SELinux support for RDMA

Dan Jurgens

Sept 23, 2015

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

Introduction

- SELinux is a Mandatory Access Control (MAC) scheme for Linux
  - Central policy is loaded upfront into the kernel
    - Standard policies are typically provided by the Linux distribution
  - Applications cannot override or modify this policy

- Benefits
  - Differentiate a user from the applications that the user runs
  - Restrict application access only to what is required to perform its task
  - Allow granular policy segregation
  - Example
    - Run 2 instances of a Web Server: “top-secret” and “standard”
    - Each server can only
      - Receive traffic from specific network interfaces
      - Open sockets on specific ports
      - Serve files from specific directories
      - Communicate only with specific peer addresses

- Type enforcement is the main security mechanism used by SELinux

Type Enforcement (TE)

- Applies to all user-visible kernel entities
  - E.g., processes, files, IPC objects, sockets
- Each entity is associated with:
  - A security descriptor
    - Assigned upon creation
    - May be modified based on policy
  - A class and a set of operations
    - Stems from the type of object
    - E,g., a socket can send() and recv()
- TE defines what a <subject> can do on an <object> based on their security descriptors
  - Specified by a policy of access rules
  - Enforced when accesses are made
- Security descriptors
  - Identify the user, role, type, and optionally security level+class of an object
  - Specified by a variable-length string: “user:role:type[:level]”
- Policy rules
  - Specify which source tag can access which target tag and for what operations
    - E.g., “allow source_t target_t:class { [op1] [op2] … }”
  - Typically, only the ‘type’ (a.k.a ‘Domain’) portion of the tag is mentioned

SELinux Network Security

- Network object labeling
  - Interfaces
    - E.g., eth2
    - Used in the past for packet tagging
      - Today packets are tagged by network traffic labeling
  - Nodes
    - Label IPv4/6 addresses and network masks
  - Ports
    - Label TCP/UDP port numbers
  - Sockets
    - Usually inherit the security descriptor of the creating process

- Network traffic labeling
  - Internal labeling
    - Tag traffic according to local OS policies
    - The de-facto standard is SECMARK
      - Extends standard iptables/netfilter to mark packets with security descriptors
  - Labeled networking
    - Labels on the wire
    - Each local system interprets the label to enforce is MAC policies
    - Supported schemes
      - Labled IPSec
      - CIPSO

- SELinux network policies define
  - What a process can do with network objects
    - For example: allow ‘ftp_t’ (the FTP process) to bind a socket to ‘ftp_data_port’ (TCP port 20)
  - What traffic a process can send/receive

- Policy enforcement
  - Object policies are enforced during system calls
  - Traffic policies are enforced per-packet

RDMA Network Objects

- Verbs objects (PDs, QPs, CQs, SRQs, MRs, etc.)
  - Do not have well-known names
    - The user doesn't even control them
  - Observation: granular MAC control policies for Verbs resources doesn't make much sense
    - E.g., allow a process to modify QP253 to RTR???

- RDMA-CM objects (RDMA IDs)
  - Similar to sockets
    - ServiceID port-spaces map TCP/UDP ports spaces
    - Similar semantics and operation
  - However
    - Actual data path governed by Verbs objects
    - There are RDMA applications that don't use RDMA-CM at all
  - Observation: RDMA-CM security policies may provide socket-like MAC control for CM service IDs
    - However, it does not provide a stronger security model
    - Simple policies are still required for P_Key enforcement

RDMA Network Objects (cont.)

- Interface objects (RDMA devices and ports)
  - Associated with every communication end-point (QP)
  - Observation: may be used for object-based policies
    - But benefit is not clear
    - Network labeling (see below) is a much better alternative
- Node objects (GIDs)
  - May be treated similarly to IPv6 addresses
  - Not always used
  - Performance penalty for UD sends (to verify address handle)
  - Does not cover UD reception
  - Observation: GIDs do not seem to be suitable for an object-based policy

RDMA Traffic Labeling

- Applications initiate IO directly on HW endpoints (QPs)
  - HW generates packets
    - Kernel is completely by-passed
  - Observation: arbitrary internal labeling or labeled networking is not an option

- RDMA traffic labeling must stem from HW architectural attributes
  - Network addresses (GIDs, LIDs) are not suitable
  - Queue keys (Q_Keys) are not suitable
  - Observation: partitions are the natural candidate
    - P_Key values are held in HCA partition tables
      - Populated by privileged network Subnet-Manager (SM)
    - P_Keys are carried on the wire of every data packet
      - The only exception is subnet datagram packets (SMPs), which are not accessible to applications
    - Every QPs is associated with a P_Key value
      - Determined by an index into the partition table
    - Partitions are strictly enforced at all times

