

Formal Modeling and Analysis of XML Firewall for Service-Oriented Systems

Haiping Xu, Mihir M. Ayachit and Abhinay K. Reddyreddy

Computer and Information Science Department

University of Massachusetts Dartmouth

North Dartmouth, MA 02747

Email: {h xu, g_ mayachit, g_ areddyreddy}@umassd.edu

Abstract

As more businesses deploy web services over the Internet, the issue of how to secure them from intruders and possible threats becomes more important. Firewalls have been designed as a major component to protect a network or a server from being attacked. However, since conventional firewalls emphasize on packet filtering at the transport and session layer, rather than verifying user permissions and examining packet contents at the application layer, they are not suitable for protecting service providers from unauthorized web service invocations. In this paper, we propose a formal XML firewall security model using role-based access control (RBAC) mechanisms. Our proposed formal model supports user authentication and role-based user authorization according to policy rules stored in a policy database that can be updated dynamically. The formal model is designed compositionally using colored Petri nets (CPN), which can serve as a high-level design for XML firewall implementation. The major components of our compositional XML firewall security model are the application model and the XML firewall model. We analyze the application model and the XML firewall model separately using an existing Petri net tool, and demonstrate how key properties of our formal models can be verified, and how a design error can be detected and corrected at an early design stage.

Keywords XML firewall, web services, service-oriented systems, role-based access control (RBAC), colored Petri net (CPN), formal verification.

1. Introduction

Web services provide a standardized way that support interoperable machine to machine interaction over the Internet (Booth et al., 2004). Web services are XML based software components that can be dynamically incorporated into different applications using remote method invocation mechanisms, such as JAX-RPC (Java API for XML-based RPC) (Nagappan et al., 2003) and WSIF (Web Service Invocation Framework)

(Juric, 2006). A web service is designed as a loosely coupled software component that can be described using the WSDL (Web Services Description Language), registered using the UDDI (Universal Description, Discovery and Integration), and invoked using standard protocols, such as the SOAP (Simple Object Access Protocol) that is bound to underlying protocols, e.g., HTTP.

As more businesses deploy web services over the Internet that dynamically interact with various applications and data sources, the issue of how to secure them from intruders and possible threats becomes more important (Mysore, 2003). Security problems in web services are severe because the Internet is a public network infrastructure, where the information available to be accessed over the Internet has different levels of business confidentiality. Furthermore, a service consumer may invoke web services using false identity, access web services with insufficient permissions, or corrupt web services by attacking the service providers (e.g., using an XML message-based denial of service attack). Thus, security consideration becomes very critical for the successful deployment of service-oriented systems.

A conventional firewall typically resides at the perimeter of a network server or a business's private network, and monitors the data traffic entering and exiting the network to prevent unauthorized access to the server or the network. Typical types of conventional firewalls include package filtering firewalls, application-level gateways, and stateful inspection firewalls (Pfleeger and Pfleeger, 2003; Fernandez et al., 2005). However, a conventional firewall may provide no security at all for web services. This is because most of the web services are SOAP based or simply XML based, which is bound to HTTP; thus, XML messages can most likely pass through port 80, the default web port, which is normally not blocked by a conventional firewall (Windley, 2003). Furthermore, a potential intruder can include malicious SOAP attachments, insert harmful SQL code or executable commands into an XML packet, or send an extremely large XML packet to overload the XML parser on the service provider side (Moradian and Håkansson, 2006; Vorobiev and Han, 2006). A conventional firewall usually does not examine the content of a packet; thus, it is not able to identify threats such as SQL injection, denial of service, schema poisoning, and XML parameter poisoning (Gralla, 2007; Vorobiev and Han, 2006). For example, a packet with XML data tampered with an SQL injection attack that can erase a whole database cannot be detected using packet filtering techniques; instead, it can only be detected by content filtering approaches. Hence, conventional firewalls are not sufficient to provide security for web services. In addition, conventional firewalls usually exist at the transport and session layer, rather than the application layer and within the data packet or content (Wrenn, 2004); therefore, security holes can be left to allow an unauthorized person to attack a service provider by accessing web services without needed permissions.

To protect web services from being attacked, we develop a compositional formal model, called XML firewall security model, which enforces access restrictions for web service invocations. Our security model is derived from a general XML firewall model presented in (Ayachit and Xu, 2006). In our proposed model,

the access to web services is only granted to those users, who are authenticated and authorized to have access to the services. The model is formally defined using the Petri net formalism, which is a mature formalism with existing theory and tool support (Murata, 1989). There are two key components in the XML firewall security model, namely, the application model and the XML firewall model. In the XML firewall model, we adopt the role-based access control (RBAC) mechanism (Feinstein et al., 1996) in order to effectively deploy user authorization and access rights. The role-based access control mechanism we use in our model is stateful. In other words, role assignment and permission granting in XML firewall depend not only on a user's identity, but also on the current state of the system.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 presents an architectural design of XML firewall protected service-oriented systems. Section 4 introduces the compositional Petri net based XML firewall security model, including the application model and the XML firewall model. Section 5 performs some formal analysis of the Petri net models using an existing Petri net tool. Section 6 gives the conclusions and future work.

2. Related Work

A closely related work to our proposed XML firewall approach is the role-based access control mechanism. The role-based access control (RBAC) model has been used as one of the most attractive solutions to providing security features in different distributed computing infrastructure (Feinstein et al., 1996). In an RBAC model, users are assigned roles with permissions, which ensure that only authorized users are given access to certain data or resources. A principle motivation behind RBAC is the ability to specify and enforce enterprise specific security policies such that it can map naturally to an organization's structure. Since in a typical organization, user and role associations change more frequently than role and permission associations, RBAC results in reduced administrative costs as compared to associating users directly with permissions. In an RBAC model, a user is a human being or a process within a system; while a role defines a collection of permissions associated with a certain job function within an organization. A permission of a role is an access mode that can be exercised on a particular object or a resource in the system. A user can be related to possibly many roles using sessions, which specify the durations of valid role assignments. Most of the RBAC models follow the same basic structure of subject, role and privilege. However, in a more sophisticated role-based access control model, access decisions for an application will depend on the combination of the required credentials of users and the context and state of the system, as well as other factors such as relationship, time and location (Zhang and Parashar, 2004). Giuri and Iglio proposed a role-based access control model that provided special mechanisms for the definition of content-based access control policies (Giuri and Iglio, 1997). By extending the notion of permission, they allowed the specification of security policies, in which the permission of an object may depend on the content of the object itself. Although much work has been done in the area of access control, most of the work is user-

centric, where only credentials of the user are considered when granting access permission. Very little work has been done to combine context information with credentials while making access control decisions. In our XML firewall model, we combine the traditional RBAC with the state information to determine access control; thus, our approach can be more flexible and effective in dynamic permission assignments.

Previous work on how to protect web service providers from being attacked is rare. Fernandez and his colleagues proposed to protect web services from unauthorized access by developing a pattern-based language for XML firewall (Fernandez, 2004; Fernandez et al., 2005). They designed two patterns for XML firewall, namely the security assertion coordination pattern using role based access control (RBAC) for access to distributed resources, and a filter pattern for filtering XML messages or documents according to institution policies. Although their approach provides useful insights about implementation of XML firewalls, the XML firewall model they proposed is not formally defined. Cremonini and his colleagues proposed an XML-based approach to combining firewalls and web services security specification (Cremonini et al., 2003). They discussed about the security requirements of web service architecture (WSA), and presented some possible design guidelines for semantics-aware firewalls that can be fully integrated within the WSA. However, technical details about implementation of their approach are still missing. More recently, Moradian and Håkansson summarized possible attacks on XML web services, including SQL injection, IP spoofing, and denial of service attacks (Moradian and Håkansson, 2006). But no solutions are proposed to protect the service providers from service-based attacks. Different from the above approaches, we propose a stateful XML firewall security model that supports dynamic role assignment and permission granting. Furthermore, since an XML firewall represents one of the critical components in a business application, to ensure a correct design, we develop a formal model using colored Petri nets, and demonstrate how existing Petri net tools can be used to verify the key properties of our net model.

