Security Annex Update

June 25, 2019

Dave Gluch

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-0657

Overview

Security Annex Standard

- provides guidance and support for using AADL for specifying, modeling, and analyzing secure system architectures
- presents the OSATE/ALISA environment as an exemplar for comprehensive policy/requirements documentation analysis and verification.
- includes security properties for classification and enforcement
  - property sets are user modifiable
  - example security components (e.g. key classifier)
- includes exemplar analysis methods and claims
  - Resolute and JAVA

Encompasses

- security domains
- trusted components

Some Questions

Are the property sets for classification and enforcement sufficient to capture the major security aspects of an architecture?

Are they sufficient to support the major security analyses of an architecture?

Is there a security consideration that is not included but should be?

- detailed modeling of encryption algorithms
- threat and vulnerability modeling

Should more be included on key/certificate management?

Example Models and Analyses

Models

- Commercial Transport Aircraft
- Mission-Specific Aircraft (reconnaissance)
- Cross Domain System

Security examples are available at

https://github.com/osate/examples.git

Supporting verification library files including property sets are available at

https://github.com/reteprelief/isse

Security Policies and Requirements

A security policy and requirements documentation and analysis approach that includes tool support

Architecture-Led Incremental System Assurance (ALISA) workbench is an exemplar of an approach

- Systematic documentation
- Assurance Cases
- Resolute and JAVA verification methods

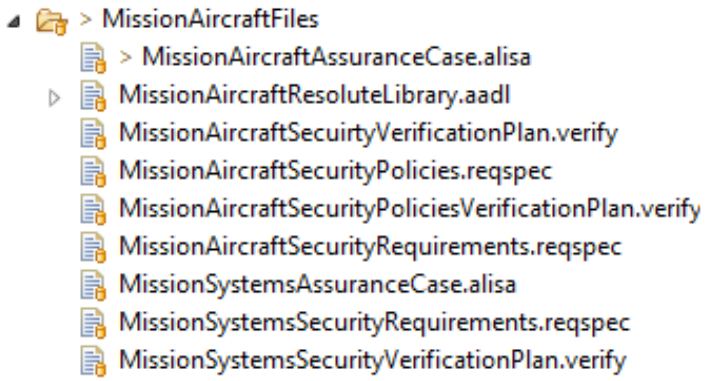
Security Policies and Requirements Documentation and Analysis

Documentation of Security Policies and Requirements

- ReqSpec of ALISA
- Naming convention to distinguish between policies and requirements

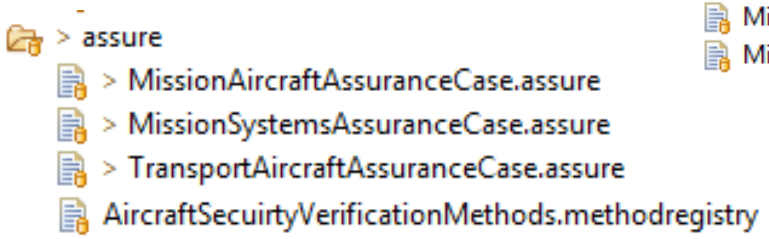
Policies and Requirements Verification – ALISA

- Verification plans (.verify)
- Assurance cases (.alisa)
- Results (.assure)

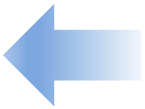


Methods

- Resolute
- Java



Security Annex Properties



Property sets that can be edited by a user

- Security Classification Property Set
  - Security Clearances (subjects)
  - Information Security Levels (objects)
  - Security Levels (subjects and objects)
  - Trusted Classification
  - Security Domains
- Security Enforcement Property Set
  - Encryption properties
  - Data Authentication
  - Subject Authentication

Security Clearances

Principal security clearance and a supplemental statement

```
Security_Clearance: inherit enumeration (TopSecret, Secret, Confidential,
No_Clearance)
applies to (system, device, processor, virtual processor, thread, thread group,
subprogram, subprogram group, process, abstract);

Security_Clearance_Supplement: inherit aadlstring
applies to (system, device, processor, virtual processor, thread, thread group,
subprogram, subprogram group, process, abstract);
```

