FORMAL METHODS FOR PROTOCOL ANALYSIS

Tom Chothia

Introduction

- We have seen that understand protocols security can be tricky.
- This lecture I'll describe some formal methods than can be useful in helping you understand protocols.
- CCS & the pi-calculus.
- Case study: the MUTE anonymous file sharing system

The Lambda Calculus

Term ::= λ var.Term
| (Term)(Term) (λx.M)N → M[N/x]
| var

inc(x) = (λx.x+1)
applyTwice(f) = λf .f f

applyTwice inc 5 = (λf.f f) (λx.x+1) 5
= (λx.x+1) (λx.x+1) 5
= (λx.x+1+1) 5
= 5+1+1 = 7

The Lambda Calculus

Term ::= λ var.Term
| (Term)(Term) (λx.M)N → M[N/x]
| var

T = λxy.x
F = λxy.y
∧ = λxy.xyF
∨ ≡ λxy.x(λuv.u)y ≡ λxy.xTy
¬ ≡ λx.x(λuv.v)(λab.a) ≡ λx.xFT

∧ T F = (λxy.xyF) (λxy.x) (λxy.y)
= (λxy.x) (λxy.y) F
= (λxy.y) = F

IMP: model for imperative languages

```
C = X:=a
    skip
    if b then C1 else C2
    while b { C }
    C1;C2
```

IMP: Imperative Languages

$$\frac{(\!|a, \sigma|\!) \twoheadrightarrow v}{\langle X := a, \sigma\rangle \twoheadrightarrow \sigma\{X \mapsto v\}} \quad \text{(A)}$$

$$\frac{}{\langle \texttt{skip}, \sigma\rangle \twoheadrightarrow \sigma} \quad \text{(B)}$$

$$\frac{\langle c_1, \sigma\rangle \twoheadrightarrow \sigma' \quad \langle c_2, \sigma'\rangle \twoheadrightarrow \sigma''}{\langle c_1; c_2, \sigma\rangle \twoheadrightarrow \sigma''} \quad \text{(C)}$$

$$\frac{(\!|b, \sigma|\!) \twoheadrightarrow \texttt{true} \quad \langle c_1, \sigma\rangle \twoheadrightarrow \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma\rangle \twoheadrightarrow \sigma'} \quad \text{(D)}$$

$$\frac{(\!|b, \sigma|\!) \twoheadrightarrow \texttt{false} \quad \langle c_2, \sigma\rangle \twoheadrightarrow \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma\rangle \twoheadrightarrow \sigma'} \quad \text{(E)}$$

$$\frac{(\!|b, \sigma|\!) \twoheadrightarrow \texttt{false}}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \twoheadrightarrow \sigma} \quad \text{(F)}$$

$$\frac{(\!|b, \sigma|\!) \twoheadrightarrow \texttt{true} \quad \langle c, \sigma\rangle \twoheadrightarrow \sigma' \quad \langle \texttt{while } b \texttt{ do } c, \sigma'\rangle \twoheadrightarrow \sigma''}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \twoheadrightarrow \sigma''} \quad \text{(G)}$$

A Calculus for Communicating Systems?

A Calculus for Communicating Systems (CCS)

```
P,Q ::= 0                P,Q  ::= 0
    | in a (b).P             |  a(b).P
    | out a <b>.P            |  a<b>.P
    | !P                     |  !P
    | new a;P                |  ν a;P
    | ( P | Q )              |  (P | Q)
    | P + Q                  |   P + Q
    | if (cond)              |  [x = y]P
          then  P
```

