TRUST DISTRIBUTION DIAGRAMS: THEORY AND APPLICATIONS

Michael E. Locasto, University of Calgary

"There are two ways to design a system. One is to make it so simple there are obviously no deficiencies. The other is to make it so complex there are no obvious deficiencies." – C. A. R. Hoare

Takeaway Message

Vary whatever you wish, but make sure you understand how such alterations affect the trust relationships in the system's design and implementation.

Conclusion 1: Trust Relationships

Systems are composed of trust relationships; we must understand how the process of varying system properties (i.e., "moving target") affects these trust relationships **and** frustrates attackers' ability to control primitives in the computing environment

Outcome: create an artifact for documenting the nature of these trust relationships

Conclusion 2: Security Coordination

Security systems routinely interfere with each other (i.e., "bickering-in-depth"); we need a framework for negotiating over security-critical resources, measurements points, data structures, and hooks

Outcome: TDDs should provide a way of understanding the composition/layering of multiple *security* mechanisms

Work That Shaped My Thinking

"We Need Assurance!", Brian Snow, ACSAC 2005

"Some Thoughts on Security after Ten Years of qmail 1.0", DJB, CSAW 2007

"High Assurance Digital Forensics: A Panelist's Perspective", Steven J. Greenwald, SADFE2009

Time Out: What Do You Mean by "Trustworthy"?

"…we equate 'trustworthy' with the notion that software 'follows expected behavior' according to some security policy (where 'behavior' consists of sequences of events that read or modify specific data structures)."

Trust Distribution Diagrams

Motivation, Theory, and Applications

Work in progress!

Observations: Software Assurance

Observation 1: The academic research community seems to have lost the art of making assurance arguments (CC EAL==red herring here)

Observation 2: Somehow "small" (as in "fewest lines of code") has become our best metric for software trustworthiness, but we posit that *the relationship between size and trustworthiness remains ill-defined*

Key Issue: Increase in Complexity

Challenge 1: difficult to argue effectively in prose

Challenge 2: difficult to construct & maintain formal proof for complex, evolving system

An Alternative to Lines of Code

"Perhaps a better measure of assurance should rely on the complexity of the trust relationships between system components."

An Alternative to Lines of Code

"Perhaps a better measure of assurance should rely on the complexity of the trust relationships between system components."

How do you depict these relationships?

Trust Distribution Diagrams

TDDs will define a graphical language for expressing the distribution, amount, and migration of trust in design-level components.

