Desenvolvimento do Modelo WSIVM para Aperfeiçoar a Segurança em SOA e Serviços Web

Rafael Bosse Brinhosa, Carla Merkle Westphall, Carlos Becker Westphall

Departamento de Informática e Estatística (INE)
Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

{brinhosa, carlamw, westphal}@inf.ufsc.br

Abstract. *The SOA architecture primarily based on Web Services is having a steady adoption, despite this growth is lower than expected when it was established, that´s mainly because of difficulties defining security related aspects. Web Services inherited a lot of well-known security problems of web applications and brought new ones. The majority of attacks today are consequences of bad input validation in the application-level. This paper presents how to implement an input validation model for Web Services, which can be used to prevent Cross-site Scripting and SQL injection through using predefined models´ specification of valid inputs. The proposed WSIVM (Web Services Input Validation Model) consists of a XML Schema, a XML Specification and a module for performing input validation according to the schema. A case study showing the performance results is also presented.*

Resumo. *O uso da arquitetura SOA baseado principalmente na utilização de serviços web tem obtido larga adoção, porém, devido às dificuldades encontradas quanto aos aspectos de segurança, este crescimento tem sido menor do que o esperado. O uso de serviços web herdou muitos problemas de segurança conhecidos em aplicações web e trouxe outros novos. Atualmente a maioria dos ataques são conseqüência da má validação de entradas de dados no nível das aplicações. Este trabalho propõe um modelo para validação de entradas de dados para serviços web que pode ser utilizado para impedir ataques como Cross-site Scripting e SQL injection através da especificação de modelos pré-definidos de entradas válidas. O modelo proposto denominado WSIVM (Web Services Input Validation Model) possui um XML Schema, uma especificação XML e um módulo que faz a validação das entradas de acordo com a especificação. Um estudo de caso demonstrando resultados de desempenho também é apresentado.*

1. Introdução

Com o advento de diferentes tecnologias de colaboração e compartilhamento de informações, novos modos de interações estão evoluindo e gerando novos requisitos para o desenvolvimento de aplicações distribuídas. Empresas estão testemunhando o aumento de colaboração e compartilhamento de informações e uma maior necessidade do uso de recursos distribuídos e de computação [Belapurkar et al., 2009].

O paradigma de arquitetura orientada a serviços (SOA - *Services Oriented Architecture*) transformou a Internet de um repositório de dados para um repositório de

serviços [Saleem et al. 2010]. No estilo SOA uma aplicação é composta por serviços reusáveis que são integrados através de suas interfaces padronizadas.

A tecnologia de serviços web (*Web Services*), fundamentada no uso de padrões abertos, facilita a troca de informações, a interoperabilidade e o reuso de software e, por isso, é considerada um dos principais componentes da arquitetura orientada a serviços. Serviços web são componentes de software descritos que podem ser descobertos e usados para implementar aplicações. Serviços web são adequados para integrar sistemas heterogêneos, pois fazem uso extensivo da linguagem XML (*Extensible Markup Language*) [Nordbotten, 2009]. A interface de um serviço web, por exemplo, é descrita usando uma linguagem baseada em XML, a WSDL (*Web Services Description Language*). Além disso, a comunicação entre as partes de uma aplicação distribuída é executada usando mensagens SOAP (*Simple Object Access Protocol*) baseadas em XML.

Há um uso prático grande de serviços web prontos e disponíveis, que podem ser usados para se obter informações sobre o clima, sobre ações da bolsa de valores, sobre o código postal (CEP) [Cep, 2011] ou ainda para disponibilizar informações organizacionais do governo federal [Siorg, 2009].

Apesar do paradigma SOA apresentar diminuição de custos com a eliminação de esforços redundantes por reusar software, a segurança é uma das principais preocupações, conforme o pesquisador do Instituto Gartner [Feiman, 2010].

Para a implementação de segurança em serviços web, diversos padrões e especificações foram criados. No entanto, apenas o uso dos padrões corretos não garante que a segurança desejada seja obtida [Viega e Epstein, 2006] [Jensen et al. 2009] [Lakshminarayanan, 2010].

O relatório do instituto SANS (*SysAdmin, Audit, Network, Security*) [Sans, 2009] que lista os maiores riscos à segurança cibernética e a comunidade OWASP (*Open Web Application Security Project*) [Owasp, 2011] descrevem que a validação de entradas de dados pode ser um dos controles mais efetivos para a segurança de aplicações web.

A validação de entradas de dados [Bertino et al., 2009] [Scholte, 2011] é um conjunto de controles que uma aplicação deve fazer nas suas entradas de dados nos aspectos léxicos, sintáticos e de checagem de tipos, integridade e origem. A falta desses controles tornou-se um dos maiores problemas dos softwares que fazem interface com a Internet devido à facilidade de sofrer ataques e a necessidade de validar as entradas de dados dos usuários de forma a permitir apenas entradas confiáveis.

Desta forma, no ambiente de SOA e serviços web, usados por várias empresas, o aperfeiçoamento da segurança com o uso de mecanismos mais robustos de validação de dados passou a ser fundamental [Owasp, 2011] [Scholte, 2011].

Este trabalho propõe um modelo para validação de entradas de dados em serviços web, fornecendo proteção contra ataques baseados na entrada de dados maliciosos. O modelo proposto é chamado WSIVM (*Web Services Input Validation Model*) e contém um mecanismo para validação, um esquema XML, uma especificação XML e um módulo que faz a validação.

O texto do artigo está organizado da seguinte maneira: na seção 2 são apresentados os trabalhos relacionados; a seção 3 descreve os problemas de segurança em serviços web; na seção 4 o modelo WSIVM proposto é apresentado; a implementação é descrita na seção 5; um estudo de caso é mostrado na seção 6; e a seção 7 finaliza o artigo com as conclusões.

