sensitive Non Interference

- **If**
 - $M_1 =_L M_2$,
- **then**
 - **C terminates on M_1 iff C terminates on M_2 ,**
 - and**
 - **$C(M_1) =_L C(M_2)$.**

allora diciamo che due memorie sono equivalenti sui valori di basso livello se dato, ovviamente un programma, C termina sulla prima memoria se e solo se lo stesso programma termina sulla seconda memoria, quindi vuol dire che abbiamo 2 memorie che sono equivalenti se: quando eseguo il programma su quella memoria anche il primo programma termina e anche il secondo programma termina e a questo punto quando terminano entrambi, devono terminare entrambi, hanno lo stesso comportamento osservabile. Allora a questo punto questa nozione di non interferenza, vedete, tiene conto del fatto che io potrei non terminare perché se uno non termina e l'altro termina non sono equivalenti e questo è esattamente il caso che stava considerando questo esempio, quindi, non essendo equivalenti, sono discriminati dalla nostra nozione di non interferenza.

Covert Channels

- **Termination channel is a covert channel:**
 - not intended for information transfer, yet exploitable for that purpose.
- **Other covert channels:**
 - timing, heat emission, metadata.
- **Information flow control can address covert channels:**
 - treat covert channels as program outputs.
- **Variations of non interference can proscribe flows to covert channels.**

abbiamo fatto vedere che la nozione di non interferenza come era stata introdotta che era quella introdotta inizialmente dove programmi terminavano può essere modificata tenendo conto della non terminazione e quindi cosa vuol dire? vuol dire che la non terminazione può esser visto come meccanismo di cover channel per far fluire dati di alto livello in punti dove non dovrebbero essere ma la stessa cosa può essere fatta con timing, con emissioni di calore, con metadati e quindi si possono trattare diverse nozioni di non interferenza che tengono conto di diverse tipologie di cover channel.



