Attack Graphs

Outline

- Attack Graphs
- MulVal
- System-wide Info Flow

Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

Towards System-Wide, Deployment-Specific MAC Policy Generation for Proactive Integrity Mediation

Sandra Rueda, Divya Muthukumaran, Hayawardh Vijayakumar, Trent Jaeger, Swarat Chaudhuri
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

September, 2011

Talk Outline

- Current State of Security
  - Attack methods are comprehensive
  - Defenses are ad hoc
- Problem: Generate proactive defense automatically
  - What do we know how to do already?
  - Develop a solution method built on such techniques
- How will such a method impact system design/deployment?
  - Prototype to generate and test system-wide MAC policies
  - Other talks: (1) integrity measurement protocol that measures such defenses and (2) process firewall that protects system call interface

Current Attacks

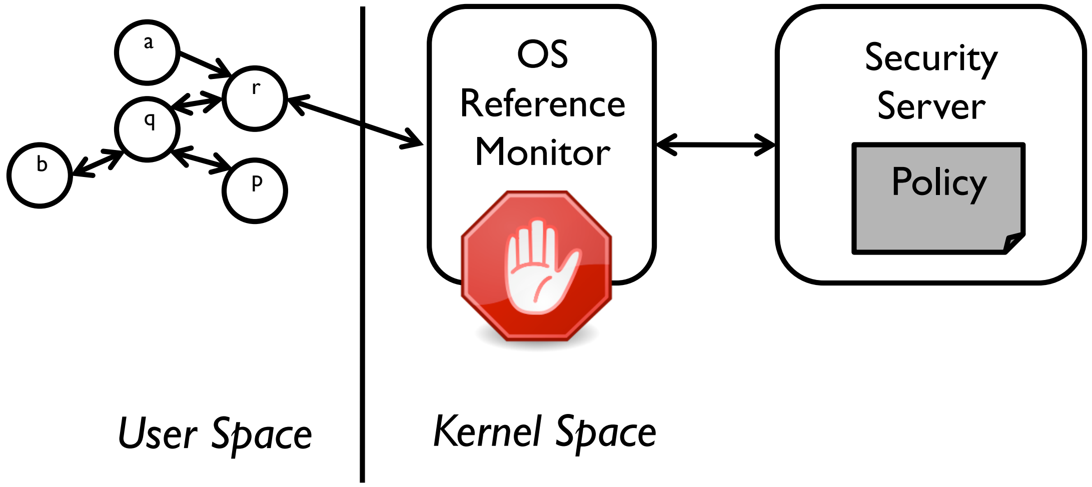
PENNSTATE 1855

- Attack unprivileged processes first
  - Then, escalate privilege incrementally via local exploits
  - Leverage (unjustified) trust between processes/hosts to propagate attacks
- Such Attack Paths are ubiquitous in current systems
  - Processes are tightly interconnected
    - Historically, all user processes have same privilege and can utilize system services
  - Any control flow vulnerability can be leveraged to run any code
    - Return-oriented programming
- Claim: Adversaries will use any undefended path

Current Defenses

- We have made progress the last 10 years or so
  - Vulnerable network services galore → hardened, privilege-separated daemons (OpenSSH)
  - Default-enabled services → hardened configurations (IIS)
  - Root system processes galore → Mandatory access control (Linux, BSD)
  - Application plug-ins in same address space → Run application code in separate processes (Chrome, OP browsers)
  - Email attachments compromise system → Prevent downloaded content from modifying system (MIC, antivirus)
  - A process in one host can easily access another host → Limit open ports (host firewalls, labeled networking)

MAC Operating Systems



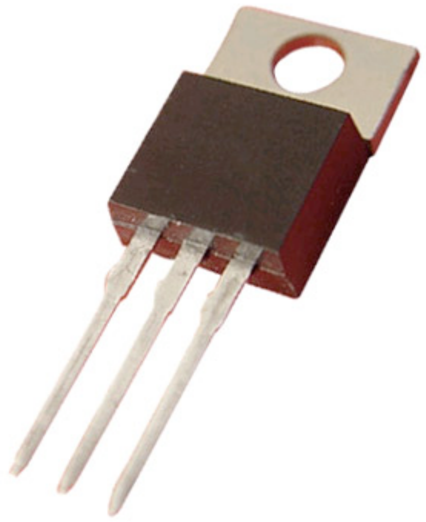
- Mandatory Access Control (MAC) operating systems
  - Define an immutable set of labels and assign them to every subject and object in the system
  - Define a fixed set of authorized operations based on the labels
- Now available in most commodity operating systems (Trusted Solaris, TrustedBSD, SELinux, AppArmor, Windows MIC*, etc)

MAC Enforcement Everywhere

- MAC enforcement in the OS alone is not enough
- Several applications are designed to serve users with multiple security requirements
  - OS cannot control what these applications do
- OS are not trusted to isolate computing (*reference monitor concept*)
  - But virtualization is (for now)
  - MAC at virtualization layer (VMM, hypervisor) can mediate system comprehensively
- OS MAC does not control operations between hosts
  - Labeled networking assigns labels to all network data (Labeled IPsec and Secmark Firewall)

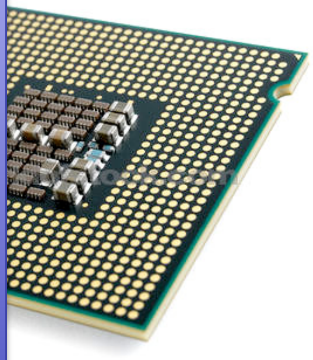
We’ve Created a Monster

- We end up with systems consisting of
  - Complex programs
  - Complex program configurations
  - Complex MAC policies
  - Systems consisting of many, independent components
- All these are built with a particular threat model in mind
  - Which is likely different than the actual deployment
- System administrators are left to fix them



Taming a Monster

- Design components to defend threats proactively
  - *Programs*: protect at some interfaces; expect high



system-wide MAC policies to defend deployments proactively. We need automated tools to generate

- Sy
  m
  th

- The two tasks are ultimately the same conceptual problem

What Do We Know How To Do?

- Compute Attack Paths (from Attack Graphs)
  - Find the sequence of steps that adversaries can take to compromise a system
- Compute Compliance
  - Find information flow and permission errors in programs and system MAC policies
- Identify Attack Surfaces
  - Find how systems and programs are accessible to adversaries
- Attack-Specific Analyses
  - E.g., input sanitization

What Do We Know How To Do?

- Compute Attack Paths (from Attack Graphs)
  - Find the sequence of steps that adversaries can take to compromise a system
- Compute Compliance
  - Find information flow and permission errors in programs and system MAC policies
- Identify Attack Surfaces
  - Find how systems and programs are accessible to adversaries
- Attack-Specific Analyses
  - E.g., input sanitization

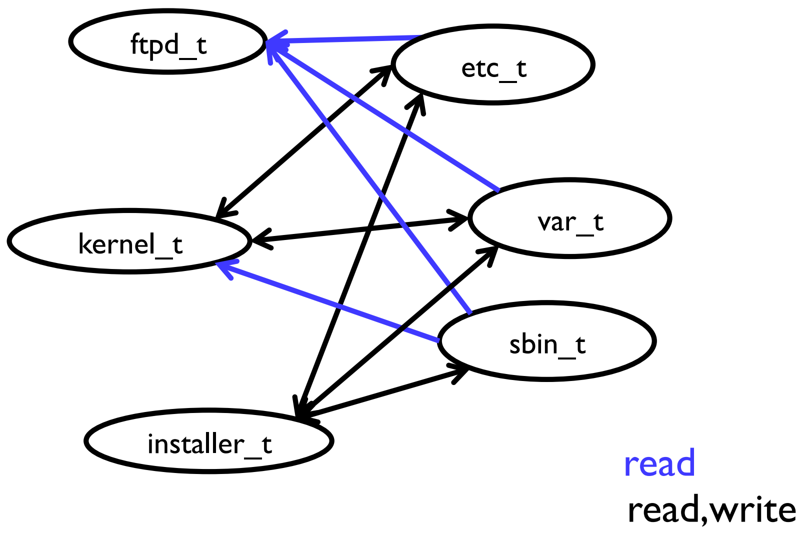
Compliance Problem

- Evaluating whether a policy permits an adversary to have unauthorized access (i.e., contains an error) is a *compliance problem*:
  - System Policy: describes a system's behavior
  - Goal Policy: describes acceptable behavior
  - Mapping function: relates elements from the system policy to elements in the goal policy
  - A compliant system policy is guaranteed to meet the requirements defined by the goal policy

Evaluating OS MAC Policy

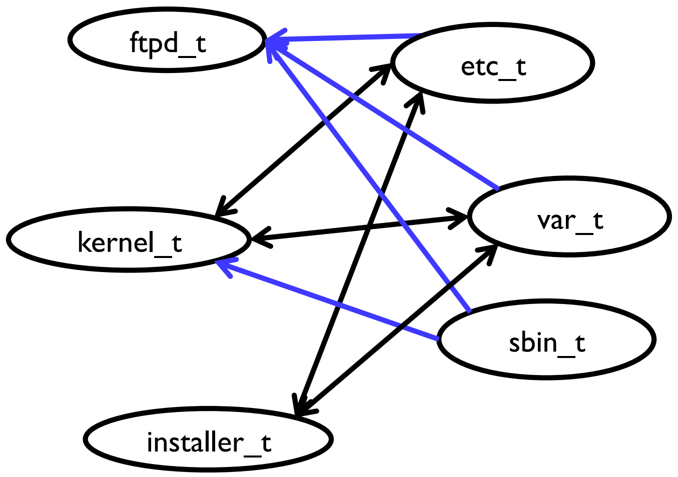
- We represent a single MAC policy with an information flow graph
  - Used in analyses for SELinux by Tresys, Stoller, Li, Jaeger, etc.

	etc_t	var_t	sbin_t
installer_t	read,write	read,write	read,write
kernel_t	read,write	read,write	read
ftpd_t	read	read	read



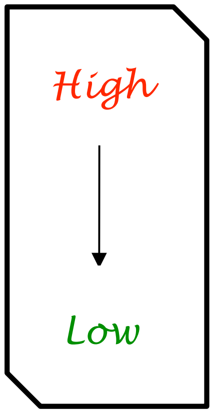
Compliance Problem

- The policy compliance problem for a single policy is set up as follows:
    - *System policy* – The policy that we are analyzing is represented as a graph



