

Interaction Partnering Criteria for COTS Components

M. Kelkar M. Smith R. Gamble

Department of Mathematical and Computer Sciences

University of Tulsa

Tulsa, OK 74104 USA

gamble@utulsa.edu

Abstract

Commercial-off-the-Shelf (COTS) software provides a choice of products to streamline enterprise applications. COTS software integration can introduce security vulnerabilities due to mismatches between security constraints coupled with inadequate knowledge of interaction requirements. Though a component can be validated against its stand-alone functional and security requirements, two aspects of the validation for its integration are missing. First, no straightforward process exists to guide the developer in identifying integration-induced security risks. Second, interaction properties contributing security risks are not part of COTS product evaluation. In the former case, a process is needed to take advantage of selection criteria. In the latter case, interaction partnering criteria - criteria indicating how closely related the security constraints of two potentially communicating components are - must be defined. We examine these issues by defining initial interaction partnering criteria and exploring their use in a security profile for COTS components.

1. Introduction

Composing COTS products into larger systems provides a cost-effective and streamlined solution to increase functionality. However, COTS components do not interact seamlessly. Their insertion into application integrations leads to both functional (communication, data, and control exchange) [1] and non-functional (performance, reliability, and security) [2] interoperability problems.

COTS component security characterization provides an initial foundation for expressing necessary security properties whose mismatched values can cause security-based interoperability conflicts [3]. Template development for uniform characteristic representation

is still needed if comparisons are to yield understandable, deterministic results. Moreover, a secure composition of components can be guaranteed only if the template covers a broad range of security properties related to many kinds of threats.

Defining security properties and mechanisms of COTS components in a uniform manner is a necessary step toward building a set of criteria to evaluate individual components interacting within an integrated system [4, 5]. Research on evaluating security properties deals mainly with product offerings and has not yet addressed evaluating products for interaction capabilities in an enterprise system while complying with security constraints. To do this, characteristics should be acquired from reliable sources, assessed for relevance, and organized for comparison.

Our objective is to develop a methodology to determine the best interaction partners among COTS components. We define *interaction partnering* among COTS components with respect to security as *the uninhibited exchange of data and control information between two components that complies with defined security constraints and requires minimal intervention of external processing code*. Interaction partnering analysis helps integrators limit adaptation mechanisms required for component integration.

The interaction partnering concept reduces security vulnerabilities by bringing direct and induced security property mismatches to the forefront of the design. This is achieved by generating a *security profile* - a uniform representation of component security properties whose comparison among interacting components yields informal metrics related to the amount of external processing needed to adapt each component to maintain security compliance in an integrated system. Through these metrics, assessments can be made to determine which components make better interaction partners. If interaction partnering is conditional, the security mismatches due to conflicting policies are

addressed both locally at component level and globally at integrated system level.

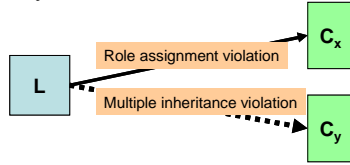


Figure 1: Interaction Partnering Challenge

For example, assume a company has a legacy component, L (Figure 1). They want a COTS component that complies with specific security criteria. The choice is between C_x and C_y . Closer scrutiny of C_x reveals that it establishes an inter-component role inheritance which contradicts the security policy of L, which for a user with role R, disallows access to a role hierarchically superior to role R. Thus, C_x allows a role assignment violation when interacting with legacy component L. C_y offers rule inheritance which allows a user to inherit access control inappropriately from two parent roles having conflicting privileges. This leads to multiple inheritance security violations. Though both policies conflict with L, C_x is a better interaction partner because minimal external processing will be needed to restrict the inter-component role inheritance.

2. Related Work

Component-based software engineering approaches provide definitions of component properties at the architectural level [1, 6]. The defined interface properties mainly consider the functional aspects of component interactions. However, conflicts are only minimally addressed when interacting component definitions do not match.

Component interface signatures incorporate properties, operations, and events as elements observable from outside the component along with the constraints and relationships between these elements [7, 8]. Direct replacement can occur by components with the same signature. Such approaches are based on functional properties, and although non-functional properties like security are addressed, the full depth and breadth of non-functional property expression is still difficult to achieve.

Software safety research examines COTS products with respect to integration, certification, and maintenance, given a defined interaction between components within an application [9]. This approach yields a COTS component selection process involving system criticality analysis to detect hazards and identify COTS component failure modes. A *safety contract* – a safety agreement between selected COTS components and the system – is used for system safety certification.

