UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rafael Bosse Brinhosa

WSIVM: MODELO DE VALIDAÇÃO DE ENTRADAS DE DADOS PARA WEB SERVICES

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Carla Merkle Westphall. Drª

Florianópolis
2010

Rafael Bosse Brinhosa

WSIVM: MODELO DE VALIDAÇÃO DE ENTRADAS DE DADOS PARA WEB SERVICES

Florianópolis – SC
2010

Catalogação na fonte pela Biblioteca Universitária da
Universidade Federal de Santa Catarina

B858W Brinhosa, Rafael Bosse
WSIVM modelo de validação de entradas de dados para Web Services [dissertação] / Rafael Bosse Brinhosa ; orientadora, Carla Merkle Westphall. - Florianópolis, SC, 2010.
97 p.: grafs., tabs.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Informática. 2. Ciência da computação. 3. Serviços da Web. 4. Segurança. 5. Programas de computador – Validação. I. Westphall, Carla Merkle. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 681

Rafael Bosse Brinhosa

WSIVM: MODELO DE VALIDAÇÃO DE ENTRADAS DE DADOS PARA WEB SERVICES

Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Ciência da Computação Área de Concentração de Sistemas de Computação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Florianópolis, 26 de fevereiro de 2010.

_____
Prof. Mario Antonio Ribeiro Dantas, Dr.
Coordenador do Programa de Pós-Graduação em Ciência da Computação

Banca Examinadora:

_____
Profª. Carla Merkle Westphall, Drª.
Orientadora

_____
Prof. Carlos Becker Westphall, Dr.

_____
Prof. Vitório Bruno Mazzola, Dr.

_____
Prof. Rômulo Silva de Oliveira, Dr.

A todos aqueles que
amo.

AGRADECIMENTOS

Agradeço primeiramente a Deus que me capacitou a chegar até aqui e que me ajudou nas horas difíceis desta caminhada.

Agradeço a minha professora orientadora Dr. Carla Merkle Westphall e ao professor Dr. Carlos Becker Westphall pela confiança, disposição e ajuda durante o período em que estive desenvolvendo meus estudos de mestrado. Também pelos seus esforços para que este objetivo foste atingido, como reuniões no seu período de férias.

Também não posso deixar de agradecer aos meus amigos que sempre estiveram presentes.

Aos meus familiares que sempre me apoiaram e comemoraram comigo as minhas conquistas.

Aos meus pais, que me ensinaram a perseguir meus sonhos e a batalhar para atingi-los e fizeram todo o possível para que eu tivesse uma boa educação capaz de levar-me a novos horizontes e a alcançar meus sonhos.

E a minha noiva Débora, que esteve em todos os momentos ao meu lado, me dando força e me incentivando a prosseguir rumo ao alvo.

Louvai ao SENHOR, porque ele é bom; porque a sua benignidade dura para sempre.

Salmos 136:1

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DOM	Document Object Model
DOS	Denial of Service (Negação de Serviço)
DTD	Document Type Definition
HTML	Hyper-Text Markup Language
HTTP	Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)
HTTPS	Secure Hypertext Transfer Protocol (Protocolo Seguro de Transferência de Hipertexto)
IP	Internet Protocol (Protocolo de Internet)
QOS	Quality of service (Qualidade de serviço)
UDDI	Universal Description, Discovery and Integration
UFSC	Universidade Federal de Santa Catarina
XML	Extensible Markup language
WSIVM	Web Services Input Validation Model

RESUMO

O uso da Arquitetura SOA baseado principalmente na utilização de Web Services está em crescimento constante, porém, devido às dificuldades encontradas quanto aos aspectos de Segurança, dentre outros fatores, este crescimento têm sido menor do que o esperado quando surgiu. O uso de Web Services herdou muitos problemas de segurança conhecidos em Aplicações Web e trouxe outros novos, sendo que a má validação de entradas de dados pode ser considerada a causa da maioria dos ataques bem sucedidos ocorridos nestes ambientes. Em SOA, com a necessidade de confiança em dados de terceiros para a integração e reutilização de serviços, a validação de entradas de dados tornou-se ainda mais importante.

Este trabalho demonstra um modelo para validação de entradas de dados para Web Services 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. Ainda apresenta um estudo de caso de seu uso demonstrando a sua eficácia e desempenho.

Palavras-chaves: Web Services; Segurança; Validação de entradas de dados

ABSTRACT

The SOA architecture based primarily on the Web Service technology is having a steady growth, 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 problems of Web Applications Security and brought new others. The majority of the attacks today are consequences of bad input validation in the Application-level. This thesis presents how to implement an input validation model for Web Services. The presented input validation model is composed of a XML Schema, a XML Specification and a module for doing the input validation. Therefore, is presented a case of study showing the effectiveness and performance of this mechanism.

1 INTRODUÇÃO

Durante anos a heterogeneidade do meio computacional foi um grande problema para a computação. O uso de diversos padrões e diferentes linguagens de programação impossibilitava as aplicações de interagir de forma simples. Quando estritamente necessário, fazia-se o uso de padrões complexos e muitas vezes proprietários para esta troca de informações entre sistemas. Estas tecnologias, por sua natureza, sempre dependiam em alguma forma de sua implementação, como no caso do uso de CORBA, DCOM e RMI (GISOLFI, 2001).

Até recentemente desenvolvedores podiam garantir que seus ambientes eram homogêneos, seguros, confiáveis e de gerenciamento centralizado. Entretanto, com o advento de diferentes tecnologias de colaboração e compartilhamento de informações, novos modos de interações estão evoluindo. Esta pressão evolucionária gerou 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, CHAKRABARTI *et al.*, 2009). Esta facilidade na troca de informações foi responsável por novos problemas a serem pesquisados pela comunidade científica como a Segurança em Web Services.

O conceito de Web Services, fundamentado por padrões abertos, facilitou a interoperabilidade entre aplicações trazendo uma verdadeira revolução. Em “O mundo é plano” Friedman (FRIEDMAN, 2005) destaca o uso de Web Services como uma das forças que “achataram” o mundo, ou seja, que auxiliaram no desenvolvimento de uma plataforma global para uma força de trabalho distribuída de pessoas e computadores capaz de fazer diferentes aplicações se comunicarem e trocar dados e informações.

Com o uso de ferramentas apropriadas, desenvolver um Web Service ou transformar uma aplicação já existente em um Web Service é relativamente fácil. No entanto, a mesma facilidade não é encontrada quando requisitos como segurança são requeridos.

Para a implementação de segurança em Web Services, diversos padrões e especificações foram criados. No entanto, apenas o uso dos padrões corretos não garante que a segurança necessária seja obtida (VIEGA e EPSTEIN, 2006).

Web Services são considerados um dos principais componentes da Arquitetura Orientada a Serviços (AOS ou SOA - Services Oriented Architecture) que fazem uso de sua principal característica inerente, a reusabilidade, como base para o fornecimento de serviços.

Enquanto SOA apresenta vários benefícios como a diminuição de custos com a eliminação de esforços redundantes, segurança é uma das principais preocupações. Segundo informações de uma pesquisa global encomendada pela CA (Ca, 2010), 43% dos executivos de TI classificam as ameaças à segurança como o item mais crítico na implementação de SOA e de Web Services (BALIEIRO, 2008).

De acordo com o instituto Gartner (GARTNER, 2010), SOA foi usado de alguma forma em mais de 50% grandes, novas aplicações e processos de negócio desenvolvidos em 2007. Em 2010 espera-se que mais de 80% dos grandes novos sistemas usem SOA em algum aspecto do desenvolvimento (ABRAMS e SCHULTE, 2008).

A capacidade e a facilidade dos sistemas interagirem com o advento da Internet aonde a troca de informações entre sistemas ou pessoas e sistemas de forma globalizada se tornou possível mudou a forma como os sistemas eram concebidos.

Antes da era da Internet, a validação de entrada de dados era menos necessária, o próprio ambiente limitava a possibilidade de ataques. Os sistemas eram utilizados por funcionários responsáveis ou pessoas de confiança que dificilmente entravam dados nos sistemas com intuitos maliciosos e quando faziam o máximo que conseguiam afetar eram os sistemas nos quais tinham acesso.

Com a Internet os dados passaram a ser inseridos nos meios colaborativos por qualquer pessoa que tenha acesso a determinado conteúdo, sem um prévio conhecimento do usuário. Desta forma, além da antiga confiança no usuário e em políticas de uso surgiu a necessidade de uma forte validação dos dados inseridos através da validação de entradas de dados.

Validação de entradas de dados, segundo a Microsoft, são controles que uma aplicação deve fazer nas suas entradas de dados nos aspectos léxicos, sintáticos e de checagem de tipos, integridade e origem (BERTINO, MARTINO *et al.*, 2009).

A falta deste tipo de controle 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.

Ataques em massa, visando lucro através do roubo de informações como senhas de banco e números de cartões de crédito têm se tornado freqüentes. Sistemas que possuem informações valiosas como a lógica do negócio e a possibilidade de obter dados relevantes são o principal alvo dos ataques.

Desta forma, com a adoção de SOA e Web Services por muitas companhias, o aumento de segurança com o uso de mecanismos mais robustos de validação de dados passou a ser vital para as organizações.

Este trabalho propõe um modelo para validação de entradas de dados em Web Services fornecendo proteção contra ataques baseados na entrada de dados maliciosos.

Como contribuições podem ser citadas: o modelo proposto WSIVM (Web Services Input Validation Model) que contém um mecanismo para validação, um esquema XML, uma Especificação XML e um módulo que faz a validação utilizando os mesmos. Ainda apresenta um estudo de caso de seu uso demonstrando a sua eficácia e desempenho.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é apresentar um modelo para validação de entradas de dados para Web Services denominado WSIVM. Este modelo deve ser eficaz contra ataques de manipulação de entradas de dados provendo um mecanismo reusável e de implementação pouco custosa. O uso deste modelo deverá tornar a entrada de dados para Web Services segura e ter baixo impacto no desempenho da aplicação bem como em sua implementação. Também deve tornar possível o monitoramento das entradas de dados indevidos no sistema possivelmente identificando usuários maliciosos. E economizar recursos do servidor impedindo que entradas inválidas sejam processadas através de validação prévia.

1.2 OBJETIVOS ESPECÍFICOS

Dentre os objetivos específicos deste trabalho podem ser citados:

- Desenvolver um módulo de acordo com o modelo WSIVM que forneça validação de entradas de dados.

- Implementar uma especificação XML e um XML Schema para especificação e validação de atributos de segurança do modelo.
- Facilitar o monitoramento de entradas de dados indevidas fazendo o registro no sistema.
- Desenvolver um estudo de caso composto por um Web Service utilizando o modelo WSIVM de validação de entradas e apresentar a mensuração de seu desempenho.

1.3 JUSTIFICATIVA E TRABALHOS CORRELATOS

A falta de validação de entradas de dados é a maior causa de ataques a aplicações Web segundo o Web Application Security Trends Report de 2009 elaborado pela Cenzic (CENZIC, 2009). Por terem naturezas semelhantes, existe uma tendência que seja a maior fonte de ataques à Web Services também no futuro, apesar de não terem sido encontradas pesquisas demonstrando esta realidade.

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, 2008) e o uso de soluções Open-source em PHP. No entanto, existem poucos mecanismos específicos para Web Services.

No que tange a mecanismos de implementação de segurança para Web Services não seguros o MIT (Massashusets Institute of Technology) possui uma implementação denominada WS-Security Wrapper ((Ssa), 2007) que trata de um intermediador entre o Web Service e o cliente fazendo a validação de certos aspectos. No entanto, este trabalho foi desenvolvido para possuir compatibilidade apenas com Web Services 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 Web Service seguro. O mecanismo proposto (WSIVM) ajuda nesta tarefa de forma diferenciada dos demais trabalhos propostos sobre este assunto. Primeiramente, porque foca principalmente no aspecto de tratamento de entrada de dados, diferentemente de trabalhos como o IAPF (Integrated Application and Protocol Framework) (SIDHARTH e LIU, 2007), trabalho que busca abordar todos os aspectos de segurança relacionados com Web Services. Além disso, outros trabalhos focam no uso de tecnologias existentes como a cifragem XML para garantir a segurança de Web Services como em (SUN e LI, 2008), mas não mencionam a validação de entradas de dados. E também existem trabalhos que focam na validação de mensagens SOAP, não validando, no entanto, o conteúdo dos dados que estão sendo recebidos.

