Securing Programmable Routers with Access Control

Jun Gao
jun.gao@cs.cmu.edu

Computer Science Department
Carnegie Mellon University

Sept. 7th, 2001, Student Seminar
In Partial Fulfillment of the Speaking Requirement
Outline

- Programmable networks
  - Security challenges
- Secure router architecture
  - Darwin programmable router
  - Access control mechanisms
  - Security policy management
- Related work
- Conclusions
Internet Router Architecture

- **Control Plane**
  - Routing process
  - Build routing table
  - Other control entities
  - E.g. monitoring

- **Data Plane**
  - Forwarding engine
  - Lookup routing table
  - Output queue
Traditional Router Design

- Closed router architecture
  - Not programmable
  - Service Providers:
    Execute fixed software
  - End Users:
    Minimal control over data handling
- Difficult to deploy new services, protocols
  - Consensus-based standardization
  - Vendor-based implementation
Programmable Networks

- **Open router architecture**
  - Can be programmed by third-party
    - Resembles the model of today’s PC
  - Extensible functionalities
    - Fast deployment of new services, protocols
    - Customizable router behavior for applications

- **Degree of programmability**
  - Active packets
    - Routers execute code carried in each packet
  - **Active extensions**
    - Outband code segments injected to routers
Programmable Networks Challenges

- **Performance**
  - Routers’ first priority is forwarding packets
  - Control plane active extensions introduce less overhead

- **Safety**
  - Safe execution of untrusted code
  - Many mechanisms: Sandboxing, SFI, PCC, etc.

- **Security**
  - Prevent network services from being disrupted by active extensions
Roadmap

- Programmable networks
  - Security challenges
- Secure router architecture
  - Darwin programmable routers
  - Access control mechanisms
  - Security policy management
- Related work
- Conclusions
Darwin Programmable Router Architecture

- Routing
- Signaling
- Delegates
- Routing control interface
- Local resource manager
- Input classifier
- PPM
- Route lookup
- Output classifier
- Scheduler

Other signaling daemon

Other routing, daemon

Other delegates

Data Plane

Control Plane
Router Control Interface (RCI)

- Collection of functions that operate on flows
  - Classify packets into flows based on filter_spec
    - Filter_spec: fields from packet headers
    - [ src IP, dst IP, protocol ID, src Port, dst Port, app ID ]
    - E.g., [ 128.2/16, 207.25.71/24, 6, 0, 80, 0 ]
      All HTTP traffic from CMU to cnn.com

- Three types of methods available to delegates
  - Flow manipulation
  - Manage resource allocation to flows
  - Allow special flow processing

- Delegates pose threats to routers by abusing RCI
Gain access to victim user’s traffic
  - Install filter spec to match victim’s traffic

Tamper services received by this traffic
Protect Programmable Routers

- Delegates disrupt router services by abusing router resources
  - No control of what delegates can do on a router
- Traditional OS security mechanisms
  - Use access control list (ACL) to protect files, directories
    - `<user, file, access rights>`
      - `jungao` `paper.tex` `rw-`
      - `others` `paper.tex` `r--`
    - OS checks the ACL when file being accessed by a user
- Two types of router-unique resources
  - Link Bandwidth
  - User Traffic
Darwin Router with Access Control

- ACL-based scheme to protect resources
  - Similar to the file system protection mechanism
- Delegates are assigned access permissions on resources
- Router checks permission whenever resources being accessed
Roadmap

➢ Programmable networks
  ➢ Security challenges

➢ Secure router architecture
  ➢ Darwin programmable routers
  ➢ Access control mechanisms
  ➢ Security policy management

➢ Related work

➢ Conclusions
Delegates’ Bandwidth Manipulation

- Hierarchical bandwidth reservation and sharing
  - Bandwidth \(\Rightarrow\) resource tree
- Delegates manipulate bandwidth via RCI operations on the tree
  - Reserve, release bandwidth
  - Example RCI function
    - \texttt{create\_node(parent\_node, QoS\_para)}
- Violation example
  - Bandwidth stealing
Access Control of Bandwidth

- Assign access rights to delegates on resource nodes
  - Operations allowed only when possesses proper permission
- Access permissions on resource nodes
  - Bandwidth allocation
    - `create`, `modify`, `delete`
  - Monitoring bandwidth usage
    - `retrieve`, `monitor`
  - Service flows
    - `use`
- Example
  - `< del_1, node_1, cmd--u >`
Bandwidth Protection Example

