Issues and Challenges of Secure Policy Specification Languages

Sailaja Arsi[1], Venkata N. Inukollu[1], Joseph E. Urban[2]
Computer Science Department[1]
Industrial Engineering Department[2]
Texas Tech University
Lubbock, Texas 79409 USA
{sailaja.arsi, narasimha.inukollu, joseph.urban}@ttu.edu

Abstract - **Security policies which describe the behavior of a system through specific rules are becoming an increasingly popular approach for static and dynamic environment applications. The SANS top 20 critical controls are a de facto standard in the software industry to protect against cyber crime. This paper shows the importance of applying the SANS critical controls to a product for producing effective results. This paper provides a policy framework, issues that a secure policy specification language faces, and challenges for secure policy specification languages.**

Keywords **-** Secure Specification Language, Policy, Policy Specification Language, Software Development Life Cycle (SDLC), SANS Critical Controls.

1 INTRODUCTION

A security policy can be defined as a set of rules that specifies the specific behavior of a system [1] and includes all the constraints within. There is a need to represent the security policies in a formal/informal specification language. Software engineering is an application of engineering to software, which is indeed a significant, methodical, and disciplined approach to representation, development, performance, and maintenance of software. Security is a component of software engineering. Due to advancements in technology, secure software engineering [2] has become an important aspect/asset of software quality. In the software development life cycle [3] (SDLC), for effective software development, security as a process should be considered at the same priority as the life cycle phase's functionalities. The idea of incorporating security into software from the beginning of development has gained acceptance. Secure software engineering is required throughout the software development life cycle.

A main goal of secure software engineering is the gathering of security requirements, design, development, maintenance, verification, and validation of secure and functioning software. In secure software engineering, during the life cycle phases, from the initial phase to deployment phase, confidentiality, integrity, and availability objectives are specified.

There is a need to add security in the requirements phase itself in order to reduce the time, cost, quality, and resources at the end of the deployment phase, if any problem occurs. For specifying the secure requirements, there should be a medium for writing the secure requirements in a formal specification language that is understandable by both stakeholders and developers. Designers/developers should follow the secure policy specifications for further development of the software. Risk management [4] is well handled if industries follow the secure policy specifications, especially in secure engineering aspects.

There are several specification languages which consider policy specifications at a primary level, but ignore some of the issues that might be considered to have a crucial role in the industry. A secure policy specification language has a major role for a successful product, if industries neglect secure policy specification languages and start the product development, then the product can lead to problems at the deployment phase, which increases cost, time, risk, and resources and reduces quality and scope. The specification languages have different models that may not have the proper structure to express the given system policy details or may have some unspecified important aspects.

In this paper, the main focus is on explaining the issues of secure policy specification languages and proposing some of the challenges of secure policy specification languages. Based on the literature, the issues of policy specification language can be addressed by user defined security policies. This paper covers the motivation behind user defined security policies, advantages, and implementation methods.

2 RELATED WORK

In order to develop a complete functioning system, an organization, project, or business should effectively use an SDLC process. The SDLC has five phases' requirements, specification, design, testing, and deployment, which includes a minimum requirement of security tasks that are required to integrate security in a software development process. In order to protect the critical assets and infrastructure, every organization must and should follow the SANS critical controls [5] to improve cyber security. Some of the specification languages developed earlier and the constraints associated with those languages have been reviewed briefly.

Garcia Clemente et al. [6] defined policy framework requirements and also defined semantics that are applied to web information systems protection. The authors worked on comparing a non-semantic security policy framework (Ponder and XACML) with the semantic security policy frameworks (Rei, KAoS, and SWRL). The paper listed the advantages of a semantic security policy framework over a non-semantic security policy framework based on approach, specification language, tools used for specification, and enforcement.

An executable specification policy language S-Promela was explained by Abbassi et al. [1]. This model handles an authorization rule, obligation rule, and prohibition rule in detail with structured representation. S-Promela is an executable specification language that supports the validation task. The paper did not have a structured representation for the delegation rule. A disadvantage is that S-Promela is a high level specification language, which could be difficult for a stakeholder to understand.

Damianou et al. [7] focused on the non-semantic based policy specification language called, Ponder, which is a declarative, strongly-typed, and object oriented language. Ponder has a rich set of policies. The Ponder policy specification language paper explained the policy rules of authorization, obligation, refrain, delegation, and information filtering. The policy rules are in a high level language.

