Workshop in Issues in the Theory of Security

Static Detection of Logic Flaws in Service Applications

C. Bodei[1], L. Brodo[2], R. Bruni[1]

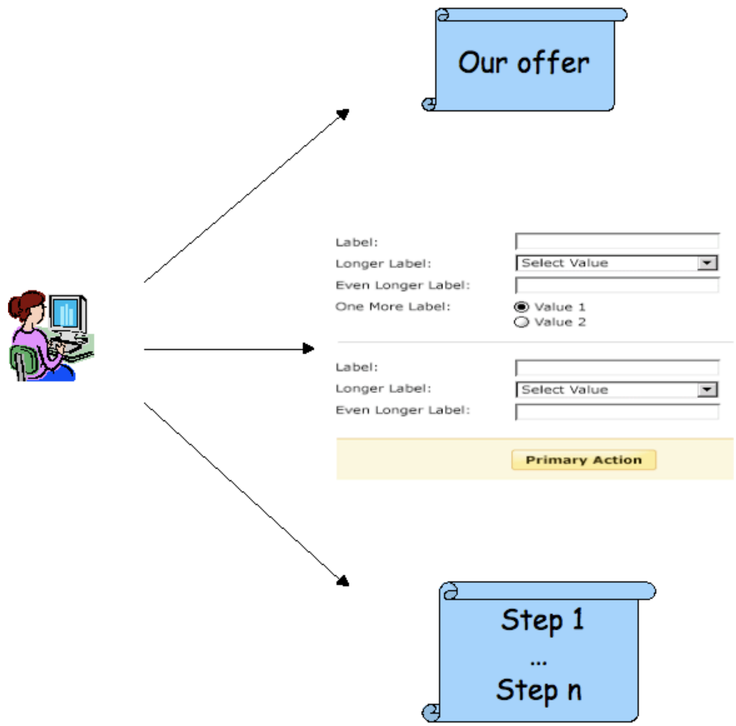
[1] Dipartimento di Informatica, Università di Pisa

[2] Dipartimento di Scienze dei Linguaggi, Università di Sassari

Web services scenario

Customers are supposed to precisely follow the intended order of the transaction steps

