A Role for Theory in UbiNet

UbiNet Workshop: London 26.09.03

Vladimiro Sassone

University of Sussex, 

Between Theory and Practice

- Claim 1: Cryptography is all that there is to Computer Security;

Between Theory and Practice

- Claim 1: Cryptography is all that there is to Computer Security;
- Wrong: Eg, 802.11 WEP key can be guessed looking at enough messages.

Between Theory and Practice

- Claim 1: Cryptography is all that there is to Computer Security;
- Wrong: Eg, 802.11 WEP key can be guessed looking at enough messages.
- Claim 2: UbiComp is about developing (the right) middleware.

Between Theory and Practice

- Claim 1: Cryptography is all that there is to Computer Security;
- Wrong: Eg, 802.11 WEP key can be guessed looking at enough messages.

- Claim 2: UbiComp is about developing (the right) middleware.
- Wrong: Notions and issues are "horizontal" (data, resources, ...); people/applications/data will move across "global computers."

Between Theory and Practice

- Claim 1: Cryptography is all that there is to Computer Security;
- Wrong: Eg, 802.11 WEP key can be guessed looking at enough messages.
- Claim 2: UbiComp is about developing (the right) middleware.
- Wrong: Notions and issues are "horizontal" (data, resources, ...); people/applications/data will move across "global computers."
- Claim 3: So, in conclusion, Theory is all you need ...

Between Theory and Practice

- Claim 1: Cryptography is all that there is to Computer Security;
- Wrong: Eg, 802.11 WEP key can be guessed looking at enough messages.

- Claim 2: UbiComp is about developing (the right) middleware.
- Wrong: Notions and issues are "horizontal" (data, resources, ...); people/applications/data will move across "global computers."

- Claim 3: So, in conclusion, Theory is all you need ...
- Wrong too, I'm afraid. Eg SSL 3.0 has be proved correct by (computer-aided) model checking. Alas, real attacks have been found in practice.

Between Theory and Practice

- Claim 1: Cryptography is all that there is to Computer Security;
- Wrong: Eg, 802.11 WEP key can be guessed looking at enough messages.
- Claim 2: UbiComp is about developing (the right) middleware.
- Wrong: Notions and issues are "horizontal" (data, resources, ...); people/applications/data will move across "global computers."
- Claim 3: So, in conclusion, Theory is all you need ...
- Wrong too, I'm afraid. Eg SSL 3.0 has be proved correct by (computer-aided) model checking. Alas, real attacks have been found in practice.

The gap between Theory and Practice matters in Practice
(although it may not in Theory)

Theory: Success Stories

Past

- Types and language safety in programming;
- Static program analysis and model checking;
- Abstraction and modularisation mechanisms;
- ...

Ok, but is Theory going to be relevant for UbiComp ??

Theory: Success Stories

Past

- Types and language safety in programming;
- Static program analysis and model checking;
- Abstraction and modularisation mechanisms;
- ...

Current (still partial)

- Memory safe languages:
- Curing legacy code (CCured); Tricking users into safe langs (Cyclone);
- Verification-driven protocol design (Fair Exchange, Private Authentication, ...);
- Secrecy and Information Confinement;
- Proof carrying code; Typed assembly languages;
- ...

Ok, but is Theory going to be relevant for UbiComp ??

Type Systems in Programming Languages

Why Types?

- Types are (statically-verified) invariants which characterise values and (the result of) computations.
- Progress & Safety: Well typed code has some good property Prop (which usually implies they can progress).
- Subjection reduction: Well typedness is preserved by program steps.
- Consequence: Prop is invariant for good programs (eg they are never stuck nor crash on illegal ops).
- Note:Static means that types are removed at runtime, and have no runtime overheads.

Types for Resource Usage

Elementary Concurrent Scenario

$$P ::= ch(x).P \mid ch\langle v\rangle.P \mid \ldots$$

A nave ping server:

$$\texttt{forever}\ \mathrm{ping}(r).r\langle\rangle$$

What can a type system do here?

Types for Resource Usage

Elementary Concurrent Scenario

$$P ::= ch(x).P \mid ch\langle v\rangle.P \mid \ldots$$

A nave ping server:

$$\texttt{forever}\ \ \mathrm{ping}(r).r\langle\rangle$$

What can a type system do here?

`1st:` Prevent errors of the kind $\mathrm{ping}\langle 3\rangle$, leading to $3\langle\rangle$.
Values classified in channels and numbers

$$T ::= \texttt{void} \mid \texttt{int} \mid \texttt{chan}[T]$$

Type for ping: $\mathrm{ping} : \texttt{chan}[\texttt{chan}[\texttt{void}]]$.

From here, the idea of usage control comes easily:

$\mathrm{ping}(r).(r\langle\rangle|r\langle\rangle)$: Duplicate answers;

$\mathrm{ping}(r).(r\langle\rangle|r(x)P)$: Intercept messages.

