Modeling and Validating the Class Security Model Using Alloy

Ramzi A. Haraty, Nancy Boss, Azzam Mourad, and Mohamad Mowafak Allaham
Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon 1102 2801
Email: rharaty@lau.edu.lb

Abstract

Formalizing security models provide system designers and security engineers with evidence that they are constructing a consistent system that will meet the specifications as implemented. While it would be difficult to formalize every security model that has ever been developed or proposed, we present formal approving to ascertain secrecy properties of the Class Security Model. We use the Alloy language and analyzer for this formalism. We present the five model descriptions and show consistency proofs.

Keywords: Alloy, confidentiality, integrity, and object-oriented polices, and modeling and validating.

1. INTRODUCTION

The goal of information systems is to control or manage the access of subjects (users, processes) to objects (data, programs). This control is governed by a set of rules and objectives called a security policy. Data integrity is defined as "the quality, correctness, authenticity, and accuracy of information stored within an information system" [1]. Systems integrity is the successful and correct operation of information resources. Integrity models are used to describe what needs to be done to enforce the information integrity policies. There are three goals of integrity:

- Prevent unauthorized modifications.
- Maintain internal and external consistency.
- Prevent authorized but improper modifications.

Before developing a system, one needs to describe formally its components and the relationships between them by building a model. The model needs to be analyzed and checked to figure out possible bugs and problems. Thus, formalizing integrity security models helps designers building a consistent system that meets its requirements and respects the three goals of integrity. This objective can be achieved through the Alloy language.

Alloy is a structural modeling language for software design. It is based on first order logic that makes use of variables, quantifiers and predicates (Boolean functions) [2]. Alloy, developed by MIT (Daniel Jackson and his team), is mainly used to analyze object models, translates constraints to Boolean formulas (predicates) and then validates them using the Alloy Analyzer [3] by checking the code for conformance to a specification. Alloy is used in modeling policies, security models and applications, including name servers, network configuration protocols, access control, telephony, scheduling, document structuring, and cryptography [4]. Alloy's approach demonstrates that it is possible to establish a framework for formally representing a program implementation and for formalizing the security rules defined by a security policy, enabling the verification of that program representation for adherence to the security policy [5][6]. Additionally, it allows users to describe a system design and check that there is no miss-understanding before writing the code.

This paper presents a method for modeling different policies, such as confidentiality, integrity and hybrid policies using well-known security models in multilevel, conflict of interest and object oriented systems. Multilevel systems are systems in which subjects and objects are hierarchically or partially ordered according to their sensitivity levels.

This paper also demonstrates how models can be checked for consistency. We apply our method to commonly known Class Security model [7]. We use the formal language Alloy to define the models and policies and the Alloy analyzer tool to validate consistency. The model forms the basis of security policies for systems that cater for security as well as integrity. The Class Security model is tailored for multilevel secure object oriented database management systems based on artificial intelligence techniques.

This remainder of this chapter is organized as follows: Section 2 provides related work. Section 3 presents the Alloy language and its features. Section 4 presents the model descriptions and discusses their consistency proof; and section 5 concludes the work.

978-1-880843-93-2 / copyright ISCA, CAINE 2013
September 25-27, 2013, Los Angeles, California, USA

2. RELATED WORK

McLean [8] stated that security models are "used to describe any formal statement of a system's confidentiality, availability, or integrity requirements." Security models provide a detailed and precise means of formally describing security policies, and proving their validity. Formalizing security models provide system designers with evidence that they are constructing a consistent system that will meet its specifications when implemented.

Confidentiality policies prevent secret data from being leaked out to the adversary in information flow, while integrity policies restrict the use of data coming from the adversary. Despite the practical interests, integrity policies are often less formally studied in the literature [9]. In [10] confidentiality policies are defined as: "Each policy is a function that specifies how the data can be released to the public in the future. When this function is applied to the annotated data, the result is considered public." Confidentiality policies that were proposed by Bell and LaPadula [11] are common in military and governmental systems.

