Mathematical FOUNDATIONS for Computer Security Policies

Michael Clarkson
The George Washington University

Mathematical Foundations of Programming Semantics
June 7, 2012
Formalization
&
Quantification
MADE IN
Stuxnet
U.S.A.

Flame Malware Statistics

86 DOMAINS for Command and Control
24 Current IP addresses hosting C&C
22 Different registration services used
8 COUNTRIES INFECTED

Percent of traffic from top countries
65%
Lebanon and Iran

Lebanon
Iran
Security today:
Security today:
Security today:
Security today:
Security today:
Security today:
Reactive
Proactive
Proactive

(foundational)
Security
CONFIDENTIALITY

INTEGRITY

AVAILABILITY
CONFIDENTIALITY

INTEGRITY

AVAILABILITY
CONFIDENTIALITY

INTEGRITY

A VAILABILITY

Formalize and verify any security policy?
CONFIDENTIALITY

INTEGRITY

AVAILABILITY

Formalize and verify any security policy?
Security vs. Correctness
Trace Properties

**Trace:** sequence of execution states

\[ t = s_0 s_1 \ldots \]

**Property:** set of infinite traces

\[ t \text{ satisfies } P \text{ iff } t \text{ is in } P \]

...individual
Program Correctness
ca. 1970s

- Partial correctness
- Termination
- Total correctness
- Mutual exclusion
- Deadlock freedom
- Starvation freedom
- ...???
Safety and Liveness

[Lamport 1977]

- **Safety**: “nothing bad happens”
  e.g., partial correctness, access control

- **Liveness**: “something good happens”
  e.g., termination, guaranteed service
Safety and Liveness
[Alpern & Schneider 1985, 1987]

- Fundamental basis
- Verification methodology
Security vs. Correctness
Security Policy

?= Correctness Property
Security is not a Property

Secure information flow:
Secret inputs are not leaked to public outputs
Security is not a Property

Secure information flow:
Secret inputs are not leaked to public outputs
Security is not a Property

Secure information flow:
Secret inputs are not leaked to public outputs

Bad thing: information leaked?
Security is not a Property

Noninterference:
Commands of high security users have no effect on observations of low security users [Goguen and Meseguer 1982]
Security is not a Property

Noninterference:
Commands of high security users have no effect on observations of low security users  [Goguen and Meseguer 1982]
Security is not a Property

Noninterference:
Commands of high security users have no effect on observations of low security users  [Goguen and Meseguer 1982]

\[ t_1: \begin{array}{cccccccc}
4 & 23 & 8 & 15 & 42 & 16 & 10 & 8 \\
\end{array} \]

\[ t_2: \begin{array}{cccc}
4 & 8 & 15 & 16 \\
\end{array} \]

Satisfaction depends on pairs of traces  
...not a property, not safety
Security is not a Property

**Service level agreements:**
e.g., average response time, over all executions, is bounded
Security is not a Property

Service level agreements:
e.g., average response time, over all executions, is bounded

Good thing: response time low enough?
Security is not a Property

Service level agreements:
e.g., average response time, over all executions, is bounded

Good thing: response time low enough?

Satisfaction depends on all traces
...not a property, not liveness
Satisfaction: \textit{sets of traces}

\textbf{Hyperproperty:} set of properties

= set of set of traces
Hyperproperties

Security policies are hyperproperties!

- **Information flow:** noninterference, relational noninterference, generalized noninterference, observational determinism, self-bisimilarity, probabilistic noninterference, quantitative leakage, corruption

- **Service-level agreements:** average response time, time service factor, percentage uptime
Safety and Liveness

- **Safety hyperproperties:** noninterference, observational determinism, ...

- **Liveness hyperproperties:** average response time, ...
Safety and Liveness

- **Safety hyperproperties**: noninterference, observational determinism, ...
- **Liveness hyperproperties**: average response time, ...

\((\leq \text{ on sets of traces: lower powerdomain})\)
Safety and Liveness

**Thm.** Every hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty.

**Thm.** Topological characterization:

- hypersafety = closed sets,
- hyperliveness = dense sets.

(Lower Vietoris construction applied to Plotkin topology on traces.)

...a fundamental basis
Verification

Thm. Stepwise refinement works with all safety hyperproperties.

Thm. Certain safety hyperproperties can be reduced to safety properties.

...verification techniques carry forward

self-composition [Barthe et al. 2004]
Expressivity

Security policies are **predicates** on systems.