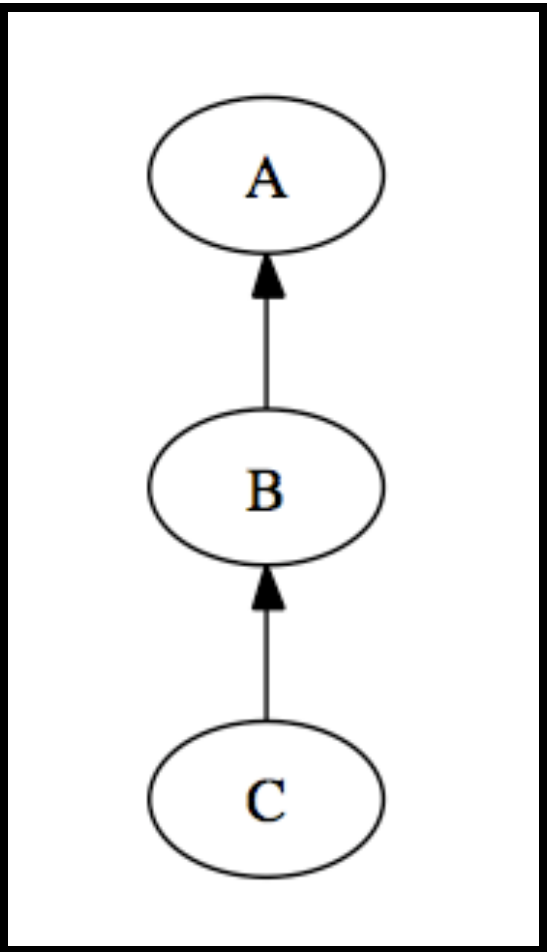
TDD Key Properties

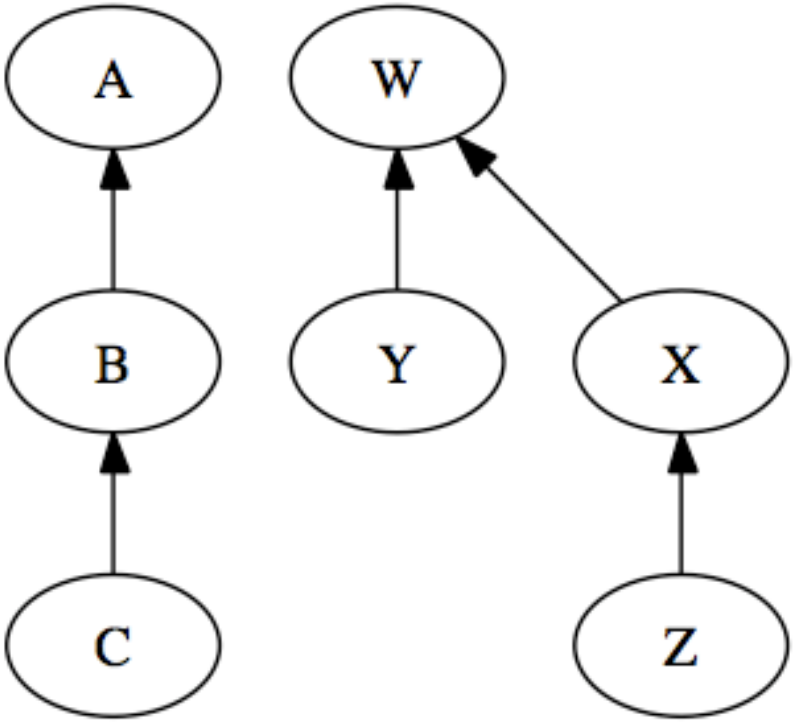
Direction of trust relationships (map)

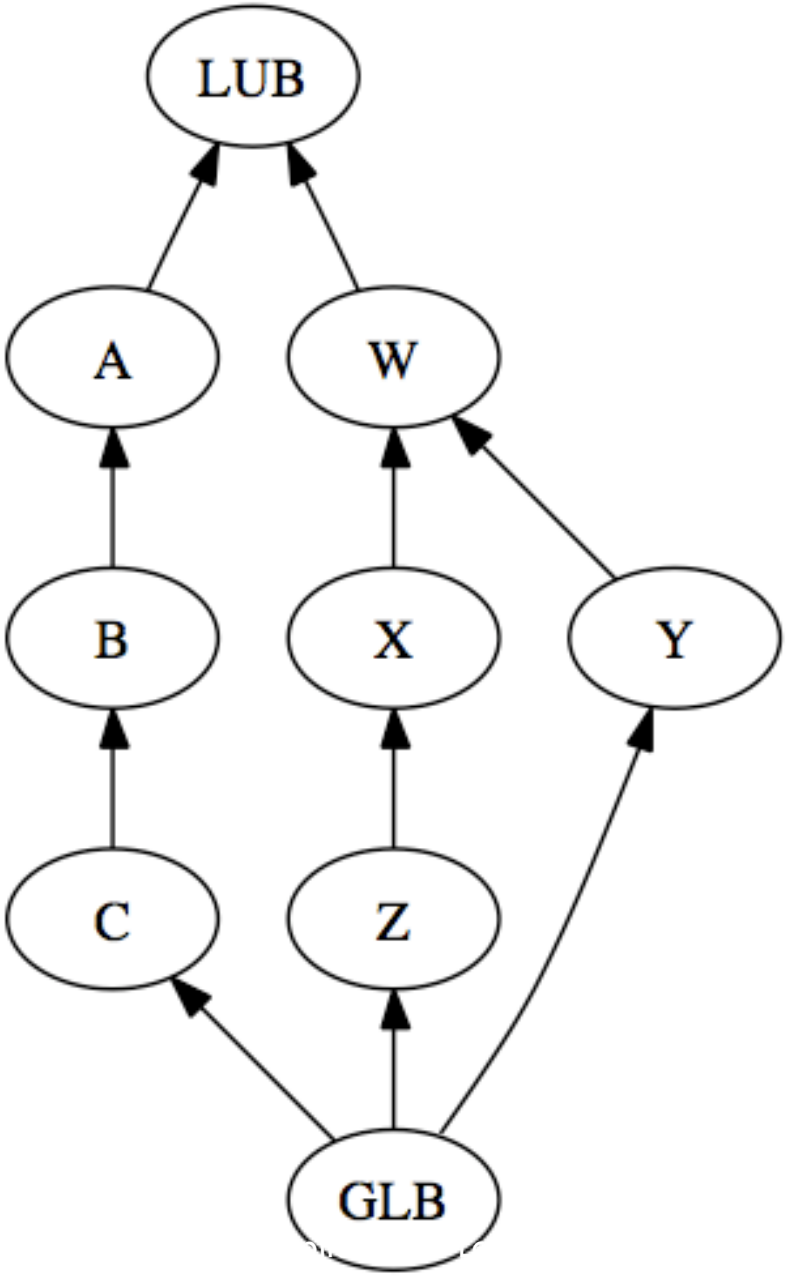
Location of trust regardless of level of trust (orthagonality)

