

Mandatory Access Control for Linux Clustered Servers

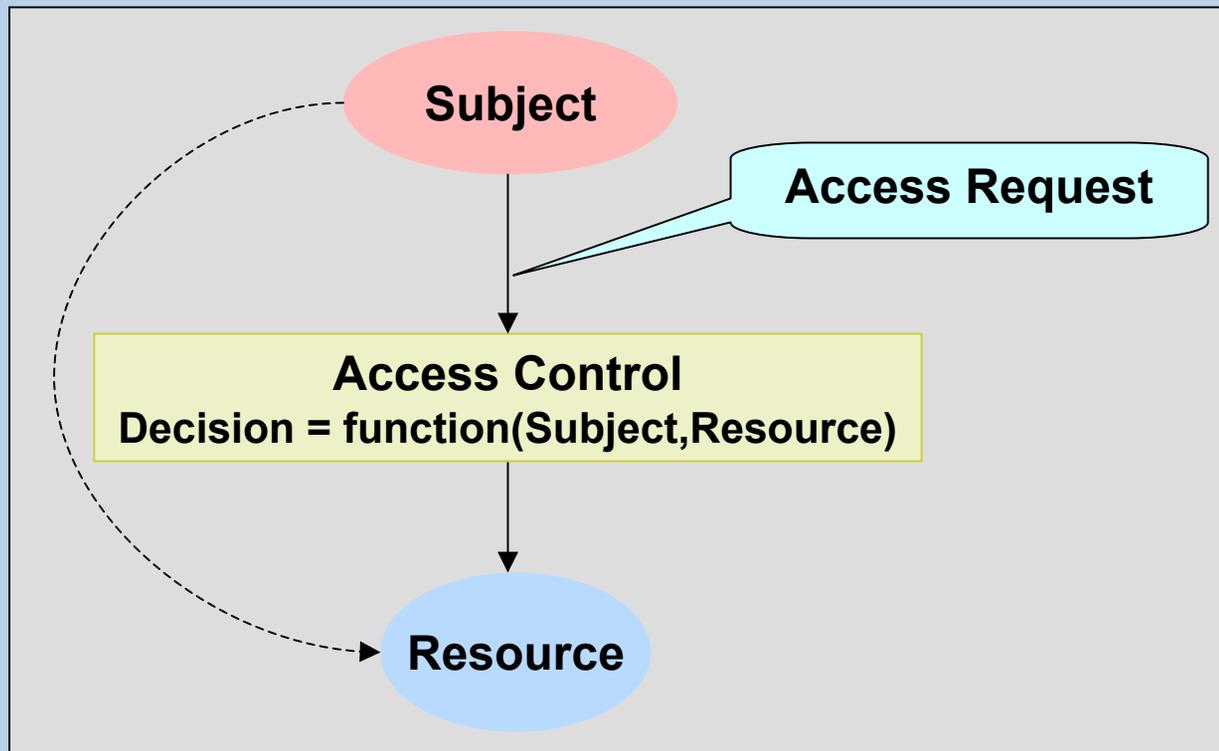
Mirosław.Zakrzewski@Ericsson.ca

ARIES Security Team
Open System Lab
Montréal – Canada

Outline

- Introduction
- DSI Characteristics
- Access Control - General Architecture
- Distributed Security Module
- Security Distribution in DSM
- Demo Architecture (Local and Remote Access)
- Challenges

Introduction (1)



Introduction (2)

- **Discretionary Access Control**
 - Ordinary users involved in the security policy definition
 - Access decisions based on user identity and ownership
 - Two category of users :
 - completely trusted administrators (root)
 - Completely untrusted ordinary user

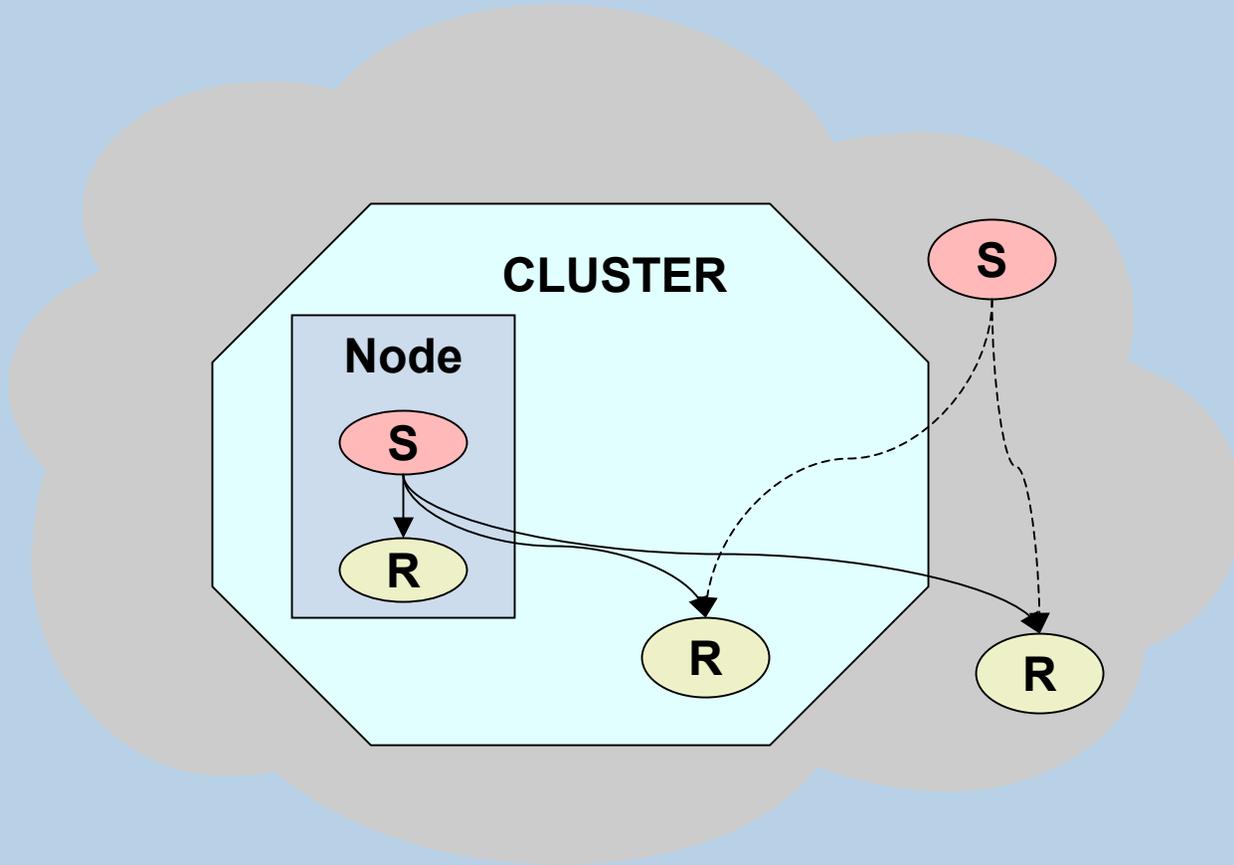
Introduction (3)

