SECURE DESCARTES: A SECURITY EXTENSION TO DESCARTES SPECIFICATION LANGUAGE

Venkata N Inukollu[1] and Joseph E Urban[2]

[1]Department of Computer Science and, Purdue University, Fort Wayne, USA
[2] Department of Computer Science, Arizona State University, USA

ABSTRACT

With increase in demand for the security aspects of software, every phase of the Software Development Life Cycle (SDLC) is experiencing major changes with respect to security. Security designers, developers, and testers are keen on improving various security aspects of a system. Specification of security requirements propagates to different phases of an SDLC and there exist different techniques and methodologies to specify security requirements. Business level security requirements are specified using policy specification languages. The current literature has specification languages that are domain based, web based, network based, syntax based, semantics based, predicate based, and protocol based. In this research effort, a generic secure policy prototype and components of the generic secure policy were defined using formal methods. The Descartes specification language, a formal executable specification language, has been developed to specify software systems. The development of a secure policy framework along with extended constructs of the Descartes specification language for specifying secure policies are some of the deliverables of this research effort. Concepts of secure policies were adopted from the SPromela, Ponder, and REI methodologies for secure policy specification, analysis, and design.

KEYWORDS

Policy language, Secure policy language, Formal methods, Descartes specification language, & SDLC

1. INTRODUCTION

Security is a critical aspect of the software development life cycle, starting from the requirements phase to the deployment phase. The software development life cycle is an interdependent process, where each phase depends on the previous phase as per the traditional software development methodologies, such as waterfall and iterative approaches. Software development typically starts from the requirements phase, where functional and non-functional requirements are described. Security requirements are considered either functional or non-functional requirements based on the context and software system to be specified [12,13,14,15] In this research effort, Security requirements are represented as policies and are derived from business rules that can be represented in two different ways [1], either by task level (i.e., low level) or business level (i.e., high level).

In a web system, a security policy can be defined as “a set of rules and practices describing how an organization manages, protects, and distributes sensitive information at several levels” [1]. The rules are defined by humans (business analysts) and so there is a high risk of ambiguity, false interpretation, and errors, which sometimes cannot be detected. If the errors in business rules are not detected and corrected in the early stages, problems would be faced in the later stages in terms of cost and time. Hence, formal specification languages help to specify, create, analyze, maintain, and protect the business rules from ambiguity and false interpretation. There are several specification languages that exist in the literature [2, 3, 4, 5, 6, 7, 8, 9, 10], which represent secure aspects of a system, but these specification languages are developed on various factors, such as domain, web, network, syntax, semantics, predicate, and protocol. There is no secure policy specification language that uses the full extent of formal methods, which help to specify secure aspects of the system. Hence, there is a need for a formal secure specification language that represents, analyzes, and validates the secure policies effectively.

The Descartes specification language is a formal specification language that has a formal method based syntax, semantics, and data structures. The primary goal of this research effort is to create a secure policy framework for the Descartes specification language that defines, analyzes, and validates the policies effectively by adding required extensions to the existing work.

2. RELATED WORK

2.1. Policy Languages and Frameworks

Secure aspects are represented in the form of high level rules, which formally constitute a policy. The specification languages which support different kinds of policies are called policy specification languages. Several authors have proposed policy specification languages, each language has its own assets and drawbacks that need to be improved. The security policies are represented in different formats depending on the syntax and semantics of the specification language. S-Promela [6], stands for Security-based Promela, which is an executable policy specification language, has been built on concepts and rules. S-Promela uses both syntax and semantic patterns as that of a regular programming language, such as C, C++, and Java. S-Promela defines the security policy as a "formal statement of the rules by which people that are given access to an organization technology and information assets must abide" [6]. Ponder [3] and XACML [5] are non-semantic based policy specification languages. Ponder is a simple and declarative language and it adapts object oriented features while representing security policies. Ponder has a set of policies that address different domain specific applications, such as networks and distributed systems. Subject, target, actions, and constraints are the four major components of Ponder, which describe domain specific services. Ponder classifies the policies as access control policies, obligation policies, group policies and meta policies. XACML (Extensible Access Control Markup Language) [5], is a specification language to address web services. XACML is a context based specification language and the context is represented in XML using an XML schema.