Secondary security clearance and a supplemental statement

```
Secondary_Security_Clearance: inherit  enumeration (TopSecret, Secret, Confidential,
No_Clearance)
applies to (system, device, processor, virtual processor, thread, thread group,
subprogram, subprogram group, process, abstract);

Secondary_Security_Clearance_Supplement: inherit aadlstring
applies to (system, device, processor, virtual processor, thread,  thread group,
subprogram, subprogram group, process, abstract);
```

No assumption about the relationship of the *Security_Clearance* property and the *Secondary_Security_Clearance* property.

Information Security Levels

```
Information_Security_Level: inherit enumeration (TopSecret, Secret,
Confidential, Unclassified)
 applies to (data, port, system, process, device, abstract);

Information_Security_Caveats: inherit aadlstring
applies to (data, port, system, process, device, abstract);
```

data classifiers and instances, including ports and associated data classifiers

Additional categories enable conceptual modeling

Generalized Security Levels

When no differentiation between subject and object is needed.

```
Security_Level: inherit enumeration (TopSecret, Secret, Confidential,
Unclassified) applies to ( system, processor, virtual processor, thread,
thread group, subprogram, subprogram group,  data, port, process, device,
abstract);

Security_Level_Caveats: inherit aadlstring applies to (system, processor,
virtual processor, thread, thread group, subprogram, subprogram group,
data, port, process, device, abstract);
```

Readily modified, for example:

```
Security_Level: inherit enumeration (High, Medium, Low)  applies to
(system, processor, virtual processor, thread, thread group, subprogram,
subprogram group, data, port, process, device, abstract);
```

Domains and Trusted Components

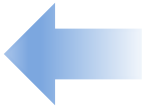
```
Security_Domain: record (
        domain_name: aadlstring;
        description: aadlstring;
        )applies to (abstract, system);

Need_To_Know_Domains: list of aadlstring applies to (all);
```

```
Trusted : aadlboolean applies to (system, process, thread, thread group,
subprogram, subprogram group, processor, virtual processor, bus, virtual
bus, abstract);
```

Security Annex Properties

Property sets that can be edited by a user

- Security Classification Property Set
  - Security Clearances (subjects)
  - Information Security Levels (objects)
  - Security Levels (subjects and objects)
  - Trusted Classification
  - Security Domains
- Security Enforcement Property Set
  - Encryption properties
  - Data Authentication
  - Subject Authentication



Encryption Property

```
Encryption: inherit record (
description: aadlstring;
encryption_form: enumeration (symmetric, asymmetric, hybrid,
authenticated_encryption, authentication_only, no_encryption, to_be_specified);
-- if the encryption form is hybrid both symmetric and asymmetric are used.
encryption_mode: list of enumeration (ECB, CBC, CFB, CTR, GCM, CBC_MAC, other,
to_be_specified);
-- list is needed for hybrid encryption
encryption_algorithm: list of aadlstring;  -- e.g. AES,  RSA, OTP
-- a list is needed for hybrid encryption
-- encryption algorithm list values must correspond to the key length list values
padding: enumeration (OAEP, other, no_padding, to_be_specified);
authenticated_encryption_type: enumeration (Encrypt_then_MAC, MAC_then_Encrypt,
Encrypt_and_MAC, AEAD, signcryption, no_authenticated_encryption);
key_length  : list of Size; -- caution confirm that the value of key_length property
for the associated key classifier or instance is consistent
-- a list is needed for hybrid encryption
-- key length list values must correspond to the encryption algorithm list values
key_type: SecurityEnforcementProperties::key_classifier; -- references a classifier
private_key: SecurityEnforcementProperties::key_instance;  -- references an instance
public_key : SecurityEnforcementProperties::key_instance; -- references an instance
single_key : SecurityEnforcementProperties::key_instance;  -- references an instance
) applies to (data, port, abstract, system, bus, memory, device, processor,
virtual processor, virtual bus, connection, process, thread);
```

