Volume 4 Issue IV, April 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

A model based testing method for Software Security Assurance

R.Nivesh[1], Dr.C.Chellappan[2]
Department of Computer Science and Engineering, GKMCET, Chennai, India-600063

Abstract: ***The importance of software security assurance is growing, but traditional development techniques have not kept pace with this need. New cost-effective tools for software quality and security assurance (SSA) are needed. This is consistent with the possible harm that could be result from the loss, incorrectness, alteration, unavailability, or misuse of the data and resources that uses, controls, and protects. This testing likes a penetration testing model to test the given software model. A penetration test can help verify whether a system is vulnerable to attack, if the defenses were sufficient, and which defenses the test defeated. Given a software model convert into Model-Implementation Description specification. The MID specification uses Petri net to capture both control and data-related requirements for functional testing, access control testing and penetration (pen test) testing with threat models. This model generates test code that can be executed quickly with the implementation under test, presents an automated test generation technique for integrated functional and security level testing of software systems. After generating test cases from the test model according to a given criterion, test code converts the test cases into executable test code by mapping model-level elements into implementation-level constructor. MISTA has implemented test generators for various test coverage criteria of test models, code generators for various scripting and programming languages, and test execution environments such as Java, C, C#, php, visual basic and HTML-Selenium IDE. MISTA has been applied to the functional and security testing of various real-world software systems. Security level testing based on the security assurance components are authentication, authorization, confidentially, availability, integrity and non-repudiation.***
Index Terms—Functional testing, model-based testing, Petri nets, security testing, software assurance.

I. INTRODUCTION

Model-based testing (MBT) is a promising approach to automated test generation by using models of a system under test (SUT)[11][12]. The modeling process of MBT helps clarify test requirements. Different from system modeling, test modeling focuses on formulating what needs to be tested, rather than capturing all system behaviors. MBT can be effective in fault detection because of the automated generation and execution of many tests. The goal of our work is to reap the benefits of MBT for integrated functional testing and security testing. Software security testing has two main objectives: testing whether or not the SUT has enforced the required access control policies, and testing whether or not the SUT is subject to potential security attacks. Access control policies are constraints on system functionality, whereas security attacks are often unexpected behaviors that violate security requirements such as confidentiality, integrity, and availability [8]. Therefore, model-based security testing requires modeling of access control policies and security threats. However, existing MBT tools have focused on functional testing. It uses Predicate-Transition (PrT) nets as an expressive formalism for building functional and security test models. PrT nets are high-level Petri nets, a well-studied formal method for modeling and verification of software systems [3][14].

Prior work has also demonstrated that PrT nets are capable of specifying access control policies and security threats. Because test models specified by PrT nets can capture both data and control flows of test requirements, MISTA can generate complete model-based test cases, including specific test inputs and test oracles. Note that model-based test cases are not yet executable with the SUT because test models are abstract descriptions of SUT's behaviors. MISTA provides an expressive way for describing the relations between the model-level elements and the implementation-level constructs in the target language or test environment so as to automatically transform the model-level tests into executable code [15]. The input to MISTA is called a Model-Implementation Description (MID) consists of a test model and a Model-Implementation Mapping (MIM). The test model represented by a PrT net can be describes a SUT's functional behavior, an access control model that specifies access control constraints on a SUT's functions, or a threat model that captures the potential security attacks against the SUT. The MIM specification maps the elements of the test model to the target implementation-level constructs. Given a MID, MISTA generates executable test code in a target language (such as Java, C#, C, C++ , HTML, and VB) according to a selected coverage criterion of the test model (such as reachability coverage,

state coverage, transition coverage, depth coverage, and goal coverage).

MISTA supports not only various programming and scripting languages, but also a number of test execution frameworks, such as JfcUnit, and WindowTest for GUI testing of Java programs; Robotium for testing Android applications; Selenium IDE, XML-RPC, and JSON-RPC for testing web applications; and Robot Framework for keyword-based testing. During the evolving development process, we applied MISTA to its own functional testing, which helped find many problems [11]. To evaluate the effectiveness of MISTA, we have applied it to the functional and security testing of several real-world software systems.