2. Trabalhos Relacionados

A falta de validação de entradas de dados é a maior causa de ataques a aplicações Web [Cenzic, 2009] [Owasp, 2011] [Scholte, 2011], sejam estas aplicações desenvolvidas com serviços web ou com outras tecnologias. Isso acontece porque a falta de validação de entradas de dados possibilita a ocorrência de vários dos ataques listados em [Sans, 2011] [Owasp, 2010]. Dentre os ataques que podem ser citados estão a injeção de códigos maliciosos usando SQL (*Structured Query Language*) e *Cross-Site Scripting* que permite a execução de códigos (*scripts*) no browser do lado do cliente com o objetivo de realizar uma ação maliciosa [Owasp, 2010].

Muitos trabalhos têm sido realizados no intuito de garantir a validação de entradas em aplicações web como a biblioteca *Microsoft Anti-Cross Site Scripting Library* [Microsoft, 2010] e o uso de soluções Open-source em PHP. No entanto, existem poucos mecanismos específicos para serviços web.

No que tange a mecanismos de implementação de segurança para serviços web não seguros, o MIT (*Massachusetts Institute of Technology*) possui uma implementação denominada WS-Security Wrapper [Ssa, 2007] que é um intermediador entre o serviço web e o cliente fazendo a validação de certos aspectos. No entanto, este trabalho foi desenvolvido para possuir compatibilidade apenas com serviços web desenvolvidos em plataforma Microsoft.Net versão 1.1 e não inclui características como a validação de entradas de dados pré-definidas.

Um mecanismo reusável e independente para entrada de dados é muito importante no processo de criação de um serviço web seguro. O mecanismo proposto neste trabalho, o WSIVM, ajuda nesta tarefa de forma diferenciada dos demais trabalhos analisados. Primeiramente, porque foca principalmente no aspecto de tratamento de entrada de dados, diferente do trabalho IAPF (*Integrated Application and Protocol Framework*) [Sidharth e Liu, 2007] que busca abordar todos os aspectos de segurança relacionados com serviços web. Além disso, outros trabalhos [Sun e Li, 2008] [Nordbotten, 2009] tem seu foco no uso de tecnologias existentes como a cifragem XML para garantir a segurança de serviços web, mas não mencionam a validação de entradas de dados.

Com relação aos aspectos de validação de entradas em aplicações web, existem alguns trabalhos como [Lin e Chen, 2009] que desenvolveram uma ferramenta que insere automaticamente a validação de entradas no lado do servidor eliminando vulnerabilidades de inserções maliciosas. No entanto, esta abordagem possui uma desvantagem que é a grande obtenção de falsos-positivos, isto é, o validador pode falhar considerando uma mensagem inválida, no entanto, a mensagem é válida.

Além dos trabalhos já relacionados existe ainda outra categoria de trabalhos, focados no desenvolvimento de *firewalls* como [Bebawy et al. 2005], que trata a proteção contra ataques de negação de serviço e de estouro de pilha. Os firewalls de XML, apresentado em [Blyth, 2009], aborda validação da estrutura do conteúdo XML,

mas não do conteúdo propriamente dito. O trabalho de [Loh et al., 2006] menciona a proteção contra injeção de SQL através do *XML Schema* e de uma lista negra pré-compilada de comandos SQL - essa abordagem costuma apresentar muitos falsos-positivos, no entanto, detalhes sobre a eficácia deste trabalho com testes mais extensos não foram apresentados.

Dentre os trabalhos relacionados observa-se que existe uma falta de trabalhos abordando a validação de entradas especificamente para serviços web.

3. Problemas de Segurança em Serviços Web

Serviços web criam novos riscos à segurança das organizações porque antigos métodos de proteção como *firewalls* e antivírus não são capazes de protegê-los. Por serem desenvolvidos para fazerem uso do protocolo HTTP (*Hypertext Transfer Protocol*) utilizando a porta 80, os *firewalls* comuns que atuam na camada de rede permitem o fluxo normal das requisições HTTP sem bloqueá-lo.

Além disso, suas funcionalidades são expostas através de arquivos WSDL, sendo que através das descrições dos métodos e variáveis do arquivo WSDL é possível obter importantes informações para um ataque conhecido como *WSDL scanning* [Belapurkar et al., 2009].

Existem ataques diretamente relacionados com a manipulação de dados: *Cross-Site Scripting* (ou XSS) e injeção de SQL. O trabalho de [Kieyzun et al., 2009] classifica os ataques XSS em dois tipos: ataques XSS de primeira ordem e de segunda ordem.

No ataque XSS de primeira ordem, a vulnerabilidade resulta de a aplicação inserir parte da entrada do usuário na própria página. O usuário malicioso usa engenharia social para convencer a vítima a clicar em uma URL que contenha código malicioso HTML/JavaScript. O navegador web do usuário mostra a página HTML e executa o JavaScript que fazia parte da URL maliciosa recebida, resultando no roubo de *cookies* de sessão ou outros dados sigilosos do usuário. Este tipo de ataque dificilmente pode ser elaborado contra serviços web.

Nos ataques XSS de segunda ordem a vulnerabilidade resulta do armazenamento das entradas maliciosas do usuário no banco de dados da aplicação, então, quando a página HTML é acessada, o código é executado e mostrado para as vítimas (por exemplo, em páginas de redes sociais). Os ataques de segunda ordem são mais difíceis de serem evitados porque a aplicação necessita validar ou sanear entradas de dados que possam conter código de *script* executável. Aplicado a serviços web é possível sofrer ataques ao apresentar dados não validados diretamente ao usuário. Por exemplo, fazendo-se uso de AJAX (***A**synchronous **J**avascript **a**nd **XML***), podem-se obter dados fornecidos por serviços web de terceiros que podem estar contaminados. Usando, por exemplo, o comando `document.write(xmlhttp.responseText);` caso a resposta desta chamada AJAX feita para um serviço web contenha dados HTML e JavaScript, estes dados serão interpretados e executados oferecendo risco ao usuário.