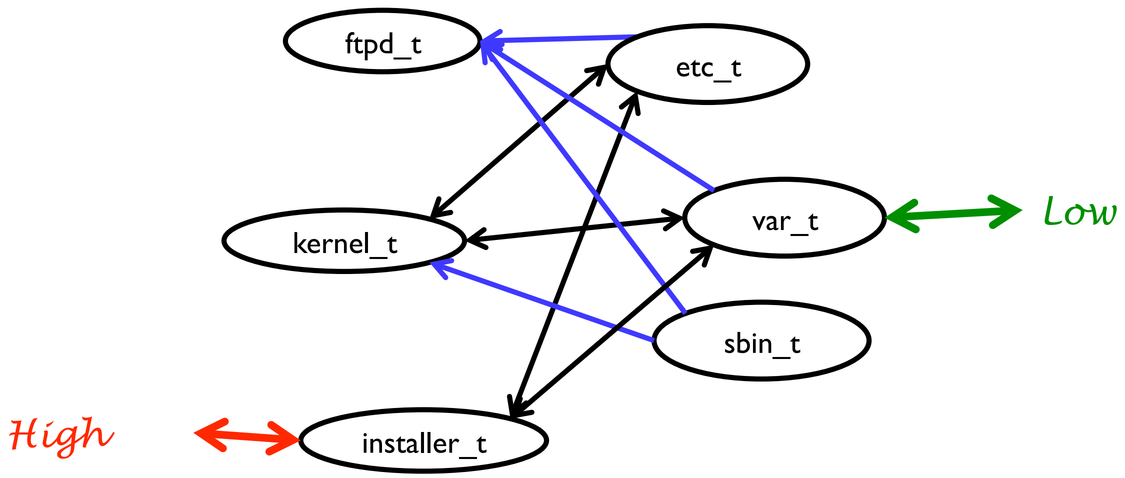
Compliance Problem

- The policy compliance problem for a single policy is set up as follows:
  - *System policy* – The policy that we are analyzing is represented as a graph
  - *Goal* – The security goal is a lattice that defines integrity levels and rules that guarantee the integrity of the system



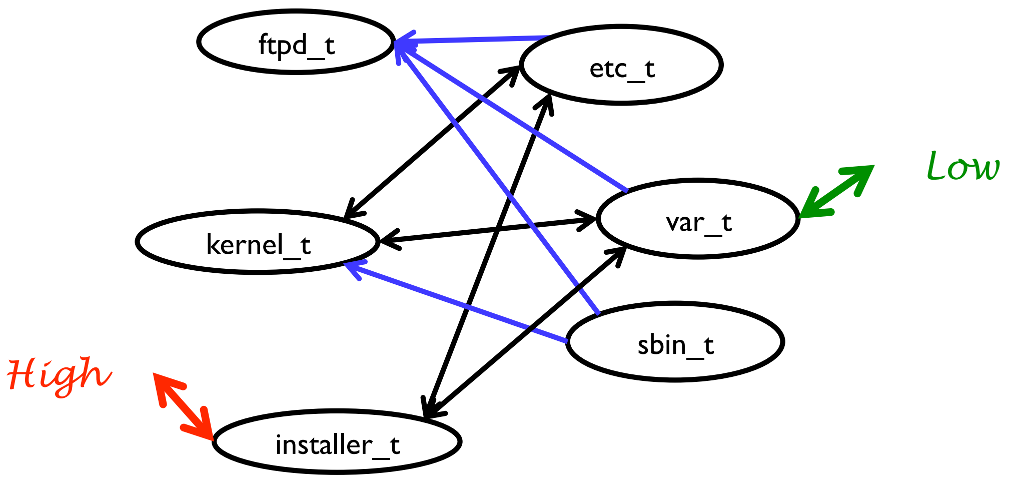
Compliance Problem

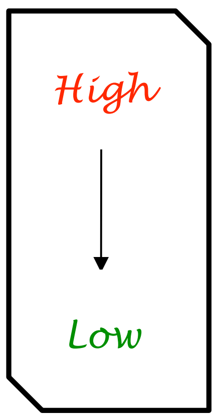
- The policy compliance problem for a single policy is set up as follows:
  - *System policy* – The policy that we are analyzing is represented as a graph
  - *Goal* – The security goal is a lattice that defines integrity levels and rules that guarantee the integrity of the system
  - *Mapping* - Assigns integrity levels to policy labels



Compliance Problem

- The policy compliance problem for a single policy is set up as follows:
  - *System policy* – The policy that we are analyzing is represented as a graph
  - *Goal* – The security goal is a lattice that defines integrity levels and rules that guarantee the integrity of the system
  - *Mapping* - Assigns integrity levels to policy labels



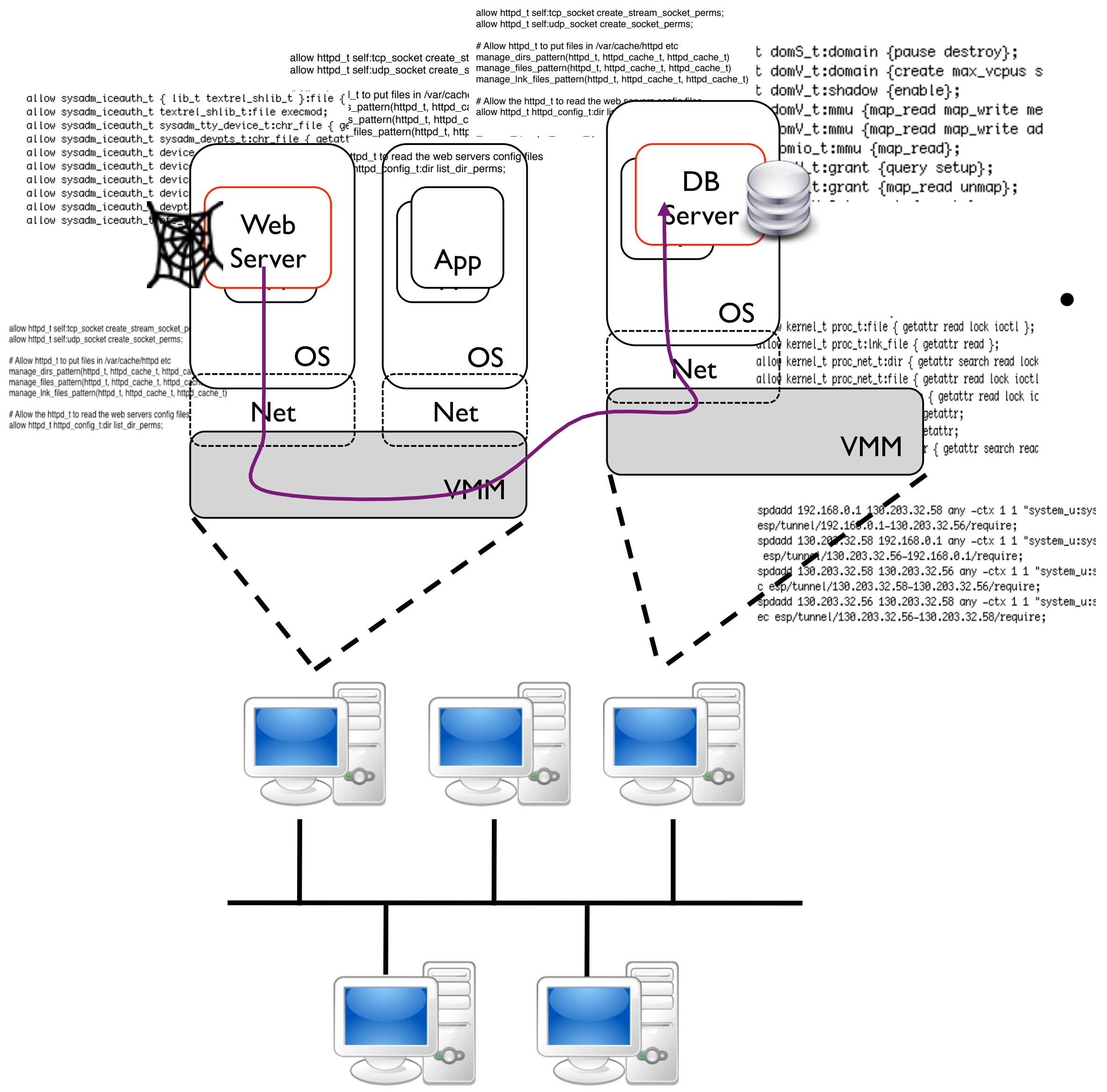


Do all flows meet the requirements defined by the goal ?

Other Compliance Problems

- Information flow compliance in programs
  - Data flow is determined by program data flows – security-typed languages, such as Jif, Sif, SELinks, FlowCaml
- Goal policy is not a lattice
  - Illegal reachability: no path from u $\rightarrow_G$ v
  - Illegal sets of permissions: annotate edges with permissions
- Goals as obligations
  - The presence of a node, edge, or path is required
  - These are functional constraints, rather than security

Compliance Challenges



```
allow httpd_t self:tcp_socket create_stream_socket_perms;
allow httpd_t self:udp_socket create_socket_perms;

# Allow httpd_t to put files in /var/cache/httpd etc
manage_dirs_pattern(httpd_t, httpd_cache_t, httpd_cache_t)
manage_files_pattern(httpd_t, httpd_cache_t, httpd_cache_t)
manage_lnk_files_pattern(httpd_t, httpd_cache_t, httpd_cache_t)

# Allow the httpd_t to read the web servers config files
allow httpd_t httpd_config_t:dir list_dir_perms;
```

```
allow sysadm_iceauth_t { lib_t textrel_shlib_t }:file {
allow sysadm_iceauth_t textrel_shlib_t:file execmod;
allow sysadm_iceauth_t sysadm_tty_device_t:chr_file { ge
allow sysadm_iceauth_t sysadm_devpts_t:chr_file { getatt
allow sysadm_iceauth_t device
allow sysadm_iceauth_t devic
allow sysadm_iceauth_t devic
allow sysadm_iceauth_t devic
allow sysadm_iceauth_  devpt
allow sysadm_iceauth_t
```

