Workshop in Issues in the Theory of Security

Static Detection of Logic Flaws in Service Applications

C. Bodei\textsuperscript{1}, L. Brodo\textsuperscript{2}, R. Bruni\textsuperscript{1}

\textsuperscript{1} Dipartimento di Informatica, Università di Pisa

\textsuperscript{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

Opening a session

Executing the steps

A claimed service
Web services

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

A claimed service

Opening a session

Executing the steps
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 Attacks : Service Specifications
[underlying protocols are perfect]
A language for web service: CaSPiS

\[
P ::= s.P \quad \text{service definition}
\]

\[
\bar{v}.P \quad \text{service invocation}
\]

\[
\Sigma \pi .P \quad \text{guarded sum}
\]

\[
r^p \triangleright P \quad \text{run-time session}
\]

\[
P > (\exists x) Q \quad \text{pipeline}
\]

\[
(\forall n) P \quad \text{restriction}
\]

\[
P \mid Q \quad \text{parallel}
\]

\[
!P \quad \text{replication}
\]

\[
p, q ::= + | - \quad \text{session polarities}
\]

\[
\pi, \pi' ::= (\exists x) \quad \text{input}
\]

\[
< v > \quad \text{output}
\]

\[
< v > \uparrow \quad \text{session return}
\]
A language for web service: CaSPiS

Ex. of service invocation a run time: \( \overline{s.P} \mid s.Q \rightarrow (v \ r) r^{-}\rightarrow P \mid r^{+}\rightarrow Q \)
A language for web service: CaSPiS

\[
\langle v \rangle Q \quad (?x)P \quad Q \quad P[v/x]
\]

\[
\langle v \rangle^\uparrow Q \quad \langle v \rangle \quad Q
\]
Bank Credit Request example

\[ S = \text{Bank} \mid \text{Controller} \mid \text{Client} \]

\[
\text{Bank} = \text{req}.\langle ?y_{ba} \rangle \text{chk}.\langle y_{ba} \rangle \langle ?w_{\text{ans}} \rangle .\langle w_{\text{ans}} \rangle ^\uparrow
\]

\[
\text{Controller} = \text{chk}.\langle ?z_{ba} \rangle \langle \text{ans} \rangle
\]

\[
\text{Client} = \text{req}. \langle ba \rangle \langle ?x_{\text{ans}} \rangle .\langle \text{ans} \rangle ^\uparrow
\]

- \text{req} is the service definition of the Bank;
- bank invokes the \text{chk} service offered by the Controller to check the client balance asset
BCR example

\[ S = \text{Bank} \mid \text{Controller} \mid \text{Client} \]

\[
\begin{align*}
\text{Bank} & = \text{req.}(?y_{ba}) \quad \text{chk.}\langle y_{ba} \rangle(?w_{ans}).\langle w_{ans} \rangle^+ \\
\text{Controller} & = \text{chk.}(?z_{ba})\langle \text{ans} \rangle \\
\text{Client} & = \text{req.} \langle ba \rangle(?x_{ans})\langle \text{ans} \rangle^+ \\
\end{align*}
\]

\[
\begin{align*}
S & \rightarrow (\forall r_{req}) (r_{req} \triangleright (?y_{ba}) \ldots \mid r_{req} \triangleright \langle ba \rangle \ldots) \mid \text{Contr} = S' \\
S' & \rightarrow (\forall r_{req}) (r_{req} \triangleright \text{chk.}\langle ba \rangle \ldots \mid r_{req} \triangleright (?z_{ba}) \ldots) \mid \text{Contr}
\end{align*}
\]
Static analysis, framework

web service specification

Caspis

Control Flow Analysis

All kind of custumers

Danger!

OK
CFA analysis

I  records which action and service prefixes are included in the scope due to services, sessions and pipelines
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

I,R ⊢ σ  P

In two steps:
1. analysing the nested structure
2. approximating the execution
BCR example

\[ S = \text{Bank} \mid \text{Controller} \mid \text{Client} \]

\begin{align*}
\text{Bank} & = \text{req.}(?y_{ba}) \quad \text{chk.} <y_{ba}> (?w_{ans}).<w_{ans}>^\dagger \\
\text{Controller} & = \text{chk.}(?z_{ba})<\text{ans}> \\
\text{Client} & = \text{req.} <ba>(?x_{ans})<\text{ans}>^\dagger \\
\end{align*}

\[ S \rightarrow (\nu \text{req}) \ (\text{req} \xrightarrow{\dagger} (?y_{ba}) \ldots \mid \text{req} \xrightarrow{\dagger} <ba>\ldots) \mid \text{Contr} = S' \]

\[ S' \rightarrow (\nu \text{req}) \ (\text{req} \xrightarrow{\dagger} \text{chk.}<ba> \ldots \mid \text{req} \xrightarrow{\dagger} (?z_{ba})\ldots) \mid \text{Contr} \]
CFA at work