Some XML firewall related products are currently available in the market for securing web services applications. For example, the Forum Systems Company developed an XML security appliance, called XWall, which resides in front of servers that contain sensitive XML tagged information (Allen, 2006). The appliance encrypts XML fields in real time, as the data goes into the server. It then decrypts it when the data exits the server. The appliance is unique as it examines data on a tag-by-tag basis, and therefore does not encrypt the unnecessary or non-critical fields. Another implementation of the XML firewall is the DataPowerXS40 XML Security Gateway (DataPower, 2006). This firewall requires the creation of a virtual firewall for every service exposed to the outside world, which then forms a path through the firewall to the back-end server supplying the web services. Each virtual firewall is configured with a custom firewall policy of actions on each XML message passing through the firewall. Policy actions are implemented through XSL style sheets and may include XML filtering, digital signatures, signature verification, schema validation, encryption, decryption, transformation and routing. XML firewall vendors, as a whole, are a mix of startup companies and older security companies looking to enter the market.

Although the above implementations with XML firewall features can help to protect web services, their functionalities are still very limited. For example, they do not support verification of user authorization, and thus, unauthorized user may access web services with insufficient permissions. In addition, existing XML firewall approaches are usually not state-based, so they cannot protect web services from certain threats such as a denial of service attack. In contrast, we propose a general solution to implementing XML firewalls that supports state-based user authentication and authorization. More importantly, our XML firewall model is formally defined using the Petri net formalism, so it supports formal verification for ensuring a correct design (e.g., deadlock-freeness), as done in our previous work (Xu and Shatz, 2003a; Xu et al., 2005). Some additional related work along this direction includes Xu and Nygard’s work, where a threat-driven model is developed using aspect-oriented Petri nets (Xu and Nygard, 2005; Xu and Nygard, 2006). Their approach supports incremental modeling of security features to improve trustworthiness of software design. Different from the above threat-oriented approach, we take a property-oriented approach to security where security features are explicitly defined in our model. Furthermore, our proposed formal model can serve as a high-level design for XML firewall implementation, and may provide a potential solution to automated software development as illustrated in (Xu and Shatz, 2003b).

3. Architectural Design

An XML firewall protected service-oriented system consists of three major types of components, namely the applications, XML firewalls, and web services. The system architecture of a service-oriented system with a single XML firewall installed is illustrated in Figure 1. As shown in the figure, a service provider may deploy a group of web services on a web server, which is protected by an XML firewall. The web services can be invoked by various applications at runtime, so the web services shall be able to interact with different applications concurrently and dynamically. Meanwhile, an application is allowed to make multiple requests to web services protected by the same XML firewall at the same time. Therefore, the XML firewall must support processing of various web service invocation requests concurrently.

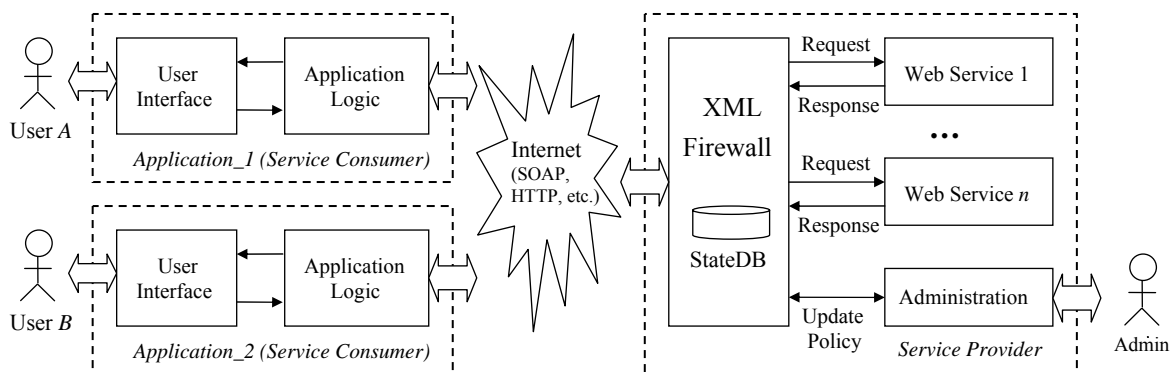


Figure 1. XML firewall protected service-oriented system

In Figure 1, we illustrate two applications that may interact with the same group of web services concurrently. It is worth to be noted that an application can also interact with different groups of web services, which are deployed by different service providers protected by their own XML firewalls (this scenario is not shown in Figure 1). On the application side, a user interacts with an application through its user interface. The application logic is the business logic of an application, which varies from application to application. The application logic processes the requests from the user, and initiates service calls that may invoke a single web service or a group of web services at the same time. The request from the application is checked by the XML firewall for authenticity and access limitations depending on state information stored in the *StateDB* database. If the request is valid, the XML firewall will pass the request to the corresponding web service; otherwise, the request is rejected. The administrator of an XML firewall can change the policies stored in the policy database through an administration module at runtime. Activities of changing policies include adding a new policy, modifying an existing policy, and deleting a policy that is no longer needed. Each web service has its own logic to process the corresponding method request, and returns the result to the XML firewall. Upon receiving the results from the web services, the XML firewall then passes the results to the application. When the application receives the results from the XML firewall, the application logic processes these results for further computation, and will send appropriate messages to the user through its user interface.

The refinement of the XML firewall module in a service-oriented system is illustrated in Figure 2, which describes the important components inside an XML firewall module.

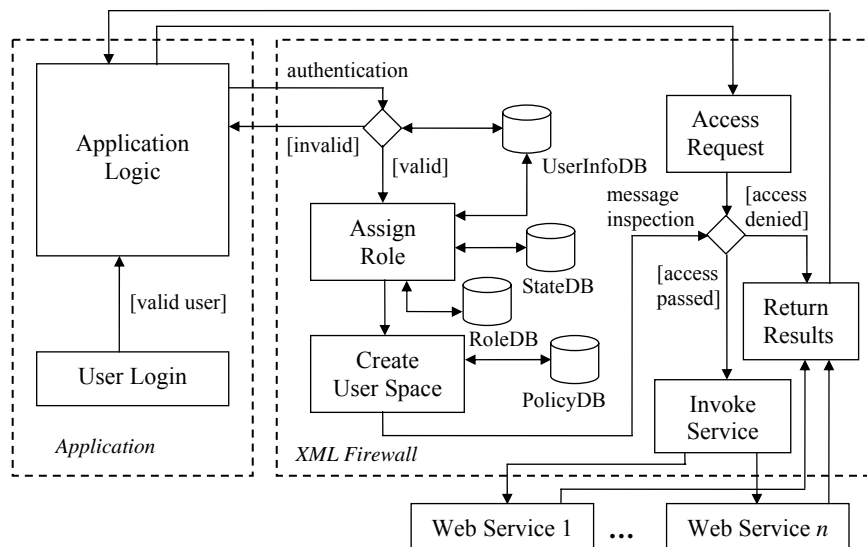


Figure 2. Refinement of the XML firewall module in Figure 1

As shown in Figure 2, to start an application, a user first needs to log into the application. If the user is a valid one, the application logic will process the user's access requests, and based on the user's requests, the

application logic initiates the needed service calls. The service call with the user's information is intercepted by the XML firewall for authentication and authorization. The user is authenticated by checking against certified user information stored in a database, called *UserInfoDB*, as shown in Figure 2. If the user's identification is valid, he is assigned a role defined in the *Role* database (i.e., *RoleDB*); otherwise, an *access denied* message is sent to the application. The role assignment is based on the system state including the user's current state, which is determined by the status of the incoming message as well as the information stored in the *StateDB* database. After the role assignment is done, a *user space*, which contains a session and access permissions of the user, is created based on policies from the *PolicyDB* database. The *user space* is then compared with the service request to determine whether the incoming request from the user has permissions to invoke a web service; meanwhile, the incoming message is inspected for any malicious contents within the user space. If the user has the needed permissions, and the XML-based message does not contain any malicious contents, the web service request will be dispatched to the corresponding web service by the XML firewall; otherwise, an *access denied* message will be sent to the application. If the web service request is a valid one, the web service will process the request, and return the result to the XML firewall, which is then passed back to the application.