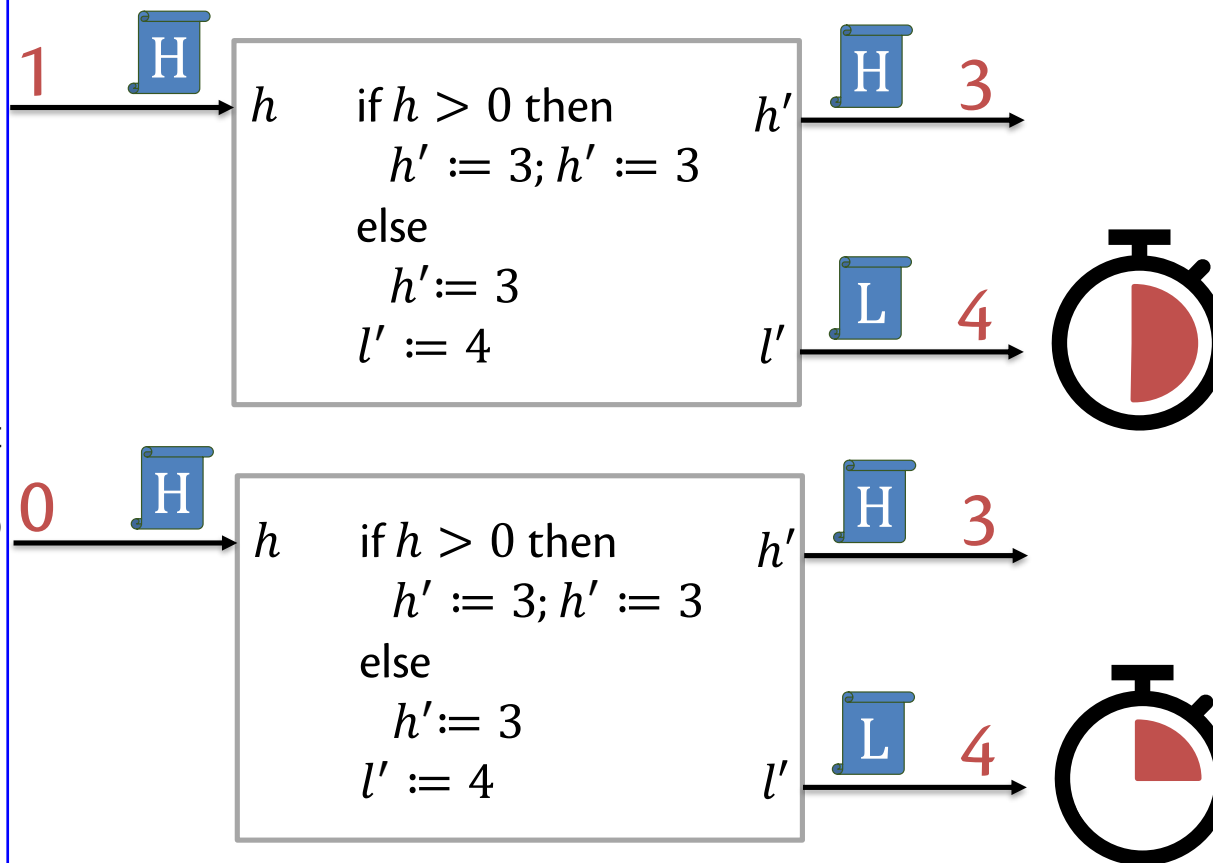
THREAT MODEL

What if the attacker can also measure execution time?

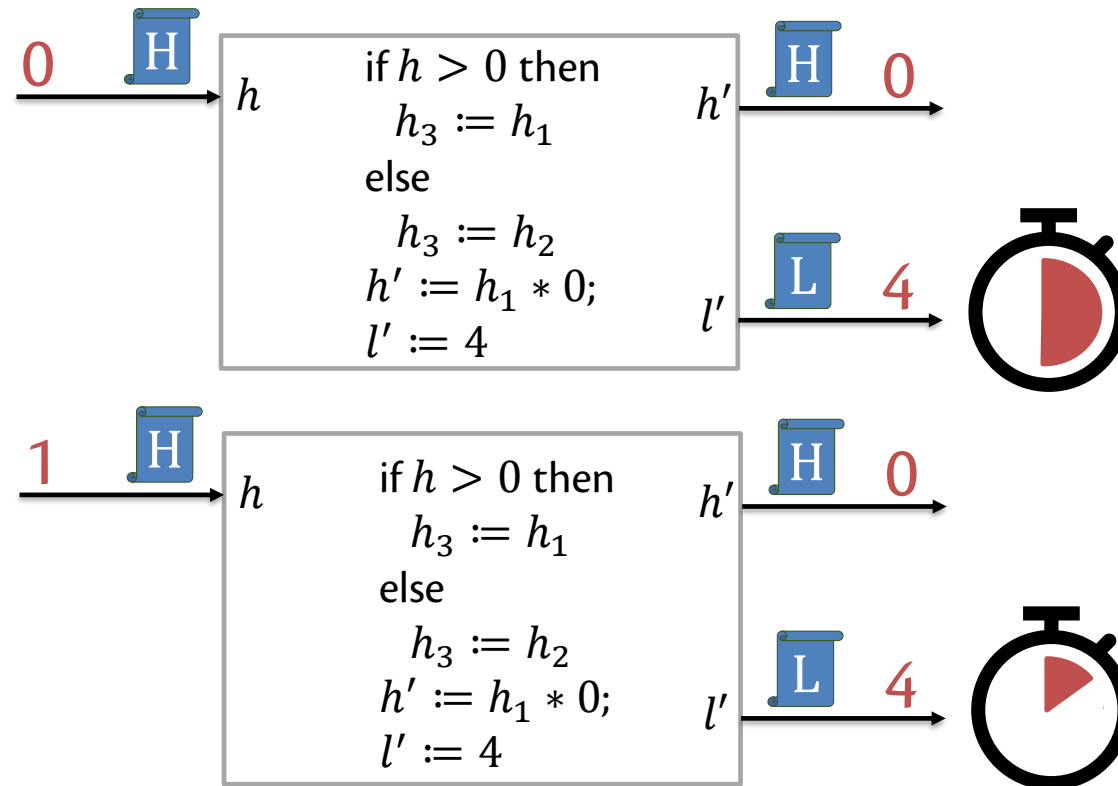
Adesso andiamo a vedere un altro esempio, sempre sul threat model e supponiamo che l'attaccante quindi il primo caso non osserva la terminazione quindi ha un tempo fissato per andare a vedere il programma, nel secondo caso l'attaccante non ha un limite temporale all'osservazione, quindi vuol dire che può anche vedere la non terminazione, il terzo caso invece se può proprio andare a controllare fisicamente il tempo di esecuzione, cioè quindi vuol dire che cerchiamo di estendere la nozione di non interferenza oltre al caso in cui l'attaccante fa dei conti temporali