Web services

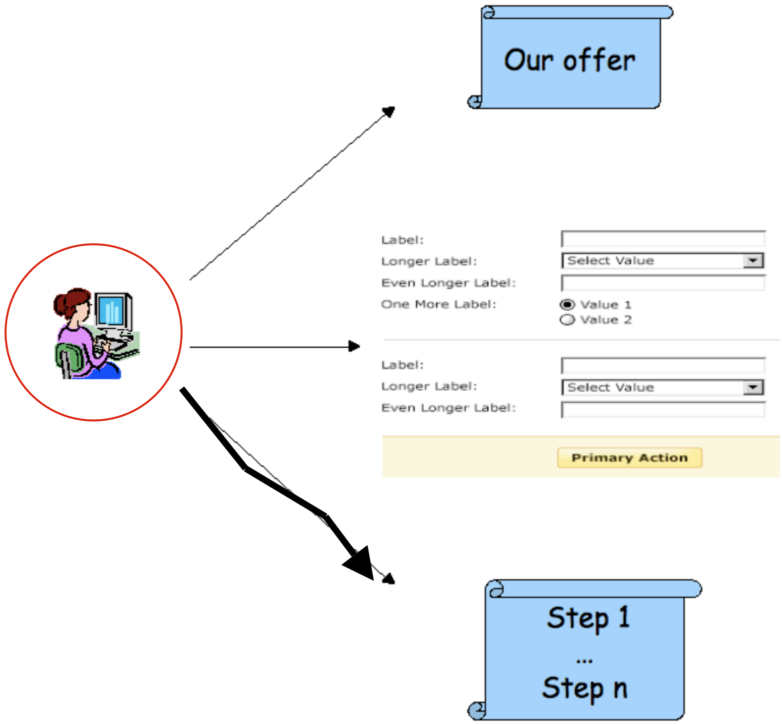


A claimed service

Opening a session

Executing the steps

Web services



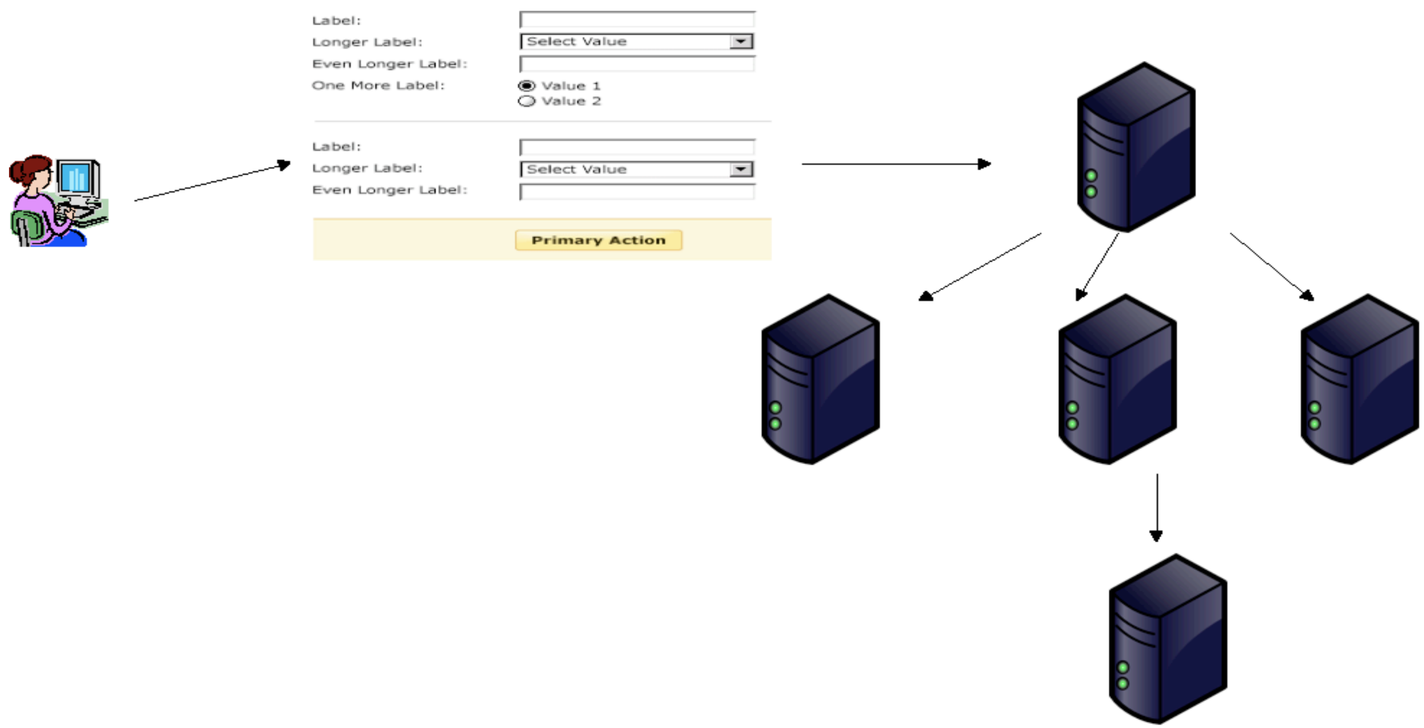
A claimed service

Opening a session

Executing the steps

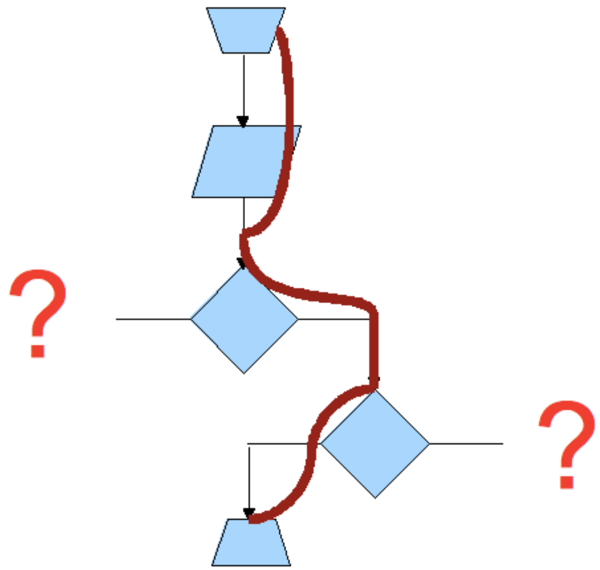
A new kind of attacker, different from the Dolev-Yao: the malicious costumer

Web services



Nested web service calls are usual: the API mechanism

Web service logics



There is a claimed goal, **but**
there could be other hidden functionalities !