O ataque de injeção de código SQL (*SQL injection*) funciona através de entradas de dados maliciosos visando a execução de comandos SQL no banco de dados [Clarke, 2009] [Kieyzun et al., 2009] [Owasp, 2010].

Em serviços web que não possuem o devido tratamento de exceção, a mensagem de erro pode conter dados valiosos para o atacante utilizar. Assim, através de tentativa e erro, o atacante pode descobrir que tecnologia de banco de dados está sendo utilizada, que tabelas existem que podem ser exploradas e todas as informações necessárias para efetuar um ataque. Ataques de injeção de SQL podem elevar privilégios, sendo possível executar comandos em modo de administrador no servidor comprometido. É possível testar se um serviço web é vulnerável através do envio de requisições SOAP com os parâmetros devidamente manipulados. Por exemplo, enviando “"1=1 –“ como parâmetro para um determinado serviço, é possível que se tenha como retorno as saídas das figuras 1 e 2. Podem-se observar nessas figuras, dois exemplos de respostas de saídas de erro do banco de dados que foram retornadas pelo servidor. Essas respostas podem ser utilizadas para descobrir detalhes de banco de dados e servir para enviar novas solicitações ao banco, obtendo-se mais detalhes, até efetuar comandos mais complexos no banco de dados, que podem resultar na elevação de privilégios, injeção de arquivos, roubo ou destruição do banco de dados.

Através da resposta de saída da Figura 1 pode-se identificar que se trata de um banco de dados MySQL e que a injeção do comando SQL foi efetuada, pelo tipo de erro retornado. Na resposta de saída da Figura 2, conclui-se que o nome de uma das colunas do banco de dados é `ItemId`, pela sintaxe do comando `select` do SQL e o banco de dados pode ser deduzido como Microsoft SQL Server, pelo uso da sintaxe de *stored procedure* “`dbo.`”.

```
ERROR: The query was not accomplished. Description: 1064 - You have an
error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '1=1'' at line 1
```

Figura 1. Exemplo de resposta de saída 1.

```
Line 11: Incorrect syntax near ''')) or  ItemId in (select ItemId
from dbo.GetItemParents('4''. Unclosed quotation mark before the
character string ')) ) ) > 0 '.
```

Figura 2. Exemplo de resposta de saída 2.

O ataque do tipo *Blind SQL injection* trata-se de um tipo de injeção de SQL no qual os resultados não são exibidos para o atacante. É muito comumente utilizado contra serviços web, pois muitos servidores evitam que as mensagens de erro geradas pelo serviço cheguem ao usuário. Em serviços web geralmente o erro HTTP 500 é retornado quando uma tentativa de *Blind SQL injection* é efetuada, no entanto, existem técnicas como a medição dos tempos de resposta do servidor que podem ser utilizadas para determinar os parâmetros necessários para efetuar o ataque com sucesso.

Apesar de ser difícil conseguir informações confiáveis sobre incidentes de segurança ou vazamentos de dados como relatado no livro [Shostack e Stewart, 2008], em [Databreaches, 2009] é possível observar que os maiores casos de ataques em que ocorreram roubos de dados tiveram relação com a injeção de dados maliciosos. Na tabela existente em [Databreaches, 2009], por exemplo, consta o caso da entidade Heartland Payment Systems no qual foram perdidos 130.000.000 de dados e pode-se ler o motivo do incidente: injeção de código SQL.

4. WSIVM – Modelo de Validação de Entradas de Dados

Nesta seção é descrito o modelo WSIVM (*Web Services Input Validation Model*) que propõe validar as entradas de dados para fornecer segurança para serviços web. Inicialmente é descrito o funcionamento de serviços web sem o uso do modelo. Na seqüência é feita a explicação do funcionamento de serviços web usando o modelo proposto.

4.1. Funcionamento de Serviços Web sem o WSIVM

De maneira tradicional, um cliente encontra o serviço web que necessita. Este encontro ou descoberta é feita através de pesquisas em repositórios de serviços web. Os repositórios armazenam as referências dos serviços web no formato UDDI (*Universal Description, Discovery and Integration*). UDDI é um protocolo padrão que especifica um método para publicar e descobrir diretórios de serviços em uma arquitetura orientada a serviços. Depois da descoberta o cliente realiza uma requisição ao serviço web (Figura 3). O serviço web retorna a resposta do cliente contendo a resposta de sua requisição (Figura 4).

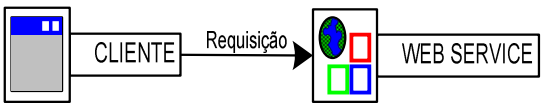

Figura 3. Requisição ao serviço web.

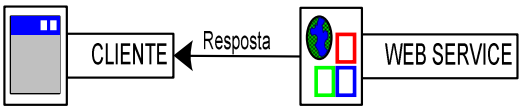

Figura 4. Resposta do serviço web.

Neste processo tradicional, a requisição do cliente é feita diretamente ao serviço web, que processa as entradas e retorna o resultado. O padrão utilizado para esta troca de mensagens é o formato SOAP baseado em XML.