II. MODEL BASED TESTING

A model recitation a SUT is typically an conceptual, unfinished staging of the SUT's preferred performance. Test cases consequential starting such a copy are efficient test resting on the same level of pensiveness as the model. These test cases are reciprocally known as an conceptual check set. The executable analysis suite can communicate unswervingly with the structure under test [4]. This is achieved through mapping the intangible test cases to existing test cases appropriate for implementation. In various model-based testing situation, models hold adequate in sequence to engender executable test suites directly. In others, elements in the figurative test suite necessity survive map to explicit statement or method calls in the software to create a concrete test suite [13]. This is term solves the "mapping problem". Tests can be derived from models in different ways[7]. Because it is difficult to generalize the experimental and based on top of heuristics, nearby is no known single best approach for test derivation. It is frequent to headed for merge every test descent correlated parameter into a package that is often known as "test requirements", "test principle" or else smooth "employ holder(s)". This enclose can surround information about those parts of a model that should be spotlight on, or the state of affairs meant for concluding easy (test stopping criteria)[6][9].

III. MODEL CHECKING

Model checkers canister moreover is use designed for test case invention. Initially the model checking was urbanized as a modus operandi to test out if possessions of a condition are legitimate in a reproduction. Once used designed for testing, software model of the structure below scrutiny, and a property to test is provided to the model checker. Within the process of proofing, if this property is valid in the model, the model checker detects witnesses and counterexamples [13]. A spectator is a path, where the property is fulfilled, whereas a counterexample is a path in the execution of the model, where the property is despoiled. These paths can again be used as test cases and test tree.

A. Security Assurance

Software security assurance is a progression that facilitates design and rigging software that defend the information and assets enclosed in plus illicit through that software. Software is itself a supply and hence be obliged to be give suitable protection. While the amount of intimidation exclusively intention software is escalating, the security of the software to facilitate fabricate or get hold of must be confident [18]. Software Security Assurance (SSA) is the progression of guarantee that software is premeditated to activate at a echelon of protection that is reliable with the impending destruction that could cause from the trouncing, imprecision, amendment, unavailability, or maltreatment of the data and possessions that it utilize, reins, and shield. The software security assurance process instigate by categorize and tag the in sequence that is to be limited in, or used by, the software. The information should be card according to its consideration. For example, in the lowly group, the contact of a security desecration is nominal

B. MISTA

Model-based testing (MBT) is a promising approach to automated test generation by using models of a system under test (SUT). The modeling process of MBT helps clarify test requirements. Different from system modeling, test modeling focuses on formulating what needs to be tested, rather than capturing all system behaviors. MBT can be effective in fault detection because of the automated generation and execution of many tests. The goal of our work is to reap the benefits of MBT for integrated functional testing and security testing. Software security testing has two main objectives: testing whether or not the SUT has enforced the required access control policies, and testing whether or not the SUT is subject to potential security attacks (i.e., security threats). Access control policies are constraints on system functionality, whereas security attacks are often unexpected behaviors that violate security requirements such as confidentiality, integrity, and availability. Therefore, model-based security testing requires modeling of access control policies and security threats[4]. However, existing MBT tools have focused on functional testing. They cannot be applied directly to security testing because security test models are different from functional test models.

Volume 4 Issue IV, April 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

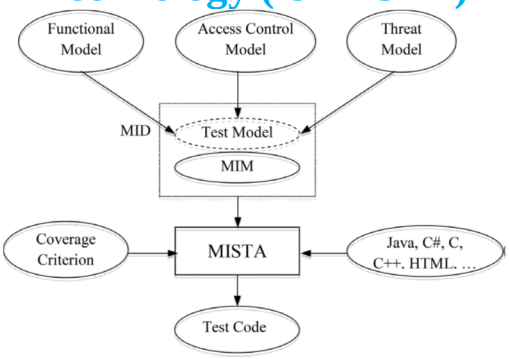
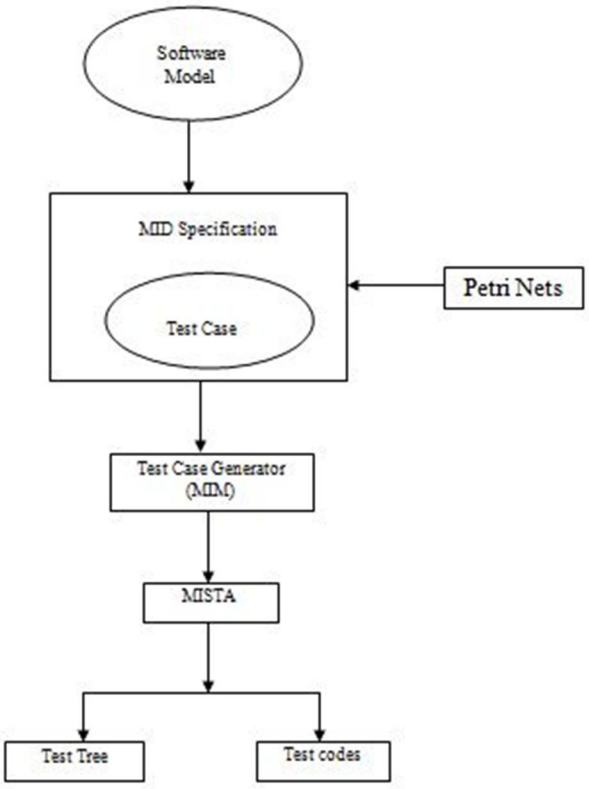