Intuitive Idea

[cryptography is perfect]

Semantic Security Attacks : Crypto-protocols

=

Application Logic Attaks : Service Specifications

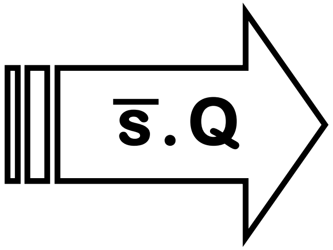
[underlying protocols are perfect]

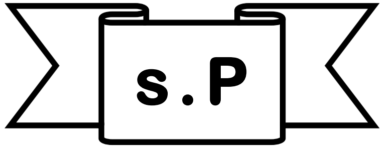
A language for web service: CaSPiS

P ::=	
s.P	service definition
$\overline{v}$.P	service invocation
$\Sigma\ \pi$.P	guarded sum
$r^p \triangleright$ P	run-time session
P > (?x) Q	pipeline
(ν n) P	restriction
P|Q	parallel
!P	replication

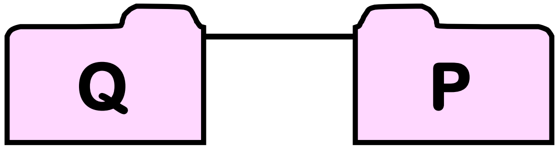
p, q ::=	
+ | −	session polarities
π , π' ::=	
(?x)	input
<V>	output
<V>$^\uparrow$	session return

A language for web service: CaSPiS

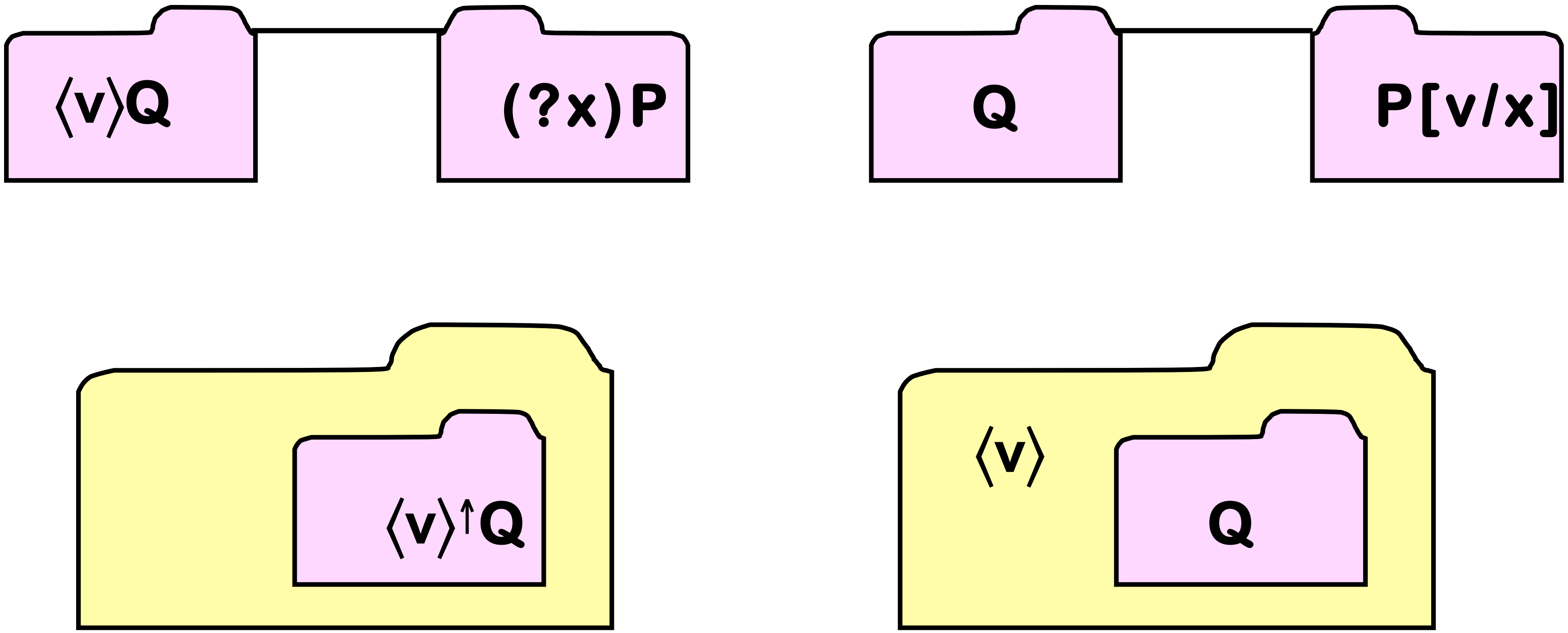




Ex. of service invocation a run time : $\bar{s}.P \mid s.Q \;\rightarrow\; (\nu\, r)\, r^{-} \triangleright P \mid r^{+} \triangleright Q$



