Protecting resources in an open and trusted peer-to-peer network

J.-F. Lalande, D. Rodriguez

Ensi de Bourges, France

METHOD 2012 - July 16, 2012

Context

Peer-to-peer technologies are widely used:

- Open source software (e.g. linux distributions)
- Commercial software
  - e.g. Skype
- Private networks (encrypted tunnels, authenticated users)

Not so much used for:

- Content delivery
- Business exchanges

Issues ?

Main issues with peer-to-peer technologies

- Contradictory with copyright laws
- The distribution process is uncontrolled
- The security guarantees are mainly for users

Issues ?

Main issues with peer-to-peer technologies

- Contradictory with copyright laws
- The distribution process is uncontrolled
- The security guarantees are mainly for users

The protocols mainly focus on safety:

- Anonymity of users (GAP, Freenet) [6, 3, 1]
- Survivability/Availability of resources [4]
- Access control ?
  - ECRS [2] $\rightarrow$ sort of confidentiality and integrity by obfuscating and checking the content that is exchanged
- Protection of resources ?
- Expressing security properties for resources ?

Our goal: solve this conflict:

Express and enforce security properties

and

Keep the peer-to-peer network open

Open ?

- Keep the exchange protocol open
- Keep the client source code open and free
- Let the user define the policies

Security properties ?

- What can be expressed ?
- How to enforce them ?

What we do not want...

Change the peer-to-peer protocol:

- Authenticate users
- Use cryptology mechanisms to protect data

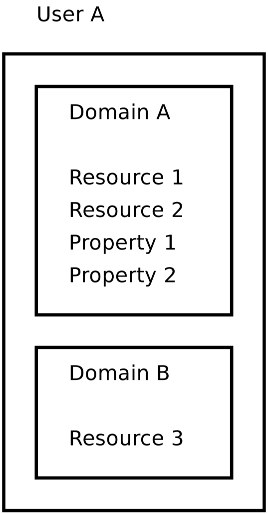
Change the peer-to-peer software:

- Use a closed source peer-to-peer client
- Rely on a trusted OS

Change the nature of the peer-to-peer network:

- Centralize the security checks
- Control the security policies of peers

Notion of domains



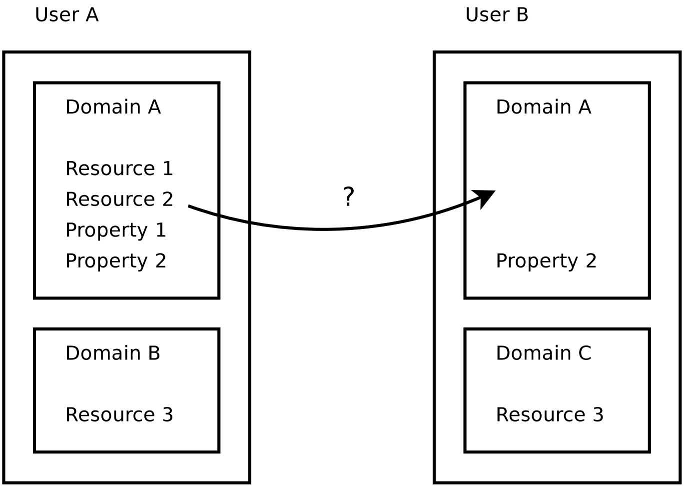
A domain is:

- a named group of resources
- associated to a set of security properties

The user is in charge of:

- create domains
- define the policy

Exchanges between domains



Protecting domains

- **integrity(sensitive_data_domain)**: the resources of the sensitive_data_domain domain must not be modified.

Protecting domains

- **integrity(sensitive_data_domain)**: the resources of the sensitive_data_domain domain must not be modified.
- **confidentiality(secret_domain)**: the resources of the secret_domain domain must stay in this domain.

Protecting domains

- **integrity(sensitive_data_domain)**: the resources of the sensitive_data_domain domain must not be modified.
- **confidentiality(secret_domain)**: the resources of the secret_domain domain must stay in this domain.
- **spread(diffusion_domain)**: the resources of the diffusion_domain domain must be available as much as possible for all peers and can freely change of domain.

Protecting domains

- **integrity(sensitive_data_domain)**: the resources of the sensitive_data_domain domain must not be modified.
- **confidentiality(secret_domain)**: the resources of the secret_domain domain must stay in this domain.
- **spread(diffusion_domain)**: the resources of the diffusion_domain domain must be available as much as possible for all peers and can freely change of domain.
- **nopublication(fee_paying)**: no new resources can be added in the fee_paying domain.

Protecting domains

- **integrity(sensitive_data_domain)**: the resources of the sensitive_data_domain domain must not be modified.
- **confidentiality(secret_domain)**: the resources of the secret_domain domain must stay in this domain.
- **spread(diffusion_domain)**: the resources of the diffusion_domain domain must be available as much as possible for all peers and can freely change of domain.
- **nopublication(fee_paying)**: no new resources can be added in the fee_paying domain.
- **noshare(confined_domain)**: the files of the confined_domain should not be shared with another peer.

