Language-Based Information-Flow Security

(Sabelfeld and Myers)

"Practical methods for controlling information flow have eluded researchers for some time."

Presented by David L. Rager

"Conventional" Approach

- Access control lists (ACLs)
  - Checks release of data but not data propagation
  - What happens if a host becomes unknowingly corrupted?
  - Approach is fundamentally doomed
- Firewalls
- Anti-virus
- Encryption

Language-based Attempts

- Java
  - Bytecode verifier
  - Sandbox mode
  - Stack inspection
- Not intended to control information flow, and therefore insufficient

The “New” Approach

- Information-flow policies
  - “confidentiality policies we wish to enforce”
  - “A natural way to apply the well-known systems principle of end-to-end design”
- Information-flow controls
  - Mechanisms that implement the above policies

Terminology

- Confinement – the ability to prevent capabilities (and authority) from being transmitted improperly
- Noninterference – no data visible publicly is affected by confidential data
- "High" security versus "low" security – the idea that some code and data is associated with being inaccessible and other code and data is public (these are not technical terms)

Covert Channels

- Channel – a mechanism for signaling information through a computing system
- Covert channel – a channel whose primary purpose is not information transfer

Types of Covert Channels

- Implicit flows – signal information through the control structure of a program
- Termination channel – signal information through the termination or nontermination of computation
- Timing channel – signal information through the time at which an action occurs rather than through the data associated with the action

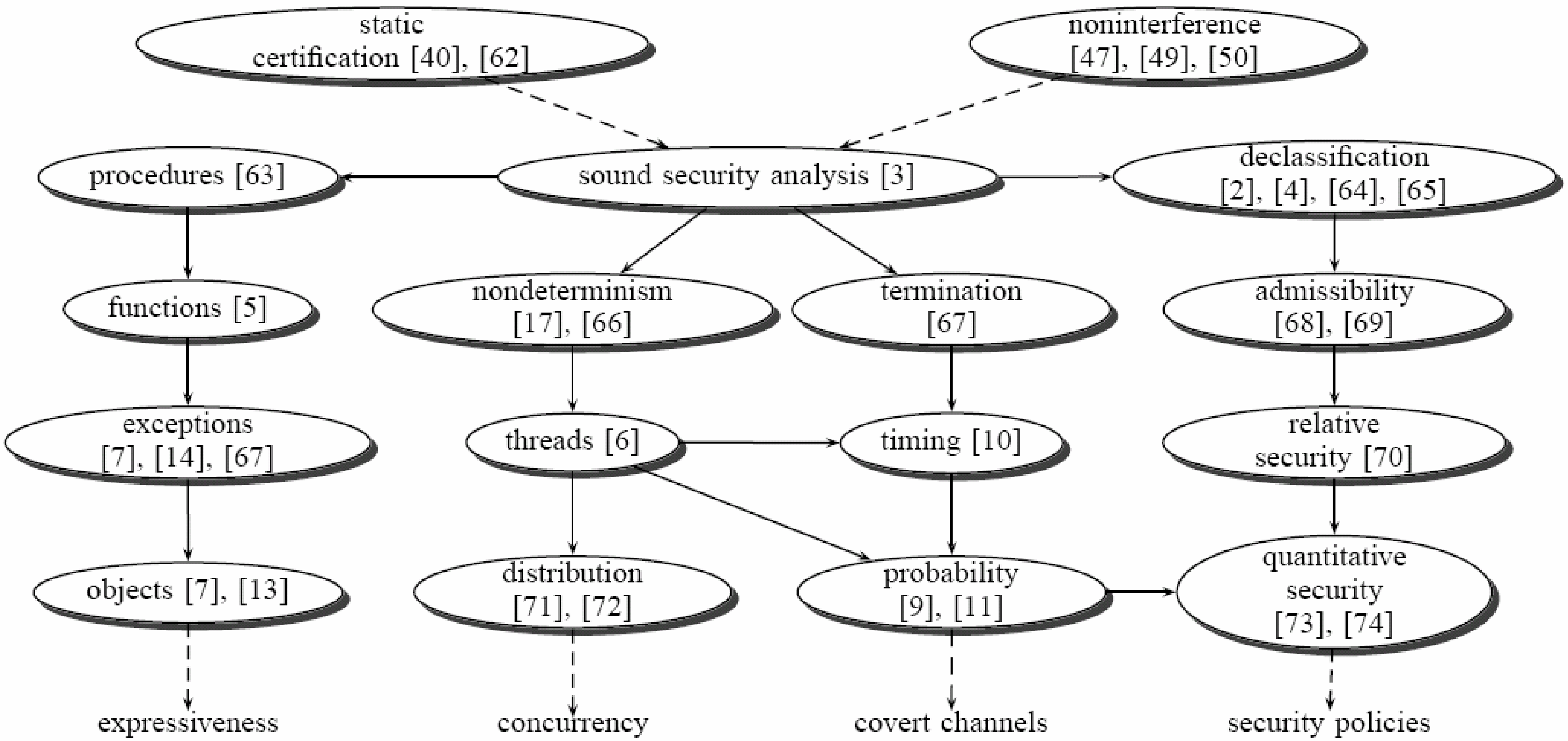
Types of Covert Channels (cont’d)