- **Mandatory Access Control**
 - policy definition and assignment of security attributes is controlled by a system security administrator
 - access decisions are based on labels that contain a variety of security-relevant information (every subject and object in the system is labelled)

Introduction (4)

- **Cluster**
 - collection of interconnected stand-alone computers working together to solve a problem as a single computing entity
 - cluster can appear as a single system to users and applications
 - from the logical point of view can be seen as a one virtual machine

Introduction (5)



Cluster Access Types

- **Cluster Local Access**

- subject and resource on the same node inside the cluster

- **Cluster Remote Access**

- subject and resource on different nodes inside the same cluster

- **Cluster Outside Access**

- subject inside cluster, resource outside cluster
- subject outside cluster, resource inside cluster

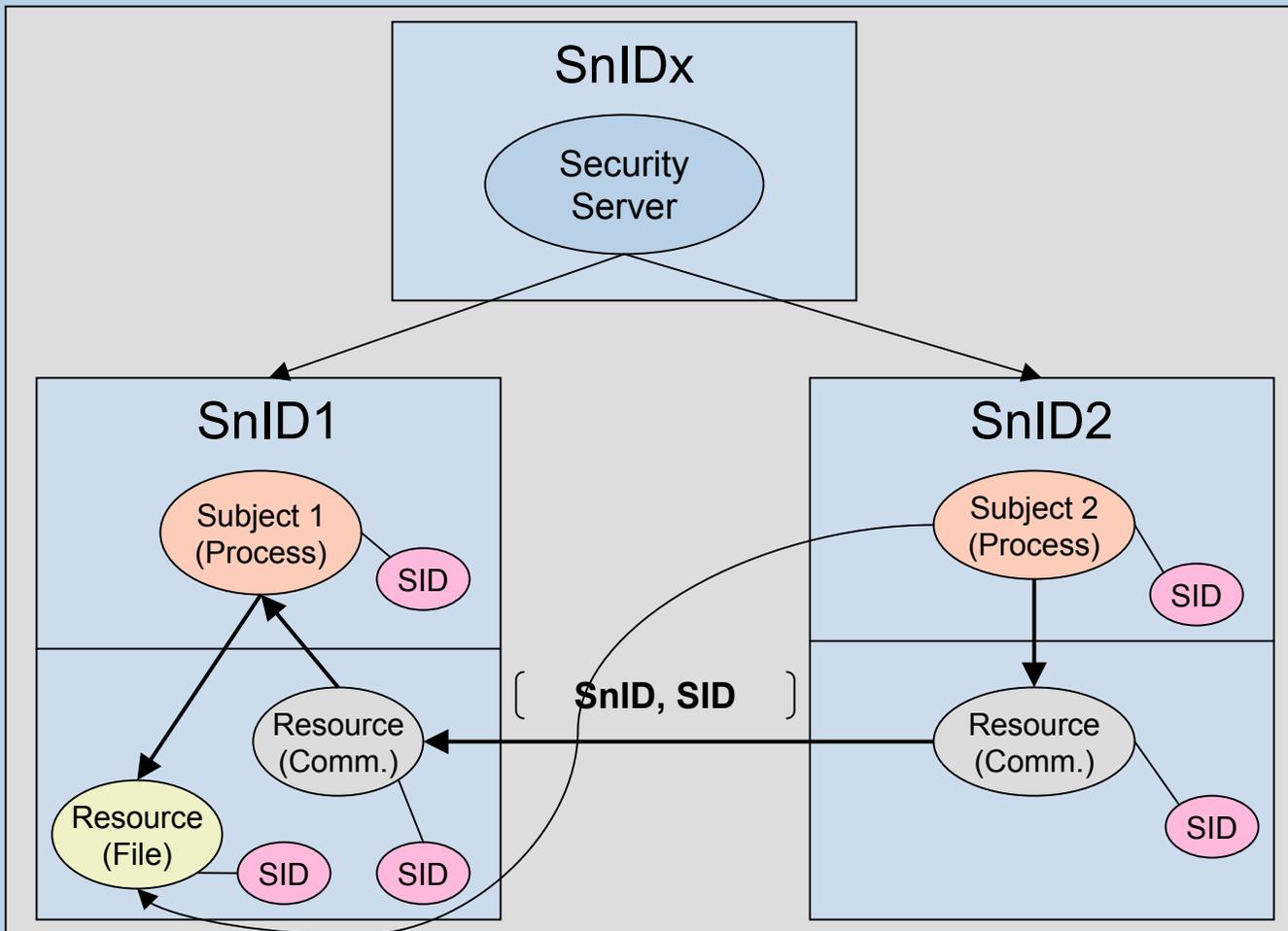
- **No Cluster Access**

- both subject and resource outside cluster

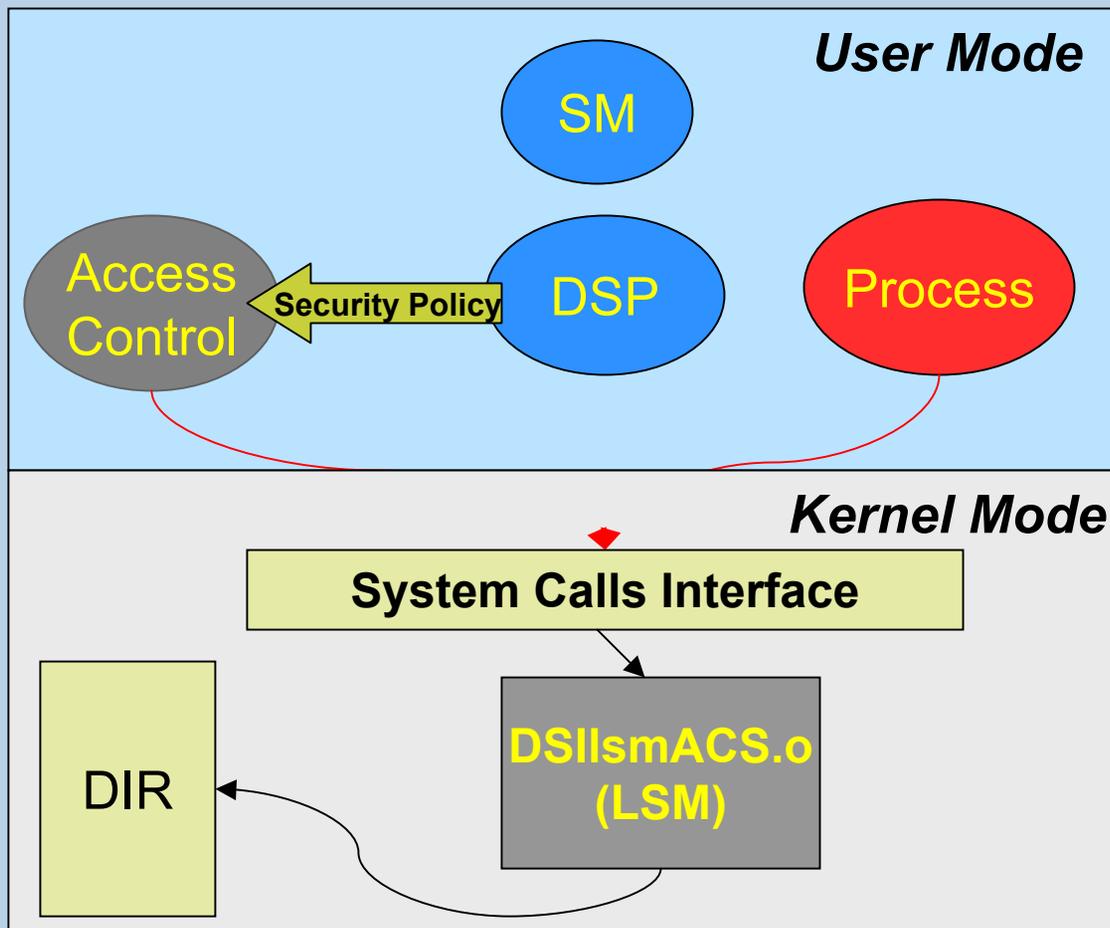
DSI Characteristics

- **Process Level Approach**
 - Controlling Single Process
- **Pre-emptive Security**
 - Run-time changes of security attributes
 - Security can be modified without stopping the system
- **Minimal Impact**
 - Performance
 - Transparency
- **Distributed**
 - Clusters

Access Control – General Architecture



Access Control – General Architecture



Legend:

- SM Security Manager
- DSP Distributed Security Policy
- LSM Linux Security Module
- DIR DSP Internal Representation