Protecting domains

- **integrity(sensitive_data_domain)**: the resources of the sensitive_data_domain domain must not be modified.
- **confidentiality(secret_domain)**: the resources of the secret_domain domain must stay in this domain.
- **spread(diffusion_domain)**: the resources of the diffusion_domain domain must be available as much as possible for all peers and can freely change of domain.
- **nopublication(fee_paying)**: no new resources can be added in the fee_paying domain.
- **noshare(confined_domain)**: the files of the confined_domain should not be shared with another peer.
- **cooperation(priv_A, priv_B)**: the peer should help the exchange of resources between priv_A and priv_B.

Conflicting properties

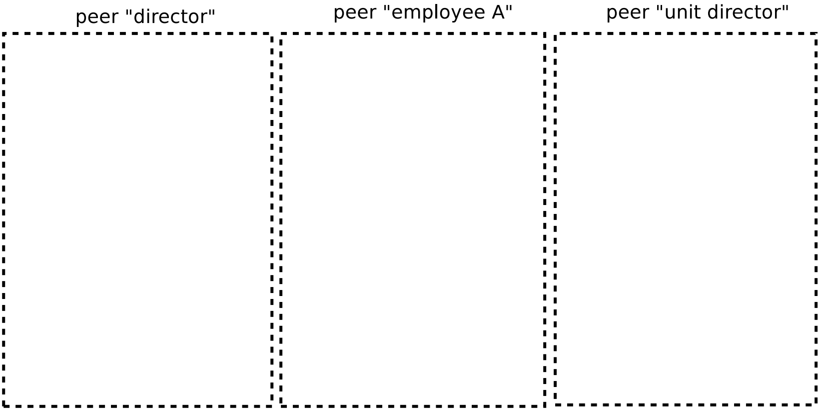
Conflicts	Conf.	Integ.	Spread	!Pub	!Share	Coop.
Conf.			x			x
Integ.						
Spread	x				x	
!Pub						
!Share			x			x
Coop.	x				x	

Conflicting properties

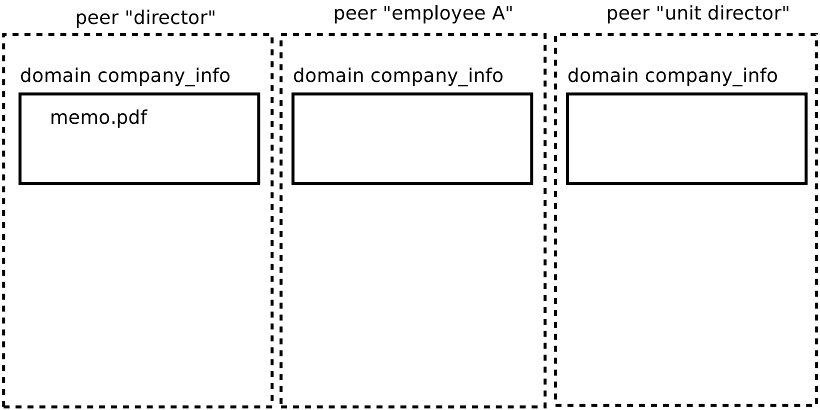
For example:

- confidentiality conflicts with spread

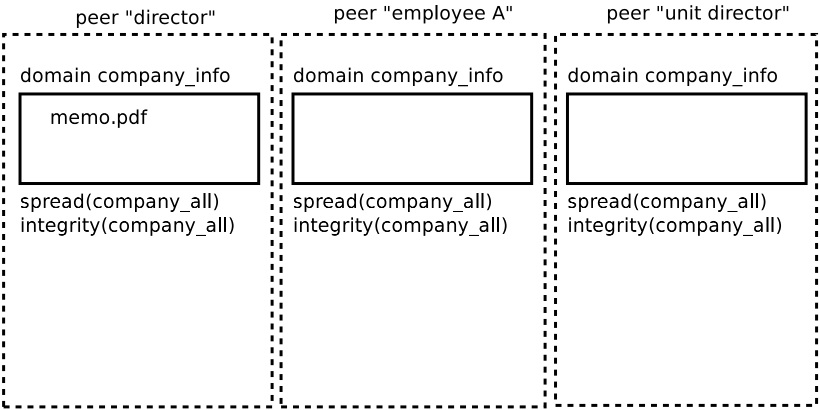
Example of scenario



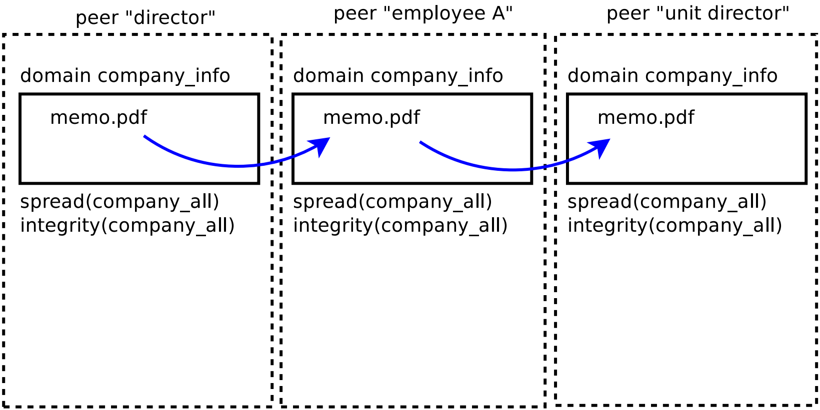
Example of scenario



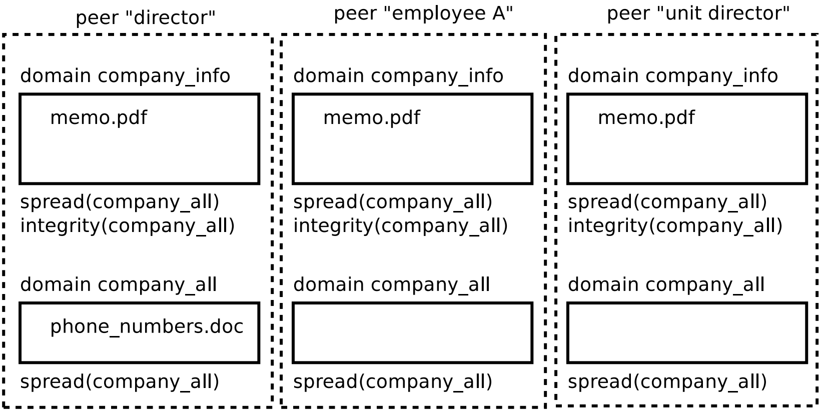
Example of scenario



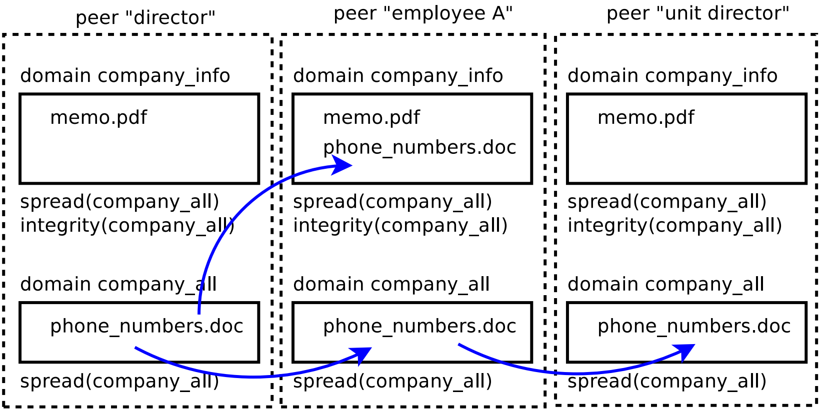
Example of scenario



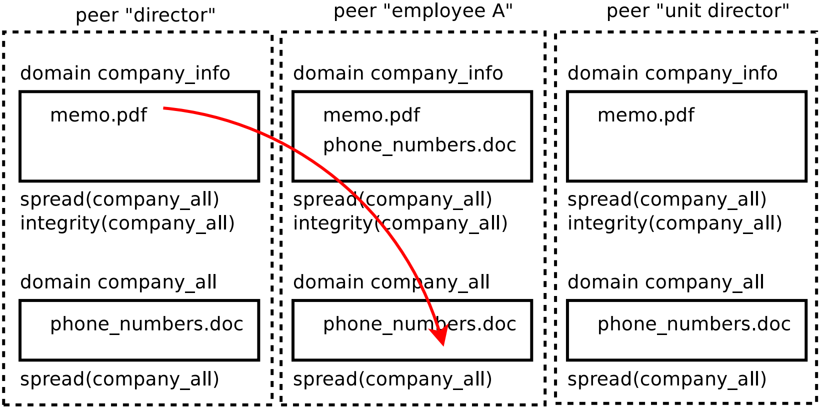
Example of scenario



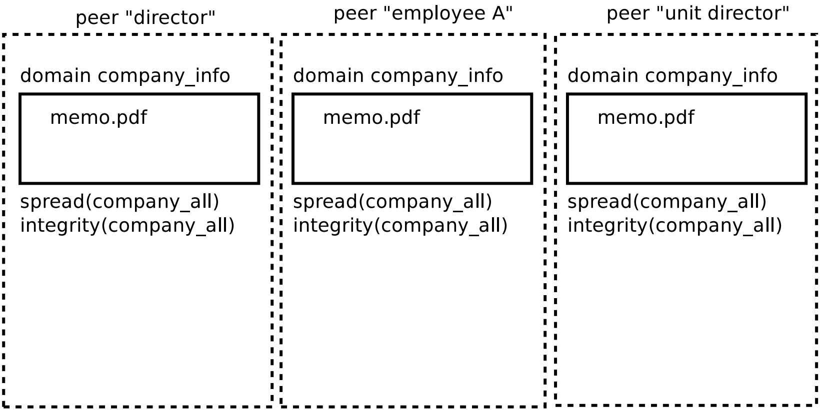
Example of scenario



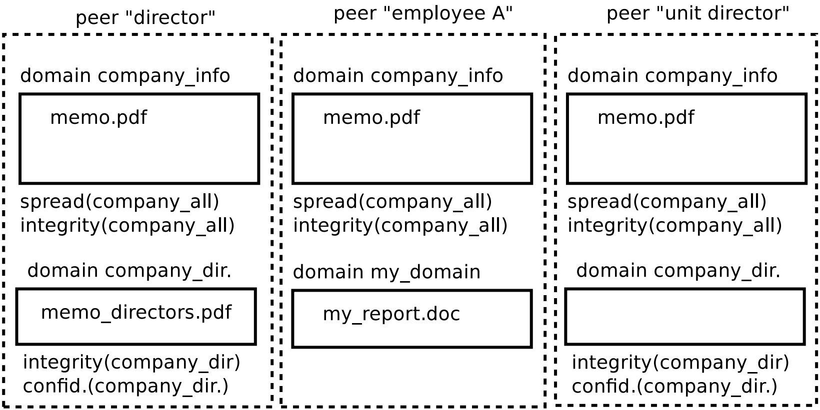
Example of scenario



