A service dependency modeling framework for Policy-based response enforcement

Nizar Kheir[1,2], Hervé Debar[1], Frédéric Cuppens[2], Nora Cuppens-Boulahia[2], Jouni Viinikka[1]

[1]Orange Labs - Caen - France
[2]Telecom Bretagne - Rennes - France

DIMVA 2009

Need for a Policy-based Response

- Multiple adjustment variables for Information Systems:
  - Performance, Convenience, Reliability, Security, . . .
- Multiple trade-offs between these variables:
  - Sacrifice convenience for a higher security.
  - Privilege Performance over Reliability . . .
- Static design-time adjustements need dynamic run-time adaptation.
- Express dynamic trade-offs using dynamic access control policies.
  - Implement dynamic access control rules.
  - Use of the Organization Based Access Control Model (Or-Bac).

OrBac overview - Policy Derivation

1. Dynamic policy rules using contexts:
   - *Sr(Prohibition, User, Login, Internal_Host, not_Working_Hours).*
2. Simple policy definition using component abstractions:
   - *Empower (Bob, User).*
   - *Consider (telnet, Login).*
   - *Use (Serv1@10.28.43.23, Internal_Host).*
3. Dynamic context evaluation/activation.
   - *Hold (Bob, telnet, Serv1@10.28.43.23, not_Working_Hours).*

Concrete Policy Activation

Is_Prohibited (Bob, telnet, Serv1@10.28.43.23)

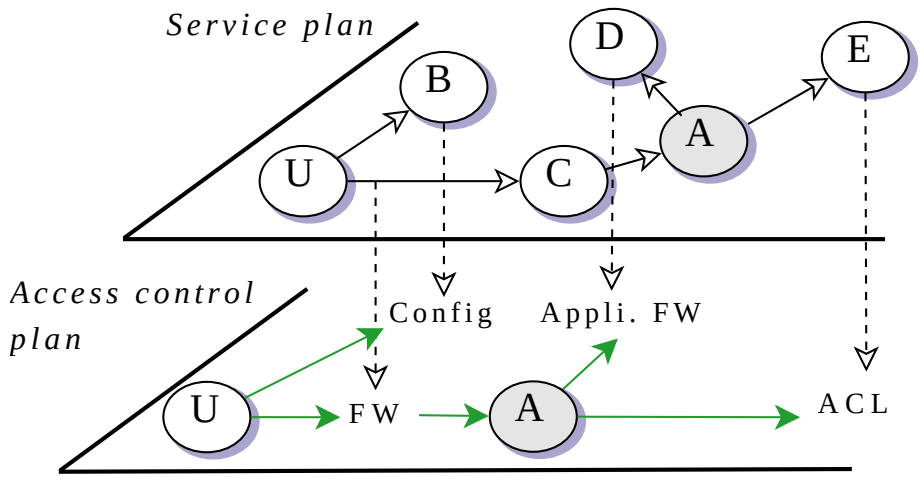
OrBac overview - Threat Response

- Alert recuperation:
  - *Alert (Source, Target, Description)*
- Mapping between alert attributes and concrete policy components:
  - *map_Subject(Source, Subject)*
  - *map_Action(Target.Service, Action)*
  - *map_Object(Target, Object)*
- Checking Threat Contexts:
  - *map_Context (description, Th_Context)*
- Activating appropriate threat contexts:
  - *Hold (Subject, Action, Object, Th_Context)*

Problem Statement

*Alert (Src, Trgt, Descr) ⇒ {Sr (Decision, Subj, Act, Obj)}**

- Challenges of a Policy Decision Process:
  - How to find suitable Policy Enforcement Points (PEPs) ?
  - How to decide about PEPs' capabilities ?
  - How to propose appropriate configurations for each selected PEP ?
- Proposal: Build mappings between service and access control layers.



Outline

1. Policy-based Response
2. A Service Dependency Framework (SDF)
   - Dependency Model Architecture
   - Dependency Model Framework
3. Use of the SDF
4. Conclusion

Properties Expected from the SDF

- The Service Dependency Framework Specifies:
  - Data provided by antecedent services ⇒ Formal service interfaces.
  - Paths through which data is accessed ⇒ Topology representation.
  - When this Data is required ⇒ Modeling service workflow.
  - Why this Data is required ⇒ Dependency Impact representation.
- Formal definition of the SDF ⇒ Use of formal semantics.
- Modularity and Extensibility ⇒ components with specific interfaces.