Data Authentication

Used with Authenticated Encryption

```
Data_Authentication: inherit record
        (
        description: aadlstring;
        authentication_form : enumeration (MAC, MIC, signature,
                        signcryption, no_autentication, to_be_specified);
        authentication_algorithm: aadlstring;
        key_length: Size; -- caution confirm that the value of key length
for the associated key classifier or instance is consistent
        hash_length: Size; -- optional, if the message is hashed before
authentication. Does not apply to authenticated encryption.
        hash_algorithm: aadlstring;
        authentication_key: SecurityEnforcementProperties::key_Instance;
        )
        applies to (data, port, abstract, system, bus, memory, device,
processor, virtual processor, virtual bus, connection);
```

Encryption Key Properties

```
abstract key
end key;
-- extend abstract key to data classifiers
data symmetricKey extends key
properties
SecurityEnforcementProperties::keyLength => 256 bits;
end symmetricKey;

data publicKey extends key
end publicKey;

data privateKey extends key
end privateKey;
```

the key_length value for a key instance must be consistent with the value of the key_length field declared in the key's associated encryption or data authentication property.

Key Related Properties

```
Key_length: Size applies to (abstract, data);
crypto_period: Time applies to (abstract, data);
Key_classifier: type classifier (abstract, data);
Key_instance: type reference (data);
text_type: enumeration (plainText, cipherText) applies to (abstract, data);
Key_distribition_method: enumeration (encrypted_channel, QKD,
direct_physical_exchange, courier, public_channel) applies to (all);
```

Key Management

Security Key Component

- Extensible to specific key components
- Key properties (e.g. length, cryptoPeriod)

Key/Certificate

- Generation
- Storage
- Distribution control
- Destruction
- Replacement

Should we include a certificate component?
For example:

```
abstract CertificateAbs
end CertificateAbs;

data Certificate extends CertificateAbs
end Certificate;

data implementation Certificate.SSL_TLS
subcomponents
Subject: data subject.certificate;
Issuer: data issuer.certificate;
PeriodOfValidity: data periodOfValidity.certificate;
AdminInformation: data adminInformation.certificate;
ExtendedInformatio: data extendedInformation.certificate;
end Certificate.SSL_TLS;
```

Subject Authentication Property

Declares that a subject (component instance) can participate or participates in authentication as specified, including authentication negotiations employing the specified authentication protocol, or that the component (e.g. a bus or virtual bus) supports the authentication specified.

```
Subject_Authentication: record
    (
    Authentication_Access_Type: enumeration (no_authentication, single_password,
smart_card,  ip_addr, two_factor, multi_layered, bio_metric, to_be_specified);
    Authentication_Protocol: enumeration (no_authentication, cert_services, EAP,
PAP, SPAP, CHAP, MS_CHAP, Radius, IAS, Kerberos, SSL, NTLM, to_be_specified) ;
    Authentication_Role: enumeration (no_authentication, authenticator, accessor);
    )
    applies to (abstract, system, bus, memory, device, processor, virtual processor,
virtual bus);
```