A language for web service: CaSPiS



Bank Credit Request example

$$S = Bank \mid Controller \mid Client$$

$$Bank = req.(?y_{ba})\ \overline{chk}.\langle y_{ba}\rangle(?w_{ans}).\langle w_{ans}\rangle^{\uparrow}$$

$$Controller = chk.(?z_{ba})\langle ans\rangle$$

$$Client = \overline{req}.\ \langle ba\rangle(?x_{ans})\langle ans\rangle^{\uparrow}$$

- **req** is the service definition of the Bank;
- bank invokes the **chk** service offered by the Controller to check the client balance asset

BCR example

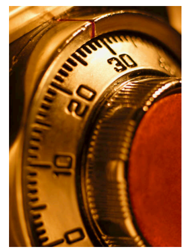
S= Bank | Controller | Client

$\text{Bank} = \text{req}.(?y_{ba})\ \overline{\text{chk}}.\langle y_{ba}\rangle(?w_{ans}).\langle w_{ans}\rangle^{\uparrow}$

$\text{Controller} = \text{chk}.(?z_{ba})\langle \text{ans}\rangle$

$\text{Client} = \overline{\text{req}}.\ \langle \text{ba}\rangle(?x_{ans})\langle \text{ans}\rangle^{\uparrow}$

$\overline{\text{req}}$

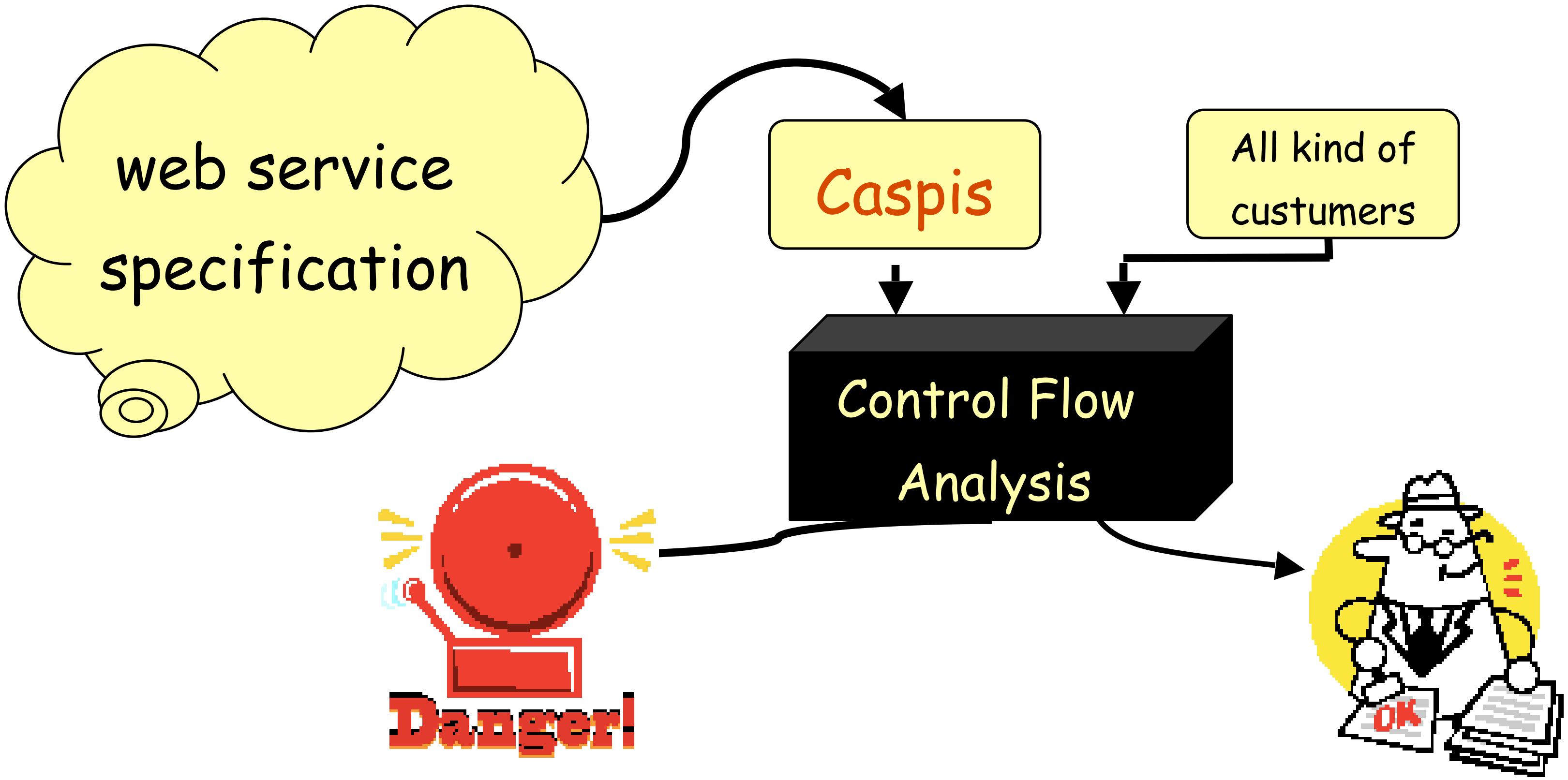


$\overline{\text{chk}}$

$S \longrightarrow (\nu\ r_{req})\ (r_{req}{}^{+} \triangleright (?\ y_{ba}) \ldots \mid r_{req}{}^{-} \triangleright \langle ba\rangle \ldots) \mid \text{Contr} = S'$

$S' \longrightarrow (\nu\ r_{req})\ (r_{req}{}^{+} \triangleright \overline{\text{chk}}.\langle ba\rangle \ldots \mid r_{req}{}^{-} \triangleright (?z_{ba}) \ldots) \mid \text{Contr}$

Static analysis, framework