4. CPN-Based Compositional XML Firewall Security Model

Petri nets are a well-founded process modeling technique that has formal semantics to allow specification, design, verification, and simulation of a system (Murata, 1989). Petri nets have been widely used to model and analyze various types of processes and systems including security protocols (Bouroulet et al., 2004), web services (Hamadi and Benatallah, 2003; Liu and Chen, 2005), manufacturing systems (Toumodge, 1995; Jalilvand and Khanmohammadi, 2004), and business processes (Aalst, 2002). A Petri net is a directed, connected, and bipartite graph, in which each node is either a place or a transition. In a Petri net model, tokens are used to specify information or conditions in the places. When there is at least one token in every input place of a transition, the transition is enabled. An enabled transition can be fired by removing one token from every input place, and depositing one token in each output place of the transition. Colored Petri nets (CPN or CP-net) are an extension of ordinary Petri nets, which allow different values (represented by different colors) for the tokens (Jensen, 1992; Jensen and Rozenberg, 1991). Colored Petri nets have a formal syntax and semantics that leads to compact models of rather complex systems for modular design and analysis (Christensen and Petrucci, 1992; Jensen, 1998). In addition, a CPN allows associating guards and executable code written in a high-level programming language – the ML language (Clack et al., 1993) – with a transition. The modeling and analysis of CPN models are supported by powerful Petri net tools, such as the CPN Tools (Ratzer et al., 2003).

Petri nets are a graphical and mathematical modeling tool applicable to many systems. In this section, we develop a compositional XML firewall security model for web services invocation using CPN. As

mentioned previously, we design our XML firewall protected service oriented system modularly with the basic components, i.e., the application module and the XML firewall module, where the interfaces between these modules are well defined. In our CPN models, we also introduce a few types of tokens that denote the different types of inputs and outputs of transitions. For example, if a transition results in a Boolean decision, a BOOL token will be placed at the output place of the transition. In addition, we associate guards with some transitions to model the decision making processes.

4.1 Application Model

An application invokes web services according to its application logic, which may involve concurrency. Figure 3 shows a CPN model for an application that invokes two web services concurrently. We assume the web services are deployed on different web hosts, so they must be protected by different XML firewalls. The two web services are represented by two *abstract* transitions *WS_Logic1* and *WS_Logic2* (denoted by boxes with thicker border line in Figure 3). An abstract transition is a high-level transition that represents an activity, which can be refined in a more detailed design. The refinement of an abstract transition into a new Petri net is beyond the scope of this paper, but it can be modeled as a *substitution* transition that stands for a CPN module in a hierarchical net structure supported by the CPN Tools (Ratzer et al., 2003; Jensen et al., 2006). In Figure 3, the XML firewall module is abstracted into a subnet with a few places and transitions (enclosed in a dashed line box in Figure 3), which will be refined into a more detailed design in Section 4.2.

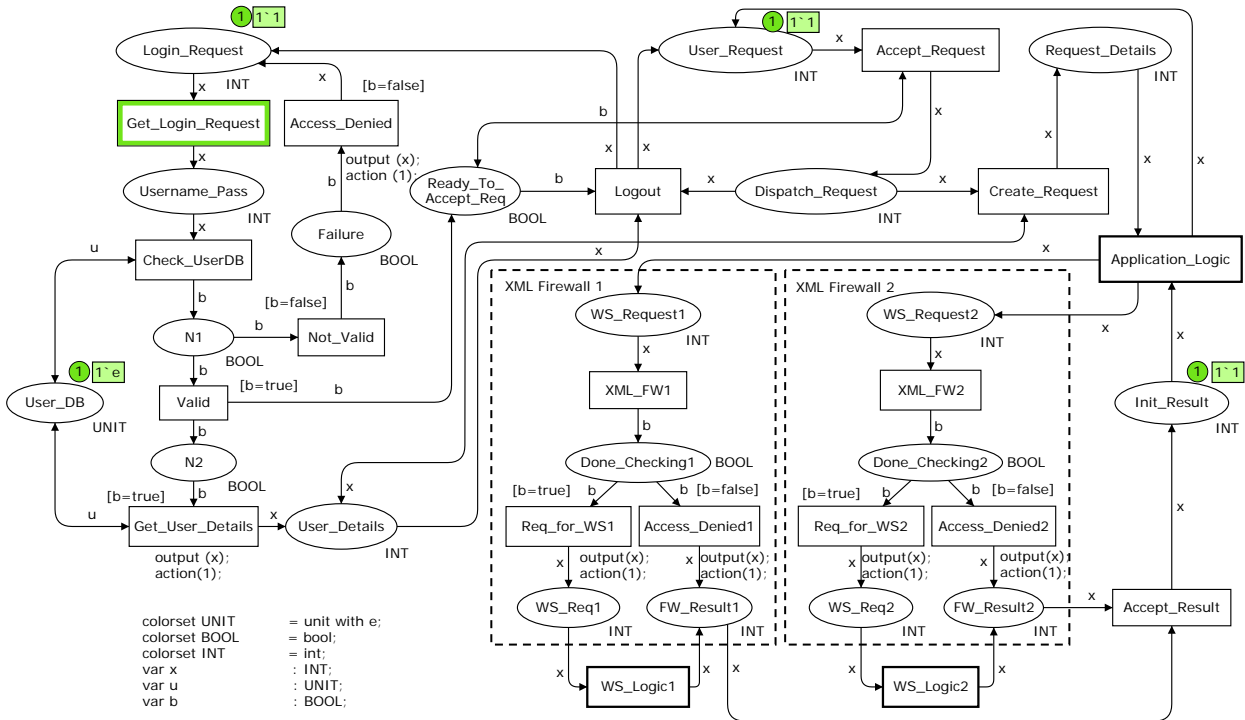


Figure 3. CPN model of an application that invokes two web services concurrently

An XML firewall can be used to protect one or a group of web services deployed on a web server (only one web service is shown in Figure 3 behind each XML firewall). Web services are invoked by various applications according to users' access requests. To protect both the application and the web services, a user is required to provide his credential (user name and password) when he logs into the application. This is represented by a token (denoted as $1 \cdot 1$ in Figure 3, i.e., one token with value 1) placed in the *Login_Request* place. The token is passed to the *Username_Pass* place when the transition *Get_Login_Request* fires. The checking of the username and password is done by firing the transition *Check_UserDB*, which verifies a user's identity with the information of certified users stored in a database called *User_DB*. Note that the information stored in the database *User_DB* is represented by a unit token denoted as $1 \cdot e$ in Figure 3. A failure result from the authentication process indicates that the user is not a valid one, so a Boolean token "false" will be deposited into place *N1*, which enables the transition *Not_Valid*. Note that the guard $[b=false]$ associated with the transition *Not_Valid* evaluates to *true* when a "false" token is present in place *N1*. The firings of the transitions *Not_Valid* and *Access_Denied* sequentially will inform the user that the access to the application was denied, and a token will be returned back to the *Login_Request* place. On the other hand, if the user is verified as a valid one after firing the transition *Check_UserDB*, a Boolean token "true" will be deposited into place *N1*, which enables the transition *Valid*. The firing of transition *Valid* deposits a token in both of the places *N2* and *Ready_To_Accept_Req*. A token in place *N2* enables the transition *Get_User_Details* that can fetch a user's detailed information from the *User_DB* database, and deposit a token into place *User_Details*. Meanwhile, a token in place *Ready_To_Accept_Req* enables both of the transitions *Accept_Request* and *Logout* to allow an access request to web services and a logout request, respectively. Note that although there is an initial token in place *User_Request* that represents a request from the user, the transition *Accept_Request* cannot fire until a token is present in place *Ready_To_Accept_Req*, which indicates that the user's authentication check has been passed, and thus, any requests from the user can now be processed. As a result of firing the *Accept_Request* transition, a token is deposited into the *Dispatch_Request* place for further processing. If the user request is a logout request, then the *Logout* transition will fire. If the *Logout* transition fires, the tokens in place *Ready_To_Accept_Req*, *User_Details*, and *Dispatch_Request* are removed, and a new token is returned back to the initial place *Login_Request* and the place *User_Request*. Since there is no token in the *Ready_To_Accept_Req* place now, a user must login again before he can make any further requests.

