Towards a Declarative Language and System for Secure Networking

by Martin Abadi and Boon Thau Loo

Presented by Spyros Hadjichristodoulou

shadjichrist@cs.sunysb.edu

for CSE-645

State University of New York at Stony Brook
Overview

Introduction

Datalog

Binder and NDlog

SeNDlog Examples

Conclusion and Future Work
Secure Network Protocols design error prone due to imperative programming language usage
Secure Network Protocols design error prone due to imperative programming language usage

Declarative languages to the rescue
Secure Network Protocols design error prone due to imperative programming language usage

Declarative languages to the rescue

SeNDlog = Binder + NDlog
Intro

- Secure Network Protocols design error prone due to imperative programming language usage
- Declarative languages to the resque
- SeNDlog = Binder + NDlog
- Access Control, formulation of protocols
Secure Network Protocols design error prone due to imperative programming language usage
- Declarative languages to the rescue
- SeNDlog = Binder + NDlog
- Access Control, formulation of protocols
- Contex (location) -> Identify components (nodes) in distributed systems
Review of Datalog

- Recursive query language
Review of Datalog

- Recursive query language
- Database community to query graph structures
Review of Datalog

- Recursive query language
- Database community to query graph structures
- Datalog program = set of rules + a query
Review of Datalog

- Recursive query language
- Database community to query graph structures
- Datalog program = set of rules + a query

1. \( p : - q_1, q_2, \ldots, q_n \)
Review of Datalog

- Recursive query language
- Database community to query graph structures
- Datalog program = set of rules + a query

\[ p : - q_1, q_2, \ldots, q_n \]

- function calls \`f\"
Review of Datalog

- Recursive query language
- Database community to query graph structures
- Datalog program = set of rules + a query

1. \[ p : - q_1, q_2, \ldots, q_n \]

- function calls `f`
- Here no negation
Language to express logics of access control
Binder

- Language to express logics of access control
- Notion of context
Binder

- Language to express logics of access control
- Notion of context

1. \( b_1 \) \text{may-access} (P, O, \text{read}) : - \text{good}(P).
2. \( b_2 \) \text{may-access} (P, O, \text{read}) : - \text{bob says may-access} (P, O, \text{read}).
Binder

- Language to express logics of access control
- Notion of context

1. \( b_1 \text{ may-access} (P, O, \text{read}) : \text{- good}(P) \).
2. \( b_2 \text{ may-access} (P, O, \text{read}) : \text{bob says may-access} (P, O, \text{read}) \).

- says = logical construct in authentication
Binder

- Language to express logics of access control
- Notion of context

1. \( b_1 \text{ may-access} (P, O, \text{read}) : \text{- good}(P) \).
2. \( b_2 \text{ may-access} (P, O, \text{read}) : \text{bob says may-access} (P, O, \text{read}) \).

- \text{says} = \text{logical construct in authentication}
- \text{principal} = \text{component (distributed) with its own context}
Binder

- Language to express logics of access control
- Notion of context

1. $b_1 \text{ may-access (P, O, read)} : - \text{ good(P)}$.
2. $b_2 \text{ may-access (P, O, read)} : - \text{ bob says may-access (P, O, read)}$.

- says = logical construct in authentication
- principal = component (distributed) with its own context
- Binder assumes untrusted networks
NDlog

- Database query language for expressing declarative networks
NDlog

- Database query language for expressing declarative networks

1. \( r_1 \) reachable \((@S,D)\) \(:\) \(-\) link \((@S,D)\).
2. \( r_2 \) reachable \((@S,D)\) \(:\) \(-\) link \((@S,Z)\), reachable \((@Z,D)\).
Database query language for expressing declarative networks

1. r1 reachable (@S,D) :- link (@S,D).
2. r2 reachable (@S,D) :- link (@S,Z), reachable (@Z,D).

Distributed transitive closure
NDlog

- Database query language for expressing declarative networks

1. \( r_1 \text{ reachable}(@S,D) : - \text{link}(@S,D). \)
2. \( r_2 \text{ reachable}(@S,D) : - \text{link}(@S,Z), \text{reachable}(@Z,D). \)

- Distributed transitive closure

- Notion of location \( \sim \) Binder's context
**NDlog**

- Database query language for expressing declarative networks

1. \( r_1 \text{ reachable} (@S,D) \) : - link (@S,D).
2. \( r_2 \text{ reachable} (@S,D) \) : - link (@S,Z), reachable (@Z,D).

