A Black-Box Approach to Detect Vulnerabilities in Web Services Using Penetration Testing

M. I. P. Salas, *Member, IEEE* and E. Martins, *Member, IEEE*

***Abstract*— Web services work over dynamic connections among distributed systems. This technology was specifically designed to easily pass SOAP message through firewalls using open ports. These benefits involve a number of security challenges, such as Injection Attacks, phishing, Denial-of-Services (DoS) attacks, and so on. The difficulty to detect vulnerabilities ─before they are exploited─ encourages developers to use security testing like penetration testing to reduce the potential attacks. Given a black-box approach, this research use the penetration testing to emulate a series of attacks, such as Cross-site Scripting (XSS), Fuzzing Scan, Invalid Types, Malformed XML, SQL Injection, XPath Injection and XML Bomb. In this way, was used the soapUI vulnerability scanner in order to emulate these attacks and insert malicious scripts in the requests of the web services tested. Furthermore, was developed a set of rules to analyze the responses in order to reduce false positives and negatives. The results suggest that 97.1% of web services have at least one vulnerability of these attacks. We also determined a ranking of these attacks against web services.**

***Keywords*— Web Service, Penetration Testing, Cross-site Scripting (XSS), Fuzzing Scan, Invalid Types, Malformed XML, SQL Injection, XML Bomb, XPath Injection.**

I. INTRODUÇÃO

SERVIÇOS WEB, são aplicações de software modulares que podem ser descritos, publicados, localizados e invocados sobre uma rede, como a World Wide Web. Devido a sua natureza distribuída e aberta, eles são mais suscetíveis a riscos de segurança [1]. Além das ameaças tradicionais expostas em inumeráveis artigos [2], [3], [4], [5], tem-se que contar com as novas ameaças associadas às tecnologias e serviços, como SOAP e XML. Um exemplo são os Ataques de Injeção e Negação de Serviços (Denial-of-Services Attack, DoS), que estão entre os mais explorados em 2013, segundo o Open Web Application Security Project [2].

Os Ataques de Injeção modificam as requisições feitas pelos usuários alterando a mensagem SOAP. Já, os ataques de Negação de Serviço indisponibilizam o serviço fornecido, encaminhando grandes quantidades de tráfego ao servidor [4]. Em qualquer dos casos, os ataques procuram gerar uma resposta não robusta no serviço web.

Os testes de segurança permitem descobrir novas vulnerabilidades e se antecipar aos riscos que podem acontecer. Um dos métodos mais utilizados são os testes de penetração, os quais permitem avaliar a segurança de um sistema ou rede, simulando um ataque a partir de uma fonte maliciosa [5]. Para isso, utiliza uma ferramenta denominada scanner de vulnerabilidades, cuja finalidade é automatizar a detecção de vulnerabilidades através da injeção de ataques no sistema alvo.

Nesta pesquisa utilizamos o scanner de vulnerabilidades soapUI junto com o add-on Security Testing para emular ataques e automatizar o envio de requisições aos serviços web. Os ataques emulados foram: SQL Injection, XPath Injection, Cross-site Scripting (XSS), Fuzzing Scan, Invalid Types, Malformed XML e XML Bomb.

Dada a abordagem de caixa preta (black-box), selecionamos uma amostra representativa de 69 serviços web. Esta amostra fornece um 90% de precisão com uma margem de ±10% de erro para cada tipo de ataque injetado. O objetivo é determinar o nível de vulnerabilidade dos serviços web contra os 7 ataques.

Foi desenvolvido um conjunto de regras para avaliar os resultados obtidos dos testes de penetração do soapUI. Os resultados sugerem que esta ferramenta tem uma baixa cobertura de vulnerabilidades e uma alta porcentagem de falsos positivos e negativos. Utilizamos um exemplo do ataque de Cross-site Scripting para descrever o processo de aplicação das regras de análise de vulnerabilidades.

O restante do artigo está organizado da seguinte forma: a Seção 2 descreve os desafios da segurança nos serviços web e o método de testes de penetração. Na Seção 3 são aplicados os testes de penetração aos serviços web com o scanner de vulnerabilidades soapUI. O desenvolvimento das regras de análise de vulnerabilidades em serviços web é descrito na seção 4. Na Seção 5 aplicamos as regras aos resultados obtidos dos testes de penetração. Seção 6 conclui o trabalho, mostrando suas principais contribuições, apontando novas direções para trabalhos futuros.