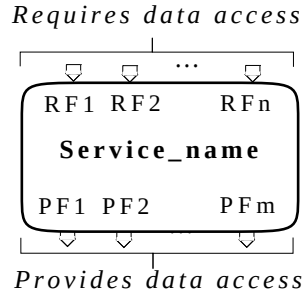
Use of the Architecture Analysis and Design Language (AADL)

AADL fulfills all those requirements through the modeling of service architectures and interactions.

SDF: Modeling services

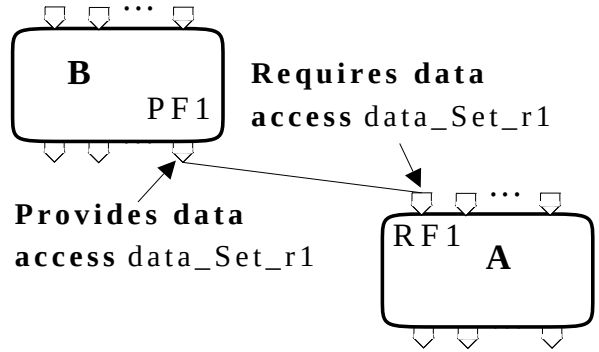
- Abstract components with specific interfaces.
- Interfaces are typed with data sets.

– Implementation of elementary service –
system Service_Name
features
RF_i: **requires data access** $data\_Set\_r_i$;
PF_j: **provides data access** $data\_Set\_p_j$;
end Service_Name;



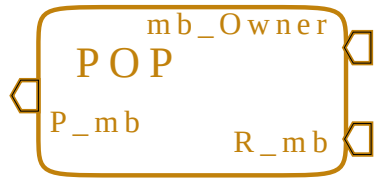
- Dependencies are represented as inter-service connections.

system implementation Dependency_Model.A
subcomponents
A: **system** dependent;
B: **system** antecedent;
connections
const_AB: **data access** B.PF1 → A.RF1;
end Dependency_Model.A;



Case study: Service definition

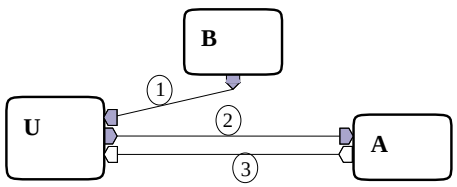
```
package serviceDB
public
– NFS service definition –
system NFS
features
    P_mb: provides data access dataDB::mBox;
end NFS;
– LDAP service definition –
system LDAP
features
    P_a: provides data access dataDB::Account;
end LDAP;
– POP service definition –
system POP
features
    mb_Owner: requires data access dataDB::Account;
    R_mb: requires data access dataDB::mBox;
    P_mb: provides data access dataDB::mBox;
end POP;
end serviceDB;
```



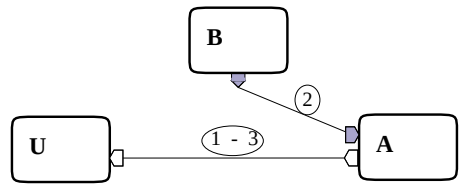
SDF: Service topology

- Shows data paths through virtual connections between service components.
- Three service dependency types:

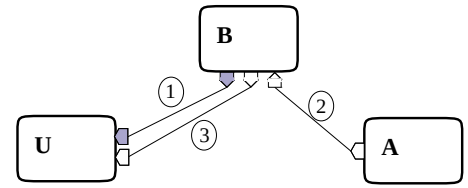
- User-side dependency:



- Service-side dependency:

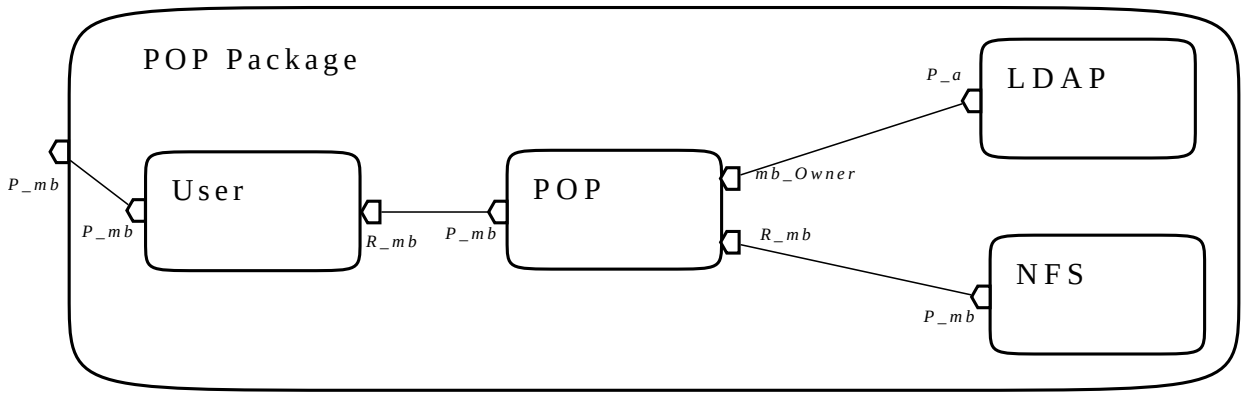


- Proxy-side dependency:



Case study: Service topology

- AADL graphical representation:



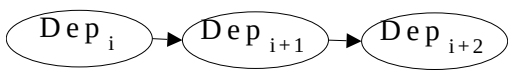
- AADL textual representation:

```
package Pop
 public System POP
  features
    P_mb: provides data access dataDB::mbox;
  end POP;
  private system implementation POP.instance
  subcomponents
    User: system user;
    Ldap: system serviceDB::Ldap;
    NFS: system serviceDB::NFS;
    Pop: system dependent.instance;
  connections
    data access Ldap.P_a⟶ Pop.mb_Owner;
    data access NFS.P_mb⟶ Pop.R_mb;
    data access Pop.P_mb⟶ User.R_mb;
  end POP.instance;
end Pop;
```

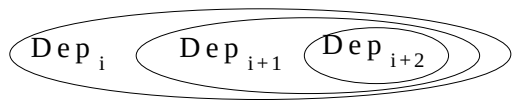
SDF: Service dependency modes

- A dependency mode expresses a service operational phase where a given dependency is required.
- It is associated with an appropriate service interface.
- Dependency mode transitions show dependency sequencing.
- Three dependency sequencing aspects:

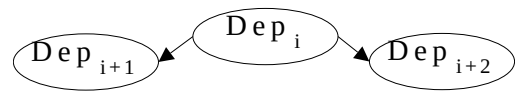
- Stateless sequencing:



- Statefull sequencing:



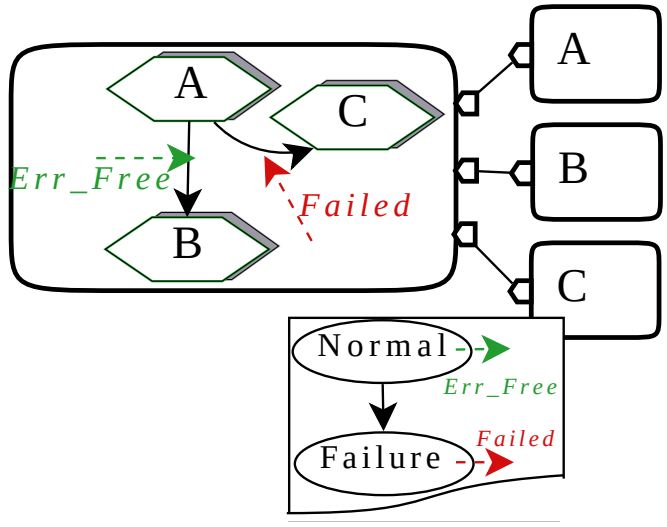
- Alternative sequencing:



SDF: Dependency Impact

- Dependency failures alter normal transitions between dependency modes.
  - A failed dependency can never be satisfied.
  - The dependent service may switch to another dependency mode.
- Use of the AADL *Guard_Transition* constructs to constrain mode transitions.