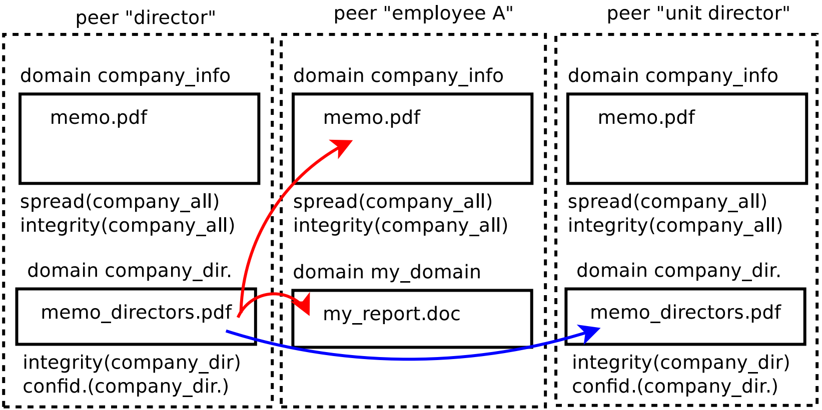
Example of scenario



Example of scenario



Example of scenario

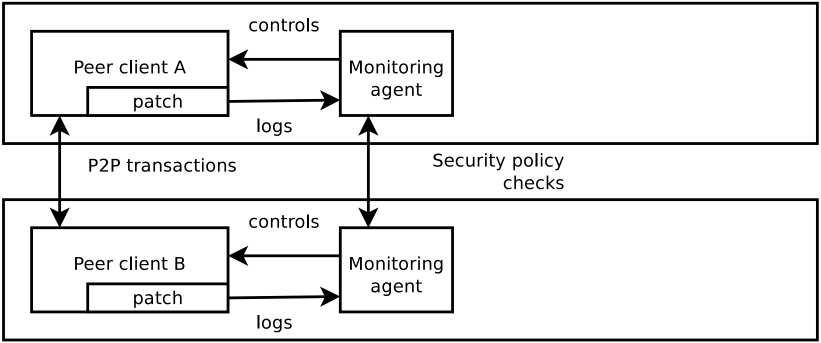


Monitoring agent I

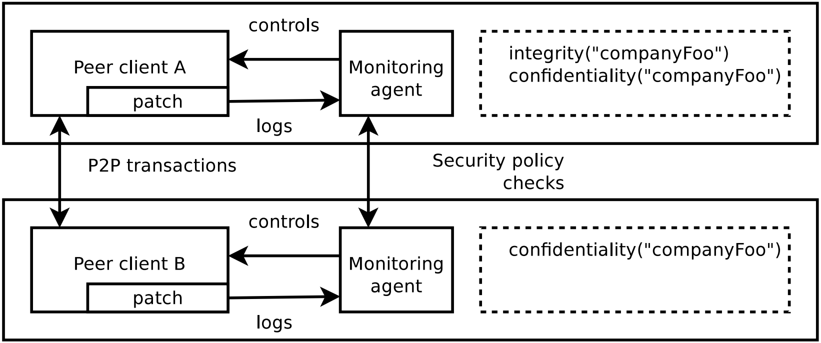
The security mechanisms are delegated to a Monitoring Agent:

- Manage the policies
- Checks policies when resources are exchanged
- Negotiate policies of domains when an exchange occuurs
- Computes the trust of other peers
- Enforces policies locally
- Controls the peer-to-peer client

