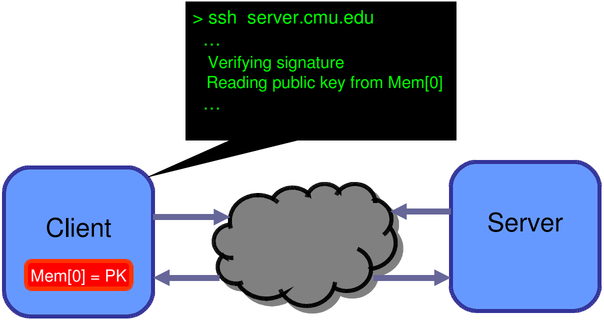
A Logic for Reasoning About Networked Secure Systems

Deepak Garg, Jason Franklin,
Dilsun Kaynar, Anupam Datta

Carnegie Mellon University

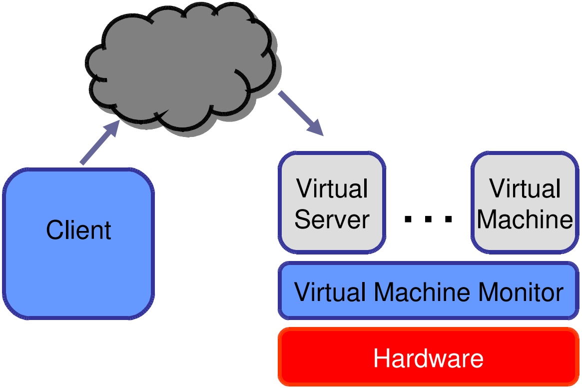
June 22, 2008

Example Secure Systems: OpenSSH



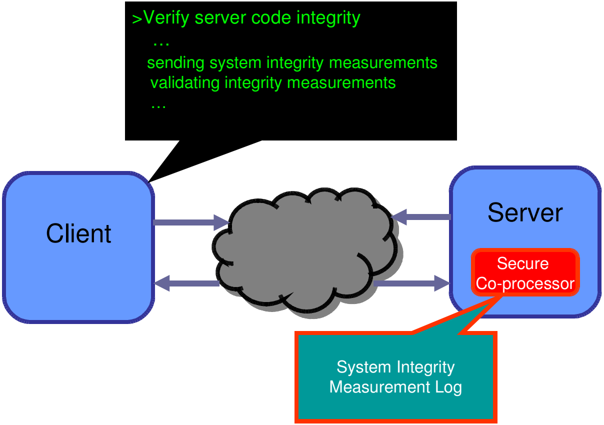
- Widely used remote secure shell [RFC 4253]
- Based on network and memory primitives

Example Secure Systems: Virtual Machine Monitors



- Widely deployed (e.g., VMware, Xen)
- Use memory protection and restricted APIs