Hyperproperties are **extensions** of those predicates.

...Hyperproperties can express any security policy!

(based on predicates and trace semantics)
Semantics

Encode into trace semantics:

• Relational semantics
• Labeled transition systems
• State machines
• Probabilistic systems

...hyperproperties applicable with all
Semantics

Encode into trace semantics:

• Relational semantics
• Labeled transition systems
• State machines
• Probabilistic systems

...hyperproperties applicable with all

Direct definitions & theorems in these semantics?
CONFIDENTIALITY
INTEGRITY
AVAILABILITY
Hyperproperties

- formal foundation
- orthogonal
- verification...
- expressive
Qualitative
Qualitative
“When you can measure what you are speaking about, and express it in numbers, you know something about it;

but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind;

it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science.”

—William Thomson, 1st Baron Kelvin
Quantitative
Quantitative
How much information does a program leak?

[Denning 1982]
How much information does a program leak?

[Denning 1982]

covert channels
How much information does a program leak?

[Denning 1982]

covert channels

CONFIDENTIALITY
How much information does a program corrupt?

INTEGRITY
Integrity?
Integrity?

Common Criteria:

Protection of assets from unauthorized modification

Databases:

Relational constraints, provenance, utility

Biba [1977]:

Guarantee that subsystem will perform as intended

Dual to confidentiality
Biba Duality

Lattice of security levels
Biba Duality

Lattice of security levels
Biba Duality

Lattice of security levels

Trusted

Untrusted

security of information
Biba Duality

Rule 1.
No read down

Lattice of security levels

security of information

Trusted

Untrusted
Biba Duality

Rule 1.
No read down

Rule 2.
No write up

Lattice of security levels

security of information
Biba Duality

Rule 1.
No read down

Rule 2.
No write up

Lattice of security levels
for confidentiality
Biba Duality

Rule 1.
No read down

Rule 2.
No write up

Lattice of security levels for confidentiality
Biba Duality

Lattice of security levels for confidentiality

Rule 1.
No read down

Rule 2.
No write up

[Bell and LaPadula 1973]
Biba Duality

- **Top Secret**
- **Secret**
- **Confidential**
- **Unclassified**

- **Trusted**
- **Untrusted**

No read up, no write down
No read down, no write up

[Blba 1977]
Biba Duality

No read up, no write down
Biba Duality

No read up, no write down
Biba Duality

No read up,
no write down
Integrity?

Common Criteria:

Protection of assets from unauthorized modification

Databases:

Relational constraints, provenance, utility

Biba [1977]:

Guarantee that subsystem will perform as intended
Dual to confidentiality

...no universal definition
**Integrity**

Corruption: damage to integrity

<table>
<thead>
<tr>
<th>Starting point</th>
<th>Corruption measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program correctness</td>
<td>Suppression</td>
</tr>
<tr>
<td>Taint analysis</td>
<td>Contamination</td>
</tr>
</tbody>
</table>
# Integrity

**Corruption**: damage to integrity

<table>
<thead>
<tr>
<th>Starting point</th>
<th>Corruption measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program correctness</td>
<td>Suppression</td>
</tr>
<tr>
<td>Taint analysis</td>
<td>Contamination</td>
</tr>
</tbody>
</table>

**Suppression**: good information missing from output

**Contamination**: bad information present in output
Integrity

**Corruption:** damage to integrity

<table>
<thead>
<tr>
<th>Starting point</th>
<th>Corruption measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program correctness</td>
<td>Suppression</td>
</tr>
<tr>
<td>Taint analysis</td>
<td>Contamination</td>
</tr>
</tbody>
</table>

**Suppression:** good information missing from output

**Contamination:** bad information present in output

...distinct, but interact
Contamination is closely related to taint analysis of Livshits and Lam, Newsome, and Song, Su et al., and Wall et al. which tracks information flow from untrusted inputs to outputs that are supposed to be trusted. Such flow results in what we call contamination of the trusted outputs. We might be willing to deem a program secure if it allows only a limited amount of contamination, but taint analysis would deem the same program to be insecure. So quantification of contamination would be useful.

Flow between untrusted and trusted objects was first studied by Biba, who identified a duality between models of integrity and confidentiality. The confidentiality dual to contamination is leakage, which is information flow from secret inputs to public outputs. Previous work has developed measures of leakage based on information theory and beliefs. This paper adapts those measures to contamination.

