

Certification Calculus for Services and Cloud Architectures

*2012 AFOSR PI Meeting
Information Operations and Security*

Rose Gamble
Tandy School of Computer Science
University of Tulsa

Current Objectives

- Develop a calculus to reason about information system compliance with security controls
 - information systems may be deployed as distributed interacting components and services, including clouds
- Evaluate the risk of non-compliance with respect to service and/or cloud use given stated security controls
- Examine audit security controls in depth and the assurances that are deliverable from within a cloud architecture

Recent Results

- Compliance predicates and hierarchy of dependencies representing security controls across multiple control documents
- X-Unity – Formal coordination language structured to express and reason over interacting, distributed architectures, such as SOA and clouds
- SecAg – an extension to WS-Agreement for factoring security controls into Service Level Agreements
 - Ontology development as derived from compliance predicates
 - Matchmaking algorithm to assess risk of service/cloud interactions
 - Extended algorithm to evaluate risk propagation based on third party invocation
- Initial scoping rules to define audit security control protocols for cloud entities
 - Preliminary filtering and composition rules of audit records from different scopes to manifest security anomalies
 - Case study implementation to illustrate audit protocols and assess overhead of audit record generation by services given specific auditable events

Publications

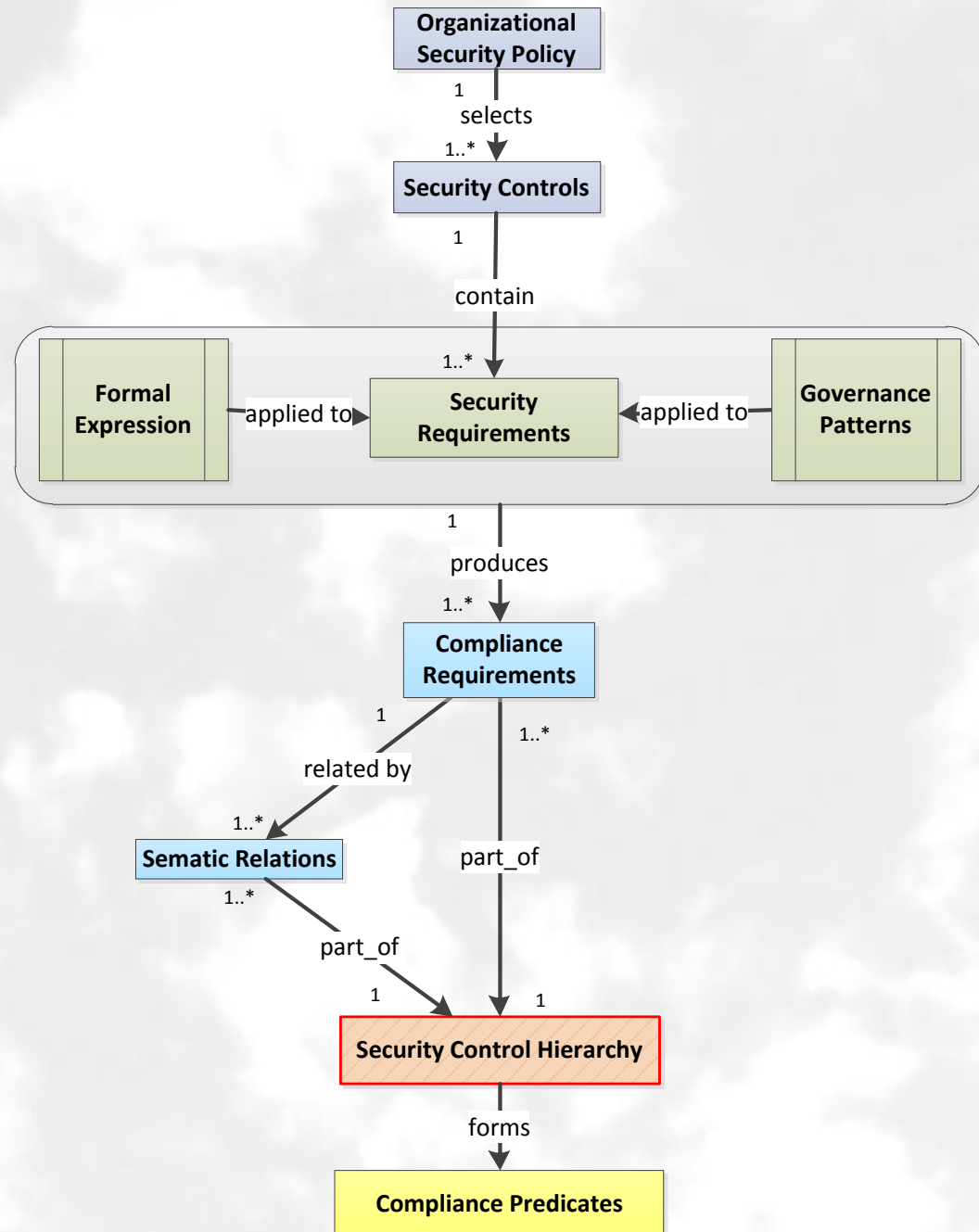
- To Appear
 - An Architecture for Cross-Cloud Auditing of Service Cloud, R. Xie and R. Gamble, *8th Annual Cyber Security and Information Intelligence Research Workshop*, to appear 2012.
 - Risk Propagation of Security SLAs in the Cloud, M. Hale and R. Gamble, *1st International Workshop on Management and Security Technologies for Cloud Computing*, to appear 2012.
 - Abstraction, Uniformity, and Comparison of Security Management Policies, M. Young, B. Nienhaus, and R. Gamble, in *Emerging Concepts in Security: Selected Articles from the 2011-2012 Security Conference*, Gurpreet Dillon, Ed., Information Institute, Washington, D.C. to appear 2012.
- Published
 - A Tiered Strategy for Auditing in the Cloud, R. Xie and R. Gamble, *5th IEEE International Conference on Cloud Computing*, 2012.
 - Architecting Web Service Attack Detection Handlers, A. Andrekanic and R. Gamble, *19th IEEE International Conference on Web Services*, 2012.
 - SecAgreement: Advancing Security Risk Calculations in Cloud Services, M. Hale and R. Gamble, *8th IEEE World Congress on Services*, 2012.
 - Security Policy Foundations in Context UNITY, M.T. Gamble, R. Gamble, and M. Hale, *7th International Workshop on Software Engineering for Secure Systems*, 2011.
 - Developing a Security Meta-Language Framework, R. Baird and R. Gamble, *44th Hawaii International Conference on System Sciences*, 2011.
 - Security Controls Applied to Web Service Architectures, R. Baird and R. Gamble, *International Conference on Software Engineering and Data Engineering*, 2010.
 - Extracting Security Control Requirements, J. Hosey and R. Gamble, *Cyber Security and Information Intelligence Research Workshop*, 2010.
 - Reasoning about Policy Noncompliance, R. Baird and R. Gamble, *Cyber Security and Information Intelligence Research Workshop*, 2010.

Other Publication Information

- Under Review
 - Ensuring Virtual Isolation in the Cloud, M. Hale and R. Gamble, submitted to *35th International Conference on Software Engineering*, under review for 2013.
 - Composing Security Audit Trials in Cloud Architectures, R. Xie and R. Gamble, submitted to the *3rd ACM Conference on Data and Application Security and Privacy*, under review for 2013
 - Secure Data Stream Processing in Cloud Environment, X. Xie, I. Ray, R. Adaiikkalavan, and R. Gamble, submitted to the *3rd ACM Conference on Data and Application Security and Privacy*, under review for 2013.
 - Hierarchically Modeling Security Control Compliance, M. Hale and R. Gamble, *IEEE Transactions on Knowledge and Data Engineering*, under review 2012.
 - A Security Meta-Language for SOAP Messaging, R. Baird and R. Gamble, submitted to the *Journal of Internet Services and Applications*, under review 2011.
- In Preparation
 - Detecting Attacks in Trust Chains of Web Services, R. Xie and R. Gamble, *Service Oriented Computing and Applications*, in preparation, 2012.
 - Reasoning about Security Control Governance in Cloud Architectures, M. Hale and R. Gamble, *Science of Computer Programming*, in preparation for submission in Nov. 2012.
 - An Architecture for Cross-Cloud Auditing of Service Clouds, R. Xie and R. Gamble, *IEEE Transactions on Cloud Computing* (new), in preparation for submission in Dec. 2012.