Qui graficamente il modello l' attaccante è come un cronometro. Il primo dice, se h maggiore di zero allora h' prende 3 e poi h' prende 3. Quindi facciamo un'operazione che impiega più tempo ridondante. Altrimenti h' uguale a 3. Infine l' uguale a 4. Il risultato in termini di alto livello, di valori di alto livello è 3, in valore di basso livello è 4, è banale, solo che ovviamente il ramo then ci mette di più del ramo else perché ho ridondanza nell'esecuzione. A questo punto il secondo esempio, vedete fa la stessa cosa, dandogli come input zero e vedete che scopre nel secondo caso che dando l' input zero è vero che ha gli stessi risultati di input/output, ma può scoprire che ci impiega tempo differente, quindi andando a utilizzare dell'informazione di natura temporale, scopre e riesce a dedurre informazioni su 0/1 della variabile di alto livello e quindi vuol dire riesce a determinare dei dati sensibili relativamente alla variabile di alto livello.

TIMING CHANNELS



Assume h_1, h_2, h_3 are high memory addresses that can be cached.

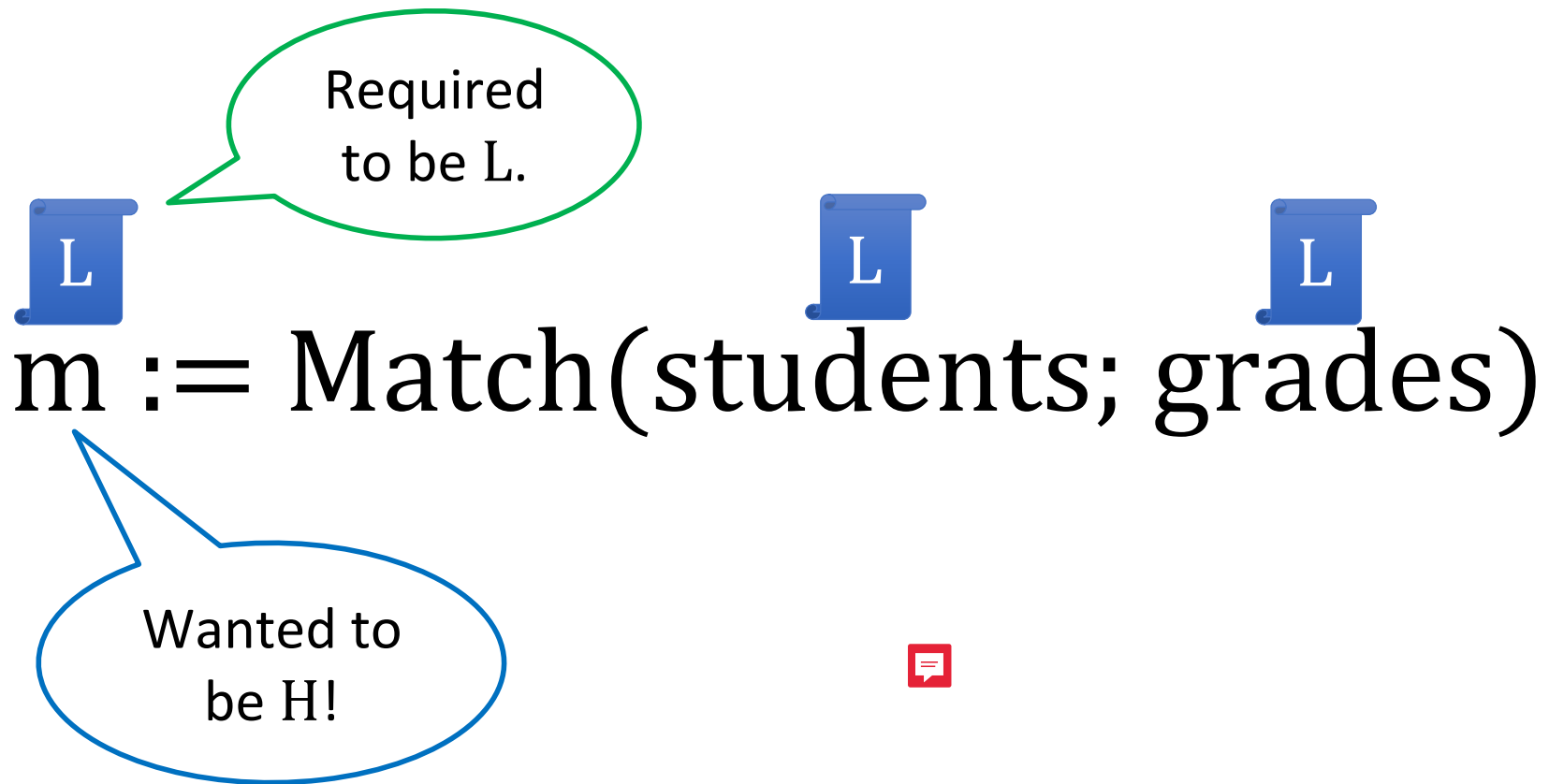


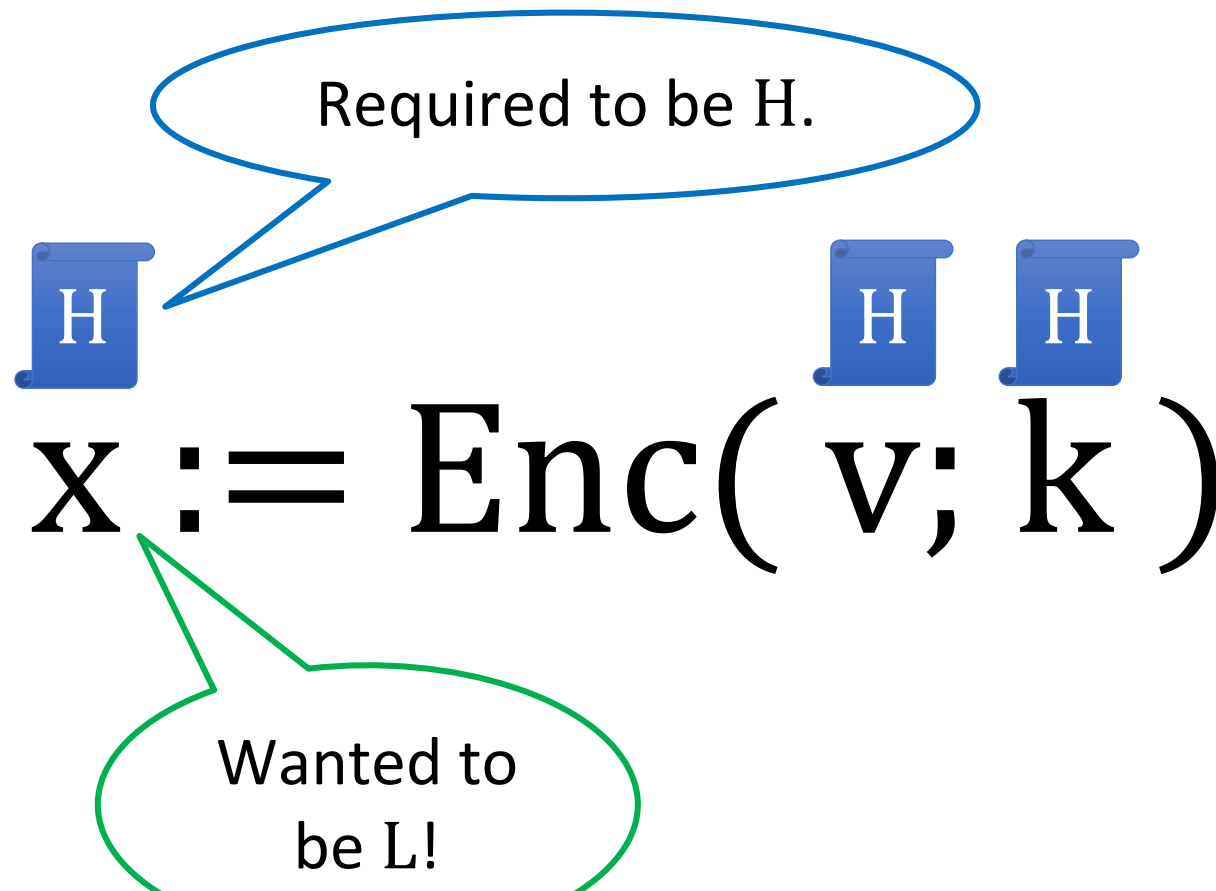
Questo era un esempio in cui davo alimentava il programma con 0 1 e andava a vedere il risultato sull'IO e vedete il risultato input output era esattamente lo stesso però se l' attaccante è in grado di controllare il tempo vede in modo particolare che se h_1 h_2 h_3 sono memorizzati nella cache in un caso o nell'altro ci mette minor tempo e quindi riesce proprio a derivare informazioni sui valori di alto livello andando ad avere cover channel relativamente a valori che sono messi in cache.