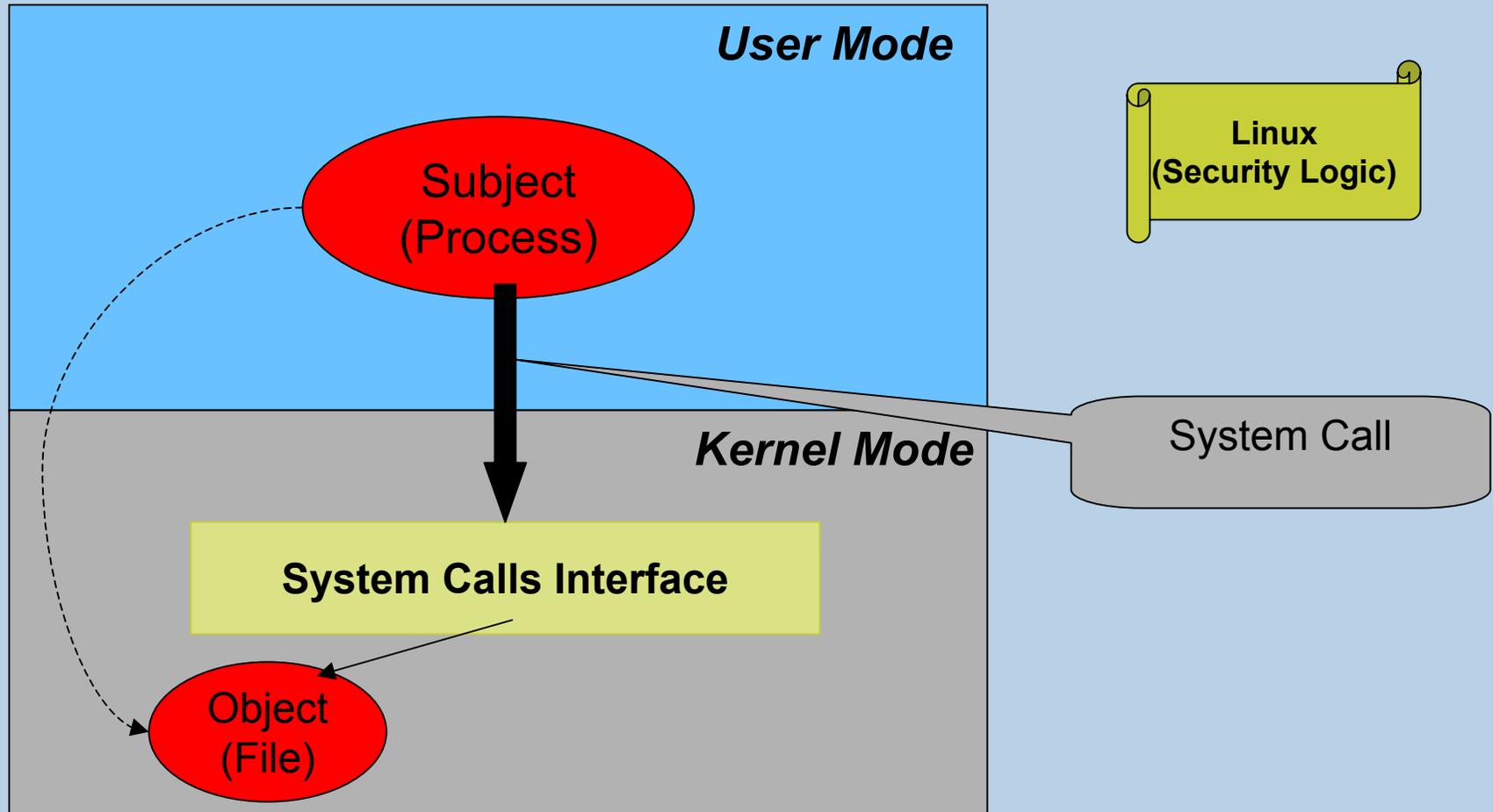
Distributed Security Module

- **Implemented in Kernel Space**
 - Speed (performance)
 - Transparency
- **LSM Framework**
 - Pre-emptive security
 - Process Level Approach
- **IP Options**
 - Distribution

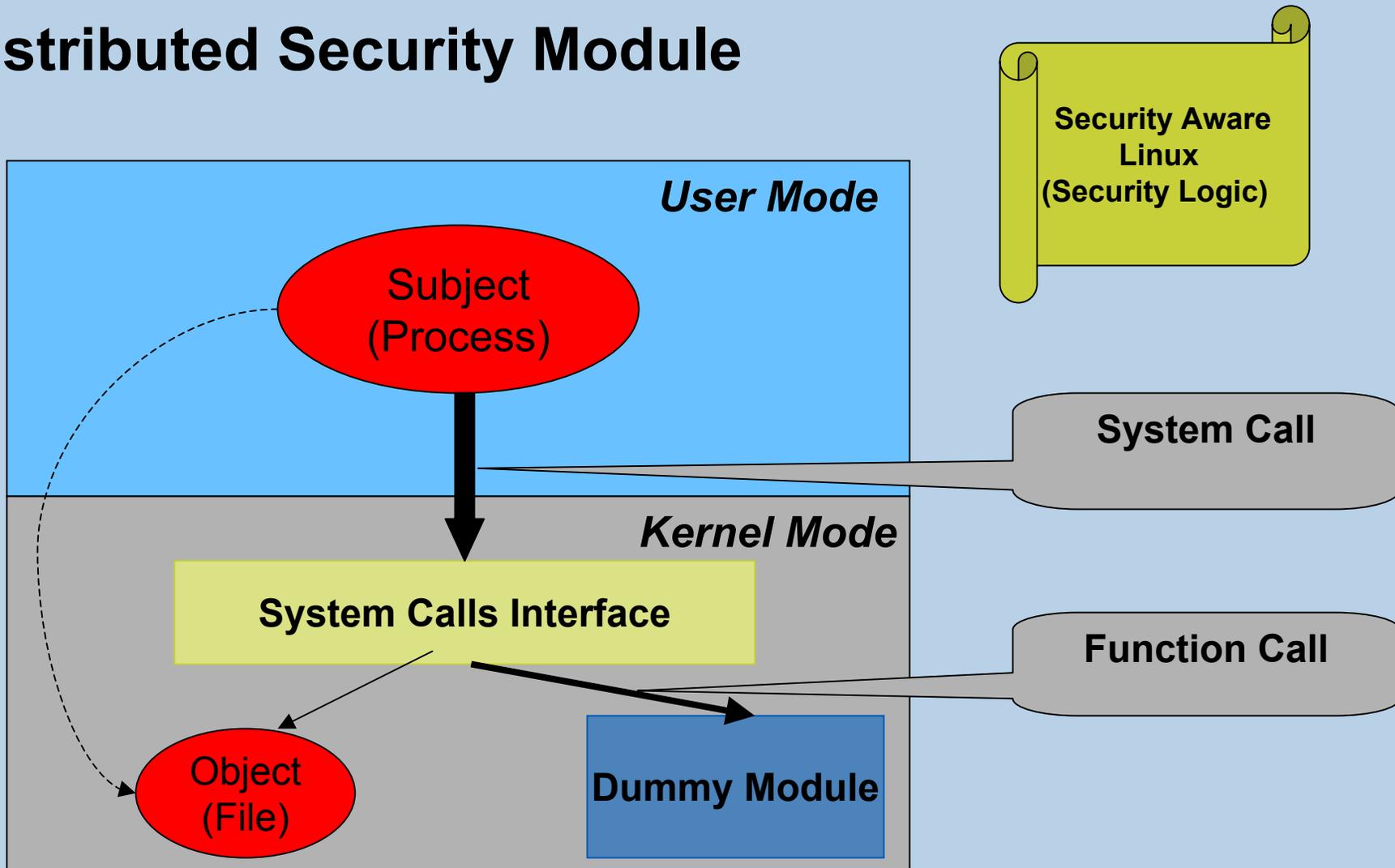
Distributed Security Module

- **Linux Security Module Framework (LSM)**
 - Patch to Linux Kernel by WireX (based on NSA prototype)
 - Security Hooks - points the kernel to allow the control of nearly every system operation
 - 140 fine-grained permission
 - 29 classes
 - Flexible (easy to add user defined security implementations)
 - Function pointers in terms of programming