Cross Domain System Example

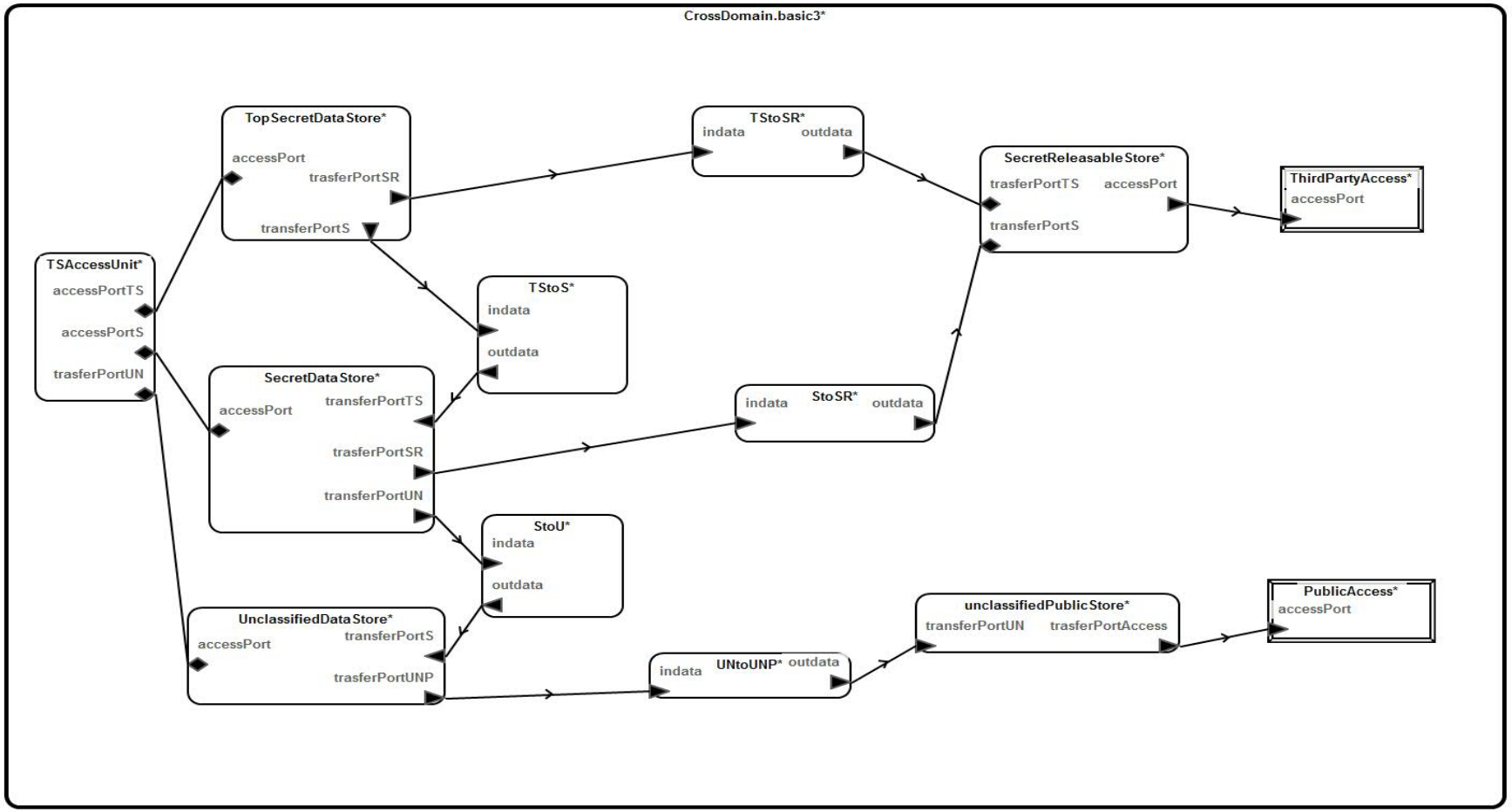
Cross Domain Solution

- three primary data stores (top secret, secret, and unclassified)
- two data stores for data that can be released (secret releasable and unclassified for public release).
- downgrading filters that downgrade top secret to secret, secret to unclassified, top secret to secret releasable, secret to secret releasable, and unclassified to unclassified public release.
- a super controller (subject) who can access and modify all three data stores

Examples are available at
https://github.com/osate/examples.git

Supporting files are available at
https://github.com/reteprelief/isse

Cross Domain AADL Model



Summary

Security Annex Summary

- Guidance and support for using AADL for specifying, modeling, and analyzing secure system architectures
- Core AADL with security-specific properties
- ALISA for policies and requirements capture
- Verification of requirements and other analyses with
  - ALISA assurance cases (Resolute and JAVA methods)
  - Libraries of claims and methods (Resolute and JAVA methods)
- Representative Examples
  - E-Enabled Aircraft
  - Cross Domain System
  - MILS