Integrity refers to the trustworthiness of data or resources. Integrity policies are usually characterized by preventing unauthorized users from making modifications to data or programs, preventing authorized users from making improper or unauthorized modifications, and maintaining internal and external consistency of data and programs. Integrity policies are defined in terms that specify how the data was computed in the past. Biba [12] studied the nature of integrity systems and proposed a security model. Lipner [13] combines the confidentiality and integrity policies and proposed a security model by mixing the Bell-LaPadula's and Biba's model.

Hybrid security policies address both confidentiality and integrity in addition to conflict of interests [14]. The Chinese Wall Model refers equally to confidentiality and integrity. In addition it specifically addresses conflict of interests. The implementation of this model has been successfully used as a defense in cases involving criminal charges in the UK [15].

In [16], the author discusses the architecture, security policy, and protection mechanisms of four National Security Agency – certified systems. The author formally compares their techniques used for protecting data against users.

In [17], the authors present a temporal multilevel secure data model. The model combines the characteristics of temporal data models and multilevel secure data models. The main focus of the model is mandatory access control, polyinstantiation, and secure transaction processing, while at the same time providing time support to record historical, present, and future data.

Hassan and Logrippo [18] proposed a method to detect inconsistencies of multiple security policies mixed together in one system and to report the inconsistencies at the time when the system is designed. The mixed models are checked for inconsistencies before real implementation. Inconsistency in a mixed model is due to the fact that the used models are incompatible and cannot be mixed. They demonstrated their method by mixing Bell-LaPadula with Role Based Access Control (RBAC) [19] in addition to Separation of Concerns.

Shaffer in [20] described a security Domain Model (DM), designed for conducting static analysis of programs to identify illicit information flows, such as control dependency flaws and covert channel vulnerabilities. The model includes a formal definition for trusted subjects, which are granted privileges to perform system operations that require mandatory access control policy mechanisms imposed on normal subjects, but are trusted not to degrade system security. The DM defines the concepts of program state, information flow and security policy rules, and specifies the behavior of a target program.

Misic and Misic in [21] addressed the networking and security architecture of healthcare information system. This system includes patient sensor networks, wireless local area networks belonging to organizational units at different level of hierarchy, and the central medical database that holds the results of patient examinations and other relevant medical records. In order to protect the integrity and privacy of medical data, they targeted the Clinical Information System Security policy and proposed the feasible enforcement mechanisms over the wireless hop.

3. FORMAL SECURITY POLICY MODELS IN ALLOY

In this section, we overview the Alloy language and demonstrate how models can be checked for consistency using Alloy and apply our method to commonly known Class Security Model.

3.1 The Alloy Language

To formalize the security models we use the Alloy language and its analyzer. Alloy is a light-weight modeling formalism using a first order predicate logic over the domain of relations. These relations are similar to relational algebra and calculus. It is a textual language developed at MIT. Alloy originates from Z. It is used for analyzing object models by checking for consistency of multiplicities and generating instances of models or a

counter example. Alloy analyzer translates constraints to Boolean formulas, and then applies SAT solvers.

3.2 Alloy Language Features

The following features present a subset of the full Alloy language that we used in formalizing our security models.

An Alloy model consists of one or more files, each containing a single module. A module consists of a header identifying the module, some imports and some paragraphs:

*module::=header import*paragraph**

A model can be contained entirely within one module. The paragraphs of module are signatures, facts, functions, predicates, assertions, run commands and check commands.

Alloy uses the following multiplicity keywords, *lone*: zero or one; *one*: exactly one; *some*: one or more; *set*: zero or more. These keywords are used as quantifiers in quantified formulas, quantified expressions, in set declarations, in relation declarations and in signature declarations.