Fundamental RDMA SELinux Support

- RDMA network security based on partitioning
  - Host kernels control the association of P_Key values with security descriptors
  - SM configuration and P_Key assignment determined by network administrator
    - SELinux policies may be used to control which processes can access the SMI
      - E.g., only SM and tools processes

- Object labeling
  - Associate QPs and RDMA IDs with a security descriptor
    - Inherited by the creating process in the absence of a specific policy
  - P_Key value labeling
    - Associates a P_Key value with a security descriptor
    - System object descriptors are a good example (like network interfaces or nodes)
      - “system_u:object_r:rdma_partition_default_t”
      - “system_u:object_r:rdma_partition_topsecret_t”
  - Other objects not labeled

Fundamental RDMA SELinux Support (cont.)

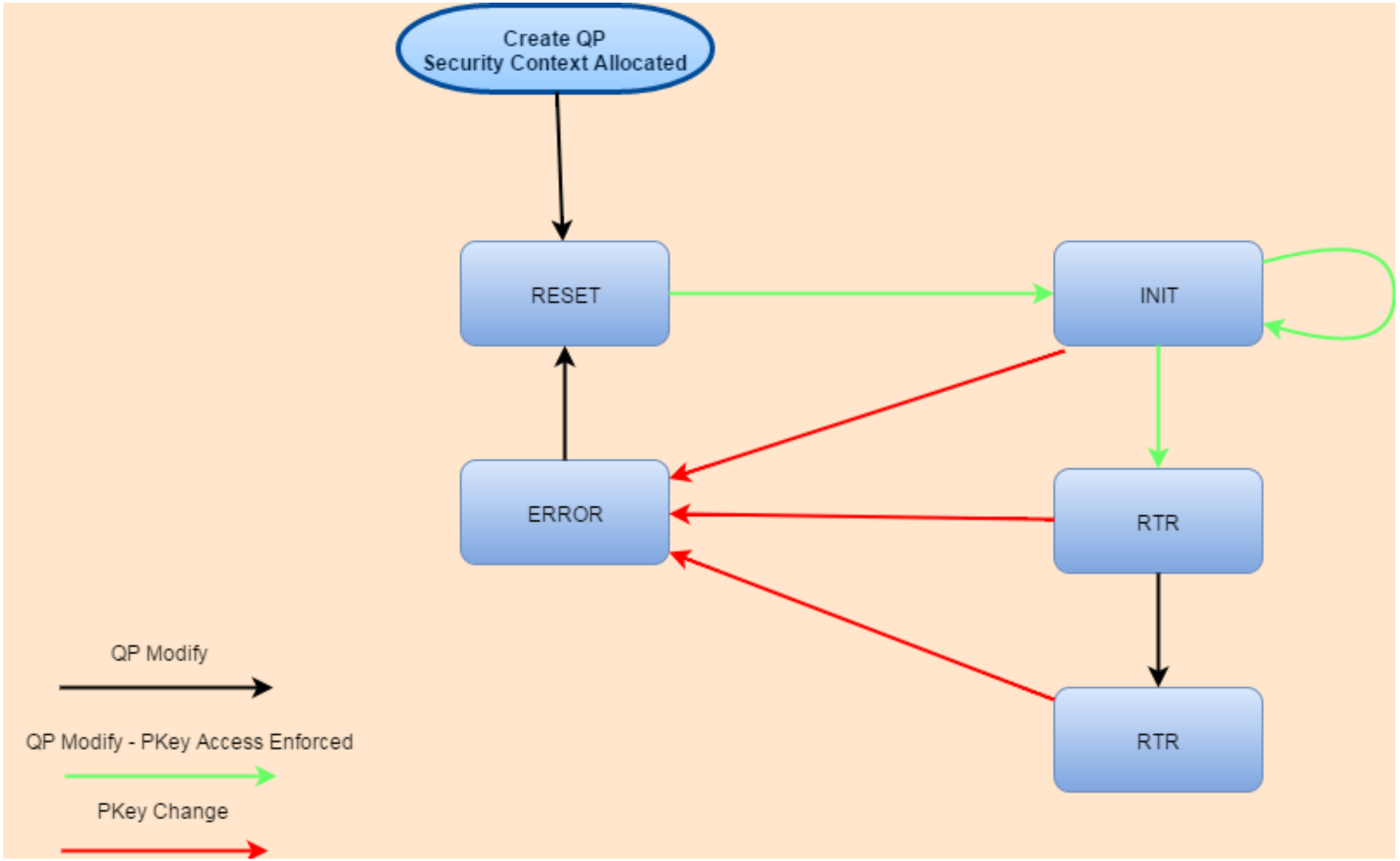
- Traffic labeling
  - Network labeling based on P_Key values
- Policies
  - Allow a process access to a P_Key value
    - E.g., “allow hpc_default_t rdma_partition_default_t : rdma_partition { modify }”, where
      - ‘hpc_default_t’ is the QP / RDMA_ID domain (type) inherited from the creating process
      - ‘rdma_partition_default_t’ is a partition security descriptor domain
      - ‘rdma_partition’ indicates that the subject is of partition type
      - ‘modify’ indicates that the QP is allowed to modify to reference the corresponding partition tag
  - Allow a process access to the SMI
- Partition enforcement
  - QP partitioning enforced at all times
    - Upon QP creation, a violation shall result in an immediate error
    - During runtime
      - Any runtime violation due to policy changes or P_Key value changes shall transition the QP into ERROR state
  - RDMA-ID
    - All ingress/egress CM MADs shall be checked according to the partition policy
    - Any violation shall result in an immediate packet drop
  - umad interface for GMPs