CFA analysis

$\mathcal{I}$ records which action and service prefixes are included in the scope due to services, sessions and pipelines

$\mathcal{R}$ maps a variable to the set of names it can be bound to

σ records the actual position in the nested structure of sessions and pipelines

$$\mathcal{I},\mathcal{R} \models^{\sigma} P$$

In two steps:

1. analysing the nested structure
2. approximating the execution

BCR example

S= Bank | Controller | Client

Bank = $req.(?y_{ba})\ \overline{chk}.\langle y_{ba}\rangle(?w_{ans}).\langle w_{ans}\rangle^{\uparrow}$

Controller = $chk.(?z_{ba})\langle ans\rangle$

Client = $\overline{req}.\ \langle ba\rangle(?x_{ans})\langle ans\rangle^{\uparrow}$

$$S \longrightarrow (\nu\ r_{req})\ (r_{req}\ {}^{+}\triangleright (?\ y_{ba}) \ldots \mid r_{req}\ {}^{-}\triangleright \langle ba\rangle \ldots) \mid Contr = S'$$

$$S' \longrightarrow (\nu\ r_{req})\ (r_{req}\ {}^{+}\triangleright \overline{chk.\langle ba\rangle} \ldots \mid r_{req}\ {}^{-}\triangleright\ (?z_{ba})\ldots) \mid Contr$$

CFA at work

First step:

$\mathcal{I},\mathcal{R} \models^{\sigma} \text{Bank} \mid \text{Controller} \mid \text{Client}$

$\mathcal{I}(*) \ni \overline{req}, req, \ldots$

$\mathcal{I}(req) \ni (?\, y_{ba}), chk$

$\mathcal{R} = \emptyset$

I,R describes the initial process

Second step of the analysis:

$\mathcal{I}(*) \ni r_{req}^{+}\, r_{req}^{-}$

$\mathcal{I}(r_{req}^{+}) \ni (?\, y_{ba}), chk$

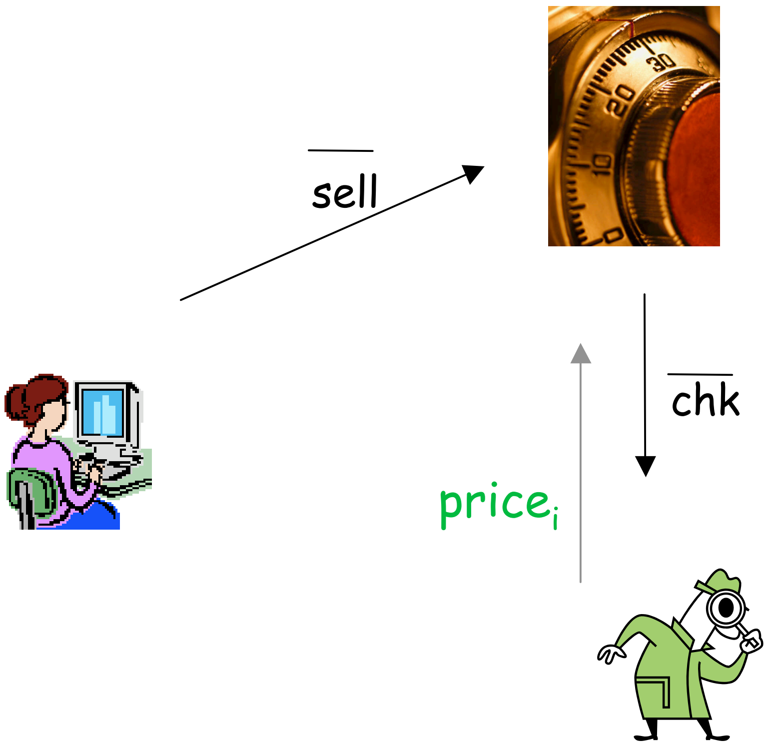
$\mathcal{I}(r_{req}^{-}) \ni \langle ba \rangle$

$\mathcal{R}(y_{ba}) \ni ba$

I,R takes the possible dynamics into account

On-line shop service example

S = (Shop | Price_chk) | Client



- the client invokes **sell** and chooses an item
- **sell** is the service definition of the Shop
- Shop invokes **chk** service offered by the Price_checker for the price of the item
- Price_checker comunicates the price directly to the client
- Shop does not check the price

On-line shop service example

$S = (Shop \mid Price\_chk) \mid Client$

Shop = sell.Σ_i ((item$_i$)

$(\overline{chk}.\langle item\rangle(x_{price}).\langle item, x_{price}\rangle^{\uparrow}$

|

(ok).(PAY, y_{price}) +

(ko)))

Price_chk = chk. Σ_i ((item$_i$) <price>)

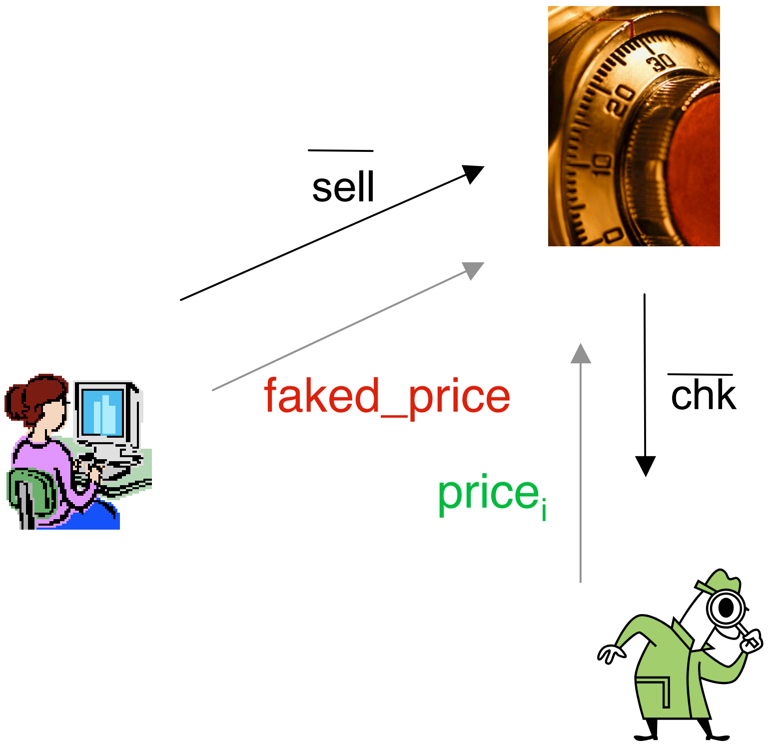
Client = $\underline{sell}$. <item$_i$>(item$_i$,x_{price}).

<ok, x_{price} > + <ko>

- the client invokes **sell** and chooses an item
- **sell** is the service definition of the Shop
- Shop invokes **chk** service offered by the Price_checker for the price of the item
- Price_checker comunicates the amount ot payment directly to the client
- Shop does not check the price

The attacker … at work

(Shop | Price_chk) | Client



- Shop does not check the price
- the maliciuos customer alters the price field, using a faked price

Which kind of attacker?

Different from Dolev-Yao attacker !