If the request made by the user is an access request to web services, the *Create_Request* transition can fire, and a token will be deposited into the *Request_Details* place. A token in the *Request_Details* place contains the information retrieved from the *User_Details* place combined with the information from the incoming user request. This enables the *Application_Logic* transition representing the business logic of the application. Note that the *Application_Logic* transition is defined as an abstract transition that can be refined into a detailed design according to the actual functionalities of the application. When the transition *Application_Logic* fires, the application applies its business logic to the incoming request, and generates

requests for web services invocation. To illustrate concurrent invocations of two web services, the CPN model contains two web services that are protected by two different XML firewalls. To simplify matters, we assume that the user has to wait for both of the results returned from the web service invocations before any further requests can be processed. The goal of the XML firewall is to perform the authentication and authorization activities for incoming user requests from an application. If the user is authorized and has the needed permissions to access a web service, then the web service is invoked. This logic is shown in Figure 3 using the *XML_FW1* and *XML_FW2* transition for *XML Firewall 1* and *XML Firewall 2*, respectively. If the user request is authentic, and the user has all the necessary permissions to invoke a web service protected by an XML firewall, a “true” token will be deposited into its *Done_Checking* place (*Done_Checking1* or *Done_Checking2*), which enables the corresponding *Req_for_WS* transition (representing the action of request for web services). If the transition *Req_for_WS* fires, a token representing this request will be deposited into place *WS_Req* (Web Service Request), and enables the corresponding *WS_Logic* transition that is defined as an abstract transition for the web service logic. After processing the request by a web service, a token representing the result will be placed in the corresponding *FW_Result* place. On the other hand, if the web service access is denied, the corresponding *Access_Denied* transition fires, and a token representing an *access denied* message is placed in the *FW_Result* place.

When there is a token in both of the *FW_Result1* and *FW_Result2* place, the *Accept_Result* transition in the application module can fire. Once the result is accepted, a token is deposited into the *Init_Result* place, which implies the availability of the return results from the web services. This enables the *Application_Logic* transition, and the return results can now be used by the *Application_Logic* transition for further processing. When the *Application_Logic* transition fires, needed computations are performed, and a token is returned to the *User_Request* place, which enables a new user access request.

4.2 XML Firewall Model

In Figure 3, the XML firewalls are designed as compositional modules (displayed inside the dashed line boxes) that have well-defined interfaces with both of applications and web services. The XML firewall module in Figure 3 can now be refined into a more detailed design as shown in Figure 4. To make the CPN model of an XML firewall self-contained, we have shown an abstraction of the application module with two places (i.e., *User_Request* and *Init_Result_1*) and two transitions (i.e., *Application_Logic* and *Accept_Result*) in Figure 4. In addition, we also include an abstract web service module that is represented by the abstract transition *WS_Logic*. Note that different from Figure 3, we only show one XML firewall in Figure 4; however, due to the compositional modular design of our net model, it is straightforward to extend the CPN model in Figure 4 into a system that includes two XML firewalls as shown in Figure 3.

As we discussed earlier, the application logic in an application handles all the incoming requests coming

from the user and invokes the corresponding web services. In Figure 4, when the *Application_Logic* generates a web service request, a token is placed into the *WS_Request* place indicating a web service invocation. The *Check_If_Existing* transition is enabled, and can fire to check if the user, who makes the request, is an existing user or a new one. If the user's identity is not found in the database *UserInfo_DB*, then the user is recognized as a first time user, and a "false" token is deposited into place *N1*, which enables the transition *First_Time_User*. For each first time user, the *PerformBG_Check* transition is fired, and a background check is performed according to users' background information stored in database *BG_DB*. A user becomes a valid member if the background check is passed, and a token is deposited into place *Valid_User*. Then the *Update_DBs* transition must fire to update the user information database *UserInfo_DB* as well as the role information database *Role_DB*. Meanwhile, a token is deposited into place *Valid_User_Req* indicating the current request is from a valid user. On the other hand, if the user authentication fails, the *Check_Failed* transition is fired, and a token indicating *access denied* is deposited into the *FW_Result_1* place.

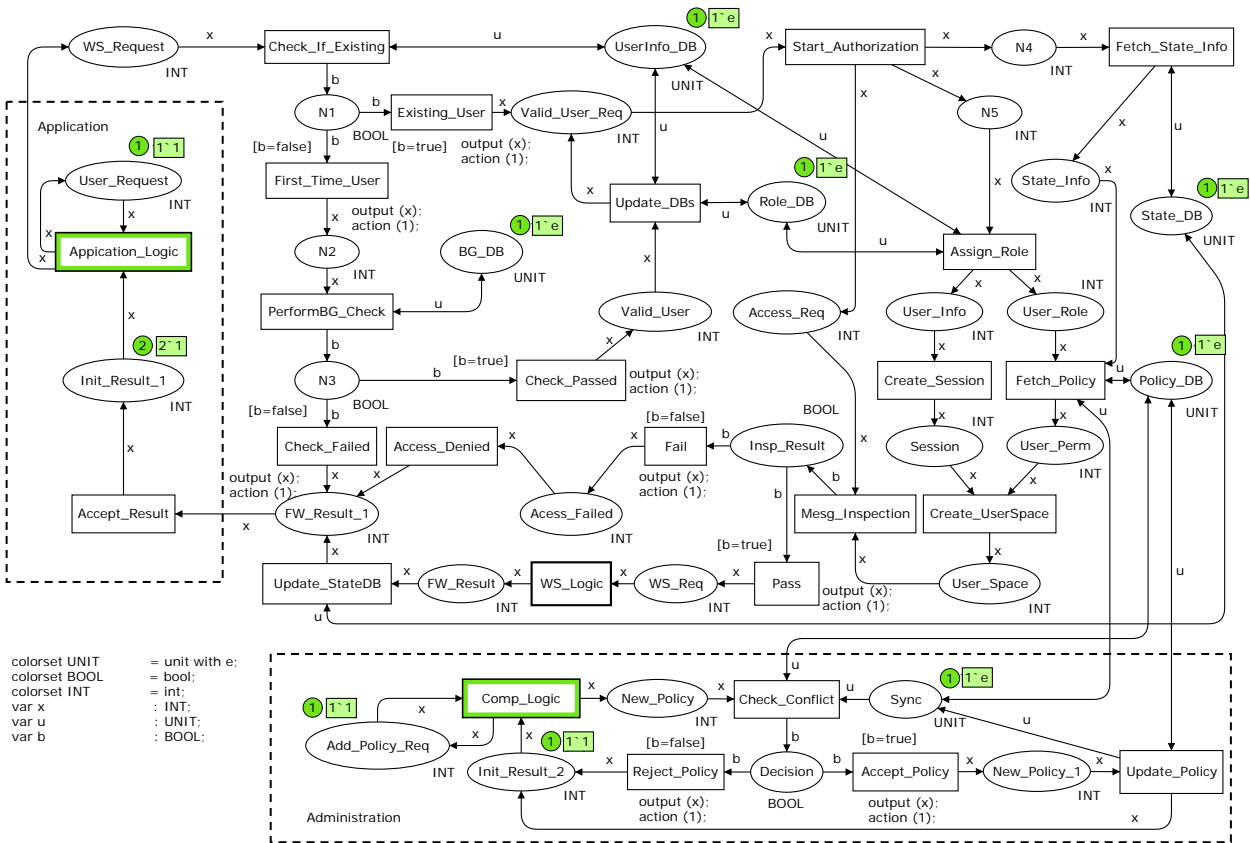


Figure 4. CPN model of an XML firewall with one application and one web service

A user is identified as a regular user if his user profile exists in the *UserInfo_DB* database. For a regular user, the *Existing_User* transition is fired, and a token is deposited into the *Valid_User_Req* place. Once a token is present in the *Valid_User_Req* place, the authorization process can start by firing the