Enforcing SELinux Policy on IB Partitions

- Enforce access to IB partitions with access controls on Pkeys.
  - Label Pkeys in the SELinux policy.
  - When QP partition key settings are changed access to the Pkey is enforced.
  - QP inherits it's creating tasks' security context.

- Changes required.
  - New LSM hooks for managing the QP security field and checking Pkey permissions.
  - Add SELinux kernel support for pkey labels and access control.
  - Changes in ib_core kernel module to enforce QP access to Pkeys and incoming and outgoing management datagrams.
    - Changes are device independent.
  - SELinux user space utilities modified to support Pkey labeling in the policy language.
  - Refpolicy changes to label the Pkeys.

QP State Transitions



Implementation Details

- Added a security field to the ib_qp structure.
- Added new LSM and SELinux hooks to allocate the security structure and free it.
  - Allocation happens at QP create, and it's security context is set to that of the calling task.
  - De-allocation happens during QP destroy.
- Added support to SELinux policy language to label P_Keys based on their value.
  - Similar to port labeling, allowing for individual P_Keys or ranges.
    - pkeycon <pkey_number> gencontext(<label>)
    - pkeycon <low_pkey_number>-<high_pkey_number> gencontext(<label>)
    - Similar to port labeling.
- Added one new LSM and SELinux hook that takes the P_Key value and a QP security context to check for permission.
  - The hook is executed in ib_modify_qp whenever IB_QP_PKEY_INDEX is set in the attribute mask.
  - It is also run against all QPs on a port when the P_Key table changes
    - This requires keeping a list of QP using a particular P_Key index on each port.
    - If permission is not allowed for the new P_Key in the index the QP is moved to the error state.

P_Key Labeling Syntax

```
attribute pkey_type;
attribute protected_pkey_type;
attribute unprotected_pkey_type;

type pkey_t, pkey_type;
sid pkey gen_context(system_u:object_r:pkey_t,s0)

type staff_allowed_pkey_t, pkey_type, protected_pkey_type;
type admin_allowed_pkey_t, pkey_type, protected_pkey_type;
type default_pkey_t, pkey_type, unprotected_pkey_type;

pkeycon 65535 gen_context(system_u:object_r:default_pkey_t,s0)
pkeycon 32769 gen_context(system_u:object_r:staff_allowed_pkey_t,s0)
pkeycon 32770 gen_context(system_u:object_r:admin_allowed_pkey_t,s0)
```

Pkey Allow Syntax

allow sysadm_t default_pkey_t:rdma_pkey modify;
allow staff_t default_pkey_t:rdma_pkey modify;

allow sysadm_t admin_allowed_pkey_t:rdma_pkey modify;
allow staff_t staff_allowed_pkey_t:rdma_pkey modify;