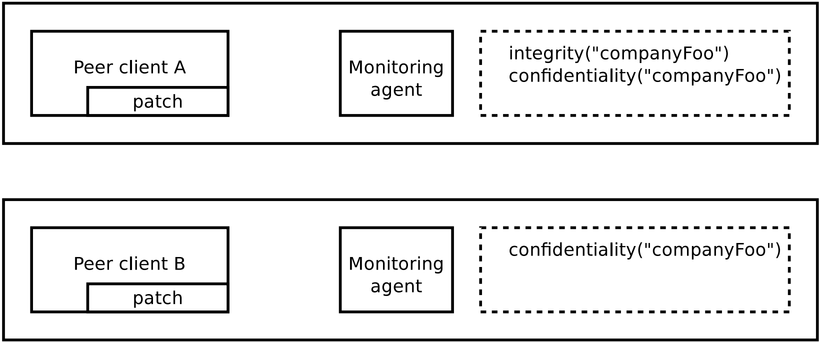
Monitoring agent II



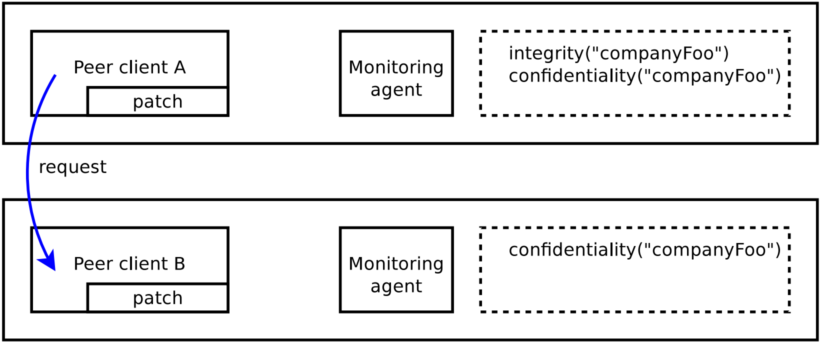
Monitoring agent III



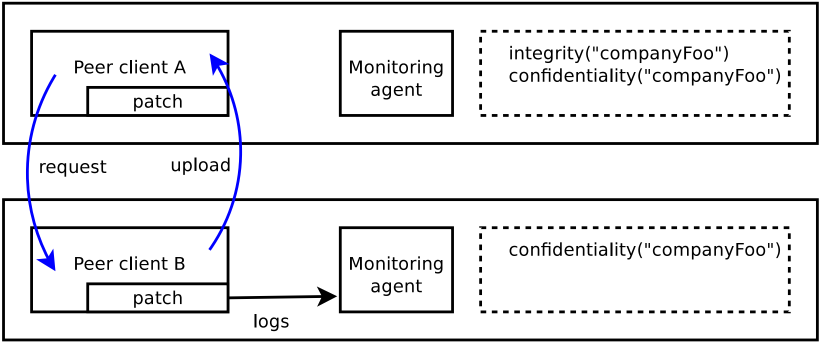
An exchange, step by step



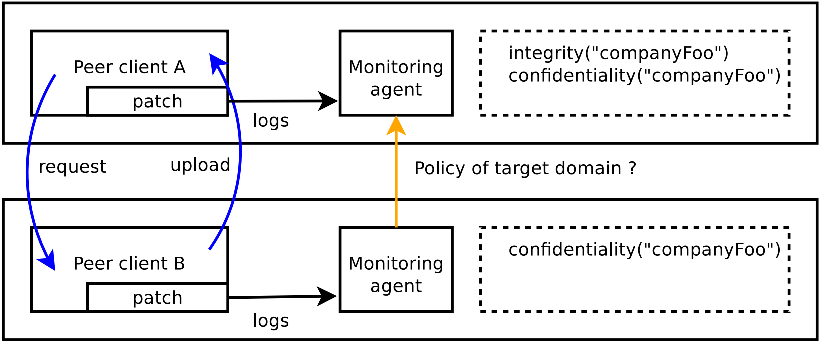
An exchange, step by step



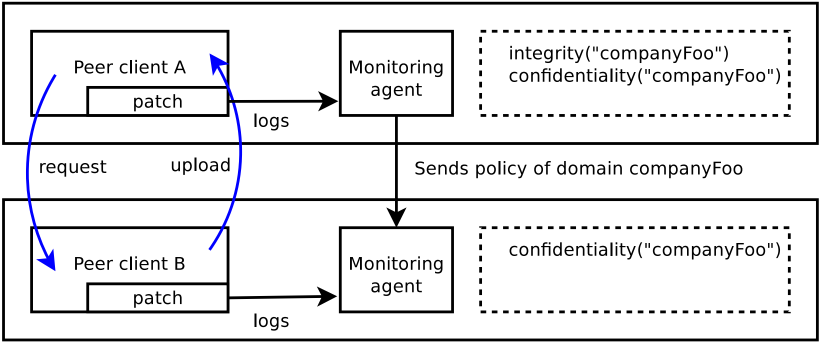
An exchange, step by step



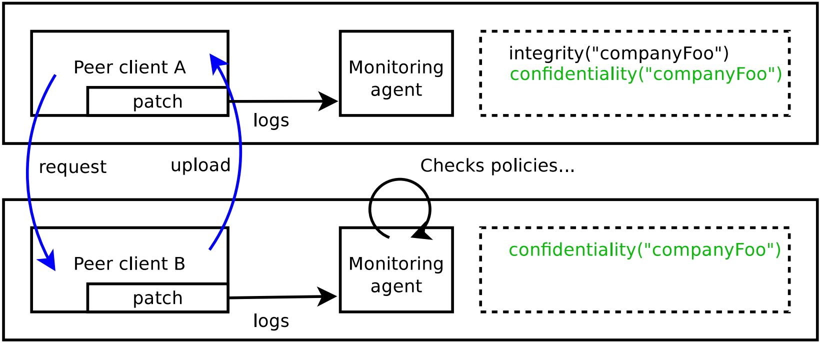
An exchange, step by step



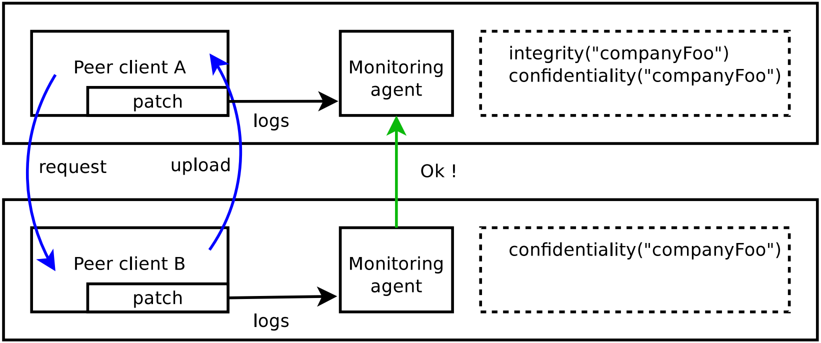
An exchange, step by step



An exchange, step by step



An exchange, step by step



Policy checks

Policy checks that should deny a request:

- target policy (peer A) is inconsistent:
  - confidentiality(companyFoo), spread(companyFoo)
- conflicts between target policy and source policy:
  - source (B): confidentiality(companyFoo)
  - target (A): spread(companyFoo)

If some checks fails:

- the peer-to-peer client download is stopped
- or the peer-to-peer client is killed

Advantages

For the implementation:

- a small modification of the peer-to-peer is needed
- any open source peer-to-peer client can be supported

For the peer-to-peer network:

- a peer A can participate without the monitoring agent
  - peer B will only upload for domain without properties
- policies are outside the peer-to-peer client
- policies can evolve to reflect new needs

Malicious peers

Peer A can be supposed to be a malicious node:

- What happens if A tries to guess source policy ?
- What happens if A anounces a fake policy ?
- Is there any security enforcement in A ?

For example, case 1:

- peer A knows that a file memo_directors.pdf exists
- peer A floods the peer-to-peer networks of requests
- For each request:
  - he tries a new domain name (to guess it)
  - he tries a new security policy (to be compatible)

⇒ evaluate the **trust** to put in a peer

Malicious peers

Peer A can be supposed to be a malicious node:

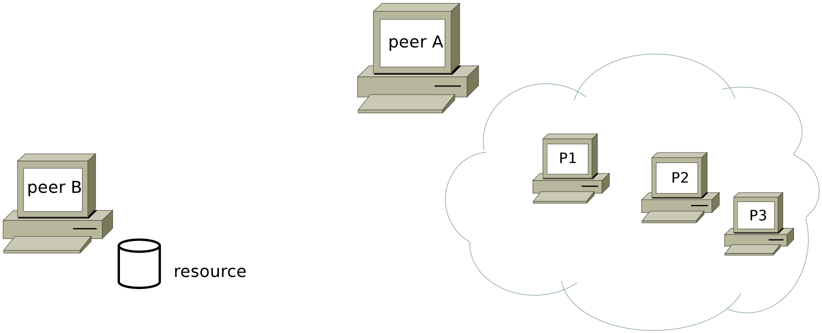
- What happens if A tries to guess source policy ?
- What happens if A anounces a fake policy ?
- Is there any security enforcement in A ?

For example, case 2:

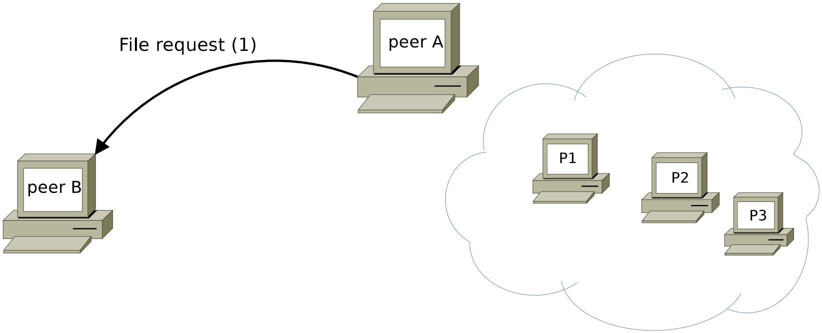
- peer A anounces the policy "confidentiality(company_directors)"
- peer A uploads files from company_directors for any request