XACML (Extensible Access Control Markup Language) is a declarative access control policy language that has been implemented in XML [8]. XACML follows a syntax that is defined in a schema, which is associated with the XML namespace. Not all stakeholders have familiarity with the XML language and it can be considered as a downside for having XACML as a specification language.

Rei was implemented by Kagal [9], a semantic based policy specification language, which provides certain constructs based on the concepts concerning duties or obligations. Rei has defined policy objects, action specifications, constraint specifications, speech acts, and meta policies. The Rei specification language has domain independent ontologies and has a flexible framework.

More recently, Finn et al. [10] studied the relationship between the Web Ontology Language (the authors named it as OWL) and the Role Based Access Control (RBAC) model. OWL is not designed to express an authorization policy because it is a web ontology language. The authors took the challenge to build a security framework for open and dynamic environments with a primary goal of defining OWL ontologies. The OWL ontology can be used to represent the RBAC security model and to show the specifications and implementation of access control systems, in order to identify the systems part of portions for which RBAC can be modeled with description logic and the other parts with logical reasoning.

There are 20 critical security controls popularly called as SANS Critical Controls that are published by the Center for Strategic International Studies (CSIS) to improve cyber security and are prioritized mitigation steps [11]. Tracking and precise automation of these top 20 critical controls has demonstrated more than a 90% reduction [11] in measured security risk within the U.S. State Department. A product which follows these 20 critical controls is successful and handles risk and real-time vulnerabilities in a system. The top 20 critical controls have different levels of impact on attack mitigation ranging from very high to low (see table). If the software or the product ignores handling any control, the impact of risk is more effective on the product. SANS critical controls are the industry standards for reducing cybercrime.

3 POLICY LANGUAGE FRAMEWORK

A policy specification language should be able to represent and maintain policies effectively. A policy specification language uses different entities, such as subject/source, object/target, actions, and constraints/conditions/restrictions. A policy needs to be defined in such a way that, the policy can handle security issues, such as confidentiality, availability, and integrity [12]. A policy specification language should be:

- well-defined with clear and unambiguous syntax and structure;
- flexible to represent new policies in the future;
- able to detect conflicts;
- able to provide extensibility features for the future policies of the same language version; and
- able to validate the existing policies.

The following are some commonly used policies [1, 6, 7, 8, 9, 10] in specification languages (authorization, prohibition, obligation, delegation, information filtering, and refrain policies):

a) *Authorization policy*: An authorization policy grants access to resources by evaluating and validating a given request. For example, consider ATM transactions. A client requesting to either withdraw or deposit money. The request will be served only when the client provides a valid pin number. For this transaction, an authorization policy is applied.
b) *Prohibition policy*: A prohibition policy is the negation of the authorization policy. A prohibition policy prevents access to the resources by validating the set of attributes. For example, when a client requests the account details of other clients or tries to perform unauthorized operations, such as withdraw, then the request will be denied.
c) *Obligation policy*: An obligation policy performs certain immediate actions that are forced to perform to the occurrence of a specific event in a system. For example, in ATM banking, users should change their PIN/password on the beginning of the month to protect from fraud.
d) *Delegation policy*: A delegation policy grants privileges/rights from a higher level to a lower level hierarchy. For example, in a company, when a team member wants to read and write certain security files, the team member needs to get permission from the team manager.
e) *Information filtering policy*: An information filtering policy [7] is needed to transform the information either input or output parameters in an action. For example, a payroll clerk is only permitted to read personnel records of employees below a particular grade.
f) *Refrain policy*: A refrain policy [7] rejects the actions that a subject needs to perform on target even though they are permitted to perform the action. For example, in a development team, test engineers must not disclose the test results to developers or analysts when the testing is in progress. Analysts and developers would probably not object to receiving the results.

Almost all the specification languages which are in existence use the above defined common policies and apply these policies to the systems environment.