Start_Authorization transition. The state information for the incoming request is generated by firing the *Fetch_State_Info* transition, which uses state information that is already stored in the database *State_DB*, as well as information extracted from the incoming request message (e.g., the time of the request). After the state information is generated, a token indicating the current state of the request is placed into the *State_Info* place. The *Assign_Role* transition is now enabled and can fire to assign roles to the user according to information stored in the databases *UserInfo_DB* and *Role_DB*. In addition, a user session is created by firing the *Create_Session* transition. The user session defines the period of time during which, a user can interact with an application when invoking a web service. If the session expires during an invocation (the session information will be passed along with a user space token to the *WS_Logic* transition as described later), the *WS_Logic* transition returns a *timeout* result back to the XML firewall, so a new web service invocation request needs to be placed. The next task is to fetch a policy from the *Policy_DB*. The *Fetch_Policy* transition can fire when there is a token in the *User_Role* place, the *State_Info* place, and the *Sync* place. A policy is fetched from the *Policy_DB* based on the user's role and user's current state. After a policy is fetched and a session is created, a *user space* is created that contains the user information, permissions and the session information. A token representing a user space will be deposited into the *UserSpace* place. Note that ideally, both the session token and the user space token should be defined as colored tokens that contain the needed information; however, to simplify our CPN model, we use tokens of type INT to represent both sessions and user spaces.

A token in the *Access_Req* place represents a web service invocation request in XML format. The *Mesg_Inspection* transition can fire in order to check the following two aspects: (1) the entire XML message is scanned to discover whether the message contains any malicious contents; (2) the web service invocation request is verified if it can be granted within the user space created according to the user's role and permissions. A Boolean token representing the result will be deposited into the place *Insp_Result*. If the message does not contain any malicious contents, and the user has the needed permissions to invoke the web service, the *Pass* transition can fire, and a web service request will be dispatched to the corresponding web service. After the web service request is processed (i.e., the firing of the *WS_Logic* transition), a token representing the result of the web service invocation is deposited into the *FW_Result* place. This token enables the *Update_StateDB* transition, which updates the state information in the database *State_DB*, and also deposits a token in place *FW_Result_1*. On the other hand, if the XML message contains any malicious contents, or the user does not have sufficient permissions to invoke a web service, the *Fail* transition fires, and a token is placed into the *Access_Failed* place. When the transition *Access_Denied* fires, a token that indicates the web service access is denied is deposited into the *FW_Result_1* place. From the above description, we can see that the *FW_Result_1* place may hold two types of tokens: one representing an *access denied* message, and another one representing the result from web service invocation. With a token in the *FW_Result_1* place, the transition *Accept_Result* defined in the simplified application module can fire. As a result, a token will be deposited into the *Init_Result_1* place, and the *Application_Logic* transition

determines the next step of actions. When the *Application_Logic* transition fires, a token will be returned to the place *User_Request*, and the CPN model for the XML firewall may return to its initial state. Note that in the *Init_Result_1* place, initially there are two tokens denoted by $2 \cdot 1$. This allows a user to make two concurrent requests to web services protected by the same XML firewall, and it requires the XML firewall have the capability of processing more than one web service request at the same time.

At the bottom of Figure 4, we introduce an *Administration* subnet that models the administration process of adding new policies into the database *policyDB*. The abstract transition *Comp_Logic* in Figure 4 represents the computation logic to capture a user's request for adding a new policy into *policyDB*. When the transition *Comp_Logic* fires, a token representing a new policy is deposited into place *New_Policy*. Then the transition *Check_Conflict* must fire to ensure the new policy is consistent with existing policies stored in the *policyDB*. If there is no conflict between the new policy and the existing policies, the new policy will be accepted by firing the transition *Accept_Policy*, and the *PolicyDB* is updated when the transition *Update_Policy* fires. Otherwise, the *Reject_Policy* transition fires, and the *PolicyDB* shall remain unchanged. Notice that we have introduced a synchronization place *Sync* that initially contains a unit token to synchronize the processes of fetching a policy and updating the *policyDB*. When the *Check_Conflict* transition fires, the unit token in place *Sync* is removed, so the transition *Fetch_Policy* cannot fire even if there are tokens in the place *User_Role* and *State_Info*. The *Fetch_Policy* transition can become enabled again once the unit token returns to the *Sync* place when the *PolicyDB* has been properly updated (i.e., when the transition *Update_Policy* fires). Due to the modular design of our CPN models, our CPN models can be easily extended to support modifying or deleting an existing policy from the *PolicyDB*.

5. Analysis of Application Model and XML Firewall Model

One of the advantages of using CPN to model XML firewall protected service-oriented systems is its support for formal analysis using existing Petri net analysis tools. In this section, we show how to use the CPN Tools (Ratzer et al., 2003) to analyze some key properties of our CPN models.

The CPN Tools is a program that supports editing, simulating, and analyzing colored Petri Nets (Jensen et al., 2006). In CPN Tools, a fast simulator is available for handling both timed and untimed Petri nets efficiently. The CPN Tools include a state space analysis engine that can generate a full or partial state space, and produce a standard state space report containing information such as boundedness, liveness, and deadlock-freeness properties. The functionality of the simulation engine and the state space facilities are developed based on a previous version of the tool, called Design/CPN (Albert et al., 1989), which is a widespread tool for colored Petri Nets. To verify the correctness of our XML firewall security models, we utilize some key definitions for Petri net behavior properties as adapted from (Murata, 1989).

Definition 5.1 Reachability: In a Petri net N with initial marking M_0 , denoted as (N, M_0) , a marking M_n is said to be *reachable* from a marking M_0 if there exists a sequence of firings that transforms M_0 to M_n . A firing or occurrence sequence is denoted by $\sigma = M_0 \ t_1 \ M_1 \ t_2 \ M_2 \ \dots \ t_n \ M_n$ or simply $\sigma = t_1 \ t_2 \ \dots \ t_n$. In this case, M_n is reachable from M_0 by σ , and we write $M_0 [\sigma > M_n$.

Definition 5.2 Boundedness: A Petri net (N, M_0) , is said to be *k-bounded* or simply *bounded* if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0 . A Petri net (N, M_0) is said to be *safe* if it is 1-bounded.

Definition 5.3 Liveness: A Petri net (N, M_0) , is said to be *live* if for any marking M that is reachable from M_0 , it is possible to ultimately fire any transition of the net by progressing some further firing sequence.

Definition 5.4 Reversibility: A Petri net (N, M_0) is said to be *reversible* if, for each marking M that is reachable from the initial marking M_0 , M_0 is reachable from M .