Compliance Predicates

- Pivotal in verifying presence of security controls in a multi-component information systems such as a SOA or Cloud
- Provides uniform, logic-based representation for comparative compliance verification across heterogeneous entities
- Broad representation
 - Derived from multiple governing documents
 - NIST sp800-53
 - DoDI 8500.2 and related STIGs
 - Common Criteria, part 2
 - FedRAMP Security Controls
 - Cloud Security Alliance, Guidance 3.0
 - Hierarchical dependencies uncovered
 - Semantic relations expressed to represent direct links among predicates



Security Controls to Compliance Requirements

- Intent Classes – equivalence class of security control intent
- Assign patterns based on security control goal within the intent class and state-based requirements

– Imposes

- Denotes the organizational selection and specification of parameters as policy entities on selected assets

Org *imposes* <policy entities> on <assets>

– Performs

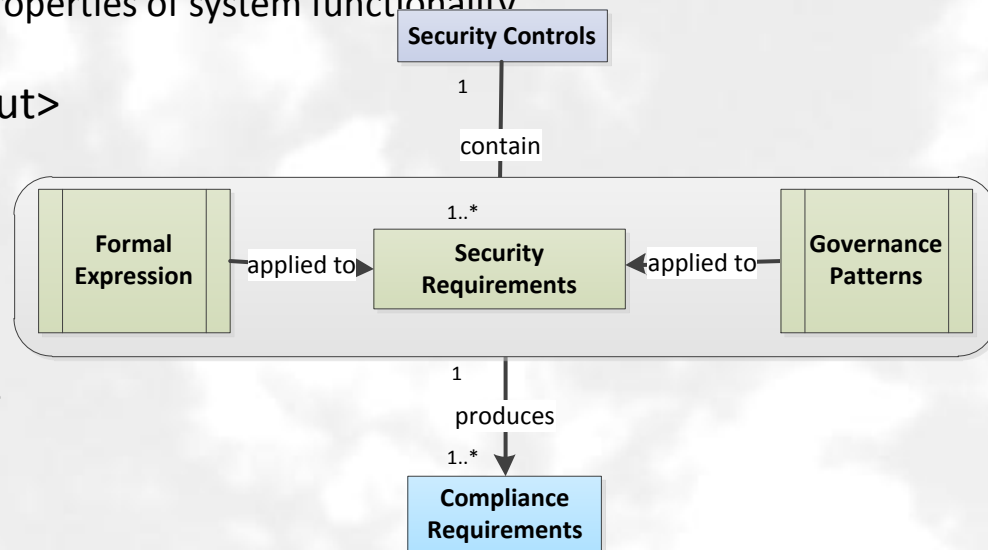
- Expresses constraints that related directly to properties of system functionality

<level> *performs* <action>
given <preconditions> and <input>
that results in <postconditions>

– Protects

- Expresses assets, processes, or functions to be protected against potential malicious activity

<level> *protects* <asset>
using mechanism <mechanism>
to prevent <activity>



Imposes

FAU_GEN.1.2:

The TSF shall record within each audit record at least the following information: a) Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event; and b) For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST, [assignment: *other audit relevant information*]

Patterned statement:

Org *imposes* minimum content parameters (type, timestamp, location, source, outcome, UID) *on* audit records

Compliance Requirement

[CC, FAU_GEN.1.2]
event : \uparrow AuditableEvents timestamp : String location : String source : Component outcome : String event_UID : String additional_content : Set
\forall record : Tuple record \square (event, timestamp, location, source, outcome, event_uid)

Performs

FAU_GEN.1.1(c) Audit data generation

The TSF shall be able to generate an audit record of the following auditable events [assignment: *other specifically defined auditable events*].

Patterned statement:

System *performs* audit record generation
given an unrecorded event occurs and declared auditable events and components
that results in an audit record

Compliance Requirement

[CC, FAU_GEN.1.1(c)]
c : Component e : \uparrow AuditableEvent Gen_Rec : \uparrow AuditableEvent \rightarrow \uparrow AuditRecord
\exists ar : \uparrow AuditRecord [ar \notin c.auditLog \wedge occurs(e) \wedge ar = Gen_Rec(e)] \wedge c.auditLog' = c.auditLog \cup {ar}]

Security Controls to Compliance Requirements

Security Controls to Compliance Requirements

AU-10.G Non-Repudiation

The information system protects against an individual falsely denying having performed a particular action.

Patterned statement:

System *protects* audit records
using mechanism {} to
prevent repudiation
<dependent on other control
mechanisms>

Compliance Requirement

[NIST, AU-10.G]
$r : \uparrow \text{AuditRecord}$ $\text{repudiation} : \text{Activity}$ $\text{prevents} : \text{Activity} \rightarrow \text{Boolean}$
$\text{repudiation} = \lambda x. \text{repudiate}(x)$ $\text{prevents}(\text{repudiation}(r)) =$ $\text{signUID}(r) \vee \text{validateUID}(r) \vee$ $\text{review}(r) \vee \text{transfer}(r)$

Protects

AU-10.E3 Non-Repudiation

The information system maintains reviewer/releaser identity and credentials within the established chain of custody for all information reviewed or released. If the reviewer is a human or if the review function is automated but separate from the release/transfer function, the information system associates the identity of the reviewer of the information to be released with the information and the information label.

Patterned statement:

InfoSys *performs* reviewer identity binding
given audit record review and
parameters (audit record, reviewer,
binding mechanism)
that results in signed reviewer UID
in the record

Compliance Requirement

[NIST, AU-10.E3]
$r : \uparrow \text{AuditRecord}$ $e : \text{Event}$ $\text{rev_uid} : \text{String}$ $\text{bindRev} : \uparrow \text{AuditRecord} \times \text{String} \rightarrow$ $\uparrow \text{AuditRecord}$ $\text{review} : \uparrow \text{AuditRecord} \rightarrow \text{Boolean}$
$\text{review}(r) =$ $\exists r' : \uparrow \text{AuditRecord} \mid$ $e = \text{audit-record-review}(r) \Rightarrow$ $[(\text{occurs}(e) \wedge r.\text{uid}.\text{verified})$ $\mathbf{B} (r' = \text{bindRev}(r, \text{rev_uid}) \wedge$ $r'.\text{revUID}.\text{signed})]$

Compliance Requirements to Security Control Hierarchy

- Semantic relations
 - subsumedBy, usedBy, structures, refines, forms

[NIST, AU-2.G]
AE : eventType x ObjectID x UserID x SubjID x HostID x AuditLevel x SecrecyLevel x Other
$\langle \forall e : AE \mid e \sqsubseteq \text{event} \rangle \wedge$ $\{ \text{labelModification, privFuncExec, login, dataRead, dataWrite, dataReadFail, dataWriteFail, dataDelete, auditStartup, auditShutdown} \}$ $\subseteq AE$

structures(AE)

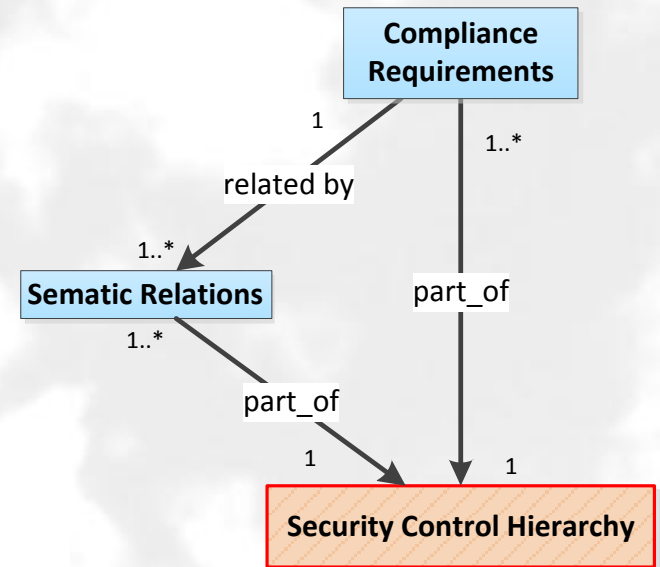
[CC, FAU_SEL.1.1]
eventType : String objectID : String userID : String subjectID : String hostID : String other : Set
event \sqsubseteq (eventType, objectID, userID, subjectID, hostID, secrecyLevel, detailLevel)

structures(event)