Delegates

Access Control List

<table>
<thead>
<tr>
<th>Delegate ID</th>
<th>node0</th>
<th>node1</th>
<th>node2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>cmdrnu</td>
<td>cmdrnu</td>
<td>cmdrnu</td>
</tr>
<tr>
<td>D1</td>
<td>------</td>
<td>cmdrnu</td>
<td>----rn-</td>
</tr>
<tr>
<td>D2</td>
<td>------</td>
<td>----rn-</td>
<td>cmdrnu</td>
</tr>
</tbody>
</table>

create, modify, delete, retrieve, monitor, use
Delegates’ Operations on Traffic

- **Access traffic**
  - Install filter specs that describe traffic
  - `flow_id = add_filter(filter_spec)`
  - Traffic $\rightarrow$ flows

- **Provide services to flows**
  - Quality of Service (QoS)
    - Provide bandwidth guarantees to flows
    - `add_filter_to_node(flow_id, node_id)`
  - Special flow processing
    - E.g., Tunnel a flow; drop a flow’s packets
    - `add_filter_to_action_instance(flow_id, action_id)`
Protect Access to Traffic

- Control what flows can be identified
  - Filter envelopes define the set of filters can be installed
    - Resemble filter specs
    - For each field, specify a set instead of just one value
  - Delegates are assigned filter envelopes
    - New filter can be installed only if within envelope

- Example
  - `<del_1, [128.2/16, 0, 6, 0, (80, 20), 0]` >
    - HTTP and FTP data traffic from CMU
  - `del_1` can install `[128.2.205.111, 0, 6, 0, 80, 0]`
    - HTTP traffic from my machine
Protect Services to Traffic

- Control what services can be applied to flows
  - Delegates are assigned access rights on flows
    - Operations allowed only if corresponding bits are on
  - Access permissions on flows
    - Provide bandwidth guarantee
      - q: allocate bandwidth for flows
    - Packet processing
      - local: only local effect, e.g., dropping
      - alter: modify packet contents, e.g., transcoding, encryption
      - route: change packet forwarding, e.g., tunneling, redirecting

- Example
  - <del_1, flow_1, -l--, (DROP)>
Traffic Access Control Example

Flow 1: (S1, D1)
Flow 2: (S2, D2)

Delegate1 on R0
- Filter envelope: (S1, D1)
- Access control list
  - <del_1, flow_1, ---r, (TUNNEL: R0, R1)>

Delegate1 prevented from
- Accessing other flows, e.g. Flow2
- Taking other actions on flow1, e.g., dropping Flow1’s packets
Roadmap

- Programmable networks
  - Security challenges
- Secure router architecture
  - Darwin programmable routers
  - Access control mechanisms
  - Security policy management
- Related work
- Conclusions
Access Control Policy Setup

- Access control policy making
  - Policy manager (PM) determines policy per delegate
  - Router installs and enforces policy

- Policy distribution schemes
  - Each router retrieves policy from PM
    - Introduces complexity to both routers and PM
  - Delegate Initiator (DI) retrieves policy from PM and sends to routers
    - A variation of Kerberos
Policy Exchange Protocol

1. DI $\rightarrow$ PM
   - $[\text{request }]_{\text{DI's key}}$

2. PM $\rightarrow$ DI
   - $[\text{session key, policy }]_{\text{DI's key}}$
   - $[\text{session key, policy }]_{\text{router's key}}$
     - Session ticket

3. DI $\rightarrow$ R
   - $[\text{code, policy }]_{\text{session key}}$
   - Session ticket

4. R $\rightarrow$ DI
   - $[\text{confirmation }]_{\text{session key}}$

5. R $\leftarrow$ PM
   - E.g. for policy revocation
Implementation

- Security components are implemented as part of the Darwin programmable router system
- **Router Security Manager (RSM)**
  - Receives delegate code and security policy
  - Creates a policy file for each delegate on each interface
  - Instantiates delegates
- **Access Control Manager (ACM)**
  - Intercepts each RCI call issued by delegates
  - Checks corresponding policy file to determine whether the operation is allowed
Related Work

- **Active Net Security Working Group**
  - Outlines the security components required in an active network node

- **Seraphim**
  - Flexible architecture to accommodate active policies for AN

- **SANE**
  - Comprehensive system ensuring nodes operating in trusted states

- **SQoSH**
  - Infrastructure providing support of controlled access to allocations of system resources
Conclusions

- Designed and implemented a secure programmable router architecture
  - Prevent active extensions’ security violation
- Use access control list (ACL) to protect router specific resources
  - Bandwidth → Resource Tree
  - User traffic → Flows/filters
- Security policy managed by trusted Policy Manager
  - Separation of policy making from policy enforcement ensures good performance
Reference


http://www.cs.cmu.edu/~jungao/pub/