Semantics

$$\text{out a} <b> \xrightarrow{\bar{a}<b>} 0 \qquad \text{in a } (x);P \xrightarrow{a(x)} P$$

$$\frac{P \xrightarrow{\bar{a}<b>} P` \qquad Q \xrightarrow{a(x)} Q`}{P \mid Q \xrightarrow{\tau} P` \mid Q`[b/x]}$$

$$\frac{P \xrightarrow{\alpha} P`}{P \mid Q \xrightarrow{\alpha} P` \mid Q} \qquad \frac{Q \xrightarrow{\alpha} Q`}{P \mid Q \xrightarrow{\alpha} P \mid Q`}$$

$$!P \rightarrow P \mid !P \qquad \frac{P \xrightarrow{\alpha} P` \quad a \notin \alpha}{\text{new a}.P \xrightarrow{\alpha} \text{new a}.P`}$$

$$\frac{P \equiv Q \qquad Q \rightarrow Q'}{P \rightarrow P} \qquad \frac{P \rightarrow P \,..\, P \rightarrow P \,\ldots\, P \rightarrow P}{P \rightarrow P}$$

Pi-calculus

- The pi-calculus extends CCS by letting channel names be rewritten.

Client:	$\nu c\,.\,\overline{\text{server}}\langle\text{finger}, c\rangle$	← ask server
	$\mid c(x).\overline{\text{print}}\langle x\rangle$	← print response
Server:	$\text{server}(\text{service}, \text{reply}).\overline{\text{service}}\langle\text{reply}\rangle$	← inet daemon
	$\mid \text{finger}(\text{reply}).\overline{\text{reply}}\langle\text{users}\rangle$	← finger daemon
	$\mid \text{time}(\text{reply}).\overline{\text{reply}}\langle\text{now}\rangle$	← time daemon

- Most models don't use this, and are really CCS!

Weak bi-simulation

- Two systems are weakly bi-similar if:
  - every input and output of one systems is matched with the same input or output and any number of internal transitions in the other system
  - and the resulting processes are also bi-similar.
- Weak bi-simulation defines the observation power of the attacker.
- Using bi-simulation, rather than trace equivalence gives the attacker the power to detected states where a possible action has been ruled out.

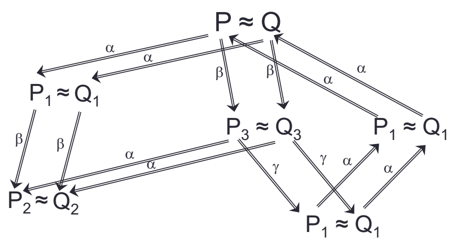
Weak bi-simulation

- A definition of looks the same:

Definition 1 (Weakly bi-similar). *Processes P and Q are weakly bi-similar if there exists an equivalence relation $\approx$ such that $P \approx Q$ and for all P_1 and Q_1 such that $P_1 \approx Q_1$, if $P_1 \stackrel{\alpha}{\Rightarrow} P_1'$ then:*

- *if α is an output or an internal action there exists a process Q_1' such that $Q_1 \stackrel{\alpha}{\Rightarrow} Q_1'$ and $P_1' \approx Q_1'$.*
- *if α is an input action, i.e., $\alpha = a(x)$, then for all names b, there exists a process Q_1' such that $Q_1 \stackrel{\alpha}{\Rightarrow} Q_1'$ and $P_1'[b/x] \approx Q_1'[b/x]$.*

A bi-simulation relation



Weak bi-simulation

- A definition of looks the same:

Definition 1 (Weakly bi-similar). *Processes P and Q are weakly bi-similar if there exists an equivalence relation $\approx$ such that $P \approx Q$ and for all P_1 and Q_1 such that $P_1 \approx Q_1$, if $P_1 \stackrel{\alpha}{\Rightarrow} P_1'$ then:*

- *if α is an output or an internal action there exists a process Q_1' such that $Q_1 \stackrel{\alpha}{\Rightarrow} Q_1'$ and $P_1' \approx Q_1'$.*
- *if α is an input action, i.e., $\alpha = a(x)$, then for all names b, there exists a process Q_1' such that $Q_1 \stackrel{\alpha}{\Rightarrow} Q_1'$ and $P_1'[b/x] \approx Q_1'[b/x]$.*

- E.g., Spec ≈ Implementation
- If Protocol[secret1/x] ≈ Protocol[secret2/x] then the protocol keeps x secret.