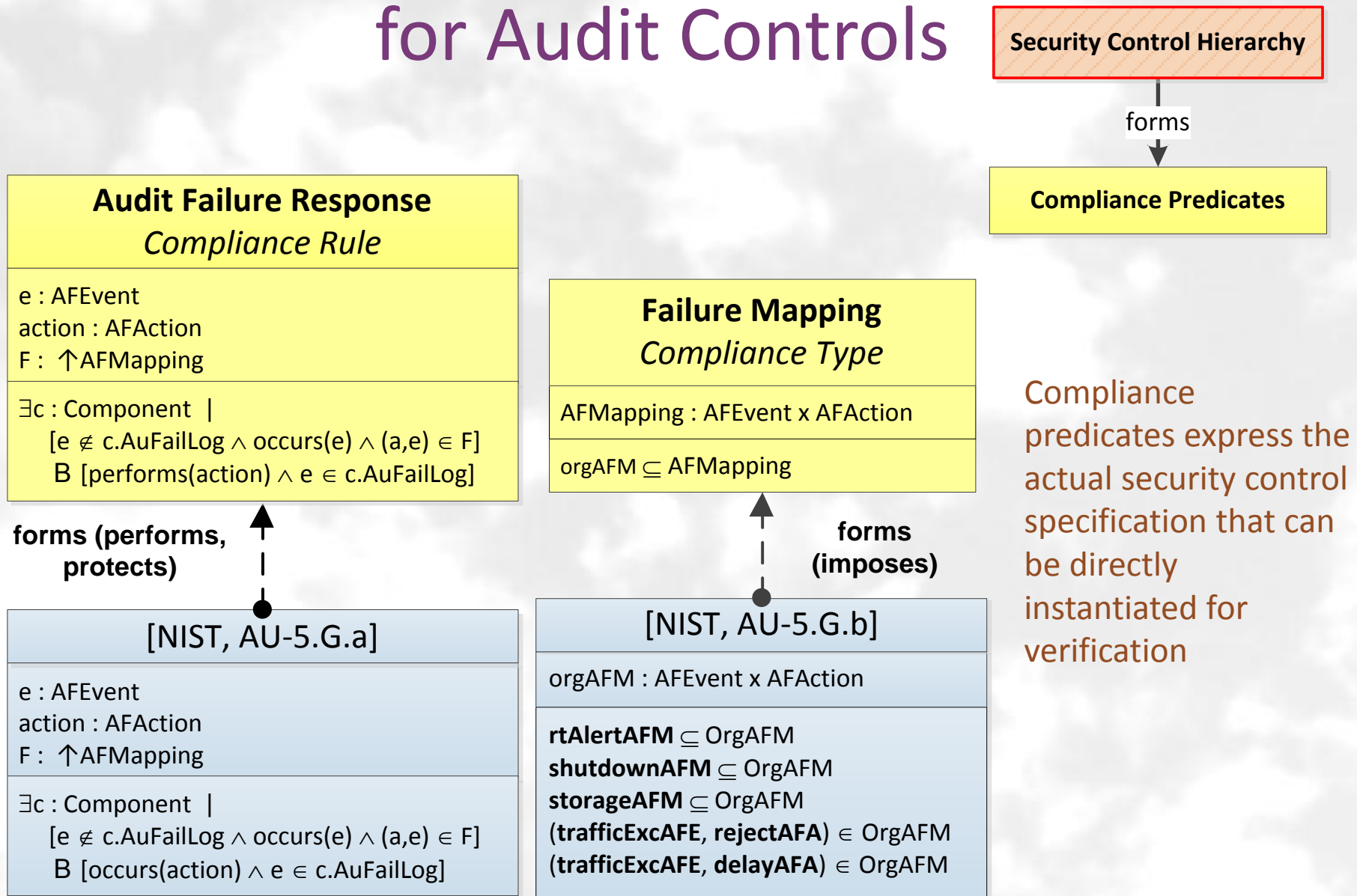
[STIG, APP3620]
secrecyLevel : String
secrecy_level = "Top Secret" \oplus "Secret" \oplus "Unclassified"

structures(event)

[CC, FAU_GEN.1.1(b)]
detailLevel : String
detailLevel = "detailed" \oplus "basic" \oplus "minimum" \oplus "unspecified"

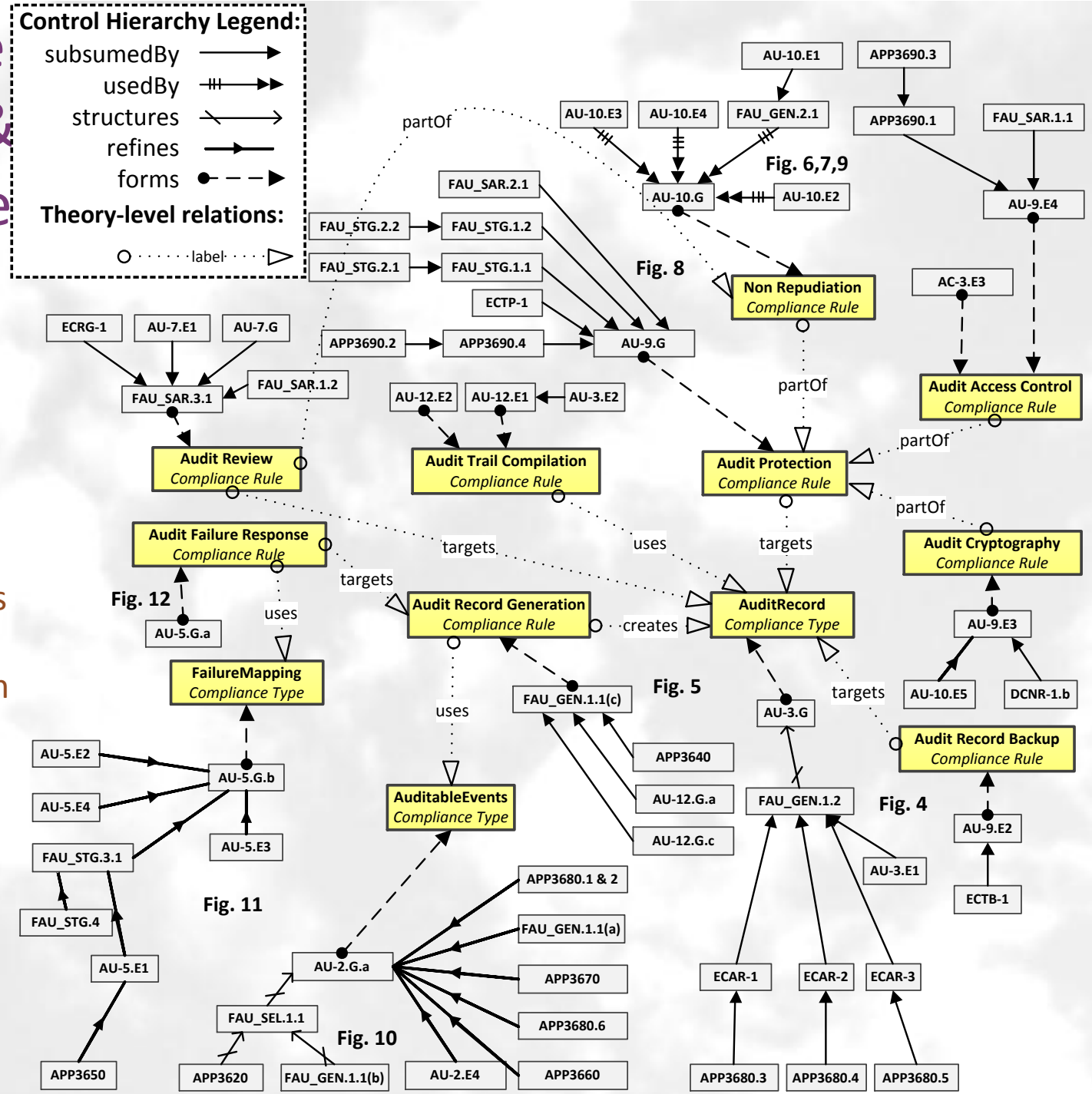


Security Control Hierarchy for Audit Controls



Compliance Predicates & Dependencies for Audit Controls

Connectively
among compliance
predicates indicates
the component
security profile with
respect to chose
security controls
and what is
impacted if one is
violated