4 ISSUES OF SECURE POLICY SPECIFICATION LANGUAGES

Policies are applied to a wide range of fields from web applications to real-time embedded applications. There are certain issues related to secure specification languages. Some of the issues are described below.

a) *Policy Representation:* From the six specification languages [1, 6, 7, 8, 9, 10] the security policies defined, have almost the same semantics, but the representations differ. Specification languages [6, 7, 9] use a high level language syntax for defining the specifications and all the security-aware semantic specifications are in a high level language. The access control policy rules in the Ponder specification language are defined in a high level object oriented language, which can create a barrier between a developer and customer, as customers might not be aware of the language, thus resulting in a customer not understanding the requirements. A specification language should be defined in such a way that the secure specifications are understandable by both stakeholders [13] and developers. To produce a successful product, developers or the organization should consider a stakeholder's interest. Therefore, the representation of the requirements should be understandable by the stakeholder in the first place. Representation of requirements is a main issue in specification languages, so as to define the requirements in a language that can be understandable by a customer, as well as a development team.

b) *Define Policies:* A policy language should be well-defined with clear and unambiguous syntax and structure. A policy architecture [5] also needs to have a well-defined independent interface. For some scenarios, specification languages do not provide any policy, which implies that the policies given by specification languages are not all sufficient enough for any environment. For some specific behaviors of a system, developers of various specification languages could not categorize what are the objects for a system, how to maintain a subject, what are the actions to perform, and how to define and implement constraints of a systems behavior.

c) *Validation of Policies:* Defining a policy is an easy aspect compared to other aspects of policy specification languages. Each entity in a policy needs to be validated. In order to validate the policy, policy specification frameworks need to collect all the facts and figures of policy data. The validation of constraints / restrictions / conditions is the critical phase in validation of policies. The REI and Ponder policy specification languages [7, 9] partially support validation of polices.

d) *Policy Extendibility:* Policy extendibility is an important issue as the feature defines the new policy definitions and implementations. When the specification languages are taken into consideration, an important feature to look at is - how far the specification languages are extendable in terms of adding a new policy to its behavior. Most of the specification languages have almost the same common policies (authorization, prohibition, and obligation) and some languages have a few more policies in addition to the common policies that are in use. Even though specification languages defined policies, not all of the policies are applicable to every environment. For the above discussed reason, extendibility becomes an important issue in specification languages.

e) *Portability of Policy:* There are usually several languages that can be used in different domains to express similar policies. Portability can be defined as an ease of the policy that is being applied to different domains. For example, consider a delegation policy. In the S-Promela specification language, Abbassi et al. [1] did not focus on applying the delegation rule to the environment. Simply, the policy loses the portability functionality. When specification languages provide policies, then the policies should possess the portability functionality within the language.

f) *Policy Conflicts:* Specification languages should have a conflict detection technique, which should be able to check that a given policy does not conflict with any other existing policy. Most of the specification languages ignore this functionality. For example, in an organization, if an IT team wants to restrict the "controlled use of administrative privileges (SANS critical control 12) [4]" then the IT technicians may get confused as to which policy needs to be assigned to get the task done, i.e., either authorization, prohibition, or delegation raises a policy conflict issue.

g) *Priority of Policies:* Prioritizing the policy is another feature to consider. When specification languages do not handle policy conflicts, then at least there should be prioritization of policies. There is no implication that, if policies do not handle policy conflicts then policies handle policy prioritization. Specification languages should handle both policy conflicts and policy priorities. When some situation like checking the inventory of authorized and unauthorized devices/software [4], which policy needs to be applied, authorization or prohibition. For scenarios such as above, we need to have a prioritization of policies.

5 CHALLENGES OF SECURE POLICY SPECIFICATION LANGUAGES

The challenges of secure specification languages are described below. These challenges are the observations from the related work and literature review.

a) *Representation of Non Functional Aspects:*

Security is one of the critical components of non-functional requirements. A secure policy represents a non-functional aspect of a system. Elicitation and analysis of non-functional requirements still have considerable challenges [14, 15]. The specification, analysis, trade-off, and the documentation of security requirements has been an area which is left almost unexplored by software and requirements engineering research [16].

b) *Dynamic Behavior Representation:*

A specification is represented with various technologies based on syntax, semantics, and diagram representations [6, 7, 8, 17]. Static behavior of a system is a simple and straight forward approach for any specification language, irrespective of the technology. Complexity increases by the introduction of dynamic behavior. Dynamic representation involves several challenges because of the following reasons:

