Unified Platform for Secure Networked Information Systems

Duke CPS296.1 Spring10

Xuanran Zong

Background & Motivation

- Underlying network
  - Accountability
  - Efficient packet tracing
  - Flow analysis
- No integration with networked information system
  - Focus on specific threat
  - Different environment

Objective

- Unified Declarative platform
  - Specify
  - Implement
  - Analyze
  - Audit
- Large-scale **secure** information system

Building blocks

- Logic-based trust management system (Binder)
- Declarative networking (NDlog)

} SeNDlog

- Data analyze via provenance

Binder

- Query language based Datalog
- Access control in untrusted network
- Context
- 'Says'
- Example:

```
b1 may-access(P,O,read) :- good(P).
b2 may-access(P,O,read) :-
    bob says may-access(P,O,read).
```

- Why do they choose Binder?

Binder vs NDlog

	Binder	NDlog
Network Assumption	Untrusted	Trusted
Export of derived tuples	No restriction	Restricted (Location specifies)
Evaluation Order	Top-down (Why?)	Bottom-up (Why?)

SeNDlog

- Unification of Binder and NDlog
- Features
  - Rules bind to particular node
  - Example

```
At N, c1,c2,...,cn
    r1 p :- p1,p2,...,pn.
    r2 p1 :- p2,p3,...,pn.
```

SeNDlog

- Communication
  - Explicit control of import and export tuples
  - Import predicate (body): N says p
  - Export predicate (head): N says p@X
  - Why do we need these restriction? Why not just simply use NDlog style?
  - Example

```
At N,
e1 p(X,Y) :- p1(X), p2(Y).
e2 p(X,Y,W) :- Y says p1(X), Z says p2(W), Z!=N.
e3 p(Y,Z)@X :- p1(X), Y says p2(Z).
e4 Z says p(Y)@X :- Z says p(Y), p1(X).
```

SeNDlog

- Honesty Constraint
  - X says p in head => X says p in body
  - Why?
- Extensions
  - Security level: efficiency security tradeoff
  - Is it necessary?

Some SeNDlog Examples

- Authenticated path-vector protocol

```
At Z,
    z1 route(Z,X,P) :- neighbor(Z,X), P=f_initPath(Z,X).

    z2 route(Z,Y,P) :- X says advertise(Y,P),
                       acceptRoute(Z,X,Y).

    z3 advertise(Y,P)@X :- neighbor(Z,X), route(Z,Y,P),
                           carryTraffic(Z,X,Y), P1=f_concat(X,P).
```

- Can also implement BGP, P2P, CDN

Another Example

- Secure Chord DHT

```
At NI,
    ni1 requestCert(NI,K)@CA :- startNetwork(NI),
            publicKey(NI,K), MyCA(NI,CA).
    ni2 nodeID(NI,N) :- CA says nodeIDCert(NI,N,K)
    ni3 CA says nodeIDCert(NI,N,K)@LI :-
            CA says nodeIDCert(NI,N,K), landmark(NI,LI).

At CA,
   ca1 nodeIDCert(NI,N,K)@NI :- NI says requestCert(NI,K),
            S=secret(CA,NI), N=f_generateID(K,S).

At LI,
   li1 acceptJoinRequest(NI) :- CA says nodeIDCert(NI,N,K).
```

One more example

- Secure DHT-based join processing

```
At alice,
    a1 storeA(X,Y)@NI :- tableA(X,Y), K=f_sha(X),
            NI=Chord::K.

At bob,
    b1 storeB(X,Y)@NI :- tableB(X,Y), K=f_sha(X),
            NI=Chord::K.

At NI,
    r1 results(X,Y)@r :- alice says storeA(X,Y),
            bob says storeB(Y,Z).
```

- One more layer of authetication

Secure Query Processing

- Pipelined semi-naïve evaluation (PSN)
  - Asynchronous
- Authenticated PSN
  - Two more operator: SigGenerator, SigChecker
  - Example

```
z2a  Δroute(Z, Y, P) :- X says Δadvertise(Y, P),
                        acceptRoute(Z,X, Y ).
z2b  Δroute(Z, Y, P):- X says advertise(Y, P),
                        ΔacceptRoute(Z,X, Y ).
```

Layering support and security extensions

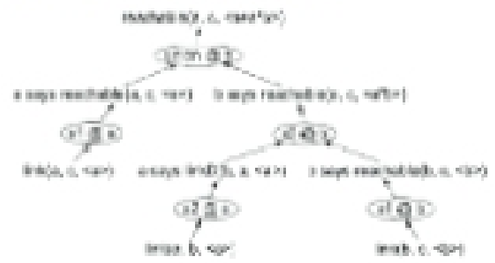
- Extract payload, discard multiple headers
- LocSpecDemux to support local overlay dataflow
- Security extension
  - Optional attribute to SigGenerator and SigChecker

Network Provenance

- Capture the how each predicate is derived
  - Diagnosis, forensics, trust management
- Naturally fit the bottom-up evaluation
- Local vs Distributed
- Online vs Offline

Network Provenance

- Authenticated provenance



Evaluation

- Comparing the performance between auth and non-auth
- Results
  - Authentication introduces latency, especially in PlanetLab
  - Auth + Provenance doubles the completion time
  - Small message transfer tends to have negligible latency overhead (Chord DHT)
  - Bandwidth intensive query tends to have more overhead (best path)

Discussion point

- Untrusted node?
- Query optimization?
- Compilation overhead? Compilation efficiency?