KAoS [6] and Rei [4] are semantic based policy specification languages. KAoS is a goal oriented specification language and is popular for behavioral specifications. KAoS defines policies, actions, actors, groups, and entities using pre-defined ontologies. The KAoS architecture consists of three layers, i.e., human interface, policy management, and enforcement. The human interface layer generates an ontology based vocabulary from natural language sentences. Rei [4] is a policy framework and has three simple language constructs: policy objects, meta policies, and speech acts to specify, create, and analyze security policies, respectively. Rights, prohibitions, obligations, and dispensations are the core concepts of the Rei specification language. Rei uses a “has” predicate to connect the core concepts to language constructs. The Rei policy objects include subject, action, conditions, and policy object.

The Graph Based Policy Framework (GBPF) [7] defines the policy as a set of rules that describe individual or collective behavior of a software system. In GBPF, security policies are represented using graphs. A graph, positive and negative constraints, and a set of rules are the components of the GBPF. A graph consists of a set of nodes and edges, constraints define the conditional behaviour, rules describe valid and invalid states of a system. GBPF represents access control polices using authorization and prohibition rules. this research effort adapted two validation approaches. The first evaluation approach was based on language features, such as syntax, semantics, data structures, policy representation, and policy validations [1, 15, 17]. The second method was based on critical security controls, such as inventory of authorized software, devices, and users, and controlled use of administrative privileges suggested by SANS [19]. Not one of the policy specification languages, handles all 20 critical security controls.

2.2. Descartes Specification Language

The Descartes specification language is a formal specification language developed by Urban [11] along with a language processor to execute specifications with abstract execution. The Descartes specification language is simple and has a well-defined syntax and semantics to write specifications effectively. Basic syntax and semantics of a Descartes specification are derived from a "Hoare tree" [11]. The Descartes specification language specifies a functional relationship between input and output, by specifying output data as a function of input and hence inputs and outputs are key aspects of the Descartes specification language. The core data structuring methods of Descartes are direct product, discriminated union, and sequence [11]. Every statement in Descartes is termed as a node. Nodes are created with meaningful names. Node names may use the underscore ("_") to create more meaningful names or to differentiate from other nodes. Indentation is used to represent sub nodes of a node in a tree. An example of a direct product in Descartes is:

```
full_name

        first_name
        last_name
```

The Descartes specification language has been extended to real-time systems by introducing a new primitive, "parallel", to represent the concurrent execution between two processes, and each process is represented with a Descartes module [20]. The object-oriented features, such as abstraction, inheritance, and aggregation were introduced using a "class" primitive [21]. Two keywords "abstract" and "interface" were introduced to the Descartes specification language to represent real-time object-oriented features [22, 28]. Agent features are represented with an "agent" keyword [23], and the process modeling features, such as activity, human actor, and tools are represented with extensions "activity", "actor", "role", and "tools" [24].

3. FRAMEWORK TO SPECIFY SECURITY POLICIES

Policy specification languages KAoS [3], Rei [4], and XACML [5] were not based on formal methods, hence they need a special tool to validate the policies externally. Ponder [2] and SPromela [6] policy specification languages use the syntax and semantics of high level programming languages such as C, C++, and Java, which makes it difficult for the stakeholders to understand. The high level programming language representation creates a communication barrier among stakeholders, specification engineers, and design engineers. Also, Ponder [2], KAoS [3], Rei [4], XACML [5], and S-Promela [6] cannot integrate security requirements in functional requirements due to the differences between policy specification languages and functional specification languages. In order to avoid these problems, this research effort created a framework for Descartes specification language, which is completely based on formal methods. The new framework can validate the policies in the framework using abstract execution, which can also be able to integrate the security requirements into functional requirements.