First step:

\[ I,R \models_{\sigma} \text{Bank | Controller |Client} \]

\[ I(R) \models (\text{? y}_{\text{ba}}), \text{chk} \]
\[ R = \emptyset \]

\[ I(\ast) \models \text{req,req, ...} \]

\[ I(R) \models (\text{? y}_{\text{ba}}), \text{chk} \]
\[ R = \emptyset \]

\[ I(\ast) \models r_{\text{req}}^+ r_{\text{req}}^- \]
\[ I(r_{\text{req}}^+) \models (\text{? y}_{\text{ba}}), \text{chk} \]
\[ I(r_{\text{req}}^-) \models \text{<ba>} \]
\[ R(y_{\text{ba}}) \models \text{ba} \]

Second step of the analysis:

\[ I,R \text{ takes the possible dynamics into account} \]
On-line shop service example

\[ S = (\text{Shop} \mid \text{Price_chk}) \mid \text{Client} \]

- the client invokes \textit{sell} and chooses an item
- \textit{sell} is the service definition of the Shop
- Shop invokes \textit{chk} service offered by the Price_checker for the price of the item
- Price_checker communicates the price directly to the client
- Shop does not check the price
On-line shop service example

\[ S = (\text{Shop} \mid \text{Price\_chk}) \mid \text{Client} \]

\text{Shop} = \text{sell.} \Sigma_i ((\text{item}_i))

\[ \left( \text{chk.} \langle \text{item} \rangle (x_{price}).\langle \text{item}, x_{price} \rangle \uparrow \right. \]

\mid

\left. (\text{ok}).(\text{PAY}, y_{price}) + \right. \]

\left. \left( \text{ko} \right) \right)

\text{Price\_chk} = \text{chk.} \Sigma_i ((\text{item}_i) \langle \text{price} \rangle)

\text{Client} = \text{sell.} \langle \text{item}_i \rangle (\text{item}_i, x_{price}).\langle \text{ok}, x_{price} \rangle + \langle \text{ko} \rangle

- the client invokes \textbf{sell} and chooses an item
- \textbf{sell} is the service definition of the Shop
- Shop invokes \text{chk} service offered by the Price\_checker for the price of the item
- Price\_checker comunicates the amount of 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 malicious 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 $r$

$$\langle v \rangle \in I(r) \land (\exists x) \in I(r) \rightarrow v \in R(x)$$

If malicious customer is executing input:

$v \in K$

If malicious customer is executing output:

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

(Knowledge Rule 1)  $v \in N \rightarrow v \in K$

(Knowledge Rule 2)  $v_1,\ldots,v_n \in K \rightarrow \langle v_1,\ldots,v_n \rangle \in K$

(Knowledge Rule 3)  $\langle v_1,\ldots,v_n \rangle \in K \rightarrow v_1,\ldots,v_n \in K$

28-29 March 2009  WITS'09 - York, UK
The attacker ... at work

\[ I,R,K \xrightarrow{\sigma} (\text{Shop} \mid \text{Price_chk}) \mid \text{Client} \]

- sell
  - \text{faked\_price}
    - price\_i
  - \text{chk}
    - R(y_{price}) \ni price\_i, faked\_price
    - K \ni price\_i, faked\_price

28-29 March 2009  WITS'09 - York, UK
CFA results: a fault in business logic

- Web service specification
- Control Flow Analysis
- Caspis
- All kind of customers
- Data validations
- Cross-data checks
- Phishing
- Danger!
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

\[ S = (\text{Shop} \mid \text{Price\_chk}) \mid \text{Client} \]

\begin{align*}
\text{Shop} & = \text{sell}. \Sigma_i ((\text{item}_i) \\
& \quad (\text{chk.}<\text{item}(x_{\text{price}}).<\text{item}, x_{\text{price}}>)^+ \\
& \quad | \\
& \quad (\text{ok}).\text{PAY} \\
& \quad + \\
& \quad (\text{ko}) )) \\
\text{Price\_chk} & = \text{chk.} \Sigma_i ((\text{item}_i) <\text{price}> ) \\
\text{Client} & = \text{sell.} <\text{item}_i>(\text{item}_i, x_{\text{price}}). <\text{ok}, x_{\text{price}}> + <\text{ko}> \\
\end{align*}

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

First step:

\[ I, R \xrightarrow{\sigma} (\text{Shop} \mid \text{Price_chk}) \mid \]

\[ \text{Client} \]

I(\ast) \ni \text{sell}, \text{sell}, \text{chk}

I(sell) \ni \text{chk}

R = \emptyset

Second step of the analysis:

\[ I, R \xrightarrow{\sigma} S' \]

I(\ast) \ni r^-_{\text{sell}}, r^-_{\text{chk}}

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

R = \text{.....}
Stopping the attacker

\[
\text{Shop} = \text{sell.} \sum_i ((\text{item}_i) \\
(\text{chk.}<\text{item}>(<x_{\text{price}}>) <x_{\text{price}}> \uparrow > (y_{\text{price}}) <\text{item}, y_{\text{price}}> \\
(\text{ok}, y_{\text{price}}).\text{PAY} \\
+ \\
(\text{ko})))
\]

\[
\text{Price_chk} = \text{chk.} \sum_i ((\text{item}_i) <\text{price}>)
\]

\[
\text{Client} = \overline{\text{sell.}} <\text{item}_i>(\text{item}_i, x_{\text{price}}). \\
<\text{ok}, x_{\text{price}} > + <\text{ko}>
\]