Securing Service-Oriented Systems Using State-Based XML Firewall

Prof. Haiping Xu
Computer and Information Science Department
University of Massachusetts Dartmouth
North Dartmouth, MA 02747
Email: hxu@umassd.edu
URL: http://www.cis.umassd.edu/~hxu/

July 2, 2008

Acknowledgment

UMass Dartmouth
developing innovative technology for a changing world

- **Abhinay K. Reddyreddy**
  Graduate Student
  Concurrent Software Engineering Lab (CSEL)
  Computer and Information Science Department
  University of Massachusetts Dartmouth

- **Mihir M. Ayachit** (former graduate student, 2006)
  Software Engineer
  Parametric Technology Corporation
  Needham, Massachusetts

WE HAVE A STRONG TEAM!
• Background and Motivations
• XML Firewall Architectural Design
• Role-Based Access Control Polices
• Real-Time Detection of XML-Based Attacks
• A Case Study with Experimental Results
• Conclusions and Future Work

Introduction to Service-Oriented Architecture and Web Services

• Service-Oriented Architecture (SOA) is a key means to share information and capabilities through published service interface.
• Web services-based SOA can significantly speed up the application development process.
  – Are Internet-based software components that support open, XML-based standards and communication protocols.
  – Are software components defined using WSDL, registered using UDDI, and invoked using SOAP.
  – Make software functionalities available over the Internet.
In the context of SOA, a service is a self-contained function in which consumers interact through a well-defined interface.

- **Service Provider** implements the service and makes it available over the network.
- **Service Requester** utilizes an existing web service by opening a network connection and sending a request.
- **Service Broker** is a centralized directory of registered web services.

**Major concern: Services are vulnerable to be attacked!**

**Examples of XML-Based Attacks**

- **XML-Based Denial of Service (XDoS):** An XDoS attack directs malicious XML-based traffic to a web service to exhaust the resources at the server side.
- **SQL Injection:** An SQL injection attack could tamper the input fields of database requests to obtain unauthorized access to data or stored procedures.
- **Overloaded Payload:** An overloaded payload attack can exhaust the XML parser of a service provider by sending huge XML data in a service request.
**Firewall**: a fireproof wall used as a barrier to prevent the spread of a fire.

- Firewall is a component that limits network access.
- Three major types of conventional firewalls
  - Packet filtering firewall
  - Stateful inspection firewall
  - Application-level firewall

**Conventional Firewalls are NOT sufficient for SOA!**

**Why XML Firewall?**

- A conventional firewall typically
  - Restricts IP addresses or TCP ports, but port 80 reserved for HTTP and SOAP traffic cannot be blocked on a server that hosts web services.
  - Does not look into packet contents, and does not support parsing or validating XML data.
  - Does not support authentication and authorization for web services access.

- A state-based XML firewall can
  - Inspect XML messages including head and data segments.
  - Support authentication and authorization using state-based info for web services invocation.
Major Features of State-Based XML Firewall

- Is based on a Petri net based XML firewall formal model we proposed previously.
- Grants only those users who are properly authenticated and authorized for access of web services.
- Adopts dynamic role-based access control (D-RBAC) for user authorization.
- Is supported by policy rules based on user & state info
  - Policy rules for user authentication and authorization.
  - Detection rules for identifying XML-based security threats.
- Can examine the contents of incoming XML-based messages (SOAP messages).

XML Firewall Protected Service-Oriented System

- Works as a proxy for web services, and is transparent for users.
- Supports concurrent web services invocations.
- Supports real-time state-based policy updating.
• **User_Info** database: records user information, e.g., a user’s trust level.
  – Supports user authentication and user authorization.
  – Supports verification of XML-based attacks.
• **State_Info** database: records system states and user’s current state.
  – Supports detection of suspicious XML-based attacks.
  – Supports verification of XML-based attacks.
• **Definition 1:** A *user session* is defined as a 5-tuple \((UID, SID, RO, ST, ET)\), where \(UID\) is a user ID, \(SID\) is a session ID, \(RO\) is a set of roles that will be assigned to the user, \(ST\) is the session start time, and \(ET\) is the session expiration time.

• **Definition 2:** A *user credential* is a 4-tuple \((UN, PW, UID, TL)\), where \(UN\) is the user name, \(PW\) is the password specified by the user at registration time, \(UID\) is the user ID, and \(TL\) is the current trust level assigned to the user.

• **Definition 3:** A *user state* is a 5-tuple \((UID, SID, TR, FR, TL)\), where \(UID\) is the user ID assigned at the time of registration, \(SID\) is the ID of the session that is initiated, \(TR\) is the total number of requests made by the user in the current session, \(RF\) is the request frequency, i.e., the number of requests made by the user in a recent time interval, and \(TL\) is the user's current trust level.