Safety contracts have analogous goals to security profiling for COTS.

Certain security specifications for COTS components are defined uniformly to evaluate individual COTS components [4, 5, 10]. However, security properties for an integrated system with COTS software are not considered. Where approaches provide samples of security mismatches by two interacting components, no definitive process for detecting mismatches is uniformly applied.

For security certification of an integrated COTS system, observable security properties can be defined at the atomic, composition, and global levels using logic programming, developing contracts between components for compatibility checking [3]. This provides a framework for composers to test the impact of security on component functionality by exposing trust-related attributes of one component to be captured by another component. A *Compositional security Contract* (CsC) performs the checks. The objectives of CsC research closely relates to ours. The difference is that CsCs are based solely on Common Criteria [11] whereas our approach takes a broader view of security properties. If CsC confirms that the properties are matched, a viable solution exists. However, when mismatch occurs, the CsC and candidate component are discarded instead of trying to find an integration solution that can be certified.

3. Distillation Process

Because of the number and range of security properties (e.g., confidentiality, integrity, availability, and non-repudiation), we approach the creation of security profiles for COTS components by defining a reusable process with which we accumulate, categorize, and assess a single property group. In this case, we examine properties related to access control conflicts. The reusable process, DIPC (Distilling Interaction Partnering Criteria) is shown in Figure 2. The goal for DIPC is to command a level of abstraction among properties that (1) have direct impact on design information, (2) can be applied to and valued by COTS products, and (3) maintain a high degree of reusability across different products and security policies.

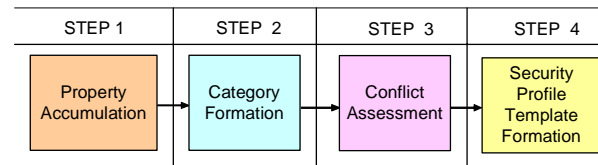


Figure 2: Distilling Interaction Partnering Criteria

The remainder of Section 3 focuses on the first two steps of DIPC. We discuss evaluation and conflict assessment of Step 3 in Section 4 and the security profile template formation of Step 4 in Section 5.

3.1. Property Accumulation

The easiest, but perhaps most tedious, way to acquire properties is through the perusal of published sources that address the understanding, coordinating, certifying, verifying, and assessing of security properties. Some predominant sources include government and military documents such as Common Criteria [11], DITSCAP [12], and the NIST Information System's certification document [13], all of which mention the need to assess critical security features for correct and complete implementation of an integrated system. These documents are primarily directives for the integrator and certifier to review the interfaces between components and determine their security compliance. Unfortunately, they lack explanation for compliance checking, ascertaining contributing properties, and identifying the conflicts that cause non-compliance.

The best sources for property accumulation are published accounts of security property expression, use, and analysis. Limiting our initial DIPC accumulation needs to access control properties, we researched documents that directly address policy conflicts (e.g., [14]), emerging vulnerabilities from system interactions (e.g., [15]), mismatches of access control use and intent (e.g., [16]), and references to COTS components and their access control constraints (e.g., [17]). A few publications directly consider access control as defined for a component [3, 18, 19].

Another DIPC task is to filter property information so that definitions are at the same level of abstraction. The key to filtering is to immediately establish a classification scheme starting with the first property and grouping properties as they are examined. This scheme may be refined multiple times. However, each refinement usually induces a smaller set of properties that are more tightly coupled within the same classification or are more distinct across multiple classifications. For example, early passes at classifying access control properties put them in a specific access control model (e.g., RBAC). We then refined that classification according to the formal language representation used (e.g., role-based) [3, 20-23]

Successive refinements resolve generalized definitions to accommodate various access control models and their respective languages. For example, authentication tokens can be defined differently on a component level than on the access control model level. A Web Service provider component, may call the

information required from the client a token (identity-based) [16] while access control models may refer to authentication tokens as the actual user name and password pair (identity-based) or associated encryption key (credential-based). We resolve the semantic problem by defining authentication token as authorization data and use identity-based and credential-based as values for more specific information (Table 1).