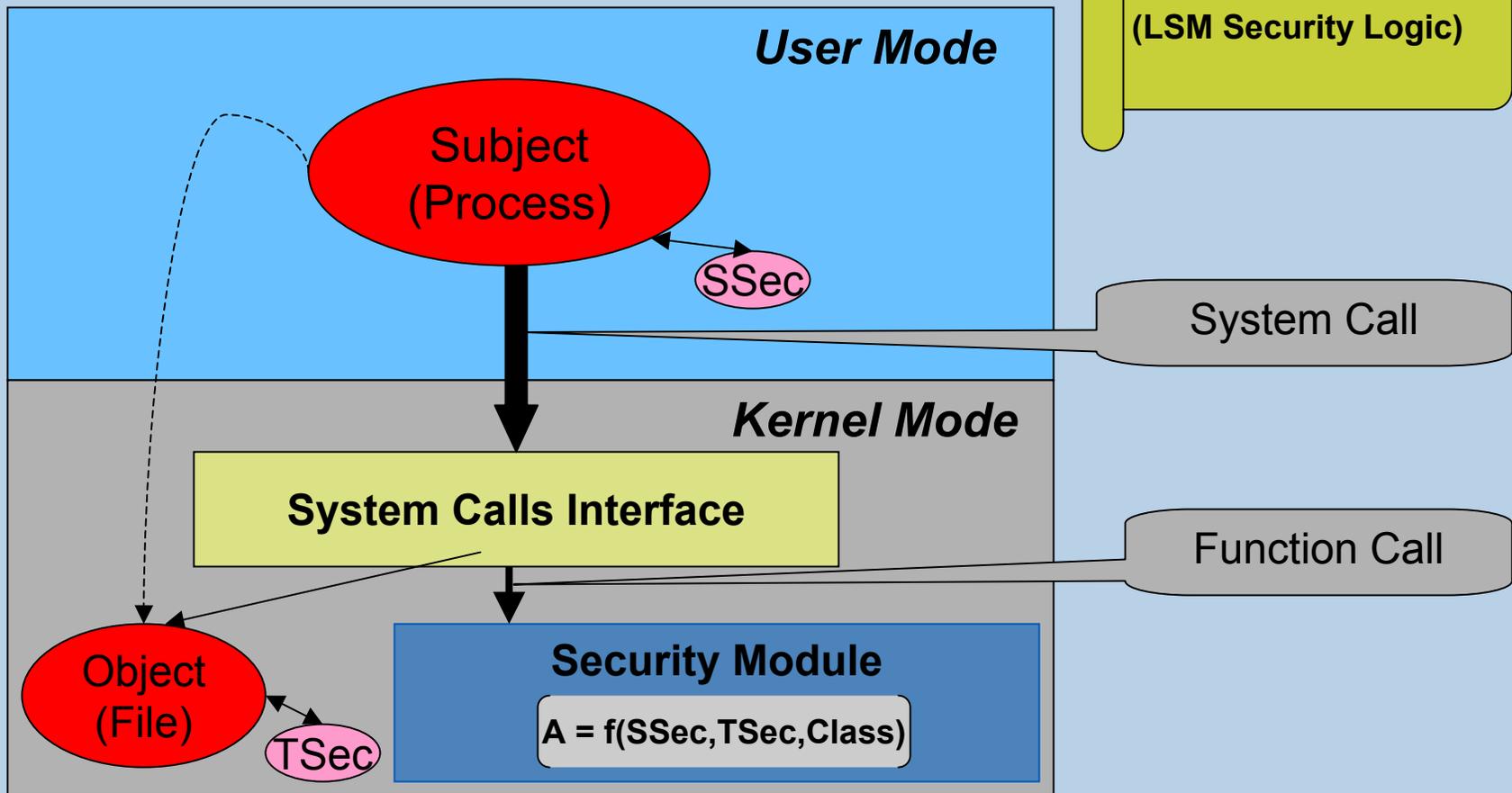
Distributed Security Module



Distributed Security Module



Distributed Security Module



Distributed Security Module

- New Code

```
<linux/security>  
<include/linux/security.h>
```

- New Global

```
struct security_operations *security_ops;  
    /* pointer to all security operation in the kernel */  
struct security_operations dummy_security_ops;  
    /* set of dummy functions */
```

Distributed Security Module

- Structure modification
 - Task Structure (<linux/sched.h>) – process representation in kernel

```
struct task_struct {  
    .  
    void *security;  
    .  
}
```

Distributed Security Module

- Function to Register and UnRegister Security Operation to the Kernel

```
int
register_security (struct security_operations *ops);

int
unregister_security (struct security_operations *ops);
```

Distributed Security Module

- Security Initialization

- Function :

int security_scaffolding_startup(void)

```
int security_scaffolding_startup(void)
{
    security_ops = &dummy_security_ops;
    return( 0 );
}
```

- Call in the startup sequence (linux/init/main.c):
in (void start_kernel(void))

Distributed Security Module

- Labels
 - Objects attached to Linux structures
 - Example : task label (object attached to task structure
struct task_struct <linux/sched.h>)

```
struct task_struct {  
    .  
    void *security;  
    .  
}
```