How direction, location, and level change over time (duration and migration)

Mapping Trust Between Components




A
W
B
Y
X
C
Z


LUB
A
W
B
X
Y
C
Z
GLB

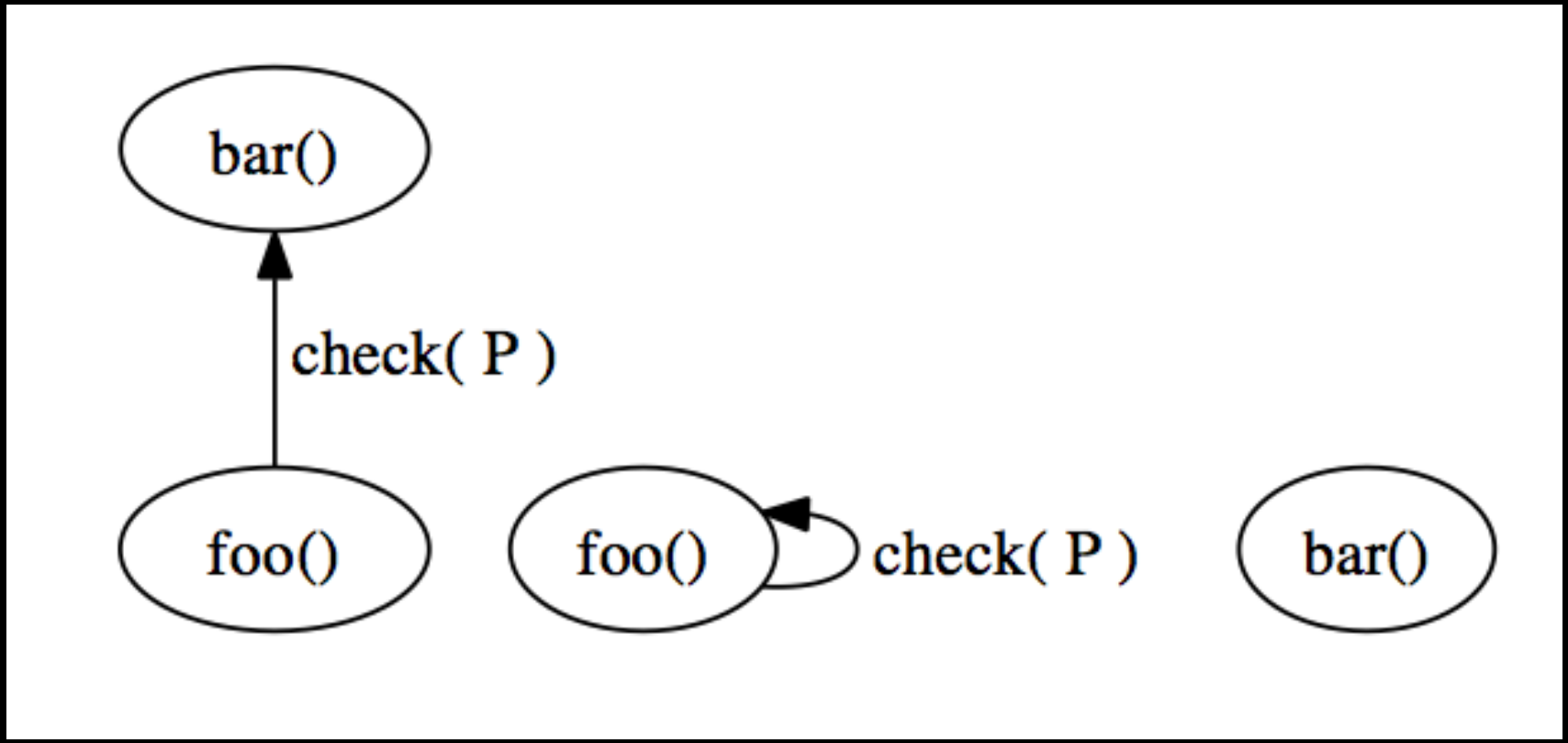
Depicting Trust Statements