⇒ evaluate the **trust** to put in a peer

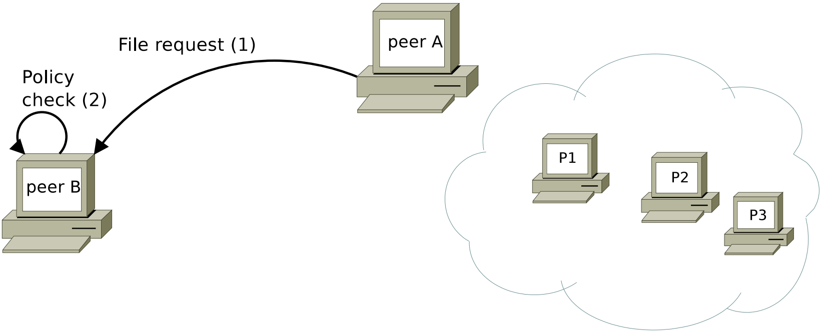
Trust



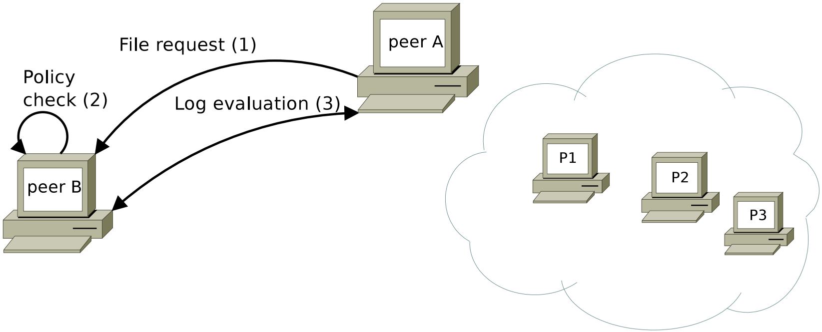
Trust



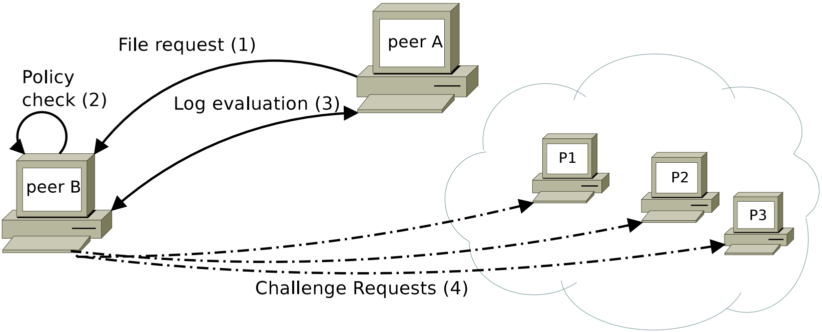
Trust



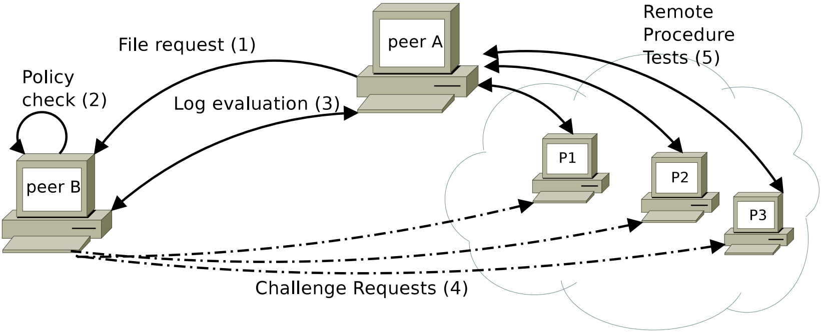
Trust


peer A
File request (1)
Policy
check (2)
Log evaluation (3)
peer B
P1
P2
P3

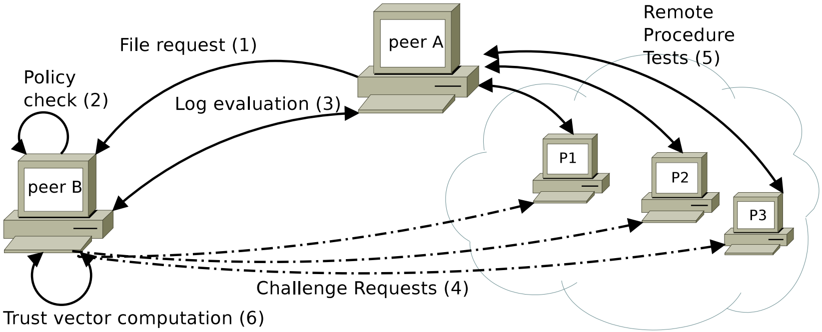
Trust


File request (1)
peer A
Policy check (2)
Log evaluation (3)
peer B
P1
P2
P3
Challenge Requests (4)

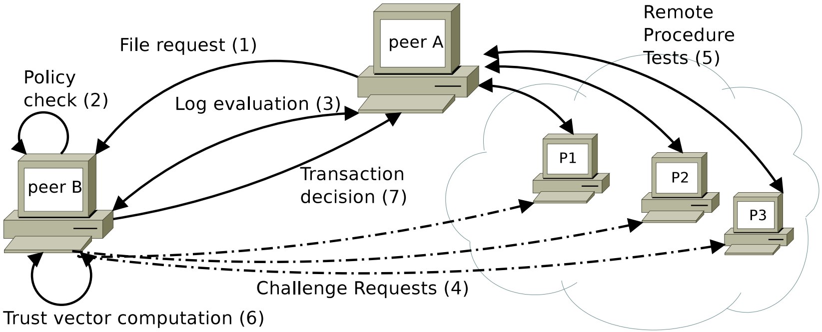
Trust


File request (1)
peer A
Policy
check (2)
Log evaluation (3)
peer B
Remote
Procedure
Tests (5)
P1
P2
P3
Challenge Requests (4)

Trust


peer A
peer B
P1
P2
P3
File request (1)
Policy check (2)
Log evaluation (3)
Challenge Requests (4)
Remote Procedure Tests (5)
Trust vector computation (6)

Trust


File request (1)
Policy check (2)
Log evaluation (3)
Challenge Requests (4)
Remote Procedure Tests (5)
Trust vector computation (6)
Transaction decision (7)
peer A
peer B
P1
P2
P3

Trust

The trust evaluation of A is a combination of:

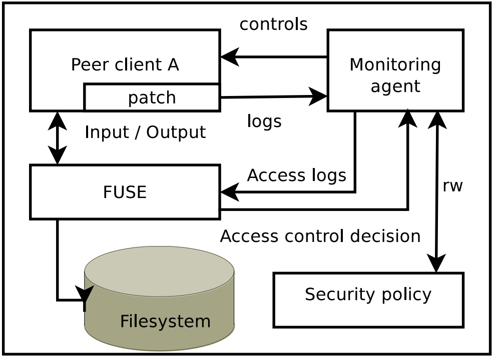
- the policy checks
- the reputation of A
- the evaluation of logs of A
- the evaluation of challenges sent to A