3.1. Framework to Specify Security Policies

The secure policy framework consists of five major components as shown in Figure 1 i.e., Policy entities/components, policies/rules, policy manager, policy knowledge base, and policy applications. The secure policy framework and components of the framework have been considered for extending the Descartes specification language to specify security policies. Each component in the framework has its own functionality and has a unique role. Each component is described in detailed in subsequent sections.

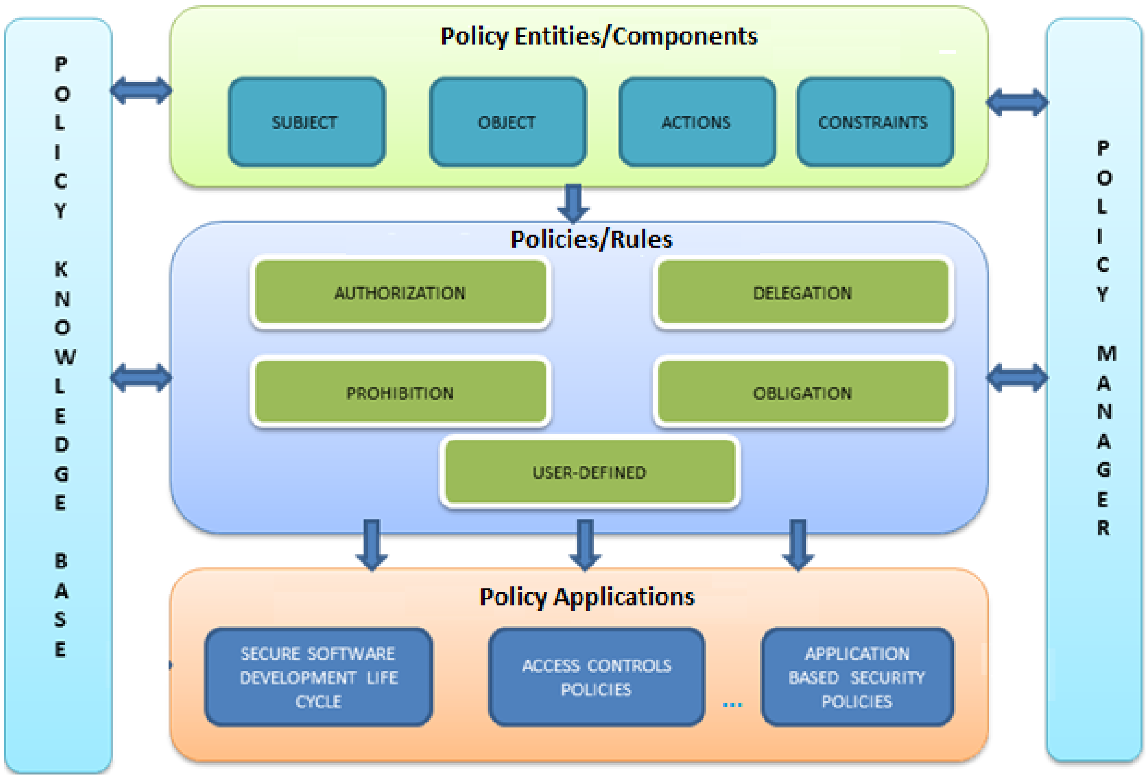


Figure 1: Secure Policy Framework

3.2. Policy Entities/Components

Policy entities/components are basic building blocks of the policy framework. Subject, object, actions, and constraints are the entities of secure policy framework. The concept of policy entities was adapted from Ponder [2], KAoS [3], Rei [4], XACML [5], and S-Promela [6]. Each entity describes the specific behavior of a system/component. The entities provide inputs to the different security policies. The secure policy framework is flexible in terms of adding a new entity to the existing entities. The facts and details of the entities exist in the policy knowledge base. Policy manager and entities interact with each other to create, configure, and maintain effective entities.

3.3. Policies/Rules

The second and a key component of the secure policy framework is defining secure policies with the help of policy entities/components. A policy/rule in the secure policy framework must be defined with the help of subject, object, actions, and constraints as show in the Figure 5. SPF is flexible in terms of adding a new policy to the existing policies. Authorization, delegation, prohibition, obligation, and user-defined are the basic policies.