Definition 5.5 Home Marking: A marking M_{home} of a Petri net (N, M_0) is said to be a *home marking* if M_{home} can be reached from any reachable marking M_n .

Definition 5.6 Dead Marking: A marking M_{dead} of a Petri net (N, M_0) is said to be a *dead marking* if, in marking M_{dead} , no transition is enabled in the net.

We first input our application net model defined in Figure 3 into the CPN Tools. The state space analysis tool produces the results as listed in Table 1.

Table 1: Analysis results of the CPN application model in Figure 3

Statistics	Boundedness Properties		

State Space	Best Integer Bounds	Upper	Lower
Nodes: 260	Dispatch_Request	1	0
Arcs: 823	Done_Checking1	1	0
Secs: 0	Done_Checking2	1	0
Status: Full	FW_Result1	1	0
	FW_Result2	1	0
Home Properties	Failure	1	0
-----	Init_Result	1	0
Home Markings	Login_Request	1	0
All	Ready_To_Accept_Req	1	0
	Request_Details	1	0
Liveness Properties	User_DB	1	1
-----	User_Details	1	0
Dead Markings	User_Request	1	0
None	Username_Pass	1	0
	WS_Req1	1	0
Dead Transition Instances	WS_Req2	1	0
None	WS_Request1	1	0
	WS_Request2	1	0
Live Transition Instances			
All			

The analysis results in Table 1 show that the full state space has been calculated, and the net has an upper bound of 1 (due to space limitation, we only list the boundedness properties of some key places of the application model in the right column of Table 1). This implies that any place in the application net model can contain at most one token at any time, and the net is bounded and safe. The reason why the application net model is bounded and safe is because there is only one token in the *Init_Result* place initially (as shown in Figure 3). Therefore, after the *Application_Logic* transition fires for the first time, it cannot fire again until the result of the previous web services invocation returns. Similarly, the lower bound of a place is the number of tokens that the place must contain at any time. For example, the lower bound of place *User_DB* is 1, thus the place *User_DB* must contain at least one token at any time.

The *home* properties in Table 1 shows that all markings, including the initial marking M_0 , are *home markings*. According to Definition 5.5, a home marking M_{home} can be reached from any reachable marking; thus, at any time, the initial marking M_0 can be reached by progressing some further firing sequence. This proves that the application CPN model is *reversible*, and the net can always return to its initial state without leaving residual tokens in the net. Since the initial marking M_0 represents that there are no web service requests being processed at the net, the reversibility property indicates that every web service request can be processed successfully.

The analysis results tell us that there are no dead markings in our net model, and all transitions are live. Since a live transition means, from any reachable marking, we can always find a firing sequence containing the transition, according to Definition 5.3, our net model is live. Thus, for any marking M that is reachable from M_0 , it is possible to ultimately fire any transition of the net. As a consequence, as long as there are valid user requests with the needed permissions, both the *WS_Logic1* and *WS_Logic2* transition can fire eventually.

The analysis results also show that there are no dead transitions. A transition is *dead* if, in all reachable markings, the transition is not enabled. Dead transitions correspond to parts of the model that can never be activated, and they can be removed from the model without changing the model behaviors (Jensen et al., 2006). Therefore, our analysis result proves that all transitions in our net model can be activated eventually.

Similarly, we input our XML firewall net model defined in Figure 4 into the CPN Tools, the state space analysis tool produces the results as listed in Table 2. The analysis results show that our net model is *2-bounded*. Since there are two tokens in the *Init_Result_1* place of the application model initially, we expect that there can be at most two tokens in the *WS_Request* place, which represent two concurrent web service requests. This is proved by the upper bound of 2 in the *WS_Request* place as shown in Table 2. Similarly, the upper bound of 2 in the *WS_Req* place shows that two concurrent web service requests can actually be made if the user has passed the authentication, and has the needed permissions.

Table 2: Analysis results of the CPN model in Figure 4

Statistics	Boundedness Properties		

	Best Integer Bounds	Upper	Lower
State Space			
Nodes: 2065	Access_Req	2	0
Arcs: 6740	Access_Failed	2	0
Secs: 2	Add_Policy_Req	1	1
Status: Full	BG_DB	1	1
	Decision	1	0
Home Properties	FW_Result	2	0

Home Markings	FW_Result_1	2	0
[1604]	Init_Result_1	2	0
	Init_Result_2	1	0
	Insp_Result	2	0
Liveness Properties	New_Policy	1	0

Dead Markings	New_Policy_1	1	0
[1604]	Policy_DB	1	1
	Role_DB	1	1
	Session	2	0
Dead Transition Instances	State_DB	1	1
None	State_Info	2	0
	Sync	1	0
Live Transition Instances	UserInfo_DB	1	1
None	User_Info	2	0
	User_Perm	2	0
	User_Request	1	1
	User_Role	2	0
	User_Space	2	0
	Valid_User	2	0
	Valid_User_Req	2	0
	WS_Req	2	0
	WS_Request	2	0

From the *home* properties of the net model as shown in Table 2, we find that there is only one home making, which has the node number 1604. Since the node number of the initial marking M_0 is always 1, the result shows that the initial marking is not a home marking; thus, the XML firewall net model is not *reversible*. Furthermore, from the *liveness* properties, the single home marking (node 1604) is a *dead marking*. From Definition 5.6, we know that, in a dead marking, no transition is enabled. Therefore, when the net model reaches the dead marking, the net becomes dead, and cannot process further by firing any transitions. This indicates a deadlock error in our net model, and the net model is not *live*. To find out the cause of the deadlock error, we again use the state space analysis tool provided by the CPN Tools to trace the dead marking. As shown in Figure 5, we find the following firing sequence σ that leads to the dead marking, i.e., $M_0 [\sigma > M_{1603}$, where the initial marking M_0 is numbered as node N1, and the dead marking M_{1603} is numbered as node N1604.

$\sigma = N1, Application_Logic, N2, Application_Logic, N4, Checking_If_Existing, N10, Checking_If_Existing, N21, Existing_User, N42, Existing_User, N76, Start_Authorization, N129, Start_Authorization, N204, Assign_Role, N303, Assign_Role, N423, Fetch_State_Info, N563, Fetch_State_Info, N715, Comp_Logic, N876, Check_Conflict, N1038, Create_Session, N1186, Reject_Policy, N1341, Create_Session, N1466, Comp_Logic, N1604.$