modes
A: **initial mode**; B: **mode**; C: **mode**;
T1: A -[C1.transit]→B;
T2: A -[C1.Failure_transit]→ C;
annex Error_Model {**
Guard_Transition ⇒
(Iface_A[Failed]) **applies to** T2;
Guard_Transition ⇒
(Iface_A[Error_Free]) **applies to** T1;
**};

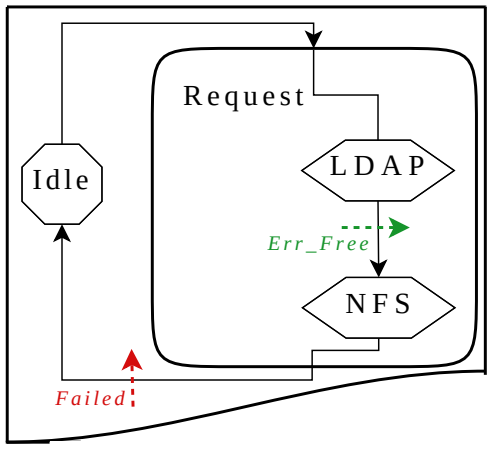


Case study: Modes and Impacts

- AADL textual representation:

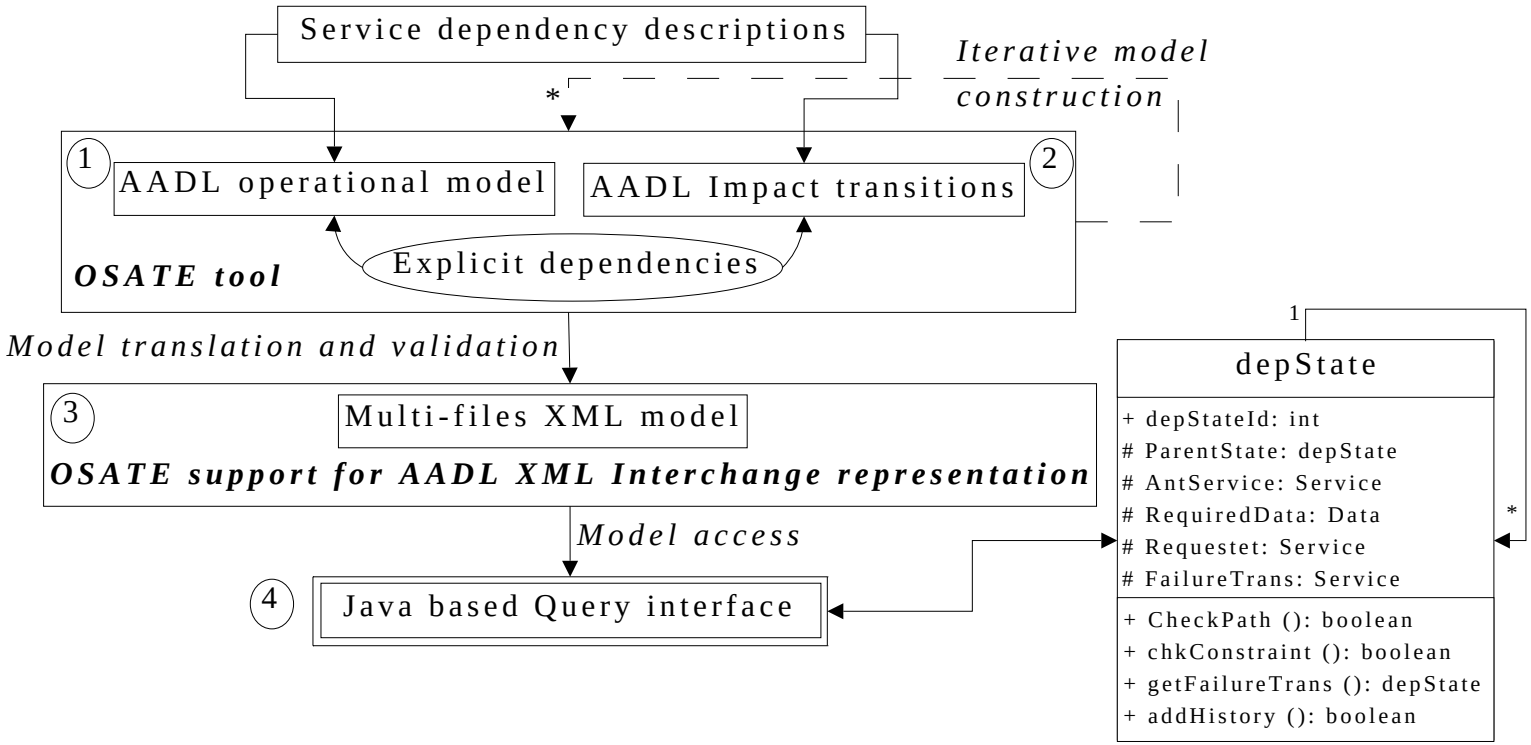
```
package Pop
 public
    ...
    system implementation op_State.Idle
    end op_State.Idle;
    system implementation op_State.Request
      modes LDAP: initial mode;
        NFS: mode;
        Idle: mode;
        T1: LDAP-[C1.transit]→ NFS;
        T2: NFS-[C2.Fail_transit]→ Idle;
      annex Error_Model {**
        Guard_Transition ⇒ (mb_Owner[Error_Free]) applies to T1;
        Guard_Transition ⇒ (R_mb[Failed]) applies to T2; **};
    end op_State.Request;
 end Pop;
```