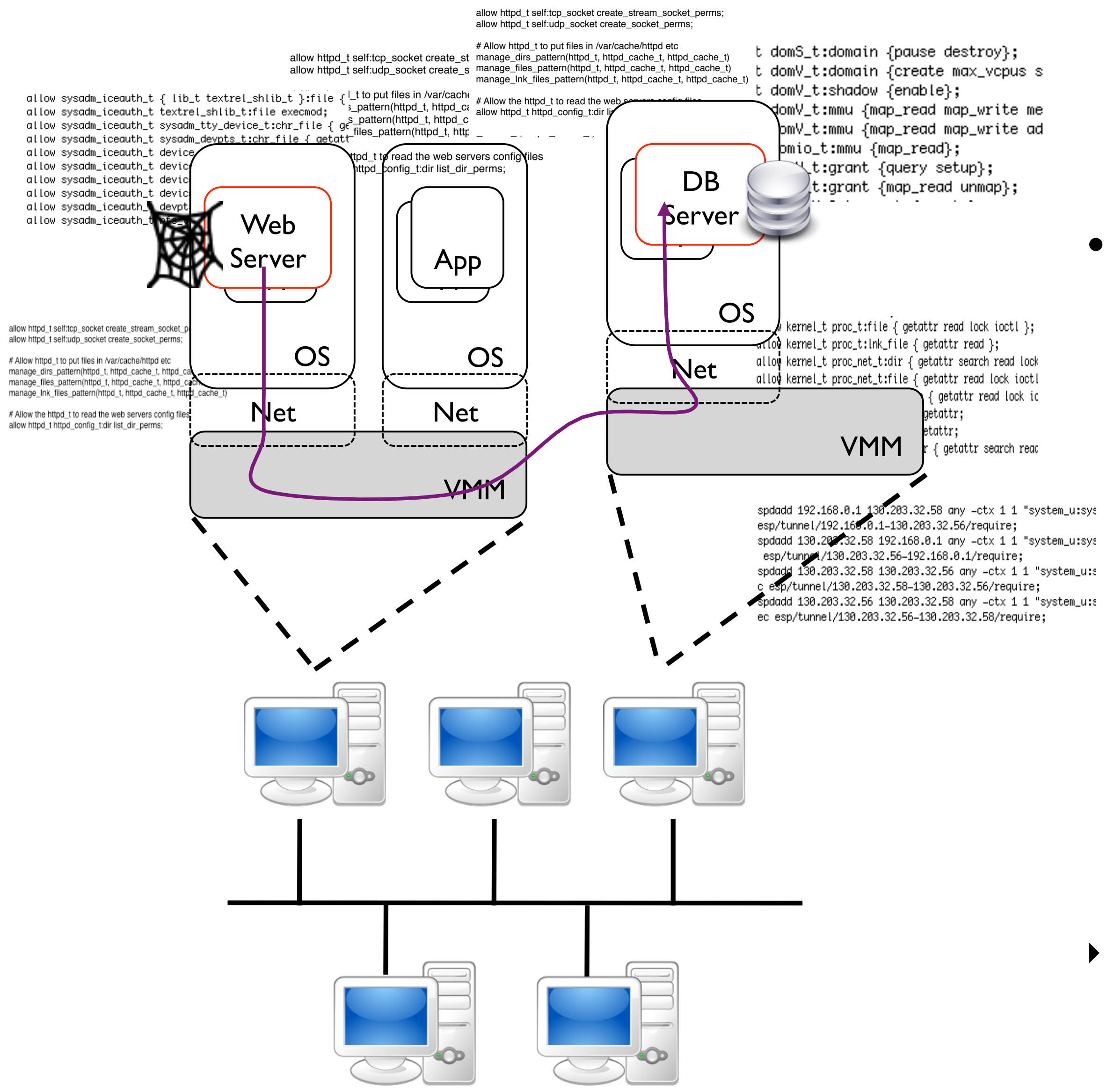
```
t domS_t:domain {pause destroy};
t domV_t:domain {create max_vcpus s
t domV_t:shadow {enable};
 domV_t:mmu {map_read map_write me
 omV_t:mmu {map_read map_write ad
 omio_t:mmu {map_read};
   t:grant {query setup};
   t:grant {map_read unmap};
```

```
allow kernel_t proc_t:file { getattr read lock ioctl };
allow kernel_t proc_t:lnk_file { getattr read };
allow kernel_t proc_net_t:dir { getattr search read lock
allow kernel_t proc_net_t:file { getattr read lock ioctl
                               { getattr read lock ic
                               getattr;
                               etattr;
                              r { getattr search reac
```

```
spdadd 192.168.0.1 130.203.32.58 any -ctx 1 1 "system_u:sys
esp/tunnel/192.168.0.1-130.203.32.56/require;
spdadd 130.203.32.58 192.168.0.1 any -ctx 1 1 "system_u:sys
 esp/tunnel/130.203.32.56-192.168.0.1/require;
spdadd 130.203.32.58 130.203.32.56 any -ctx 1 1 "system_u:s
c esp/tunnel/130.203.32.58-130.203.32.56/require;
spdadd 130.203.32.56 130.203.32.58 any -ctx 1 1 "system_u:s
ec esp/tunnel/130.203.32.56-130.203.32.58/require;
```

- Construct Data Flow Graph
  - Multiple independently-developed policies
    - Different policy languages
    - Different policy concepts
    - Policies may interact in multiple ways

Compliance Challenges



```
allow httpd_t self:tcp_socket create_stream_socket_perms;
allow httpd_t self:udp_socket create_socket_perms;

# Allow httpd_t to put files in /var/cache/httpd etc
manage_dirs_pattern(httpd_t, httpd_cache_t, httpd_cache_t)
manage_files_pattern(httpd_t, httpd_cache_t, httpd_cache_t)
manage_lnk_files_pattern(httpd_t, httpd_cache_t, httpd_cache_t)

# Allow the httpd_t to read the web servers config files
allow httpd_t httpd_config_t:dir list_dir_perms;
```

```
allow sysadm_iceauth_t { lib_t textrel_shlib_t }:file {
allow sysadm_iceauth_t textrel_shlib_t:file execmod;
allow sysadm_iceauth_t sysadm_tty_device_t:chr_file {
allow sysadm_iceauth_t sysadm_devpts_t:chr_file { getatt
```

```
t domS_t:domain {pause destroy};
t domV_t:domain {create max_vcpus s
t domV_t:shadow {enable};
domV_t:mmu {map_read map_write me
omV_t:mmu {map_read map_write ad
omio_t:mmu {map_read};
t:grant {query setup};
t:grant {map_read unmap};
```

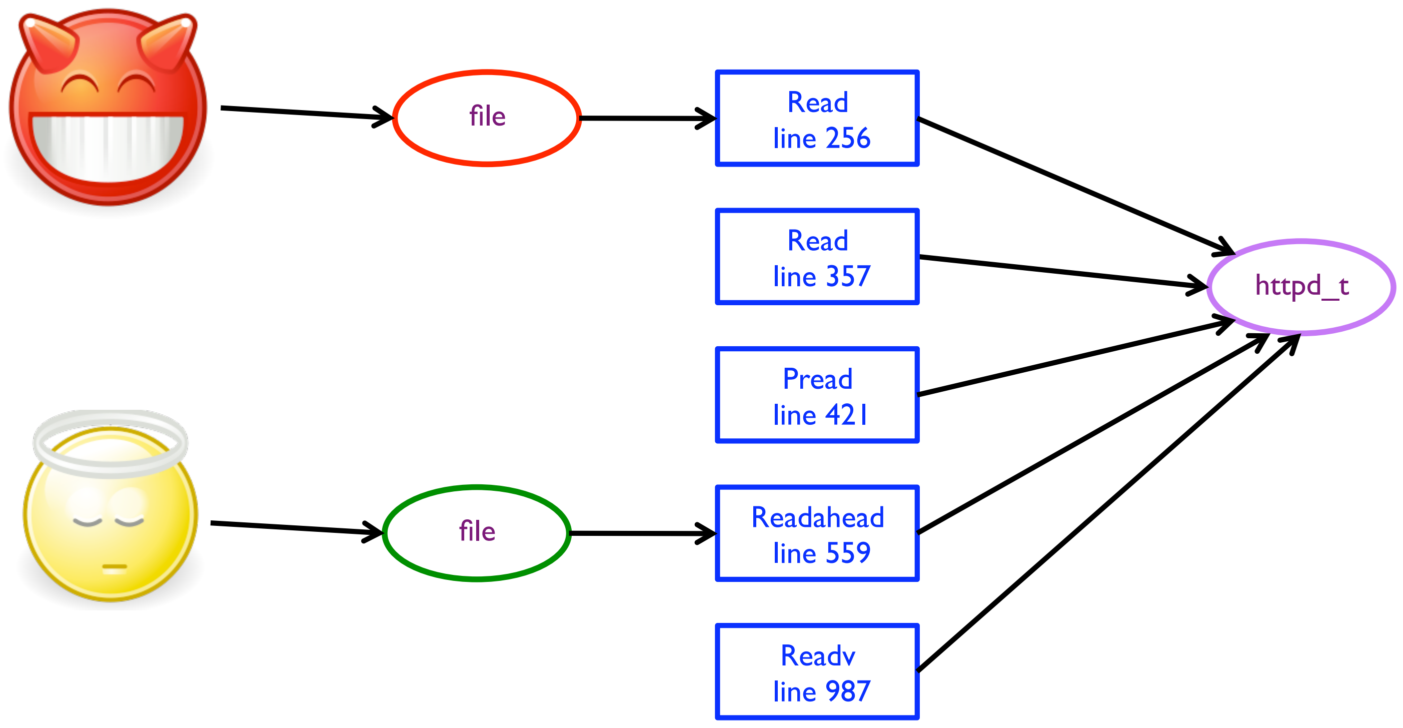
```
kernel_t proc_t:file { getattr read lock ioctl };
allow kernel_t proc_t:lnk_file { getattr read };
allow kernel_t proc_net_t:dir { getattr search read lock
allow kernel_t proc_net_t:file { getattr read lock ioctl
```

```
spdadd 192.168.0.1 130.203.32.58 any -ctx 1 1 "system_u:sys
esp/tunnel/192.168.0.1-130.203.32.56/require;
spdadd 130.203.32.58 192.168.0.1 any -ctx 1 1 "system_u:sys
 esp/tunnel/130.203.32.56-192.168.0.1/require;
spdadd 130.203.32.58 130.203.32.56 any -ctx 1 1 "system_u:s
c esp/tunnel/130.203.32.58-130.203.32.56/require;
spdadd 130.203.32.56 130.203.32.58 any -ctx 1 1 "system_u:s
ec esp/tunnel/130.203.32.56-130.203.32.58/require;
```

- Goals and mappings are manually-specified
  - Lattice policy is not specified
  - Mapping is not specified
  - Our experience indicates that the size of the goal increases with the size of the distributed system
  - Manual specification is prone to error
- Then, how do you fix errors?

Attack Surfaces

- Where are 'vulnerabilities'?
  - A flaw, accessible to an adversary, with an ability to compromise that flaw
- Program input interfaces (e.g., read system calls) that are accessible to adversaries [Howard of Microsoft]



Attack Surface Challenges

- How to identify attack surfaces of individual programs
  - All interfaces have access to all process permissions
  - Some interfaces are obvious (network), but others are questionable
- Researchers have used value of data behind interface
  - But this does not say anything about accessibility
- Difficult to identify attack surfaces from the program alone
  - Depends on its deployment
- Goal: Use MAC policies to identify attack surfaces – defenses must be placed there

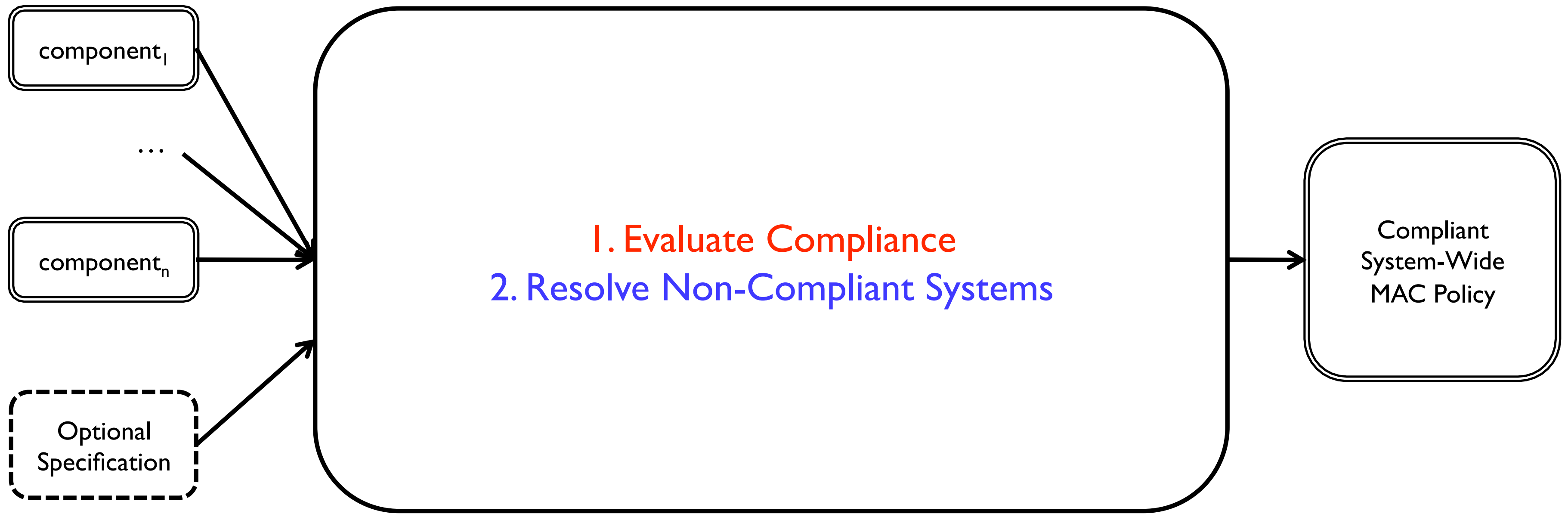
Goal Statement

Generate a compliant, system-wide MAC policy that minimizes the cost of defense (attack surfaces) mostly-automatically for distributed systems consisting of multiple, independent MAC-enforcing components.