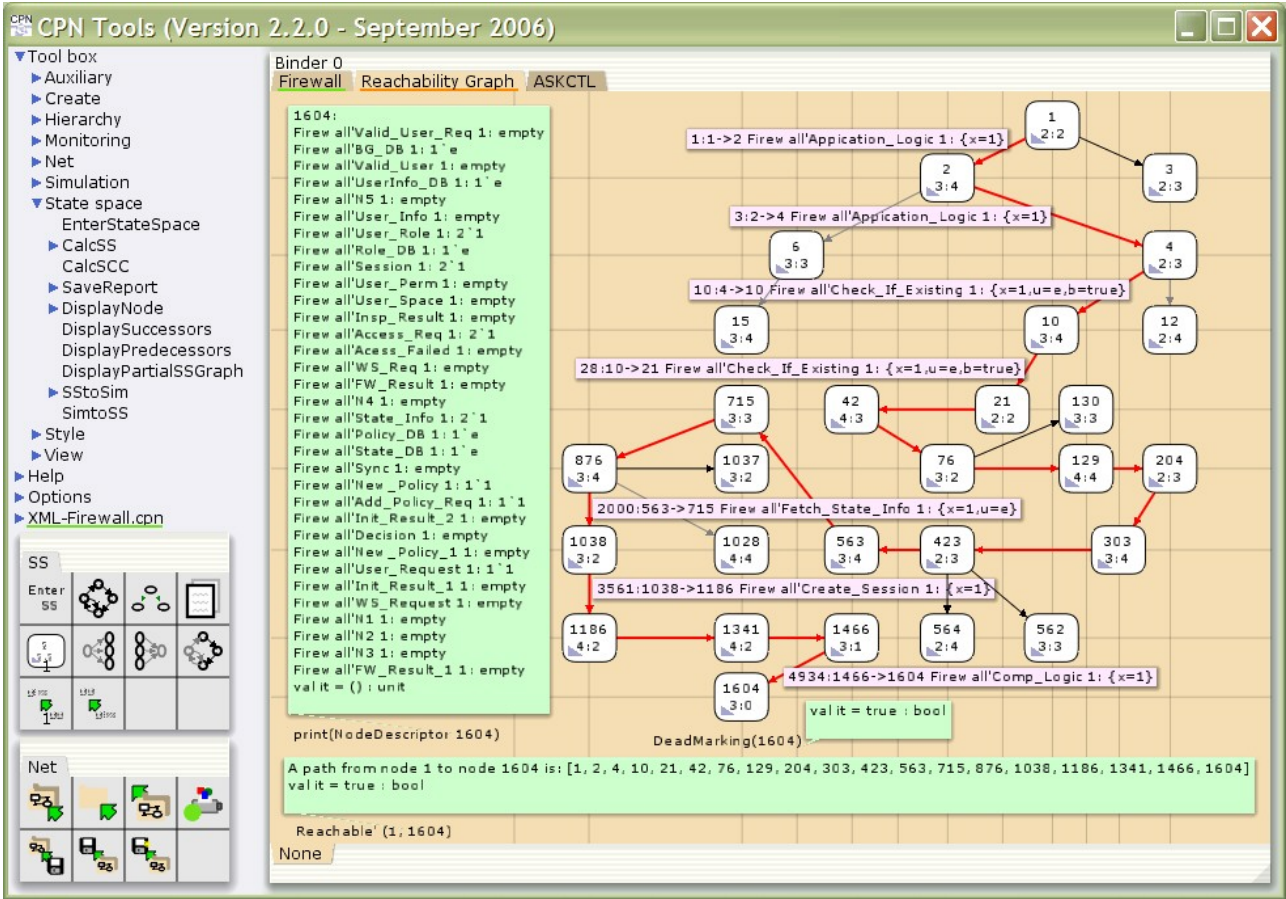


Figure 5. State space tracing of the dead marking state M_{1603} (i.e., Node 1604)

By simulating the XML firewall net model according to the firing sequence σ , it is easy to see that the dead marking M_{1603} (N1604) is due to the firing of the transition *Check_Conflict*, which takes away the unit token in place *Sync*. If the new policy is accepted and the policy database has been properly updated (i.e., when the transition *Update_Policy* fires), the unit token will be returned to the *Sync* place. In this case, the *Fetch_Policy* transition can fire as long as there are tokens in place *State_Info* and *User_Role*. However, if the new policy is rejected (as illustrated in the firing sequence σ), there will be no token returned to the *Sync* place; in this case, the transition *Fetch_Policy* becomes disabled forever, and thus, a deadlock situation occurs. The deadlock error can be corrected by adding a new arc from the transition *Reject_Policy* to place *Sync*, so a unit token can be returned to the *Sync* place when the new policy is rejected. Now we input our revised net model into the CPN Tools again, and we get the analysis results as listed in Table 3.

Table 3: Analysis results of the revised CPN model in Figure 4

Statistics	Boundedness Properties		
State Space	Best Integer Bounds	Upper	Lower
Nodes: 1475	Access_Req	2	0
Arcs: 5135	Access_Failed	2	0
Secs: 1	Add_Policy_Req	1	1
Status: Full	BG_DB	1	1

	Decision	1	0
Home Properties	FW_Result	2	0
-----	FW_Result_1	2	0
Home Markings	Init_Result_1	2	0
All	Init_Result_2	1	0
	Insp_Result	2	0
Liveness Properties	New_Policy	1	0
-----	New_Policy_1	1	0
Dead Markings	Policy_DB	1	1
None	Role_DB	1	1
	Session	2	0
Dead Transition Instances	State_DB	1	1
None	State_Info	2	0
	Sync	1	0
Live Transition Instances	UserInfo_DB	1	1
All	User_Info	2	0
	User_Perm	2	0
	User_Request	1	1
	User_Role	2	0
	User_Space	2	0
	Valid_User	2	0
	Valid_User_Req	2	0
	WS_Req	2	0
	WS_Request	2	0

From the analysis results in Table 3, we can see that all markings including the initial marking are *home markings*. Thus, our revised XML firewall net model is *reversible*. Furthermore, there are no *dead markings*, and all transitions are *live*. This proves that our revised net model is *live*. As a result, as long as there are valid user requests with needed permissions, the *WS_Logic* transition can fire eventually.

Note that the CPN models we have developed in this paper are compositional. This means we can easily develop a CPN model that consists of multiple applications, multiple firewalls, and multiple web services. Since both of the application model and the revised XML firewall model have been proved to be *reversible*, *bounded*, and *live*, due to the modular design of our formal approach, a compositional model with multiple applications, firewalls and web services is also reversible, bounded, and live.

6. Conclusions and Future Work

The security issues in service-oriented systems have become more and more important. Effective security mechanisms are critical for ensuring the successful deployment of web services. In this paper, we introduced a compositional CPN model for XML firewall protected service-oriented systems. We used the colored Petri net formalism because it has a distinct advantage of being easy to understand and use due to its graphical notations and powerful rules for defining system structure and dynamic behaviors (Murata, 1989, Jensen 1992). A colored Petri net provides an executable model that directly defines the concept of a system's state space. Although most research on automated analysis of concurrent and distributed systems uses some type of state-space exploration approach and cannot avoid the associated state-space explosion problem, based on our significant experience with Petri nets for many years, the Petri net formalism is capable of achieving an effective balance between theoretical concepts and practical techniques.

Our proposed model supports secured web services invocation, which only allows user requests with needed permissions. The effectiveness of our approach is due to the incorporation of the role-based access control (RBAC) mechanism into our security model, so user roles and permissions for web services invocation can be assigned dynamically. Although there are some existing implementations of XML firewall with limited functionality, our proposed approach provides a better solution to protecting service providers, where state-based user authentication and authorization are supported explicitly for web services invocation. More importantly, our XML firewall security model is formally defined using CPN, thus certain behavioral properties such as deadlock-freeness can be formally verified. The compositional CPN model we proposed consists of the application model and the XML firewall model, which can be analyzed separately; therefore the state-space explosion problem in our formal approach is not significant. To demonstrate the advantages of our formal approach, we used the CPN Tools to verify some key properties of our net model. Our analysis results show that our proposed net model (the revised model) is live and bounded, which indicate that our net model is deadlock free and only requires bounded resources. Different from other existing work, our approach ensures a correct design of XML firewall, which can serve as a reliable high-level software design for implementation. In our future work, we plan to refine our CPN models into a more detailed design using colored tokens with more semantics such as users, their roles, access permissions, and constraints, and show how to implement XML firewalls based on our proposed formal CPN models.

Acknowledgments: This material is based upon work supported by the Chancellor's Research Fund and UMass Joseph P. Healey Endowment Grants, and the Research Seed Initiative Grant, College of Engineering, UMass Dartmouth. We thank all anonymous referees for the careful review of this paper and the many suggestions for improvements they provided.