Types for Resource Usage

Elementary Concurrent Scenario

$$P ::= ch(x).P \mid ch\langle v\rangle.P \mid \ldots$$

A nave ping server:

$$\texttt{forever}\ \mathrm{ping}(r).r\langle\rangle$$

What can a type system do here?

Let's enrich types with "usage prescriptions"

$$T ::= \ldots \texttt{chan}[T^U] \qquad U ::= ?_m \mid !_m \mid ?_m!_n$$

If ping can be assigned

$$\mathrm{ping} : \texttt{chan}[\,\texttt{chan}[\texttt{void}?_0!_1]\,]$$

clients are safe.

Types for Resource Usage

Elementary Concurrent Scenario

$$P ::= ch(x).P \mid ch\langle v\rangle.P \mid \ldots$$

A nave ping server:

$$\texttt{forever}\ \mathrm{ping}(r).r\langle\rangle$$

What can a type system do here?

Let's enrich types with "usage prescriptions"

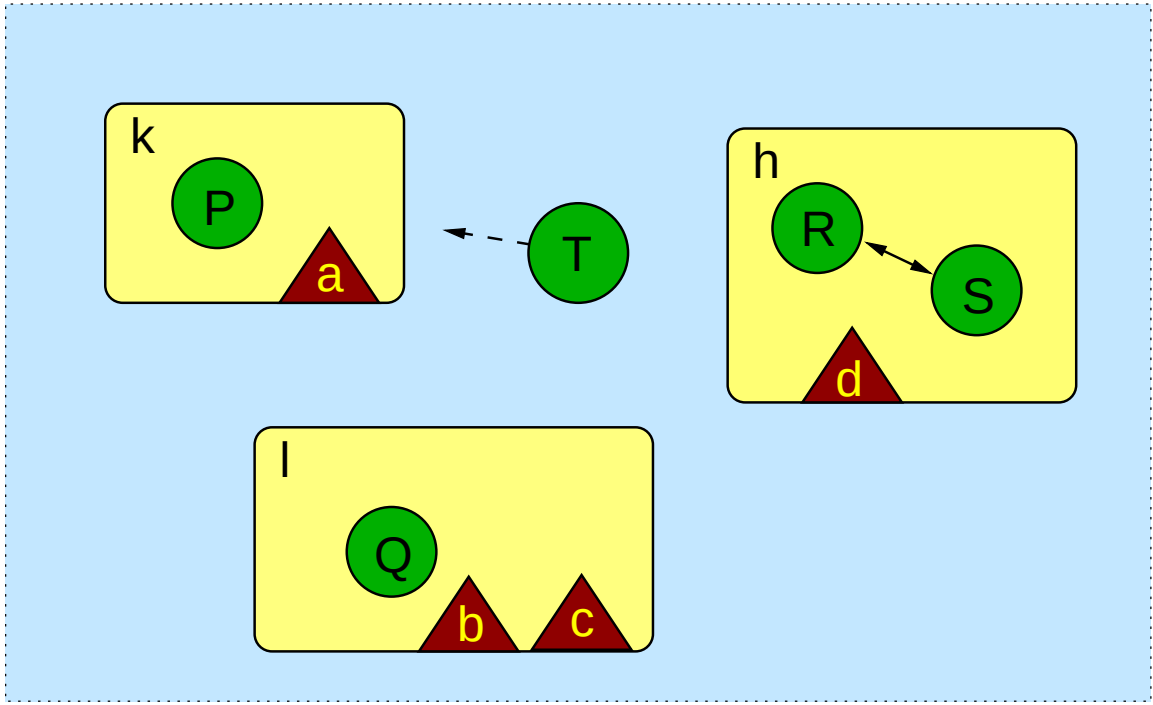
$$T ::= \ldots \texttt{chan}[T^U] \qquad U ::= ?_m \mid !_m \mid ?_m!_n$$

If ping can be assigned

$$\mathrm{ping} : \texttt{chan}[\,\texttt{chan}[\texttt{void}?_0!_1]\,]$$

clients are safe.
An impressive portfolio of applications: deadlock freeness, race conditions, data secrecy, ... has been developed.