Discussion

- The notion of Non Interference may be too restrictive than necessary.
- Certain operations may increase the restrictions imposed on outputs comparing to restrictions imposed on inputs

allora il punto è che abbiamo diverse nozioni di non interferenza abbiamo quella sensitiva alla terminazione, quella non sensitiva alla terminazione, quella che riesce anche a fare un po' di informazione di timing e quindi in questo in questo caso andare a operare e dedurre informazioni sui valori di alto e basso livello, però può essere molto restrittiva, perché in alcuni casi io potrei voler avere dei meccanismi di declassificazione, cioè se io ad esempio dico valori di alto e basso livello, poi i valori di alto livello devono essere usati in termini delle informazioni di alto livello e in alcuni casi potrei volere essere un po' più flessibile, in modo particolare quando voi pagate con la carta di credito sul web, in un sito di commercio elettronico, quando la transazione è terminata cosa vi viene fuori come risultato? A parte che il pagamento è andato a buon fine, che tipo di informazione viene fuori sulla vostra carta di credito? Tipicamente vengono fuori i primi quattro bit che dicono su quale circuito siete e gli ultimi quattro bit della carta di credito. Però è strano perché la vostra carta di credito è un valore high, confidenziale e viene mandata su un canale pubblico, che ha valore low, quindi vuol dire che o i sistemi di pagamento non sono sicuri rispetto all' information flow, perché abbiamo un flusso di informazione sensibile che va su un canale pubblico oppure c'è qualcosa di mezzo che è evidente che rende ancora sicura il trattamento, pur permettendo un flusso di informazione dall'alto verso il basso.





andata a vedere quest'altro esempio, supponete di avere un meccanismo di encryption dei dati privati rispetto a una chiave ovviamente la chiave è alta, ovviamente il dato privato è alto, quindi a quel punto nell'esempio che abbiamo visto, prendiamo il sup, bene il risultato dell'encryption è alto livello, allora a questo punto non la mandiamo su un canale pubblico, perché avremo un leakage di information flow, però noi vogliamo mandare il dato cifrato sul canale pubblico, quindi vuol dire che lo dobbiamo declassificare a low.