Fig 3.1 Context diagram of MISTA.

Fig. 3.1 shows the context diagram of MISTA. The input to MISTA is called a Model-Implementation Description (MID), which consists of a test model and a Model-Implementation Mapping (MIM). The test model represented by a PrT net can be a functional model that describes a SUT's functional behavior, an access control model that specifies access control constraints on a SUT's functions, or a threat model that captures the potential security attacks against the SUT. Transition pair coverage was originally proposed for FSMs, but is not meaningful for Petri nets. In addition, MISTA provides several techniques for dealing with the complexity of test models, including partitioning of test data, partial ordering of concurrent transition firings, and pair wise combination of inputs. Pair wise combination is also supported in Test Optimal. Studied the combination of random test generation methods with coverage criteria, and developed approaches to obtaining the approximation of a given coverage. It is possible to adopt this technique in MISTA for dealing with large test models. It's also test the security assurance of the software model. Fig 3.2 shows the implementation of security assurance test model.



e

Fig 3.2 Test code generation with MISTA

IV. MODULE DESCRIPTION

Create a software model as MID specification using Petri Net in MISTA. MISTA is called a Model-Implementation Description

(MID), which consists of a test model and a Model-Implementation Mapping (MIM).Given a Model-Implementation Description (MID) specification, MISTA test model generates test code that can be executed immediately with the implementation under test. MISTA uses a high-level Petri net to imprison both control and data-related requirements for functional testing, access control testing, or penetration(pen test) testing with threat models. After generating test cases from the test model according to a given criterion, MISTA test model converts the test cases into executable test code by mapping model-level elements into implementation-level constructs the test code and test trees.

A. Software model Creation

1) *MID specification:* MID(MODEL-IMPLEMENTATION DESCRIPTIONS), as the front-end input language for MISTA, lays the foundation for the automated test generation technique in our approach. A MID specification consists of a test model (PrT net) and a MIM description. The former does not use the implementation details of the SUT, whereas the latter relies on the test model as well as the SUT. First present PrT nets and MIM, and then describe the running example to be used throughout this paper. Ex. Bank account transaction, block Game, Cruise control, and self test.
2) *PrT Nets for Test Modeling:* The PrT nets in this paper, as in our previous work, are a lightweight version of the original PrT nets. Suppose constants start with an upper-case letter or a number, and variables start with a lower-case letter. A term is a constant, a variable, or a function (t_1,..t_n), and each t_i is a term. A term is called a ground term if it has no free variable. A label is a tuple of terms.
3) *Definition of PrT net*

A PrT net N is a tuple < P,T,F,I,L,φ,> where the elements are defined as follows.

P - a finite set of places (also called predicates).

T - a finite set of transitions.

F - a finite set of normal arcs from places to transitions and from transitions to places, ie. F ⊆ P x T ∪ T x P, .

I - a finite set of inhibitor arcs from places to transitions.

L - a labeling function on arcs F ∪ I. L(f) is the label for arc f ∈ F ∪ I When the label of an arc is not specified, the default label is a no-argument < >.

Φ - a guard function on T . The guard condition of transition t, φ(t), is a first-order logical formula, which can evaluate true or false.

M_0 - a set of one or more initial markings.

