BITS Pilani
Pilani | Dubai | Goa | Hyderabad

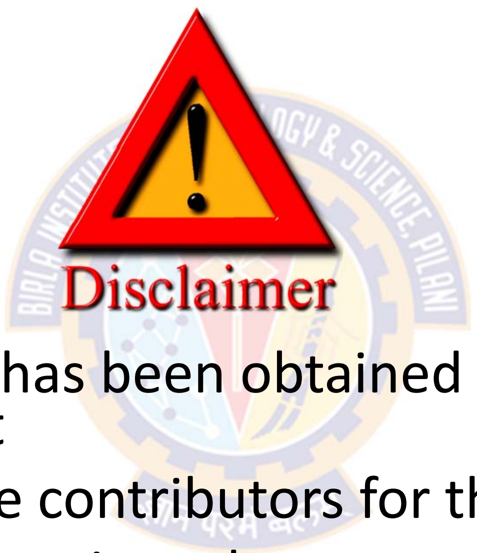
Cyber Security

Security Architecture: Policies, Models and Mechanisms

Dr. Ramakrishna Dantu

Associate Professor, BITS Pilani

Disclaimer and Acknowledgement



- The content for these slides has been obtained from books and various other source on the Internet
- I here by acknowledge all the contributors for their material and inputs.
- I have provided source information wherever necessary
- I have added and modified the content to suit the requirements of the course

Agenda

- Introduction to security policies, models and mechanisms
- The Nature of Security Policies
- Types of Security Policies
- The Role of Trust
- Types of Access Control
- Policy Languages
- The CIA Classification:
  - Confidentiality Policies:
  - Integrity Policies:
  - Availability Policies:

Policy Languages

Policy Languages

Overview

- A *policy language* is a language for expressing a security policy
- Policy language can be high-level or low-level
- High-level policy languages
  - Policy constraints on entities expressed abstractly
- Low-level policy languages
  - Policy constraints expressed in terms of program options, input, or specific characteristics of entities on system

Policy Languages

High-Level Policy Languages

- Constraints are expressed independent of the enforcement mechanisms
  - Constraints restrict entities, actions
- Constraints are expressed unambiguously
  - Such precision requires a precise language, usually mathematical or programmatic formulation of policy
    - Common English is not precise enough.

Policy Languages

High-Level Policy Language - Ponder

- One such policy language is Ponder
- Ponder is a declarative language for specifying security and management policies
- Provides support for several different types of policies:
  - Authorization policies
  - Delegation policies
  - Information filtering policies
  - Obligation policies
  - Refrain policies

Policy Languages

High-Level Policy Language - Ponder

- Entities or subjects are organized into hierarchical domains
  - Network administrators
    - *Domain* is /NetAdmins
    - Subdomain for network admin trainees is /NetAdmins/Trainees
  - Developers for network infrastructure
    - *Domain* is /NetDevelopers
  - Network engineers
    - *Domain* is /NetEngineers
    - Subdomain for network engineer trainees is /NetEngineers/Trainees
  - Routers in LAN
    - *Domain* is /localnetwork/routers
    - Subdomain that is a testbed for routers is /localnetwork/testbed/routers

Policy Languages

High-Level Policy Languages

- Example of how Ponder can be used
  - Consider a policy requiring separation of duty in the issuance of checks
  - The policy, which is to be enforced dynamically, requires that two different members of the /Accounting domain approve the check
  - The Ponder policy specification for this is:

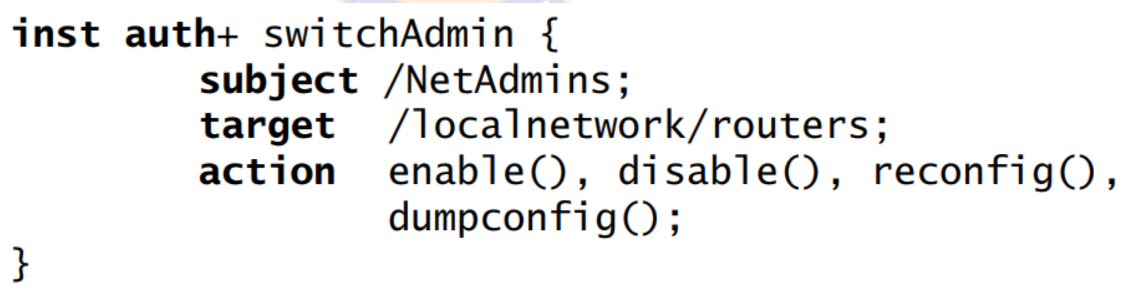
```
inst auth+ separationOfDuty {
    subject  s=/Accountants;
    target   t=checks;
    action   approve(), issue();
    when     s.id <> t.issuerid;
}
```

  - The when constraint requires that the userid associated with the check issuance (t.issuerid) cannot be the accountant who approves the issuance (s.id)

Policy Languages

High-Level Policy Languages

- *Authorization* policy specifications
  - Enforced by controllers associated with the objects
    - These objects, on which actions can be performed, fall into two classes
    - "allowed actions" and "disallowed actions"
  - The following states that network administrators can:
    - enable and disable routers on the local network
    - they can also reconfigure them, and cause them to dump the configuration:



```
inst auth+ switchAdmin {
        subject /NetAdmins;
        target  /localnetwork/routers;
        action  enable(), disable(), reconfig(),
                dumpconfig();
}
```

Policy Languages

High-Level Policy Languages

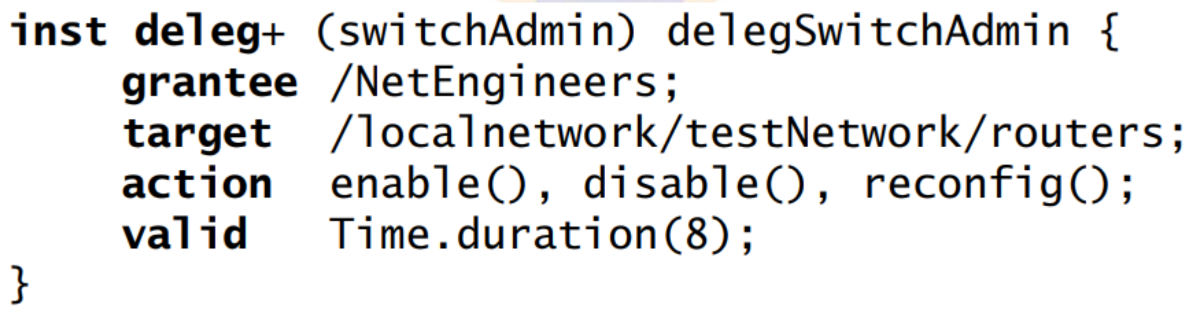
- *Authorization* policy specifications
  - Disallowed actions:
    - The following states that network engineer trainees cannot run performance tests on these routers during the day between 8AM and 5PM

```
inst auth- testOps {
     subject /NetEngineers/trainees;
     target   /localnetwork/routers;
     action   testperformance();
     when     Time.between("0800", "1700");
}
```

Policy Languages

High-Level Policy Languages

- *Delegation policy* specifications
  - Describe the delegation of rights
  - Here, the network engineers are delegated the authority to enable, disable, and reconfigure routers in the testbed
  - The delegation comes from the network administrators, and is good for 8 hours:

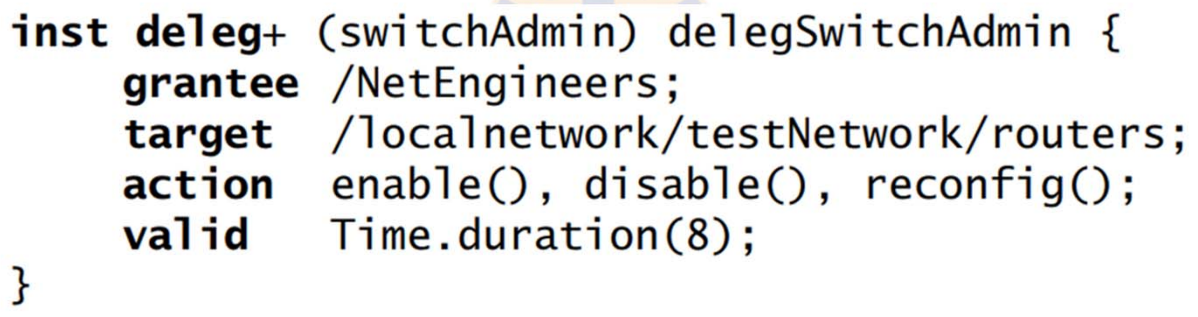


```
inst deleg+ (switchAdmin) delegSwitchAdmin {
     grantee /NetEngineers;
     target  /localnetwork/testNetwork/routers;
     action  enable(), disable(), reconfig();
     valid   Time.duration(8);
}
```

Policy Languages

High-Level Policy Languages

- *Delegation policy* specifications
  - In this policy specification, the "grantee" is the subject (or subject domain) that is the one being given the authority to carry out actions
  - The specification delegates authorizations from the policy switchAdmin
  - Only the authorization for actions enable, disable and reconfig are delegated
  - When a delegation under this policy occurs, it is valid for 8 hours, after which it is automatically revoked



```
inst deleg+ (switchAdmin) delegSwitchAdmin {
     grantee /NetEngineers;
     target  /localnetwork/testNetwork/routers;
     action  enable(), disable(), reconfig();
     valid   Time.duration(8);
}
```

Policy Languages

High-Level Policy Languages

- *Information filtering* policy specifications
  - Control the dissemination of information
  - The policy says that network administrators can dump:
    - everything from the local network routers between 8:00 p.m. and 5:00 a.m., and
    - configuration information from the routers on the local network at any time

```
inst auth+ switchOpsFilter {
     subject /NetAdmins;
     target  /localnetwork/routers;
     action  dumpconfig(what)
             { in partial = "config"; } // default filter
             if (Time.between("2000", "0500"))
                { in partial = "all"; }
}
```

Policy Languages

High-Level Policy Languages

- *Refrain policy* specifications
  - Similar to the authorization denial policy specifications, but that they are enforced by the subjects, not the target controllers
  - For e.g., network engineers cannot send test results to network developers while testing is in progress
    - presumably because it might cause them to take actions that would affect the testing

```
inst refrain testSwitchOps {
     subject   s=/NetEngineers;
     target    /NetDevelopers;
     action    sendTestResults();
     when      s.teststate="in progress"
}
```

Policy Languages

High-Level Policy Languages

- *Refrain policy* specifications
  - The name s represents the domain of network engineers
  - The when constraint holds when the state of the test is "in progress,"
  - When that constraint holds, the policy specification requires that the network engineers refrain from taking the action sendTestResults with the network developers as the target

```
inst refrain testSwitchOps {
     subject  s=/NetEngineers;
     target   /NetDevelopers;
     action   sendTestResults();
     when     s.teststate="in progress"
}
```

Policy Languages

High-Level Policy Languages

- The *obligation policy* specification
  - Requires that specific actions be taken when certain events occur
  - For example, what happens when three consecutive login attempts fail?
    - Net security admins will disable account and log event

```
inst oblig loginFailure {
    on       loginfail(userid, 3);
    subject  s=/NetAdmins/SecAdmins;
    target   t=/NetAdmins/users ^ (userid);
    do       t.disable() -> s.log(userid);
}
```

Policy Languages

High-Level Policy Languages

- The *obligation policy* specification
  - On the third failure (loginfail(userid, 3)), the network security administrators (who are a subset of the network administrators) will disable the account (t.disable), and then make an entry into the log in the network security administrator’s domain (s.log(userid))

```
inst oblig loginFailure {
     on         loginfail(userid, 3);
     subject    s=/NetAdmins/SecAdmins;
     target     t=/NetAdmins/users ˆ (userid);
     do         t.disable() -> s.log(userid);
}
```

innovate achieve lead

Thank You!