- Probabilistic channel – signal information by changing the probability distribution of observable data
- Resource exhaustion channel – signal information by the possible exhaustion of a finite, shared resource
- Power channel – embed information in the power consumed by the computer

Four Directions of Language-Based Security

- Enriching *expressiveness* of the language
- Exploring impact of *concurrency*
- Analyzing *covert channels*
- Refining *security policies*

Four Directions of Language-Based Security



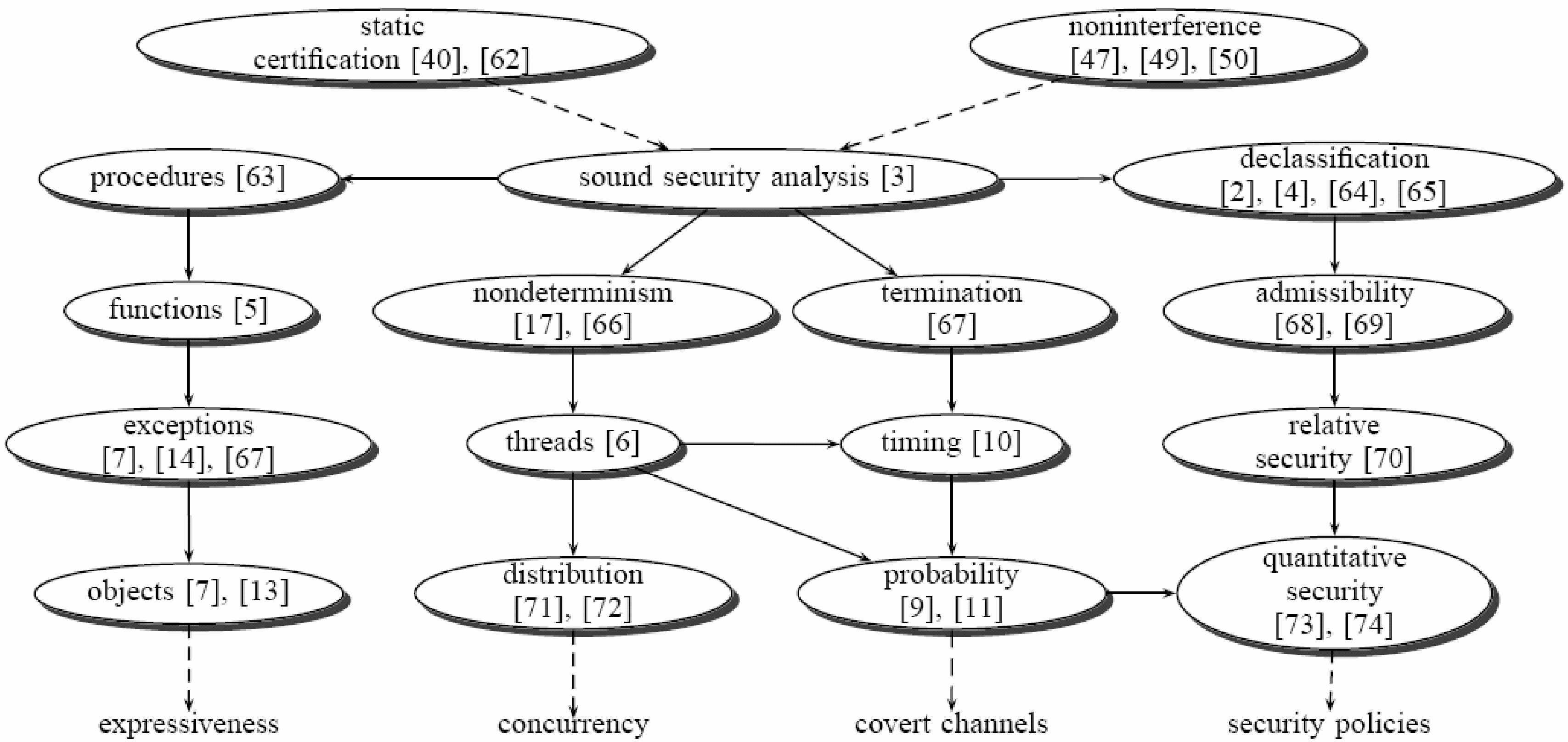
Expressiveness

- Polymorphism
  - The function *h* can be overloaded to have different definitions depending on whether its context is high or low
- Functions
  - SLam is based off the lambda calculus and proposes a type system for confidentiality and integrity

Expressiveness (cont'd)

- Exceptions
  - *Path labels* can be used to allow finer-grained tracking of implicit flows caused by exceptions
- Objects
  - JFlow language extends Java with a type system for tracking information flow
  - Barthe and Serpette created an OO language based on Abadi-Cardelli functional object calculi and show their type system enforces noninterferance

Four Directions of Language-Based Security



Concurrency

- Nondeterminism
  - Consider the observable behavior of the program to be the set of its possible results
  - Secure if high inputs do not affect the set of *possible* low outputs
  - *Possibilistic* security

Concurrency

- Thread concurrency
  - If two high security programs execute in parallel, they can “do evil”
- Example

High assurance level program 1:

h:=0; l:= h // secure since 0 is a public constant

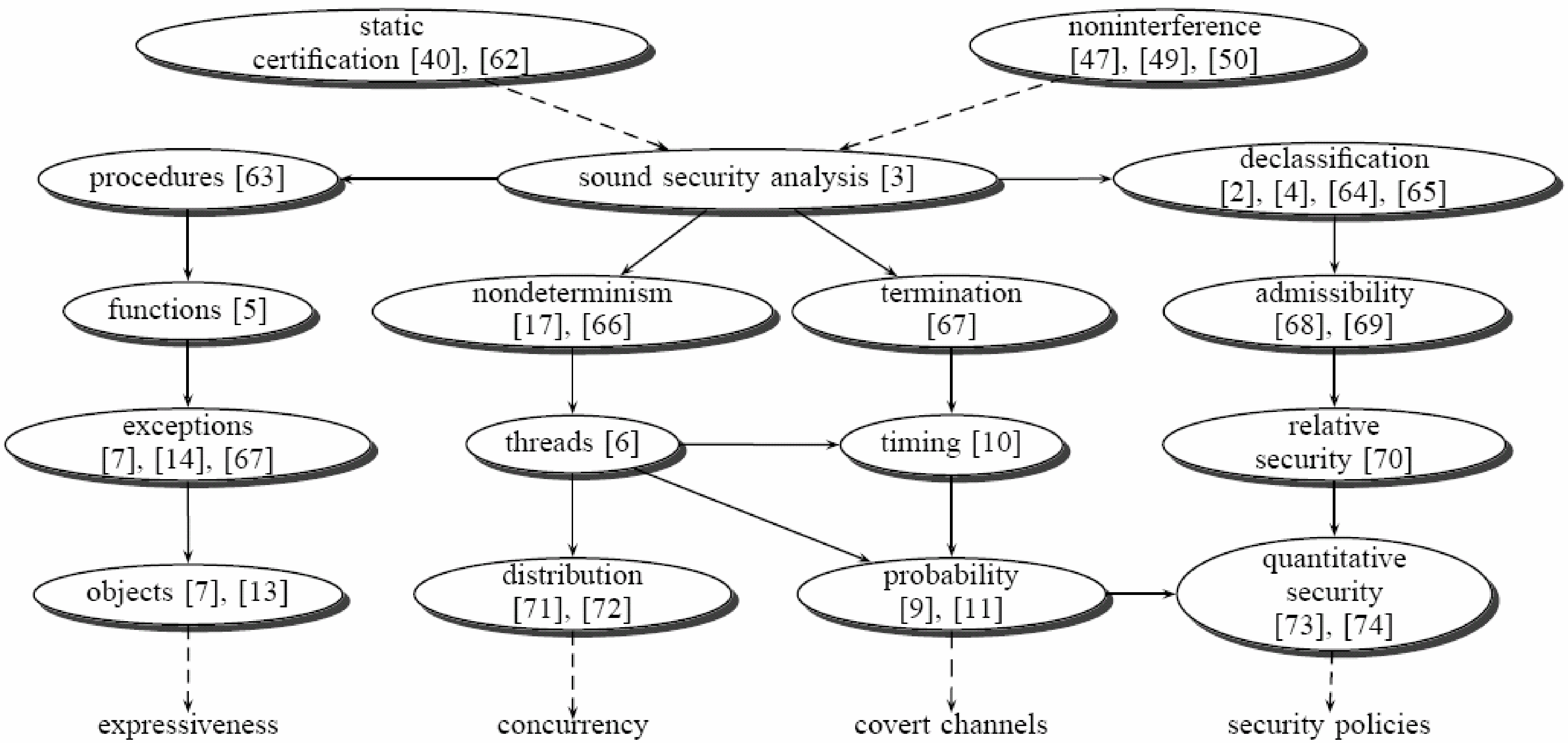
High assurance level program 2:

h:=h’ // if this program interleaves in program 1’s execution, then h’ will become public

Concurrency

- Distribution
  - Messages are exchanged and these exchanges can often be observed
  - Often distributed systems don't completely trust each other
  - Components of distributed systems can fail (or be subverted)

Four Directions of Language-Based Security



Covert Channels

- Termination Channels
  - If an attacker can observe termination some programs are insecure
  - Ex:

    while h = 1 do skip

- Solution
  - No while loop may have a high guard
  - No high conditional may contain a while loop in its branch

Covert Channels

- Timing Channels
  - If an attacker can observe termination some programs are insecure
  - Ex (C_{long} is a series of time consuming operations):
    if h = 1 then C_{long} else skip
- One solution to this example
  - No high conditional may contain a while loop in its branch
  - Wrap each high conditional in a protect statement whose execution is atomic
- Practical example: RSA encryption attack[101]