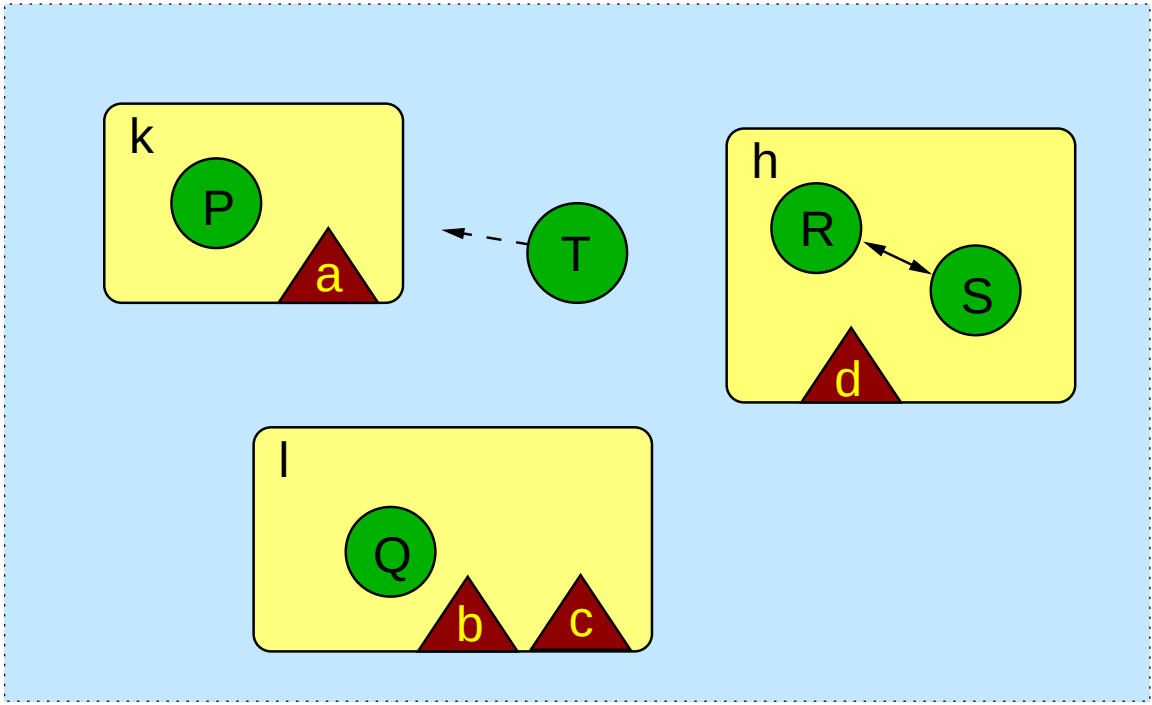
Locations and Border Xing



Global Ubiquitous Systems in first approximation consist of:

- A collection of independent distributed sites offering services/resources to migrating agents.
- Resources: All sort of things a agent may long for (CPU time, space, printers, ...).
- Agents: mobile processes of general nature.

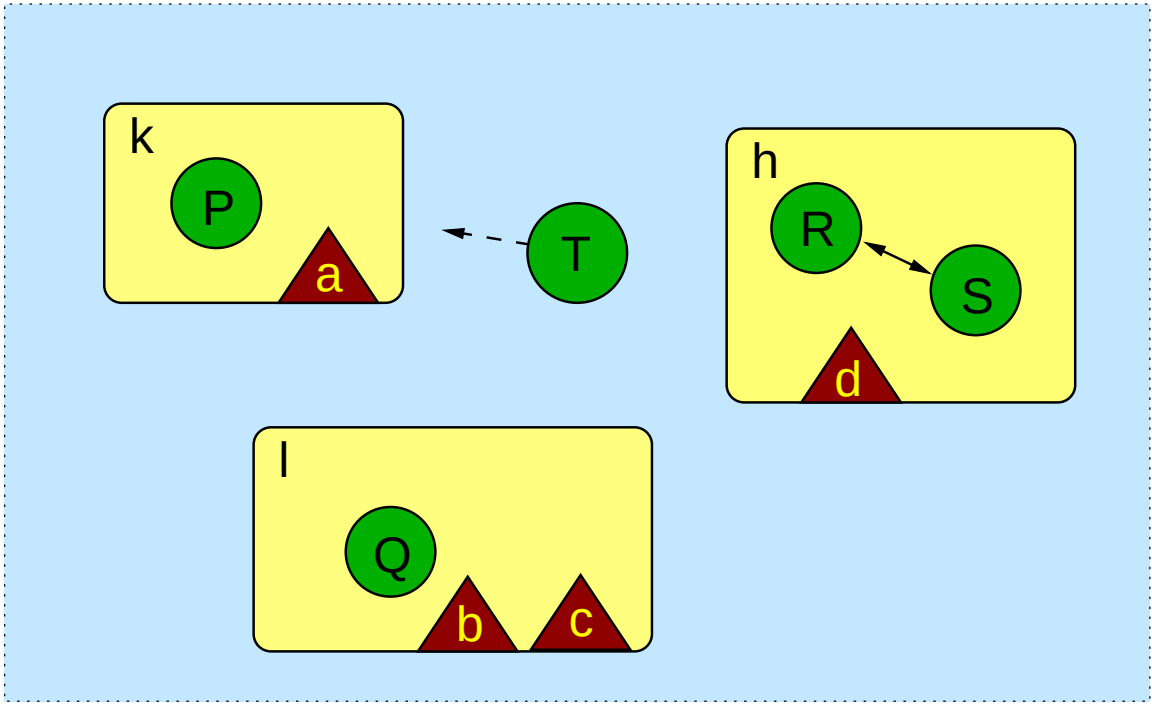
Locations and Border Xing



More precisely... Locations or sites, and border crossing are the basic elements:

- Locations are sites containing processes: $\ell[\![P]\!]$
- Agents travel between locations $\ell[\![\text{goto}\, k\, .P \mid Q]\!] \mid k[\![R]\!] \longrightarrow \ell[\![Q]\!] \mid k[\![P \mid R]\!]$

Locations and Border Xing



More precisely... Locations or sites, and border crossing are the basic elements:

- Locations are sites containing processes: $\ell[\![P]\!]$
- Agents travel between locations $\ell[\![\text{goto}\, k\, .P \mid Q]\!] \mid k[\![R]\!] \longrightarrow \ell[\![Q]\!] \mid k[\![P \mid R]\!]$

Types to control resource access is a current research topic.

- Which resource are available at a given location?
- How do we make sure the are used accordingly?

Types for Access control

A small sample of the existing theories

- Type systems with locations guarantees that the following makes sense.

$$\text{goto}\,\ell\,.\,\text{“use res”}\,.P$$

  It also allows a selective (per-users) usage of resources

- Location types

$$\ell : \mathtt{loc}[\mathscr{R}, \mathscr{S}, \mathscr{D}]$$

  can specify which locations we are prepare to receive and dispatch process from and to.

- Type systems also support time/space quantitive bound analysis. In particular, a system of capacity types guarantees that if

$$\ell : \mathtt{cap}[n, N]$$

  at any time the space allocated inside ℓ will be $n \leq x \leq N$.

The Gap, again...

Yes, but you cannot type the Internet.

All this works as long as you trust the certified types or are willing to typecheck migrating code yourself (bytecode verification, PCC,...)

In UbiComp, security must work be coupled with trust.

This is hard, because of delegation and dynamic policies

Understanding Ubiquitous Delegation

$a:$	$p \mapsto \mathsf{trusted};$	$b:$ $p \mapsto \ulcorner a \urcorner(p);$
	$q \mapsto \ulcorner b \urcorner(q);$	$q \mapsto \mathsf{untrusted};$
	$z \mapsto \ulcorner p \urcorner(z);$	$z \mapsto \ulcorner a \urcorner(z);$

Understanding Ubiquitous Delegation

$a:$	$p \mapsto \text{trusted};$	$b:$	$p \mapsto \ulcorner a \urcorner(p);$
	$q \mapsto \ulcorner b \urcorner(q);$		$q \mapsto \text{untrusted};$
	$z \mapsto \ulcorner p \urcorner(z);$		$z \mapsto \ulcorner a \urcorner(z);$

Delegation, formally: Global trust as a fixpoint.

$$\pi : (\mathcal{P} \to \mathcal{P} \to D) \to (\mathcal{P} \to D) \qquad \text{Local Policy}$$

$$\Xi : (\mathcal{P} \to \mathcal{P} \to D) \to (\mathcal{P} \to \mathcal{P} \to D) \qquad \text{Collected Policies}$$

Global Trust: $\mathit{fix}(\Xi) : \mathcal{P} \to \mathcal{P} \to D.$

Understanding Ubiquitous Delegation

$a:$	$p \mapsto \text{trusted};$	$b:$	$p \mapsto \ulcorner a \urcorner(p);$
	$q \mapsto \ulcorner b \urcorner(q);$		$q \mapsto \text{untrusted};$
	$z \mapsto \ulcorner p \urcorner(z);$		$z \mapsto \ulcorner a \urcorner(z);$

Delegation, formally: Global trust as a fixpoint.

$$\pi : (\mathcal{P} \to \mathcal{P} \to D) \to (\mathcal{P} \to D) \qquad \text{Local Policy}$$

$$\Xi : (\mathcal{P} \to \mathcal{P} \to D) \to (\mathcal{P} \to \mathcal{P} \to D) \qquad \text{Collected Policies}$$

Global Trust: $\mathit{fix}(\Xi) : \mathcal{P} \to \mathcal{P} \to D.$

p : trusted	q : untrusted	z : ???

The integration of Security and Trust opens an important and wide line of research involving: algorithms trust Ubi evaluation/approximation, middleware, models and semantics and analysis techniques, ...

Conclusion

- Theory has a past record being relevant in pratice.
- It appears to be currently pursuing issues very pertinent to UbiComp.
- UbiComp provide real, important challenges.

So, the ground is set for strong cooperation. We should not separate theory from practice at such an early stage of development of such an important application.