Benefits of Representation

- The compliance predicates provide a representational substitution for natural language and heterogeneous document structures
 - Reduce ambiguity in expectations, application, and impact assessment upon violation
 - Introduce improved mechanisms for instantiation and verification
 - Specification of controls and services/components allows direct formal verification of embedded, critical properties
 - Authorization, encryption, authentication, and audit can be understood logically in terms of format requirements and behavior during certain state changes in the system
 - Bridge expression and reasoning capabilities needed to divulge detailed properties and expectations in cloud computing
 - Yield an ontological basis for comparison and risk analysis
 - Allows for risk assessment at the service/cloud interface level
 - “Surface” level assessment needed to understand third party risk propagation
 - Provide a perspective for cloud compliance given certain security concerns in cloud models
 - Loss of control over data access and transfer
 - Third party input that may involved information tainting
 - Isolation of data for sanctioned sharing among interacting services

X-UNITY for Cloud Architectures

- Coordination language specification
 - Tuple is the foundational element for manipulation and sharing
 - Compliance predicates related to access control, audit, and encryption are directly associated with each tuple to make it security-aware
 - Allows for abstract service specification
 - Security policies can be specified against program state changes
 - Introduces security awareness into the service
 - Traces of state changes assessed against policy allow for verification of compliance
 - Dynamic interaction of services is represented as tuple sharing
- Cloud model can be specified using the hierarchal foundation of the language
 - Introduces issues with sharing among service compositions
 - By default, data should be isolated within the service compositions, especially when services create proxies for each composition the participate in.

X-UNITY Coordination Language Specification

System *CloudName*

Service *ServiceName₁*

...

Service *ServiceName_N*

Components

Instances of services deployed in the cloud, declared with parameters, unique names, and functional type

Governance

CompositionMgr

Forms service compositions. Has access to all exposed variables in the global tuple space and uses **allocate-service** and **deallocate-service**.

SpaceMgr

Allocates and de-allocates session tuple spaces and propagates the proper credentials to service proxies. Has access to all exposed variables in the global tuple space and uses **allocate-space** and **deallocate-space**.

GeneralMgr

Other global impact statements that can access any exposed program or service variable

end *CloudName*

Service *ServiceName* (parameters)

declare

internal – internal variable declarations

exposed – exposed variable declarations

context – context variable declarations

initially – initial conditions of variables

assign

– assignment statements to declared service variables

context

– context rules affecting context variables

policy

declarations to associate security controls with exposed variables via **access**, **audit**, and **encrypt**

session (params) //accepts proxy parameters

declare

internal – scoped to the session

exposed – scoped to session tuple space

context – interact with variables in the same session space or global space

initially – initial conditions of session variables

appear as **allocate-space()** parameters as

determined by *SpaceMgr* called at

assign – assignments to declared session variables

context

context rules affecting session context variables

end *ServiceName*

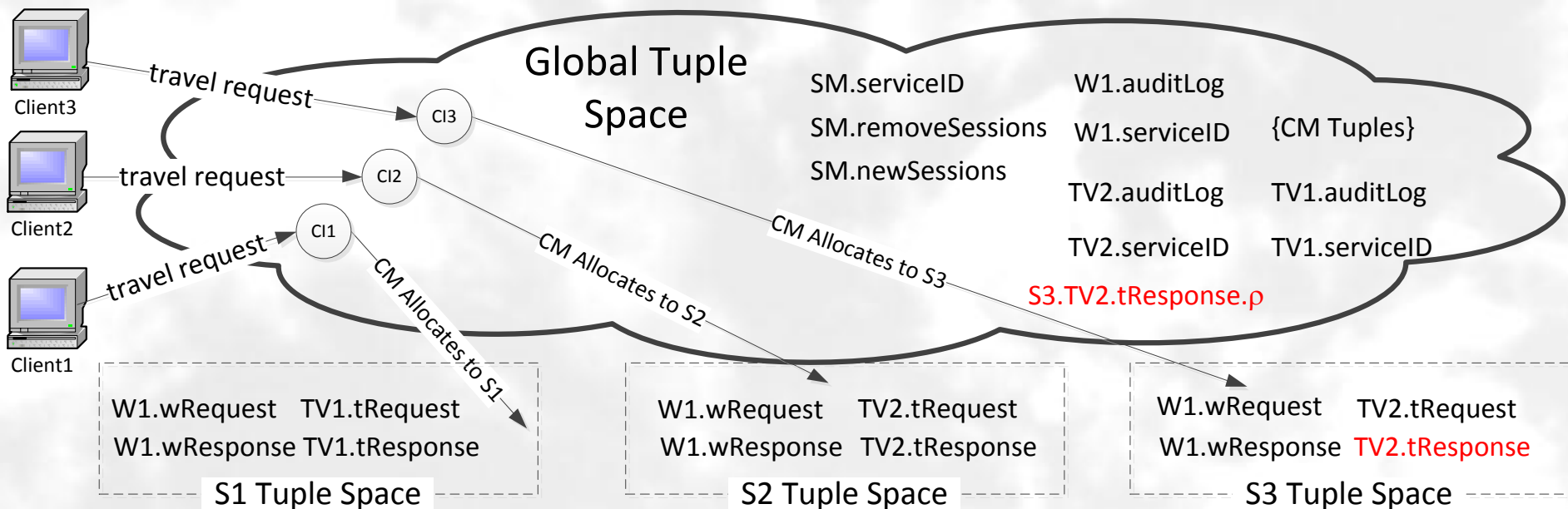
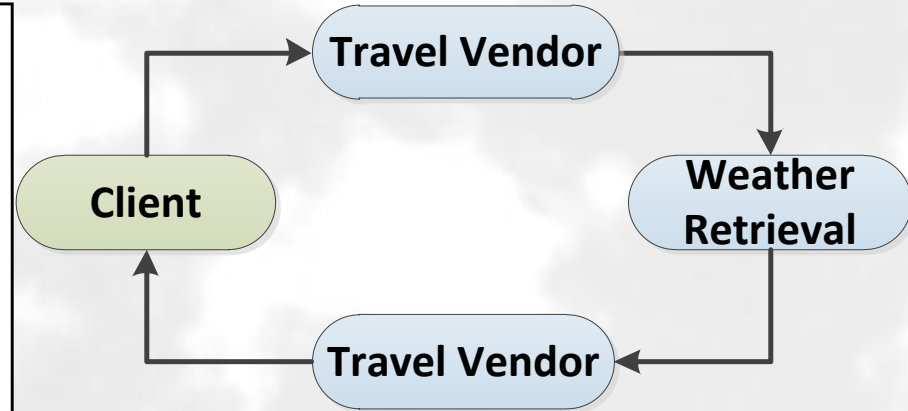
Service Specification and Verification of Virtual Isolation

Components

Weather(W1, weatherPrivateKey)
TravelVendor(TV1, travelVendor1PrivateKey)
TravelVendor(TV2, travelVendor2PrivateKey)
ClientInterface(CI1)
ClientInterface(CI2)
ClientInterface(CI3)