Through the Biba duality, we obtain a measure for corruption from a measure for leakage.

Suppression is connected to program correctness, which is often phrased in terms of specifications and implementations. For a given input, an implementation should produce an output $o$ that conveys a value $v$ permitted by a specification. However, $o$ and $v$ need not be identical: an implementation might output all the bits in the binary representation of $v$ but in reverse order, or it might output $v \oplus k$ where $k$ is a known constant. It suffices that knowledgeable users can recover $v$ from $o$.

The output of an incorrect implementation would fail to fully convey value $v$. For example, an implementation might output only the first few bits of $v$; or it might output $v$ with probability $p$ and output garbage with probability $1-p$; or it might output $v \oplus u$ where $u$ is an untrusted input. In each case, we say that specification-violating suppression of information about the correct output value has occurred. Throughout this paper, we use a programming notation to write specifications as well as implementations, so henceforth we use the more succinct term program suppression instead of specification-violating suppression.

Newsome et al. adapt the same information-theoretic metric to measure what they call influence.
Quantification

Information theory: information is surprise

⇒ Quantify number of bits suppressed/contaminated
Quantification

Information theory: information is surprise

→ Quantify number of bits suppressed/contaminated

\[
\text{Suppression} = H(\text{Spec} \mid \text{Impl})
\]

\[
\text{Contamination} = I(\text{U}_{\text{in}}, \text{T}_{\text{out}} \mid \text{T}_{\text{in}})
\]
Quantification

Information theory: information is surprise

- Quantify number of bits suppressed/contaminated

Suppression = $H(\text{Spec} \mid \text{Impl})$

Contamination = $I(\text{U}_{\text{in}}, \text{T}_{\text{out}} \mid \text{T}_{\text{in}})$

...need to treat programs as distribution transformers
Semantics

Use probabilistic semantics of while programs

[Kozen 1981]
Semantics

Use probabilistic semantics of while programs

[Kozen 1981]

What if adversary can nondeterministically affect program choices? e.g., scheduler
Semantics

Use probabilistic semantics of while programs
[\text{Kozen 1981}]

What if adversary can nondeterministically affect program choices? e.g., scheduler

Probabilistic+nondeterministic semantics?
Duality

Contamination: Generalizes [Newsome et al. 2009], dual to [Clark et al. 2005, 2007]

Suppression: No (interesting) dual!

...integrity isn’t as simple as we thought
CONFIDENTIALITY

INTEGRITY

AVAILABILITY
Leakage and Suppression

**Declassifier:** program that reveals (leaks) some information, and hides (suppresses) the rest
Leakage and Suppression

**Thm.** Leakage + suppression = $I(input)$
(assuming only security level is trusted+secret)
Leakage and Suppression

Thm.  Leakage + suppression = I(input)
      (assuming only security level is trusted+secret)

...what isn’t leaked is suppressed
Leakage and Suppression

**Thm.** Leakage + suppression = $l(input)$
*(assuming only security level is trusted+secret)*

...what isn’t leaked is suppressed

...quantification yields a foundation for understanding behavior of declassifiers
Database Privacy

- **k-anonymity** [Samarati & Sweeney 1998]
  no bound on leakage or suppression

- **ℓ-diversity** [Machanavajjhala et al. 2007]
  suppression $\geq \log(\ell)$

- **γ-amplification** [Evfimievski et al. 2003]
  leakage $\leq \log(\gamma)$

- **ε-differential privacy** [Dwork et al. 2006]
  leakage about individual $\leq \varepsilon$
Database Privacy

- $k$-anonymity [Samarati & Sweeney 1998]
  no bound on leakage or suppression

- $\ell$-diversity [Machanavajjhala et al. 2007]
  suppression $\geq \log(\ell)$

- $\gamma$-amplification [Evfimievski et al. 2003]
  leakage $\leq \log(\gamma)$

- $\varepsilon$-differential privacy [Dwork et al. 2006]
  leakage about individual $\leq \varepsilon$

...foundation for comparison of security
Quantitative
Summary

• Foundations for **formalization** of security
• Foundations for **quantification** of security
1. Quantification of Integrity. Submitted for journal publication, 2011. With Fred B. Schneider.


Mathematical Foundations of Computer Security Policies

Michael Clarkson
The George Washington University

Mathematical Foundations of Programming Semantics
June 7, 2012