- policy description consists of both functional and non-functional requirements;
- policy is abstract information which will not provide detailed information about the behavior of a system; and
- unpredictable behavior of a system for a given policy.

c) *Secure Policy Representation of Distributed Systems:*

A distributed system is a collection of different components over a network to carry a single task by communicating with different components using various communications methods. Functional specification of a distributed system is represented with various specification languages [18, 19, 20]. Secure policy representation of distributed systems are challenging because of the following reasons:

- a distributed system is a hybrid system which contains various components, such as web clients and network applications, hence defining a single secure policy that describes an entire distributed system is difficult;
- handling of policy priorities and resolution of policy conflicts are good research challenges if there exists a policy specification language that specifies distributed systems; and
- dynamic behavior of distributed systems are unpredictable and open many challenges.

d) *Different Specification Languages for Security and Privacy:*

The policy specification languages given by [5, 6, 7, 9] are for security. Rei, Ponder, and S-Promela are applied to environments with security as main aspects and thus called as secure policy specification languages. Security and privacy are closely related technologies, but both are two different areas. Security is about protection. Security specifies how information is being protected from malicious actors or other unwanted parties that are trying to exploit the security of a system. Privacy is about governance, informational self-determination, and use. Privacy specifically makes sure the policies and rules are in place to ensure the information is being collected, shared, and used in appropriate ways. Moreover, security is necessary, but not sufficient for addressing privacy. There has been limited research on applying specification languages for privacy aspects.

Table I : The SANS 20 critical controls applied to respective policies of specification languages

SANS Critical Controls	**Impact on Attack Mitigation [11]**	**Policy Specification Languages**		
		S-Promela	**Ponder**	**Rei**
1) Inventory of authorized and unauthorized devices.	Very high	✓	✓	✓
2) Inventory of authorized and unauthorized software.	Very high	✓	✓	✓
3) Secure configurations for h/w and s/w on mobile devices, laptops, work stations, and servers.	Very high	×	×	×
4) Continuous vulnerability assessment and remediation.	Very high	×	×	×
5) Malware defenses.	High/Medium	✓	✓	✓
6) Application software security.	High	×	×	×
7) Wireless device control.	High	✓	✓	✓
8) Data recovery capability.	Medium	✓	✓	✓
9) Security skills assessment and appropriate training to fill gaps.	Medium	×	×	×
10) Secure configurations for network devices such as firewalls, routers, and switches.	High/Medium	✓	✓	✓
11) Limitation and control of network ports, protocol, and services.	High/Medium	✓	✓	✓
12) Controlled use of administrative privileges.	High/Medium	Handles Partially	✓	✓
13) Boundary defense.	High/Medium	✓	✓	✓
14) Maintenance, monitoring, and analysis of audit logs.	Medium	×	✓	×
15) Controlled access based on the need to know.	Medium	×	×	×
16) Account monitoring and control.	Medium	✓	✓	✓
17) Data loss prevention.	Medium/Low	✓	✓	✓
18) Incident response and management.	Medium	✓	✓	✓
19) Secure network engineering.	Low	×	×	×
20) Penetration test and red team exercises.	Low	×	×	×

✓ - The SANS critical control is handled by the specification language.
× - The SANS critical control that is not handled by the specification language.

6 SUMMARY AND FUTURE WORK

The policy specification languages describe the security policies in the requirements and specification phase of the software development life cycle. The Policy specification languages are based on syntax, semantics, and are model driven. The policy specification languages have un-resolved issues and challenges due to the language constructs and constraints. Secure policy specification languages have issues, that can be resolved by adding a common policy with dynamic functionalitites. The future work is on defining a dynamic common policy for handling the issues of security policy specification languages and satisfying the challenges of secure policy specification languages, that are mentioned in this paper. Security has a vital role in policy specifications and defining a new policy for each environment in the system would be inappropriate. So, the future is on defining a user defined security policy with dynamic functionalities. As the name specifies a user is given the main responsibility for maintaining security and privacy of the user.

ACKNOWLEDGMENTS

This research was partially supported by the National Science Foundation (Grant No. 1241735). Opinions, findings, and conclusions/recommendations are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Abbassi R. and El Fatmi S. G., "S-promela: An Executable Specification Security Policies Language," *Proceedings of the First International Conference on Communications and Networking, 2009,* Hammamet, November 3-6, 2009, pp. 1-8.