Com relação aos aspectos de validação de entradas em Aplicações Web, existem alguns trabalhos como Lin e Chen (LIN e CHEN, 2009) que desenvolveram uma ferramenta que produz funções para validação de entradas dependendo do banco de dados e do

framework da Aplicação. A ferramenta insere automaticamente a validação de entradas no lado do servidor eliminando vulnerabilidades de inserções maliciosas. Já Hayati (HAYATI, 2008) propõe uma extensão da UML em um framework integrado para prover uma modelagem orientada a segurança. O framework busca resolver os problemas de segurança usando um nível mais alto de abstração, no design da aplicação e também provê mecanismos para a validação de entradas de dados.

Além dos trabalhos já relacionados existe ainda outra categoria de trabalhos, focados no desenvolvimento de *firewalls* como o Web Service Firewall Nedgty (BEBAWY, SSABRY *et al.*, 2005), que diferentemente deste trabalho é focado na proteção contra ataques de negação de serviço e de estouro de pilha. E os XML Firewalls como demonstrado por (BLYTH, 2009) que aborda validação da estrutura do conteúdo XML, mas não do conteúdo propriamente dito e o trabalho de (LOH, YAU *et al.*, 2006) que menciona a proteção contra SQL injection através do XML Schema e de uma lista negra pré-compilada de comandos SQL, essa abordagem costuma apresentar muitos falso-positivos, no entanto, detalhes sobre a eficácia deste trabalho com testes mais extensos não foram apresentados.

Pode-se observar que existe uma falta de trabalhos abordando a validação entradas especificamente para Web Services. Este trabalho visa propor um modelo de solução para este problema.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho é dividido nos seguintes capítulos, de forma a apresentar gradativamente os conceitos e a especificação da proposta:

Capítulo 1 – Introduz o tema do trabalho apresentando a proposta.

Capítulo 2 – Aborda o conceito de Web Services e suas principais tecnologias utilizadas na implementação da proposta.

Capítulo 3 – É feita uma análise de segurança em Web Services, nesse capítulo são apresentados novos riscos trazidos por esta tecnologia, principais ataques dando ênfase para o ataque de manipulação de entradas e os principais mecanismos para a segurança em Web Services.

Capítulo 4 – É descrita a proposta de criação do WSIVM com seus aspectos e características começando por um detalhamento em alto nível até como foi feita a implementação.

Capítulo 5 – É apresentado um estudo de caso completo que demonstra na prática o modelo proposto e o seu desempenho.

2 WEB SERVICES

Nesse capítulo serão abordados os principais conceitos e tecnologias utilizadas na construção de Web Services, esses aspectos são importantes para a posterior explicação da proposta.

Web Services podem ser considerados pedaços de aplicações separados por funções únicas (serviços) expostos na Web para utilização de determinado público. Sendo concebido com base em um conjunto de padrões abertos que garantem as suas principais características: interoperabilidade e reusabilidade.

Web Services foram desenvolvidos com base em tecnologias existentes na web como URI, HTTP, XML dentre outros, conforme mostrado na Figura 1 (BERTINO, MARTINO *et al.*, 2009).

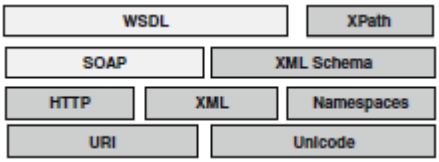

Figura 1 - Fundamentos da Arquitetura de Web Services (BERTINO, MARTINO *et al.*, 2009)

Seu uso tem revolucionado o jeito como as Aplicações se comunicam, sendo que funções ou serviços de Aplicações Web agora podem ser facilmente acessadas por Aplicações Locais que podem ter suas funções ou serviços acessados por um aparelho de celular, por exemplo.

Grandes companhias têm empenhado seus esforços em reduzir custos utilizando o conceito de Web Services e da Arquitetura SOA para pegar pedaços de suas aplicações que são utilizados por outras aplicações e transformá-los em Serviços que podem ser reutilizados. Estes serviços transformados em Web Services podem ser utilizados por diversas aplicações independentemente da linguagem.

As principais tecnologias que fundamentam Web Services serão explicadas brevemente.

2.1 XML

A tecnologia XML espalhou-se muito desde o seu surgimento, sendo concebida como uma linguagem para descrição de dados, o XML é utilizado como a base para o desenvolvimento de Web Services, sendo a tecnologia utilizada nas mensagens SOAP, no WSDL e nos serviços de diretório. Ele tornou possível que aplicações desenvolvidas por diferentes linguagens pudessem se comunicar sem problemas.

O XML é usado nos mais diversos tipos de transações e publicações de dados por ser uma forma simples de se escrever dados atribuindo significados (HUNTER, 2007).

Ele tem vantagens distintas de outras formas de representação de dados como:

- XML é fácil de ler e de ser entendido, seja por pessoas ou máquinas;
- Existe um grande número de plataformas que suportam XML e são capazes de ler, escrever e manipular dados no formato XML;

No entanto, existem algumas desvantagens de sua utilização como o grande tamanho dos arquivos por sua natureza descritiva e o alto consumo de processamento para a sua interpretação. Como forma de reduzir estas desvantagens diversos mecanismos tem sido desenvolvidos como interpretadores de XML otimizados e compactadores de mensagens.

2.2 SOAP

De acordo com a especificação atual do W3C (W3c), SOAP é um protocolo leve para a troca de informações em um ambiente descentralizado e distribuído. Em outras palavras, é uma forma padrão para a troca de informações usando XML.

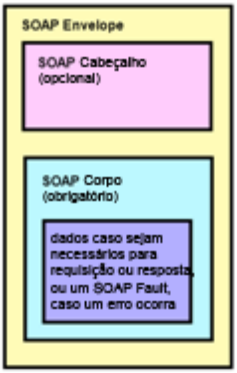

Figura 2 - Mensagem SOAP

As mensagens SOAP possuem um formato XML, contendo um elemento raiz chamado de Envelope, um elemento opcional Header que é o cabeçalho e um elemento obrigatório que é o Body ou corpo da mensagem. O elemento Body contém a informação principal (requisição, resposta ou dados de falha) e o elemento Header pode conter informações adicionais como instruções de roteamento e segurança.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

Figura 3 - Exemplo de uma mensagem SOAP (W3c)

2.3 WSDL

O WSDL foi primeiramente proposto em Março de 2001 em uma nota para o W3C enviada pela Ariba, IBM e Microsoft (HUNTER, 2007). A atual definição do WSDL é de um formato XML para descrever serviços de rede como um conjunto de pontos de operação em mensagens contendo informações orientadas a documentação ou orientadas a procedimentos (Figura 4).

As operações e mensagens são descritas abstratamente e então portadas para protocolo de rede e um formato de mensagem que definem um ponto de operação. Pontos de operação concretos são combinados em pontos de operação abstratos, ou serviços.

O WSDL é extensível, permitindo a descrição de seus pontos de operação e de suas mensagens independentemente dos formatos de mensagem ou protocolos de rede que são utilizados para a comunicação (W3c).

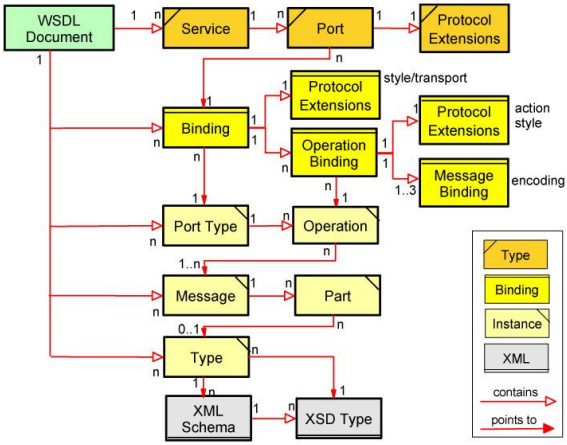

Figura 4 - Elementos de um documento WSDL e suas relações.

Os arquivos WSDL descrevem os serviços de baixo para cima, começando pelos tipos de dados e terminando com a localização do serviço e são divididos em: seção *types* define os tipos de dados usados, seção *message* define as mensagens individuais usando os identificadores da seção *portType*, seção *portType* define a interface e as operações de entrada, seção *binding* define o protocolo e o formato que são utilizados para fornecer o serviço e por último a seção *service* define