Modeling the attacker

Malicious customer's knowledge:

Synchronization in session r

$$\langle v \rangle \in I(r) \ \wedge \ (?x) \in I(r) \longrightarrow \ v \in R(x)$$

If malicious customer is executing input:

$v \in K$

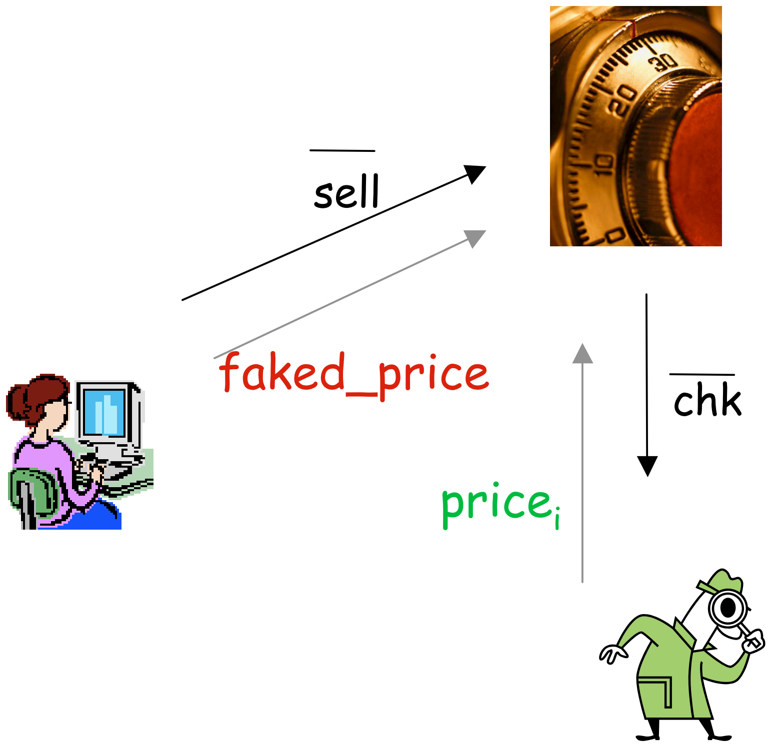
If malicious customer is executing output:

$$\forall\ v' : v' \in K \longrightarrow v' \in R(x)$$

(Knowledge Rule 1)	$v \in N$	$\longrightarrow$	$v \in K$
(Knowledge Rule 2)	$v_1,\ldots,v_n \in K$	$\longrightarrow$	$\langle v_1,\ldots,v_n \rangle \in K$
(Knowledge Rule 3)	$\langle v_1,\ldots,v_n \rangle \in K$	$\longrightarrow$	$v_1,\ldots,v_n \in K$

