A Logic for Reasoning About Networked Secure Systems

Deepak Garg, Jason Franklin, Dilsun Kaynar, Anupam Datta

Carnegie Mellon University

June 22, 2008
Example Secure Systems: OpenSSH

- Widely used remote secure shell [RFC 4253]
- Based on network and memory primitives

> ssh server.cmu.edu
  ...
  Verifying signature
  Reading public key from Mem[0]
  ...

Client

Mem[0] = PK

Server
Example Secure Systems: Virtual Machine Monitors

- Widely deployed (e.g., VMware, Xen)
- Use memory protection and restricted APIs
Example Secure Systems: Trusted Computing

> Verify server code integrity
  ... sending system integrity measurements
  validating integrity measurements ...

- Upcoming technology (Intel TXT, AMD SVM, Microsoft Bitlocker)
- Uses special registers, restricted APIs
Motivation and Project Goals

- **Model** secure systems and adversaries
- **Specify** security properties
- **Prove** that systems satisfy properties

- Composition of systems and proofs (e.g., SSH over VMM)
- Insights into implementation (e.g., trusted Grub bootloader)
- Comparison of alternative system designs (e.g., remote attestation vs late launch)
Motivation and Project Goals

- **Model** secure systems and adversaries
- **Specify** security properties
- **Prove** that systems satisfy properties

- **Composition** of systems and proofs (e.g., SSH over VMM)
- **Insights** into implementation (e.g., trusted Grub bootloader)
- **Comparison** of alternative system designs (e.g., remote attestation vs late launch)
Technical Contributions

- Framework: Logic of Secure Systems ($LS^2$)
  - Based on Protocol Composition Logic (PCL)

- Programming language to specify systems and adversaries
  - Operational semantics defines reduction traces

- Logic to specify security properties
  - Predicates interpreted over traces

- Proof system to establish security properties
  - Soundness theorem ensures provable properties hold over all traces
Technical Contributions

- **Framework:** Logic of Secure Systems ($LS^2$)
  - Based on Protocol Composition Logic (PCL)

- **Programming language** to specify systems and adversaries
  - Operational semantics defines reduction traces

- **Logic** to specify security properties
  - Predicates interpreted over traces

- **Proof system** to establish security properties
  - Soundness theorem ensures provable properties hold over all traces
Technical Contributions

- **Framework:** Logic of Secure Systems ($LS^2$)
  - Based on Protocol Composition Logic (PCL)

- **Programming language** to specify systems and adversaries
  - Operational semantics defines reduction traces

- **Logic** to specify security properties
  - Predicates interpreted over traces

- **Proof system** to establish security properties
  - Soundness theorem ensures provable properties hold over all traces
Technical Contributions

- **Framework**: Logic of Secure Systems ($LS^2$)
  - Based on Protocol Composition Logic (PCL)

- **Programming language** to specify systems and adversaries
  - Operational semantics defines reduction traces

- **Logic** to specify security properties
  - Predicates interpreted over traces

- **Proof system** to establish security properties
  - **Soundness theorem** ensures provable properties hold over all traces
Outline

1. Design choices
2. Programming language and logic
3. Semantics and soundness
4. Conclusion
Outline

1. Design choices
2. Programming language and logic
3. Semantics and soundness
4. Conclusion
Secure System Primitives and Adversaries

Client

Nonce (n)

SIG(n, K_S^{-1})

Server

Network (Standard)
- Send, receive
- Crypto (sign, encrypt)

Local (New)
- Shared RAM and files
- Protection (access control)
- Steal, corrupt data
- Corrupt code

- Identify secure system primitives
- Model adversary capabilities, as opposed to enumerating attacks
Secure System Primitives and Adversaries

Client

Nonce (n)

Server

SIG(n, K_s^{-1})

<table>
<thead>
<tr>
<th>Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network (Standard)</td>
</tr>
<tr>
<td>- Send, receive</td>
</tr>
<tr>
<td>- Crypto (sign, encrypt)</td>
</tr>
</tbody>
</table>
Secure System Primitives and Adversaries

- Client
  - Nonce (n)
  - SIG(n, K_{S^{-1}})

- Server

<table>
<thead>
<tr>
<th>Primitives</th>
<th>Adversary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network (Standard)</td>
<td>- Send, receive</td>
</tr>
<tr>
<td></td>
<td>- Crypto (sign, encrypt)</td>
</tr>
<tr>
<td></td>
<td>- Symbolic (Dolev-Yao)</td>
</tr>
</tbody>
</table>
Secure System Primitives and Adversaries