Supondo que o serviço web executa a entrada do usuário sem fazer nenhum tipo de validação e que uma entrada de usuário faça parte da consulta SQL descrita em (1).

```
SELECT nome, idade FROM clientes WHERE nome=entrada;          (1)
```

Se o usuário fornecer como entrada o nome “Paulo”, que é uma entrada válida para o nome, o serviço web retornará como resposta o nome e a idade do cliente “Paulo”. Entretanto, se o usuário fornecer a seguinte *string* como entrada maliciosa “`Paulo' UNION SELECT nome, senha FROM clientes; --`“, a consulta SQL ficaria representada em (2).

```
SELECT nome FROM clientes WHERE
nome='Paulo' UNION SELECT nome, senha FROM clientes; --   (2)
```

Como o serviço web não realiza validação desta entrada representada em (2), retornará um valor sigiloso como a “senha do cliente” para o usuário que enviou a requisição maliciosa para o serviço web. Para o WSDL, a requisição é válida, pois se trata de uma *string* conforme especificado na descrição do serviço. No entanto, a confiança na entrada do usuário e a falta de validação da mesma acarretaram numa vulnerabilidade que divulgou dados sigilosos.

4.2. Funcionamento de Serviços Web com o WSIVM

O modelo WSIVM (*Web Services Input Validation Model*) propõe validar as entradas de dados para fornecer segurança para serviços web.

O modelo proposto traz algumas vantagens em comparação com a validação de entradas feita normalmente em uma aplicação, já que: (a) evita o desperdício de processamento do servidor com mensagens inválidas; (b) diminui a possibilidade de ataques de negação de serviço utilizando o conteúdo das mensagens; e, (c) é independente da tecnologia utilizada para o desenvolvimento interno dos serviços.

Com o WSIVM, o exemplo descrito na seção 4.1 é executado da seguinte forma: o usuário faz a requisição de forma usual conforme demonstrado na Figura 3, no entanto, esta requisição é validada pelo WSIVM (Figura 5).

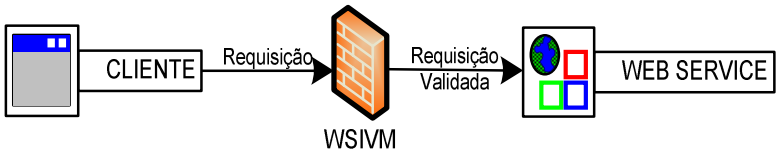

Figura 5. Requisição SOAP com WSIVM.

Caso seja enviada uma requisição maliciosa, como demonstrado no exemplo da entrada representada em (2), ao invés de enviar a senha, o mecanismo de validação do WSIVM faz a validação e retorna um erro genérico para o usuário. Desta forma o serviço web não recebe a solicitação mal-intencionada (Figura 6).

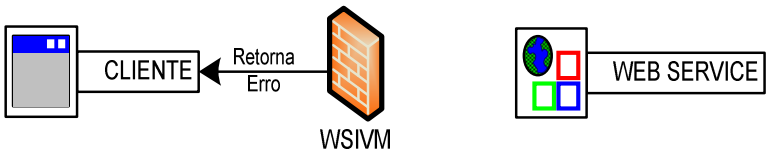

Figura 6. WSIVM bloqueia solicitação maliciosa.

De forma mais detalhada, o que ocorre quando uma mensagem chega para ser validada no WSIVM, é que a entrada enviada pelo usuário é validada através de um módulo que foi desenvolvido com a especificação XML que se encontra no servidor, conforme o algoritmo da Figura 7.

```
1. Recebe requisição SOAP
2. Processa mensagem XML
3. Checa parâmetros
4. Se checagem tiver falha
   Então Retorna mensagem de erro
   Senão Serviço web processa a requisição
```

Figura 7. Algoritmo da Validação das Mensagens.

Este processo evita o consumo desnecessário de recursos do servidor, assim, considerando o exemplo, o serviço web não executaria a consulta SQL caso fosse enviada uma requisição maliciosa.

A interação dos componentes do modelo WSIVM é representada na Figura 8.

O *WSIVMModule* é um módulo responsável por chamar os outros componentes.

O *WSIVMValidator* mapeia a mensagem SOAP obtendo os campos do corpo da mensagem e envia para o *WSIVMXMLLoader*.

O *WSIVMXMLLoader* carrega os elementos e as regras especificadas na especificação XML e verifica com o *WSIVMVerifier* a validade ou não da resposta.

O *WSIVMVerifier* contém todas as regras pré-definidas para validação das entradas e é responsável por validá-las.

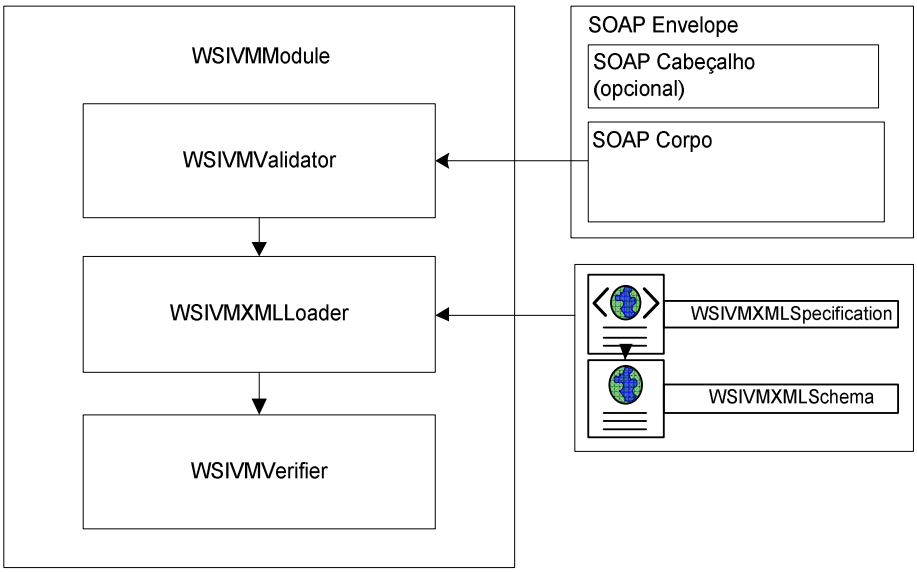

Figura 8. Funcionamento do WSIVM.

