



Using SELinux to secure DDS applications: Configurations and Lessons Learned

Gerardo Pardo-Castellote, Ph.D.

**The Real-Time
Middleware Experts**

Gerardo Pardo-Castellote, Ph.D.
Co-chair OMG DDS SIG
CTO, Real-Time Innovations
gerardo.pardo@rti.com

Outline

- Problem Statement
- Desired DDS Security Model
- Overview of SE Linux
- Using DDS and SE Linux
- Lessons and Future Work

Problem Statement

- DDS is being used to integrate modular systems
- Components need to exchange information via DDS
- But this systems have security concerns:
 - They want to ensure data integrity...
 - and ...access by only authorized principals.

Typical security goals are

- **System Confidentiality**
 - Anti-reverse engineering
 - Protect key-system design info
 - Protect algorithms
 - Existence of operating modes or subsystems
 - Protect control messages
 - Protect discovery information
- System Integrity
- Data Confidentiality
- Data Integrity

Typical security goals are

- System Confidentiality
- **System Integrity**
 - Protection from un-authorized modification (software components, configuration data)
 - Availability (DOS prevention)
- Data Confidentiality
- Data Integrity

Typical security goals are

- System Confidentiality
- System Integrity
- **Data Confidentiality**
 - Enforce Mandatory Access Control on data
 - Enforce need-to-know (Role-Based Access control)
- Data Integrity

Typical security goals are

- System Confidentiality
- System Integrity
- Data Confidentiality
- **Data Integrity**
 - Prevent unauthorized data modification

Outline

- Problem Statement
- **Desired DDS Security Model**
- Overview of SE Linux
- Using DDS and SE Linux
- Lessons and Future Work

- DDS needs a net-centric (pub-sub friendly) security model:
 - Security model defined in terms of DDS concepts (Domains, Topics)
 - Authorization should be separate from apps
 - De-coupled via the Global Data Space
 - Should support one-to-many & multicast
 - Allows persistence and other DDS services

- Leverage two security models:
 - Domain-based MAC (Basic)
 - Topic-based RBAC (Advanced)

Mandatory Access Control (MAC)

- Typical in Military systems
- Each “security object” is assigned a “security level” (Unclassified, Classified, Secret, Top-Secret).
 - Order: Unclassified < Classified < Secret < Top-Secret.
- Each **principal** is also assigned a “clearance level”
- MAC policy made of two rules:
 - (1) A **principal** is only allowed to read information where
security level \leq clearance level
 - (2) A **principal** is only allowed to **write** information where
security level \geq clearance level

Notes:

(1) prevents access to restricted information by users that have not been granted the right level of clearance.

(2) prevents users that have access to more secure information from leaking it to users that should not have access to it.

SELinux (Security-Enhanced Linux) project added a Mandatory Access Control architecture to the Linux kernel.

- Information is only accessible to the owners as well as the *principals* that have the right “role”.
 - E.g. Medical records may only be accessible to the patient itself as well as somebody assigned the role of “medical_staff”.
- E.g. an organization could create roles representing the various job functions.
 - The rights to perform certain access operations could be assigned to specific roles.
 - Each member of the staff is assigned particular roles, and thus become authorized to perform certain access operations.
- Notes:
 - Role-based access control provides the capability of what in A&D is generally called “**need to know**”.
 - Systems including Microsoft Active Directory, SELinux, Solaris, Oracle DBMS, PostgreSQL 8.1, SAP R/3 and many others effectively implement some form of RBAC

Desired Net-Centric Security Solution



- The approach based in two ideas:
 - Secure Domains
 - Confidential Topics
- Secure Domains
 - Limit each Global Data Space (DDS **Domain**) to contain information at a single level of security
 - Only authorized participants are allowed to join a Domain
 - All domain communications are confidential
 - All information is accessible to all members of the domain
- Confidential need-to-know Topics
 - Within a Global Data Space allow participants to read and write **Topics** on a “need to know basis”
 - Separate “right to read” from “right to write”
 - Each Topic is authorized and protected separately

- Each DDS **domain** implements MAC at a single security level.
- Each DDS domain is configured with a CA. The PK of the CA is compiled into each application
- The CA gives certificates to each Participant that is allowed to join the domain
- When participants discover each other they use the pre-configured CA PK to verify their peer's certificate to join the domain
- Result:
 - Limits access to the domain only to the principals (the DDS **Participants**) that have the proper security clearance.
 - Mandatory Access Control (MAC) is therefore applied to each DDS **Domain** separately.