Suppose $M^k{}_0(p)$ is an initial marking, and is the set of tokens residing in place p. A token in is a tuple of ground terms $_{< X1 ,..... Xn >}$ We also denote it as $_{p(X1 ,..... Xn)}$. For a zero-argument token $_{< >}$ in p, we simply denote it as p. The token an initial marking represent test data or system settings (e.g., options and preferences) or both. In a shopping cart system, for example, token product (VGN-Z17) and token quantity represent the product VGN-Z17 and the quantity 3. A transition may be associated with a list of variables as formal parameters. These variables typically appear in the related arc labels. The guard condition of stack(x, y) is $x! = y$ An arrow (e.g., from holding to stack) represents a normal arc; a line segment with a small circle represents an inhibitor arc.

V. MODEL-IMPLEMENTATION MAPPING

A MIM specification relates all elements in a given PrT net, < P,T,F,I,L,φ,M_0 > , to the programming constructs in the target language or test environment L_1. It is used to convert model-level tests into executable test code.

A. MIM Specification

A MIM specification is a 7-tuple < I D,f_0, f_c, f_a, f_m, l_s,h > where the elements are as follows.

ID - the identity of the SUT tested against the test model.

$f_0 : O_M \rightarrow O_I$ - the object function that maps the objects in the test model to the objects in the SUT. Given an object x in the test model, $f_0(x)$ is an object in the SUT.

$f_c : T \rightarrow CODE_I$ - the component (or method) mapping function that maps transitions (component calls) in the PrT net to code blocks (test operations) in the SUT.

$f_a : P \rightarrow CODE_I$ - the accessor function that maps the places in the PrT net to code blocks (called accessor) in the SUT. An accessor is typically a sequence of assertions that read and check system states.

$f_m : T \rightarrow CODE_I$ - the mutator function that maps the places in the PrT net to code blocks (called mutators) in the SUT. A mutator is a piece of code that can change system states.

Volume 4 Issue IV, April 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

$l_s \subseteq P$ - a list of places in the PrT net that are implemented as system settings in the SUT. These places are called setting predicates.
h - the helper code function that defines user-provided code to be included in the test code