3.4. Policy Applications

The policies/rules are extendable to represent access control policies, such as role based and group based policies. Applications of a policy/rule can help to improve software quality in a secure software development life cycle.

Secure software development life cycle: Applications of a secure policy can be extendable to a software development life cycle by writing secure specifications effectively, such as providing inputs to secure design, conversion of secure policy specification into high level programming code, and generating security test cases from a secure policy specification.

Access control policies: Role based access controls (RBAC) [26, 27, 28, 30] are used in both industry and research efforts. RBAC is a procedure to restrict the permissions/access to authorized users with the help of different roles. Secure policies can be extendable to represent access control policies.

Application based security policies: Secure policies can be capable of handling different domain applications, such as web applications, network applications, and embedded systems by introducing domain specific extensions.

3.5. Policy Knowledge Base

A policy knowledge base contains entities, data, and details about the policies. Policy history, such as policy conflicts, and priority of policies will be stored in the policy knowledge base. The term knowledge base resembles a database/data store in database management systems. A policy knowledge base will be a key component in validating secure policies using a Java compiler.

3.6. Policy Manager

A policy manager acts as a moderator in a secure policy framework. Maintaining policies, assigning priorities to policies, resolving conflicts in the policies are the key functionalities of the policy manager. Policy manager resolve the conflicts by assigning priorities to each secure policy and consider the order of policies.

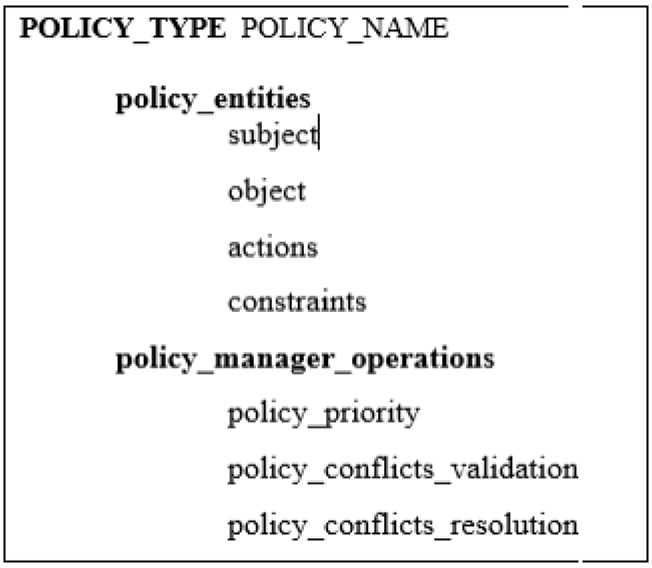
4. SECURE DESCARTES - SECURITY EXTENSIONS TO THE DESCARTES SPECIFICATION LANGUAGE

The Descartes specification language is a simple yet powerful formal specification language used to specify various systems, such as object oriented, real-time systems, real-time object oriented, agent systems, and process modelling. Existing work on the extensions available to the Descartes specification language do not define, specify, analyze, and validate the secure polices. The existing work was enhanced by this research effort.

In order to describe the security policies, basic entities/elements of security policies must be modeled. The policy type, subject, object, actions, and constraints are the basic entities of a security policy. To specify and validate security policies in the Descartes specification language, three concepts have been added in the form of extensions. The added concepts are: 1) policy entities; 2) policy type; and 3) policy manager operations. These extensions represent domain specific statements, expressions, or methods.

4.1. Abstract/High Level View of a Secure Policy

Abstract representation of a security policy consists of three parts: policy type, policy entities, and policy manager operations. Each part will be defined, described, and represented in the subsequent sections. A security policy is represented as shown in Figure 2.