- AADL graphical representation:

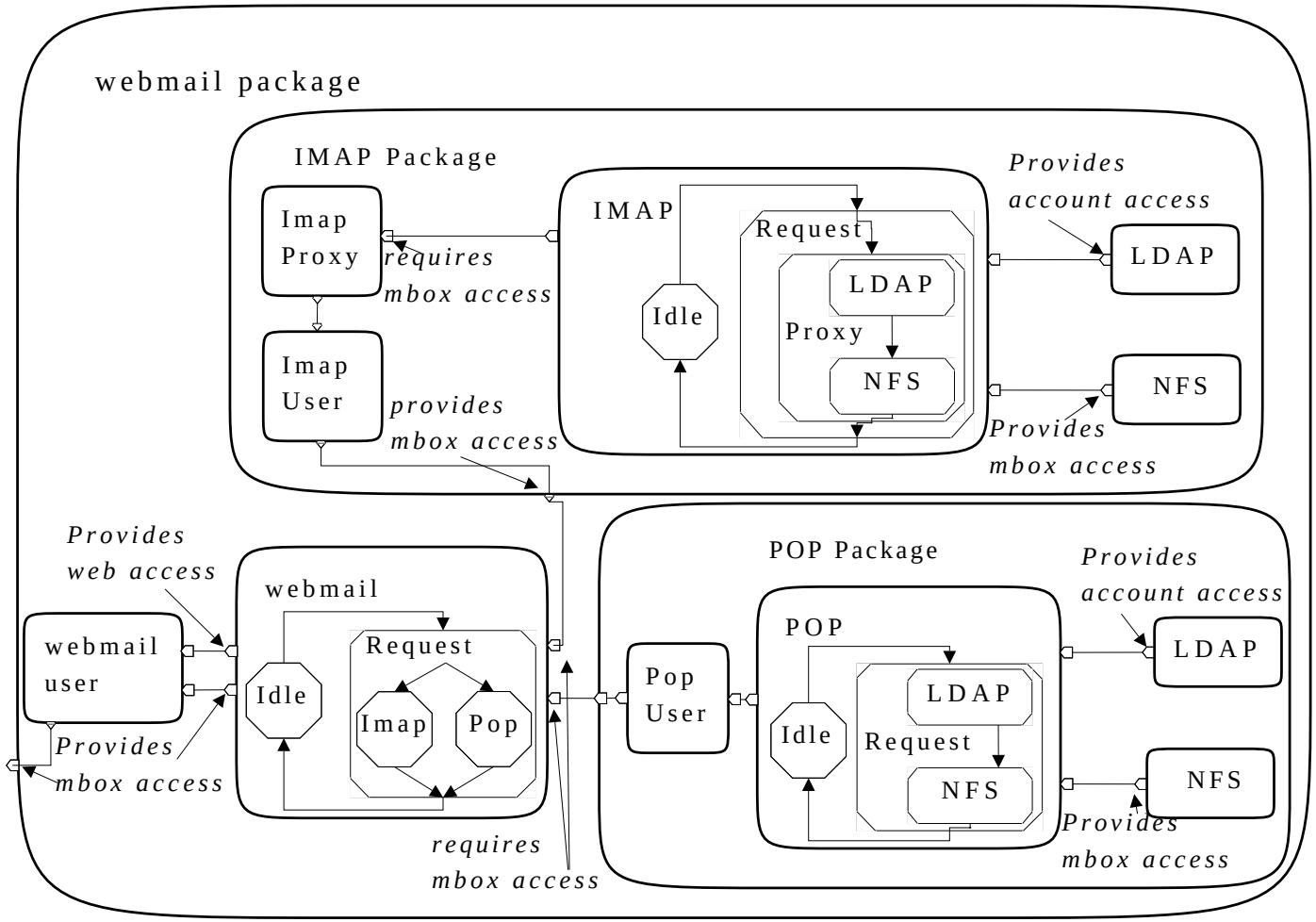


Outline

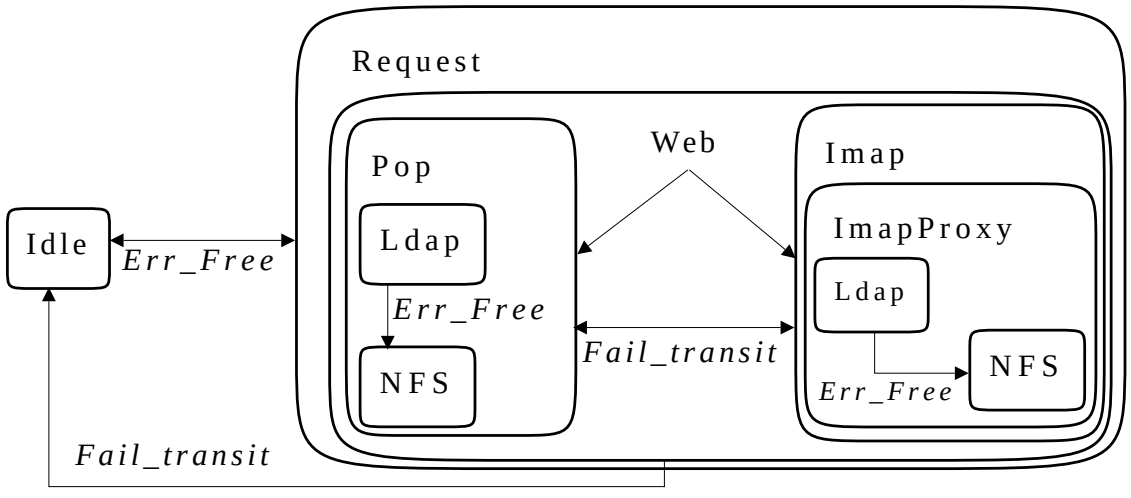
SDF: Dependency Model Framework



Case study: Resulting AADL Model



Case study: webmail DFSM



Outline

1. Policy-based Response
2. A Service Dependency Framework (SDF)
   - Dependency Model Architecture
   - Dependency Model Framework
3. Use of the SDF
4. Conclusion

Process description

- Input: Concrete response rule $Sr(Decis., Subj, Act, Obj)$.
  1. Get Service 'S_{dep}' implementing Act.
  2. Build abstract DFSM for S_{dep}.
  3. Derive concrete DFSM using transitive closure with Sr as input.
  4. Select minimal set of dependency states with:
     1. Case of permission: At least one access path is set.
     2. Case of prohibition: No access path is set.