Covert Channels

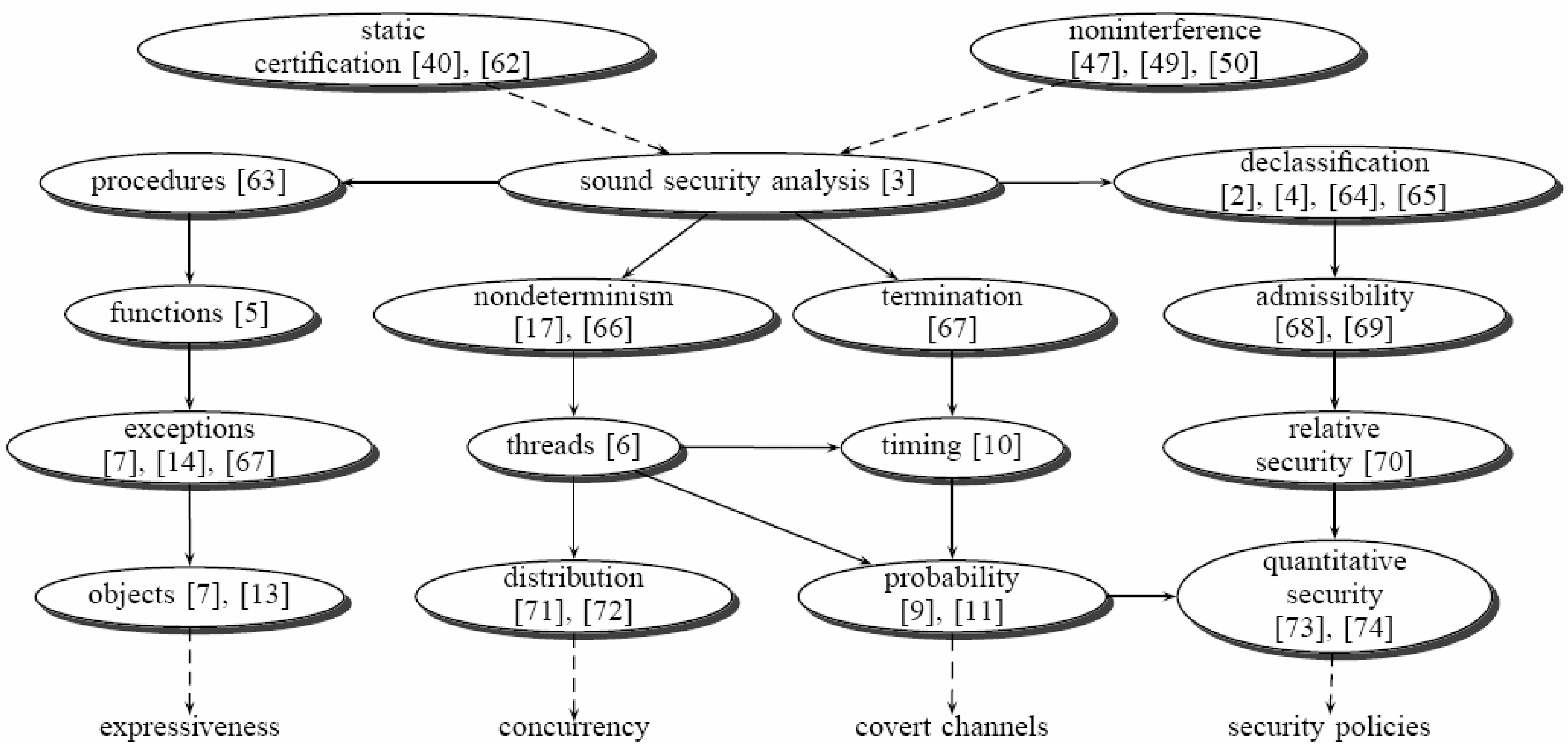
- Probabilistic Channels
- Ex:

  l:=PIN $[]_{9/10}$ l:=rand(9999)

  $[]_{9/10}$ means perform the left side 90% of the time and the right side 10% of the time

  - Possibilistically secure
  - Why isn't it probabilistically secure?

Four Directions of Language-Based Security



Security Policies

- Declassification
  - Noninterference rejects downgrading of security levels
  - Think of cryptography
- Admissibility
  - Explicitly states which dependencies are allowed between data (including those caused by downgrading)
  - An admissible program has no other information flows than those intended by the protocol specification
- Quantitative security
  - A limited number of information leaks is acceptable

Open Challenges

- System-wide security
  - Correctly integrating particular security implementations into a system is hard
- Certifying compilation
  - Must trust the type checkers and compilers
  - Remember Robert's Openmcl presentation?
  - A solution: proof carrying code
- Abstraction-violating attacks
  - Ex: cache attacks
- Dynamic policies
  - Need to support the changing of permissions across the lifetime of data

Conclusion

- Conventional methods of security (access control lists, virus detection, firewalls) insufficient
- Four Directions of Language-Based Security
  - Enriching *expressiveness* of the language
  - Exploring impact of *concurrency* on security
  - Analyzing *covert channels*
  - Refining *security policies*