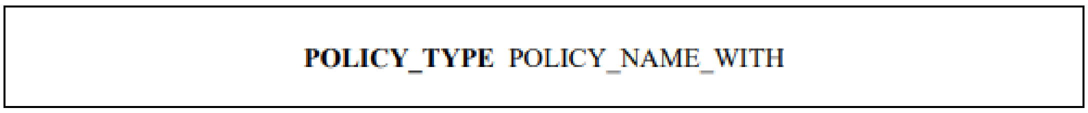
```
POLICY_TYPE POLICY_NAME

      policy_entities
            subject
            object
            actions
            constraints
      policy_manager_operations
            policy_priority
            policy_conflicts_validation
            policy_conflicts_resolution
```

Figure 2: Abstract/high level view of a secure policy

4.2. Policy Type

A security policy describes the set of rules to protect a software system, and those rules are represented with the help of policy entities such as subject, object, actions, and constraints. A security policy is represented as shown in Figure 3.



POLICY_TYPE POLICY_NAME_WITH

Figure 3: Representation of Policy Type

Policy type represents the type of policy and valid policy types are authorization, prohibition, delegation, obligation, or user-defined. These are new extensions introduced in the Descartes specification language in the form of primitives to represent the policy types. Authorization defines the access rights to resources by validating the received requests. Prohibition, also called negative authorization [2], prevents access to the resources by validating a set of attributes. Delegation grants rights or permissions or privileges from a higher level authority to a lower level in the hierarchy. An obligation policy describes the immediate actions that take place after a specific set of events in a system. A policy that cannot be considered as authorization, prohibition, delegation, or obligation, can be defined as user-defined policy. Policy name could be any meaningful name as per the Descartes specification language naming standards, which describes the output of a secure policy. Finally, POLICY_INPUTS represents one or more inputs for a given security policy.

An example representation of a policy type is specified as shown in Figure 4.

```
authorization   ALLOW_RESOURCE_ACCESS_WITH
```

Figure 4: Example of Policy Type Representation

Consider a security policy, an organization consisting of technical and non-technical staff, where authorized staff should be given access to organization resources. The goal of the security policy is to grant access rights to authorized people. Hence, policy type is specified as authorization.

4.3. Policy Entities

Subject, object, actions, and constraints are policy entities. The reserved keyword "policy_entities" is used to specify entities in a given security policy

4.3.1 Subject

Subject describes an active entity in a system, i.e., users, components, and subsystems. A user is a human actor who uses a software system, component is any software or hardware device which seeks information, and subsystem is part of a complex system which performs independent operations. Active entities are domain specific information. The reserved keyword "subject" is used to specify active entities in a given security policy.

4.3.2 Object

Object describes passive entities in a system. Passive entities are independent and have no processing capabilities, such as resources and service providers. Resources are assets in software system and assets are distributed through service provider The Ponder specification language refers to an object as a target [2]. The reserved keyword "object" is used to specify passive entities in a given security policy.