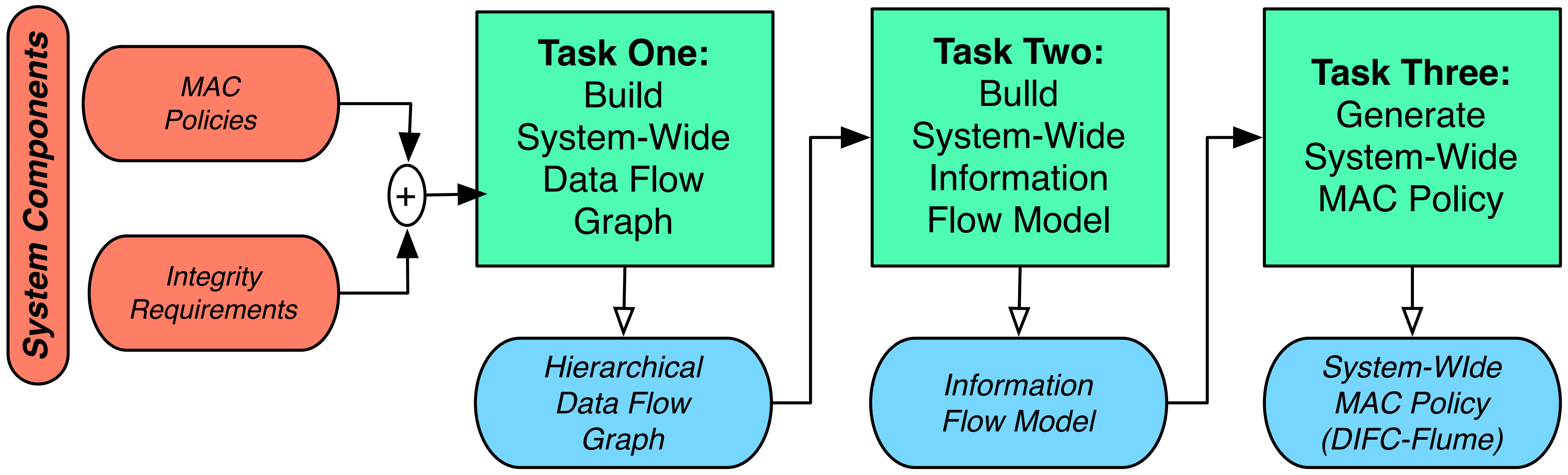
Ideally, Approximately

- Solve as an *optimization problem*
- Find the minimum cost solution that satisfies a goal policy consisting of security and functional constraints (likely, an NP-complete problem)
  - Compliance was defined in terms of security policies only (lattice)
  - Also, need to prevent the removal of necessary function
- Could apply SMT solver or greedy algorithm to solve such a problem
- **Barrier:** While we think that we can predict a meaningful, conservative set of security constraints, little is known about what function is permissible
- **Instead**: *For a particular functional specification, find the minimum cost solution that complies with a goal policy (security only)*

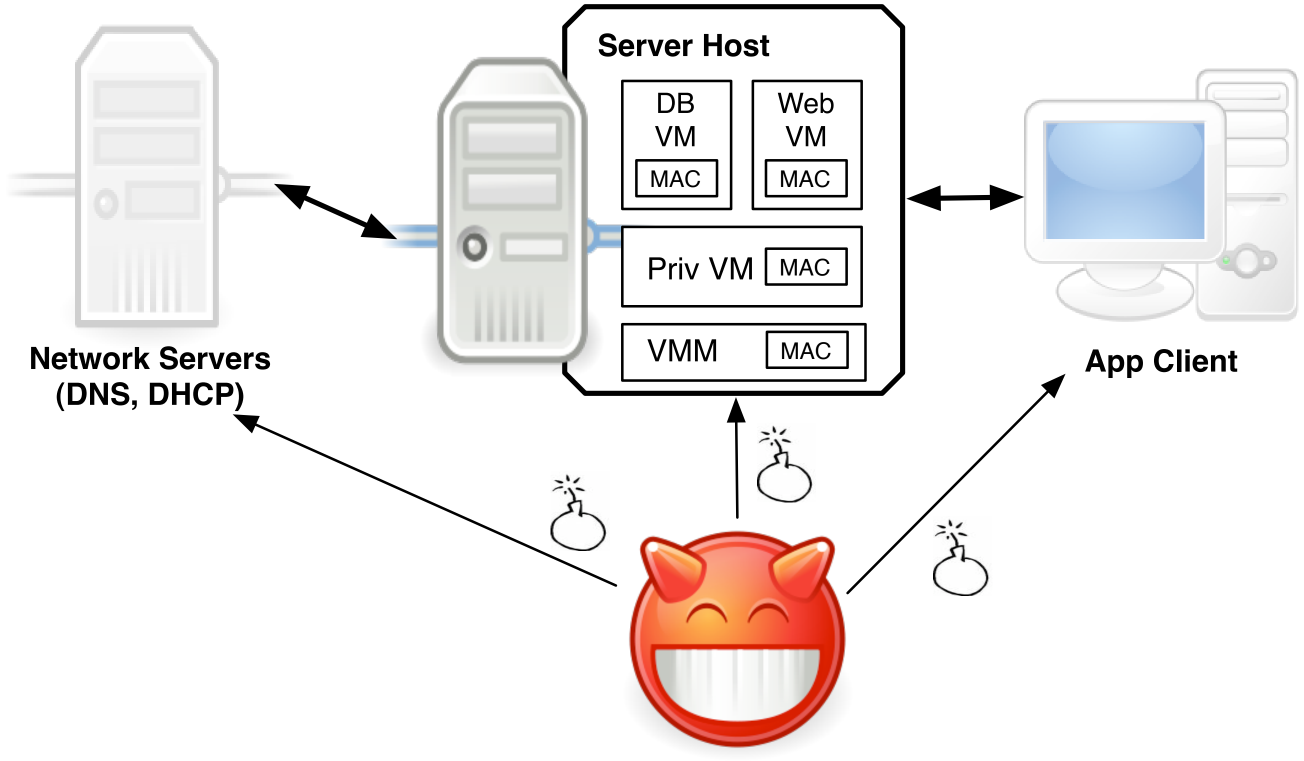
Distributed Compliance Evaluation



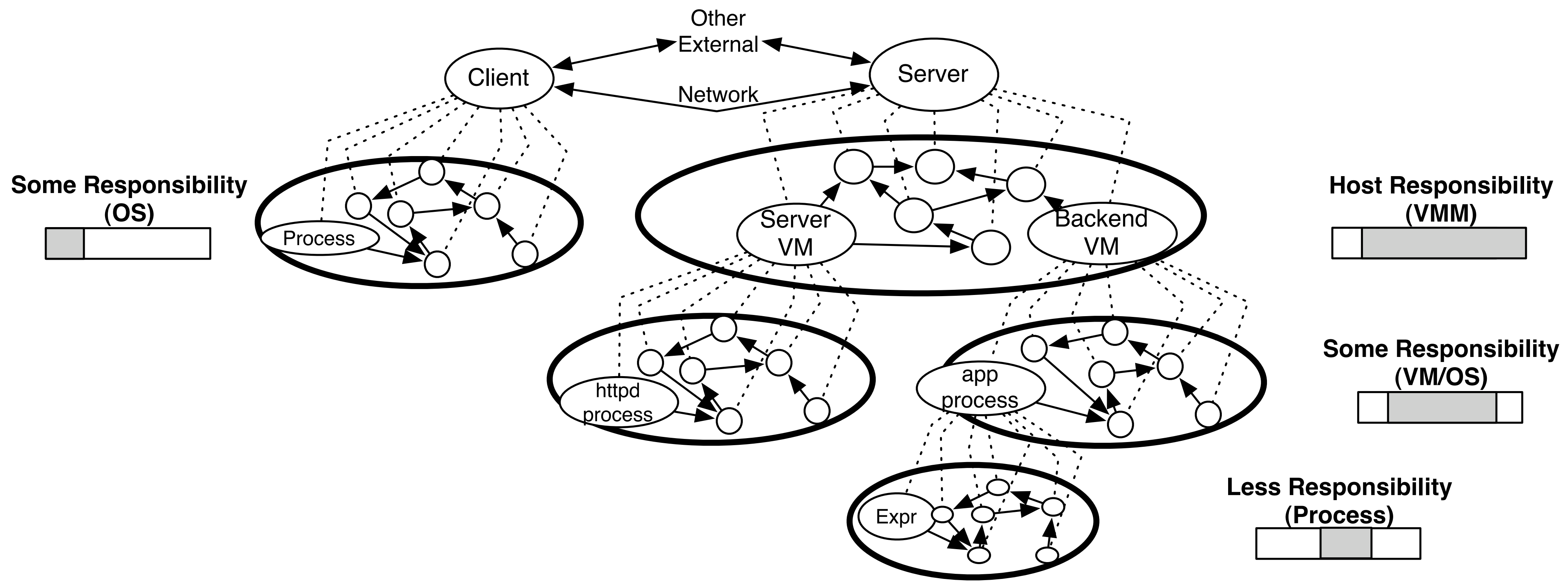
Distributed Compliance Evaluation


System Components
MAC Policies
Integrity Requirements
+
Task One: Build System-Wide Data Flow Graph
Hierarchical Data Flow Graph
Task Two: Bulld System-Wide Information Flow Model
Information Flow Model
Task Three: Generate System-Wide MAC Policy
System-WIde MAC Policy (DIFC-Flume)

Example System

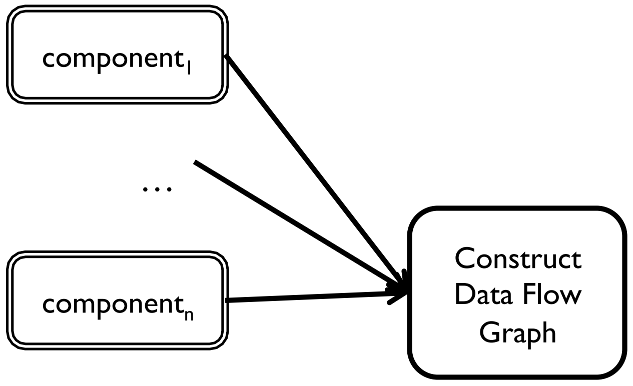


System Data Flows?