Case study: Anonymous file sharing

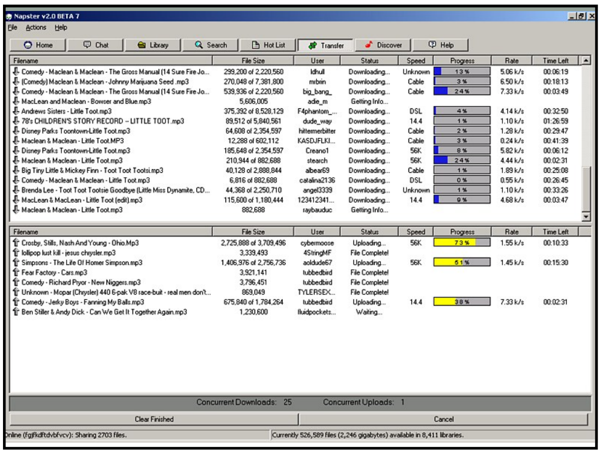
Analysing the MUTE Anonymous File-Sharing System Using the Pi-calculus

Tom Chothia

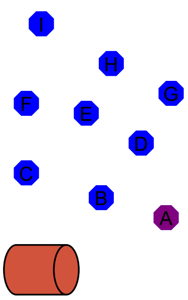
CWI, Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands.

Abstract. This paper gives details of a formal analysis of the MUTE system for anonymous file-sharing. We build pi-calculus models of a node that is innocent of sharing files, a node that is guilty of file-sharing and of the network environment. We then test to see if an attacker can distinguish between a connection to a guilty node and a connection to an innocent node. A *weak bi-simulation* between every guilty network and an innocent network would be required to show possible innocence. We find that such a bi-simulation cannot exist. The point at which the bi-simulation fails leads directly to a previously undiscovered attack on MUTE. We describe a fix for the MUTE system that involves using authentication keys as the nodes' pseudo identities and give details of its addition to the MUTE system.

1 Introduction



"Old" Peer-to-Peer File-Sharing



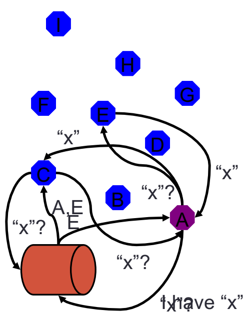
Peers "A" to "I",

"A" would like to get file "x"

First generation networks used a central database.

Peers told this database what files they have.

"Old" Peer-to-Peer File-Sharing



Peer "A" asks the database for "x".

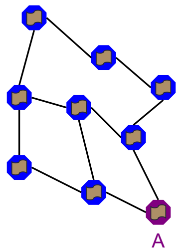
The database tells "A" which peer has the file.

"A" downloads the file from that peer.

"A" tells the database it has the file.

Other peers can then download the file from "A"

Peer-to-Peer File-Sharing

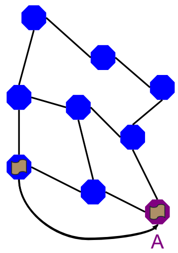


In newer networks peers record the IP address of other peers.

A searcher sends a request to all of it's "neighbours".

This is forwarded to all of there neighbours, up to a fixed hops.

Peer-to-Peer File-Sharing

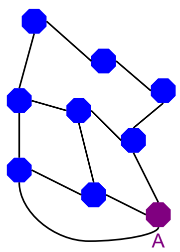


The search request includes A's IP address.

Any peer with the requested file contacts A directly.

Peer "A" may then request the file.