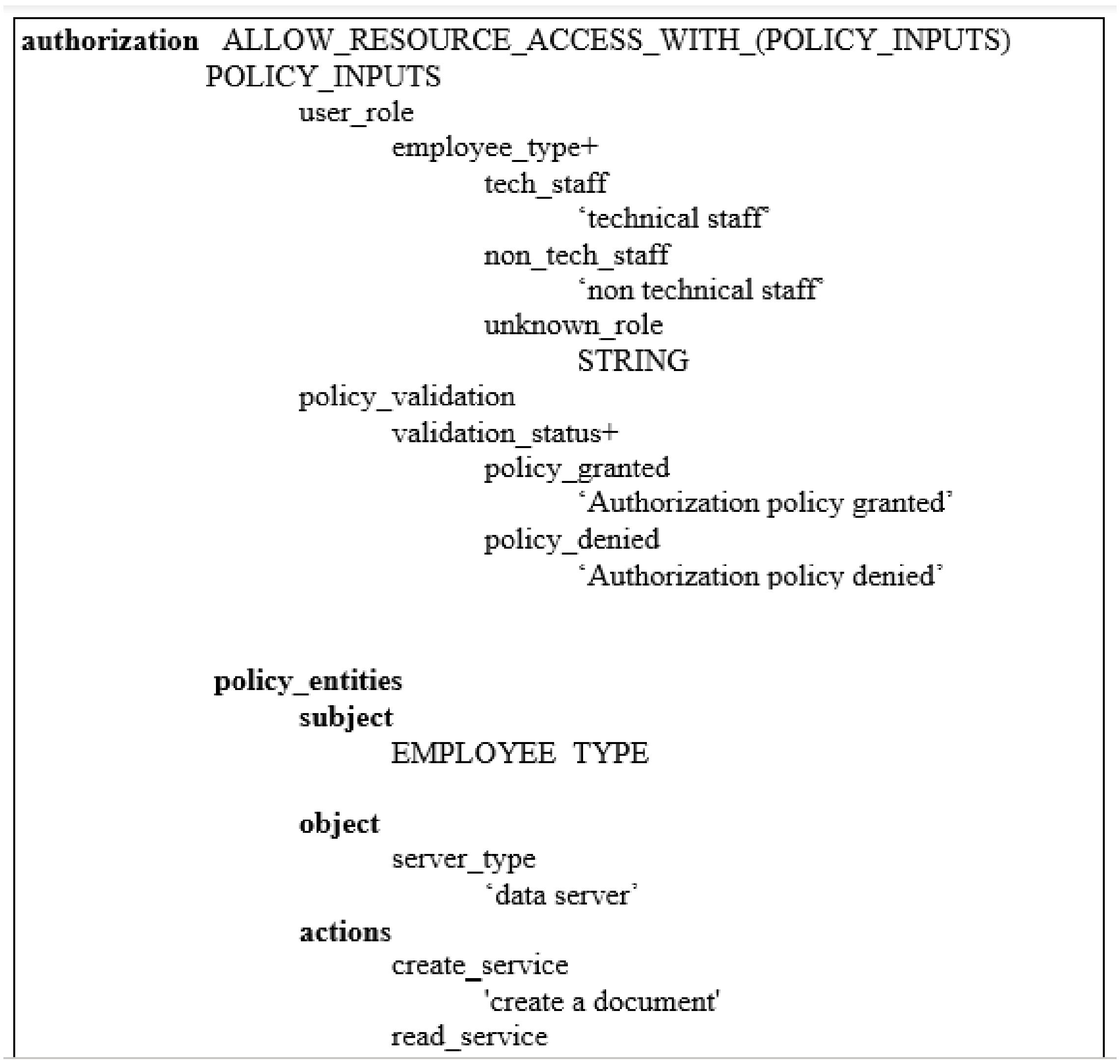
4.3.3 Actions

Actions describe tasks, processes, or methods that take place on an object when a subject satisfies the existing constraints in the policy. Tasks represent an activity, process represents a series of steps to achieve a goal, and methods define a procedure. The actions are domain specific tasks. The reserved keyword "actions" is used to specify the tasks, operations, or methods in a given security policy. 4.3.4 Constraints Constraints describe conditions or restrictions on existing policies. Constraints are inferred from subject, object, and actions. The reserved keyword "constraints" is used to specify conditions or restrictions in a given secure policy.

4.4. Secure Descartes Policy

Authorization defines the access rights to resources by validating the received request with a set of attributes. In secure Descartes, authorization represents the type of actions a subject can perform on a specified object by satisfying defined constraints. Authorization grants permission to perform the actions only if constraints are satisfied. The reserved keyword "authorization" is used to specify the secure policy type. An authorization policy is represented in secure Descartes as shown in Figure 5.