1. Construct Data Flow Graph

- We find that MAC-enforcing components in distributed systems are
  - *Encapsulated*: data flows are mediated by MAC policy
  - *Hierarchical*: each has at most one parent
  - *Reusable*: same flows may appear multiple times
- We use an hierarchical graph data structure defined by Alur *et al.* [Alur2004] to concisely represent data flows



Data Flow Graph Definition

A *hierarchical state machine* K is a tuple $(K_1, ...K_n)$ of *modules*, where each module K_i has the following components:

- A finite set V_i of *nodes*.
- A finite set B_i of *boxes*.
- A subset I_i of V_i, called *entry nodes*.
- A subset O_i of V_i, called *exit nodes*.
- An *indexing function* $Y_i : B_i \rightarrow \{i+1, ..., n\}$ that maps each box of the i-th module to an index greater than i. That is, if $Y_i(b) = j$ for box b of module K_i, then b can be viewed as a reference to the definition of module K_j.
- If b is a box of the module K_i with $j = Y_i(b)$, then pairs of the form (b, u) with $u \in I_j$ are the *calls* of K_i and pairs of the form (b, v) with $v \in O_j$ are the *returns* of K_i.
- An *edge relation* E_i consisting of pairs (u, v), where the source u is either a node or a return of K_i and v is either a node or a call of K_i.

Policies to Data Flow Graph

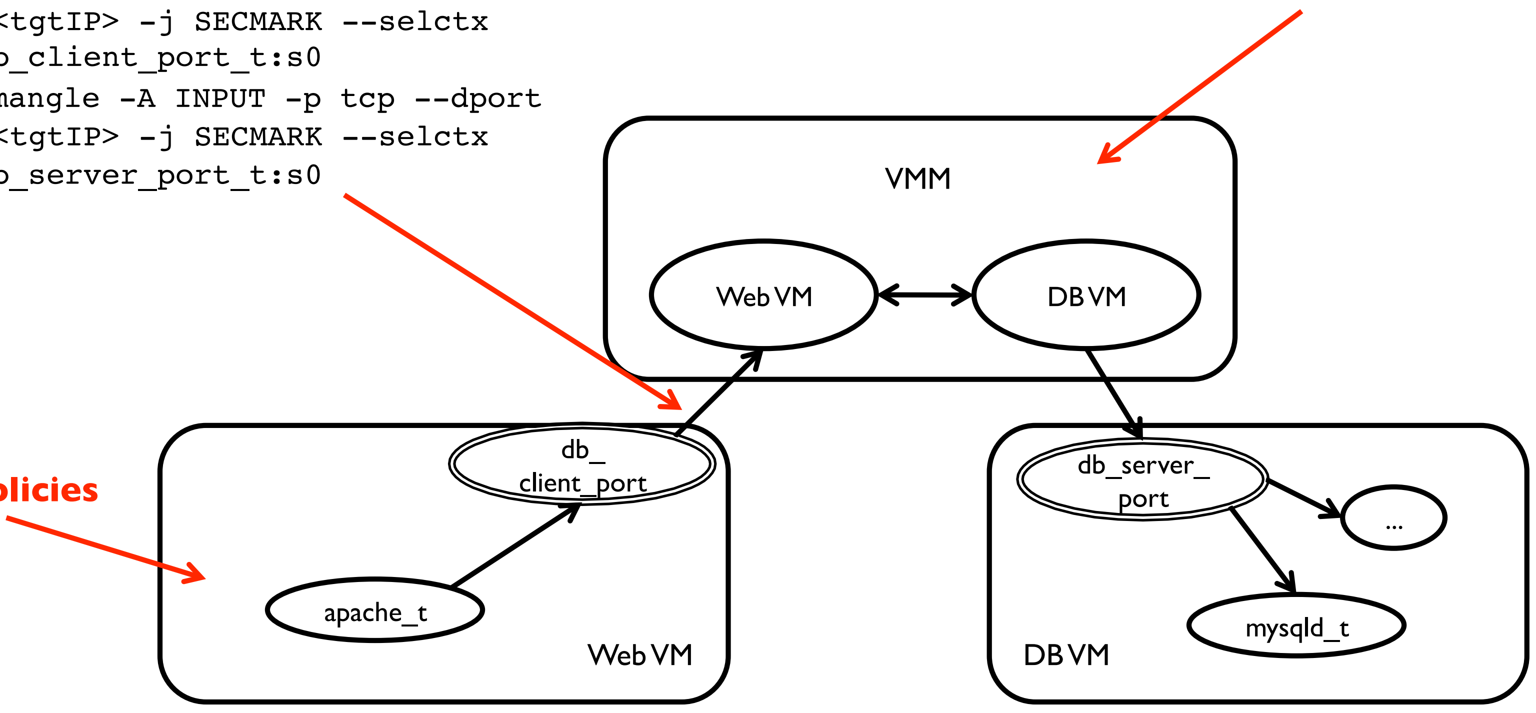
Secmark Host Firewall Policies:

Web VM: `iptables -t mangle -A OUTPUT -p tcp --dport 3306 -s <srcIP> -d <tgtIP> -j SECMARK --selctx system_u:object_r:db_client_port_t:s0`

DB VM: `iptables -t mangle -A INPUT -p tcp --dport 3306 -s <srcIP> -d <tgtIP> -j SECMARK --selctx system_u:object_r:db_server_port_t:s0`

Xen Security Modules Flask/sHype Policies

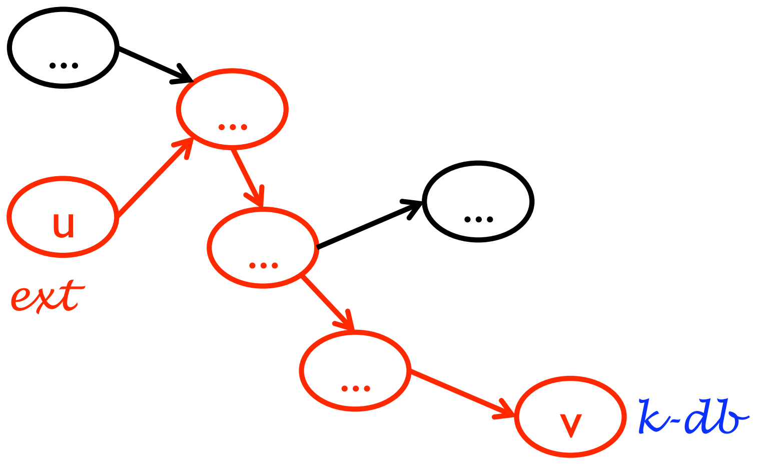
SELinux OS MAC Policies

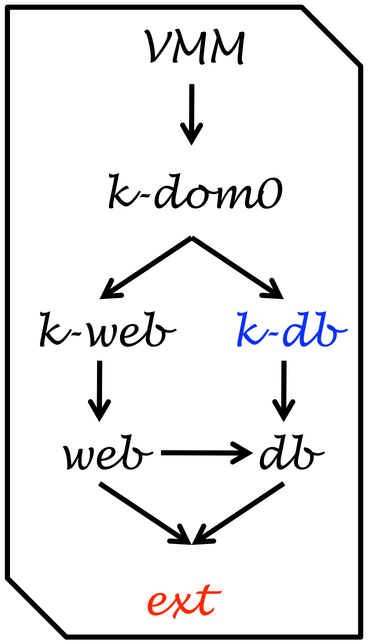


Information Flow Model

System policy:	$G = (V, E)$
Goal:	$\mathcal{L} = (L, \preceq)$
Mapping function:	$map : V' \rightarrow L,\ \ V' \subseteq V$
Compliance:	$\forall u, v \in V.\ (u \hookrightarrow_G v) \rightarrow (map(u) \hookrightarrow_{\mathcal{L}} map(v))$

Information Flow Errors: $\exists u, v \in V.\ u \hookrightarrow_G v \wedge map(u) \not\hookrightarrow_{\mathcal{L}} map(v)$



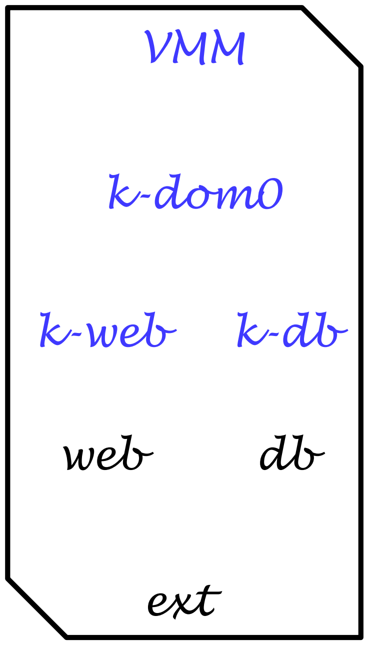


2. Build the Info Flow Model

- Problem: No explicit security constraint information
- Problem: Distributed systems are too large to annotate manually
- Insight: It's all around

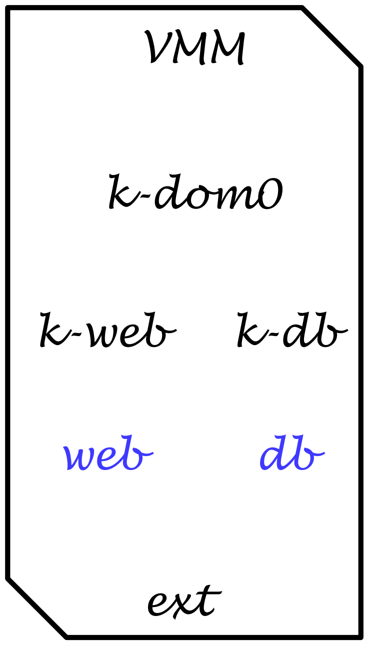
Identify Integrity Levels

- Problem: No explicit security constraint information
- Problem: Distributed systems are too large to annotate manually
- Insight: It's all around
- (1) Trusted Computing Bases: (OS) modify kernel objects and (VMM) modify VMM objects
- (2) Application Data: Deploy VMs with a particular application in mind
- (3) Apps trust TCB
- (4) Some Apps Depend on Others: E.g., Web applications depend on DB



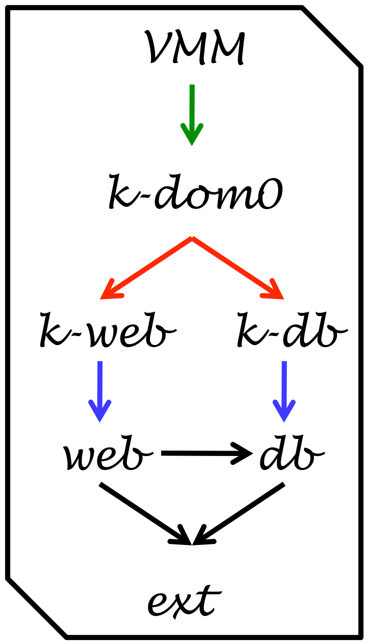
Identify Integrity Levels

- Problem: No explicit security constraint information
- Problem: Distributed systems are too large to annotate manually
- Insight: It's all around
- (1) Trusted Computing Bases: (OS) modify kernel objects and (VMM) modify VMM objects
- (2) Application Data: Deploy VMs with a particular application in mind
- (3) Apps trust TCB
- (4) Some Apps Depend on Others: E.g., Web applications depend on DB