o serviço e o endereço aonde este serviço está disponível (veja o exemplo na Figura 5).

```
<?xml version=”1.0”?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"

xmlns:tns="http://example.com/stockquote.wsdl"

xmlns:xsd1="http://example.com/stockquote.xsd"

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema
targetNamespace="http://example.com/stockquote.xsd"

xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol"
type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price"
type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body"
element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
```

```
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
            <input
message="tns:GetLastTradePriceInput"/>
            <output
message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
            <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first
service</documentation>
        <port name="StockQuotePort"
binding="tns:StockQuoteBinding">
            <soap:address
location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Figura 5 - Exemplo de WSDL 1.1 (W3c)

2.4 DIRETÓRIOS

Quando surgiu o conceito de Web Services, o UDDI (Universal Description, Discovery and Integration) era a única

especificação conhecida para publicação e descoberta de serviços e existiam três grandes serviços de diretórios globais, IBM, Microsoft e SAP, conhecidos como UDDI Business Registry ou UBR. No entanto, este mecanismo global de publicação de serviços não funcionou e em janeiro de 2006 foi descartado. Cada fabricante possui seus próprios produtos para publicação, descoberta e chamadas de serviços, que podem ser ou não baseados na UDDI e são usados normalmente em ambiente local. Alguns sites também fornecem listas de Web Services para utilização tais como:

- http://www.wsindex.org/
- http://www.webservicelist.com/
- http://www.xmethods.com

A UDDI esta na sua versão 3.0 sendo atualmente suportada pelo OASIS (OASIS, 2010).

3 SEGURANÇA EM WEB SERVICES

Neste capítulo são abordados os principais conceitos e tecnologias utilizadas na segurança de Web Services.

O ambiente web representa um dos principais locais para a ação de usuários maliciosos, sendo que qualquer aplicação ou interface exposta neste ambiente corre um alto risco de sofrer ataques. Para evitar que estes ataques sejam bem sucedidos surgiram diversos mecanismos de segurança implementados para suprir diversas necessidades como: autenticação, autorização, integridade, confidencialidade, auditoria, disponibilidade e contabilização (JOSUTTIS, 2008). Com o surgimento dos Web Services e a propagação da arquitetura SOA, novos padrões foram criados e estão sendo amadurecidos com o propósito de protegê-los.

Podem-se observar serviços sendo publicados na Web: informações do clima, ações da bolsa de valores, cadastros de clientes, serviços de busca, compras, informações sobre pacientes e aplicações do governo. Estes serviços podem ser compostos fornecendo serviços de maior valor agregado, inclusive utilizando-se outras tecnologias como P2P, demonstrado em Birman (BIRMAN, CANTWELL *et al.*, 2009). Empresas também já começaram a publicar seus serviços como informações de cadeias de suprimento (ERP) e busca de produtos. Os Web Services tornaram-se um meio popular de divulgar dados que podem ser representados de diferentes formas através da interpretação de arquivos XML.

Na medida em que aplicações mais importantes estão sendo expostas através de Web Services seja na Internet ou localmente, maior tem se tornado a preocupação com a segurança dos mesmos. Neste sentido é importante que medidas pró-ativas sejam tomadas para que os Web Services sejam seguros desde sua implementação e não somente após ataques começarem a ser realizados com sucesso.

Devido à falta de conscientização e até mesmo de conhecimento dos desenvolvedores de aplicações Web com relação à implementação de segurança e validação de entradas, existe um legado de aplicações com falhas que vêm sendo exploradas dia a dia por usuários maliciosos e publicadas em sites dedicados a isto como: http://www.zone-h.org/. Usando scripts automatizados, times de usuários maliciosos exploram falhas de sites não protegidos como modo de fazer ataques de cunho político, espalhar *malwares* para roubar

senhas de banco ou publicidade do próprio grupo como forma de demonstração de poder. Conforme a Figura 6 obtida do relatório da Microsoft Security Intelligence Report (SIR) da segunda metade de 2009, as vulnerabilidades de aplicações são as mais comumente exploradas.

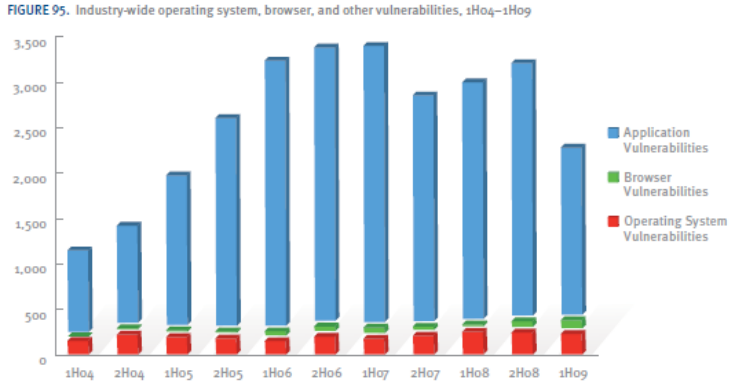

Figura 6 - Vulnerabilidades (MICROSOFT, 2009)

São poucas as brechas divulgadas em relação Web Services, mas isto pode se tornar uma realidade caso medidas de segurança não sejam tomadas para impedir que este se torne um dos principais alvos do futuro.

Outro fator importante muitas vezes menosprezado é a segurança de Web Services de uso interno nas companhias, mesmo que estes serviços não estejam expostos na web, existem grandes riscos que informações sigilosas venham a vazar caso não sejam implementados mecanismos de segurança adequados. Por exemplo, de acordo com especialistas da indústria, 70% dos ataques a bancos de dados são praticados por usuários internos das companhias (FYFFE, 2008).

As falhas presentes em aplicações web vêm crescendo em um ritmo impressionante, conforme Figura 7, e esta tendência deve se repetir em Web Services.

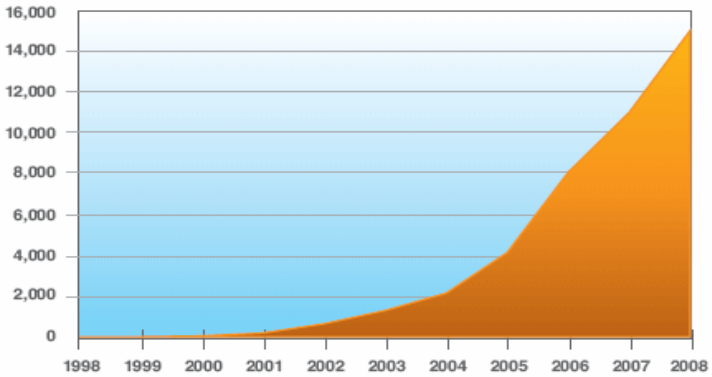

Figura 7 – Contagem acumulativa de falhas em aplicações web (Ibm, 2009).

3.1 NOVOS RISCOS DE SEGURANÇA COM O USO DE WEB SERVICES

Web Services expõe novos riscos à segurança das organizações porque antigos métodos de proteção como firewalls e anti-vírus não são capazes de protegê-los. Por serem desenvolvidos para fazerem uso do protocolo HTTP 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*.

O fato de um serviço ser utilizado por diversas aplicações possibilita que através do seu comprometimento todos os recursos sejam afetados de forma sistêmica.

Devido à falta de maturidade das novas tecnologias que estão sendo construídas para o consumo e provimento de Web Services falhas de segurança são encontradas e corrigidas com freqüência, sendo necessário que atualizações sejam feitas para evitar ataques. Exemplos de ataques relacionados a tecnologias podem ser encontrados em “A survey of attacks on web services” (JENSEN, GRUSCHKA *et al.*, 2009).

3.2 PRINCIPAIS ATAQUES À WEB SERVICES

Apesar de existirem diversas listas de falhas de segurança relacionadas à Web Services, quase todas possuem os mesmos itens relacionados, mudando apenas a taxonomia utilizada de uma para a outra. Alguns dos principais ataques encontrados:

- **Alteração de Mensagem:** uma entidade do XML schema pode buscar alterar o conteúdo de uma mensagem comprometendo a integridade da mesma;
- **Confidencialidade:** entidades não-autorizadas podem buscar ganhar acesso a informações confidenciais dentro da mensagem;
- **Homem do meio (Man-in-the-middle):** um atacante pode agir como um intermediário SOAP legítimo buscando interceptar as mensagens trocadas entre dois ou mais participantes. Os participantes da mensagem pensam estarem recebendo mensagens corretas quando na verdade estas foram alteradas ou até originadas pelo atacante;
- **Falsificação de Identidade:** acesso não-autorizado usando ataques de autenticação e intromissão.
- **Falhas de conteúdo:** falhas explorando o conteúdo dos elementos do corpo do XML;
- **Ataque de negação de serviço (DoS):** explora-se a capacidade do serviço visando a sua indisponibilidade. Isto pode ser feito, por exemplo, enviando num número maior de mensagens SOAP do que o servidor possa suportar.
- **Envenenamento do Schema:** manipulação do Web Service schema para alterar dados processados pela aplicação;
- **Alteração maliciosa dos Parâmetros XML:** inserção de scripts ilegítimos ou conteúdo em Parâmetros XML;
- **Coercive Parsing:** inserção de conteúdo ilegítimo no corpo do XML visando a execução ou erros durante o parsing;
- **Alteração de rotas XML:** redirecionar os dados endereçados por um caminho (path) XML.

Os ataques relacionados com manipulação de dados, que são o foco deste trabalho, serão melhor explicados no tópico 3.3.

3.3 ATAQUES DE MANIPULAÇÃO DE DADOS

Existem ataques diretamente relacionados com a manipulação de dados, seguem os mais comuns:

Cross-site Scripting ou XSS: Kieyzun (KIEYZUN, GUO *et al.*, 2009) classifica os ataques XSS em dois tipos:

Ataques XSS de primeira ordem, também conhecido com Tipo 1 ou refletido, a vulnerabilidade resulta da 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 Web Services.

Ataques XSS de segunda ordem, também conhecidos como Tipo 2 ou armazenado. 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 scripts executável.

Aplicado a Web Services é possível sofrer ataques ao apresentar dados não validados diretamente ao usuário. Por exemplo, fazendo-se uso de AJAX obter dados fornecidos por Web Services de terceiros que podem estar contaminados, sem previamente validá-los como neste caso:

```
document.write(xmlhttp.responseText);
```

Caso a resposta da chamada AJAX para um Web Service contenha dados HTML e JavaScript, estes dados serão interpretados e executados oferecendo risco ao usuário.

SQL injection: este tipo de ataque têm se tornado o mais comum dos ataques de injeção de dados nos últimos tempos, ferramentas automáticas foram criadas para explorá-los tornando possível a invasão de milhares de sites em um espaço curto de tempo.

Funciona através de entradas de dados maliciosos visando a execução de comandos SQL no banco de dados (CLARKE, 2009).

Em Web Services que não possuem o devido tratamento de exceção, a mensagem de erro pode conter dados valiosos para o atacante utilizar e através de tentativa e erro torna possível descobrir que tecnologia de banco de dados está sendo utilizada, que tabelas existem e podem ser exploradas e todas as informações necessárias para efetuar um ataque. Ataques de SQL injection podem levar a elevação de privilégios, sendo possível executar comandos em modo de Administrador no servidor comprometido. Geralmente encontram-se falhas de SQL injection através do uso de ferramentas de busca na web. É possível testar se um Web Service é vulnerável através do envio de requisições SOAP com os parâmetros devidamente manipulados, por exemplo, enviando “"1=1 –“ como parâmetro para um determinado serviço tendo como retorno:

```
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
```

Ou:

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

Blind SQL injection: trata-se de um tipo de SQL injection no qual os resultados não são exibidos para o atacante, este tipo de ataque é muito comumente utilizado contra Web Services, pois muitos servidores evitam que as mensagens de erro geradas pelo Serviço cheguem ao usuário. Em Web Services 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.

Por muito tempo Cross-Site Scripting era a falha mais comumente explorada até recentemente ser ultrapassada pelo ataque de SQL injection conforme apresentado na Figura 8. Este fato deve-se principalmente ao fato de vários ataques em massa atingindo milhões de aplicações web terem sido deflagrados.

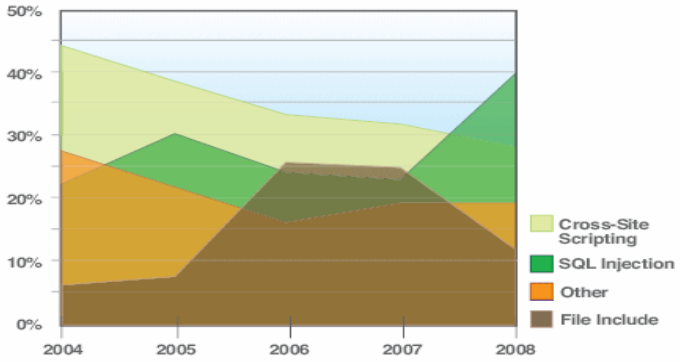

Figura 8 – Vulnerabilidades de aplicações web por técnica de ataque utilizada (IBM, 2009).

Ataques de manipulação de entrada foram desprezados por muito tempo devido aos seus perigos não serem facilmente identificado. No entanto, existe atualmente um maior esclarecimento sobre o assunto, que têm amplamente difundido através de iniciativas como o Open Web Application Security Project (OWASP, 2010) ou Web Application Security Consortium (WASC, 2010).

No entanto, certas empresas tiveram que aprender com seus erros, tendo sido alvo de ataques que resultaram na perda de milhões de dólares. Como por exemplo, no caso da Heartland, empresa que atua na área de pagamentos de cartão de crédito, que através de um ataque direcionado utilizando uma falha de SQL injection teve um malware introduzido em seus servidores internos que comprometeu o gateway e roubou o número de milhares de cartões de créditos, apesar de a empresa ser certificada PCI (VIJAYAN, 2009).

Apesar de ser difícil conseguir informações confiáveis sobre incidentes de segurança ou vazamentos de dados como relatado no livro "The New School of Information Security" de Shostak (SHOSTACK e STEWART, 2008), é possível observar que os maiores casos de ataques em que ocorreram roubos de dados tiveram relação com a injeção de dados maliciosos, conforme mostrado na Tabela 1.

Tabela 1 - 10 maiores perdas de dados ou brechas de segurança já encontradas (DATABREACHES.Net, 2009).

Ranking	Número de dados ou pessoas	Entidade	Data do Incidenet ou Publicação	Tipo de Incidente
1	130,000,000	Heartland Payment Systems	20/01/2009	Hack, Malware
2	94,000,000	TJX, Inc.	17/01/2007	Hack, Malware
3	90,000,000	TRW/Sears Roebuck	22/06/1984	Hack
4	70,000,000	National Archives and Records Administration	01/10/2009	Disposal
5	40,000,000	CardSystems Solutions	17/06/2005	Hack
6	30,000,000	Deutsche Telekom	01/11/2008	Exposure
7	26,500,000	U.S. Department of Veterans Affairs	22/05/2006	Laptop Roubado
8	25,000,000	HM Revenue and Customs / TNT	18/10/2007	Fitas Perdidas
9	18,000,000	Auction.co.kr	17/02/2008	Hack
9	18,000,000	National Personnel Records Center	12/07/1973	Fogo
10	17,000,000	Countrywide Financial	01/08/2008	Ataque Interno
10	17,000,000	T-Mobile	06/10/2008	Disco roubado ou perdido

Em Web Services, os ataques de manipulação de entradas de dados podem se alastrar de forma a atingir diversos sistemas, haja vista, um mesmo serviço pode ter suas informações publicadas em diversas aplicações.

3.4 IMPLEMENTAÇÃO DE SEGURANÇA

Para a implementação de segurança em Web Services existem diversas especificações e métodos, que possuem implementações nas linguagens de programação mais utilizadas e podem ser classificados genericamente como baseadas no nível de transporte com o uso de SSL ou TSL ou baseadas no nível de mensagem com o uso de especificações WS-* como as mostradas na Figura 9. Existem também abordagens usando uma combinação de ambas as implementações ou estratégias diferenciadas, porém, estas duas abordagens são as mais utilizadas.

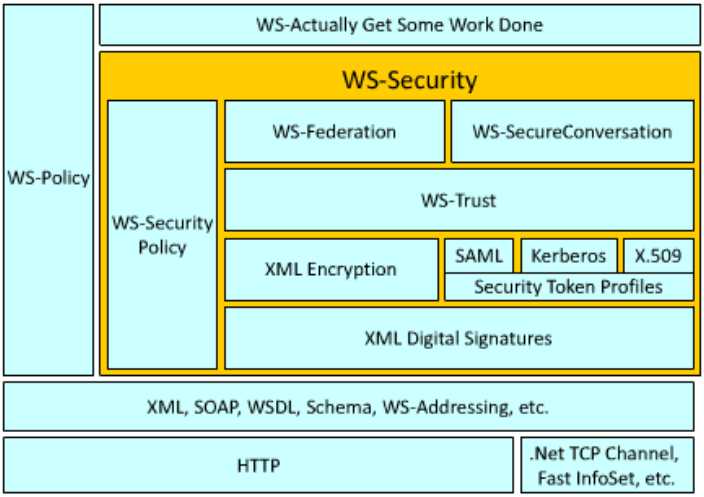

Figura 9 - Especificações de segurança para Web Services

3.4.1 Segurança em Web Services baseada no nível de Transporte

Serve para utilização de segurança ponto a ponto, garantindo confidencialidade e integridade (MATTSSON e GREEN, 2008). A implementação pode ser feita usando protocolos criptográficos como SSL v3 (FREIER, KARLTON *et al.*, 1996) ou TSL (DIERKS e RESCORLA, 2006).

A segurança apenas no nível de transporte não garante controle de acesso sendo que deve ser usada em conjunto com outras especificações para a garantia de segurança do serviço. Também a aplicação cliente enviando a mensagem não pode garantir que outros *proxies* ou roteadores irão decifrar a mensagem antes de enviar para o serviço destinatário. A menos que todos os pontos pelos quais a mensagem transita sejam seguros, a mensagem pode acabar sendo aberta no meio do caminho (BUSTAMANTE, 2008).

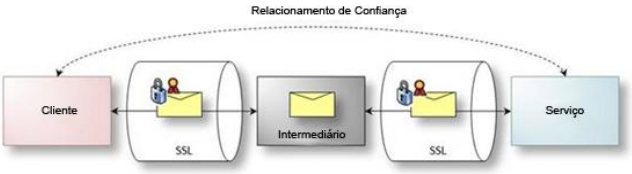

Figura 10 - Segurança no Transporte protege as mensagens ponto a ponto. Baseada em Bustamante (BUSTAMANTE, 2008).

3.4.2 Segurança em Web Services baseada no nível das Mensagens

É um conjunto de especificações para a implementação de segurança no nível das mensagens de modo fim a fim. É o mais completo meio utilizado para a garantia de segurança do serviço, no entanto, possui algumas desvantagens como a sobrecarga das mensagens trocadas com o uso das especificações necessárias o que impacta diretamente no desempenho.

Figura 11 - Segurança na Mensagem protege as mensagens fim a fim. Baseada em Bustamante (BUSTAMANTE, 2008).

WS-Security: esta especificação define os mecanismos básicos para a troca de mensagens seguras propondo um conjunto de extensões ao SOAP que podem ser usados para construir Web Services seguros agregando garantia de integridade de conteúdo e confidencialidade às mensagens.

Possui três mecanismos principais: capacidade de enviar *tokens* seguros como parte da mensagem, integridade da mensagem e confidencialidade da mensagem.

É uma especificação flexível, tornando possível usar diferentes mecanismos de segurança como PKI, Kerberos e SSL (OASIS, 2004).

Utiliza o cabeçalho das mensagens SOAP para incluir as funções de segurança, utilizando a camada de aplicação.

WS-Policy Framework e WS-Policy: define um arcabouço e um modelo para a definição de políticas que se referem a capacidades específicas do domínio, requisitos e características gerais das entidades de um sistema baseado em Web Services (W3c, 2007). Permite a especificação de políticas como Qualidade de Serviço e segurança usando XML, também é utilizado para a definição de requisitos pelos consumidores do serviço que o Web Service disponibiliza.

Entre os principais componentes estão:

- Asserções: podem ser requisitos de um Web Service ou os anúncios das políticas do mesmo.
- Operadores: são utilizadas para definir possíveis combinações de políticas. Por exemplo ***wsp:ExactlyOne*** define que apenas um nodo filho precisa ser satisfeito.
- Intersecção de Políticas: é uma característica opcional que pode ser usada para definir a junção de características comuns entre duas Políticas Alternativas.

WS-PolicyAssertions e **WS-PolicyAttachment** também fazem uso da WS-Policy Framework. O primeiro para definir um conjunto de Asserções para a especificação WS-Policy. Já o segundo serve para definir como incluí-la no WSDL e na UDDI.

WS-Trust: define uma extensão ao WS-Security para provimento de métodos para geração, renovação e validação de *tokens* de segurança, assim como formas de estabelecer e avaliar a presença de relações seguras entre as partes envolvidas na troca de mensagens seguras. Através da especificação WS-Trust é definido o formato das mensagens que serão trocadas para o envio e recebimento de tokens de segurança e os mecanismos usados para a troca de chaves (OASIS, 2004).

WS-Privacy (P3P): esta especificação está atualmente implementada com o novo nome de P3P (Plataforma para Preferências de Privacidade) e serve para especificar como as práticas de privacidade devem ser estabelecidas e implementadas por Web Services.

Trabalha em conjunto com o WS-Policy e WS-Trust descrevendo um modelo de como associar privacidade ao conteúdo das mensagens, quanto à preferência do usuário ou às práticas organizacionais. Esta especificação não parece estar sendo muito usada, apesar de ter sido anunciada a sua criação pela IBM em 2002, sendo que também existe o P3P.

WS-SecureConversation: define extensões que permitem o estabelecimento, o compartilhamento e a derivação de chaves de sessão usando segurança do contexto. O uso de segurança do contexto possui as funcionalidades:

- Define como a segurança do contexto é estabelecida;
- Descreve como contextos de segurança são agrupados;
- Especifica como chaves derivadas são computadas e transmitidas.

WS-Federation: estende o escopo do gerenciamento de identidade possibilitando a federação de confiança. Companhias e parceiros podem utilizar a especificação WS-Federation para compartilhar de forma segura suas características de identidade. Permite intermediar identidades confiáveis, atributos e autenticação entre as partes.

WS-Authorization: descreve como políticas de acesso para Web Services são especificados e gerenciados. Descreve como exigências devem ser especificadas em *tokens* de segurança e como estas exigências são interpretadas nos pontos finais.

Outros mecanismos também são utilizados com diferentes propósitos.

SAML: implementação de *Single Sign On* utilizado na federação de identidades. Proporciona a entidades de negócio fazer asserções com relação à identidade, atributos, e merecimento para outras entidades, como uma companhia parceira (OASIS).

XKMS: especifica protocolos para distribuir e registrar chaves públicas (PKI).

O uso destas especificações influencia diretamente no desempenho de Web Services, sendo que segundo (LIU, PALLIKARA *et al.*, 2005) o uso das especificações WS-* diminuem consideravelmente o desempenho por agregarem segurança nas mensagens SOAP, aumentando conseqüentemente o tamanho do arquivo XML a ser transmitido o que impacta no desempenho da rede e no tempo de processamento das mensagens tanto no receptor quanto no transmissor. A escolha do tipo de algoritmo criptográfico também impacta, porém de forma menos significativa, no desempenho.

Em (LIU, PALLIKARA *et al.*, 2005) e (SHIRASUNA, SLOMINSKI *et al.*, 2004) é demonstrado que o WS-SecureConversation é geralmente o mecanismo mais rápido para segurança no nível de mensagem, no entanto, esta abordagem possui

alguns problemas de escalabilidade quando o serviço precisa ser acessado por um grande número de clientes.

Apesar do uso destas especificações sanarem grande parte dos riscos de segurança de Web Services, apenas utilizá-las não é suficiente para evitar todos os ataques mencionados previamente.

4 WSIVM – MODELO DE VALIDAÇÃO DE ENTRADAS DE DADOS

Há muito tempo a indústria vem procurando maneiras de lidar com falhas de segurança causadas por entradas de dados maliciosas. Mecanismos que funcionam no lado do cliente como o plugin NoScript para o navegador web Mozilla Firefox ou a proteção contra Cross-site scripting do Internet Explorer 8.0 protegem de certa forma o lado do cliente, mas não são eficazes na proteção contra servidores. Para as aplicações Web é possível encontrar soluções diferentes para cada tecnologia para lidar com este problema no lado do servidor.

No entanto, existe um legado de aplicações desenvolvidas sem nenhuma validação de entradas que estão sendo transformadas ou adaptadas para o uso como Web Services.

Por outro lado existe uma integração cada vez maior entre tecnologias sendo que serviços providos por diferentes fontes podem ser utilizados em uma mesma aplicação.

Buscando fornecer segurança na validação de entradas de dados para Web Services foi concebido o modelo WSIVM.

Este modelo de validação de entradas de dados para Web Services traz algumas vantagens em comparação com a validação de entradas na aplicação, como por exemplo:

- evita o desperdício de processamento do servidor com mensagens inválidas;
- impede que a mensagem chegue ao destino sem antes ser validada;
- garante que mensagens com conteúdo malicioso não sejam recebidas pela aplicação;
- diminui a possibilidade de ataques de negação de serviço utilizando o conteúdo das mensagens;
- é independente da tecnologia utilizada para o desenvolvimento interno dos serviços.

O modo mais comum de fazer a utilização de um Web Service apresenta-se da seguinte forma, após encontrar o Web Service que necessita, através da UDDI ou um repositório de serviços, o cliente, que pode ser uma aplicação, um usuário ou uma página, por exemplo,

solicita uma requisição informando o que precisa ao Web Service, conforme mostrado na Figura 12.

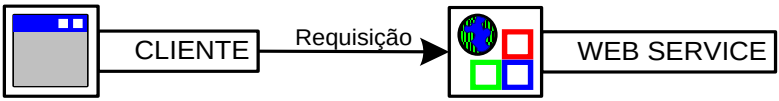

Figura 12 - Requisição ao Web Service

O Web Service retorna a resposta do cliente contendo a resposta de sua requisição.

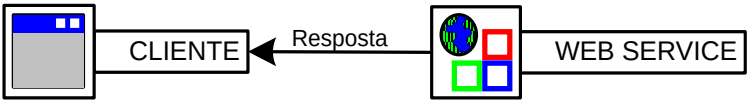

Figura 13 - Resposta do Web Service

Este processo é o mais comumente utilizado. A requisição do cliente é feita diretamente ao Web Service, este por sua vez, processa as entradas e retorna o resultado. O padrão utilizado para esta troca de mensagens é o formato SOAP baseado em XML, conforme explicado na seção 2.2. Caso o Web Service execute a entrada do usuário sem validá-la para uma consulta SQL, por exemplo:

```
SELECT nome, idade FROM clientes WHERE
                nome=entrada;
```

Entrada na qual o parâmetro é passado ao Web Service de acordo com a descrição de chamadas de métodos contida no WSDL.

Se o usuário passar uma entrada válida, por exemplo, o nome “Paulo”, a declaração SQL será:

```
SELECT nome, idade FROM clientes WHERE nome='Paulo';
```

Que retorna o nome e a idade do cliente “Paulo”. No entanto, caso o usuário construa uma entrada maliciosa para ser passada, por exemplo:

```
Paulo' UNION SELECT nome, senha FROM clientes; -
```

A consulta SQL ficaria desta forma:

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

Retornando um valor sigiloso como a senha do cliente para o usuário que enviou a requisição maliciosa para o Web Service. 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 explorada divulgou dados sigilosos.

Com o WSIVM, este exemplo seria executado da seguinte forma. O Usuário faz a requisição de forma usual conforme demonstrado na Figura 12, no entanto, esta requisição é validada pelo WSIVM (Figura 14).

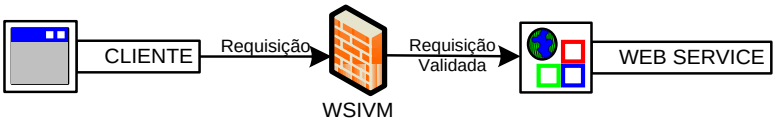

Figura 14 - Requisição SOAP com WSIVM

Caso seja enviada uma requisição maliciosa, como demonstrado no exemplo anterior ao invés de enviar a lista de usuários com senha o mecanismo de validação do WSIVM faz a validação e retorna um erro genérico para o usuário, desta forma o Web Service não recebe a solicitação mal-intencionada (Figura 15).

Figura 15 - WSIVM bloqueia solicitação maliciosa

Analisando de forma mais aprofundada o mecanismo, o que ocorre quando uma mensagem chega para ser validada, é que a entrada enviada pelo usuário é validada através de um módulo de validação que foi desenvolvido com a especificação XML que se encontra no servidor conforme o fluxograma Figura 16.

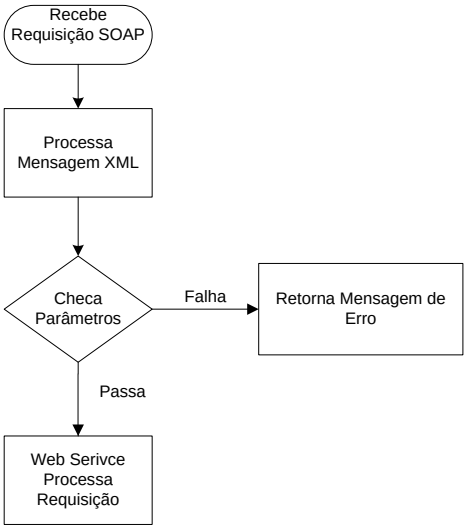

Figura 16 - Validação das Mensagens

Este processo evita o consumo desnecessário de recursos do servidor como no exemplo dado no qual foi efetuada a consulta SQL.

Em (LIN e CHEN, 2006) um mecanismo semelhante foi desenvolvido, focado para o uso em aplicações Web, que propõe a validação de entradas de dados de forma automática. No entanto, esta abordagem possui uma desvantagem que é a grande obtenção de falsos-positivos, ou seja, o validador falha uma mensagem por considerá-la inválida, no entanto é válida, ou vice-versa.

No modelo WSIVM (Web Services Input Validation Model) foi utilizada um abordagem de lista branca ou white-list, no qual apenas valores pré-definidos são aceitos e todos os demais são considerados inválidos. Esta abordagem substituiu a abordagem de lista negra ou black-list por se mostrar mais confiável, conforme citado por Tsipenyuk (TSIPENYUK, CHESS *et al.*, 2005). 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 interação dos componentes do modelo WSIVM funciona da forma mostrada na Figura 17:

WSIVMModule é um módulo Rampart que implementa a classe Module, juntamente com o arquivo Services.xml é reponsável por chamar os outros componentes.

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

WSIVMXMLLoader carrega os elementos operação e as regras especificadas na especificação XML, nome da entrada e valor da entrada usando o SAX e verifica com o WSIVMVerifier a validade ou não da resposta .

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

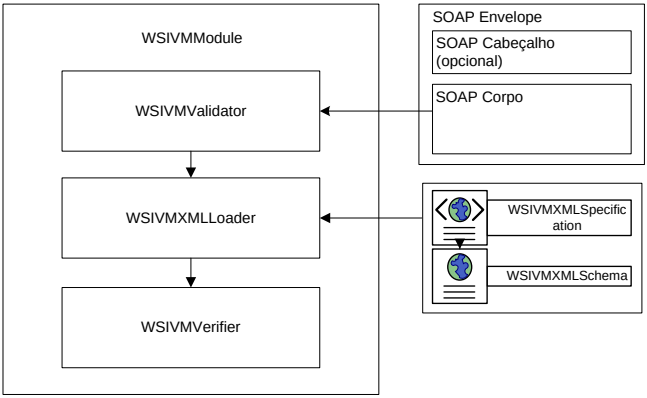

Figura 17 - Funcionamento do WSIVM

As classes do modelo podem ser observadas no diagrama de classes da Figura 18.

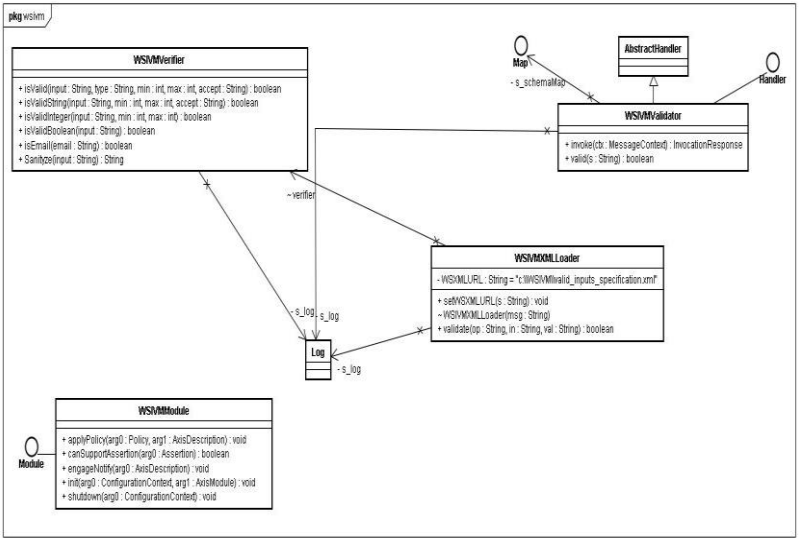

Figura 18 - Diagrama de classes WSIVM

A implementação do modelo foi desenvolvida usando o servidor web Apache Tomcat e o framework para mensagens SOAP Apache Axis2.

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.

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.

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 Web Service.

Em termos gerais o funcionamento do modelo ocorre da seguinte forma: o cliente que pode ser uma aplicação, uma página web ou qualquer mecanismo capaz de se comunicar com um Web Service envia uma mensagem ao Web Service, por exemplo, com os dados para o cadastramento de um usuário. Esta mensagem passa pelo servidor Web que no caso utilizado é o Apache Tomcat. O Servidor Web transmite esta mensagem SOAP para o Apache Axis. O Apache Axis envia a mensagem para ser validada pelo WSIVM. Após a mensagem ser validada, é enviado um erro ao Cliente, caso a mensagem seja

inválida ou é retransmitida ao Web Service para validação, caso seja considerada válida (Figura 19).

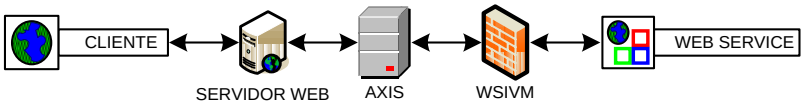

Figura 19 - WSIVM

Segue a descrição detalhada dos components do modelo WSIVM.

WSIVMXMLSchema - Input Validation Specification Schema

Define o formato da especificação XML bem como os atributos válidos. A Figura 20 exemplifica o diagrama do XML Schema.

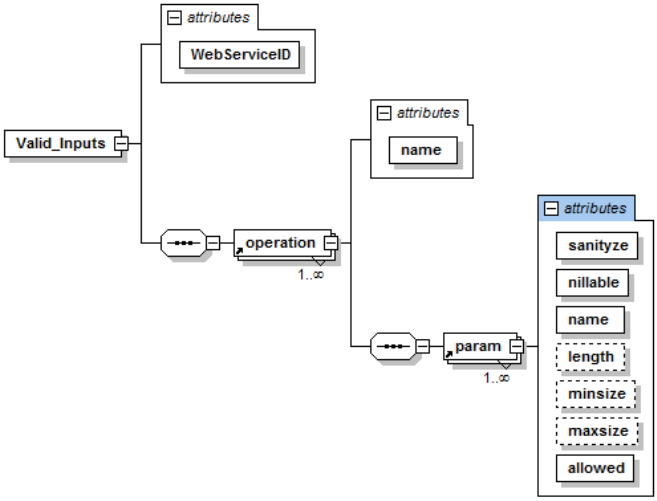

Figura 20 - Exemplo de Diagrama do XML Schema

WSIVMXMLSpecification - Input Validation Specification

Especifica os parâmetros válidos de acordo com um conjunto de atributos pré-definidos. É utilizada para a validação das entradas do usuário.

Dentre eles:

- **OperationName:** nome da operação ou função exposta do Web Service 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

WSIVM - Rampart module

Este 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 as classes Java anteriormente descritas para as validações.

Quando o cliente do Web Service efetua uma operação enviando seus dados, o WSIVM recebe os dados, valida de acordo com a especificação e transmite para que o provedor do Web Service execute ou retorna um erro para o cliente caso os dados não sejam válidos. Este processo é demonstrado na Figura 21 – WSIVM em funcionamento.

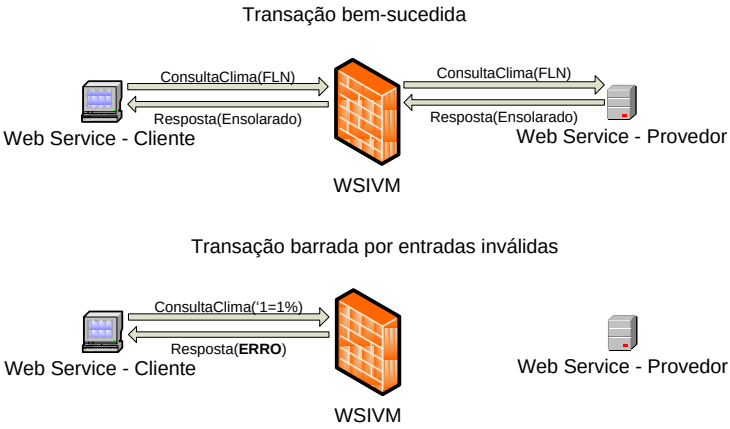

Figura 21 – WSIVM em funcionamento

Este módulo é composto por uma 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.
- Classes Java referentes à validação de entradas: WSIVMModule, WSIVMValidator, WSIVMXMLLoader e WSIVMVerifier.

O arquivo module.xml desenvolvido possui o conteúdo descrito na Figura 22.

```
<module name="wsivm"
class="br.brinhosa.axis2.wsivm.WSIVMModule">
  <Description>Web Services Input Validation
Mechanism module. Define the XML Specification
place for using.
  </Description>
  <InFlow>
    <handler name="WSIVMValidator"
class="br.brinhosa.axis2.wsivm.WSIVMValidator">
      <order phase="PreDispatch"/>
    </handler>
  </InFlow>
</module>
```

Figura 22 - Module.xml

Para seu uso deve-se colocar o arquivo wsivm.mar dentro do diretório Tomcat 6.0\webapps\axis2\WEB-INF\modules.

Uma descrição detalhada do uso do WSIVM é apresentada no próximo capítulo.

5 ESTUDO DE CASO

Como estudo de caso, foi desenvolvido um sistema hipotético de cadastro de alunos para uma universidade denominado GerenciadorUniversidade. Este estudo de caso visa demonstrar o impacto no desempenho do Web Service através do uso do WSIVM na validação de entradas.

O sistema é composto por uma aplicação cliente denominada ClienteGerenciador e um Web Service servidor denominado Gerenciador.

Para o desenvolvimento do Web Service foi utilizada a linguagem Java e os testes de desempenho foram realizados utilizando o programa soapUI (SOAPUI, 2010).

Para o desenvolvimento do Web Service GerenciadorUniversidade criou-se uma classe com as operações buscaAluno e cadastraAluno e uma classe para lidar com as operações do banco de dados. Este Web Service 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 o 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 Web Service 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 23), aonde estão descritos os parâmetros para validação das entradas, conforme previamente explicado no capítulo 4.

```
<?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="false"/>
                <input name="idade" type="Integer" min="0"
max="150" accept="number" sanityze="false"/>
                <input name="email" type="String" min="0" max="200"
accept="email" sanityze="false"/>
                <input name="comentario" type="String" min="0"
max="200" accept="text" sanityze="false"/>
                <input name="site" type="String" min="0" max="300"
accept="url" sanityze="false"/>
                <input name="data" type="String" min="0" max="200"
accept="regex" regexpattern= "(\\d{4})-(\\d{2})-(\\d{2})"
sanityze="false"/>
        </operation>
        <operation name="buscaAluno">
                <input name="id" type="Integer" min="0" max="10000"
accept="number"/>
        </operation>
</valid_inputs_specification>
```

Figura 23 - 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.

```
<service>
      <parameter name="ServiceClass"

locked="false">exemplo.wsivm.universidade.Gerencia
dor</parameter>
      <operation name="cadastraAluno">
            <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageRe
ceiver"/>
      </operation>
      <operation name="buscaAluno">
```

```
            <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageRe
ceiver"/>
      </operation>
         <module ref="wsivm"/>
   <parameter
name="validationXML">file:///C:/WSIVM/valid_inputs
_specification.xml</parameter>
</service>
```

Figura 24 - Services.xml – GerenciadorUniversidade

Foi gerado o pacote denominado Gerenciador.aar contendo a classe Gerenciador e MySQL. no pacote e o descritor MANIFEST.MF e o arquivo Services.xml na pasta META-INF.

Este pacote pode ser gerado como um pacote .jar e renomeado como .aar. E colocado na pasta Services do Apache Axis2.

Foi então copiado o arquivo do módulo (wsivm.mar) para dentro do diretório ..\Tomcat 6.0\webapps\axis2\WEB-INF\modules.

Após estes procedimentos foi inicializado o Apache Tomcat para que os serviços estivessem disponíveis para os testes.

Neste experimento foram realizados dois testes, um usando o modelo de validação de entradas WSIVM e outro não.

O soapUI oferece uma interface amigável para a realização dos testes. Os testes são realizados fazendo chamadas diretas ao Web Service.

O seguinte cenário foi configurado para a realização dos testes, 150 usuários, sendo inicializados gradativamente, seguindo a ordem de 1 usuário sendo inicializado a cada 2 segundos, o teste é rodado 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.

A mensagem SOAP é enviada enviada conforme o exemplo da Figura 25.