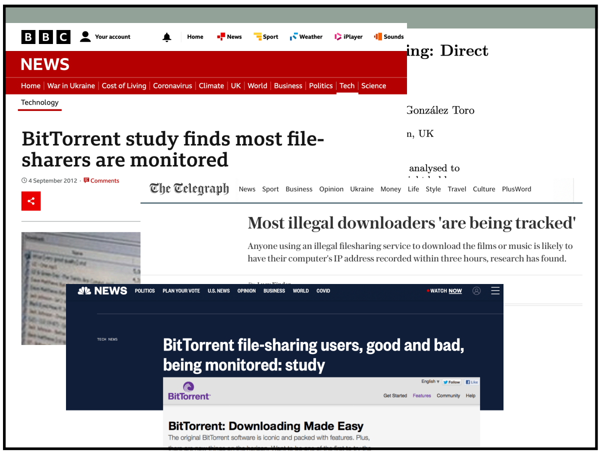
Peer-to-Peer File-Sharing

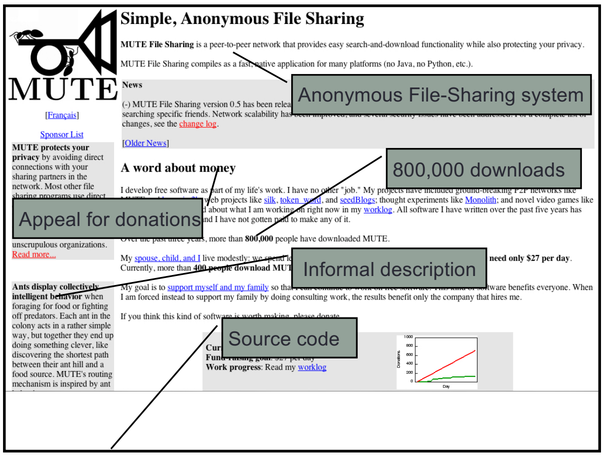


No anonymity from peers inside the network:

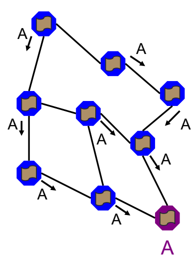
The search message gives the searcher's IP address and name of the files they are looking for.

By requesting a file, you can find out the IP address of all peers that are offering the file.





MUTE: Search

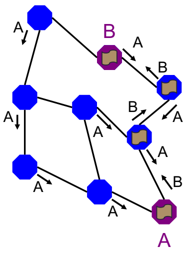


The search takes place as before, but this time the message uses its pseudo ID as the "from ID".

Each peer builds a routing table by records the ID and the connection.

A probabilistic time-to-live counter limits the search.

MUTE: Reply



If B wants to reply it sends a message to A's pseudo ID.

This message is routed using the ad-hoc routing table.

The route to B is also recorded

Anonymity Provided by MUTE

It should be impossible for an attacker who
- is acting as any number of peers,
- and can observe any number of connections,
- but does not know what the network looks like,
  i.e., cannot be sure that they have surrounded a peer

to be able to link any IP address and pseudo ID with an arbitrarily high probability.

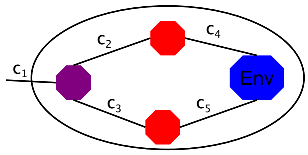
The Pi-calculus Model of MUTE

- We make pi-calculus models of
  - an "innocent" peer: I, that only forward messages.
  - A "guilty" peer : G, that forwards messages and replies to requests for a file.
  - And an environment: E,
- These processes are parameterised on the communication channels that run between them...
- and the choices the peer makes at start up.

The Pi-calculus Model of MUTE

- We can use these processes as building blocks to make any network.

e.g. $I(c_1,c_2,c_3) \mid G(c_2,c_4) \mid G(c_3,c_5) \mid E(c_4,c_5)$
e.g. new c_2,c_3,c_4,c_5; $I(c_1,c_2,c_3) \mid G(c_2,c_4) \mid G(c_3,c_5) \mid E(c_4,c_5)$



Model of an Innocent Peer.