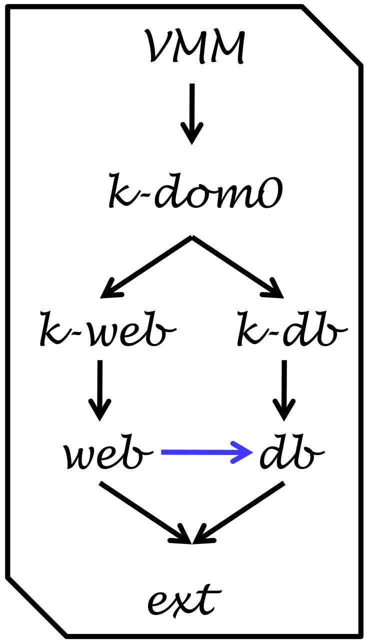
Relate Integrity Levels

- Problem: No explicit security constraint information
- Problem: Distributed systems are too large to annotate manually
- Insight: It's all around
- (1) Trusted Computing Bases: (OS) modify kernel objects and (VMM) modify VMM objects
- (2) Application Data: Deploy VMs with a particular application in mind
- (3) Apps trust TCB
- (4) Some Apps Depend on Others: E.g., Web applications depend on DB



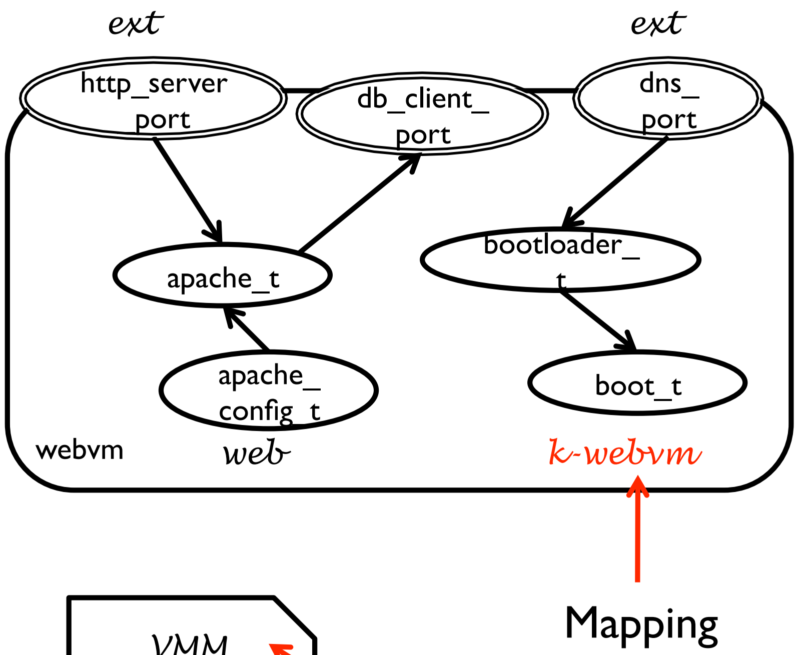
Relate Integrity Levels

- Problem: No explicit security constraint information
- Problem: Distributed systems are too large to annotate manually
- Insight: It's all around
- (1) Trusted Computing Bases: (OS) modify kernel objects and (VMM) modify VMM objects
- (2) Application Data: Deploy VMs with a particular application in mind
- (3) Apps trust TCB
- (4) Some Apps Depend on Others: E.g., Web applications depend on DB


$$\begin{tikzcd} & VMM \arrow[d] & \\ & k\text{-}dom0 \arrow[dl] \arrow[dr] & \\ k\text{-}web \arrow[d] & & k\text{-}db \arrow[d] \\ web \arrow[rr] \arrow[dr] & & db \arrow[dl] \\ & ext & \end{tikzcd}$$

Expert Knowledge

- Level/Mapping inference
- Lattice inference



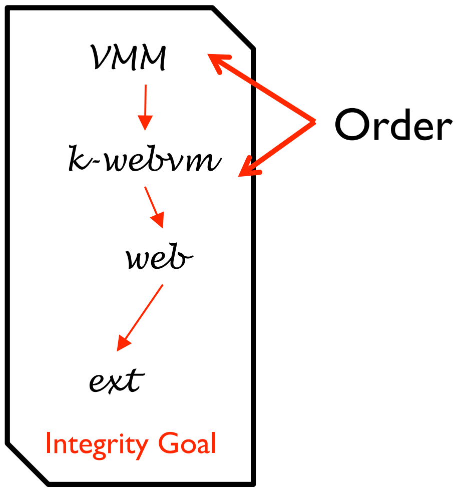
Examples

Level/Mapping inference:
- Resources to protect :

```
map(VM, boot_t,ID),ID='k-'+VM
map(webvm,boot_t,k-webvm)
```

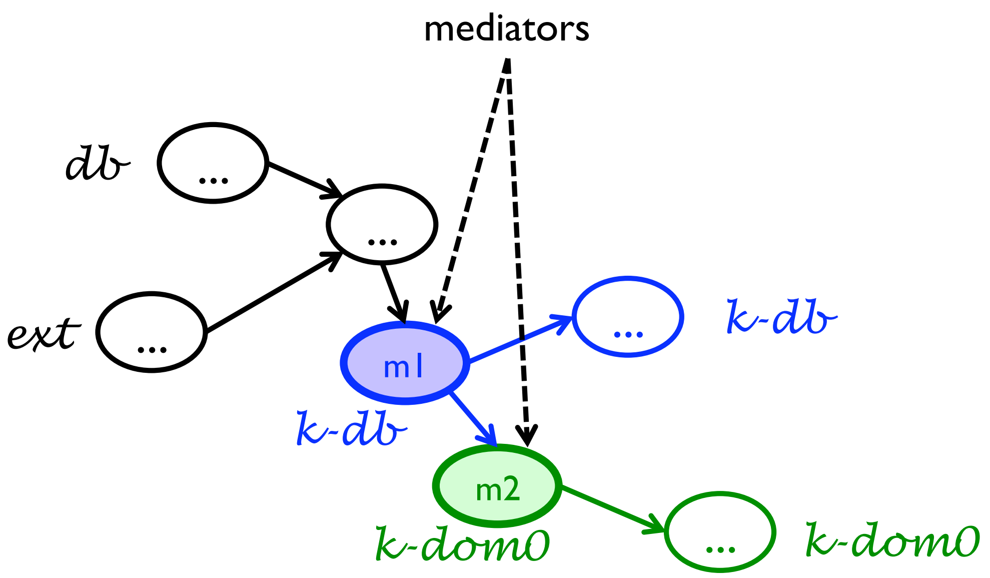
Lattice inference:
- Order:VMs depend on the underlying VMM

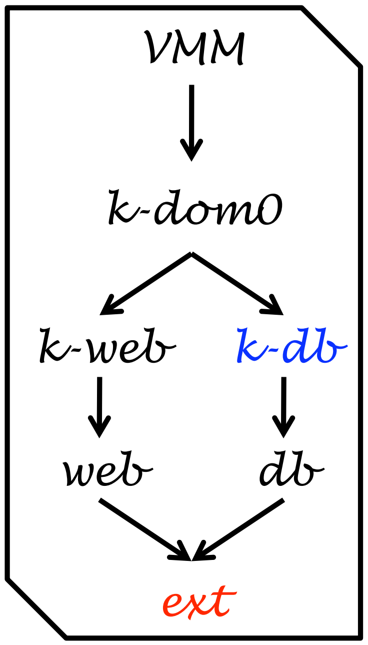
```
flow(H,L):- component(L,H,_)
flow(VMM,k-webvm)
```



Resolve by Mediation

- We resolve a information flow errors by suggesting mediators
  - A mediator is a program expected to implement procedures to sanitize inputs so the integrity of the data raises (endorsement)



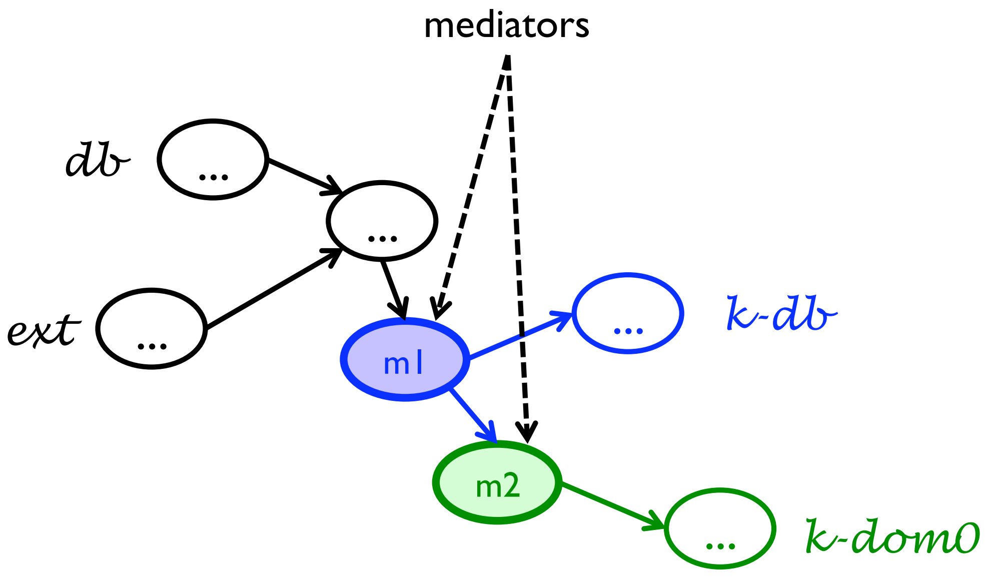


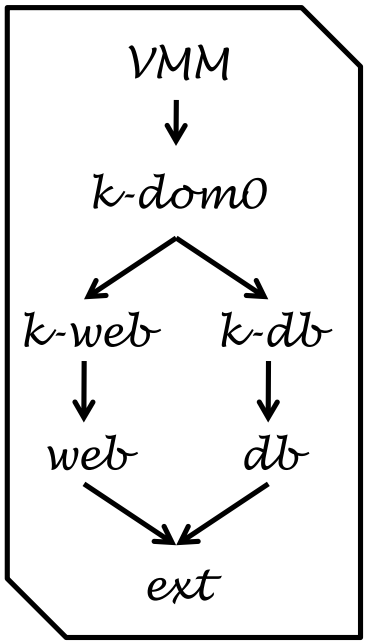
3a. Place Mediators

- [McCamant and Ernst PLDI 2008]: Solve max flow problem to quantify information leakage. Inspired us to look into min-cut.
- View *information flow constraints as a graph* between incomparable security labels.
- A *cut of the graph* should correspond to places in the code where mediation statements should be placed such that all *information flow errors* are resolved.