```
<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 25 - Exemplo de mensagem SOAP enviada pelo soapUI

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, no eixo Y é mostrado o valor em milisegundos do tempo de resposta (Figura 26).

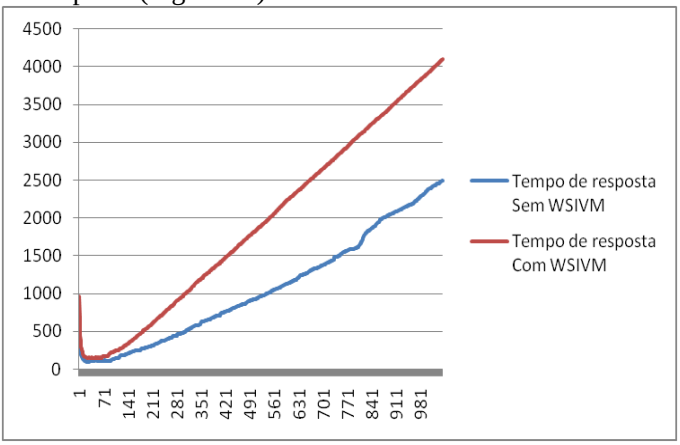

Figura 26 - Tempos de resposta com e sem o uso do WSIVM

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, no eixo Y é mostrado o número de bytes por segundo (Figura 27).

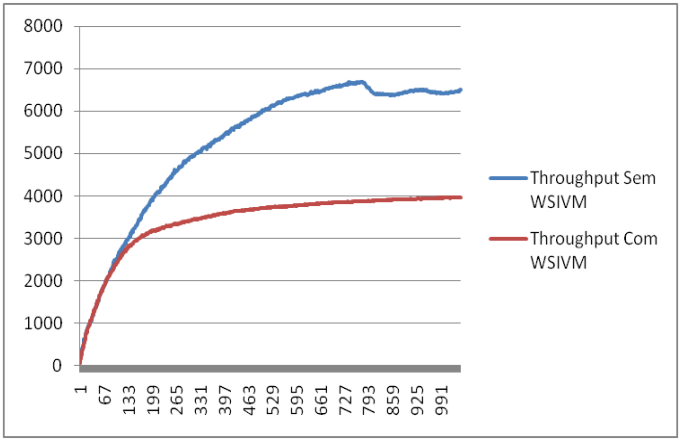

Figura 27 - Throughput com e sem validação WSIVM

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

Tabela 2 - Resultados consolidados do Estudo de caso

WS	Min	Max	Média	Bytes Transferidos	Bytes por segundo (*throughput*)	Inserções no Banco
Sem WSIVM	35	27848	2494,85	1974195	6506	10078
Com WSIVM	64	13346	4541,24	1236330	4012	5134
Total	83%	-52%	82%	-37%	-38%	-49%

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.

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.

O aumento de segurança geralmente impacta negativamente no desempenho e isto pôde ser observado neste estudo de caso.

No entanto, a validação dos campos não permitindo que dados inválidos fossem inseridos pode compensar a perda de desempenho.

Em testes avaliando-se a eficácia do mecanismo com relação a proteção contra ataques, nenhuma entrada indevida foi processada desde que devidamente configurado para filtrar as entradas inválidas. No entanto, testes mais extensivos, usando inclusive caracteres double-byte, utilizado em idiomas como o Japônes, devem ser realizados no futuro.

6 CONCLUSÃO E TRABALHOS FUTUROS

A interoperabilidade trazida pela arquitetura de Web Services causou uma verdadeira revolução na computação, tornando possível que, de forma prática, aplicações pudessem trocar informações usando uma linguagem comum. No entanto, nunca antes aplicações foram tão expostas na Internet quanto agora com o uso de Web Services.

Esta exposição facilita a obtenção de informações da lógica do negócio por usuários maliciosos e o seu uso para efetuar ataques através da inserção de dados indevidos em suas interfaces expostas na web.

Este tipo de ataque é amplamente discutido e observado em aplicações web. No entanto, há muito espaço para crescimento científico na área de validação de entradas de dados para Web Services.

SOA e Web Services tem amplo uso e apesar de fazerem parte de um processo lento de mudança de paradigmas é possível verificar que estão em constante crescimento, sendo que os problemas de segurança encontram-se cada vez mais evidentes.

Este trabalho buscou uma solução para impedir ataques de injeção de dados em Web Services, fornecendo um modelo para proteção de suas interfaces 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 Web Service que faz seu uso.

Grandes desafios foram observados no desenvolvimento desta pesquisa como a dificuldade de definir um conjunto adequado de mecanismos para a implementação do modelo.

Um processo de validação de entradas que garanta que apenas dados válidos serão processados foi apresentado.

Na elaboração deste trabalho foram excluídos determinados aspectos do escopo visando obter um trabalho mais conciso. No entanto, existem diferentes aspectos que podem ser abordados em trabalhos futuros como:

- Proteção contra ataques de negação de Serviço, ou DoS (Denial of Service), visando garantir a disponibilidade, possibilitando continuar as operações enquanto possível e provendo a recuperação após os ataques.
- Consideração de aspectos de segurança para Web Services compostos.

- Desenvolver um gerador semi-automático de especificações de segurança a partir do WSDL.
- Verificar mensagens SOAP, estrutura, parâmetros, tamanho da mensagem e XPath.
- Usar Inteligência Artificial ou um sistema de detecção de anomalias.
- Fazer um esquema de retroalimentação do filtro de validação de entradas inválidas.

7 REFERÊNCIAS BIBLIOGRÁFICAS

(SSA), S. S. A. L. **WS-Security Wrapper**. 2007. Disponível em: <http://wsswrapper.sourceforge.net/>.

ABRAMS, C.; SCHULTE, R. W. **Service-Oriented Architecture Overview and Guide to SOA Research**: Gartner, Inc. 2008.

BALIEIRO, S. **Segurança desencoraja uso de SOA e webservice**. 2008. Disponível em: < http://info.abril.com.br/corporate/noticias/112008/12112008-0.shtml>. Acesso em: 25 de dezembro de 2008.

BEBAWY, R. et al. Nedgty: Web services firewall. In: IEEE International Conference on Web Services - ICWS, Orlando, 2005. **Proceedings…** Orlando: IEEE, 2005. p.597-601.

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

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

BIRMAN, K. et al. Building Collaboration Applications That Mix Web Services Hosted Content with P2P Protocols. In: IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES (ICWS), Los Angeles, 2009. **Proceedings...** Los Angeles, CA: IEEE, 2009. p.509-518.

BLYTH, A. An Architecture for an XML Enabled Firewall. **International Journal of Network Security.** v. 8, n. 1, p. 31-36, 2009.

BUSTAMANTE, M. L. Windows Communication Foundation: Application Deployment Scenarios. 2008. Disponível em: <http://msdn.microsoft.com/en-us/library/cc512374.aspx>.

CA. IT Management software and solutions – CA Brazil. 2010. Disponível em: <http://www.ca.com/br/>.

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

CLARKE, J. **SQL Injection Attacks and Defense**. Syngress Media Inc, 2009.

DATABREACHES.NET. Top 10 Worst Data Losses or Breaches, updated | Office of Inadequate Security. 2009. Disponível em: <http://www.databreaches.net/?p=7691>.

DIERKS, T.; RESCORLA, E. RFC 4346: The transport layer security (TLS) protocol version 1.1. **IETF RFC4346**. 2006.

FREIER, A.; KARLTON, P.; KOCHER, P. The SSL Protocol, V3. 0. **IETF draft, http://wp. netscape. com/eng/ssl3/3-spec. htm**, 1996.

FRIEDMAN, T. L. **O mundo é plano: uma breve história do século XXI**. Rio de Janeiro: 2005.

FYFFE, G. Addressing the insider threat. **Network Security,** v. 2008, n. 3, p. 11-14, 2008. ISSN 1353-4858. Disponível em: <http://www.sciencedirect.com/science/article/B6VJG-4S2F2KJ-9/2/0a0cd0668d5f59c71de1ace25a4ae64b>.

GARTNER. Gartner Technology Business Research Insight. 2010. Disponível em: <http://www.gartner.com/technology/home.jsp>.

GISOLFI, D. Web services architect, Part 3: Is Web services the reincarnation of CORBA?, 2001. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-arc3/>.

HAYATI, P., NASTARAN JAFARI, S. M. REZAEI, AND SAEED SARENCHE. Modeling Input Validation in UML. In: AUSTRALIAN CONFERENCE ON SOFTWARE ENGINEERING, 19., Perth, Western Australia. **Proceedings…** Perth: ACS, 2008.

HUNTER, D. Beginning XML. Indianapolis, Ind.; Chichester, 2007. ISSN 9786610900565 6610900566. Disponível em: <http://0-www.myilibrary.com.mercury.concordia.ca?id=90056>.

IBM. IBM Internet Security Systems X-Force® 2008 Trend & Risk Report. 2009. Disponível em: <http://www-935.ibm.com/services/us/iss/xforce/trendreports/xforce-2008-annual-report.pdf >.

JENSEN, M.; GRUSCHKA, N.; HERKENHÖNER, R. A survey of attacks on web services. **Computer Science-Research and Development,** v. 24, n. 4, p. 185-197, 2009.

JOSUTTIS, N. M. **SOA na Prática**. Alta Books, 2008.

KIEYZUN, A. et al. Automatic creation of SQL injection and cross-site scripting attacks. In: IEEE COMPUTER SOCIETY WASHINGTON,

DC, USA, 31., 2009. **Proceedings…** Washington: IEEE, 2009. p.199-209.

LIN, J.-C.; CHEN, J.-M. **An Automatic Revised Tool for Anti-Malicious Injection**. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY, 6., 2006. **Proceedings…** Korea: IEEE Computer Society, 2006.

LIN, J.; CHEN, J. An Automated Mechanism for Secure Input Handling. **Journal of Computers,** v. 4, n. 9, p. 837, 2009.

LIU, H.; PALLIKARA, S.; FOX, G. **Performance of Web Service Security**. 2005.

LOH, Y. et al. **Design and Implementation of an XML Firewall**. 2006.

MATTSSON, U.; GREEN, O. Securing Data Beyond PCI in a SOA Environment: Best Practices for Advanced Data Protection. **Insecure Magazine,** v. 19, 2008.

MICROSOFT. Microsoft Anti-Cross Site Scripting Library V3.0 Beta. 2008. Disponível em: <http://www.microsoft.com/downloads/details.aspx?FamilyId=051ee83c-5ccf-48ed-8463-02f56a6bfc09&displaylang=en>. Acesso em: 20 de dezembro de 2008.

_____. The Security Development Lifecycle : SIR Volume 7 Released. 2009. Disponível em: <http://blogs.msdn.com/sdl/archive/2009/11/04/sir-volume-7-released.aspx>.

OASIS. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). 2004. Disponível em: <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>.

OASIS, S. SAML XML.org | Online community for the Security Assertion Markup Language (SAML) OASIS Standard. Disponível em: <http://saml.xml.org/>.

OASIS, U. UDDI | Online community for the Universal Description, Discovery, and Integration. 2010. Disponível em: <http://uddi.xml.org/>.

OWASP. Open Web Application Security Project. 2010. Disponível em: <http://www.owasp.org>.

SHIRASUNA, S. et al. Performance comparison of security mechanisms for grid services. In: IEEE/ACM International Workshop on Grid Computing, 5., Pittsburgh, 2004. **Proceedings…** Pittsburgh: IEEE]ACM, 2004. p. 360-364.

SHOSTACK, A.; STEWART, A. **The new school of information security**. Boston: Addison-Wesley, 2008. p. 288.

SIDHARTH, N.; LIU, J. A Framework for Enhancing Web Services Security. In: ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE - COMPSAC, 31., Beijing, 2007. **Proceedings…** v. 1. Beijing: IEEE, 2007.

SOAPUI. the Web Service, SOA and SOAP Testing Tool - soapUI. 2010. Disponível em: <http://www.soapui.org/>.

SUN, L.; LI, Y. XML and web services security. In: INTERNATIONAL CONFERENCE ON COMPUTER SUPPORTED COOPORATIVE WORK IN DESIGN, 12., Xi'an (China), 2008. **Proceedings…** Xi'an: IEEE, 2008.

TSIPENYUK, K.; CHESS, B.; MCGRAW, G. Seven pernicious kingdoms: a taxonomy of software security errors. **Security & Privacy, IEEE,** v. 3, n. 6, p. 81-84, 2005. ISSN 1540-7993.