Governance

CompositionMgr
SpaceMgr
GeneralMgr



Sample Service Specification

Weather Service Specification

Travel Vendor Session Specification (only)

Service *Weather*(*params* : *List of Parameters*)

declare

internal

sessions : Set of (SessionID, Certificate)

meta-data : Tuple

weatherServiceKey : PrivateKey

exposed

id : ServiceID **access** {'*', {r}}

auditLog : set of AuditRecord **access** {(GeneralMgr, {r})}

encrypt {AES}

context

newSessions : Set of (SessionID, Certificate, ServiceID)

remSessions : Set of (SessionID, ServiceID)

initially

id, *weatherServiceKey*, *meta-data*, *sessions*, *auditLog* :=

params[0], *params*[1], *params*[2], \emptyset , \emptyset

context

newSessions **uses** *ns*!*newSessions* in *SessionMgr*

given // *SessionMgr* selected service to create a proxy

n : *n* ∈ *ns* :: *id* = *n*[2]

where

newSessions **becomes** *ns*

sessions := *sessions* ∪ {(*n*[0], *n*[1])}; **allocate-proxy**(*n*[0])

ns − {*n*} **impacts** *ns*

remSessions **uses** *rs*!*remSessions* in *SessionMgr*

given // *SessionMgr* signaled session revocation

n : *n* ∈ *rs* :: *id* = *n*[1]

where

remSessions **becomes** *rs*

sessions := *sessions* − {*n*}; **deallocate-proxy**(*n*[0])

rs − {*n*} **impacts** *rs*

session

declare

exposed

wRequest : (ServiceID, Date, Location) **access** {'TV', {w}}

wResponse : (ServiceID, JSON) **access** {'TV', {r}}

initially

wRequest, *wResponse* := \emptyset , \emptyset

assign

⟨*req* : *req* = *wRequest* ::

wResponse := (*req*[0], *Forecast*(*req*[1], *req*[2]));

wRequest := \emptyset ⟩

policy

wRequest **spaces**{all}: * **audit** {w}

wResponse **spaces**{all}: * **audit** {r}

* · * **audit** {r-fail, w-fail}

end *Weather*

session

declare

exposed

tRequest : (ServiceID, FromDate, ToDate, From, To) **access** {'CI', {w}}

tResponse : (ServiceID, JSON) **access** {'CI', {r}}

context

weatherReq : Tuple(ServiceID, Date, Location)

weatherRes : Tuple (ServiceID, JSON)

initially

tRequest, *tResponse* := \emptyset , \emptyset

assign

⟨*req* : *req* = *tRequest* ::

tResponse := (*req*[0], *TravelPlan*(*req*[1], *req*[2]) +

weatherRes[1]) **reacts-to** *weatherRes* ≠ \emptyset ⟩

context

weatherReq **uses** *r*!*wRequest* in *p*

given

tRequest ≠ \emptyset

where

weatherReq **becomes** (*id*, *tRequest*[1], *tRequest*[4])

weatherRes **uses** *r*!*wResponse* in *p*

where

weatherRes **becomes** *r*

\emptyset **impacts** *tRequest*

weatherRes **uses** *r*!*wResponse* in *p*

where

weatherRes **becomes** \emptyset

\emptyset **impacts** *tRequest*

reactive

Sample Service Specification

```
Service Weather(params : List of Parameters)
declare
  internal
    sessions : Set of (SessionID, Certificate)
    meta-data : Tuple
```

context

newSessions **uses** *ns!newSessions* in *SessionMgr*

given // *SessionMgr* selected service to create a proxy

$n : n \in ns :: id = n[2]$

where

newSessions **becomes** *ns*

sessions := *sessions* $\cup \{(n[0], n[1])\}$; **allocate-proxy**(*n*[0])

ns - {*n*} **impacts** *ns*

remSessions **uses** *rs!remSessions* in *SessionMgr*

given // *SessionMgr* signaled session revocation

$n : n \in rs :: id = n[1]$

where

remSessions **becomes** *rs*

sessions := *sessions* - {*n*}; **deallocate-proxy**(*n*[0])

rs - {*n*} **impacts** *rs*

on Specification (only)

Sees its new session in the tuple space and allocates a proxy to the session

access {'CI', {r}}

Date, Location)
JSON)

$n(req[1], req[2]) +$
 $erRes \neq \emptyset$

$quest[1], tRequest[4])$

```
assign
  <req : req=wRequest ::
    wResponse := (req[0], Forecast(req[1], req[2]));
    wRequest := <>
```

policy

```
wRequest spaces{all}: * audit {w}
wResponse spaces{all} : * audit {r}
* * audit {r-fail, w-fail}
```

end Weather

```
weatherRes becomes r
  <> impacts tRequest
weatherRes uses r!wResponse in p
where
  weatherRes becomes <>
  <> impacts tRequest
  reactive
```


Sample Service Specification

Weather Service Specification

Travel Vendor Session Specification (only)

Service *Weather*(*params* : List of Parameters)

declare

internal

sessions : Set of (SessionID, Certificate)

meta-data : Tuple

weatherServiceKey : PrivateKey

exposed

id : ServiceID **access** {'*', {r}}

auditLog : set of AuditRecord **access** {(GeneralMgr, {r})}

encrypt (AES)

context

newSess

remSess

initially

id, *weather*

params

context

newSession

given

n : n

where

newSe

session

ns - {

remSession

given

n : n

where

remSe

session

rs - {

session

declare

exposed

wReq

wResp

initially

wReques

assign

<req : req

wResp

wReq

policy

wRequest s

wResponse spaces

** : * audit*

{r-fail, w-fail}

end *Weather*

session

declare

exposed

wRequest : (ServiceID, Date, Location) **access** {'(TV,' {w}}

wResponse : (ServiceID, JSON) **access** {'(TV,' {r}}

initially

wRequest, *wResponse* := \emptyset , \emptyset

assign

<req : req = *wRequest* ::

wResponse := (*req*[0], *Forecast*(*req* [1], *req*[2]));

wRequest := \emptyset

policy

wRequest spaces{all} : * **audit** {w}

wResponse spaces{all} : * **audit** {r}

* : * **audit** {r-fail, w-fail}

Answers the weather request within the session space

e, *ToDate*, *From*, *To*) **access**

access {'(CI,' {r}}

Date, *Location*)

JSON)

m(*req* [1], *req*[2]) +

ierRes $\neq \emptyset$

request[1], *tRequest*[4]))

p

Applies security policy to where the data is allowed to reside and how it is audited

\emptyset **impacts** *tRequest*
reactive

Sample Service Specification

Service Weather(params : List of Parameters)

declare

internal

sessions : Set of (SessionID, Certificate)

meta-data : Tuple

weatherServiceK

exposed

id : ServiceID ac

auditLog : set of

encrypt {AES

context

newSessions : Set

remSessions : Set

initially

id, weatherServiceK

params[0], para

context

newSessions uses ns

given // Sessio

n : n ∈ ns :: id

where

newSessions b

sessions := se

ns - {n} impa

remSessions uses rs

given // Sessio

n : n ∈ rs :: id

where

remSessions b

sessions := se

rs - {n} impa

session

declare

exposed

wRequest : (S

wResponse : (

initially

wRequest, wResp

assign

⟨req : req=wReq

wResponse :=

wRequest := ∅

policy

wRequest spaces{a

wResponse spaces{a

* · * audit {r-fail, w-fail}

end Weather

context

weatherReq uses *r!wRequest* in *p*

given

tRequest ≠ ∅

where

weatherReq becomes (*id*, *tRequest*[1], *tRequest*[4]))

weatherRes uses *r!wResponse* in *p*

where

weatherRes becomes *r*

∅ impacts *tRequest*

weatherRes uses *r!wResponse* in *p*

where

weatherRes becomes ∅

∅ impacts *tRequest*

reactive

Travel shares the weather request tuple because of the session space sharing rules and sets the request