Multicut Problem

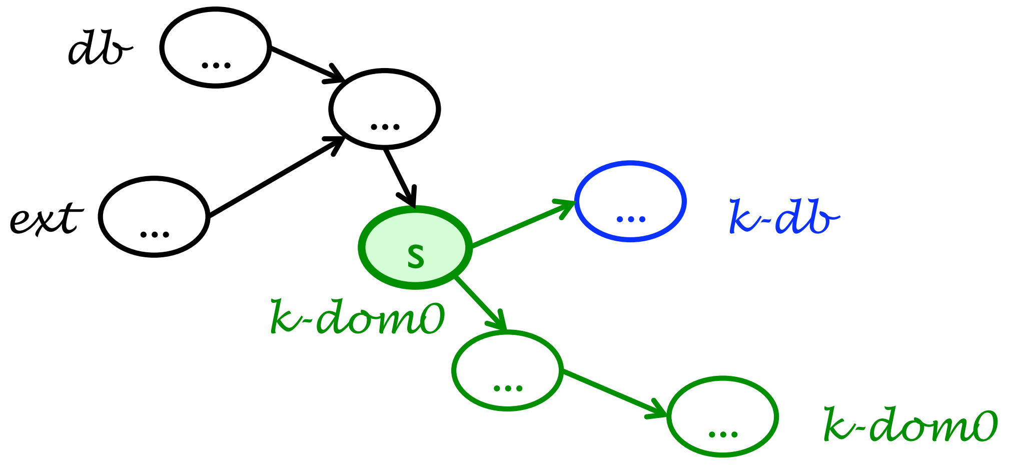
- Finding a minimum cost set of mediation points for an arbitrary lattice is a multicut problem for directed graphs which is NP-hard

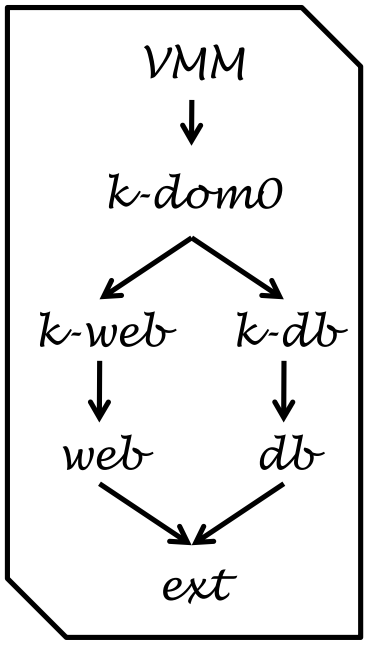




Mediation Dominance

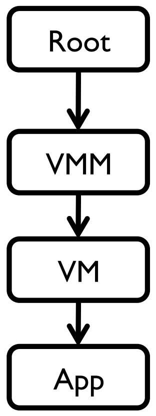
- Greedy approach: cut per sink and unions solutions
- We take advantage of the lattice ordering
  - if $l_i \preceq l_j$ then solving a cut problem in graph G for label l_i solves any overlapping cut problem for a label l_j

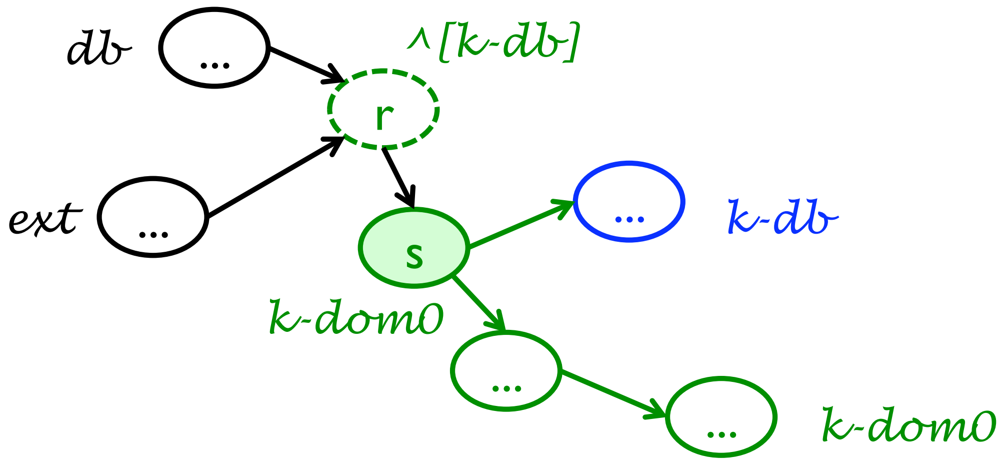


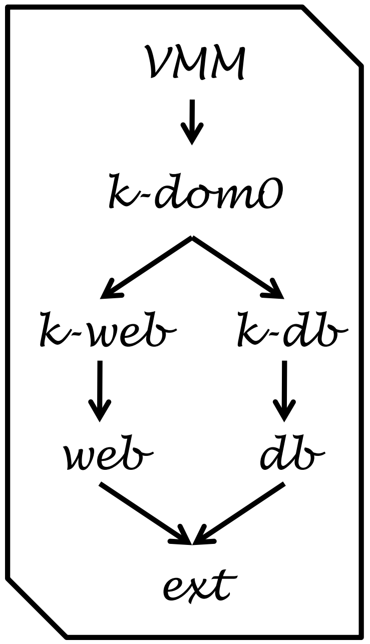


Mediation Constraints

- Not all nodes can mediate for all sinks
  - We compute mediation constraints based on the hierarchical structure of the components

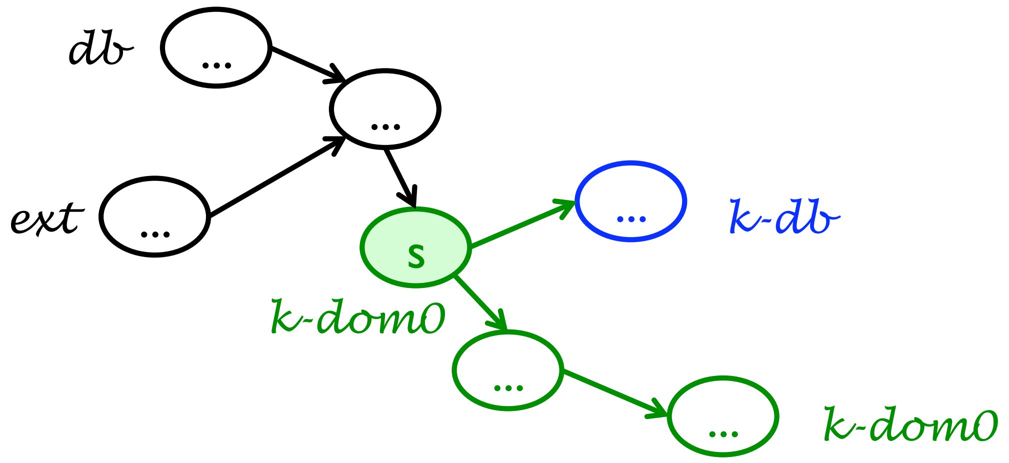






Mediation Resolution

- Result
  - Set of mediators that resolve all information flow errors



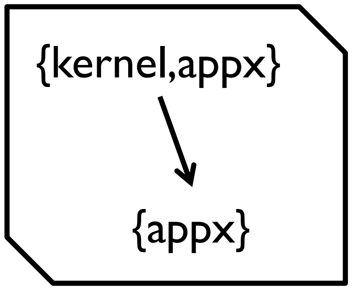
cutset(*k-dom0*) = {s}
cutset(*k-db*)={}

Mediators to System-Wide Policy

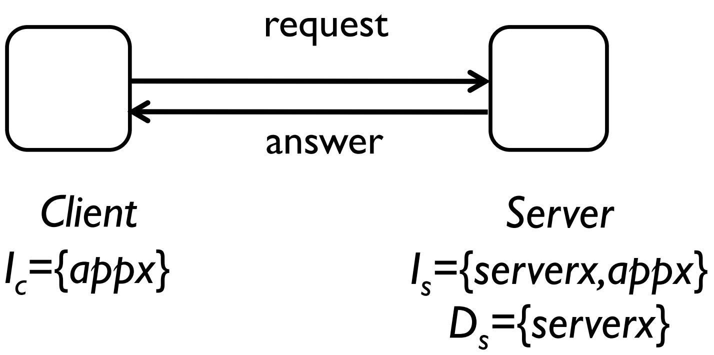
- After resolution we have:
  - An integrity lattice and the corresponding mapping to MAC policies
  - A set of mediators
- Since we do not have functional requirements we do not modify the original policies (future work)
  - Use subset of operations (see Evaluation)
- We generate a system-wide MAC policy capable of expressing mediation
  - Recent “practical integrity” models – We chose the Flume policy
  - We automate generation of Flume integrity policy for a deployment

Flume

- Lattice-based integrity policy
  - Label: set of integrity tags, L = {kernel,appx}
  - Ordered under the subset relation



- Each process, p, has an integrity label I_p
  - For every id t in I_p, p has endorsed every input to satisfy t
  - Communication: sender's integrity must be higher than receiver's integrity
  - Some processes have capabilities so they can change their labels (add/remove tags)



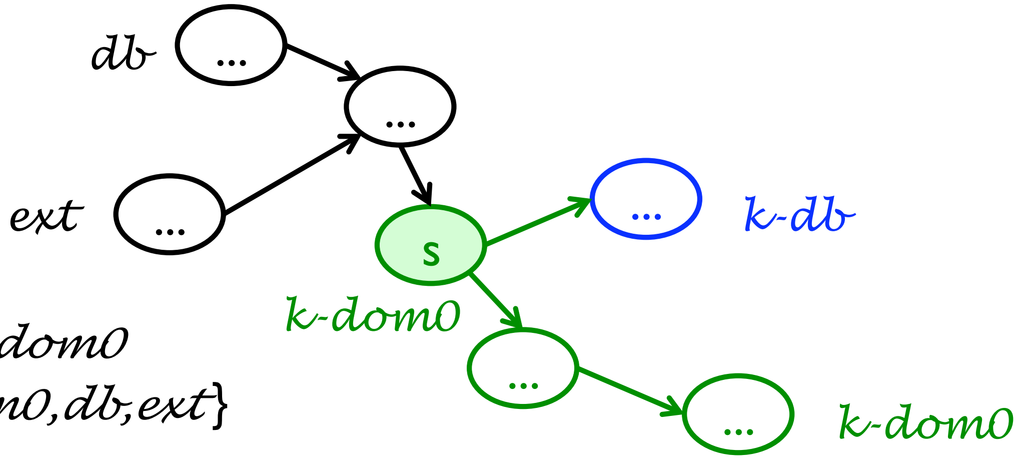
{appx} $\subseteq$ {serverx,appx}

request: ✗

answer: ✓

3b. Generating Flume Policy

- Processes with capabilities correspond to our mediators
- We want to generate Flume labels and capabilities
  - Mediator m:
    - L_m: GLB of the integrity levels that reach the node
    - D_m: integrity levels that may reach m
  - Non-mediator n:
    - L_n: GLB of the integrity levels that reach the node
    - D_n: {}
- Convert from levels to Flume tags
  - Flume label == levels dominated