5. Desenvolvimento da Implementação

A implementação do modelo foi desenvolvida usando o servidor web Apache Tomcat e o framework Apache Axis2 para mensagens SOAP [Apache, 2010] (Figura 9). O Apache Axis2 foi escolhido para a implementação deste trabalho por sua capacidade de extensão através de módulos e a facilidade de interceptar mensagens SOAP através dos módulos.

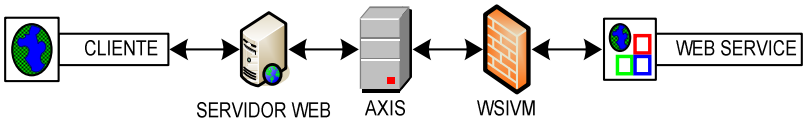

Figura 9. WSIVM.

Para a implementação do módulo de validação para o Apache Axis2 foi utilizado o módulo Rampart, que é o modo de extensão do Apache Axis2.

A fase de interceptação pode ser especificada no arquivo `Module.xml`. Foi escolhido fazer a interceptação da mensagem na fase de `PreDispatch`, que é a fase imediatamente anterior ao envio da mensagem para processamento no serviço web.

O funcionamento da implementação ocorre da seguinte forma: o cliente que pode ser uma aplicação, uma página web ou qualquer mecanismo capaz de se comunicar com um serviço web envia uma mensagem ao serviço web. Esta mensagem passa pelo servidor web que no caso utilizado é o Apache Tomcat. O servidor Web transmite essa mensagem SOAP para o Apache Axis. O Apache Axis envia a mensagem para ser analisada pelo WSIVM. Após a mensagem ser analisada, caso seja considerada inválida, é enviado um erro ao cliente pelo WSIVM. Caso seja considerada válida, é

transmitida para ser processada normalmente pelo serviço web retornando o resultado ao cliente (Figura 9).

Segue a descrição detalhada dos componentes do modelo WSIVM: WSIVMXMLSchema, WSIVMXMLSpecification e WSIVM Rampart module.

O **WSIVMXMLSchema** é a especificação do esquema de validação das entradas. Define o formato da especificação XML bem como os atributos válidos.

O **WSIVMXMLSpecification** é a especificação da validação das entradas. Especifica os parâmetros válidos de acordo com um conjunto de atributos pré-definidos. É utilizada para a validação das entradas do usuário. Dentre os parâmetros possíveis nas entradas estão:

- **OperationName:** nome da operação ou função exposta do serviço web a que se refere a validação
- **SanityzeOperation:** define se os parâmetros desta operação ou função podem ser reformulados se preciso para a remoção de caracteres não aceitos
- **ParamName:** nome do parâmetro ou campo a que se refere a validação
- **Allowed:** tipo de campo permitido, sendo válidos (text, html, html+java-script, email, number e all)
- **Length:** especifica o tamanho exato do campo
- **Maxsize:** especifica o tamanho máximo do campo
- **Minsize:** especifica o tamanho mínimo do campo
- **Nillable:** determina se é possível que o campo seja nulo ou não (true ou false)
- **regEx:** permite especificar uma expressão regular para validação

O **WSIVM Rampart module** é o principal componente do mecanismo implementado. Trata-se de um módulo do Apache Axis 2 que recebe os dados do cliente e faz a validação de acordo com a especificação XML chamando classes Java para realizar as validações. Este módulo é composto por um arquivo `wsivm.mar` que possui os seguintes componentes compactados:

- module.xml: contém a descrição do módulo, a classe que fará a validação e a fase em que a validação ocorrerá;
- MANIFEST.MF: arquivo de manifesto Java; e
- classes Java referentes à validação de entradas: WSIVMModule, WSIVMValidator, WSIVMXMLLoader e WSIVMVerifier.

6. Estudo de Caso

Como estudo de caso foi desenvolvido um sistema hipotético de cadastro de alunos para uma universidade denominado GerenciadorUniversidade. O sistema é composto por uma aplicação cliente denominada ClienteGerenciador e um serviço web servidor denominado Gerenciador.

Para o desenvolvimento do serviço web foi utilizada a linguagem Java e os testes de desempenho foram realizados utilizando o programa soapUI [Soapui, 2010].

Para o desenvolvimento do serviço web GerenciadorUniversidade criou-se uma classe com as operações `buscaAluno` e `cadastraAluno` e uma classe para lidar com as operações do banco de dados. Este serviço web foi desenvolvido sem nenhuma validação de entradas nas operações das classes Java propositalmente, deixando para o WSIVM a validação das mesmas.

A operação `buscaAluno` recebe um número de matrícula (Id) que deve ser um número inteiro maior que zero e com valor máximo de 10000 e retorna o cadastro do aluno contendo uma *String* com suas informações. A operação `cadastraAluno` recebe as informações do aluno que não devem conter código HTML ou Javascript e cadastra no banco de dados MYSQL. No banco de dados foi criada uma tabela `alunos` com os campos id (identificador auto-incremental), nome, idade, email, comentario, site e dataniver (data de aniversário).

Após a criação do serviço web e do banco, foi criada uma classe Java denominada `testaGerenciador` para testá-lo localmente.