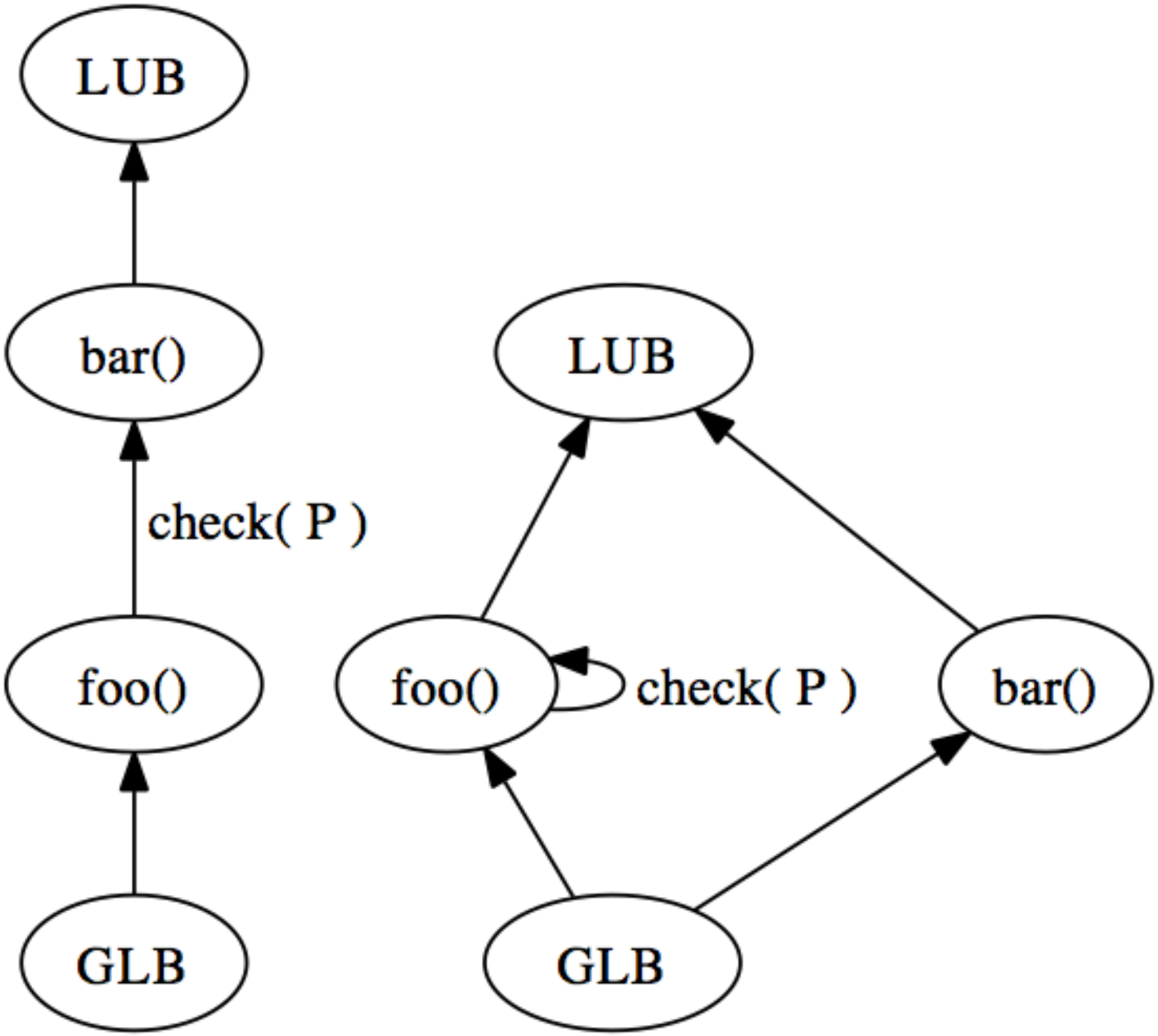
Trust Policy =

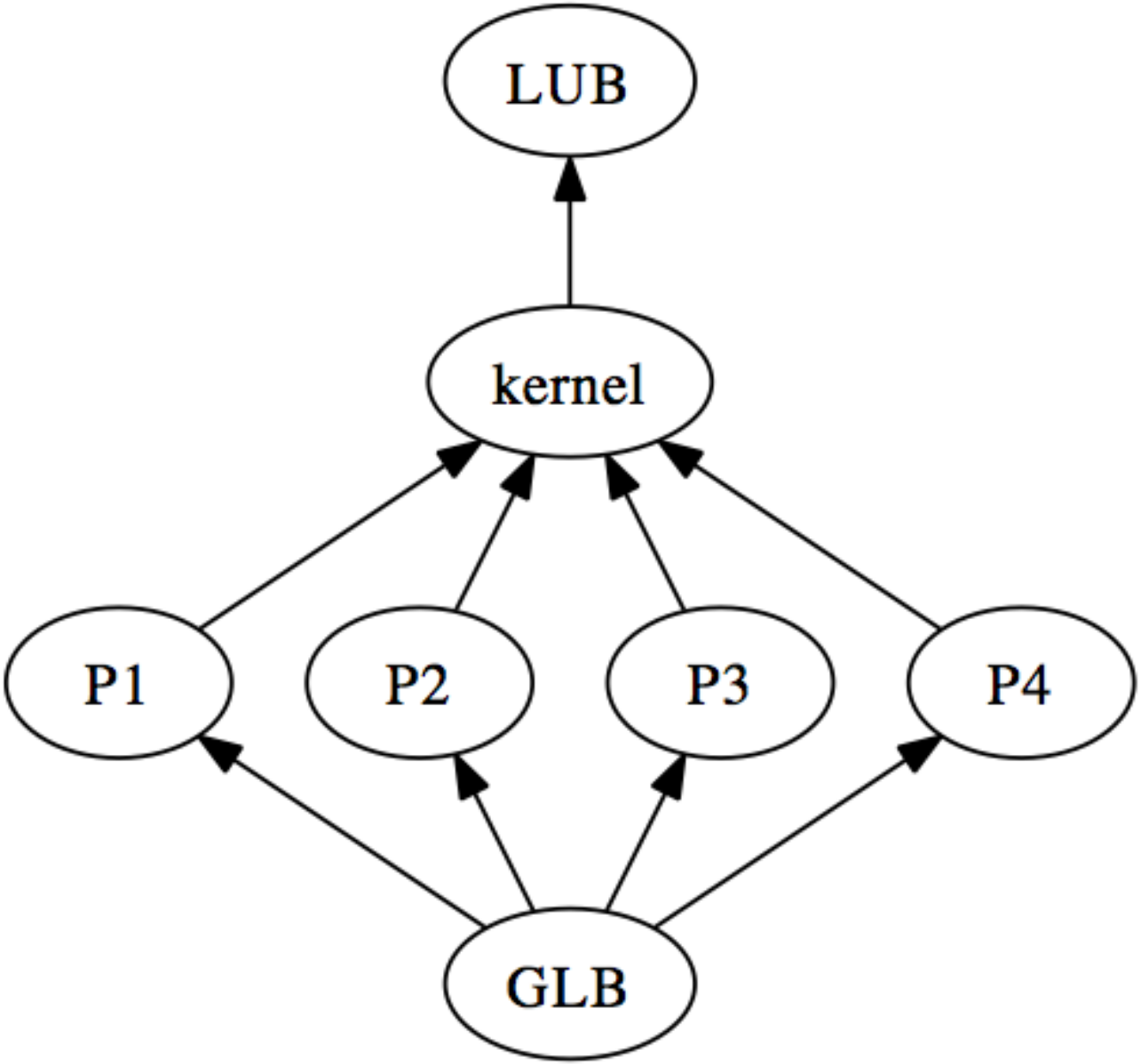
Trust Statements + Consequences

"Function `foo` trusts function `bar` to check property P."

"…if `bar` does not, `foo` will henceforth check P itself."