• **Definition 4:** A *firewall state* is a triple \((RE, DE, RT)\), where \(RE\) is the number of requests that are received by the XML firewall but not yet forwarded to the web server, \(DE\) is the number of requests that are being processed by the detection modules in the SOAP filter, \(RT\) is the number of requests in a recent time interval, e.g., the last five minutes.

**Examples of User and State Information for XDoS Detection**

**Architectural Design of State-Based XML Firewall**
A role is an abstraction of a position
- Represents a set of responsibilities.
- Represents a set of permissions.

Role-based authorization polices
- Specify the roles that a user may adopt.
- Specify the permissions associated with each role.
- Can be updated dynamically at runtime.

Separation of policies from the application code
- Policy language must be expressive and computable
- We choose Prolog for policy specification.

isValidRole(patient). isValidRole(doctor). isValidRole(nurse).
isValidRole(staff). isValidRole(pharmacist).
assignRole(U,R) :- isValidRole(R).
canInvoke(R,T,billingService,accessBill):-
contains(R,[staff,pharmacist,patient]),
contains(T,[normal,high]).
canInvoke(R,T,billingService,computeBill):-
contains(R,[staff,pharmacist]),
contains(T,[normal,high]).
canInvoke(R,T,accessService,readRecord):-
contains(R,[doctor,nurse,patient]),
contains(T,[normal,high]).
canInvoke(R,T,accessService,writeRecord,P,U):-
contains(R,[doctor,nurse]),
contains(T,[normal,high]), assignPatient(P,U),
assignRole(P,patient), assignRole(U,R).
canInvoke(R,T,contactService,accessContact):-
contains(R,[staff,doctor,nurse,patient]),
contains(T,[normal,high]).
Architectural Design of State-Based XML Firewall

UMass Dartmouth
developing innovative technology for a changing world

Real-Time Detection of XML-Based Attacks

UMass Dartmouth
developing innovative technology for a changing world

• SOAP filter is responsible for real-time detection of XML-based attacks
  – Detection of suspicious SOAP messages
  – Verification of XML-based attacks
• Example of suspicious XDoS attack detection rules

checkThreshold(W,S,X):- threshold(W,SI,Y),X > Y.
threshold(accessService,busy,20).
threshold(accessService,normal,40).
threshold(accessService,free,60).
Real-Time Detection of XML-Based Attacks (contd.)

Example of XDoS attack verification

```
xdosVerify(U,T):- inspectHistory(U,T,V).
inspectHistory(U,T,V):-
  T = high, dataConnect(U,3,V), V = '3',
  degradeTrustLevel(U,normal).
inspectHistory(U,T,V):-
  T = normal, dataConnect(U,5,V), V = '3',
  degradeTrustLevel(U,low).
inspectHistory(U,T,V):-
  T = low, dataConnect(U,7,V), V = '3',
  degradeTrustLevel(U,permanentlyBlocked)

dataConnect(U,X,V):-
  java_object('DataConnect',[],data),
  data<-getHistorySessionStatus(U,X) returns V.

degradeTrustLevel(U,T):-
  java_object('DataConnect',[],data),
  data<-recordTrustLevel(U,X).
```

A Case Study (SQL Injection Attack)

- Develop a prototype XML firewall and a service-oriented system for hospital management.
- Simulate an SQL injection attack by accessing the web service `accessService`.
- Consider User1 with a patient role who is assigned to nurse User2. A legitimate request from the nurse could be
  ```
  writeRecord("User2", "User1", "The patient reacted abnormally to new drugs.", "Observation")
  INSERT INTO patientRecords VALUES('User2', 'User1', 'The patient reacted abnormally to new drugs.', 'Observation');
  DELETE FROM users; -- dummystring');
  ```
Log Information for SQL Injection Detection

A Case Study (XDoS Attack)

- Simulate request flooding attacks on a service (report generation service)
  - Use large number of requests from the attacker.
  - Record the response behavior from a normal user.
- The attacked service takes around 10 seconds as normal processing time.
- Perform two experiments with thresholds for the firewall are set to 80 and 60, respectively.
Experimental Results for XDoS Attacks

Response Time vs. Number of Requests Per Minute

Conclusions

- We introduced a new security mechanism called **state-based XML firewall** to protect service-oriented systems.
- The system supports attack detection and attack verification.
- Experimental results show that our approach can effectively protect web services from XML-based attacks.
Future Work

- Study new types of XML-based attacks and show how their corresponding attack verification modules can be easily integrated.
- Adopt agent-based technology to introduce more intelligence into our current design.
- Develop an agent-based XML firewall with learning capability that can protect service-oriented systems for novel attacks.