Para o funcionamento do WSIVM foi especificado o WSIVMXMLSpecification – GerenciadorUniversidade (Figura 10), onde estão descritos os parâmetros para validação das entradas.

```
<?xml version="1.0" encoding="UTF-8"?>
<valid_inputs_specification xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
WebServiceID="GerenciadorUniversidade" xsi:noNamespaceSchemaLocation="valid_inputs_specification.xsd">
        <operation name="cadastraAluno">
        <input name="nome" type="String" min="5" max="20" accept="text" sanityze="true"/>
        <input name="idade" type="Integer" min="0" max="150" accept="number" sanityze="true"/>
        <input name="email" type="String" min="0" max="200" accept="email" sanityze="true"/>
        <input name="comentario" type="String" min="0" max="200" accept="text" sanityze="true"/>
        <input name="site" type="String" min="0" max="300" accept="url" sanityze="true"/>
        <input name="data" type="String" min="0" max="200" accept="regex" regexpattern= "(\\d{4})-(\\d{2})-
(\\d{2})" sanityze="true"/>
        </operation>
        <operation name="buscaAluno">
        <input name="id" type="Integer" min="0" max="10000" accept="number" sanityze="true"/>
        </operation>
</valid_inputs_specification>
```

Figura 10. WSIVMXMLSpecification – GerenciadorUniversidade.

A WSIVMXMLSpecification – GerenciadorUniversidade foi especificada de acordo com o WSIVMXMLSchema padrão do modelo e ambos os arquivos foram colocados na pasta C:\WSIVM do Windows.

Foi criado então o arquivo `Services.xml`, requerido para o Apache Axis 2 (Figura 11).

Foi gerado o pacote denominado `Gerenciador.aar` contendo a classe Gerenciador e MySQL e o descritor MANIFEST.MF e o arquivo Services.xml na pasta META-INF. Este pacote pode ser gerado como um pacote .jar, renomeado como .aar e colocado na pasta Services do Apache Axis2. Foi então copiado o arquivo do módulo (`wsivm.mar`) para dentro do diretório ..\Tomcat6.0\webapps\axis2\WEB-INF\modules.

Neste experimento foram realizados dois testes, um usando o modelo de validação de entradas WSIVM e outro não. O seguinte cenário foi configurado para a realização dos testes: 150 usuários são inicializados gradativamente seguindo a ordem

de 1 usuário sendo inicializado a cada 2 segundos. O teste é executado por 300 segundos que equivalem a 5 minutos. A base de dados é limpa para poder analisar o número de operações de cadastro de alunos efetuadas com sucesso.

```
<service>
        <parameter name="ServiceClass"
locked="false">exemplo.wsivm.universidade.Gerenciador</parameter>
        <operation name="cadastraAluno">
        <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
        <operation name="buscaAluno">
        <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
            <module ref="wsivm"/>
    <parameter
name="validationXML">file:///C:/WSIVM/valid_inputs_specification.xml</param
eter>
</service>
```

Figura 11. Services.xml – GerenciadorUniversidade.