Looks for non-empty response from Weather to process

∅ impacts *tRequest*
reactive

Sample Cloud Property: Virtual Isolation

- Guaranteeing information isolation during dynamic information sharing events
 - Virtual isolation requires hiding items in a session tuple space with respect to other sessions and the global environment
- Ensuring that tuples (data) are related to service proxy executions and viewable only by approved services

Sample Cloud Property: Virtual Isolation

- Guaranteeing information isolation during dynamic information sharing events
 - Limit access to items in a session tuple space
 - Ensure that tuples (data) are related to service proxy executions and viewable only by approved services

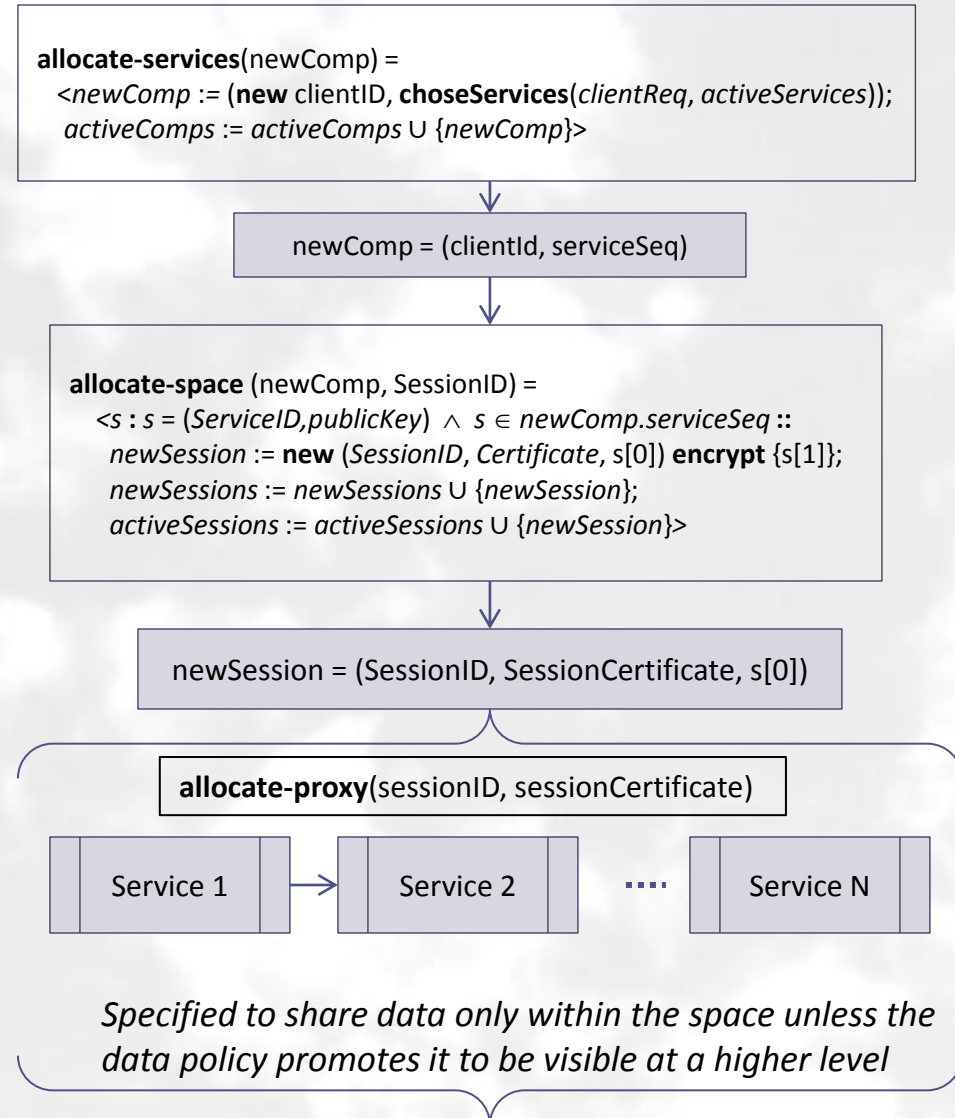
virtual isolation \equiv

$$\langle p, p', s : p, p' \in \text{Proxies} \wedge s \in \text{activeSessions} ::$$

$$[\text{in}(p, s) \wedge \neg \text{in}(p', s)] \Rightarrow$$

$$\langle x : x \in \text{variables}(p) :: \neg \text{visible}(x, p') \vee$$

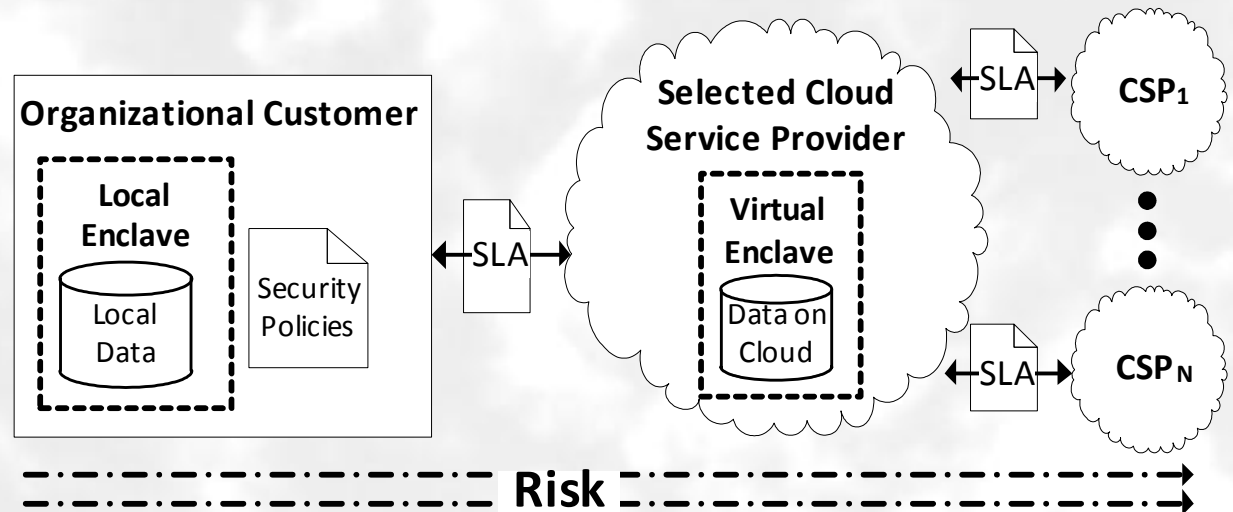
$$[\text{var}[x].\rho = (\text{allow}(x') \vee \text{allow-redact}(x'))$$

$$\Rightarrow \text{visible}(x', p') \rangle \rangle$$


Service, spaceMgr, and compositinMgr **deallocate** by removing the relevant tuples from the session and global space (if applicable).

Risk Assessment

- Cloud architectures
 - Requires Service Level Agreements (SLAs) and other standards for evaluation and choice
 - Specifying security in SLAs needs uniformity in representation, terminology and comparison