A signature represents a set of atoms and is declared using the '*sig*' keyword - such as *sig A {}* to define a signature named *A*. The types of signatures are: *subset*, *top-level*, and *abstract*, and a signature with a *multiplicity* keyword:

- A top-level signature represents mutually disjoint sets that does not extend another signature: *sig A{}*
- A subset signature represents a set of elements that is a subset of the union of its parents: *one sig B extends A{}*
- An abstract signature represents only the elements that belong to one of the signatures that extend it: *abstract sig A{}*
- A signature with *multiplicity* keyword constrains the signature's set to have the number of elements specified with the keyword.

Facts, functions and predicates are packages of constraints. A fact is a constraint that always holds. A predicate is a template for a constraint that can be instantiated in different contexts. A function is a template for an expression, and an assertion is a constraint that is intended to follow from the facts of a model. Examples of facts, predicates, and assertions are:

fact {no iden & parent}

pred access(state: State, next: state, u: User, r: Resource)
{next.accessed = state.accessed + u ->r}

assert example1 {
A.sens = SecretNT
B.sens = SecretT}

Run and *Check* commands are used to instruct the Alloy Analyzer to perform various analyses, a *run* command causes the analyzer to search for an instance that shows the consistency of a function or a predicate, whereas, a check command causes it to search for a counterexample showing that an assertion does not hold:

check example
run BibaModel

4. THE CLASS SECURITY MODEL

In [7], the author presented a security model for a multilevel secure object oriented database management systems (MLS/OODBs). Several business and military systems require a MLS/OOD but few studies address their security. In an MLS/OODB, database users are assigned classifications levels, and data items are assigned sensitivity levels. An MLS/OODB is responsible to ensure that users can access only those data items for which they have been granted permission.

4.1 Security Classes and Constraints

The author of [7] used the standard military security approach that consists of two components: a set of security classes and a set of non-hierarchical compartments. The security classes are absolutely ordered from the lowest to the highest as follows: unclassified < confidential < secret < top secret. Within each security class there can be zero or more compartments; for example, conventional, chemical, and nuclear. Subjects represent users, or the processes that execute on behalf of users. Users are trusted, but processes are not. Objects, on the other hand, correspond to data items. The author proposed additional policies to ensure security. These constraints can be summarized in the following policies:

The Class Security Policy: The sensitivity level of a class shall be identical to or lower than the sensitivity level of its subclasses.

The Instance Security Policy: The sensitivity level of all instances (objects) of a class shall be identical to or higher than that of its class.

Those policies guarantee that proper access to objects will not be violated directly.

4.2 Formal Class Security Model

In formalizing the Class Security Model in Alloy, we use four security classes ordered from the lowest to highest – Unclassified -> Confidential -> Secret -> Top Secret. Within each security class there can be zero or more compartments. The compartments are Nuclear, Traditional and Biological.

Definition: A security class S1 dominates security class S2 if and only if S1 is hierarchically higher than S2 and contains all of its compartments. Our model consists of ordering departments and sub departments in a hierarchical order with a parent relation linking a department with its sub departments.

The model starts by defining hierarchical order for the subDepartment signature:
open util/ordering[subDepartment]

Compartments, Classification, Department and subDepartments are defined as signatures. A classification represents a security class, which can be Top Secret, Secret, Confidential and Unclassified:
abstract sig Classification {sensitivity: set Compartment}

For example, Unclassified will not contain any compartment, whereas, a Confidential security class can contain no compartment, or Biological(B), Traditional(T), Nuclear(N), Biological and Nuclear (BN), Biological and Traditional(BT), Nuclear and Traditional(NT) or Biological, Traditional and Nuclear(BTN).

Each classification has a sensitivity that contains a set of compartments: '*set*' in Alloy is used to define zero or more compartments. The classifications we use in our model are the following: Unclassified, Confidential, ConfidentialN, ConfidentialB, ConfidentialT, ConfidentialBT, ConfidentialNT, ConfidentialBN, ConfidentialBTN, Secret, SecretB, SecretT, SecretN, SecretBT, SecretBN, SecretNT, SecretBTN, TopSecret, TopSecretB, TopSecretN, TopSecretT, TopSecretBT, TopSecretBN, TopSecretNT, and TopSecretBTN.