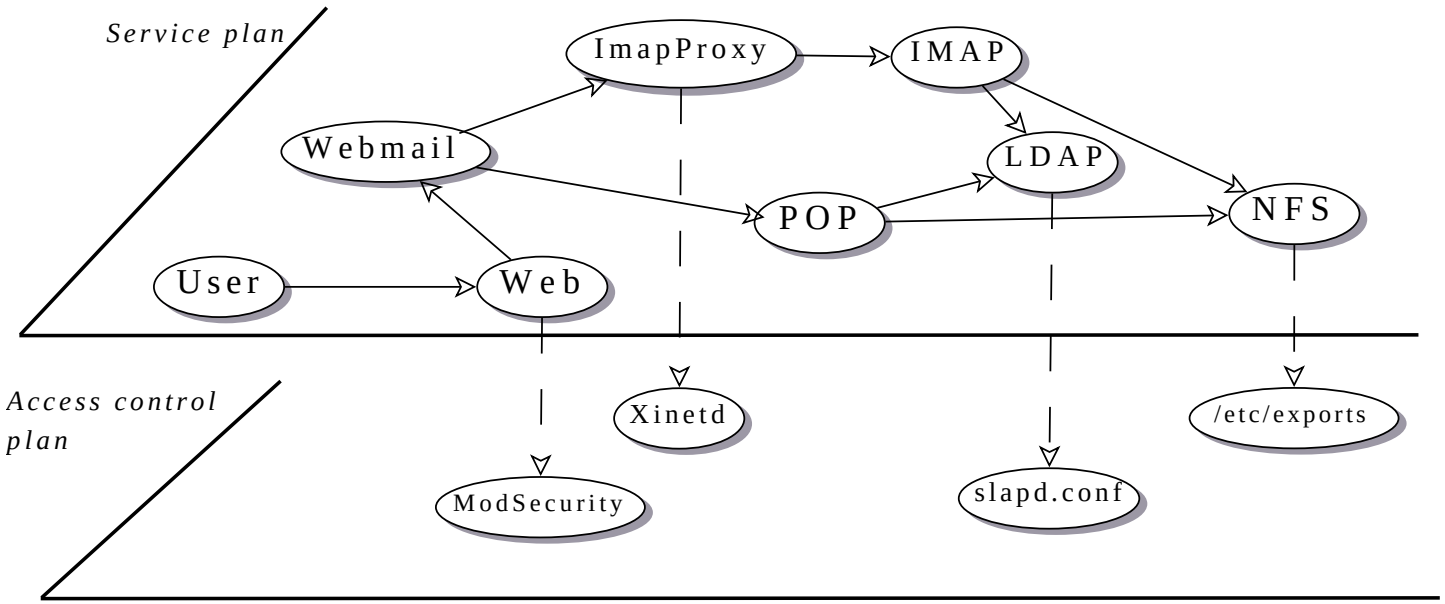
Model simulation - Response proposals

- The Or-Bac based response policy described as:

  – The abstract response rule –
  Sr (prohibition, att_Source, victim_Serv, target_Data, attack_Threat)
  – The Or-Bac Hold fact which transforms alerts into contexts –
  Hold (Subject, Action, Object, Th_Context) :-
  alert (Source, Target, description) &
  map_Subject (Source, Subject) &
  map_Action (Target.Service, Action) &
  map_Object (Target, Object) &
  map_Context (description, Th_Context).

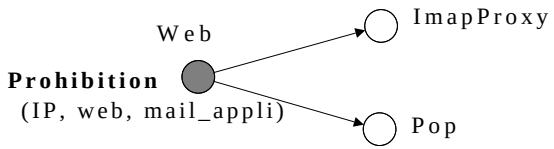
- Attacks tested using this model:
  - Reconnaisance attack against the webmail service.
  - Brute Force attack against the webmail service.
  - Arbitrary code execution against the POP service.
  - Arbitrary code execution against the IMAP service.

Summary of the email test bed



Model outcomes

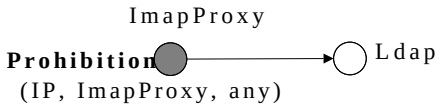
Alert(IP, webmail, -, Reconnaissance attack)
Prohibition (IP, webmail, any)



ModeSecurity: secrule request_uri " *webmail*"\
remote_addr "@streq *IP* " *deny*

- a - Reconnaissance attack

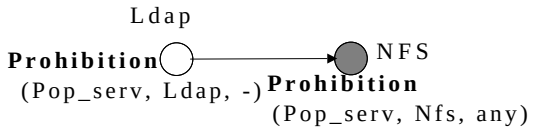
Alert(IP, Imap, -, arbitrary code)
Prohibition (IP, Imap, any)