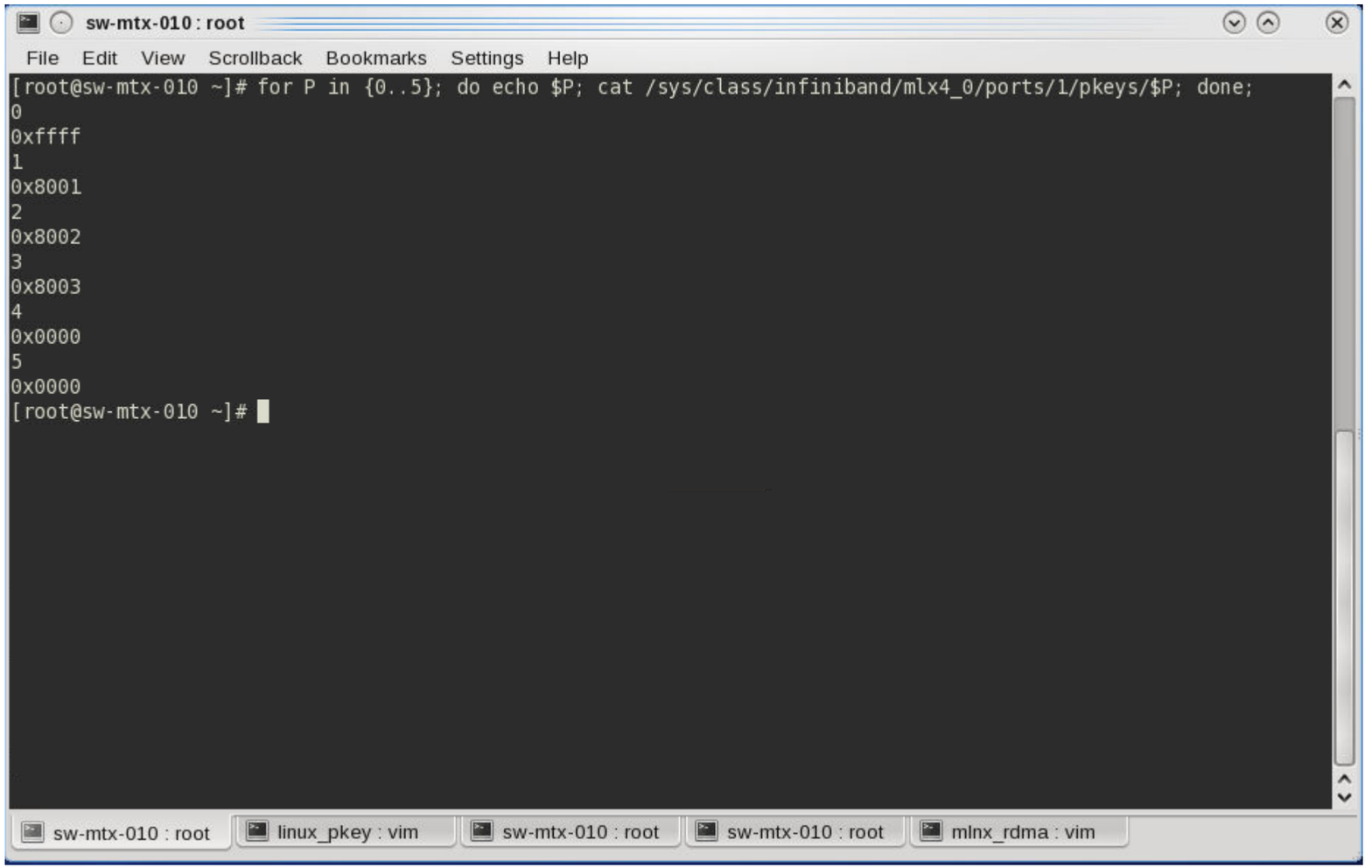
Submission Plan

- First new LSM hooks must be added to the upstream Linux kernel.
- After that SELinux kernel and users space changes can be submitted.
- After that the reference policy changes to provide default rules for Pkey access can be submitted.