B. Shopping cart

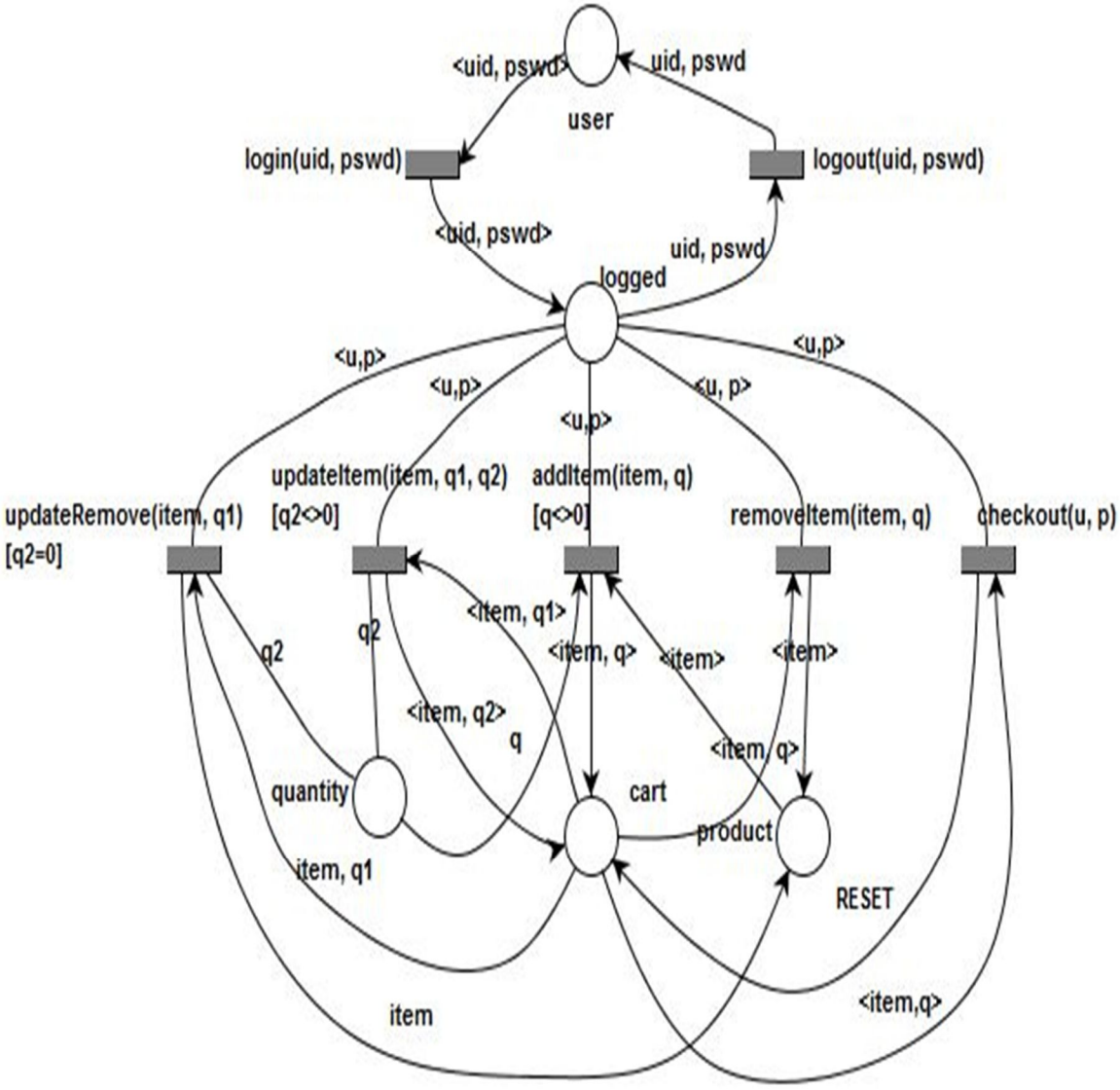


Fig. 5.2 PrT net for the shopping cart.

Fig 5.2 show a shopping cart perti net model. This test model is implementation-independent. The shopping cart could be implemented in different programming languages. Suppose the SUT is coded in C#. The four components are realized as methods in the Block class. To test the SUT against the above test model, we define the MIM specification according to the relations between the model and the SUT. The ID of the SUT is Block. There is no setting predicate in this example. Table 5.1 shows the component, and object functions. The object function relates model-level block identities to the implementation blocks and its defining the software security assurance each level of testing. The simulator tests the each places and transition of the given model. It mapped the different inputs for given model.

Volume 4 Issue IV, April 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Table 5.1 Levels of security Assurance

TEST LEVELS	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4	LEVEL 5
Authentication	<uid, pswd>	Verify user id	Password status	Login security level	Secured user
Authorization	Login(uid,pswd), logout(uid,pswd)	Logged	Second authorization	Identified authorized user	Allow access
Confidentiality	User data	Verifying user data and validate users account	Made transaction		
Availability	Checkin<u,p>	Checking available resource	Checkout(u,p)		
Integrity	Validated authentication	Checking amount	Transactions confirmation	Secure payment	
Non-repudiation	<item,q>, <uid,pswd>	Delivery status	Acknowledgement from user	Response to user request	

C. Testing The Model

Tests generated by each test generation strategy are organized in a transition tree. When a test framework such as JUnit and NUnit is used, several parts are not needed including the calls to the setup and teardown methods in each test method, the test suite methods, and the main method. When the target language is html for Selenium IDE, we use an html header, output each test sequence to an html file (as a Selenium test), include the setup and teardown code directly in each test sequence code and output the test suite code to an html file with a hyperlink to each individual test. After the test suite code is loaded into the Selenium IDE, all tests can be executed automatically.

VI. CONCLUSION

Complete model-level tests can be computed because the test model specifies both the control and the data dependencies of test targets. The mapping makes it feasible to transform the model-level tests into the executable form. Various case studies have demonstrated that MISTA is efficient and effective. The main contribution is integrated model-based testing of system functions, access control policies, and security threats. The technique can generate executable tests with respect to a variety of coverage criteria of test models, represented by PrT nets. It also supports a number of programming languages, and test execution frameworks. It used in Bank Account transaction, Cruise control and simulator. The MISTA generate the test code and test tree.