'*authorization*' is a keyword in secure Descartes and is followed by any meaningful policy name. The following case study provides a better understanding of the authorization policy.
Case study: In an organization, employees could be distinguished as technical staff and non-technical staff. Technical staff could have the permissions or rights to access the data of the company for processing company documents. Hence, the technical staff will be able to create, update, and delete the documents to effectively handle the operations of the company. Whereas, nontechnical staff could have limited access to organizational documents, for example reading documents. Non-technical staff are prohibited from updating or deleting documents. A secure Descartes specification for the case study is as shown in Figure 6.



```
authorization  ALLOW_RESOURCE_ACCESS_WITH_(POLICY_INPUTS)
               POLICY_INPUTS
                    user_role
                         employee_type+
                              tech_staff
                                   'technical staff'
                              non_tech_staff
                                   'non technical staff'
                              unknown_role
                                   STRING
                    policy_validation
                         validation_status+
                              policy_granted
                                   'Authorization policy granted'
                              policy_denied
                                   'Authorization policy denied'


               policy_entities
                    subject
                         EMPLOYEE  TYPE

                    object
                         server_type
                              'data server'
                    actions
                         create_service
                              'create a document'
                         read_service
                    actions
                     create_service
                                 'create a document'
                     read_service
                           'read a document'
                     update_service
                           'update a document'
                     delete_service
                           'delete a document'
                     constraints
                           constraint_selection+
                                 read_service_constraint
                                 NON_TECH_STAFF
                           read_and_write_service_constraint
                                 TECH_STAFF

      return
      EMPLOYEE_TYPE+
                     TECH_STAFF
                     READ_AND_WRITE_SERVICE_CONSTRAINT
                                 ACTIONS
                                 CREATE_SERVICE
                                    READ_SERVICE
                                       UPDATE_SERVICE
                                    DELETE_SERVICE
                                 POLICY_GRANTED
                     NON_TECH_STAFF
                     READ_ SERVICE _CONSTRAINT
                           READ_SERVICE
                              POLICY_GRANTED
                     UNKNOWN_ROLE
                           POLICY_DENIED
```

ALLOW_RESOURCE_ACCESS_WITH_ is a policy name in which subject specifies the employee role, and object is the data server. Create, update, and delete services represent actions, and constraints validate user role. POLICY_INPUTS consists of user role. If the employee type is either a technical staff or non-technical staff, then the policy is granted along with associated actions, otherwise the policy is denied.

4.5. Secure Descartes Policy

Three methods were used to validate the secure extensions made to the Descartes specification language. The first method was based on policy language features, the second method was based on the SANS critical security controls, and the third method was based on the IEEE Std. 830-1998. Secure Descartes satisfying 95% of all the features on each validation methods.

5. CONCLUSION AND FUTURE WORK

Designed and developed a generic policy framework to represent, analyze, and validate the secure policies. The Ponder [2], KAoS [3], Rei [4], XACML [5], S-Promela [6], GBPF [7], SPL [8], and TPL [9] policy specification languages were developed on various factors, such as domain, web, network, syntax, semantics, predicate, and protocol. Introduced a new policy type called "user-defined" to specify custom needs and user defined policies. Generating security test cases and test data from the secure Descartes polices. And generating secure design models and specifying secure polices using the object-oriented Descartes specification language extensions are some future research directions from this research effort.

REFERENCES

[1] García, F. J., Martínez, G., Botía, J. A., and Skarmeta, A. G., "Representing Security Policies in Web Information Systems," Proceedings of the 14th International World Wide Web Conference, 2005.

[2] Damianou, N., Naranker, D, Emil, L., and Morris, S., "The Ponder Policy Specification Language," Proceedings of the International Workshop on Policies for Distributed Systems and Networks, 2001, pp. 18-38.

[3] Kagal, L., Finin, T., and Joshua, A., "A Policy Language for Pervasive Computing Environment," Proceedings of the International Workshop on Policies for Distributed Systems and Networks, 2003, pp. 63-74.

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

[5] Godik, S., Anderson, A., Parducci, B., Humenn, P., and Vajjhala, S., OASIS eXtensible Access Control 2 Markup Language (XACML) 3, Technical Report, 2002.

[6] 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, pp. 1 - 8.

[7] Koch, M., and Parisi-Presicce, F., "Describing Policies with Graph Constraints and Rules," Proceedings of the First International Conference on Graph Transformation, 2002, pp. 223-238.