Client

Nonce (n) SIG(n, K_{S^{-1}})

K_{S}

Server

Nonce (n)

SIG(n, K_{S^{-1}})

Primitives

Network (Standard)
- Send, receive
- Crypto (sign, encrypt)

Adversary

- Symbolic(Dolev-Yao)

Deepak Garg (CMU) Secure Systems FCS-ARSPA-WITS '08 9 / 24
Secure System Primitives and Adversaries

Client

\[ K_S \]

Nonce (n)

SIG(n, K_S^{-1})

Server

Primitives

Network (Standard)
- Send, receive
- Crypto (sign, encrypt)

Adversary
- Symbolic (Dolev-Yao)

Deepak Garg (CMU) Secure Systems FCS-ARSPA-WITS '08 9 / 24
Secure System Primitives and Adversaries

Client

\[ K_S \]

Nonce (n)

SIG(n, K_S^{-1})

Server

Primitives

Network (Standard)
- Send, receive
- Crypto (sign, encrypt)

Adversary
- Symbolic (Dolev-Yao)
Secure System Primitives and Adversaries

Primitives

Network (Standard)
- Send, receive
- Crypto (sign, encrypt)

Adversary
- Symbolic (Dolev-Yao)

- Identify secure system primitives
- Model adversary capabilities, as opposed to enumerating attacks
Secure System Primitives and Adversaries

Client

K_S

nonce (n)

SIG(n, K_S^{-1})

Server

Primitives

Network (Standard)
- Send, receive
- Crypto (sign, encrypt)

Local (New)
- Shared RAM and files
- Protection (access control)

Adversary

Network (Standard)
- Symbolic (Dolev-Yao)

Local (New)
- Steal, corrupt data
- Corrupt code

Deepak Garg (CMU) Secure Systems FCS-ARSPA-WITS '08 9 / 24
New Primitives and Adversary Capabilities in $LS^2$

- **Secure system primitives**
  - Read, write locations of memory (RAM and persistent storage)
  - Exclusive-write locks for integrity
  - (Extension with exclusive-read locks for secrecy)

- **Adversary capabilities**
  - Read memory
  - Write to unlocked memory
  - Lock unlocked memory
Outline

1. Design choices
2. Programming language and logic
3. Semantics and soundness
4. Conclusion
Programming Language

- Thread-oriented (process-calculus + explicit state)
  - Secure systems and adversaries modeled as threads

Action  \( a \) ::= send \( e \)

- receive
- sign \( e, K^{-1} \)
- verify \( e, K \)
- read \( l \)
- write \( l, e \)
- lock \( l \)
- unlock \( l \)
- proj\(_1\) \( e \)
- proj\(_2\) \( e \)
- match \( e, e' \)
- new

Program  \( P, Q \) ::= \( x_1 := a_1; \ldots; x_n := a_n \)

- See paper for details and operational semantics
Specifying Security Properties

- Properties specified in a logic

- Logic models explicit time (real numbers)
  - Action happened at a specific time
  - A program executed in a specified interval of time

- Time needed to model some systems of interest
  - E.g., Pioneer, Genuinity, TESLA

- In reasoning,
  - Time used to order events
  - Time used to state invariants
Logic: Syntax

Predicates

\[ R ::= \text{Send}(U, e) \mid \text{Receive}(U, e) \]
\[ \quad \mid \text{Sign}(U, e, K) \mid \text{Verify}(U, e, K) \]
\[ \quad \mid \text{Read}(U, l, e) \mid \text{Write}(U, l, e) \]
\[ \quad \mid \text{Lock}(U, l) \mid \text{Unlock}(U, l) \]
\[ \quad \mid \text{Match}(U, e, e') \mid \text{New}(U, n) \]

\[ M ::= \text{Mem}(l, e) \]
\[ \quad \mid \text{IsLocked}(l, U) \]
\[ \quad \mid \text{Contains}(e, e') \]
\[ \quad \mid e = e' \mid t \geq t' \]
\[ \quad \mid \text{Honest}(\hat{X}) \]
\[ \quad \mid \text{Honest}(\hat{X}, \vec{P}) \]

Formulas

\[ A, B ::= R \mid M \mid \top \mid \bot \mid A \land B \mid A \lor B \]
\[ \quad \mid A \supset B \mid \neg A \mid \forall x. A \mid \exists x. A \mid A @ t \]