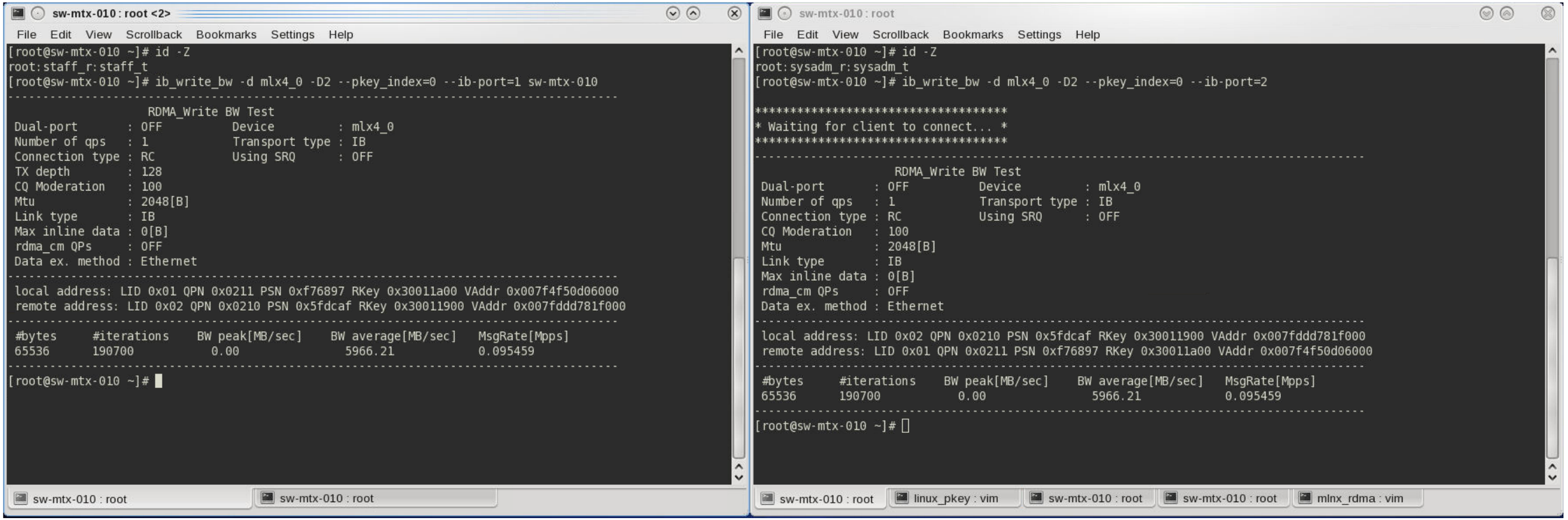
Demonstration Configuration

- Two roles
  - Staff_r
  - Admin_r
- Four available partitions
  - Default (0xFFFF) – both allowed
  - Staff allowed (0x8001)
  - Admin allowed (0x8002)
  - Neither allowed (0x8003)