Example Secure Systems: Trusted Computing



- 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

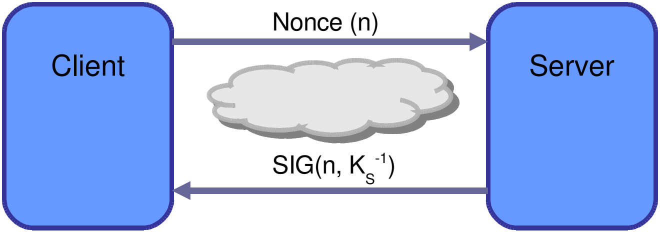
Outline

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

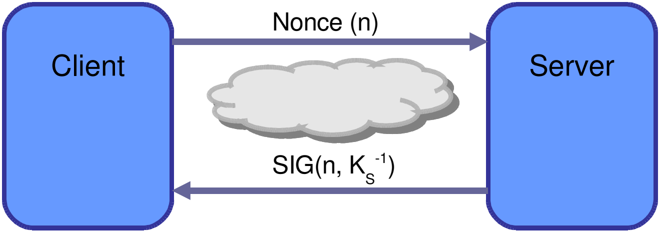
Outline

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

Secure System Primitives and Adversaries

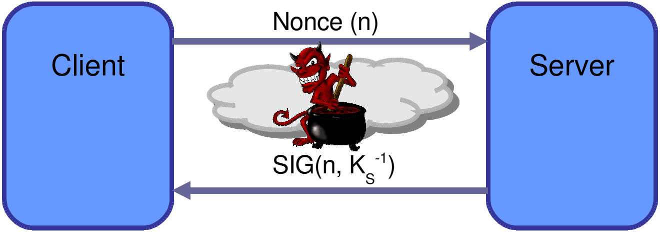


Secure System Primitives and Adversaries



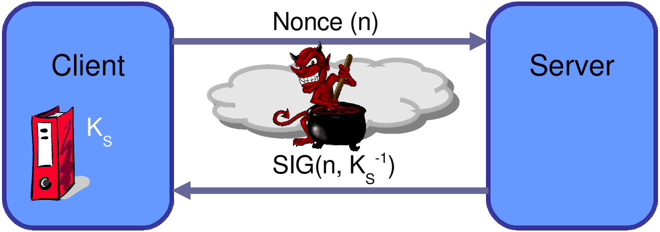
	Primitives	
Network (Standard)	- Send, receive - Crypto (sign, encrypt)	

Secure System Primitives and Adversaries



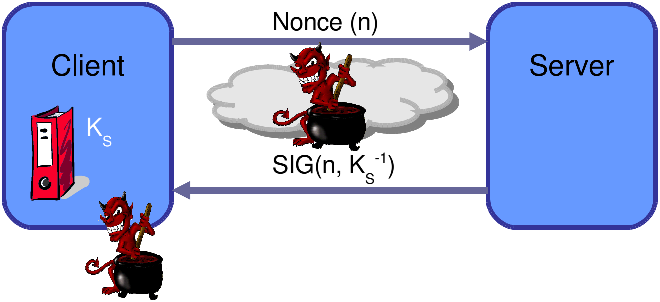
	Primitives	Adversary
Network (Standard)	- Send, receive - Crypto (sign, encrypt)	- Symbolic(Dolev-Yao)

Secure System Primitives and Adversaries



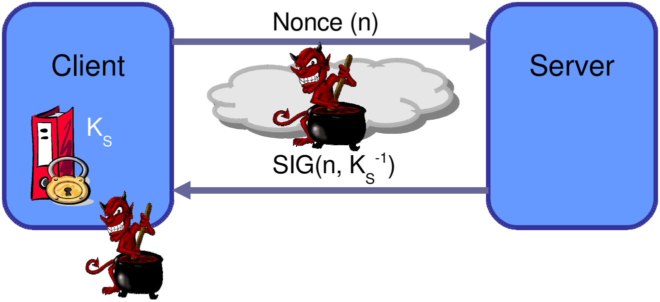
	Primitives	Adversary
Network (Standard)	- Send, receive - Crypto (sign, encrypt)	- Symbolic(Dolev-Yao)

Secure System Primitives and Adversaries



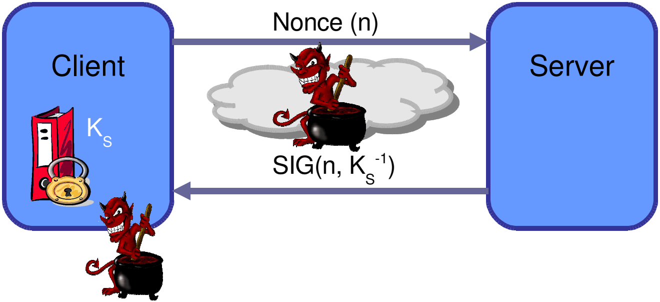
	Primitives	Adversary
Network (Standard)	- Send, receive - Crypto (sign, encrypt)	- Symbolic(Dolev-Yao)

Secure System Primitives and Adversaries



	Primitives	Adversary
Network (Standard)	- Send, receive - Crypto (sign, encrypt)	- Symbolic(Dolev-Yao)

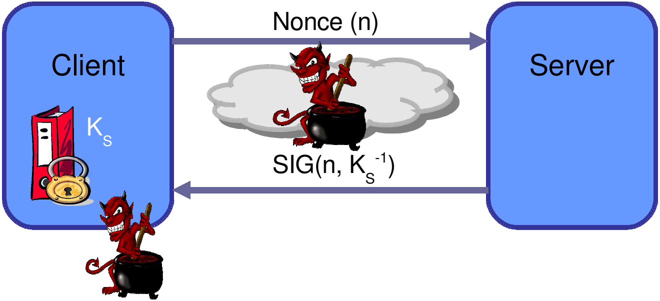
Secure System Primitives and Adversaries