- The RBAC model applied to DDS Topics
- Each **Topic** is assigned a list of “reader roles” and a list of “writer roles”.
 - ‘reader roles’ are the roles of the principals that can read the Topic
 - ‘writer roles’ are the roles of principals that can write the Topic
- Each **Participant** attached to the **Domain** is assigned a set of roles.
 - Write access to the **Topic** given only to **Participants** that have a role that appears in the list of “writer roles” for the **Topic**
 - Read access to the **Topic** given only to **Participants** that have a role that appears in the list of “reader roles” for the **Topic**
- Result:
 - Limits access to the Topic only to the principals (the DDS **Participants**) that have the “need to know” for that Topic
 - A single domain can carry traffic of multiple security levels

Role-based Access Control for Domains



- Access to join domains is mediated by the security plugin
 - Domains can be used to segregate security levels
- Roles can be used to limit:
 - Which users can join a domain
 - Who can create topics in the domain

ROLES:

Commander → engineer, staff ; SECRET

Soldier → admin, staff ; SECRET

Staff → user ; UNCLASSIFIED

Domain ID	Security Level	Join Domain roles	Create Topic roles
3	Unclassified	staff, user, engineer, admin	engineer, admin
4	Secret	staff, admin	admin

Role-based Access Control for Topics



- Access to topics within a domain can also be mediated by the security plugin
- Roles can be used to limit who can write/read individual topics

ROLES:

Commander → engineer, staff ; SECRET

Soldier → admin, staff ; SECRET

Staff → user ; UNCLASSIFIED

Domain ID	Topic	Write roles	Read roles
3	Square	engineer	staff, user
3	Circle	admin	staff, user
3	Triangle	staff	engineer

Role-based Access Control for QoS



- Access to topics can be parameterized further
- Roles can be used to limit allowed levels of service to write/read individual topics

ROLES:

Commander → engineer, staff ; SECRET

Soldier → admin, staff ; SECRET

Staff → user ; UNCLASSIFIED

Domain ID	Topic	Write roles	Read roles
3	Square	engineer	staff [RELIABLE], user [BE only]
3	Circle	admin	staff [KEEP ALL], user[KEEP LAST 1]
3	Triangle	staff	engineer

Outline

- Problem Statement
- Desired DDS Security Model
- **Overview of SE Linux**
- Using DDS and SE Linux
- Lessons and Future Work

Overview of SE Linux

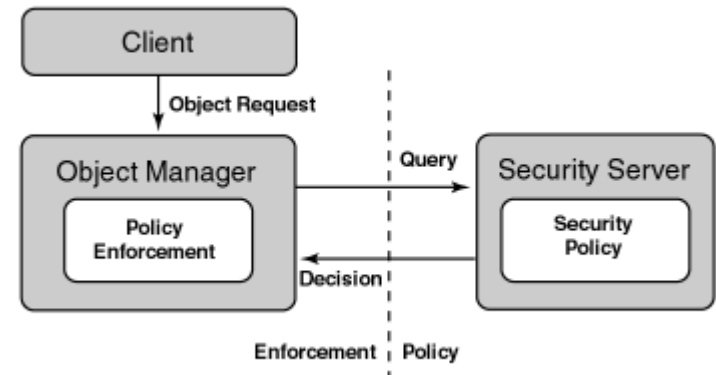
- Linux feature primarily developed by NSA
- Integrated into the kernel version 2.6. Available out-of-the-box in distributions from RedHat, Ubuntu, etc.
- Provides hybrid of concepts and capabilities drawn from:
 - mandatory access controls,
 - mandatory integrity controls,
 - role-based access control (RBAC)
 - type enforcement architecture.
- Enforces mandatory access control policies
 - Confine user programs and system services to the minimum amount of privilege they require to do their jobs.
- Configured by means of policies
 - Reputation of being hard to program... Require expert consulting