Many property statements are straightforward. Exceptions require matching statements to known properties or waiting until a new property emerges for further match. For instance, "*requires agreement on protocol, syntax and semantics for each piece of shared authorization data* [14]" brings authorization data to the forefront as an entity to consider. Less obvious is the authorization data entity in the following statement: "*There might be a potential conflict regarding the identification a Web service provider requires to provide the service, and the identification a Web service client is willing to reveal* [17]. "

Redundancy can be problematic unless only standard definitions are used, if they exist [1]. Redundancy indicates importance, even if there is some conflict among the property definitions. The DIPC process regards and eliminates redundancy by (a) taking the union of definitions and values and assigning them to a single property or (b) partitioning the conflicting pieces of the definitions across multiple, non-redundant properties. For example, temporal security properties have overlapping definitions across publications, some of which are called session based properties. These we group under the property *temporal* (Table 2).

The granularity issue requires rephrasing low-level properties to fit with more coarse-grained descriptions. For example, XML based access control research defines access as subjects having various rights on different parts of an XML document [14, 15, 24]. Some Web service papers define access as the ability to discover and invoke a web service and have its WSDL document be based on WS-security standards [16, 17, 24, 25]. Since a privilege level must be defined to see which domain is given priority, these properties are represented by *Privilege Level* (Table 3).

Once a set of properties is established, we proceed to place them in more concrete categories. We describe these categories in the next section.

3.2 Category Formation

The categorization of accumulated properties provides more substance to relationships among properties. For a property to be placed into a category, it must have distinct values that can be expressed at

relatively the same granularity as the values of other properties in that category. This better organizes them into a security profile, determine dependencies in and among categories, and facilitate conflict resolution. It is also the first step in reusing DIPC because as more properties become relevant, their placement in a category will inherit the organizational distinctions that have already been analyzed.

Because we used an opportunistic classification scheme in the first DIPC accumulation step, we must not rely solely on that for categorization. Rather, we ground DIPC in the application framework of integrated systems relying on COTS components. In this context, access control is generally defined as allowing or disallowing components to perform a specified operation on other components based on the security attributes of those components [14, 16]. This objective strategy, along with the examination of the resulting properties, yields an initial three categories: (1) exchanging authorization data, (2) controlling access to the components, and (3) strategically combining the policies which define access data and control in a system. We validate the worthiness of the categories by checking that the majority of accumulated properties can align with only one. Different valuations of certain properties may be needed for complete alignment. In one case, a property is further divided across two categories.

The first category is directly related to data issues with the properties that fall under *authorization data* are shown in Table 1. The second category focuses on control issues and is appropriately called *access control*. These properties are related to functional restrictions to information. Thus, in this category we have the access inheritance, separation of duty (SOD), cardinality, and temporal constraints (Table 2). This category is formed by grouping the properties which are based on user-based and role-based access control models. Properties arising due to implementation of other access control models that are either independent or extensions of these basic models are not in this table.

The properties under *access control policy combining strategies* (Table 3) are based on the precedence of action defined in the policy, comprehensiveness of the access policy definition, configuration of the component, order of the access policy, or privilege level defined in the policy. Usually components can have multiple policy combining strategies which are stated according to priority. The precedence type of policy combining strategy is usually given first preference. Thus, if one of the interacting components allows access and the other one denies access, the decision of which policy overrides depends

on the precedence rule defined. If both components allow or deny access but one defines policy in more detailed form with some exceptions in it, then the strategy next-in-line is applied. The decision application for this strategy is based on the type of policies that need to be combined.

4. Evaluation and Conflict Assessment

Once the categories are established, the goal for evaluation and assessment is two-fold. First, we interpret the issues that arise from mismatched values in a single category. Then we extend the comparisons across tables. Methodological advancement of this goal needs further empirical testing to validate the true presence of the conflicts that emerge. However, the initial categories form only a foundation for three major types of conflicts between interacting components. These conflicts are interoperability conflicts, security conflicts, and discrepancy conflicts.

4.1 Interoperability Conflicts

Interoperability conflicts are considered to be high-risk and occur when access can be defined for the individual component but not in a global context. Components are unable to interact if mismatches occur for properties within the Authorization Data Category (Table 1), which produces a high level of security vulnerability. Resolving these conflicts requires an extensive amount of glue code and wrappers to be evaluated and inserted into the system. For example, two components using identity-based and credential-based access control respectively have no way to communicate authentication tokens correctly without using some form of escrow service. Without resolving this issue, access remains undefined in the global context. Because access remains undefined, the security properties in this category carry more weight than properties under other categories.