L_s=k-dom0
D_s={k-dom0,db,ext}

Modeling Mediation Cost

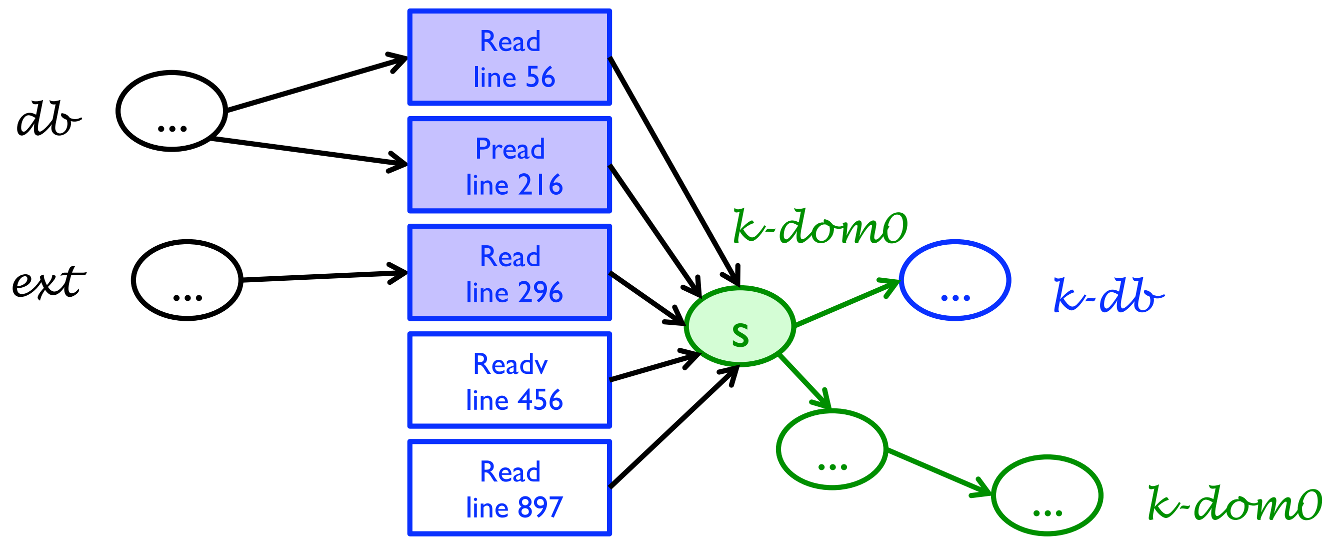
- We want to minimize the cost of mediation
  - The cost of mediators making information flow decisions correctly
- How is this determined?
  - Cuts identify the set of programs that must enforce information flow requirements
- What is mediation in programs?

Mediation Cost Options

- Per program
  - The mediation requirements of each program are the same
    - *Implies reusing same programs in multiple mediation cases*
- Per level transformation
  - Each mediation decision is the same
    - *Implies that the number of Flume capabilities is the cost* (default solution for multicut)
- Per program entry point
  - Adversaries may access the program in multiple ways (attack surface)
    - *Implies program has subset of interfaces that may require mediation*
    - How do we know which interfaces are accessible?

Attack Surface Cost

- Minimize attack surface size per cut problem
  - Result is the number of security decisions X number of entry points accessible to adversaries
  - Reuse same interfaces in subsequent cuts (may require multiple mediations at same interface)
  - Estimate from runtime analysis (like MAC policies themselves)

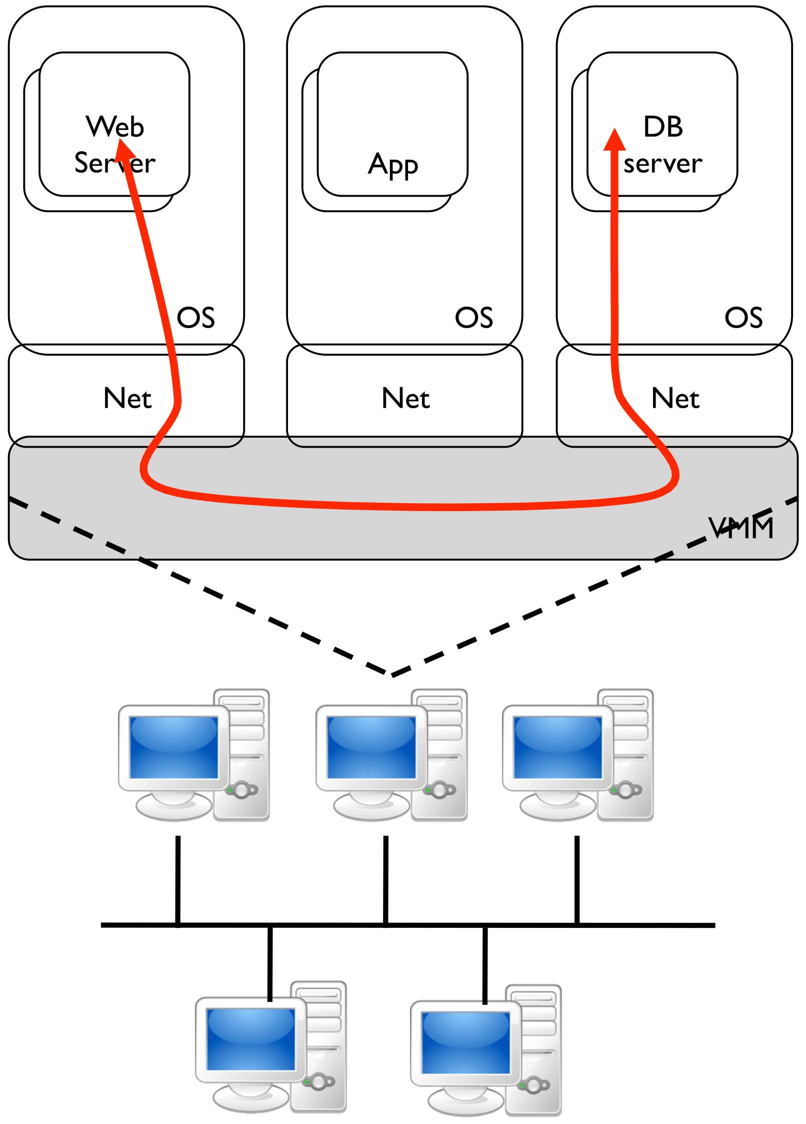


The Goal

- How should systems be built and deployed to achieve compliance?
- Build Software
  - Define mediated interfaces for programs
  - Which system calls are allowed to receive adversary data?
- Build OS Distributions
  - Create OS distribution deployment by specifying: (1) packages and network/VMM policies; (2) MAC policy; and (3) information flow model (semi-automated)
  - Generate MAC policy for deployment that complies with information flow model using program mediation (or revise model or MAC policy)
- Deploy Systems
  - Select OS distributions, choose program configurations, define network policy
  - Verify automatically that the deployment satisfies information flow model – can use in remote attestations also (for tomorrow's talk)

Experimental Testbed

- Distributed system with
  - XSM/Flask at the VMM layer
  - SELinux in the guest VMs
  - iptables with the Secmark extension governing network communications
- We customized the SELinux policies according to the applications the VMs would run:
  - Dom0
  - Database server
  - Web server
  - User VM



Questions

- We use our tool to explore different configurations for a distributed system

1. How many interfaces do developers need to mediate to make this deployment compliant ?
2. How do changes to functional requirements affect the mediation results ?

Question 1

1. How many interfaces do developers need to adjust to make this deployment compliant ?
   - Summarizing mediators (cut set)
   - Unique subjects: some subjects are repeatedly picked as mediators across different VMs (insmod_t for kernel_dom0, kernel_dbsrv, etc.)
   - The size of the cut represents the effort to implement filtering interfaces where needed

	Static	
Sink	**Sub**	**Int**
Kernel-dom0	32	1069
Kernel-dbsrv	0	0
dbdata	3	91
Kernel-uservm	6	469
Kernel-websrv	3	288
webdata	6	101
Total	50	2018

TCB subjects

APP subjects

Big Effort!

- Sub: Subjects
- Int: Interfaces

Question 2

2. How do changes to functional requirements affect the mediation results ?

- Runtime: permissions that are actually exercised at run time
- The main difference between static and runtime data is caused by definition of attributes in the MAC policy

Sink	Static policy		Runtime data	
	Sub	Int	Sub	Int
Kernel-dom0	32	1069	23	197
Kernel-dbsrv	0	0	0	0
dbdata	3	91	2	22
Kernel-uservm	6	469	7	138
Kernel-websrv	3	288	2	104
webdata	6	101	1	37
Total	50	2018	35	498

Reduction.
Runtime could guide policy tightening!

Execution Time

	System Configurations			Time (sec)			
	VMs	nodes	edges	HSM	GCM	Cut	DIFC
Q1	4	8905	77091	18.7	1.0	32.6	8.2
Q2	4	8610	35105	18.7	0.8	13.7	4.9

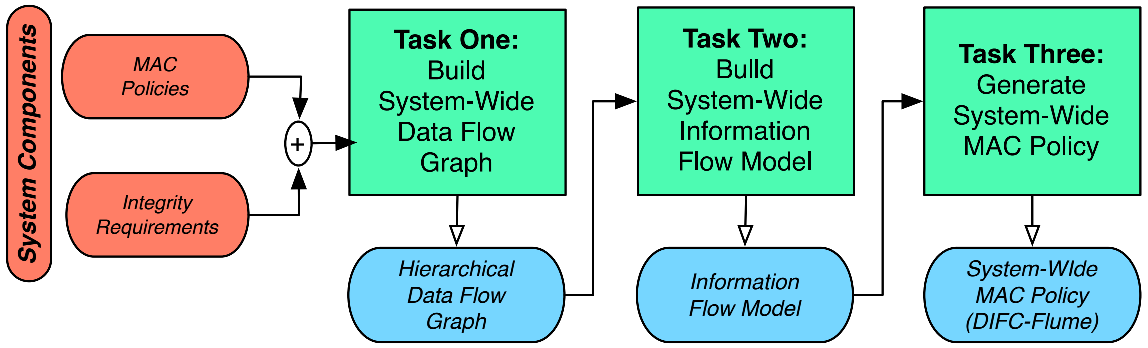
- HSM: Parse policies and generate HSM model
- GCM: Generate graph-cut model
- Cuts: Compute system-wide cuts
- DIFC: Generate DIFC policy

Project Tasks

- Collect and represent policies in OpenStack cloud system
  - Can we generate data flow graphs and compliance models for MAC and other relevant policies in OpenStack cloud system?
- Formalize definitions for cut problem, including cost functions and solution composition, for cloud systems
  - Can we resolve realistic system-wide compliance problems with minimum cost (approximately)?
- Explore methods to produce reasonable functional options to explore
  - Can we generate options/constraints for the policy designer that enables them to determine which permissions to authorize?
- Extend the research system to support solving such problems and testing on real cloud deployments
  - Can we produce cloud deployments that proactively protect themselves?

Summary

- We have made a lot of progress improve host security over the last ten years, but we are still *reactive*
- To defend systems proactively, we must design security defenses for the deployment
- We define a methodology to generate system-wide MAC policies that comply with information flow requirements automatically
- Such a methodology enables OS distributors to create compliant systems that system administrators and remote parties can verify automatically – proactive evaluation end-to-end


System Components
MAC Policies
Integrity Requirements
+
Task One: Build System-Wide Data Flow Graph
Hierarchical Data Flow Graph
Task Two: Bulld System-Wide Information Flow Model
Information Flow Model
Task Three: Generate System-Wide MAC Policy
System-WIde MAC Policy (DIFC-Flume)

Questions