O soapUI oferece uma interface amigável para a realização dos testes. Os testes são realizados fazendo chamadas diretas ao serviço web. A mensagem SOAP é enviada conforme o exemplo da Figura 12.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:univ="http://universidade.wsivm.exemplo">
   <soap:Header/>
   <soap:Body>
      <univ:cadastraAluno>
         <univ:nome>John</univ:nome>
         <univ:idade>12</univ:idade>
         <univ:email>john@hsj.com</univ:email>
         <univ:comentario>Passou</univ:comentario>
         <univ:site>http://www.gol.com</univ:site>
         <univ:dataniver>1980-09-12</univ:dataniver>
      </univ:cadastraAluno>
   </soap:Body>
</soap:Envelope>
```

Figura 12. Exemplo de mensagem SOAP enviada pelo soapUI.

A Figura 13 apresenta o gráfico de resultados com os tempos de resposta dos testes com e sem validação de entradas. No eixo X é mostrado o tempo percorrido do teste e no eixo Y é mostrado o valor em milissegundos do tempo de resposta.

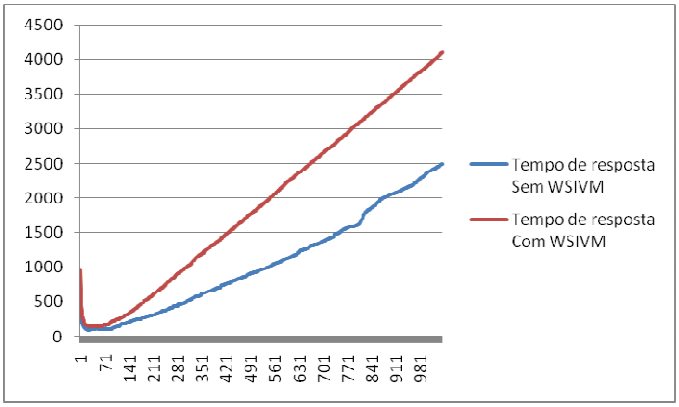

Figura 13. Tempos de resposta com e sem o uso do WSIVM.

A Figura 14 apresenta o gráfico de resultados com o *throughput* dos testes com e sem a validação de entradas do WSIVM. No eixo X é mostrado o tempo percorrido do teste e no eixo Y é mostrado o número de bytes por segundo (B/s).

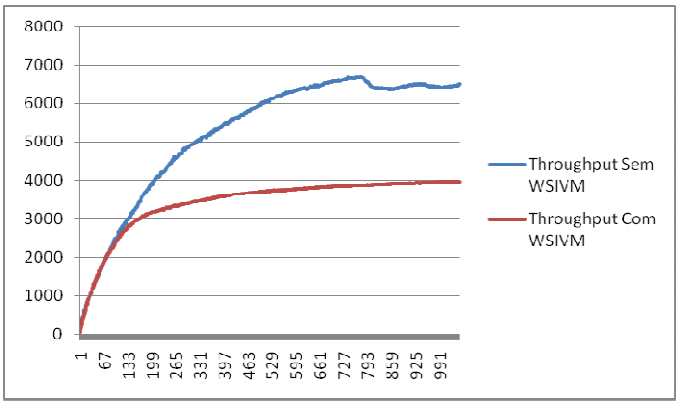

Figura 14. Throughput com e sem validação WSIVM.

Pode-se observar que a taxa de transferência de bytes por segundo (B/s) ou *throughput* cai bastante com o uso do WSIVM.

Tabela 1. Resultados consolidados do Estudo de caso.

Comparação	Tempo Min	Tempo Max	Tempo Médio	Bytes Transferidos	Bytes por segundo (*throughput*)	Inserções no Banco de dados
Sem WSIVM	35 ms	27848 ms	2494,85 ms	1974195 B	6506 B/s	10078
Com WSIVM	64 ms	13346 ms	4541,24 ms	1236330 B	4012 B/s	5134
Total	83%	-52%	82%	-37%	-38%	-49%

Na tabela 1, os cálculos que aparecem na linha Total em cada uma das colunas foram feitos da seguinte forma: diminui-se o resultado "Com WSIVM" do "Sem WSIVM" da coluna e este valor é dividido pelo valor "Sem WSIVM" da coluna.

Pode-se observar um aumento significativo na média dos tempos de resposta com o uso do WSIVM, com um aumento de 82%. O *throughput* total diminuiu em 38%. E o resultado do teste que é o cadastramento de alunos no banco de dados diminuiu em 49%, de 10078 alunos cadastrados para 5134.

O tempo gasto no processamento das mensagens XML teve impacto no desempenho das transações que são validadas uma a uma e comparadas com as regras especificadas de entradas consideradas válidas. A interpretação das mensagens é uma tarefa custosa que demanda grande quantidade de processamento e de memória para que a validação seja feita antes do processamento pelo serviço web e o retorno da resposta.

Também houve uma diminuição no número total de bytes transferidos, pois, devido ao maior tempo de processamento da mensagem, houve um menor número de mensagens processadas e conseqüentemente um menor número de respostas.

No caso apresentado, devido ao tempo gasto para a validação de cada mensagem, a aplicação pôde processar menos mensagens no mesmo período de tempo, resultando num menor número de inserções de alunos no banco de dados.

Uma diminuição no desempenho era esperada devido ao tempo para interpretar e percorrer as árvores XML para validação dos campos o que costuma ser custoso do ponto de vista de processamento. No entanto, a validação dos campos não permitindo que dados inválidos fossem inseridos pode compensar a perda de desempenho. Esta perda de desempenho poderá ser abordada em trabalhos futuros: testes utilizando outros mecanismos de interpretação de arquivos XML poderão ser efetuados, bem como o uso de um serviço web que demande mais processamento, comprovando o ganho com menor desperdício de processamento com mensagens inválidas. Mesmo assim, a proteção dos serviços obtida com o uso do modelo é uma vantagem que deve ser considerada.

Nos testes feitos nenhuma entrada indevida foi processada já que o ambiente estava devidamente configurado para filtrar as entradas inválidas.

7. Conclusões

Como as entradas de dados não validadas é o maior desafio para qualquer equipe de desenvolvimento de aplicações no ambiente Web e é a origem dos problemas de segurança de muitas aplicações [Owasp, 2010] [Owasp, 2011], um mecanismo reusável e independente para validação da entrada de dados, como o WSIMV proposto neste trabalho, é uma contribuição importante para a segurança de serviços web.

O WSIVM tem seu foco na validação das entradas de dados, permitindo a aceitação apenas de entradas válidas, já que é baseado na abordagem de lista branca, na qual apenas valores pré-definidos são aceitos e os demais são considerados inválidos.

Este modelo é particularmente interessante para o caso de serviços web que demandem grande quantidade de processamento das entradas de dados, pois, garantindo-se que apenas entradas válidas sejam aceitas, evita-se o desperdício de processamento pela aplicação.

Fazer a validação de entradas através do modelo apresentado é uma solução para aplicações legadas que não foram concebidas com validação de entradas de dados, pois se fazendo a validação nos pontos de entrada do serviço web, diminui-se a necessidade de fazer um número maior de alterações na aplicação já existente, reduzindo custos de desenvolvimento.

Além disso, conforme [Tsipenyuk et al., 2005] a abordagem de lista branca é mais confiável que a de lista negra. Na abordagem de lista negra todos os valores são considerados válidos menos os explicitamente especificados. Este tipo de abordagem tem alguns problemas, por exemplo, caso seja desejado validar um campo para que não contenha código HTML e seja criada uma lista negra com base na versão atual do HTML, caso surjam novas versões, esta lista pode deixar de ser considerada válida.

A abordagem de lista branca utilizada no WSIVM resulta na diminuição de falsos-positivos e num meio mais confiável de validação de entradas de dados. Já o trabalho de [Lin e Chen, 2009] possui como desvantagem a grande obtenção de falsos-positivos, isto é, o validador pode falhar considerando uma mensagem inválida, no entanto, a mensagem é válida.