II. DESAFIOS DA SEGURANÇA NOS SERVIÇOS WEB

Em [1], Holgersson define os principais desafios relacionados a padrões e interoperabilidade nos serviços web. Ambos desafios enfatizam a relativa imaturidade desta tecnologia sobre abordagens de segurança, qualidade de serviços (Quality of Services –QoS), escalabilidade, entre outros. Ibrahim [6] classifica e agrupa os desafios da segurança, envolvendo ameaças, ataques e problemas de segurança, tais como:

- Ameaças ao nível de serviço: ataques a WSDL e UDDI por injeção de código malicioso (malware), suplantação de identidade, negação de serviços, falsificação de esquemas XML e sequestro/roubo de sessão.
- Ameaças ao nível de mensagem: ataques de manipulação, interceptação e reenvio de mensagens SOAP.

M. I. P. Salas, Universidade Estadual de Campinas (UNICAMP), Campinas, São Paulo, Brasil, marcelopalma@ic.unicamp.br

E. Martins, Universidade Estadual de Campinas (UNICAMP), Campinas, São Paulo, Brasil, eliane@ic.unicamp.br

Diversas pesquisas propõem metodologias e técnicas para melhorar a robustez dos serviços web contra ataques informáticos usando testes de penetração. Por exemplo, Os autores [7] desenvolvem uma ferramenta chamada WS-Attacker. Esta ferramenta avalia a resistência de 4 serviços web contra ataques de suplantação no WS-Addressing e SOAPAction. Em [8], os autores avaliam a segurança de 300 serviços web com 4 scanners de vulnerabilidades usando diversos tipos de ataques, concluindo que estas ferramentas apresentam um alto número de falsos positivos e baixa cobertura de ataques, entre outras limitações.

Em [5], os autores comparam testes de penetração com a técnica de análise de código estático para a detecção de vulnerabilidades usando o ataque de SQL Injection, os resultados sugerem que a segunda técnica melhora a detecção deste ataque e reduz os falsos positivos. Por em quanto, outras pesquisas utilizam WS-Security para mitigar possíveis ataques de segurança, e.g. em [9], os autores propõem uma arquitetura de middleware usando serviços web com WS-Security para possibilitar o envio e recepção de informação a través de uma rede de sensores sem fio (RSSF).

A. Técnica de Testes de Penetração

Os Testes de Penetração permitem emular ataques através de scanners de vulnerabilidades com o intuito de revelar possíveis vulnerabilidades. As vulnerabilidades detectadas variam de um scanner para outro. Existem diversos scanners, tanto comerciais (e.g. HP Web Inspect, IBM Rational AppScan) quanto de código aberto (e.g. soapUI, WSDigger e WebScarab) [8], [10].

B. Segurança em Serviços Web

Segurança é uma qualidade de sistema que garante a ausência de acesso ou manipulação não autorizada ao sistema [11]. Seus principais atributos são a confidencialidade (a informação só deve ser revelada para usuários autorizados), integridade (a informação não pode ser modificada por usuários não autorizados) e disponibilidade (o acesso ao sistema não pode ser negado, de forma maliciosa para usuários autorizados) [1].

As violações de segurança acontecem pela exploração de vulnerabilidades existentes no sistema. Vulnerabilidades são falhas (faults) introduzidas, intencional ou acidentalmente, durante o desenvolvimento do sistema. Existem inúmeras causas de vulnerabilidades, entre as quais podemos citar a complexidade dos sistemas, a falta de mecanismo para avaliar as entradas fornecidas. Um ataque explora as vulnerabilidades, de forma maliciosa ou não, podendo comprometer as propriedades de segurança. O resultado de um ataque bem sucedido é uma intrusão no sistema por uma vulnerabilidade encontrada [11]. A Fig. 1 ilustra esses conceitos.

Nesta seção nos concentramos nas ameaças e vulnerabilidades utilizadas na fase de testes de penetração.

SQL Injection, consiste em injetar uma consulta SQL na mensagem SOAP. Seu objetivo é acessar no banco de dados, ler a informação, alterá-la (inserir, atualizar e eliminar). Se o serviço não valida corretamente os dados, o ataque pode comprometer o banco de dados e o servidor [2].