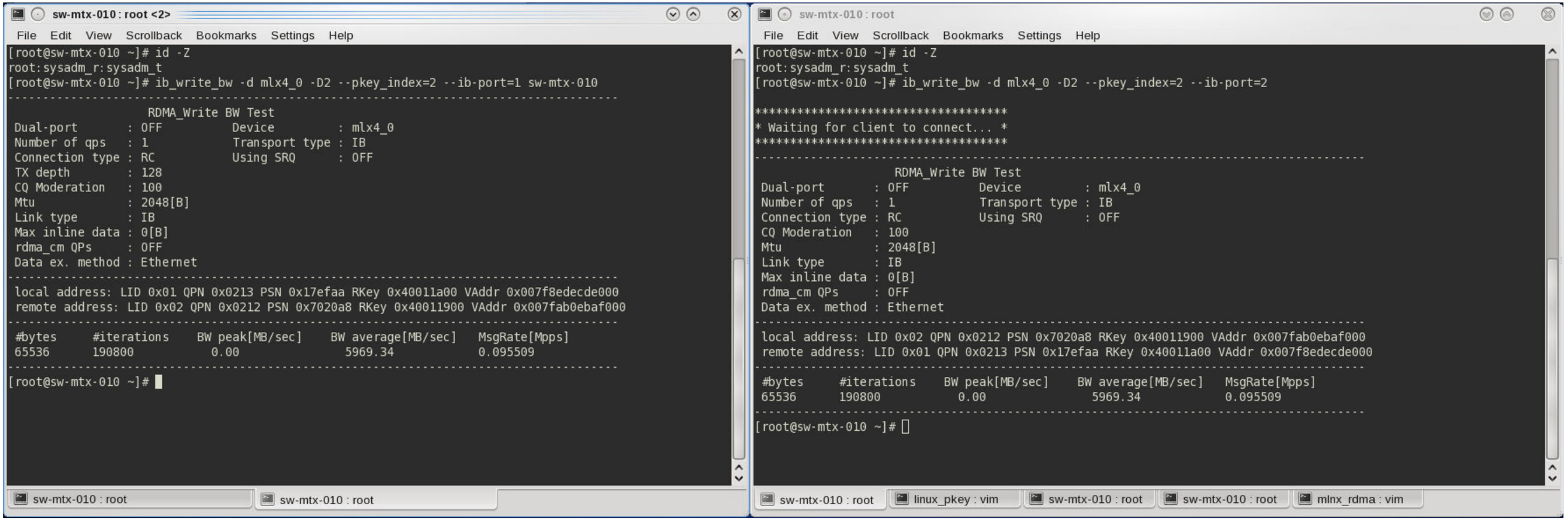
Pkey Table


sw-mtx-010 : root
File Edit View Scrollback Bookmarks Settings Help
[root@sw-mtx-010 ~]# for P in {0..5}; do echo $P; cat /sys/class/infiniband/mlx4_0/ports/1/pkeys/$P; done;
0
0xffff
1
0x8001
2
0x8002
3
0x8003
4
0x0000
5
0x0000
[root@sw-mtx-010 ~]#
sw-mtx-010 : root
linux_pkey : vim
sw-mtx-010 : root
sw-mtx-010 : root
mlnx_rdma : vim