VIEGA, J.; EPSTEIN, J. Why Applying Standards to Web Services Is Not Enough. **IEEE security & privacy.**, New York, NY**,** v. 4, n. 4, p. 25-31, 2006. ISSN 1540-7993.

VIJAYAN, J. Heartland data breach sparks security concerns in payment industry. **Computerworld.** January, v. 22, 2009.

W3C. W3C SOAP Specifications. Disponível em: <http://www.w3.org/TR/soap/>.

_____. Web Service Definition Language (WSDL). Disponível em: <http://www.w3.org/TR/wsdl>.

_____. HTTP - Hypertext Transfer Protocol. 2007/10/24. 2007. Disponível em: <http://www.w3.org/Protocols/>. Acesso em: 26 de dezembro de 2007.

WASC. Web Application Security Consortium. 2010. Disponível em: <http://www.webappsec.org/>.

ANEXOS

ANEXO 1

WSIVMModule.java

```
package br.brinhosa.axis2.wsivm;

import org.apache.axis2.AxisFault;
import
org.apache.axis2.context.ConfigurationContext;
import
org.apache.axis2.description.AxisDescription;
import org.apache.axis2.description.AxisModule;
import org.apache.axis2.modules.Module;
import org.apache.neethi.Assertion;
import org.apache.neethi.Policy;

public class WSIVMModule implements Module
{

    public void applyPolicy(Policy arg0,
AxisDescription arg1) throws AxisFault {
    }

    public boolean canSupportAssertion(Assertion
arg0) {
        return false;
    }

    public void engageNotify(AxisDescription
arg0) throws AxisFault {
    }

    public void init(ConfigurationContext arg0,
AxisModule arg1)
        throws AxisFault {
    }

    public void shutdown(ConfigurationContext
arg0) throws AxisFault {
    }
                        }
```

WSIVMVerifier.java

```
package br.brinhosa.axis2.wsivm;
import java.util.StringTokenizer;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.mail.Internet.AddressException;
import javax.mail.Internet.InternetAddress;


import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.commons.validator.UrlValidator;

public class WSIVMVerifier {
     /** Logger for this handler. */
     private static final Log s_log;
     static {
          s_log =
LogFactory.getLog(WSIVMValidator.class);
     }

     public boolean isValid(String input, String
type, int min, int max, String accept){
          boolean resultado = true;
          String _input = input;
          String _type = type;
          int _min = min;
          int _max = max;
          String _accept = accept;

          if (_type.equals("Integer")){
               if
(this.isValidInteger(_input,_min,_max)){
                    resultado = true;
                    return resultado;
               }
          }

          if (_type.equals("Boolean")){
               if (this.isValidBoolean(_input)){
                    resultado = true;
```

```
                    return resultado;
               }
          }

          if (_type.equals("String")){
               if
(this.isValidString(_input,_min,_max,_accept)){
                    resultado = true;
                    return resultado;
               }
          }
          return resultado;
     }
     public boolean isValidString(String input,
int min, int max, String accept){
          boolean resultado = false;
          System.out.println("isValidString");
          String _input =  input;
          int _min = min;
          int _max = max;
          String _accept = accept;
          if (_input.length()>_max){
               resultado = false;
               System.out.println("String >
max");
               s_log.error("String > max");
               return resultado;
          }
          if (_input.length()<_min){
               resultado = false;
               System.out.println("String <
min");
               s_log.error("String < min");
               return resultado;
          }
          if (_accept.equals("email")){
               if (this.isEmail(_input)){
                    resultado = true;
                    return resultado;
               }
               else
                    return false;
          }
```

```
          if (_accept.equals("url")){
               System.out.println("isValidURL");
               UrlValidator results = new
UrlValidator();
               resultado =
results.isValid(_input);
               return resultado;

          }
          return resultado;
     }
     public boolean isValidInteger(String input,
int min, int max){
          boolean resultado = true;
          System.out.println("isValidInteger");
          int value;
          if (input.length() != 0) {
               try {
                    value =
Integer.parseInt(input);
                    if
(!(value>=min&&value<=max)){
                         resultado = false;
                         System.out.println("Min
or Max invalid");
                         s_log.error("Min or Max
invalid");
                    }
               } catch (NumberFormatException
nfe) {
                    System.out.println("Invalid
number format");
                    s_log.error("Invalid number
format");
                    resultado = false;
               }

          }
          return resultado;
     }

     public boolean isValidBoolean(String input){
          boolean resultado = false;
```

```
          System.out.println("isValidBoolean");
          String _input =  input;
          if
(_input.equals("true")||_input.equals("false"))
               resultado = true;
          else{
               s_log.error("Boolean not valid");
          }
          return resultado;
     }

     public boolean isEmail(String email){
          //boolean resultado = true;
          System.out.println("isValidEmail");
          try {
               InternetAddress.parse(email);

               System.out.println("Valid email:
"+email);
          return true;
          }
          catch (AddressException a){
               System.out.println("Invalid email:
"+email);
               s_log.error("Invalid email:
"+email);
               return false;
          }
     }


     public String Sanityze(String input){
          String resultado = "";

          return resultado;
     }
     //validar html chamando
http://validator.w3.org/check?uri=http://www.exam
ple.com/&output=soap12
http://validator.w3.org/docs/api.html

}
```

WSIVMXMLLoader.java

```
package br.brinhosa.axis2.wsivm;

import java.io.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import
javax.xml.parsers.ParserConfigurationException;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
public class WSIVMXMLLoader {
     private String WSXMLURL =
"c:\\WSIVM\\valid_inputs_specification.xml";
     private static final Log s_log;
     WSIVMVerifier verifier = new
WSIVMVerifier();
     static {
          s_log =
LogFactory.getLog(WSIVMXMLLoader.class);
     }
     public void setWSXMLURL(String s){
          WSXMLURL = s;
          System.out.println(WSXMLURL);
     }
     WSIVMXMLLoader(String msg){

          s_log.info("Funcionando");

          try {
               DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
               DocumentBuilder builder =
factory.newDocumentBuilder();
               System.out.println("Parser
doing!");
```

```
                InputStream is = new
ByteArrayInputStream( msg.getBytes());
                Document documento =
builder.parse( is);
                System.out.println("Parser
done!");

                NodeList all =
documento.getElementsByTagName("*");
                String op
=documento.getFirstChild().getNodeName().substrin
g(documento.getFirstChild().getNodeName().indexOf
(":")+1);
                System.out.println("WSName: "+op);
                for (int i=1; i < all.getLength();
i++) {
                     System.out.println("Input:
"+all.item(i).getNodeName());
                     String in =
all.item(i).getNodeName().substring(all.item(i).g
etNodeName().indexOf(":")+1);
                     System.out.println("Short
Input: "+in);
                     String val =
all.item(i).getTextContent();
                     System.out.println("Value:
"+val);
                     validate(op, in, val);

                }

           } catch (IOException e) {
                System.err.println("Erro de
IO:"+e);
                //System.exit(1);

           } catch (SAXParseException spe) {
                System.out.println("\n** Erro no
parsing "
                          + ", linha " +
spe.getLineNumber()
                          + ", uri " +
spe.getSystemId());
```

```
                System.out.println("    " +
spe.getMessage() );
                //System.exit(1);

            } catch (ParserConfigurationException
pce) {
                System.out.println("Erro na
configuração do parser");
                //System.exit(1);

            } catch (SAXException e) {
                System.err.println("Erro de
xml:"+e);
                //System.exit(1);

            } catch (Exception e) {
                e.printStackTrace();
                System.err.println("Outro erro:
"+e);
                //System.exit(1);

            } finally {
                //System.exit(0);
            }

     }
     public boolean validate(String op, String
in, String val){
          boolean results = true;
          System.out.println("Validate "+op+":
"+in+": "+val);
          try {
                DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
                DocumentBuilder builder =
factory.newDocumentBuilder();
                Document documento =
builder.parse(WSXMLURL);
                Element emissor  =
(Element)documento.getElementsByTagName("operatio
n").item(0);

     System.out.println(emissor.getAttribute("nam
```

```
e"));
                NodeList operations =
documento.getElementsByTagName("operation");
                for (int i=0; i <
operations.getLength(); i++) {
                    Element operation =
(Element)operations.item(i);

                    if
(operation.getAttribute("name").equals(op)){

     System.out.println("Operação encontrada:
"+operation.getAttribute("name"));
                        NodeList ends =
operation.getElementsByTagName("input");
                        if (ends.getLength() > 0)
{
                            for(int j=0; j <
ends.getLength(); j++) {
                                Element input =
(Element)ends.item(j);
                                if
(input.getAttribute("name").equals(in)){

     System.out.println(in+" -
"+input.getAttribute("type")+" -
"+input.getAttribute("min")+" -
"+input.getAttribute("max")+" -
"+input.getAttribute("accept"));
                                    if
(!verifier.isValid(val,
input.getAttribute("type"),
Integer.parseInt(input.getAttribute("min")),
Integer.parseInt(input.getAttribute("max")),
input.getAttribute("accept"))){

     results = false;

     System.out.println("Invalid Values");
                                        return
false;
                                    }
                                    else
```

```
                                                {

     System.out.println("Valores validos para:
"+input.getAttribute("name")+" "+val);
                                                }

                                            }
                                        }
                                    }
                                }
                            }
                        } catch (Exception e) {
                            e.printStackTrace();
                            System.err.println("Outro erro:
"+e);
                            //System.exit(1);
                        } finally {
                            //System.exit(0);
                        }
                        return results;
                }

}
```

Gerenciador.java

```
package exemplo.wsivm.universidade;

import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Connection;

import com.mysql.jdbc.ResultSet;
import com.mysql.jdbc.Statement;

public class MySQL {

     private String host;
     private String user;
     private String pass;
     private String database;
```

```
    public Connection c;

    public MySQL( String host, String database,
String user, String pass ) {
        this.pass = pass;
        this.user = user;
        this.host = host;
        this.database = database;
    }

    public boolean connect() {
        boolean isConnected = false;
        String url;

        url = "jdbc:mysql://"+this.host+"/"
        +this.database+"?"
                +"user="+this.user
                +"&password="+this.pass;

        try {

    Class.forName("com.mysql.jdbc.Driver").newIn
stance();
            //System.out.println(url);
            this.c =
DriverManager.getConnection(url);

            isConnected = true;

        } catch( SQLException e ) {
            e.printStackTrace();

    System.out.println(e.getMessage());
            isConnected = false;
        } catch ( ClassNotFoundException e ) {
            e.printStackTrace();

    System.out.println(e.getMessage());
            isConnected = false;
        } catch ( InstantiationException e ) {
            e.printStackTrace();

    System.out.println(e.getMessage());
```

```
                isConnected = false;
            } catch ( IllegalAccessException e ) {
                e.printStackTrace();

     System.out.println(e.getMessage());
                isConnected = false;
            }

            return isConnected;
        }
        public java.sql.ResultSet executar( String
query ) {
            java.sql.Statement st;
            java.sql.ResultSet rs;

            try {
                st = this.c.createStatement();
                rs = st.executeQuery(query);
                this.c.close();
                return rs;
            } catch ( SQLException e ) {
                e.printStackTrace();
                try{
                    this.c.close();}
                catch (Exception f){
                }
            }

            return null;
        }

        public int inserir( String query ) {
            java.sql.Statement st;
            int result = -1;

            try {
                st = this.c.createStatement();
                result = st.executeUpdate(query);
                this.c.close();
            } catch ( SQLException e ) {
                e.printStackTrace();
                try{
                    this.c.close();}
```

```
                catch (Exception f){

                }
            }
            return result;
        }

                                }
```

ANEXO 2

Artigo publicado na SECURWARE2008.
A Security Framework for Input Validation

Rafael Bosse Brinhosa, Carlos Becker Westphall, Carla Merkle Westphall
Network and Management Laboratory - Postgraduate Program in Computer Science - Technological Center
Federal University of Santa Catarina, Brasil
{brinhosa, westphal}@inf.ufsc.br

Abstract

Input manipulation attacks are becoming one of the most common attacks against Web Applications and Web Services security. As the use of firewalls and other security mechanisms are not effective against application-level attacks, new means of defense are needed. This paper presents a framework proposal to solve this problem, securing applications against input manipulation attacks. The proposed mechanism offers a reusable approach by the use of XML files and a XML Schema for security parameters specification. Furthermore, a case of study and experiment results are presented. The experiment demonstrates how common input manipulation flaws could be observed.

1. Introduction

Web Applications and Web Services are frequently vulnerable to a large number of attacks. Input manipulation attacks, such as Cross Site Scripting (XSS), are becoming the most common kind of attack performed by malicious users.