A. Future Work

Concerning future work, there are three directions we see. One direction is to compare fault-detection capability, time performance, and scalability of the test generators for different coverage criteria. Such a comparative study is expected to result in useful guidelines for cost-effective testing. The second direction is to use symbolic methods to produce an abstract reachability graph for test generation purposes. This graph will increase the ability to deal with more complex test models. The third direction is to introduce notations for modeling real-time systems so that time-critical test sequences can be generated automatically. A possible approach is to enhance PrT nets by associating transitions with time intervals as in Time Petri nets. The security assurance is included in the software model. Include the security assurance techniques model in this tool and test the software model. Future works have to implement the bank operation and cruise controller and simulator.

REFERENCES

[1] Hitesh Tahbildar, Bichitra Kalita, “Automated Software Test Data Generation: Direction Of Research”, International Journal, Vol.2, No.1, Feb 2011
[2] Mehmet Kara, “Review On Common Criteria As A Secure Software Development Model” International Journal, Vol 4, No 2, April 2012
[3] Mark B. Cooray, James H. Hamlyn-Harris, and Robert G. Merkel, “Dynamic Test Reconfiguration for Composite Web Services”, IEEE, VOL. 8, NO. 4, July/August 2015
[4] Dianxiang Xu, IEEE, Weifeng Xu, Senior Member, IEEE, Michael Kent, Lijo Thomas, and Linzhang Wang, “An Automated Test Generation Technique for

Software Quality Assurance", IEEE, VOL. 64, NO. 1, MARCH 2015
[5] Xiaoxing Yang, Ke Tang, Senior Member, IEEE, and Xin Yao, Fellow, "A Learning-to-Rank Approach to Software Defect Prediction", IEEE, VOL. 64, NO. 1, MARCH 2015
[6] Gregory Gay, Student Member, Matt Staats, Michael Whalen, Mats P.E. Heimdahl, "The Risks Of Coverage-Directed Test Case Generation", IEEE, 2015
[7] Erik Rogstad and Lionel C. Briand, Fellow, "Clustering Deviations for Black Box Regression Testing of Database Applications",IEEE,2015
[8] Michael Grottke, Dong Seong Kim, Rajesh Mansharamani, Manoj Nambiar, Roberto Natella, Kishor S. Trivedi, "Recovery From Software Failures Caused By Mandelbugs", IEEE 2015
[9] Anitha.A, "A Brief Overview Of Software Testing Techniques And Metrics", International Journal,Vol. 2, Issue 12, December 2013
[10] Abhijit A. Sawant, Pranit H. Bari and P. M. Chawan."Software Testing Techniques and Strategies", International Journal, Vol. 2, Issue 3, May-Jun 2012
[11] J. Zender, I. Schiefewrdecker, and P. Mosterman, Eds., Model-Based Testing for Embedded Syst". Boca Raton, FL, USA: CRC Press, 2011.
[12] M. Utting and B. Legeard, "Practical Model-Based Testing: A Tools Approach. San Francisco", CA, USA: Morgan Kaufmann, 2006.
[13] H. J. Genrich, "Predicate/transition nets," in Petri Nets: Central Models and Their Properties. New York, NY, USA: Springer, 1987,pp. 207–247.
[14] K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. New York, NY, USA: Springer-Verlag, 1992, vol. 26.
[15] T. Murata, "Petri nets: Properties, analysis and applications," Proc.IEEE, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[16] W. Reisig, "Petri nets and algebraic specifications," Theoret. Comput.Sci., vol. 80, pp. 1–34, 1991.
[17] D. Xu, "A tool for automated test code generation from high-level Petri nets," in Proc. 32nd Int. Conf. Applicat. and Theory of Petri Nets and Concurrency (Petri Nets 2011), LNCS 6709, Springer-Verlag, Berlin, Heidelberg, Germany, Newcastle, U.K., Jun. 2011, pp. 308–317.
[18] D. Xu, L. Thomas, M. Kent, T. Mouelhi, and Y. Le Traon, "A model-based approach to automated testing of access control policies," in Proc. 17th ACM Symp. Access Control Models and Technologies (SACMAT'12), Newark, NJ, USA, Jun. 2012.
[19] D. Xu, M. Tu, M. Sanford, L. Thomas, D. Woodraska, and W. Xu, "Automated security test generation with formal threat models," IEEETrans. Depend. Secure Comput., vol. 9, no. 4, pp. 525–539, Jul./Aug.2012.