- distributed transitive closure
- Notion of location ~ Binder's context
- @ denotes the location of each corresponding tuple
Binder vs NDlog

- Trusted (NDlog) vs Untrusted (Binder) Networks
Binder vs NDlog

- Trusted (NDlog) vs Untrusted (Binder) Networks
- Export of Derived Tuples (says vs @)
Binder vs NDlog

- Trusted (NDlog) vs Untrusted (Binder) Networks
- Export of Derived Tuples (says vs @)
- Bottom-up (Binder) vs Top Down (NDlog)

Spyros Hadjichristodoulou
Binder vs NDlog

- Trusted (NDlog) vs Untrusted (Binder) Networks
- Export of Derived Tuples (says vs @)
- Bottom-up (Binder) vs Top Down (NDlog)
- Binder: easier to implement rule resolving against security policies
Binder vs NDlog

- Trusted (NDlog) vs Untrusted (Binder) Networks
- Export of Derived Tuples (says vs @)
- Bottom-up (Binder) vs Top Down (NDlog)
- Binder: easier to implement rule resolving against security policies
- NDlog: set-oriented optimizations, avoiding infinite recursive loops, better fit for the incremental continuous execution model of network protocols
SeNDlog (intro)

- A unification of Binder and NDlog
SeNDlog (intro)

- A unification of Binder and NDlog
- As expressive as Binder and NDlog
SeNDlog (intro)

- A unification of Binder and NDlog
- As expressive as Binder and NDlog
- Support authenticate communication
SeNDlog (intro)

- A unification of Binder and NDlog
- As expressive as Binder and NDlog
- Support authenticate communication
- Enable differentiation of nodes according to their roles
SeNDlog (intro)

- A unification of Binder and NDlog
- As expressive as Binder and NDlog
- Support authenticate communication
- Enable differentiation of nodes according to their roles
- Flexibly support trusted and untrusted environments
SeNDlog (intro)

- A unification of Binder and NDlog
- As expressive as Binder and NDlog
- Support authenticate communication
- Enable differentiation of nodes according to their roles
- Flexibly support trusted and untrusted environments
- Amenable to efficient execution and optimizations (distributed query engine with bottom-up evaluation strategy)
SeNDlog (rules (i))

- A set of rules resides at a particular node
SENDlog (rules (i))

- A set of rules resides at a particular node

1. At $N$,
2. $r_1 \ p \ :- \ p_1, \ p_2, \ldots, \ p_n$.
3. $r_2 \ p_1 \ :- \ p_2, \ p_3, \ldots, \ p_n$. 

Spyros Hadjichristodoulou
Towards a Declarative Language and System for Secure Networking
**SENDlog (rules (i))**

- A set of rules resides at a particular node
  
  1. At N,
  2. \( r_1 \ p : - \ p_1, \ p_2, \ldots, \ p_n \).  
  3. \( r_2 \ p_1 : \ p_2, \ p_3, \ldots, \ p_n \).  

- N is a variable or constant
SENDlog \((\text{rules } \ (i))\)

- A set of rules resides at a particular node

1. At \(N\),
   2. \(r_1 \ p : - \ p_1, \ p_2, \ldots, \ p_n.\)
   3. \(r_2 \ p_1 : \ p_2, \ p_3, \ldots, \ p_n.\)

- \(N\) is a variable or constant
- Represents the principal where rules reside
A set of rules resides at a particular node

At $N$,
1. $r_1 \ p : - \ p_1, \ p_2, \ldots, \ p_n$.
2. $r_2 \ p_1 : \ p_2, \ p_3, \ldots, \ p_n$.

$N$ is a variable or constant

Represents the principal where rules reside

Instantiated with local info upon installation (IP, IP + pubkey, IP + pubkey + username, ...)

SeNDlog (rules (i))
SeNDlog (rules (ii))

- Additional conditions, checked at runtime
SeNDlog (rules (ii))

- Additional conditions, checked at runtime

1. At N, c1, c2, ..., cn
2. r1 p :- p1, p2, ..., pn.
3. r2 p1 :- p2, p3, ..., pn.
SeNDlog (rules (ii))

- Additional conditions, checked at runtime
  1. At \( N, c_1, c_2, \ldots, c_n \)
  2. \( r_1 p : - p_1, p_2, \ldots, p_n \).
  3. \( r_2 p_1 : - p_2, p_3, \ldots, p_n \).

- Rules are executed only if all conditions are met at runtime
Communicating Contexts

- Import and export tuples
  1. Maintenance messages as part of a protocol's updates
  2. Distributes derivation of security decisions
Communicating Contexts

- Import and export tuples
  1. Maintenance messages as part of a protocol's updates
  2. Distributes derivation of security decisions
- Import predicates ``N says p''
Communicating Contexts

- Import and export tuples
  1. Maintenance messages as part of a protocol's updates
  2. Distributes derivation of security decisions
- Import predicates ``N says p''
- Export predicates ``N says p@X''
Communicating Contexts

- Import and export tuples
  1. Maintenance messages as part of a protocol's updates
  2. Distributes derivation of security decisions

- Import predicates `N says p`
- Export predicates `N says p@X`

- So an SeNDlog rule is a Datalog rule where the body can include import predicates and the head can be an export predicate
Communicating Contexts

1 At N,
2 \[ e1 \quad p(X,X) : - \quad p1(X), \ p2(Y). \]
3 \[ e2 \quad p(X,Y,W) : - \ Y \text{ says } p1(X), \ Z \text{ says } p2(W), \ Z \neq N. \]
4 \[ e3 \quad P(Y,Z)@X : - \ p1(X), \ Y \text{ says } p2(Z). \]
5 \[ e4 \quad Z \text{ says } P(Y)@X : - \ Z \text{ says } p(Y), \ p1(X). \]
Communicating Contexts