Este trabalho buscou uma solução para impedir ataques de injeção de dados em serviços web, fornecendo proteção com um mecanismo reusável, que economiza o

processamento de chamadas maliciosas e é capaz de fornecer a validação de entradas de dados independentemente da implementação do serviço web que faz seu uso.

No estudo de caso foi possível observar que o aperfeiçoamento de segurança teve impacto negativo sobre o desempenho do serviço web implementado, o que é algo bastante comum nas pesquisas em segurança. Entretanto, a validação das entradas diminui a possibilidade de inserção de dados inválidos e, portanto, impede ataques ou a parada da execução correta do serviço web, compensando a queda do desempenho.

Existem diferentes aspectos que podem ser abordados em trabalhos futuros: (a) otimizar a implementação para melhorar o desempenho do modelo proposto (b) desenvolver um gerador semi-automático de especificações de segurança a partir do WSDL; (c) verificar mensagens SOAP e caminhos no formato XPath; (d) usar inteligência artificial ou um sistema de detecção de anomalias; e (e) fazer um esquema de retroalimentação do filtro de validação de entradas inválidas.

Referências

Apache. (2010) "Welcome to Apache Axis2/Java". Disponível em: http://axis.apache.org/axis2/java/core/

Bebawy, R. et al. (2005) "Nedgty: Web services firewall", In: Proc. of the IEEE International Conference on Web Services - ICWS, Orlando, IEEE, p.597-601.

Belapurkar, A. et al. (2009), Distributed Systems Security Issues, Processes and Solutions, Hoboken, NJ: John Wiley e Sons.

Bertino, E. et al. (2009), Security for Web Services and Service-Oriented Architectures, Springer-Verlag New York Inc.

Blyth, A. (2009) "An Architecture for an XML Enabled Firewall. International Journal of Network Security", v. 8, n. 1, p. 31-36, ISSN 1816-3548.

Cenzic. (2009) "Web Application Security Trends Report". Disponível em: <http://www.cenzic.com/downloads/Cenzic_AppSecTrends_Q1-Q2-2009.pdf>.

Cep. (2011) "CEPWebService". Disponível em: http://www.i-stream.com.br/webservices/cep.asmx

Clarke, J. (2009), SQL Injection Attacks and Defense. Syngress Media Inc.

Databreaches. (2009) "Top 10 Worst Data Losses or Breaches, updated". Disponível em: <http://www.databreaches.net/?p=7691>.

Feiman, Joseph. (2010) "Security in the SOA World: Methodologies and Practices", Enterprise Integration Summit, April 13-14, Sao Paulo, Brazil. Disponível em: http://www.gartner.com.br/tecnologias_empresariais/pdfs/brl37l_a4.pdf

Jensen, M.; Gruschka, N.; Herkenhöner, R. (2009). A survey of attacks on web services. Computer Science-Research and Development, v. 24, n. 4, p. 185-197.

Kieyzun, A. et al. (2009) "Automatic creation of SQL Injection and cross-site scripting attacks", In: *Proceedings of the 31st International Conference on Software Engineering* (ICSE '09). IEEE Computer Society, Washington, DC, USA, 199-209.

Lakshminarayanan, Sitaraman. (2010) "Interoperable Security Standards for Web Services", IT Professional, vol. 12, no. 5, pp. 42-47, Sep./Oct.

Lin, J.; Chen, J. (2009) "An Automated Mechanism for Secure Input Handling", Journal of Computers, v. 4, n. 9, pp. 837-844, Academy Publisher.

Loh, Y.; Yau, W.; Wong, C.; Ho, W. (2006) "Design and Implementation of an XML Firewall", In: Proceedings of the *2006 International Conference on Computational Intelligence and Security*, vol. 2, pp.1147-1150, 3-6 Nov.

Microsoft. (2010) "Microsoft Anti-Cross Site Scripting Library V 4.0". Disponível em: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=f4cd231b-7e06-445b-bec7-343e5884e651.

Nordbotten, N.A. (2009) "XML and Web Services Security Standards", *Communications Surveys & Tutorials, IEEE*, vol.11, no.3, pp.4-21.

Owasp. (2010) "OWASP Top 10 Web Application Security Risks". Disponível em: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Owasp. (2011) "OWASP Code Review Guide. Codereview-Input Validation". Disponível em: http://www.owasp.org/index.php/Codereview-Input_Validation

Saleem, M.Q.; Jaafar, J.; Hassan, M.F. (2010) "Model driven security frameworks for addressing security problems of Service Oriented Architecture", In: Proc. of the *2010 Int. Symp. Information Technology (ITSim),* vol.3, pp.1341-1346, 15-17 June.

Sans. (2009) "Os Maiores Riscos de Segurança Cibernética". Disponível em: http://www.sans.org/top-cyber-security-risks/pdfs/portuguese.pdf

Scholte, T.; Balzarotti , D.; Kirda, E. (2011) "Quo Vadis? A Study of the Evolution of Input Validation Vulnerabilities in Web Applications", In: Proc. of the Int. Conference on Financial Cryptography and Data Security '11, St. Lucia.

Shostack, A.; Stewart, A. (2008), The new school of information security. Boston: Addison-Wesley.

Sidharth, N.; Liu, J. (2007) "A Framework for Enhancing Web Services Security", In: Proc. of the *Computer Software and Applications Conference, 2007, COMPSAC 2007, 31st Annual International*, vol.1, no., pp.23-30, 24-27 July.

Siorg. (2009) "Sistema de Informações Organizacionais do Governo Federal – SIORG – Descrição do Web Service SIORG versão 2.0", Outubro. Disponível em: http://catalogo.governoeletronico.gov.br/pasta_servico/web-service-do-siorg

Soapui. (2010) "The Web Service, SOA and SOAP Testing Tool – soapUI". Disponível em: http://www.soapui.org/

Ssa. (2007) "WS-Security Wrapper", Sosnoski Software Associates Ltd. Disponível em: http://wsswrapper.sourceforge.net/

Sun, L.; Li, Y. (2008) "XML and web services security", In: Proceedings of the *12th International Conference on Computer Supported Cooperative Work in Design*, *CSCWD 2008*, pp.765-770, 16-18 April.

Tsipenyuk, K.; Chess, B.; McGraw, G. (2005) "Seven pernicious kingdoms: a taxonomy of software security errors", Security & Privacy, IEEE, v. 3, n. 6, p. 81-84.

Viega, J.; Epstein, J. (2006) "Why Applying Standards to Web Services Is Not Enough", IEEE security & privacy, New York, NY, v. 4, n. 4, p. 25-31.