The attacker … at work

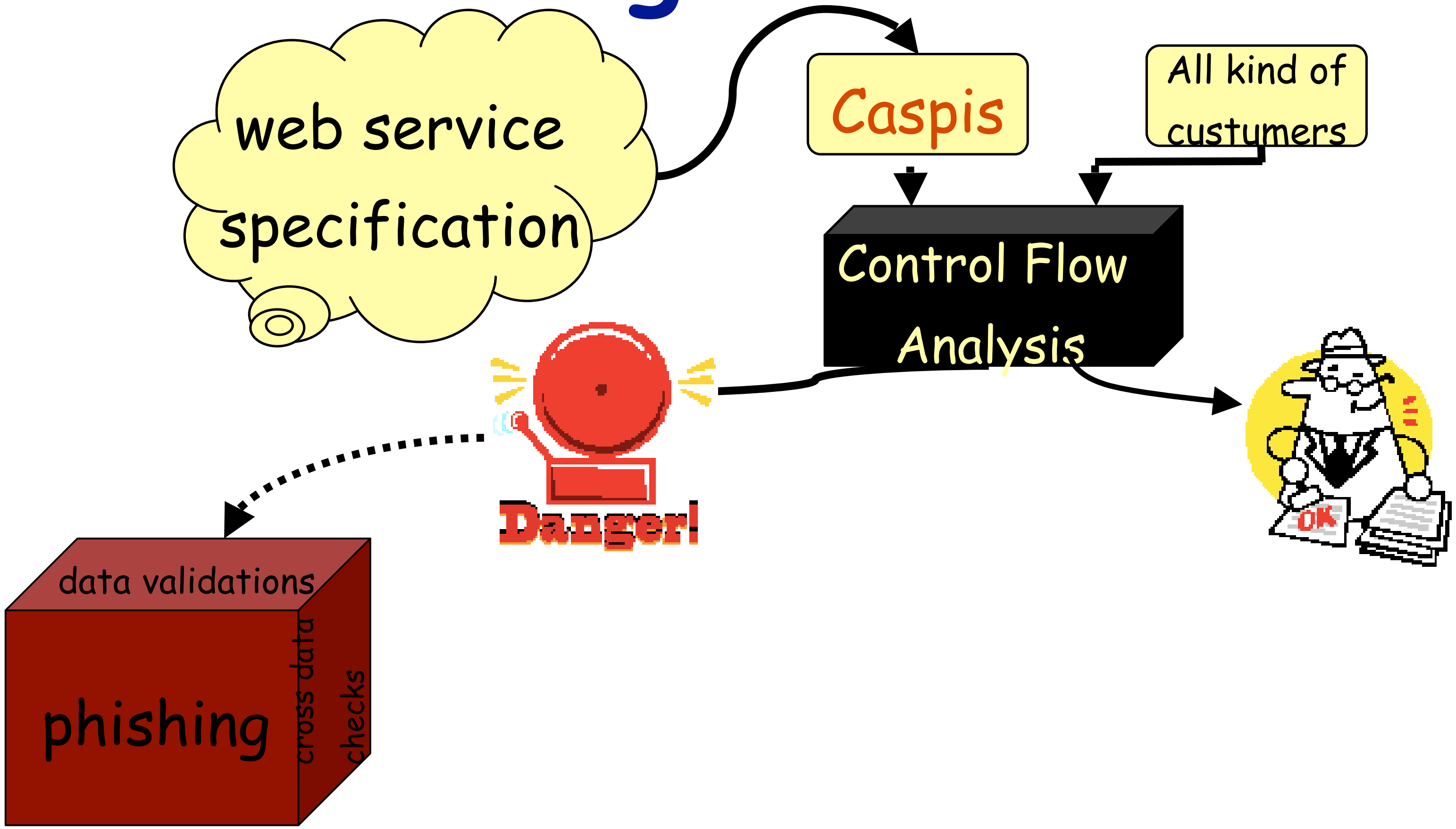
$I,R,K \models^{\sigma}$ (Shop | Price_chk) | Client



$K \ni \text{price}_i, \text{faked\_price}$

$R(y_{price}) \ni \text{price}_i, \text{faked\_price}$

CFA results: a fault in business logic



Conclusion

- We focus on the right level of abstraction to describe service application design (e.g. CaSPiS)
- We distil automatic techniques to detect logic flaws at design time:
  - Control Flow Analysis

Thank you.

On-line shop service example

$S = (Shop \mid Price\_chk) \mid Client$

$Shop = sell.\Sigma_i \, ((item_i)$

$(\overline{chk}.\langle item \rangle(x_{price}).\langle item, x_{price} \rangle^{\uparrow}$

$\mid$

$(ok).PAY$

$+$

$(ko) \,))$

$Price\_chk = chk. \, \Sigma_i \, ((item_i) \, \langle price \rangle)$

$Client = \overline{sell}. \, \langle item_i \rangle(item_i, x_{price}).$

$\langle ok, x_{price} \rangle + \langle ko \rangle$

- **sell** is the service definition of the Shop;
- Shop invokes **chk** service offered by the Price_checker
- Price_checker comunicates the amount ot payment directly to the client.

CFA at work

First step:

$I,R \models^{\sigma}$ (Shop | Price_chk) | Client

$I(*) \ni \underline{sell}, \overline{sell}$, chk

$I(sell) \ni chk$

$R = \emptyset$

Second step of the analysis:

$I,R \models^{\sigma} S'$

$I(*) \ni r\text{-}_{sell}, r\text{-}_{chk}$

$I(r+_{sell}) \ni r\text{-}_{chk}$

$R = ...$

Stopping the attacker

Shop $= sell.\Sigma_i\ ((item_i)$

$(\overline{chk}.\langle item\rangle(x_{price}).\langle x_{price}\rangle^{\uparrow} >\ (y_{price})\ \langle item,\ y_{price}\rangle$

$(\ (ok,\ y_{price}).PAY$

$+$

$(ko)\ ))$

Price_chk $= chk.\ \Sigma_i\ ((item_i)\ \langle price\rangle)$

Client $= \overline{sell}.\ \langle item_i\rangle(item_i, x_{price}).$

$\langle ok,\ x_{price}\rangle + \langle ko\rangle$