References

- Aalst, W. M. P. van der (2002). Making work flow: on the application of Petri nets to business process management. In J. Esparza and C. Lakos, editors, *Application and Theory of Petri Nets 2002*, Vol. 2360, *Lecture Notes in Computer Science*, pages 1-22. Springer-Verlag, Berlin, 2002.
- Albert, Ken, Jensen, K., and Shapiro, R. (1989). DESIGN/CPN: a tool package supporting the use of colored nets. *Petri Net Newsletter*, No. 32, pages 22-35. Bonn, Germany: Gesellschaft für Informatik (GI), Special Interest Group on Petri Nets and Related System Models, April 1989.
- Allen, D. (2006). *Forum Systems' XWall Web Services Firewall*. Retrieved on February 29, 2006, from <http://www.networkmagazine.com/shared/article/showArticle.jhtml?articleId=18900090>
- Ayachit, M. M. and Xu, H. (2006). A Petri net based XML firewall security model for web services invocation. *Proceedings of the International Conference on Communication, Network, and Information Security (CNIS 2006)*, October 2006, MIT, Cambridge, Massachusetts, USA, pp. 61-67.

- Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, I. M., Ferris, C., and Orchard, D. (2004). Web services architecture. *W3C Working Group Note*, February 11, 2004. Retrieved on January 18, 2007, from <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>
- Bouroulet, R., Kludel, H., and Pelz, E. (2004). A semantics of security protocol language (SPL) using a class of composable high-level Petri nets. *Proceedings of the Fourth International Conference on Application of Concurrency to System Design (ACSD'04)*, 2004.
- Christensen, S. and Petrucci, L. (1992). Towards a modular analysis of colored Petri nets, *Proceedings of the 13th International Conference on Application and Theory of Petri Nets (ICATPN-92)*, In: Jensen, K.: *Lecture Notes in Computer Science*, Vol. 616, Sheffield, UK, pages 113-133. Springer-Verlag, June 1992.
- Clack, C., Myers, C., and Poon, E. (1993). *Programming with Standard ML*. Prentice-Hall, 1993.
- Cremonini, M., Vimercati, S. D. C., Damiani, E., Samarati, P. (2003). An XML-based approach to combine firewalls and web services security specifications. *Proceedings of the 2003 ACM Workshop on XML Security*, pp. 69-78.
- DataPower (2006). *WebSphere DataPower SOA Appliances: XS40 XML security gateway*. Retrieved on March 15, 2006, <http://www.datapower.com/products/xs40.html>
- Feinstein, H., Sandhu, R., Coyne, E., and Youman, C. (1996). Role-based access control models. *IEEE Computer*, 29(2):38-47, 1996.
- Fernandez, E. B. (2004). Two patterns for web services security. *Proceedings of the 2004 International Symposium on Web Services and Applications (ISWS'04)*, Las Vegas, NV, 2004.
- Fernandez, E. B., Larrondo-Petrie, M. M., Seliya, N., Delessy-Gassant, N., and Schumacher, M. (2005). A pattern language for firewalls. In M. Schumacher, E. B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad (Eds.), *Security Patterns*, Wiley 2005.
- Giuri, L. and Iglío, P. (1997). Role templates for content-based access control. *Proceedings of the Second ACM Workshop on Role Based Access Control*, Virginia, USA, 1997.
- Gralla, P. (2007). *XML Firewalls*. The Web Services Advisor, January 7, 2007. Retrieved on January 9, 2007, from http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci855052,00.html
- Hamadi, R., Benatallah, B. (2003). A Petri net-based model for web service composition. *Database Technologies 2003*, Eds. K. D. Schewe, X. Zhou, Australian Computer Science Society Inc., Sydney, Australia, 2003, pp. 191-200.
- Liu, B. and Chen, H. (2005). Web service composition and analysis: a Petri-net based approach. *Proceeding of the First International Conference on Semantics, Knowledge and Grid (SKG'05)*, 2005.
- Jalilvand, A. and Khanmohammadi, S. (2004). Modeling of flexible manufacturing systems by timed Petri net. *Proceedings of the International Conference on Computational Intelligence*, 2004, pp. 141-144.
- Jensen, K. and Rozenberg, G. (eds.) (1991). *High-level Petri nets: theory and application*. New York: Springer-Verlag.

- Jensen, K. (1992). *Coloured Petri nets: basic concepts, analysis methods and practical use*. Vol. I : Basic Concepts. EATCS Monographs on Theoretical Computer Science. New York Springer-Verlag.
- Jensen, K. (1998). An introduction to the practical use of coloured Petri nets. In W. Reisig and G. Rozenberg (Editors): *Lectures on Petri Nets II: Applications*, Lecture Notes in Computer Science, Vol. 1492, Springer-Verlag 1998, pp. 237-292.
- Jensen, K., Kristensen, L. M., and Wells, L. (2006). Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*. Springer-Verlag, 2006.
- Juric, M. B. (2006). Extending BPEL with WSIF for enterprise application integration. *BPEL Cookbook: Best Practices for SOA-Based Integration and Composite Applications Development*, Packt Publishing, July 2006.
- Moradian, E. and Håkansson, A. (2006). Possible attacks on XML Web Services. *IJCSNS International Journal of Computer Science and Network Security*, Vol.6, No.1B, January 2006, pp. 154-170.
- Murata, T. (1989), Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4): 541-580, April 1989.
- Mysore, S. (2003). Securing web services - concepts, standards, and requirements, *White paper*, Sun Microsystems, October 2003.
- Nagappan, R., Skoczylas, R., and Sriganesh, R. P. (2003). *Developing Java web services*, Wiley, 2003.
- Pfleeger, C. P. and Pfleeger, S. L. (2003). *Security in Computing*, 3/e Prentice Hall, 2003.
- Ratzer, A.V., Wells, L., Lassen, H. M., Laursen, M., Qvortrup, J. F., Stissing, M. S., Westergaard, M., Christensen, S., and Jensen, K. (2003). CPN Tools for editing, simulating, and analysing coloured Petri nets. *Proceedings of the 24th International Conference on the Application and Theory of Petri Nets*, Eindhoven, Netherlands, June 2003.
- Toumodge, S. (1995). Applications of Petri nets in manufacturing systems: modeling, control, and performance analysis. *IEEE Control Systems Magazine*, Vol. 15, Issue 6, December 1995.
- Vorobiev, A. and Han, J. (2006). Security attack ontology for web services. *Proceedings of the Second International Conference on Semantics, Knowledge, and Grid (SKG'06)*, 2006, pp. 42.
- Windley, P. J. (2003). Closing the XML security gap. *InfoWorld*, October 17, 2003. Retrieved on December 22, 2006, from http://www.infoworld.com/article/03/10/17/41TCxmlfire_1.html
- Wrenn, G. (2004). Securing web services: a job for the XML firewall. *Web Services Tips for XML Developers*, SearchWebService.com, March 8, 2004. Retrieved on January 18, 2007, from http://searchwebservicestechtarget.com/tip/1,289483,sid26_gci955191,00.html
- Xu, D. and Nygard, K. E. (2005). A threat-driven approach to modeling and verifying secure software. *Proceedings of the 2005 IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*, November 2005, pp. 342-346.

- Xu, D. and Nygard, K. E. (2006). Threat-driven modeling and verification of secure software using aspect-oriented Petri nets. *IEEE Transactions on Software Engineering (IEEE TSE)*, April 2006, Vol. 32, No. 4, pp. 265-278.
- Xu, H. and Shatz, S. M. (2003a). A framework for model-based design of agent-oriented software. *IEEE Transactions on Software Engineering (IEEE TSE)*, January 2003, Vol. 29, No. 1, pp. 15-30.
- Xu, H. and Shatz, S. M. (2003b). ADK: an agent development kit based on a formal model for multi-agent systems. *Journal of Automated Software Engineering (AUSE)*, October 2003, Vol. 10, No. 4, pp. 337-365.
- Xu, H., Zhang, Z., and Shatz, S. M. (2005). A security based model for mobile agent software systems. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, August 2005, Vol. 15, No. 4, pp. 719-746.
- Zhang, G. and Parashar, M. (2004). Context-aware dynamic access control for pervasive applications. *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2004)*, 2004 Western MultiConference (WMC), San Diego, CA, USA, January 2004.