	Primitives	Adversary
Network (Standard)	- Send, receive - Crypto (sign, encrypt)	- Symbolic(Dolev-Yao)

- Identify secure system primitives
- Model adversary capabilities, as opposed to enumerating attacks

Secure System Primitives and Adversaries



	Primitives	Adversary
Network (Standard)	- Send, receive - Crypto (sign, encrypt)	- Symbolic(Dolev-Yao)
Local (New)	- Shared RAM and files - Protection (access control)	- Steal, corrupt data - Corrupt code

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

$$
\begin{array}{llcl}
\text{Action} & a & ::= & \texttt{send}\, e \\
& & \mid & \texttt{receive} \\
& & \mid & \texttt{sign}\, e, K^{-1} \\
& & \mid & \texttt{verify}\, e, K \\
& & \mid & \texttt{read}\, l \\
& & \mid & \texttt{write}\, l, e \\
& & \mid & \texttt{lock}\, l \\
& & \mid & \texttt{unlock}\, l \\
& & \mid & \texttt{proj}_1\, e \\
& & \mid & \texttt{proj}_2\, e \\
& & \mid & \texttt{match}\, e, e' \\
& & \mid & \texttt{new} \\
\text{Program} & P, Q & ::= & x_1 := a_1; \ldots; x_n := a_n
\end{array}
$$

- 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	$::=$	$\text{Send}(U, e) \mid \text{Receive}(U, e)$
		$\mid$	$\text{Sign}(U, e, K) \mid \text{Verify}(U, e, K)$
		$\mid$	$\text{Read}(U, l, e) \mid \text{Write}(U, l, e)$
		$\mid$	$\text{Lock}(U, l) \mid \text{Unlock}(U, l)$
		$\mid$	$\text{Match}(U, e, e') \mid \text{New}(U, n)$
	M	$::=$	$\text{Mem}(l, e)$
		$\mid$	$\text{IsLocked}(l, U)$
		$\mid$	$\text{Contains}(e, e')$
		$\mid$	$e = e' \mid t \geq t'$
		$\mid$	$\text{Honest}(\hat{X})$
		$\mid$	$\text{Honest}(\hat{X}, \vec{P})$
Formulas	A, B	$::=$	$R \mid M \mid \top \mid \bot \mid A \wedge B \mid A \vee B \mid$
			$A \supset B \mid \neg A \mid \forall x.A \mid \exists x.A \mid A \ @\ t$
Defined Formula	A on i	$=$	$\forall t.\ ((t \in i) \supset (A \ @\ t))$
Modal Formulas	J	$::=$	$[P]_U^{t_b, t_e}\ A \quad \mid \quad [a]_{U,x}^{t_b, t_e}\ A$

Proof System of the Logic

- Some axioms
  - Memory persists:

    $$\vdash (\mathrm{IsLocked}(l, U) \text{ on } [t_b, t_e) \wedge (\mathrm{Mem}(l, e) \ @\ t_b) \wedge (\forall e'. \neg \mathrm{Write}(U, l, e') \text{ on } [t_b, t_e))) \supset (\mathrm{Mem}(l, e) \text{ on } [t_b, t_e))$$

  - Locks persist:

    $$\vdash ((\mathrm{IsLocked}(l, U) \ @\ t) \wedge (\neg \mathrm{Unlock}(U, l) \text{ on } [t, t'))) \supset (\mathrm{IsLocked}(l, U) \text{ on } [t, t'])$$

- 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

$$\boxed{\Gamma \vdash J \text{ in } LS^2\text{'s proof system}}$$

$$\Gamma \;=\; \text{Honest}(\hat{K}_s, \mathit{Server}(K_S^{-1})),\;\; \hat{U} \neq \hat{K}$$

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

- 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:
  - $\mathcal{T} \models^{t} A$
  - $\mathcal{T} \models [P]_{U}^{t_b,t_e} A$
- Example:
  - $\mathcal{T} \models [P]_{U}^{t_b,t_e} A$ 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

$$\frac{\vdash [a]_{I,x}^{t_b,t_m}\ A_1 \qquad \vdash [P]_{I}^{t_m,t_e}\ A_2 \qquad (t_m \text{ fresh})}{\vdash [x := a; P]_{I}^{t_b,t_e}\ \exists t_m.\exists x.\ ((t_b < t_m < t_e) \wedge A_1 \wedge A_2)}\text{Seq}$$