We define the Unclassified classification, Confidential classification with no sensitivity, ConfidentialB classification with Biological compartment, ConfidentialT with Traditional compartment, and ConfidentialBT with Biological and Traditional Compartment as:
one sig Unclassified extends Classification{}
one sig Confidential extends Classification{}
one sig ConfidentialB extends Classification{}{sensitivity = Biological}
one sig ConfidentialT extends Classification{}{sensitivity = Traditional}
one sig ConfidentialBT extends Classification{}{sensitivity = Biological + Traditional}

In our model we prevent cycles of length one to appear. We implement this property by including a fact:
{no this & parent}
that forces a model with no subdepartment parent to itself. To prevent cycles of length two to appear another fact is defined:
fact { no iden & parent }

For example, if subdepartment A is parent to subdepartment B, B cannot be a parent to A.

The Class Security Policy is defined to ensure security for object-oriented settings. The policy states that the sensitivity level of a class shall be identical to or lower than the sensitivity level of its subclass. And the sensitivity level of all instances of a class shall be identical to or higher than that of its class. For example, Unclassified is the lowest security class in the model; thus, it can be parent to any subclass with any compartment; whereas, other security classes follow the hierarchical order Confidential < Secret < TopSecret. A Confidential security class can be a parent to a Secret security class and parent to TopSecret security class. The difficulty comes when ordering security classes that contain compartments. In addition to ordering the security class, we also order the compartments. Figure 1 describes the parent relation between compartments. For example, B can be parent to B, BT, BN and BTN; T parent to T, NT, BN, and BTN; N parent to N, NT, BN and BTN; BT, BN, and NT parent to BTN and itself; and BTN can be a parent to itself only.

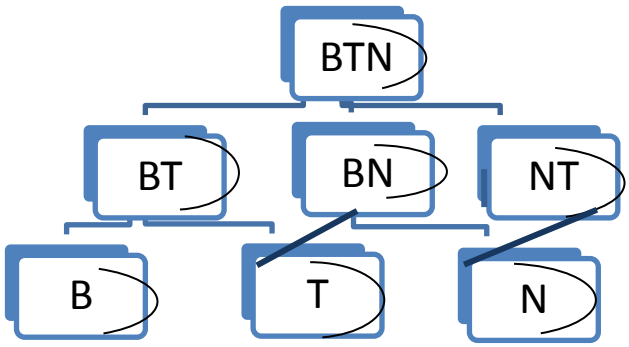

Figure 1 Compartment tree order.

We define the not allowed parent relation (i.e., the one that violates the Class Security Policy), by defining all the impossible parent relation between subDepartment with security level and its compartments:
no((Confidential.~sens).parent & ConfidentialB.~sens)

This statement states that there is no parent relation between departments having Confidential security class and departments having Confidential security class with Biology compartment.

After defining the Class Security Policy for all sensitivity classes, we call on Alloy to generate a random model, forcing A to be parent to B, B parent to C and D, C parent to D, D parent to E F parent to G, G parent to H and I, H parent to I and I parent to J:

fact {
A.parent = B
B.parent = C + D
C.parent = D
D.parent = E + F
E.parent = F
F.parent = G
G.parent = H + I
H.parent = I
I.parent= J}

4.3 Class Security Model and Alloy Analysis

Figure 2 shows the result produced by the logic analyzer. The model is consistent and there is an instance found.

```
Executing "Run ClassSecurityPolicy"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  14708 vars. 535 primary vars. 33570 clauses. 361ms.
  Instance found. Predicate is consistent. 39ms.
```

Figure 2 Alloy output showing consistency.

The random instance is shown in figure 3, where subdepartment A is parent to subdepartment B, B parent to C and D, C parent to D, D parent to E and F, E parent to F, F parent to G, G parent to H and I, H parent to I and I parent to J as defined in the model. Alloy forces the subDepartments to be on specific security levels that obeys the Class Security Policy in parent relation. In figure 3, there were no cycles of any length and all parent relations do not violate the Class Security Policy. Unclassified can be parent to ConfidentialB since Unclassified has a lower sensitivity level than ConfidentialB. ConfidentialB can be parent to ConfidentialBTN, since ConfidentialB has the same sensitivity level as ConfidentialBTN, and B is contained in BTN.