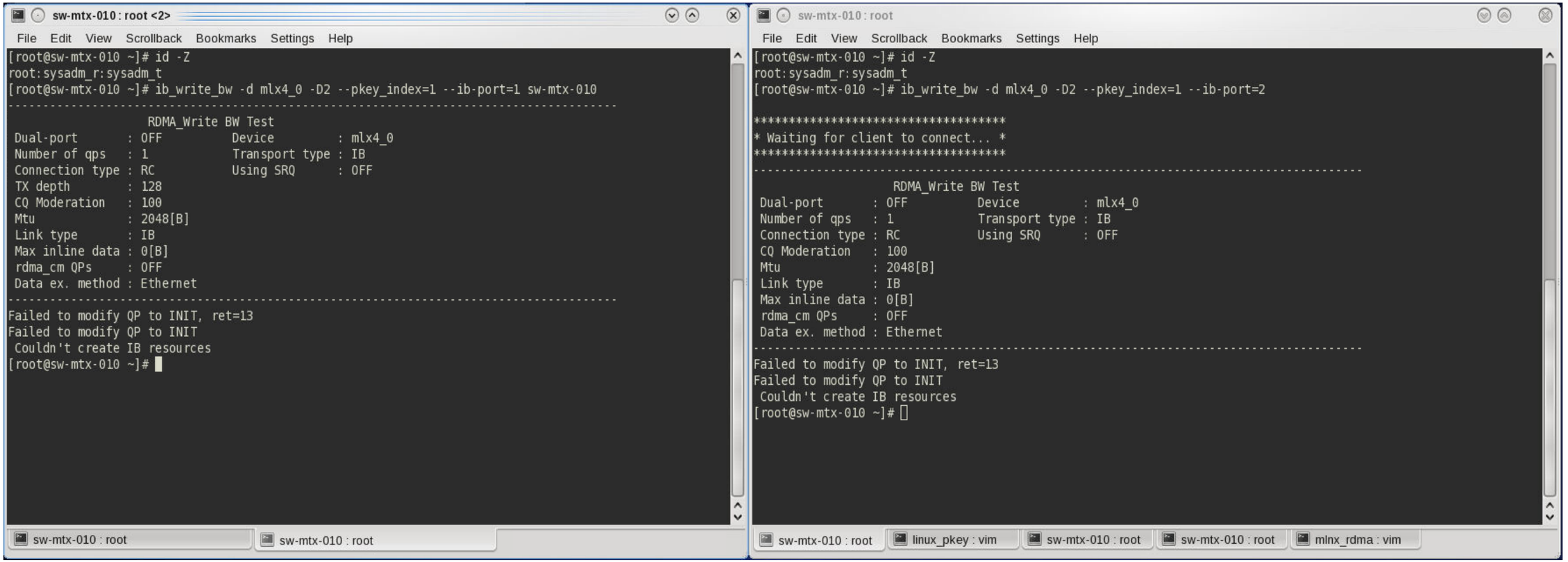
Admin to Staff on Default Partition



Admin to Admin on the Admin Partition

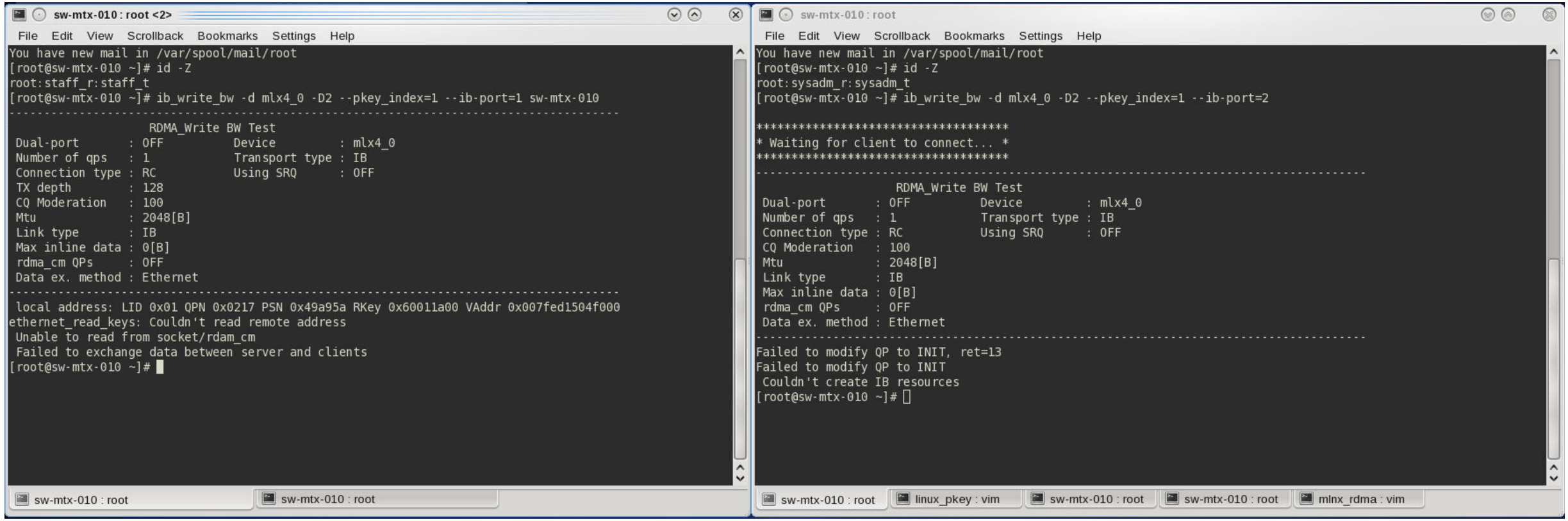


Admin to Admin on Staff Partition



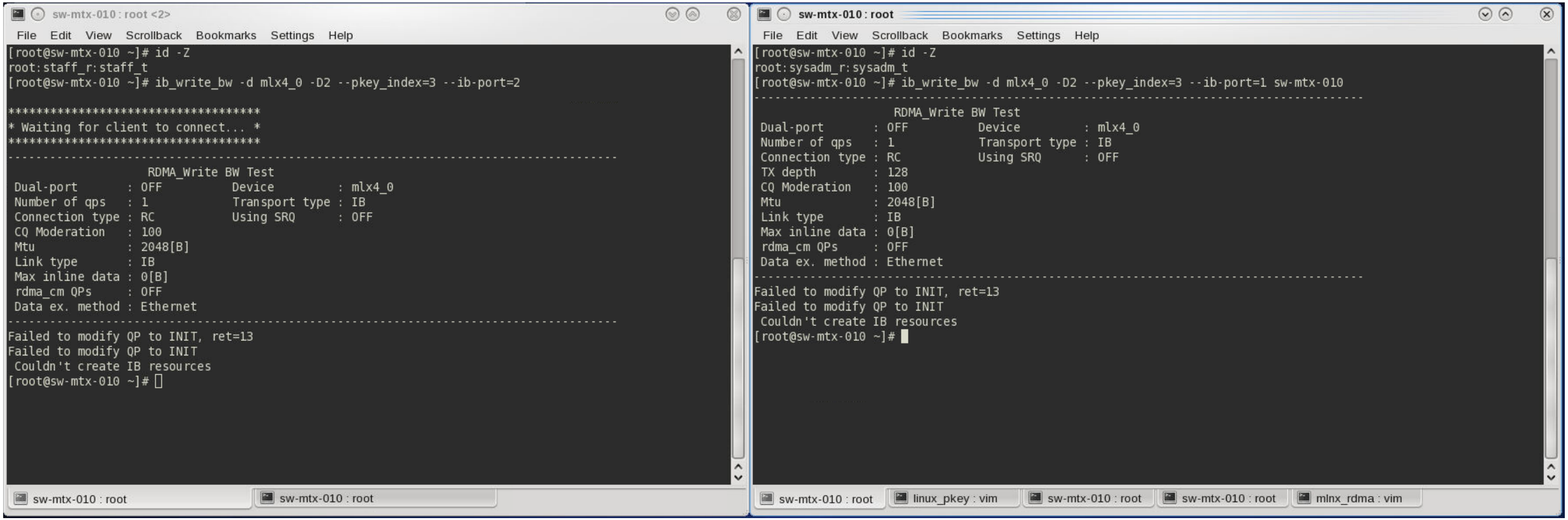
- Both sides of the connection encountered an EACCESS error.