Xinetd: service *Imap* {no access = *IP*}

- c - Arbitrary Code Execution

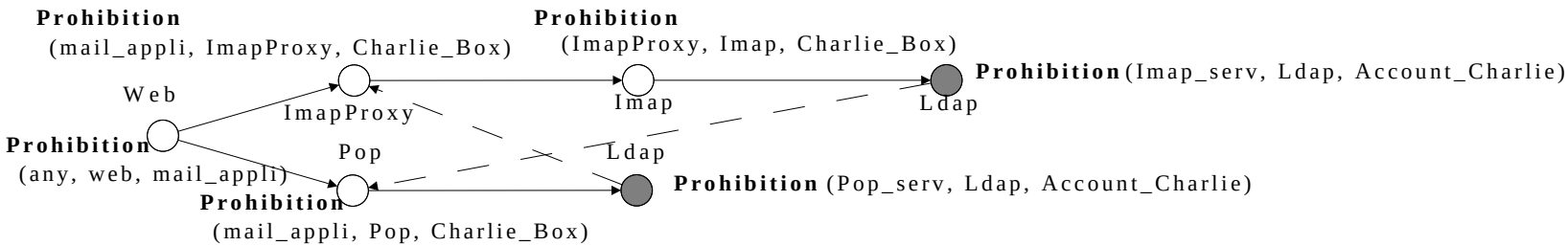
Alert(IP, Pop, -, arbitrary code)
Prohibition (IP, Pop, any)



/etc/exports: rm */home/nfs/mbox IP<pop>*\
(rw, sync, root_squash)

- d - Arbitrary Code Execution

Alert(-, webmail, Charlie Box, Brute Force attack)
Prohibition (any, webmail, Charlie Box)

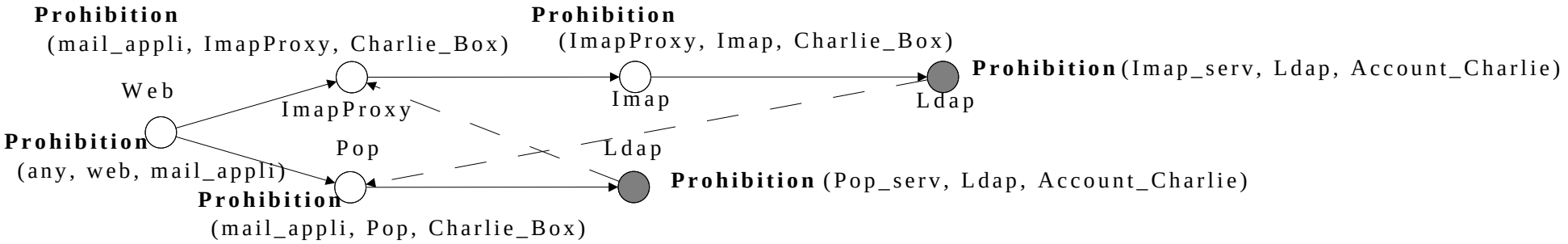


Ldap: access to dn.base = " *cn=charlie, ou=accounts, dc=testbed* " by dn.base = " *cn=Pop,ou=accounts, dc=testbed*" *none*
Ldap: access to dn.base = " *cn=charlie, ou=accounts, dc=testbed* " by dn.base = " *cn=Imap,ou=accounts, dc=testbed*" *none*

- b - Brute Force attack

Model outcomes

Alert(-, webmail, Charlie Box, Brute Force attack)
Prohibition(any, webmail, Charlie Box)



Ldap: access to dn.base = " *cn=charlie, ou=accounts, dc=testbed* " by dn.base = " *cn=Pop,ou=accounts, dc=testbed*" *none*
Ldap: access to dn.base = " *cn=charlie, ou=accounts, dc=testbed* " by dn.base = " *cn=Imap,ou=accounts, dc=testbed*" *none*

- b - Brute Force attack

Conclusion

- A definition of a Service Dependency Framework which provides:
  - Formal classification of dependencies using formal semantics.
  - Abstraction of model components to simplify the representation of complex architectures.
  - A modular and extensible framework.
- A systematic process which uses service dependencies to provide candidate responses.
- Response selection based on minimum configuration changes.

Thank you

Questions ?