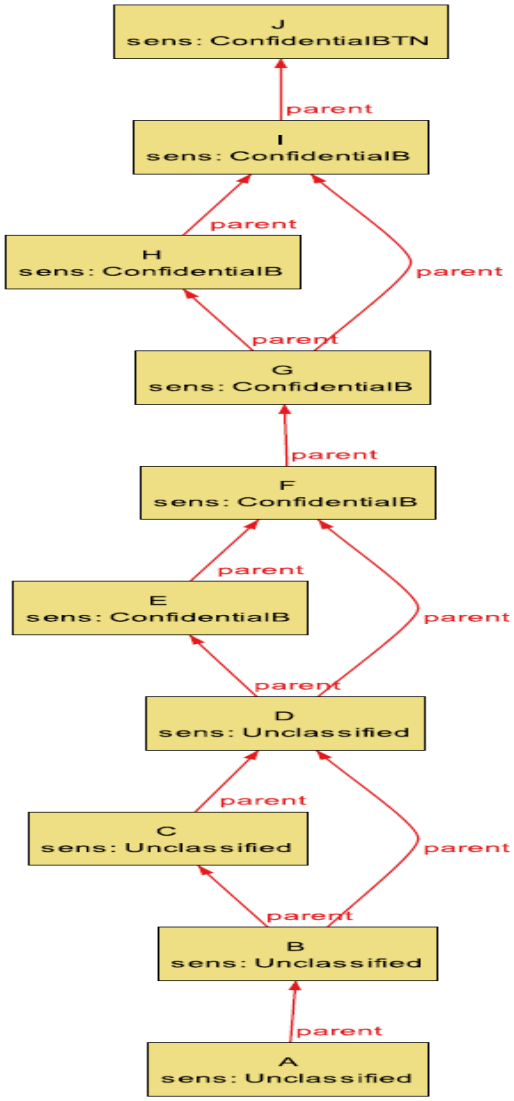

Figure 3 A random instance.

We then defined another fact to give each department a sensitivity level as the following:

fact example { A.sens = SecretNT
B.sens = SecretT
C.sens = SecretNT }

Running this example resulted in an inconsistent model. Figure 4 shows the result produced by the logic analyzer when the inconsistent model is checked. This model was inconsistent because it violates the Class Security Policy. A (SecretNT) cannot be parent to B (SecretT) since SecretNT is not contained in Secret.

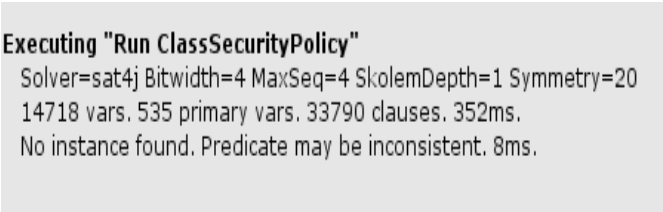

Figure 4 Alloy output showing inconsistency.

5. CONCLUSION

We presented the Class Security Model that represents confidentiality and object oriented policies. Then, we formalized the model to check their consistency. We applied our method to the model. We used system examples based on the defined security models. We used the formal language Alloy to define the models and policies, since it allows expressing systems as set of logical constraints in a logical language based on standard first-order logic. We also used the Alloy analyzer tool to validate consistency, using the examples. We specified the system users and subjects. Alloy then compiled a Boolean matrix for the constraints to check if a model is valid, or whether there are counterexamples. As for future work, we believe that more work can be done to integrate multiple secrecy models in a mixed mode, and formalize them to find potential interactions.

6. REFERENCES

[1] Summers, C. (1997). *Computer Security: Threats and Safeguards*. McGraw Hill. New York. ISBN-13: 978-0070694194.