4.2 Security Conflicts

A security conflict occurs when access is allowed or denied in a single domain and is similarly not allowed or denied across all domains in the system. This type of conflict is more likely to happen when one or more access control properties exist (Table 2). These values are not evaluated on a comparison basis like the interoperability conflicts in Table 1. The existence of these properties shows the need for more in-depth evaluation techniques. Security conflicts are considered less severe than interoperability conflicts because the resolution strategies are generally less extensive. However, the level of possible security risk involved remains the same.

Table 1: Authorization Data Category

PROPERTY	Authorization Data Type	Authorization Data Interactivity	Interpretation of Authorization Data
DEFINITION	Format used to produce or accept authorization token	Number of times a component produces and/or accepts authorization data	How a component responds to the received authorization data and what it expects
POSSIBLE VALUES	None	None	None
	Identity-based: Something the component <i>knows</i> (passwords, mother's maiden name, etc)	Single Sign-On: Credentials are accepted or produced only one time - sign on once for multiple systems	Abduction : Replies back and expects reply if the credentials are valid and access is allowed
	Credential-based: Something the component <i>has</i> (third party item, encryption keys, etc)	Multi-Step Sign On: Challenge-Response - additional credentials may be required during authentication process	Deduction: Replies back and expects a reply if the access is denied or failed

Table 2: Access Control Category

PROPERTY	Inheritance	Separation of Duty	Cardinality	Temporal
DEFINITION	Ordering of access control policy elements.	Two conflicting entities can not be simultaneously assigned to a particular entity	Numerical restrictions on access control entity assignment	Time for which access is defined (session based, time dependent)
POSSIBLE VALUES	None	None	None	None
	User-based: Seniority levels are defined for users in an access control policy	User Based: Two conflicting users cannot access a role simultaneously	User-based: Number of users that can be assigned to a certain role/rule	User-based: Time that a user can be assigned or activate a particular role or rule
	Role-based: Roles defined in an access control policy are hierarchical	Role Based: A user can not access two conflicting roles simultaneously	Role-based: Number of roles which can be assigned to a particular user	Role-based: Time for which a role can be assigned to a particular user

Table 3: Access Policy Combining Category

PROPERTY	Precedence	Granularity	Configuration	Applicability	Privilege Level
DEFINITION	What action defined in a set of policies override	Policies are given priority based on completeness or comprehensiveness	Master or slave component policies override the other	Access control statements are evaluated in the order they are found	The policy having least or most privilege is given priority
POSSIBLE VALUES	None	None	None	None	None
	Deny: If any statement denies access, access is denied	Fine-grained: More detailed policy takes priority over general policy	Master: Master component policies override slave	First Applicable: Evaluation in First come first serve basis	Least Privilege: The policy having least privilege is given priority
	Permit: If any statement permits access, access is permitted	Course-grained: Properties with less detailed information gets priority	Slave: Slave component policies override master	Only One applicable: If more than one policy is found to be applicable, then the result is indeterminate	Most Privilege: The policy having most privilege is given priority

Proper mapping between users and roles can alleviate the need to use bulky glue code. Such accurate mapping between components becomes more challenging with each additional access control property used by the component. All of the properties in this category allow a component to have multiple implementations of the property, leading to an added layer of possible conflict evaluation complexity.

4.3 Discrepancy Conflicts

Discrepancy conflicts encapsulate conflicts arising from the Access Policy Combining Category (Table 3). Even if mismatches occur, components will still be able to interact with each other, provided they have proper

access definitions. Some access policies may remain unaddressed if conflicts arise.

For example, say a server (master) component is interacting with a client (slave) component where the client uses a least-privilege policy combining strategy and the server uses a most-privilege policy combining strategy. A discrepancy remains as to which strategy should be followed for combining policies—master/slave or most-/least- privilege. By redefining individual component policy statements or declaring a global combining strategy that trumps the component policies, the conflict might be resolved.

5. Security Profile Template Formation

Each security property discussed in the category tables forms initial set of interaction partnering criteria for a component. These criteria build the security profile for a component where the corresponding value for each property is stated. Security profiles of components can then be compared to detect mismatches in the partnering criteria, which lead to the above-mentioned conflicts. This helps integrators identify the type of conflicts caused by a pair of components attempting to interact with each other. Further refinement to the profile is part of ongoing research.

6. Conclusion

In this paper, we describe the initial phases of DIPC to determine relevant security interaction partnering criteria for COTS components. Through this process, we elevate access control properties to a common granularity of expression whose values can be easily established via the documented interface and security mechanisms of a COTS product. This helps bridge the gap between ad hoc implementation of integration solutions that may cause added security problems and the design of a security compliant integrated system. This is because the property definitions and values facilitate an evaluation of component interactions using a security profile to detect possible mismatches.