Anatomy of WS-Agreement for a SLA

WS-Agreement Template

Name

Context

Terms

Service Description Terms

Service Properties

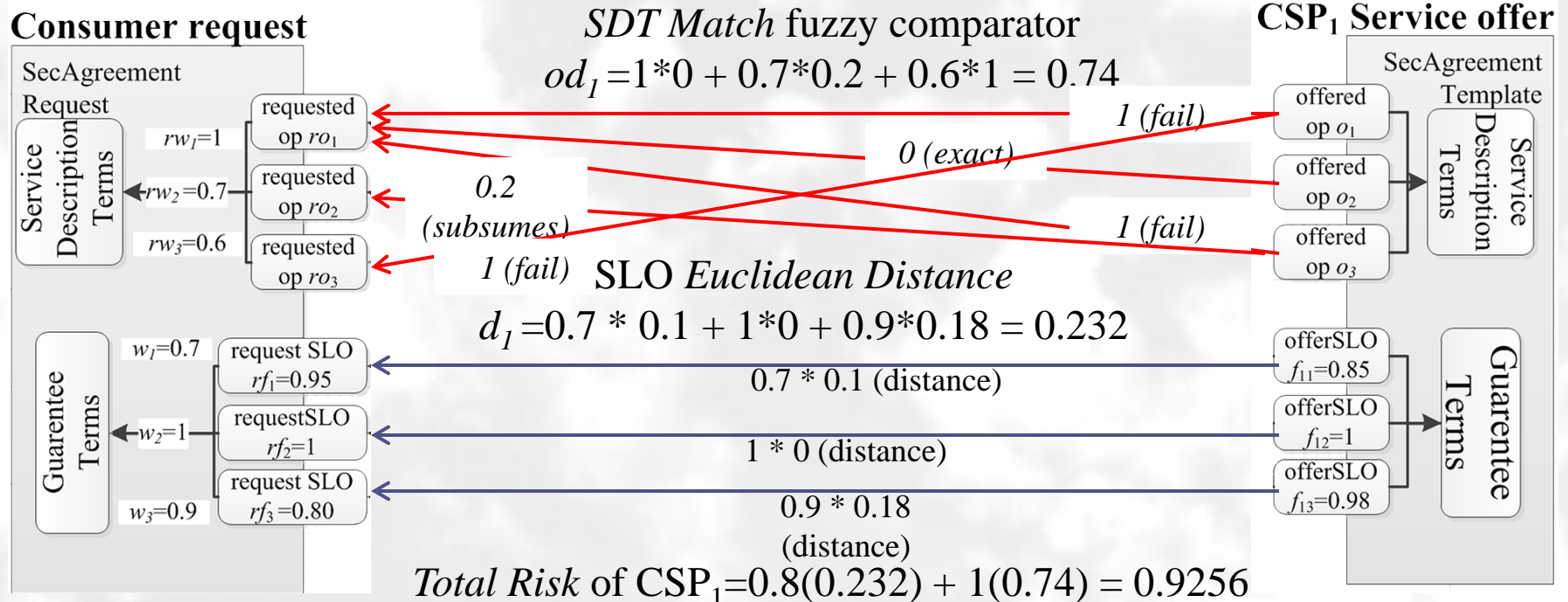
Guarantee Terms

Agreement Creation Constraints

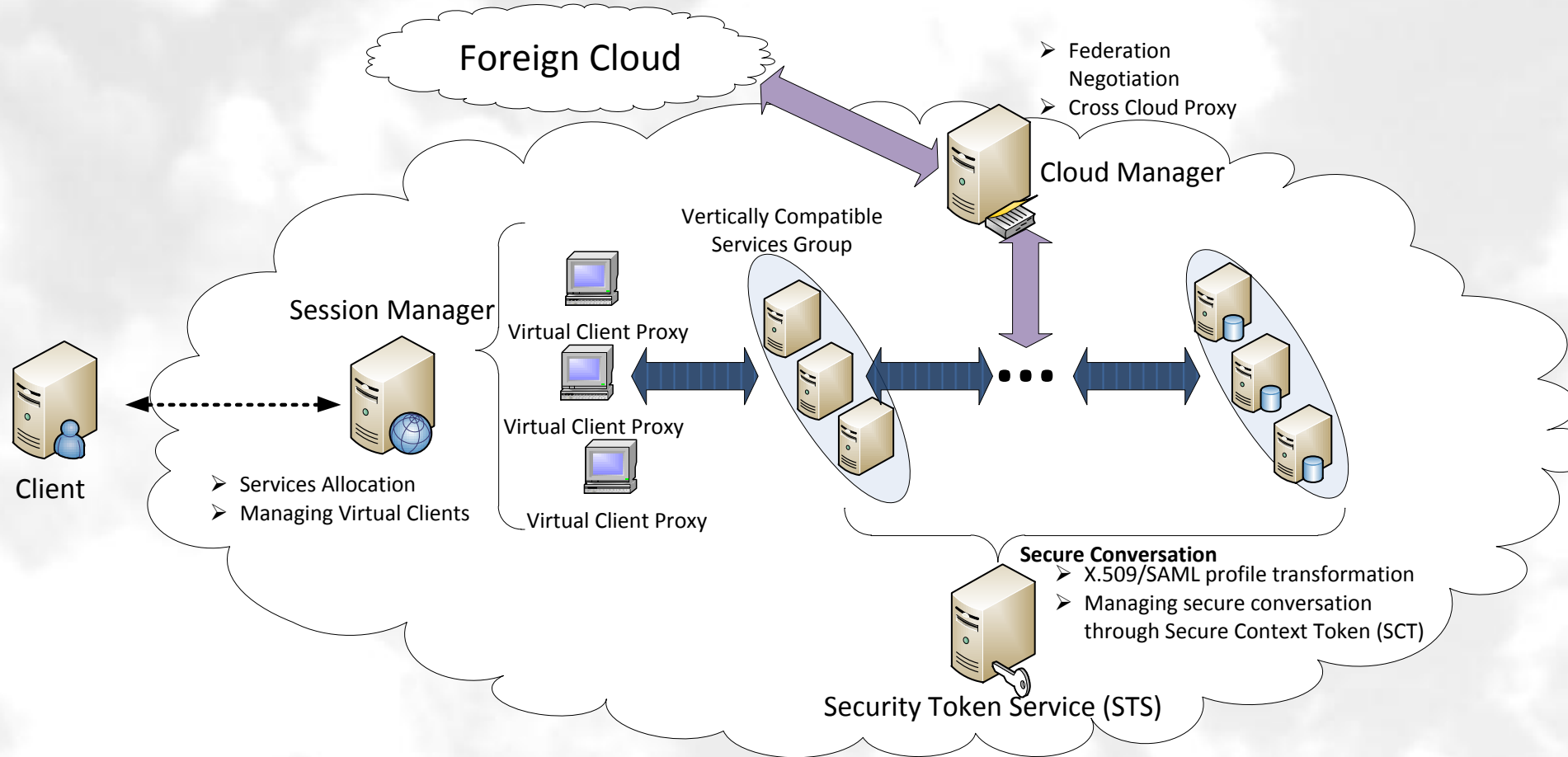
- WS-Agreement elements:
 - XML schema for SLA formation
 - Service Description terms (SDTs) qualify the service functionality offerings
 - Service Properties define metrics over SDTs that quantify performance
 - Guarantee Terms specify Service Level Objectives (SLO)s and the context(s) in which they apply
- **SecAgreement** provides a promising alternative
 - Extends WS-Agreement (SLA formation WS standards specification)
 - Allows for security to be defined over the terms of service
 - Provides schema for expressing quality of security service (QoSS) as service level objectives (SLOs)
 - Allows organizational risk to be attached to SLA terms and used by risk based matchmaking algorithms to assess cloud offerings
 - Matchmaking allows a risk assessment of service and cloud providers

Example Matchmaking

- Matchmaking consists of two parts
 - First, the SDTs are compared using the *match* fuzzy comparator
 - od_1 is the result, describes the total distance between requested and offered SDTs
 - Second, the SLOs are compared using a Euclidean distance between service levels
 - d_1 is the result, describes the total distance between requested and offered SLOs
- Combining the two yields the total risk to the consumer incurred by using CSP_1