[2] Jackson, D. (2000). *Alloy: A Lightweight Object Modeling Notation*, *Technical Report 797*, MIT Laboratory for Computer Science, Cambridge, MA.

[3] Jackson, D., Schechter, I., and Shlyakhter I. (2000). *Alcoa: the Alloy Constraint Analyzer*, Proceedings of the International Conference on Software Engineering, Limerick, Ireland.

[4] Wallace, C. (2003). *Using Alloy in Process Modelling*, Information and Software Technology Journal, ISSN: 0950-5849, pp. 1031-1043.

[5] Jackson, D. (2012). *Alloy 3.0 Reference Manual*. Retrieved on April 1, 2012 from: http://alloy.mit.edu/reference-manual.pdf.

[6] Seater, R. and Dennis, G. (2012). *Tutorial for Alloy Analyzer 4.0*. Retrieved on April 1, 2012 from: http://alloy.mit.edu/tutorial4.

[7] Haraty, R. A. (1999). *A Security Policy Manager for Multilevel Secure Object Oriented Database Management Systems*, Proceedings of the IASTED International Conference-Applied Modeling and Simulation, Cairns, Australia.

[8] McLean, J. (1994). *Encyclopedia of Software Engineering,* (ed. John Marciniak), Wiley & Sons, Inc.

[9] Peng, L., Mao, Y., and Zdancewic, S. (2003). *Information Integrity Policies*, Proceedings of the Workshop on Formal Aspects in Security and Trust (FAST), Pisa, Italy.

[10] Peng L. and Zdancewic, S. (2005). *Unifying Confidentiality and Integrity in Downgrading Policies*, Proceedings of the LICS'05 Affliated Workshop on Foundation of Computer Security (FCS), Chicago, Illinois.

[11] Bell, D. E. and LaPadula, L. J. (1973). *Secure Computer Systems: Mathematical Foundations*, The Mitre Corporation, Technical Report 2547, Volume I.

[12] Biba, K. J. (1977). *Integrity Considerations for Secure Computer Systems*, Technical Report MTR-3153, The Mitre Corporation.

[13] Lipner, S. B. (1982). *Non-Discretionary Controls for Commercial Applications*, Proceedings of the Symposium on Security and Privacy, Oakland, CA: IEEE Computer Society, pp. 2-10.

[14] Mankai, M. and Logrippo, L. (2005). *Access Control Policies: Modeling and Validation*, Proceedings of the Fifth NOTERE Conference, Gatineau, Canada, pp. 85-91.

[15] Financial Services Authority (2011), *Rules Chapter III*, London, UK.

[16] Haraty, R. A. (1999). *C2 Secure Database Management Systems – A Comparative Study*. Proceedings of the ACM Symposium on Applied Computing. San Antonio, Texas.

[17] Haraty, R. A. and Bekaii, N. (2006). *Towards a Temporal Multilevel Secure Database.* Journal of Computer Science. ISSN: 1549-3636. Volume 2, Number 1.

[18] Hassan, W. and Logrippo, L. (2009). *Detecting Inconsistencies of Mixed Secrecy Models and Business Policies*, University of Ottawa, Technical Report, Ottawa, Canada.

[19] Ferraiolo, D. F. and Kuhn, D. R. (1992). *Role-Based Access Control*, Proceedings of the 15th National Computer Security Conference, Baltimore, MD.

[20] Shaffer, A. Auguston, M., Irvine, C. and Levin, T. (2008). *A Security Domain Model to Assess Software for Exploitable Covert Channels*. Proceedings of the ACM SIGPLAN Third Workshop on Programming Languages and Analysis for Security, Tucson, Arizona. ACM Press, pp. 45-56.

[21] Misic, J. and Misic, V. (2007). *Implementation of Security Policy for Clinical Information Systems over Wireless Sensor Networks*. Ad Hoc Networks Journal 5, pp. 134–144. ISSN: 1570-8705.