1. At N,
2. \( e1 \quad p(X,X) : \neg \ p1(X), \ p2(Y). \)
3. \( e2 \quad p(X,Y,W) : \neg \ Y \ says \ p1(X), \ Z \ says \ p2(W), \ Z \neq N. \)
4. \( e3 \quad P(Y,Z)@X : \neg \ p1(X), \ Y \ says \ p2(Z). \)
5. \( e4 \quad Z \ says \ P(Y)@X : \neg \ Z \ says \ p(Y), \ p1(X). \)

- Honesty constraint: ``X says p'', X is constant or variable, X is N or ``X says P'' is in the body of the rule
Communicating Contexts

1. At $N$,

2. $e_1 \quad p(X,X) : - p_1(X) \land p_2(Y)$.

3. $e_2 \quad p(X,Y,W) : - Y \text{ says } p_1(X) \land Z \text{ says } p_2(W) \land Z \neq N$.

4. $e_3 \quad P(Y,Z)@X : - p_1(X) \land Y \text{ says } p_2(Z)$.

5. $e_4 \quad Z \text{ says } P(Y)@X : - Z \text{ says } p(Y) \land p_1(X)$.

- Honesty constraint: "$X \text{ says } p", X \text{ is constant or variable, } X \text{ is } N \text{ or } "X \text{ says } P" \text{ is in the body of the rule}

- Simple and secure implementation
Communicating Contexts

1. At N,

2. \( e_1 \) \( p(X,X) : - p_1(X), p_2(Y). \)

3. \( e_2 \) \( p(X,Y,W) : - Y \text{ says } p_1(X), Z \text{ says } p_2(W), Z \neq N. \)

4. \( e_3 \) \( P(Y,Z)@X : - p_1(X), Y \text{ says } p_2(Z). \)

5. \( e_4 \) \( Z \text{ says } P(Y)@X : - Z \text{ says } p(Y), p_1(X). \)

- Honesty constraint: ``X says p'', X is constant or variable, X is N or ``X says P'' is in the body of the rule

- Simple and secure implementation

- Whenever a ``says'' statement is exported a proof must be provided
At N,

1. $e_1 \ p(X,X) : - \ p_1(X), \ p_2(Y)$.
2. $e_2 \ p(X,Y,W) : - \ Y \ says \ p_1(X), \ Z \ says \ p_2(W), \ Z \neq N$.
3. $e_3 \ P(Y,Z)@X : - \ p_1(X), \ Y \ says \ p_2(Z)$.
4. $e_4 \ Z \ says \ P(Y)@X : - \ Z \ says \ p(Y), \ p_1(X)$.

- Honesty constraint: ``X says p'', X is constant or variable, X is N or ``X says P'' is in the body of the rule
- Simple and secure implementation
- Whenever a ``says'' statement is exported a proof must be provided
- Different levels of ``says''
Authenticated Path Vector Protocol

1 At Z,
2  z1  route (Z,X,P) :- neighbor (Z,X),
   P=f_initPath (Z,X).
3  z2  route (Z,Y,P) :- X says advertise (Y,P),
   acceptRoute (Z,X,Y).
4  z3  advertise (Y,P1)@X :-
   neighbor (Z,X), route (Z,Y,P),
   carryTraffic (Z,X,Y), P1=f_concat (X,P).
Secure DHT Node Identifiers

1  At NI,
2    nil requestCert(NI,K)@CA :-
3        startNetwork(NI),
4        publicKey(NI,K),
5        MyCA(NI,CA).
6  ni2 nodeID(NI,N) :-
7    CA says nodeIDCert(NI,N,K)
8  ni3 CA says nodeIDCert(NI,N,K)@LI :-
9    CA says nodeIDCert(NI,N,K),
10   landmark(NI,LI).
Secure DHT Node Identifiers

11 At CA,
12\[cal \ nodeIDCert(NI,N,K)@NI : -\]
13\[NI \ says \ requestCert(NI,K),\]
14\[S=secret(CA,NI),\]
15\[N=f_generateID(K,S).\]
16
17 At LI,
18\[li1 \ acceptJoinRequest(NI) : -\]
19\[CA \ says \ nodeIDCert(NI,N,K).\]
Conclusion and Future Work

- SeNDlog = Binder + NDlog
Conclusion and Future Work

- SeNDlog = Binder + NDlog
- Declarative language for secure networking
Conclusion and Future Work

- SeNDlog = Binder + NDlog
- Declarative language for secure networking
- Consider access control features from other languages
Conclusion and Future Work

- SeNDlog = Binder + NDlog
- Declarative language for secure networking
- Consider access control features from other languages
- Enhancements to the P2 declarative networking system
Conclusion and Future Work

- SeNDlog = Binder + NDlog
- Declarative language for secure networking
- Consider access control features from other languages
- Enhancements to the P2 declarative networking system
- Investigate analysis and cross-layer optimization opportunities
Thanks for your attention :-)