bar()
check(P)
foo()
foo()
check(P)
bar()


LUB
bar()
check(P)
foo()
GLB
LUB
foo()
check(P)
bar()
GLB


LUB
kernel
P1
P2
P3
P4
GLB

Tasks that Require Further Work

- Exact syntax and semantics
- Represent *evolving* graph structure
- Leverage complexity measure as a coherent basis for qualitative trustworthiness arguments (next slide)

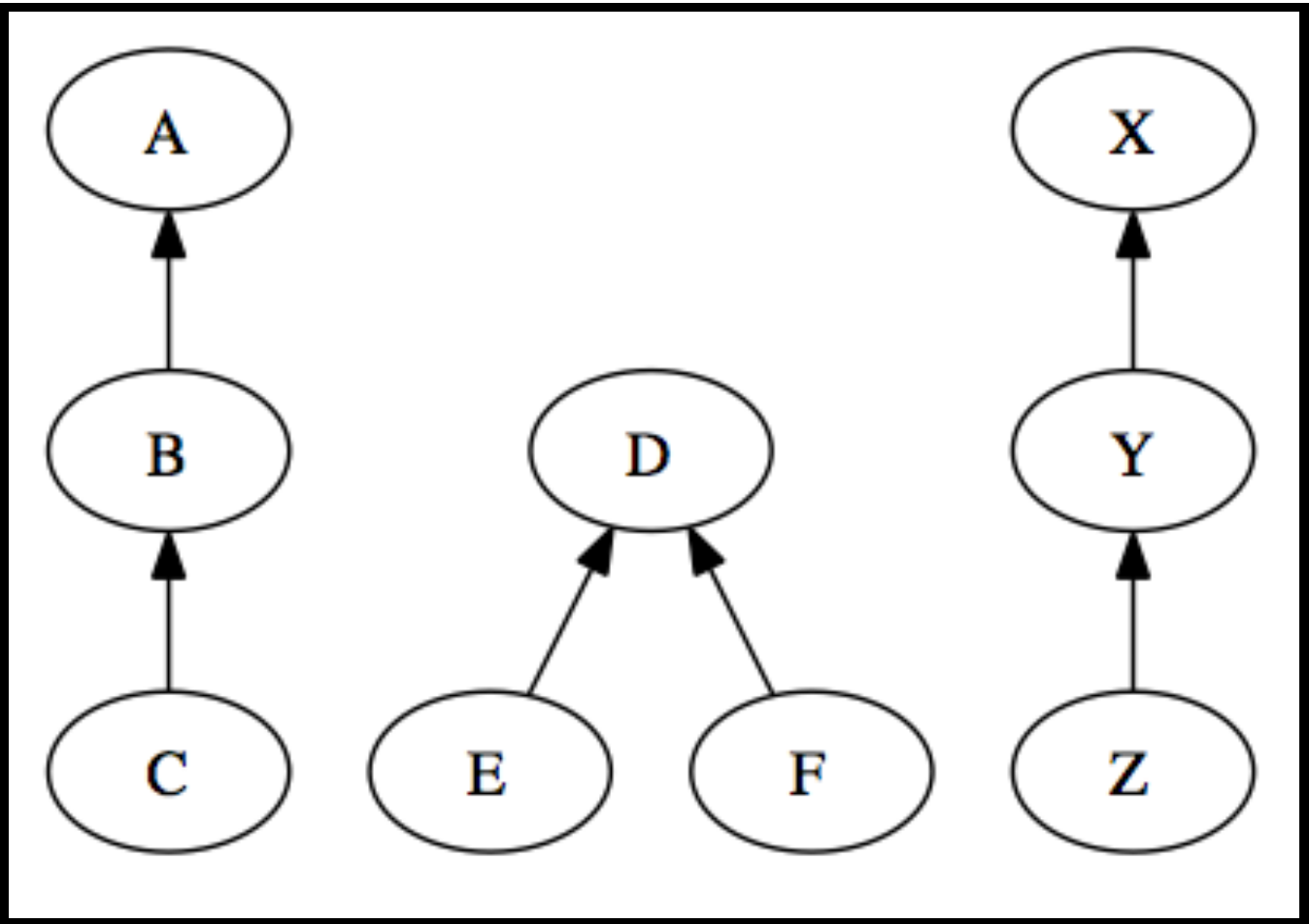
- Possible models
  - Jackson Structured Programming
  - Harel's Statecharts
  - Lattices