SE Linux Security Architecture: Flask

- The overall security architecture is called Flask
 - Designed by the NSA
 - assistance from the University of Utah and the Secure Computing Corp.
- Flask
 - Security policy's logic is encapsulated within a separate component of the operating system (security server) along with general interface for obtaining security policy decisions
- The SE Linux implementation of Flask security server defines a hybrid security policy composed of:
 - Type Enforcement (TE),
 - Role-based access control (RBAC),
 - Optional multilevel security (MLS), widely used in military security.
- Security policy is compiled by a separate program called *checkpolicy*,
 - Policy read by the security server at boot time (file `/ss_policy`)
 - Policy file variable for each time the system boots.
 - Policies can also change during system operation (as long as the policy is configured to allow such changes) by using the `security_load_policy` interface

Flask

- Flask has two policy-independent data types for security labels: *security context* and the *security identifier*.
 - The security context is a variable-length string that represents the security label.
 - The security identifier (SID) is an integer that serves as a simple handle to the security context
 - SID can only be interpreted by the security server
 - SID is mapped by security server to the security context
- Flask does the actual system binding through constructs called object managers.
 - Object Managers handle SIDs and security contexts opaquely

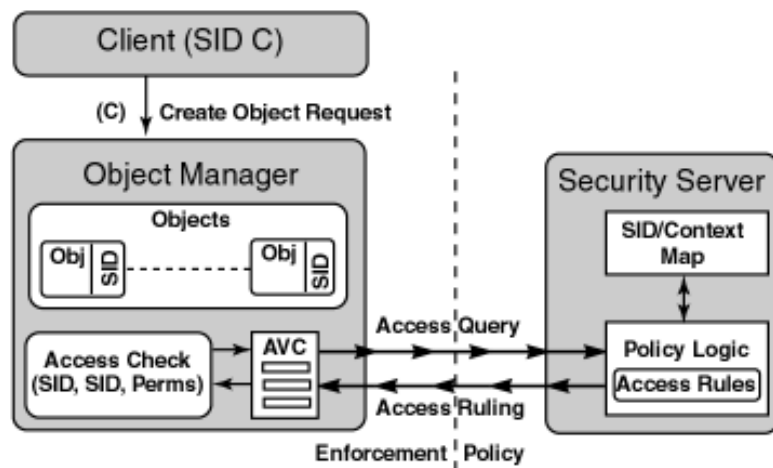


SE Linux Security Context

- SE Linux makes access control decisions based on a the “Security Context” of a principal and the target object
- A Security Context consists of 4 fields separated by colons:
 - user:role:type:mls
 - As in: joe:sysadm_r:mozilla_t:SystemHigh
- Can also be represented by a 32bit integer (SID)

Access Decisions

- Object managers consult the security server to get an access decision based on:
 - Subject Label
 - Object Label
 - Object Class.
- Object Class is an integer that identifies the kind of object it is
 - E.g. Regular file, a directory, a process, TCP socket, ...
- For each Object class there is an access vector defining the permissions
 - Permissions are usually defined by the services the object can support and the security policy that is being enforced.
 - Permissions are interpreted based on the class, because different services (operations) exist for different kinds of objects.
 - Vectors can be cached in the Access Vector Cache (AVC) or stored with the object
 - Caching prevent object managers from flooding the security server with requests for decisions that have already been performed.



Notes on SE Linux Model

- Subjects of access control decisions enforced on Processes
 - So applications running on a single process cannot be differentially secured by the kernel
 - E.g. All components running in an app server are indistinguishable to the Linux kernel
- Objects protected by SE Linux are:
 - files, directories, inter-process communication mechanisms, ports, devices, etc.
- Network support
 - IPSec labeling. Transmits security context to peer process
 - Secmark. Leverages IPtables to Label packets based on the network properties of each packet. But relies on having a trusted network