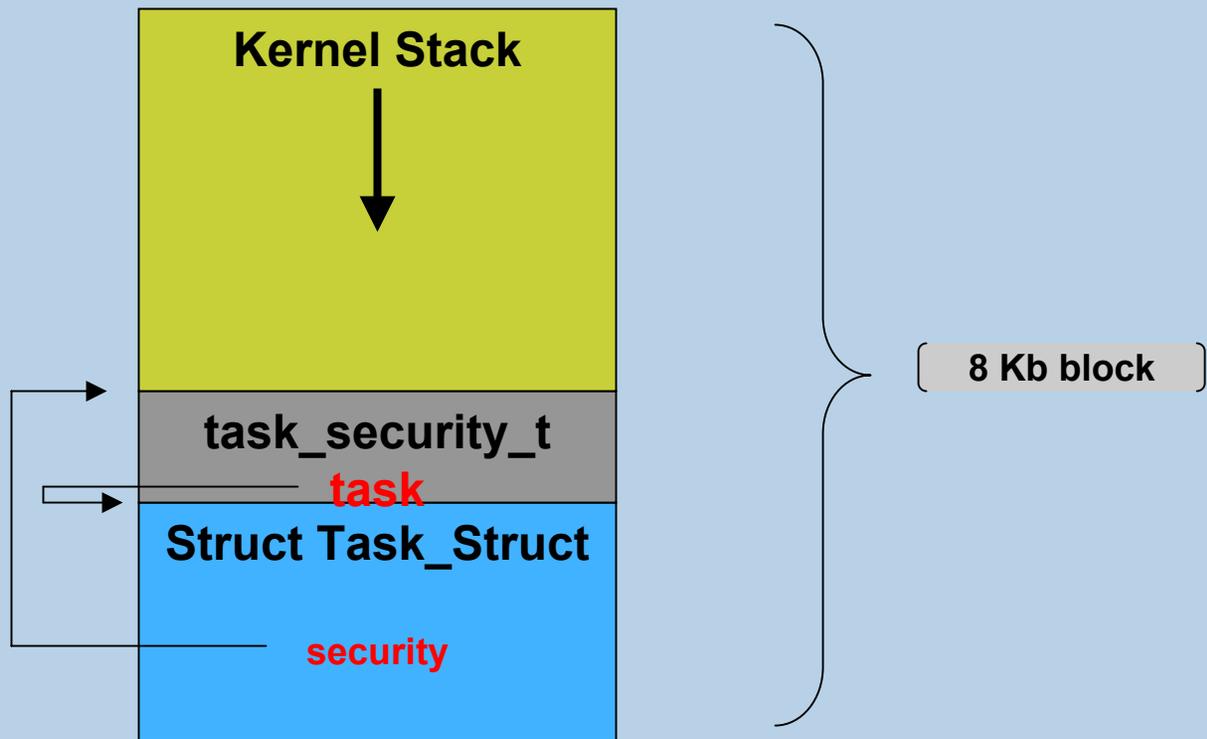
Distributed Security Module

– Task Security Label Format

```
typedef struct {  
    int      sid;  
    int      osid;  
    int      magic;  
    void     *task;  
} task_security_t;
```

Distributed Security Module

- Task Label in relation to task structure



Distributed Security Module

- Task Label Attachment
 - All running tasks are labelled when the security module is loaded (sid is set to default value)
 - After the security module is loaded the tasks are labelled using security hooks (two step process) :
 - **Fork** : sid of parent
 - **Exec** : sid can be modified based on the sid's loaded from the security server (SID is embedded in the ELF format)

Distributed Security Module

- Example (fork system call)

```
int do_fork(unsigned long clone_flags,
            unsigned long stack_start,
            struct pt_reg *regs,
            unsigned long stack_size)
{
    int retval;
    struct task_struct *p;
    retval = -EPERM;
    if(clone_flags & CLONE_PID)
    {
        if(current->pid)
            goto fork_out;
    }
}
```

Distributed Security Module

- Example (fork system call) - continue

```
retval = security_ops->task_ops->create(clone_flags);
if(retval)
    goto fork_out;

retval = -ENOMEM;
. . .
p->security = NULL;
if(security_ops->task_ops->alloc_security(p))
    goto bad_fork_cleanup;
. . .
fork_out:
    return (retval);
```

Distributed Security Module

- LSM patch over Kernel 2.4.17 Installation

<http://lsm.immunix.org>

get the patch

```
lsm-full-2002_01_15_patch_against_kernel_2.4.17
```

```
mv lsm-full-2002_01_15-2.4.17.patch lsm-full-2002_01_15-2.4.17.patch.gz
```

```
gunzip lsm-full-2002_01_15-2.4.17.patch.gz
```

```
cd /usr/src/linux
```

```
patch -p1 < /home/lmcmzak/lsm-full-2002_01_15-2.4.17.patch
```

rebuild the kernel

Distributed Security Module

- **Security System Calls**
 - Set Node ID
 - Change Task SID
 - Set Policy
 - Check Alarms
 - Set Process Image – SID Assignment

Security Distribution

- **Security Information transfer**
 - IP level (first)
 - IP header modification
 - Kernel hooks for IP traffic handling
 - Security information (SID, SnID) transfer as an option in IP header
 - Implementation based on Selopt implementation for SELinux by James Morris
- **IP Options**
 - Commercial Internet Protocol Security Option (CIPSO)
 - Federal Information Processing Standard (FIPS) - 188

Security Distribution

- **Network Labels**

- Labels used when performing remote access (subject and resource on different nodes)
- Security Node ID (SnID) and Security ID (SID) of the subject are added to the IP message
- On the receiving side these two information are extracted and used to build the network security ID (NSID)
$$\text{NSID} = \text{Function} (\text{SnID}, \text{SID})$$
- NSID is used as a local label for access control decisions

Security Distribution

- Network Buffer Label
 - Socket Buffer (<linux/skbuff.h>) – object to contain network packets in kernel

```
struct sk_buff {  
    .  
    void *lsm_security;  
    .  
}
```

Prototype – Network Labels

- sk_buff Security Label Format

```
typedef struct {  
    int          sid;  
    int          magic;  
    struct sk_buff *sk_buff;  
} sk_buff_security_t;
```

Prototype – Network Labels

- sk_buff Security Label Attachment (sending side)
 - Security ID of sk_buff is taken from Security ID of the sending socket
 - Security Node ID is set up by the security server and is global in LSM module