[2] Essafi M., Labed L., and Ben Ghezala H., "Towards a Comprehensive View of Secure Software Engineering," *Proceedings of the International Conference on Emerging Security Information, Systems, and Technologies, 2007,* Valencia, October 14-20, 2007, pp. 181-186.

[3] Yu W. D. and Le K., "Towards a Secure Software Development Lifecycle With SQUARE+R," *Proceedings of the IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), 2012*, Izmir, July 16-20, 2012, pp. 565-570.

[4] Ebert C., Murthy B. K., and Jha N. N., "Managing Risks in Global Software Engineering," *Proceedings of the IEEE International Conference on Global Software Engineering, 2008,* Bangalore, August 17-20, pp. 131-140.

[5] SANS Critical Security Controls. *Critical Security Controls for Effective Cyber Defense*: http://www.sans.org/critical-security-controls accessed June 12. 2014.

[6] Garcia Clemente F. J., Perez G. M., Botía Blaya J. A., and Gómez Skarmeta A. F., "Representing Security Policies in Web Information Systems," *Proceedings of the 14th International World Wide Web Conference on Policy Management for the Web Workshop,* Chiba, Japan, May 10-14, 2005.

[7] Damianou N., Naranker D., Emil L., and Morris S., "The Ponder Policy Specification Language," *POLICY '01 Proceedings of the International Workshop on Policies for Distributed Systems and Networks,*

Springer-Verlag London, UK, 2001, pp. 18-38.

[8] *eXtensible Access Control Markup Language 3* (XACML) Version 2.0, OASIS, February 1, 2005.

[9] Kagal L., *Rei: A Policy Language for the Me-Centric Project*, HP Labs Technical Report, September 30, 2002.

[10] Finin T., Joshi A., Kagal L., Jianwei N., Sandhu R., William H. W., and Thuraisingham B., "ROWLBAC - Representing Role Based Access Control in OWL," *Proceedings of the 13th Symposium on Access Control Models and Technologies,* Estes Park, Colorado, USA, June 11-13, 2008.

[11] *Real-Time Auditing for the 20 Critical Security Controls,* Tenable Network Security Report, 2014.

[12] Nunes F. J. B., Belchior A. D., and Albuquerque A. B., "Security Engineering Approach to Support Software Security," *Proceedings of the 2010 6th World Congress* on *Services (SERVICES-1),* Miami, FL, July 5-10, 2010, pp. 48-55.

[13] van Lamsweerde, A., "Requirements Engineering: from Craft to Discipline", *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering,* pp. 238-249.

[14] Ullah S., Iqbal M., and Khan A. M., "A Survey on Issues in Non-Functional Requirements Elicitation," *Proceedings of the 2011 International Conference* on *Computer Networks and Information Technology (ICCNIT),* Abbottabad, July 11-13, 2011, pp. 333 - 340.

[15] Bajpai V. and Gorthi R. P., "On Non-Functional Requirements: A Survey," *Proceedings of the 2012 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS),* Bhopal, March 1-2, 2012, pp. 1-4.

[16] van Lamsweerde A., "Elaborating Security Requirements by Construction of Intentional Anti-Models," *Proceedings of the ICSE'04: 26th International Conference on Software Engineering*, Edinburgh, ACM-IEEE, 2004.

[17] Booch G., Jacobson I., and Rumbaugh J., *The Unified Modeling Language Reference Manual*, Essex, UK, Addison-Wesley Longman Ltd, 1999.

[18] Jorge Cortes G., and Felipe Rolando Menchaca G., "Graphical Specification Language for Distributed Systems," *Proceedings of the 15th International Conference on Computing, 2006,* Mexico City, November, 2006, pp. 385-390.

[19] Lambiri C., and Ionescu D., "Models for Distributed Systems Specification," *Proceedings of the Canadian Conference on Electrical and Computer Engineering, 1995,* Montreal, Quebec, September 5-8, 1995, pp. 813-816.

[20] Mikolajczak B., and Ottlik A., "Specification of Distributed Systems with Actors Using Object-Oriented Petri Nets," *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,* Nashville, TN, Volume 4, Oct 8-11, 2000, pp. 3134-3140.