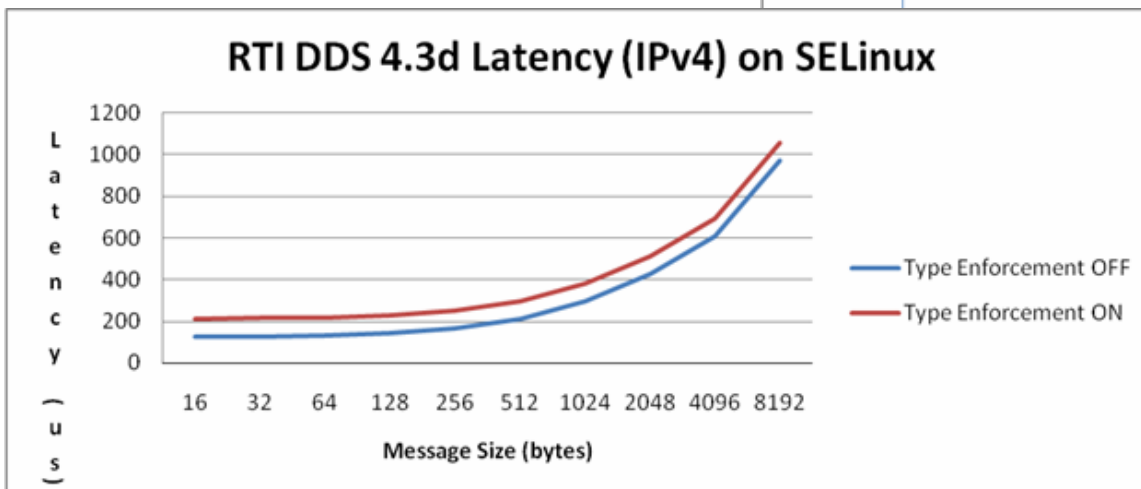
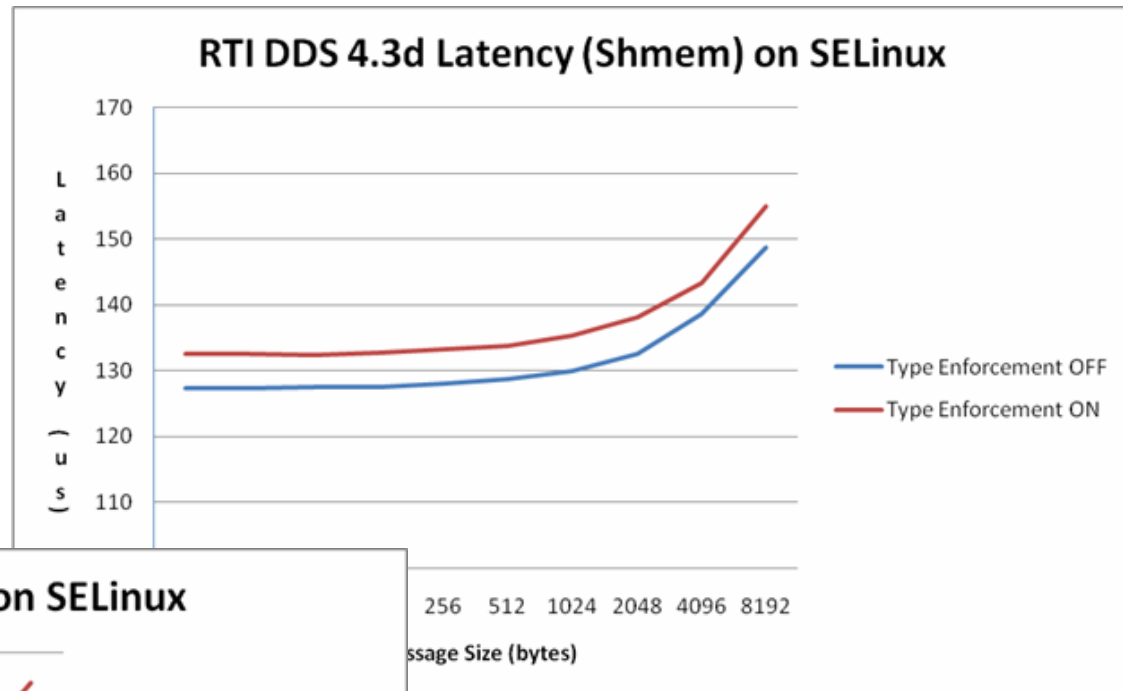
Outline

- Problem Statement
- Desired DDS Security Model
- Overview of SE Linux
- **Using DDS and SE Linux**
- Lessons and Future Work

DDS SELinux Overhead : Latency



- Policy enforcement on DDS's access of kernel resources is fixed overhead



Ethernet Transport Security



- Three suggested general options
 - IPSec
 - TLS / DTLS
 - Custom, message-level protocol
- Some options have multiple implementations
 - TLS / DTLS implemented in OpenSSL and NSS
 - Tradeoffs discussed later
- Kerberos not addressed, but potentially viable
 - Provides authentication and traffic encryption
 - Requires central server – undesirable here