[8] Riberio, C., and Guedes, P., " An Access Control Language for Security Policies with Complex Constraints," Proceedings of Network and Distributed System Security Symposium, 2001.

[9] Herzberg, A., Mass, Y., Mihaeli, J., Naor, D., and Ravid, Y., "Access Control meets Public Key Infrastructure, or: Assigning Roles to Strangers," Proceedings of IEEE Symposium on Security and Privacy, 2000, pp. 2-14.

[10] ANSI/IEEE, IEEE std. 830-1998 Recommended Practice for Software Requirements Specification, (ANSI/IEEE), 1998.

[11] Urban, J. E., A Specification Language and Its Processor, Ph.D. Dissertation, University of Louisiana at Lafayette, 1977.

[12] van Lamsweerde, A., "Goal-oriented requirements engineering: A guided tour," Proceedings of Fifth IEEE International Symposium on Requirements Engineering, 2001, pp. 249-262.

[13] Glinz, M., "On Nonfunctional Requirements," Proceedings of 15th IEEE International Conference on Requirements Engineering, 2007, pp. 15-19.

[14] IEEE, Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.

[15] Jacobson, I., Rumbaugh, J., and Booch, G., The Unified Software Development Process. Reading: Addison Wesley, 1999.

[16] Firesmith, D., “Specifying Reusable Security Requirements,” Journal of Object Technology, 2004, pp. 61-75.

[17] Hall, A., "Seven Myths of Formal Methods," Software, IEEE , 2002, pp.11-19.

[18] Clarke, Edmund M., and Jeannette M. Wing. "Formal Methods: State of the Art and Future Directions." Journal of ACM Computing Surveys, 1996, pp. 626-643.

[19] SANS Critical Security Controls. Critical Security Controls for Effective Cyber Defense: http://www.sans.org/critical-security-controls accessed June 12, 2015.

[20] Sung, K. Y. and Urban, J. E., "Real-time Descartes: a Real-Time Specification Language," Proceedings of the Third Workshop on Future Trends in Distributed Computing Systems, 1992, pp. 79-85.

[21] Wu, Y., A Methodology for Deriving a Booch Object Oriented Design from Extensions to the Descartes Specification Language, MS Thesis, Arizona State University, Dec, 1994.

[22] Horne, B., Subburaj, V. H., and Urban, J. E., "Extensions to the Descartes Specification Language for the Development of Real-time Object Oriented Systems," Proceedings of the International Conference on Computing, Networking and Digital Technologies, 2012, pp. 225-236.

[23] Medina, M.A. and Urban, J.E., "An Approach to Deriving Reactive Agent Designs from Extensions to the Descartes Specification Language," Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems, 2007, pp. 363-367.

[24] Urban, J. E., Subburaj, V. H., and Ramamoorthy, L., "Extending the Descartes Specification Language Towards Process Modeling," Proceedings of the Federated Conference on Computer Science and Information Systems, 2011, pp. 337-340.

[25] Kanada, Y., “Taxonomy and description of policy combination methods,” Proceedings of a Conference on Policies in Distributed Systems and Networks, 2001, pp. 171-184

[26] Wei, Q. and Adams, C., "Exploring User-to-Role Delegation in Role-Based Access Control," Proceedings of the Eighth World Congress on Management of eBusiness, 2007, pp. 21-31.

[27] Bai, Q. and Zheng, Y., "Study on the Access Control Model," Proceedings of the Cross Strait Quad-Regional Conference on Radio Science and Wireless Technology, 2011, pp. 830 – 834.

[28] Arsi, S., Inukollu, V. N., & Urban, J. E. Issues and Challenges of Secure Policy Specification Languages.

[29] Sakhnini, N., Inukollu, V. N., & Urban, J. E. (2016, April). Automatic parallel programming using the descartes specification language. Proceedings of the 7th IEEE..International Conference on Information and Communication Systems ,pp. 298-303, 2016.

[30] Inukollu, V. N., Arsi, S., & Ravuri, S. R. (2014). Security issues associated with big data in cloud computing. International Journal of Network Security & Its Applications, 6(3), 45.