Defined Formula

\[ A \text{ on } i = \forall t. ((t \in i) \supset (A @ t)) \]

Modal Formulas

\[ J ::= [P]^{t_{b}, t_{e}}_{U} A \mid [a]^{t_{b}, t_{e}}_{U, x} A \]
Proof System of the Logic

- Some axioms
  - Memory persists:
    \[ \vdash (\text{IsLocked}(l, U) \text{ on } [t_b, t_e] \land (\text{Mem}(l, e) \land t_b)) \land (\forall e'. \neg \text{Write}(U, l, e') \text{ on } [t_b, t_e])) \supset (\text{Mem}(l, e) \text{ on } [t_b, t_e]) \]
  - Locks persist:
    \[ \vdash ((\text{IsLocked}(l, U) \land t) \land (\neg \text{Unlock}(U, l) \text{ on } [t, t'])) \supset (\text{IsLocked}(l, U) \text{ on } [t, t']) \]

- Local reasoning: Proofs analyze only system components, not adversaries (cf. Hoare Logic and PCL)
  - Non-trivial with shared memory (what if another thread changes memory?)
  - Feasible because of appropriate memory protections

- In ongoing work we are using the proof system to analyze trusted computing protocols
Correctness Theorem for Example

\[ \Gamma \vdash J \text{ in } LS^2\text{’s proof system} \]

\[ \Gamma = \text{Honest}(\hat{K}_S, \text{Server}(K^{-1}_S)), \quad \hat{U} \neq \hat{K} \]

\[ J = \left[ \text{Client}(m, K_S) \right]_{t_b}^{t_e}_U \exists n. \exists t_g. \exists t_s. \exists U'. ((t_b < t_g < t_s < t_e) \wedge (\text{New}(U, n) @ t_g) \wedge (\hat{U'} = \hat{K}_S) \wedge (\text{Sign}(U', n, K^{-1}_S) @ t_s)) \]

- Proof reasons about memory and network primitives
- Protocol secure in presence of local and network adversary
- See full paper for details
Outline

1. Design choices
2. Programming language and logic
3. Semantics and soundness
4. Conclusion
Semantics and Soundness

Semantics of logic defined w.r.t. traces of programs ($\mathcal{T}$)
  ▶ A trace is a sequence of reductions of a set of threads
  ▶ We associate monotonically increasing time points with reductions

Semantic relations:
  ▶ $\mathcal{T} \models ^t A$
  ▶ $\mathcal{T} \models [P]_{t_b,t_e}^U A$

Example:
  ▶ $\mathcal{T} \models [P]_{t_b,t_e}^U A$ if whenever the reductions of thread $U$ in the interval $[t_b, t_e)$ on trace $\mathcal{T}$ match $P$, it is the case that $A$ holds.

Soundness Theorem:
  If $\Gamma \vdash \varphi$ then $\Gamma \models \varphi$
Outline

1. Design choices
2. Programming language and logic
3. Semantics and soundness
4. Conclusion
Ongoing Work

- Application to trusted computing
  - E.g., remote attestation protocol
  - E.g., sealed storage protocol

- More primitives and stronger adversary
  - Special hardware: PCRs, secure coprocessor
  - Adversaries that can reset machines
  - Adversaries that can modify code

- Unchanged memory model

- Composition of systems and proofs
  - E.g., sealed storage after remote attestation
Conclusion

- Advanced secure systems, formal techniques lacking
- Identifying relevant primitives, and modeling them
  - E.g., shared memory, memory protection, ...
- Specifying adversary capabilities instead of enumerating attacks
  - E.g., steal and corrupt data, corrupt code, reset machines
- Reasoning about security properties in presence of such adversaries
  - $LS^2$ supports local reasoning
- Technical contribution:
  - Programming language, logic, proof system, semantics
  - Soundness theorem
Thank You

Questions?
Extra Slides
Dense Time

- We assume a dense model of time
- Density does not appear in proof system
- Density needed to prove soundness

\[
\vdash [a]_{t_b,t_m}^I A_1 \quad \vdash [P]_{t_m,t_e}^I A_2 \quad (t_m \text{ fresh})
\]

\[
\vdash [x := a; P]_{t_b,t_e}^I \exists t_m. \exists x. ((t_b < t_m < t_e) \land A_1 \land A_2)
\]