Declassification

- **What:** specify what information may be declassified
 - e.g., LastFourDigits(credit card number) should be low
- **Who:** specify who may declassify information
 - e.g., high object owner can write to low objects
- **Where:** specify which pieces of code may declassify
 - e.g., encryption function can write to low objects
- **When:** specify when information may be declassified
 - e.g., software key may be shared after payment has been received

Vuol dire che per usarla in pratica non solo dobbiamo avere un meccanismo di tracciare il flusso di informazioni ma abbiamo anche metterci nei meccanismi di classificazione: ad esempio, che cosa deve essere declassificato, ad esempio i 4 numeri finali della carta di credito, ad esempio che facevano prima, devono essere di basso livello, possono essere messi su un canale pubblico, chi declassifica? O l'utente titolare che può scrivere su oggetti di basso livello. Dove lo può fare? In particolari spazi del codice, ad esempio il risultato dell'encryption è sicuramente high ma quando io voglio mandare su un socket allora in quel caso li devo poterla declassificare a low perché deve fluire su un canale di comunicazione pubblica. Quando ad esempio posso fare vedere i quattro numeri finale della carta di credito? Quando il pagamento è stato effettuato, non prima. Quindi ci sono tutti dei meccanismi che permettono di declassificare il flusso dell'informazione.

Information Flow Control

- a lattice L, \sqsubseteq of security levels (labels),
- a program,
- an environment Γ that maps variables to labels.

Threat model

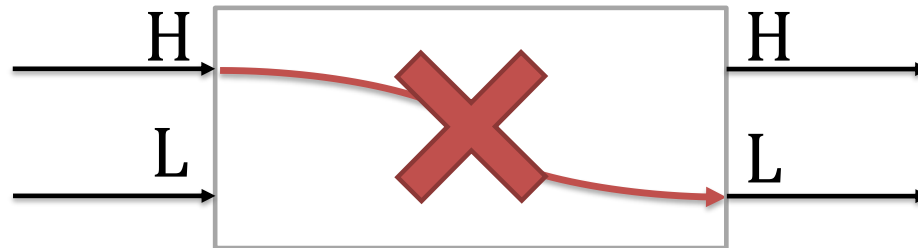
**Attacker knows L inputs,
knows parts of program code,
and can observe L outputs.**



Allora riassumendo, quello che abbiamo fatto, information flow parte da una nozione di reticolo, il reticolo definisce le proprietà di sicurezza, il programma è un ambiente che mappa le variabili al loro livello di sicurezza. Qual è il modello dell attaccante? Vede il programma, vede l' input in ingresso e all' in uscita e può conoscere parte del programma, può sapere se il programma ad esempio può terminare o no. Può fare dei controlli di timing allora a questo punto l' attaccante può soltanto vedere però i risultati e può influenzare il basso livello. Si possono dare delle nozioni di non interference con questi modelli dell attaccante.

Executions of the program should cause only allowed flows (with respect to the lattice).

**The program is expected to satisfy noninterference (NI):
Different H inputs, keeping L inputs fixed, should not cause
different L outputs.**



A questo punto, quello che noi abbiamo dato, è una nozione di non interference nel senso semplice, dove vogliamo evitare che il risultato di basso livello sia influenzato da un valore di alto livello, quindi vuol dire che non ci sono dei leakage di informazione dall'alto verso il basso e questo ci dice il modello del comportamento, la nozione formale del modello del comportamento e il modello dell'attaccante.



NEXT: ENFORCEMENT MECHANISM STATIC TYPE SYSTEM

Andremo a vedere dei meccanismi di enforcement, in modo particolare, andremo a vedere un linguaggio di programmazione, andremo a vedere un semplice linguaggio imperativo, andremo a vedere come può un sistema di tipi statico garantire delle proprietà di non interferenza e poi faremo vedere come si può estendere questo ad esempio al caso dell' object orientation.