$\Rightarrow$ evaluates the trust B can put in A

Prototype

Local enforcement of policies: FUSE module

- is configured by the monitoring agent
- protects resources from other processes
- informs the monitoring agent of accesses



Simulation hypothesis

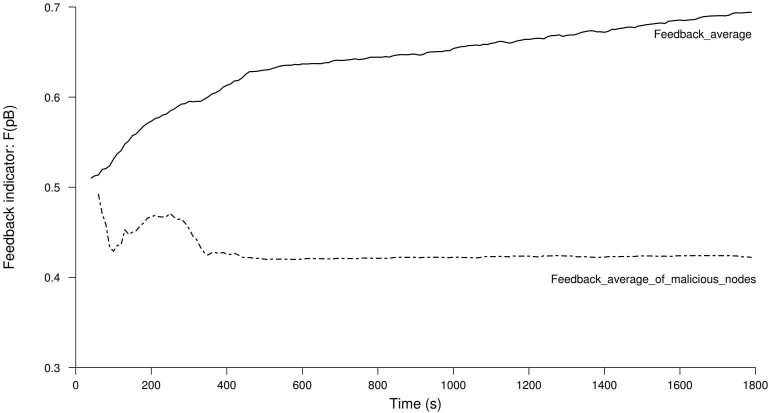
Simulation with 100 peers:

- Discrete event simulator for peer-to-peer protocols [5]
- At each update each peer has:
  - 5% of chance to a add a new file
  - 1% of chance to delete a file
  - 30% of chance to download a file choosen randomly
- 95% regular peers, 5% of malicious peers

For policies, history of transactions:

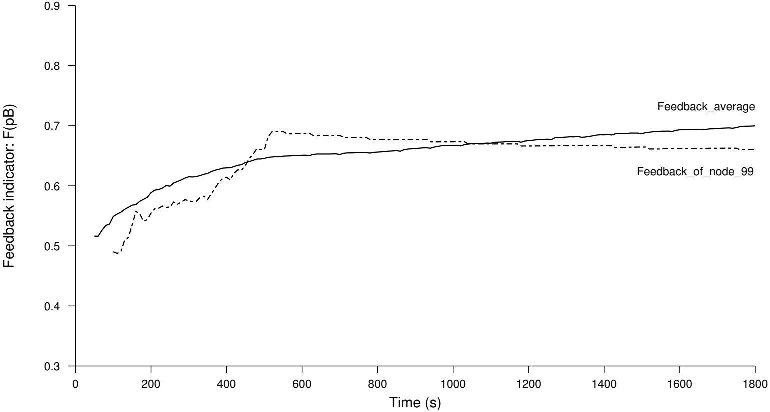
- static random consistent policies
- evaluation of history of transactions:
  - considered good for regular peers
  - considered bad for malicious peers

Simulation results I



Evolution of trust for regular and malicious peers

Simulation results II



Peer 99 becomes malicious after 500s of simulation

Conclusion and perspectives I

Security properties associated to domains

- managed by a monitoring agent
- compatible with open peer-to-peer clients and protocols
- defined by the user (can evolve)
- enforced (eventually) locally
- enforced by evaluating trust of peers

Difficulties for evaluating simulations:

- difficult to automatically simulate users
  - how to simulate domains ?
  - how to simulate download requests ?
  - how to simulate policy evolving ?

Conclusion and perspectives II

Our other works related to this one

- open distributed crisis management tool
  - e.g. ensure confidentiality of some information
- security properties for cloud computing resources
- self protection of Android applications

All these systems have open frameworks !

- Users need security guarantees
- The system/network cannot be trusted or modified

How to bring more security to these systems ?

Questions

Questions ?

References I

Tom Chothia.

Analysing the mute anonymous file-sharing system using the pi-calculus.

In Elie Najm,
Jean-François Pradat-Peyre, and Véronique Viguié Donzeau-Gouge, editors,
26th IFIP WG 6.1 international conference on formal techniques for networked an
number 4229 in Lecture Notes in Computer Science, pages 115–130.
Springer, September 2006.

Grothoff Christian, Grotoff Krista, Tzvetan Horozov, and Jussi T. Lindgren.

An encoding for censorhip-resistant sharing-ecrs.

Technical report, University of Purdue (USA), University of Denver (USA), University of Helsinky (FINLAND), 2003.

References II

Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley.
Protecting free expression online with freenet.
IEEE Internet Computing, 6(1):40–49, 2002.

Theodore W. Hong and Ian Clarke.
The persistence of memory in freenet, 2004.

Aleksandra Kovačević, Sebastian Kaune, Nicolas Liebau, Ralf Steinmetz, and Patrick Mukherjee.
Benchmarking Platform for Peer-to-Peer Systems.
it - Information Technology, 49(5):312–319, 2007.

Bennett Krista and Grothoff Christian.
Gap - pratical anonymous networking.
Technical report, Departement of Computer Sciences, University of Purdue (USA), 2002.