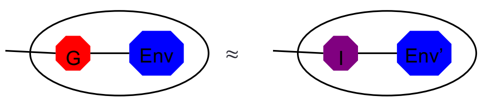
```
InnocentPeer( (ci1,co1), (ci2,co2), for, ttl )
= new my_id,
   !(in ci1 (kind,to_id,from_id,phase,counter);
     out <memory, (from_id,co1)>;
         | if [kind = search] then Forward
         | if [kind = reply]  then Reply
         ... }
   | ! (out<co1 ,(search,_,my_id,1,ttl) }

Reply = if (to_id = my_id) then {stop}
      | if (to_id ≠ my_id) then
           new loop; out loop;
          !( in loop;
             in memory(x, channel);
             out memory(x, channel);
             if (x ≠ to_id) then out loop;
              | if (x = to_id) then
                out channel(kind, to id, from id , phase, counter)
```

Correctness

We want to show that the environment provides cover for a guilty peer:

i.e., for all guilty peers G and environments Env there exists an innocent peer I and environment Env' such that
$G \mid E \approx I \mid E'$



Checking the Model

We check that:

new c_2.(Guilty_Peer(c_1,c_2,p_g) | Env (c_2,e_1))
$\approx$ new c_2.(Innocent_Peer(c_1,c_2,p_i) | Env (c_2,e_2))

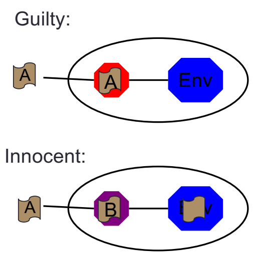
This is small enough to do by hand
~ 24 states to check.

Failure of Bisimulation

A state that cannot be matched:
- receive a search message from the attacker,
- sent a reply from the guilty peer / the environment,
- receive a well-formed search message from the guilty peer's ID,
- receive a reply message "from" the guilty peers ID.

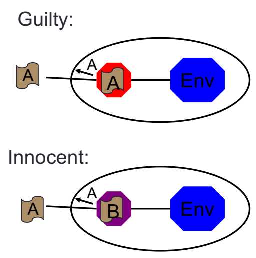
- Then the innocent network can perform .
  - a receive the same reply back
- The guilty network cannot perform this action.

What Happened?



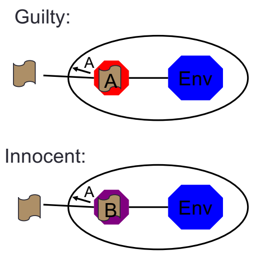
- The guilty network shares its file and makes its ID public.
- The innocent network matches this with a file from the environment.
- The attacker makes a search message using this ID as the **from** address and sends it back.

What Happened?



- A record of that ID and the connection is added to peers routing table.
- This is an "incorrect" entry in the routing table.
- It routes messages away from their real owner and to the attacker.

What Happened?



- Now the attacker sends a reply message **to** that ID.
- The innocent peer may send it back.
- Whereas, the guilty peer will always accept the message.

Description of the Real Attack

- The attacker tries to steal an address by sending many fake message.
- The attacker then sends 50+ reply messages, use the target ID.
- If the reply is not returned to the attacker then there is a high probability that it belongs to the neighbour.

Fixing MUTE

- The final solution was for peers to generate RSA signature keys “kS” and an authentication keys “kA”.
- Message header has the form:

  (#(kA), from ID, message ID-time_stamp,
  FLAGS:(S_{kS}(messageID-time_stamp), kA))

- This fix was added to the latest version of MUTE.

Conclusion

- Modelling protocols in process calculi is a great way to understand them
- We can test anonymity using bisimulation:

  For all p_g,e_1 there exists p_i,e_2 s. t.
  $\text{Guilty\_Peer}(p_g) \mid \text{Env}(e_1) \approx \text{Innocent\_Peer}(p_i) \mid \text{Env}(e_2)$

- 500,000 people downloaded and used the fixed version (before they all gave up and used BitTorrent instead).

Summary

- A formal language lets us model and analyse complex protocols.
- This is an excellent tool for protocol analysis.
- If you ever analyse a protocol, or a similar complex system give this a try.
- BUT, this is all ancient history. Today we have excellent fully automatic tools to do the analysis for us.
- These tools will be the subject of tomorrows lecture.