- Positive
 - Supports DDS and CORBA
 - OS managed – completely transparent
 - Supports passing SELinux security contexts
 - Strong security – confidentiality and integrity

- Negative
 - OS managed – not application controllable
 - Atypical use of IPSec – common use is VPN
 - Difficult to provide secure and open channels
 - All traffic between two Nodes / Components identical
 - Will use single IPSec configuration
 - Multicast not currently supported

- Positive
 - Supported by CORBA and DDS
 - Well tested, standard mechanism for both
 - Strong security – confidentiality and confidentiality

- Negative
 - Selecting secure / open channel complex
 - Requires associating session with Writer not Participant
 - May be possible to work around (“Secure Domains”)
 - Use for authentication may require development
 - CORBA appears to provide basic APIs
 - DDS has infrastructure that can be leveraged but work needed
 - Multicast not currently supported

- Encrypt payloads not protocol
- Positive
 - Control over secure vs. open at message level
 - Could work with multicast
- Negative
 - Custom development – time / effort / risk
 - Requires re-inventing security (e.g., replay attacks)
 - Will require session negotiation for symmetric keys
 - I.e., custom protocol in addition to other infrastructure
 - Examples exists of course, but still difficult

Multicast and Transport Security



- Multicast needed for DDS
 - Key for high-throughput and low-latency
 - Increases in importance with more subscribers
- IPSec and DTLS typically disable multicast
 - DDS transparently downgrades with DTLS
 - Results in many point-to-point connections
- Root problem – requires shared session key
 - DTLS and IPSec do not today support shared keys
 - Some experimental proposed standard mechanisms
 - <http://www.ietf.org/html.charters/msec-charter.html>
 - Proposal currently dead, but contains valuable ideas

Multicast and Separation



- IPSEC and (D)TLS provides strong separation
 - Point-to-point encryption provides separation
 - Separate session key per-connection
 - No real need for keys per-domain / topic / etc.
- Multi-cast introduces additional complications
 - Shared key could be at Domain or Topic level
 - Choice depends on granularity of separation needed
- Strong message authentication is possible
 - Sender signs header / encrypts with shared key
 - Unique identity provided with multicast

- NSS
 - FIPS certified
 - Broader support for hardware tokens / acceleration
 - Strongly supported by Red Hat
 - Does not currently support DTLS
 - Not currently support by CORBA or DDS

- OpenSSL
 - Currently supported by CORBA and DDS
 - Supports DTLS
 - Broad vendor and community support
 - FIPS certified with some caveats

Outline

- Problem Statement
- Desired DDS Security Model
- Overview of SE Linux
- Using DDS and SE Linux
- **Lessons and Future Work**

How could SE Linux be used to Secure DDS?

- SE Linux Type Enforcements can be used to restrict access to domains
 - DDS maps domains to ports and SE Linux can restrict access which processes can open a given port number
- SE Linux could also be used to restrict reception on certain Topics
 - DDS Interoperability allows tying Topics to separate ports
- SE Linux does not have enough granularity to restrict writing to a Topic

Gaps / Issues

- Restricting who can ‘send to a Topic’
- Restricting who can access specific content for a given Topic
 - E.g. not all “Tracks”, only tracks with a certain classification
- Protecting the data while on the network
 - Requires Encryption
 - Can use PKI and TLS, but not for multicast
- Providing no repudiation
 - Requires digital signatures
 - Can use PKI and TLS, but not for multicast
- Complexity of SE Linux policy programming. Out-of-the-box policies over 50,000 LOC

Conclusions

- SE Linux could be used to enhance DDS security
- Can provide
 - Coarse-granularity control:
 - Mandatory Access Control to Domains
 - Medium-granularity:
 - Read Access to Topics by Role
- Securing data on the network requires the use of a secure transport: e.g. IPSEC, TLS
- SE Linux by itself not sufficient to enforce the desired DDS role-based access control model
 - Requires some of the policy enforcement to occur in the DDS infrastructure. E.g. restrict write access to Topics.
 - Support of multicast will require either
 - A special “secure multicast” transport
 - Or rely data encryption that used different Keys for different Topics
- Not a solved problem. Needs additional research