TDD Complexity: An Evaluation Tool?

- "less complex" (entropy measure of patterns)
- "more robust" (contains redundancy, no SPoF)
- "checkable" (model checking)
- "survivable" (contains layers to jettison to save core)
- "nimble" (trust migrates btwn component subsets)

TDDs depict patterns of trust that may repeat in different contexts within a system

Graph Patterns



"Trust Distribution Diagrams: Theory and Applications" Locasto et al. 6 December 2010

Composing Security Mechanisms

"Bickering-in-depth"

Defense-in-Depth

Doctrine of defense-in-depth says:

> "You should be able to add a new security mechanism to deepen your independent layers of security."

Bickering-in-Depth

Security software doesn't play nice, and these systems routinely interfere with each other according to our preliminary experiments [S&P mag 2009]

Key issue seems to stem from indiscriminant and conflicting modifications of kernel objects and other important resources [uninformed.org]

Preliminary Experiments

- Scenario: install multiple security software programs on a host…and observe ensuing chaos (BSOD, etc.)
- Compiling Apache: 2 minutes vs. 45 minutes
- Numerous detections of "incompatible" software during installation…but installation proceeds anyway
- CA Internet Security and Clam AV → lose network
- PC Tools Anti-Virus and Webroot → shutdown
- Anonymizer on top of the whole mess: 75% of startups freeze

Negative Outcomes

Loss in performance

Loss in protection efficacy

Potentially disastrous fusion of policy

Poor management strategies arising from dealing with above rather than actual threats

Application for TDDs?

Understand (separately) what critical data structures and measurement points two different TCBs attempt to control

Diagram these trust relationships

The exercise of composing the TDDs will help show where overlap and potential conflict exist

Programming Latent Functionality

From an attacker's viewpoint, our computing environments contain latent functionality

Formulating exploits (or ROP gadget chains) are a way of composing this latent functionality to achieve an attacker's goal (control, exfiltration, etc.)

Moving Target Defense must provide effective methods for assessing the potential for latent functionality and breaking sequences of this composed latent functionality

Conclusion

TDD: Diagram trust relationships to give us a sense of what state our system is in

Possible (high-level) application areas:

Moving Target Defense

Layering Software Security Mechanisms

Contact

email: locasto@ucalgary.ca

web: http://pages.cpsc.ucalgary.ca/~locasto/

Related Work

- Locasto, Bratus, Schulte, S&P mag Nov/Dec 2009
- Skywing, “Anti-Virus Software Gone Wrong”, What Were They Thinking? May 2006, uninformed.org
- Skape, “Annoyances Caused by Unsafe Assumptions”, What Were They Thinking? April 2005, uninformed.org
- TDDs: Theory and Applications