The input manipulation attack typically occurs from the application interface and could be used to exploit the Application Server, providing access to databases, files and systems. This attack against Web Applications is very well-know but with the growth of Web Services use, Web Services applications are becoming the new target of input manipulation attacks.

According to Gartner group, Web Services will reopen 70% of the attacks paths closed by firewalls over the past decade [1].

Usually the use of SSL and firewalls are not effective against application-level attacks.

Hence applications are developed using different languages and technologies, this paper proposes a framework for securing applications against input manipulation attacks using a reusable approach.

This paper is organized as follows: section 2 describes related work, section 3 presents the proposed framework, describes XML aspects and a flowchart of the framework. A case of study is the topic of section 4 and section 5 indicates experiment results. Finally in section 6 the paper presents some conclusion and future work.

2. Related Work

There are some articles related to this subject. Hayes[2] defines an Input Validation testing technique. This technique was developed to address the problem of statically analyzing input command syntax and requirements specifications to generate test cases for input validation testing. Lin [3] has developed an advanced tool that can produce an input validation function depending on the database server and the application framework. The tool can automatically insert input proper validation function into the server-side program to eliminate vulnerabilities based on malicious injection. Liu[4] introduce some invariant properties regarding input validation by analyzing the control and data dependency, among inputs accessed and effects raised in a program. Then proposes a method for the automated recovery of input validation from program source code. Based on the information recovered, techniques to aid the understanding and maintenance of the Input Validation using program slicing are presented. Li [5] presents a model driven framework to help user input validation testing in a visual environment which guides the test work development. The work of Li [5] defines a meta-model of Web application for user input validation testing. Based on the meta-model and analyzing HTML files, a light weight method is given to create the model. Hayati [6] proposes to extend UML into a new integrated framework to provide modeling driven security engineering. The framework aims at achieving a different way to design more secure software. This approach has some advantages such as preventing from common input tampering attacks, having both security and convenience in software at high level of abstraction and ability of solving the problem of weak security background for developers.

Most of the papers of the literature commented here are related to security testing or automatic mechanisms. Security testing is important, but more important than finding security flaws is preventing them. Also automatic mechanisms have their limitations, such as not recognizing inputs correctly depending on the Application's complexity, what could result in malicious attacks. The main contribution of this paper when compared with related research work is that this paper presents a more reusable and universal solution, using separate XML files for specifying security attributes in the development phase.

3. Framework Proposal

Attacks to Web Applications and Web Services are very common due to incorrect or non-input validation.

The lack of the right input validation could result in different kinds of attack; the most common are defined bellow:

- Cross Site Scripting (XSS): consists of executing scripts on Web pages through the exploration of not correctly validated fields or URL.
- SQL injection: consists of the execution of SQL commands through the manipulation of the URL variables or fields.
- Hidden field manipulation: manipulation of hidden fields in order to explore some vulnerability.
- Buffer Overflows: sending messages bigger than the maximum allowed in order to execute arbitrary commands.

The proposed framework has the primary objective of validating the user inputs before the Application starts to execute. It consists of a XML Schema, a XML file, a server mechanism for input validation and the front-end application.

The XML Schema defines the valid XML specification for the framework. The XML file defines the valid inputs for the required Application fields.

The server mechanism is called to validate the user inputs according to the XML specifications.

The front-end application is represented by any application which uses the framework for validating inputs.

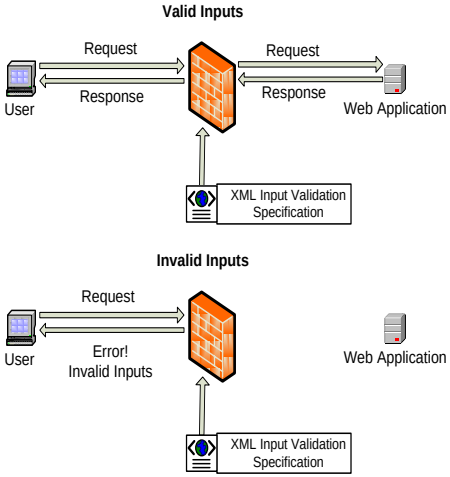

Figure 1 Framework in action.

When the user sends a request for an Application, the Input Validation mechanism receives the Request and checks for any inconsistence according to the pre-defined XML Input Validation specification. If the inputs sent are valid, the request is passed to the Web Application, otherwise, the user receives an error response.

The framework can be used with already developed or new applications.

When using the framework with already developed application, she must adapt the application methods for calling the mechanism to validate inputs after receiving them.

Although, when using the framework with new applications, he can develop the application methods calling the mechanism every time an input is received.

This framework has some important characteristics. It works in the server-side - this means that any input from the Client will be validated before being processed. Some developers usually do the input invalidation in the client-side, but this is not correct and secure because the validation can be bypassed.

Another important characteristic of the framework is that it uses a single validation mechanism for the entire system, enabling many applications from the same server to make use of the same mechanism.

The framework is also XML based, what simplifies the application maintenance: if any changes are necessary, the developer just needs to change the XML application file.

This proposed framework addresses the big problem of Input Validation and with the use of this framework Web Applications and Web Services will be secured against the various types of Input manipulation attacks.

3.1. XML Schema and XML File

The XML Schema is used for validating the XML file that contains the system's security specification. It consists of different attributes for input validation specification like:

- Section name;
- Field name;
- Allowed: numbers, strings, html, html+script;
- Length.
-

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSpy v2008 sp1 (http://www.altova.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="valid_inputs">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="section" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="pageID" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="section">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="input" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" use="required">
        <xs:simpleType>
```

Figure 2 XML Schema

The XML file describes the allowed inputs the Application supports, everything which is not explicitly allowed in the XML file is denied.

```
<?xml version="1.0" encoding="UTF-8"?>
<valid_inputs xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" pageID="siteBrinhosa"
 xsi:noNamespaceSchemaLocation="valid_inputs.xsd">
   <section name="formMail">
      <input name="email3" allowed="email" maxsize="20"/>
      <input name="message1" allowed="text" maxsize="200"/>
   </section>
   <section name="blog">
      <input name="names" allowed="html" maxsize="100"/>
      <input name="messagerich" allowed="html+script" maxsize="200"/>
   </section>
   <section name="form">
      <input name="name1" allowed="text" maxsize="100"/>
      <input name="age1" allowed="number" maxsize="20"/>
   </section>
</valid_inputs>
```

Figure 3 XML file specifying the valid inputs.

3.2. Mechanism properties

The mechanism reads the XML specification and applies the specified rules, validating the inputs, and returning a message if the inputs are considered valid or not.

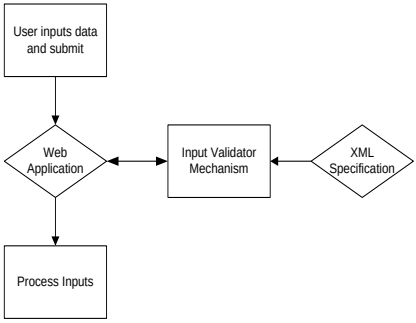

Figure 4 Framework flow-chart

4. Case of study

This section describes a case of study in detail showing how our approach addresses Input Validation for ensuring Application Security.

The framework makes it easy to construct secure Web applications.

The first step in the development was the construction of a simple web application using PHP and HTML, consisting of a HTML Web email form with the following text fields: Name, Email, Subject and Message and two buttons: Send and Reset. This is a typical contact's form.

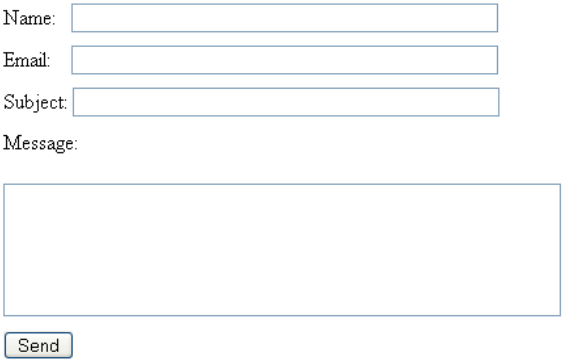

Figure 5 Web email form

The next step was the development of a PHP application, which receives the user inputs, validates, sends an email message to a specified email and stores the data in the database.

The PHP was specified for sending the inputs to the Input Validator mechanism developed also in PHP.

Then a XML security specification file according to the XML Schema was developed, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<valid_inputs
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
pageID="siteEmail"
xsi:noNamespaceSchemaLocation="valid_inputs.xsd">
<section name="formMail">
<input name="name" allowed="text" maxsize="20"/>
<input name="email" allowed="email" maxsize="20"/>
<input name="message1" allowed="text" maxsize="200"/>
</section>
</valid_inputs>
```

The Input Validator mechanism receives the user inputs and validates these inputs using the XML specification through PHP XML functions and regular expressions. For this case of study the mechanism was implemented with two types of pre-defined allowed inputs “text” and “email”. The “text” attribute accepts alphanumerical but no html or java-script codes. The “email” attribute just accepts inputs using the format “username@provider.com”.

Then the Web Application was tested using a hacker mindset sending malicious inputs such as:

- <’”/*;
- *%')%20UNION%20SELECT%20
- "'<script>alert("x")</script>

All the values tested returned an error message to the user stating that inputs are not valid. From this case of study, we could verify that the framework works properly.

5. Experiment results

In order to verify how easy the injection flaws are encountered, an experiment was done following the steps bellow:

- Access a search web site engine;
- Search for the words “site:.college namedomain login”, where “college name” was replaced by one of the five previously selected colleges;
- Enter the web site and type “username” for the username field, “password” for the password field, click on the “Ok” button and check the results;
- Enter the web site again and type “*/'``”;.

”<script>alert('1');</script>**” for the both fields, username and password and click on the “Ok” button and check the results. If the result page is abnormally changed, with two lines inserted, a database error occurred or a popup appeared and the web page has a flaw.

This experiment was done accessing 15 different web sites related to each College: for 5 Colleges the experiment results are presented at table 1.

TABLE I
TESTING INJECTION FLAWS ON COLLEGE WEB SITES

ollege	Sites with injection flaw	Sites without injection flaw
	2	13
	1	14
	2	13
	5	15
	6	9

With this simple experiment, we proved that Web Applications injection flaws are very easy to find, even in College related web pages.

6. Conclusions and Future Work

This paper presents the following major contributions:

- An independent specification of Security for Input validation, through the use of a separated XML file;
- A very reliable framework validating the user-inputs in the server-side instead of in the client-side, assuring that all inputs will be validated before processing;
- A reusable approach with the implementation of the Validator mechanism;
- The proposed architecture could be used for Web Applications or Web Services;
- An experiment result that present how easy is to find Web Applications with Security flaws related to the lack of input validation.
- Regarding this study, in future research it can be implemented a system for Input attacks detection and response, for instance, blocking the attacker access to the Web Application.

7. References

[1] Yu, Weider D., Dhanya Aravind, and Passarawarin Supthaweesuk. "Software Vulnerability Analysis for Web Services Software Systems." Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06) (2006).

[2] Hayes, Jane H., and A. J. O Utt. "Increased Software Reliability Through Input Validation Analysis and Testing."

[3] Lin, Jin-Cherng, and Jan-Min Chen. "An Automatic Revised Tool for Anti-Malicious Injection." Proceedings of the Sixth IEEE International Conference on Computer and Information Technology (CIT\'06) (2006).
[4] Liu, Hui, and Hee Beng K. Tan. "An Approach to Aid the Understanding and Maintenance of Input Validation." 22nd IEEE International Conference on Software Maintenance (ICSM\'06) (2006).
[5] Li, Nuo, Mao-Zhong Jin, and Chao Liu. "Web Application Model Recovery for User Input Validation Testing." International Conference on Software Engineering Advances(ICSEA 2007) (2007).
[6] Hayati, Pedram, Nastaran Jafari, S. M. Rezaei, and Saeed Sarenche. "Modeling Input Validation in UML." 19th Australian Conference on Software Engineering (2008).
[7] Lin, Jin-Cherng, Jan-Min Chen, and Cheng-Hsiung Liu. "An Automatic Mechanism for Adjusting Validation Function." 22nd International Conference on Advanced Information Networking and Applications - Workshops (2008).
[8] Park, Jaechul, and Bongnam Noh. "Web Attack Detection: Classifying Parameter." International Journal of Web Services Practices 2 (2006): 68-74.
[9] OWASP. “OWASP Top 10 The Ten Most Critical Web Application Security Vulnerabilities”. November, 2007.
[10] Offutt, Jeff, Ye Wu, Xiaochen Du, and Hong Huang. "Bypass Testing of Web Applications." Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE’04) (2004).
[11] Sonntag, Michael. "Ajax Security in Groupware." Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'06) (2006).