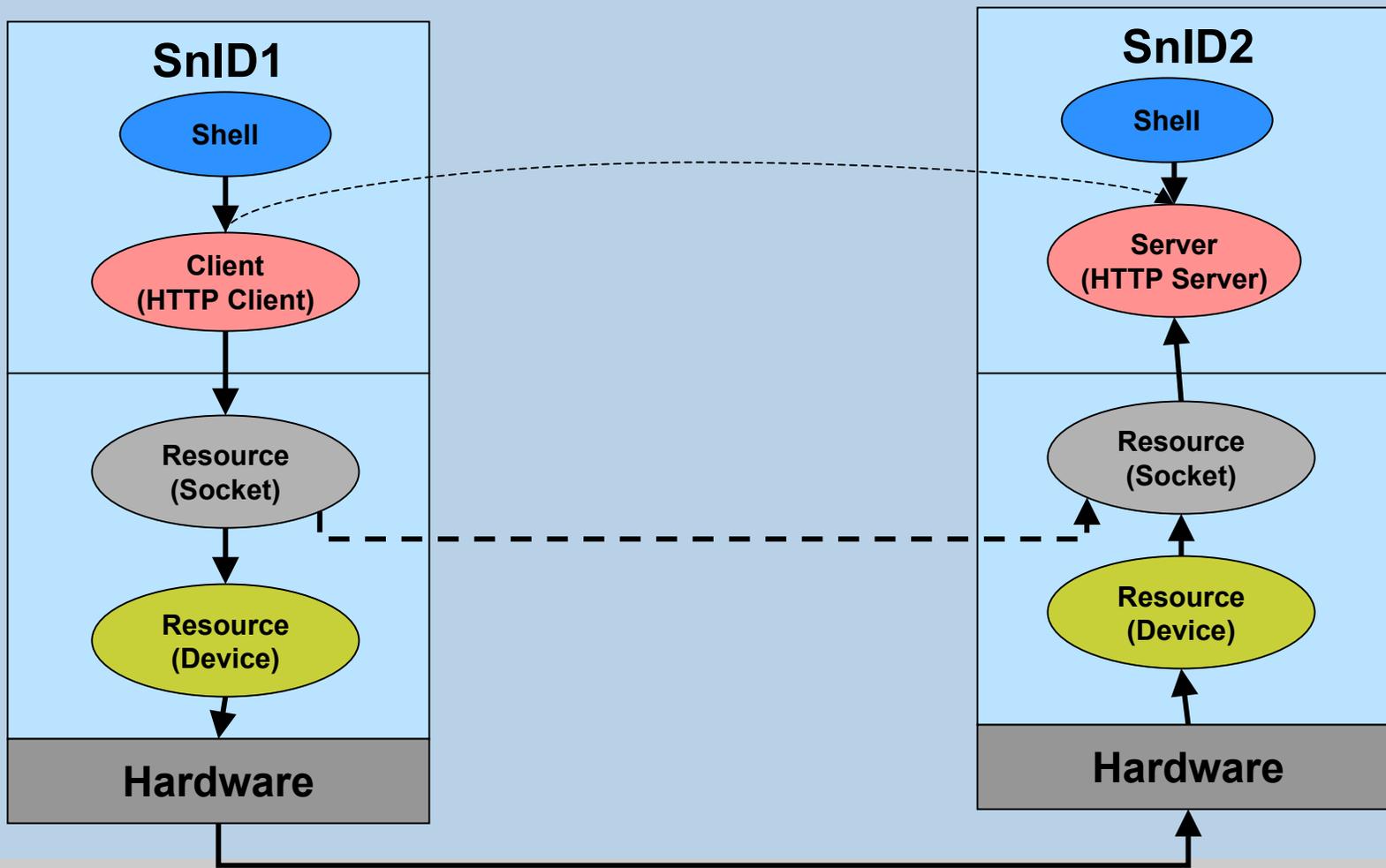
Prototype – Network Labels

- Security Information in Network Message
 - Message is modified on IP layer (adding options)
 - Security Node ID is taken from LSM module and attached to the message
 - Security ID is taken from sk_buff Security Label and attached to the message

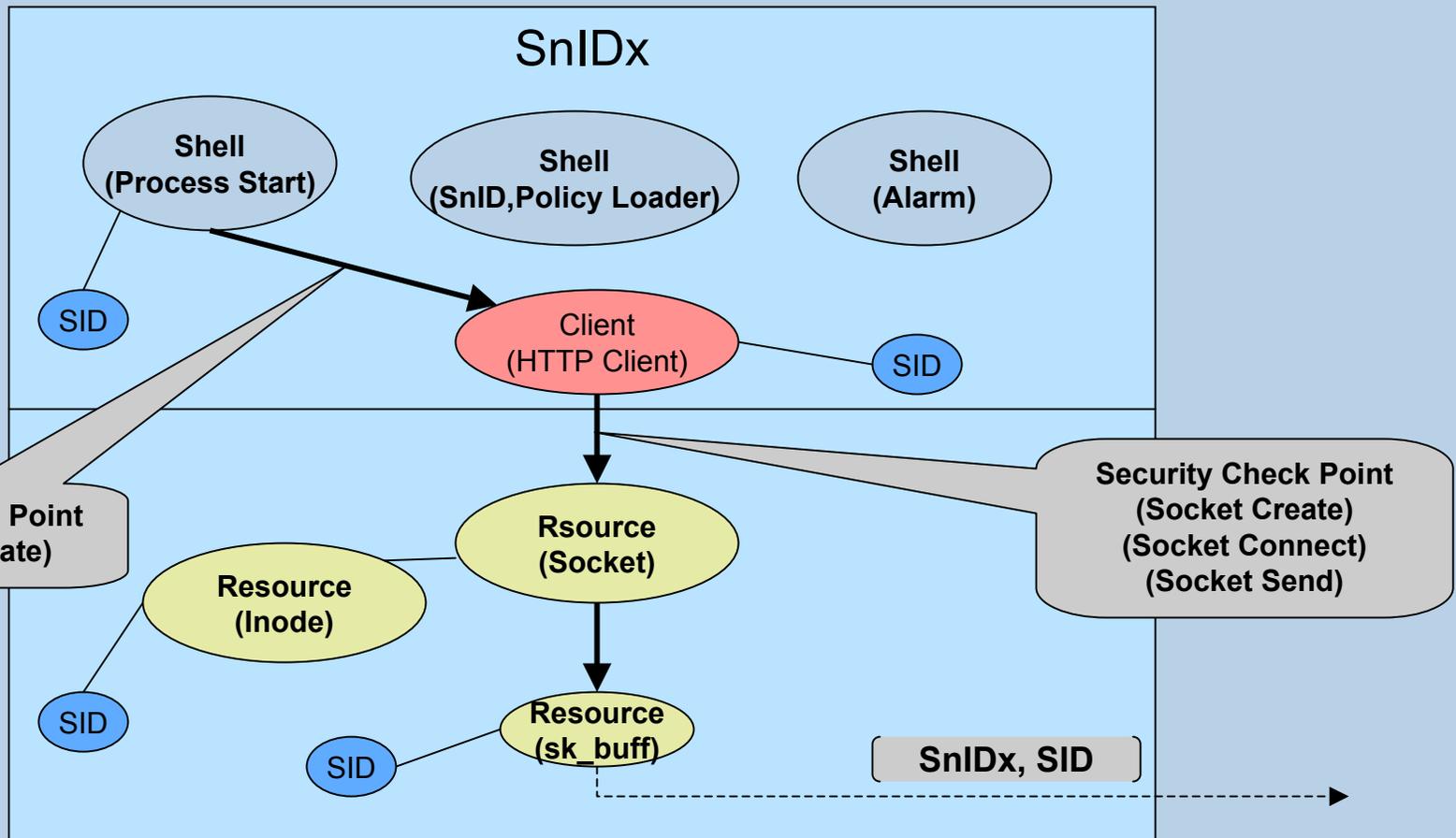
Prototype – Network Labels

- sk_buff Security Label Attachment (receiving side)
 - Extracting Security Node Id (SnID) and Security ID (SID) from the incoming message
 - Converting SnID and SID pair to Network Security ID (NID) based on the conversion table :
$$\text{NID} = \text{Fun}(\text{SnID}, \text{SID})$$
 - NID will be treated as a local label (local access control)