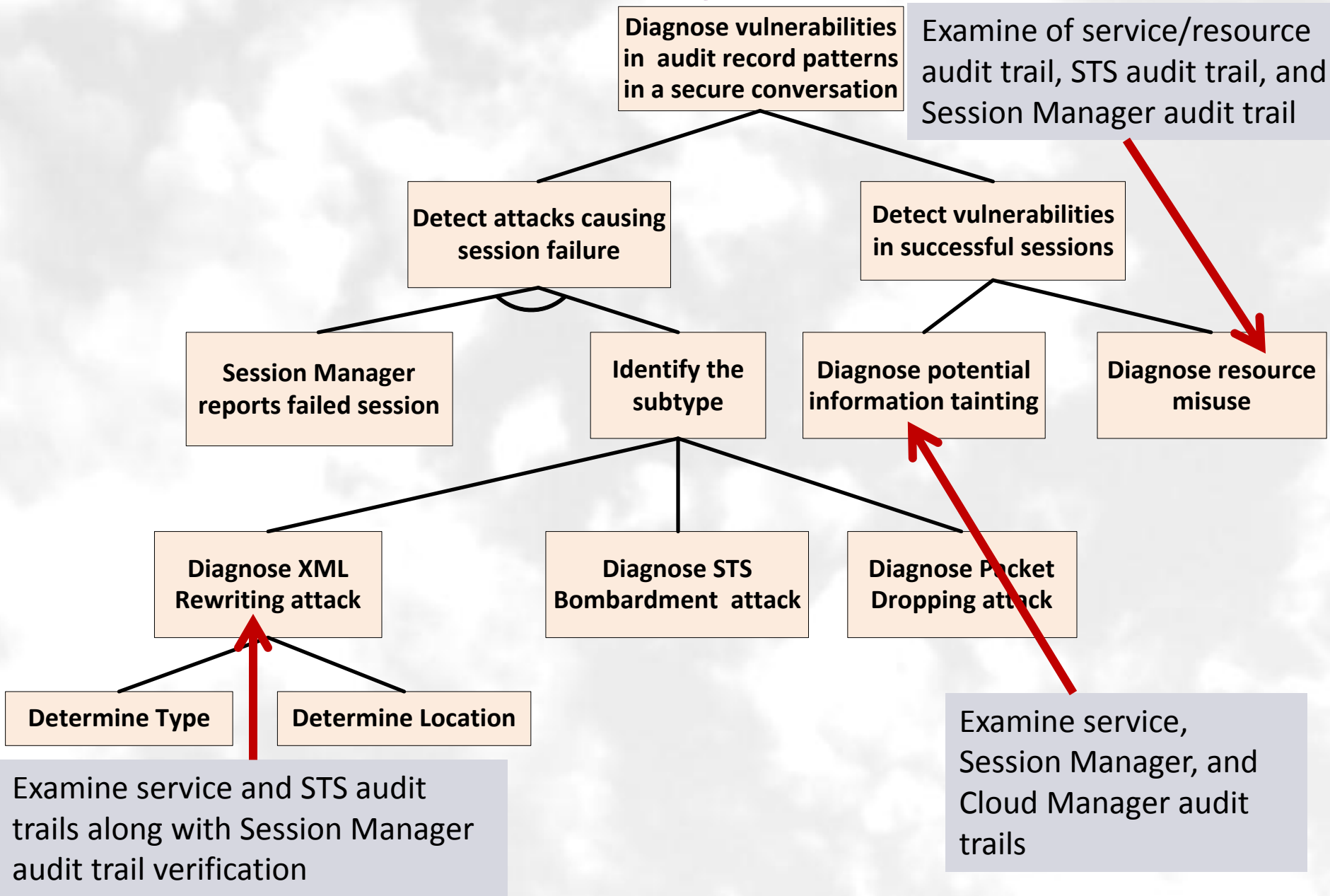


Audit Scoped Cloud Architecture

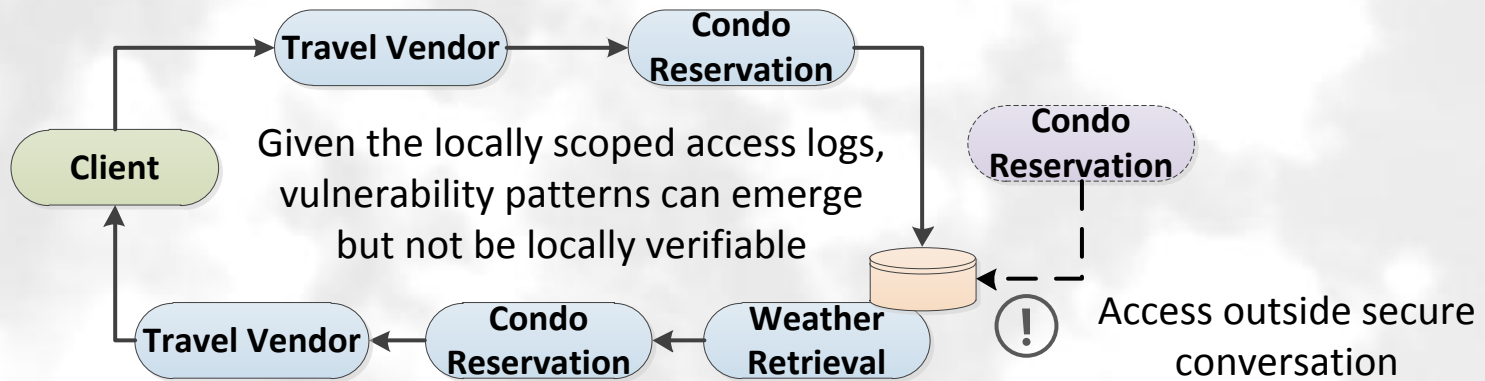


- Entities must comply with requirements for information capture
- Security requirements direct the scope of information and any sharing constraints

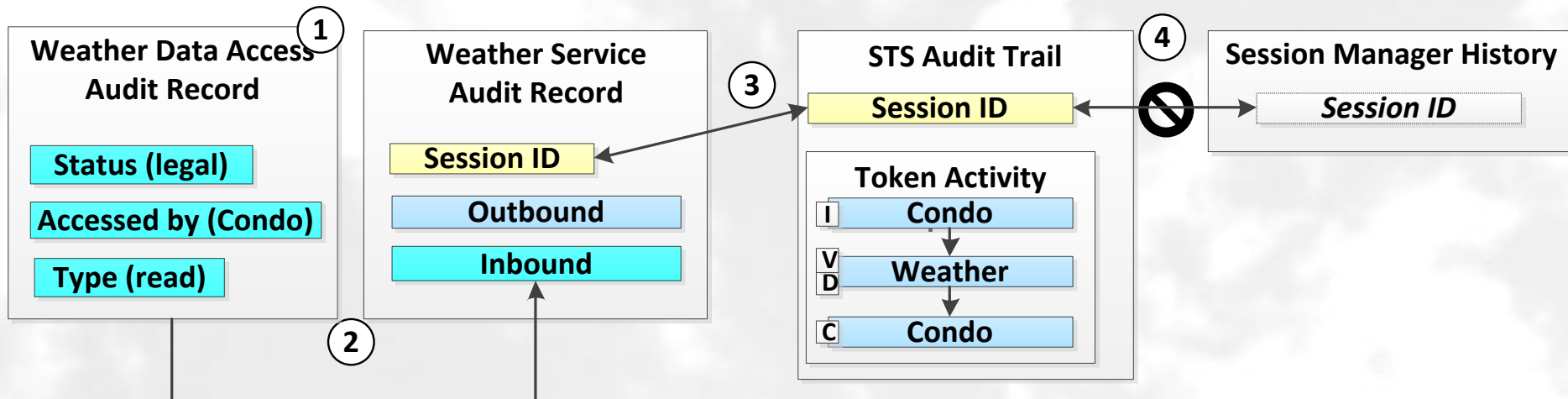
Vulnerability Patterns



Example: Resource Misuse



1. Some form of alert associated with accessing a resource owned by a service
2. Access request appears in an inbound message in Service Audit Trail
3. Session ID is recorded in STS Audit Trail
4. Session ID is not found in Session Manager History



Conclusions

- Security controls form a hierarchy among multiple governing documents allowing parameterization to propagate throughout the hierarchy to instantiate compliance predicates
- Compliance predicates relate to controls within the same class, as well as to other classes
- Formal proofs can be constructed to verify compliance with components, services, and clouds specified in X-UNITY, which promotes security-awareness of data and processes
- Audit controls have specific importance in cloud computing
 - Information isolation
 - Vulnerability patterns detection using audit trail composition and monitoring
 - Common terminology can be used for SLAs and risk evaluation using matchmaking algorithms
- Continued effort
 - Overhead analysis
 - Specification and verification of cloud architectures with respect to information sharing and flow control
 - Formulation of vulnerability patterns as security patterns for risk assessment, rapid detection and prevention

Questions?