Acknowledgement: This work is supported in part by a US AFOSR award FA9550-05-1-0374.

7. References

- [1] L. Davis, R. Gamble, and J. Payton, "The impact of Component Architectures on Interoperability," *Journal of Systems and Software*, 2002.
- [2] C. M. Geisterfer and S. Ghosh, "Software Component Specification: A Study in Perspective of Component Selection and Reuse," Int'l Conf. on COTS based Software Systems, Florida, 2006.
- [3] K. M. Khan and J. Han, "Deriving Systems Level Security Properties of Component Based Composite Systems," Australian Software Engineering Conf., 2005.
- [4] W. J. Lloyd, "A Common Criteria Based Approach for COTs Component Selection," *J. Object Technology*, vol. 4, pp. 27-34, 2005.
- [5] Q. Zhong and N. Edwards, "Security Control for COTS Components," *IEEE Computer*, 31(6):67-73, June 1998.
- [6] L. Davis, et al., "A Notation for Problematic Architecture Interactions," European Software Eng. and ACM Foundations of Software Eng., Austria, 2001.
- [7] J. Han, "A Comprehensive Interface Definition Framework for Software Components," Asia-Pacific Software Eng. Conf., 1998.
- [8] Y. Myers and I. Exman, "Software Codons for Fast Program Reassembly from Components," IEEE Int'l Conf. on Software-Science, Technology & Eng., 2003.
- [9] F. Ye and T. Kelly, "Use of COTS Components in Safety Critical Applications – A Defensible Approach," Int'l Workshop on Models and Processes for the Evaluation of COTS Components, 2004.
- [10] S. A. Hissam, D. Carney, and D. Plakosh, "DoD Security Needs and COTs-Based Systems," *SEI Monographs on the Use of Commercial Software in Govt. Systems*, 1998.
- [11] "Common Criteria for Information Technology Security Evaluation," <http://www.commoncriteriaportal.org/public/files/ccpart2v2.3.pdf>, June 2005.
- [12] "Department of Defense Information Technology Security Certification and Accreditation Process," http://www.dtic.mil/whs/directives/corres/pdf/85101m_0700/p85101m.pdf, July 2000.
- [13] R. Ross, et al., "Guide for the Security Certification and Accreditation of Federal Information Systems," NIST Special Publication 800-37, May 2004.
- [14] M. Lorch, et al., "First Experiences using XACML for Access Control in Distributed Systems," ACM Workshop on XML Security, Fairfax, VA, 2003.
- [15] A. Gummadi, J. Yoon, B. Shah, and V. Raghavan, "A Bitmap-Based Access Control for Restricted Views of XML Documents," ACM Workshop on XML Security, Fairfax, VA, 2003.
- [16] F. Koshutanski and F. Massacci, "An Access Control Framework for Business Processes for Web Services," ACM Workshop on XML Security, Fairfax, VA, 2003.
- [17] R. Kraft, "Designing a Distributed Access Control Processor for Network Services on the Web," ACM Workshop on XML Security, 2002.
- [18] R. Bhatti, et al., "X-GTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control," *ACM Trans. On Information and System Security*, vol. 8, pp. 388-423, 2005.
- [19] M. J. Covington, et al., "Securing Context-Aware Applications Using Environment Roles," ACM Symp. on Access Control Models and Technologies, 2001.
- [20] D. F. Ferraiolo, S. Gavrila, V. Hu, and D. R. Kuhn, "Composing and Combining Policies under the Policy Machine," ACM Symp. on Access Control Models and Technologies, Sweden, 2005.
- [21] R. J. Hulsebosch, et al., "Context Sensitive Access Control," ACM Symp. on Access Control Models and Technologies, Sweden, 2005.
- [22] I. Ray, N. Li, R. France, and D. Kim, "Using UML to Visualize Role-Based Access Control constraints," ACM Symp. on Access Control Models and Technology, New York, 2004.
- [23] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park, "Formal Model and Policy Specification of Usage Control," *ACM Trans. on Information and System Security*, vol. 8, pp. 351-387, 2005.
- [24] S. D. Di Vimercati, S. Marrara, and P. Samarati, "An Access Control Model for Querying XML Data," Workshop on Secure Web Services, Fairfax, VA, 2005.
- [25] XACML, "eXtensible Access Control Markup Language (XACML) version 2.0," T. Moses, Ed. 2005.