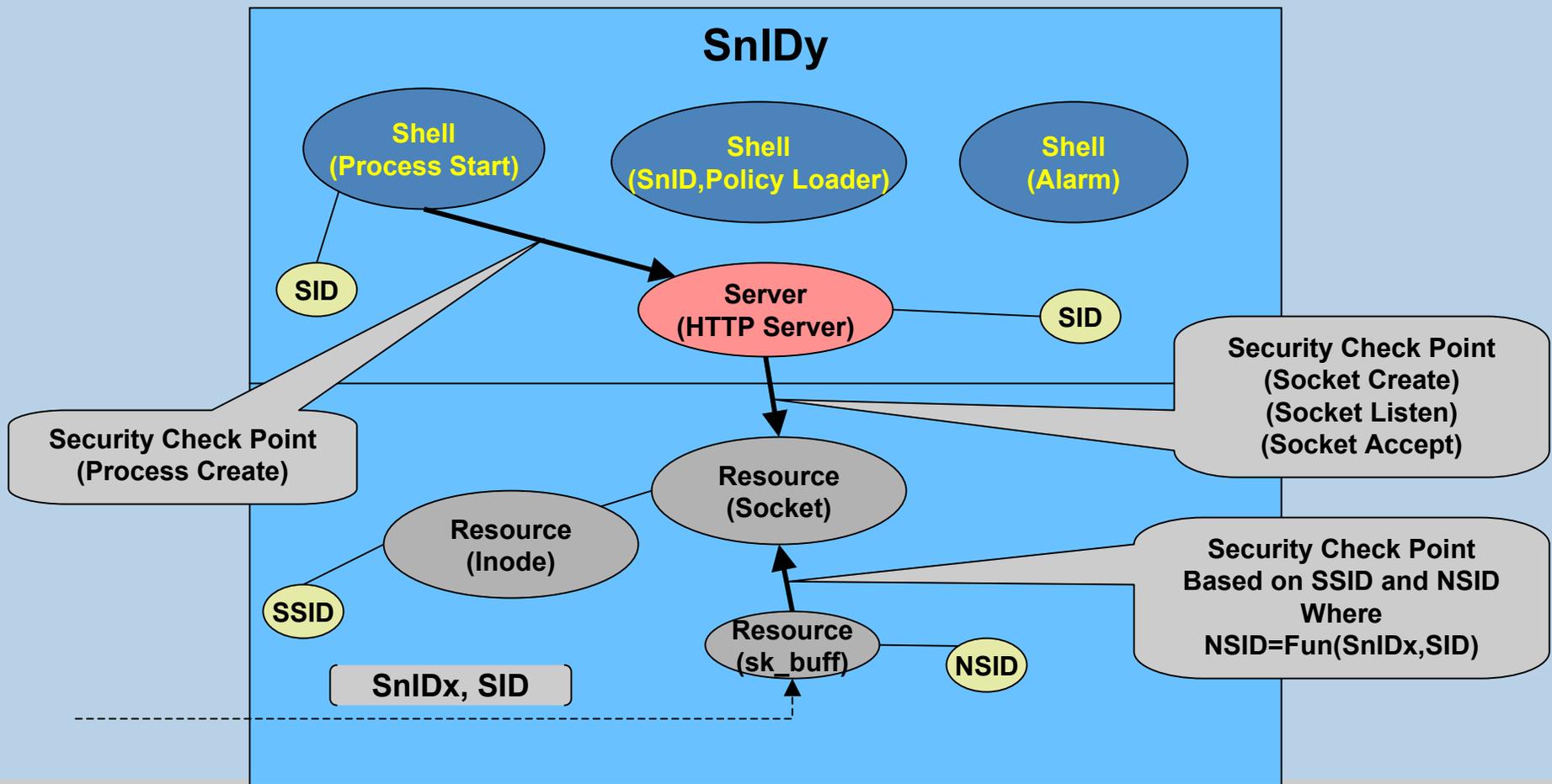
Demo Architecture



Remote Access Control - Demo (sending side)



Remote Access Control - Demo (receiving side)



Challenges: Performance testing

- **Test Types**

- UDP Local Access (Send Message)
- UDP Remote Access (Loopback)

- **Results**

- Performance with IP packet modification
- Performance without IP packet modification
- Buffer overflow

Performance Test Results (1)

- Performance with IP packet modification
(all numbers are in microseconds)

	Linux 2.4.17	Linux 2.4.17 with DSM	% Overhead
UDP Local Access (Send Message)	16.388	19.7	+20%
UDP Remote Access (Loopback)	133.44	173.88	+30%

Performance Test Results (2)

- Performance without IP packet modification
(all numbers are in microseconds)

	Linux 2.4.17	Linux 2.4.17 with DSM	% Overhead
UDP Local Access (Send Message)	16.388	17.084	+4.2%
UDP Remote Access (Loopback)	133.44	140.64	+5.4%

Ongoing work

- Performance optimization
- Server resource access on behalf of a client
- Security information protection
- Security information transfer on lower levels of the protocol stack
- Test the new cluster security against different types of attacks
- Investigate the impact of the security information on the resources outside the cluster

References

All references are available from the paper.

Thank You.



Mirosław Zakrzewski

Ericsson Research – Corporate Unit

Ericsson Canada Inc.

8400 Decarie Blvd

Town of Mount Royal

Quebec H4P 2N2

Phone: 1.514.345.7900 x6458

Fax: 1.514.345.6105

Email: Miroslaw.Zakrzewski@Ericsson.ca