Staff to Admin on Staff Partition

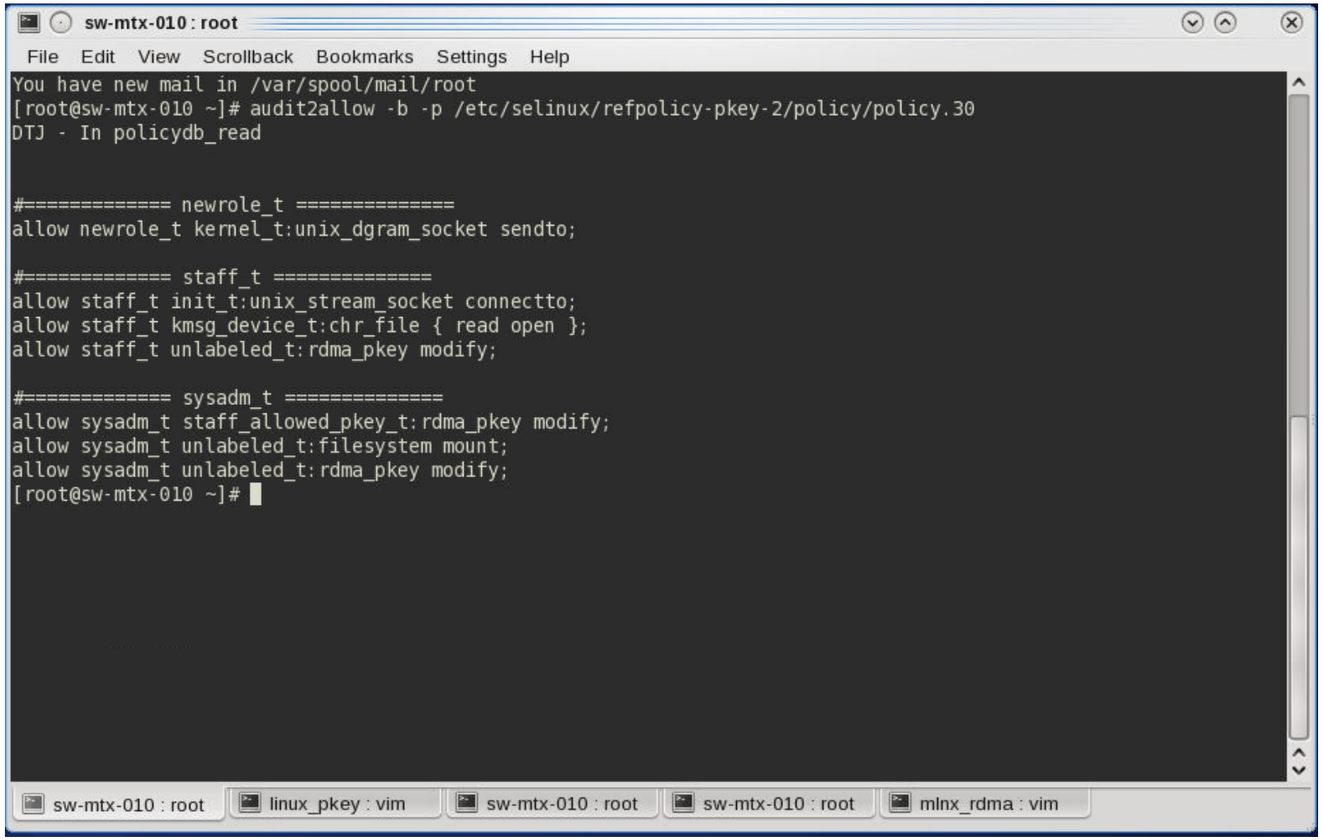


- Note in this case only the Admin side encounters an EACCESS error. The Staff side just has an error connecting.

Admin to Staff on a Neither Allowed Partition



Audit2Allow



- This tool generates policy code to allow violations in the audit log.
- If we added the three allow lines for “rdma_pkey” the access errors in the demo would be allowed.

Discussion Points

- Should we control access to partition values or <partition, port, device> tuples
- Hex format for P_Key values in policy language.
- Subject of partition rules:
  - Pkeys
  - QPs/RDMA IDs/umad FDs
- “Action” word, currently modify.
- Event on QP modify to ERR.
- SMI interface control mechanism.

Thank You

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™