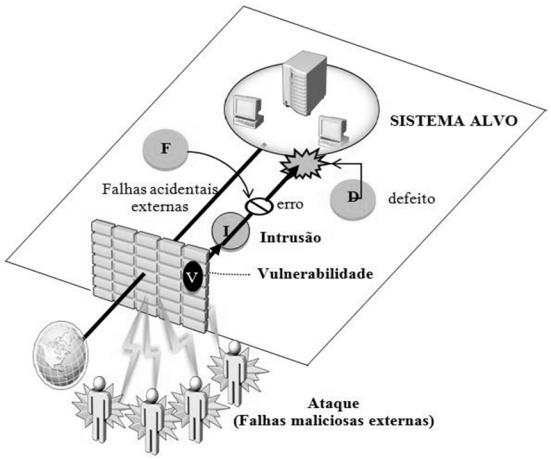

Figura 1. Ameaças à segurança dos serviços web [11].

XPath Injection, explora os comandos XPath para atacar o servidor através do envio de informações intencionalmente mal formadas. Isto pode levar a descobrir a estrutura de dados XML, elevar privilégios de usuários, modificar os documentos XML, entre outros [3], [4].

Cross-site Scripting (XSS), utiliza vulnerabilidades existentes no serviço web com o objetivo de injetar código malicioso no servidor através dos parâmetros descritos no WSDL. Dada a relação de confiança estabelecida entre o serviço web e o servidor, o primeiro assume que o código recebido é legítimo e, portanto, o segundo permite o acesso a informações confidenciais tal como o identificador de sessão. Com isso, o atacante envia requisições maliciosas ao serviço web e, por conseguinte, pode sequestrar a sessão, coletar informações e realizar transações não autorizadas [2], [4].

Fuzzing Scan (Fuzzing), este ataque gera entradas aleatórias no serviço web através dos parâmetros descritos no WSDL com a esperança de provocar algum tipo de erro imprevisto [12].

Invalid Types, o atacante envia valores que estejam fora dos limites esperados das variáveis descritas no WSDL, a fim de desencadear falhas no sistema alvo (crash) [2], [3], [12].

Malformed XML aproveita as vulnerabilidades do analisador XPath inserindo fragmentos mal formados de XML nas mensagens SOAP, deixando etiquetas abertas ou adicionando atributos e elementos não definidos. O ataque faz com que o serviço web exponha suas informações confidenciais e gere falhas no sistema alvo (crash) [12].

XML Bomb, sobrecarrega o analisador XPath explorando o fato que XML permite definir entidades e parâmetros dentro da requisição da mensagem SOAP, e.g. definir mais de 100.000 parâmetros, gerando uma requisição que fica armazenada na memória e gera alto consumo da CPU [13].

III. TESTES DE PENETRAÇCÃO EM SERVIÇOS WEB

Entre vários motivos para realizar um ataque a um serviço web, se destaca as invasões por questões financeiras, pessoais, cometer fraudes, sabotagem ou espionagem. O processo envolve uma análise nas atividades do sistema, que involucra a busca de vulnerabilidades em potencia. Esta seção descreve o procedimento de injeção de ataques por testes de penetração.

A. Arquitetura de Testes de Penetração

Nossa arquitetura utiliza o scanner de vulnerabilidades soapUI versão 4.5. Esta ferramenta utiliza o add-on Security Testing para enviar um conjunto de requisições maliciosas aos serviços web com o objetivo de: i) reproduzir diferentes tipos de ataques; ii) tentar provocar um comportamento não robusto; e iii) identificar e tratar as possíveis vulnerabilidades de segurança nos serviços web [12].

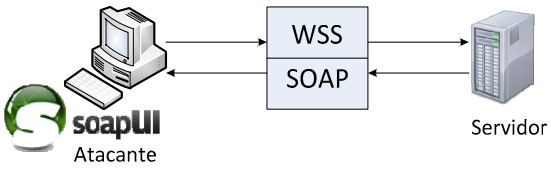

Figura 2. Arquitetura de testes de penetração.

A Fig. 2 descreve a arquitetura cliente (atacante)↔servidor para o envio e recepção de mensagens SOAP, as quais podem ser protegidas pela especificação WS-Security (WSS). O soapUI inicialmente envia uma requisição para detectar se o serviço web está ativo ou não. Em caso afirmativo, o add-on seleciona um ataque e gera os scripts em função dos parâmetros de entrada descritos no WSDL do serviço web.

A continuação, a ferramenta envia as requisições com os scripts de ataques. Para finalizar, o add-on coleta os resultados em logs ─em formato de texto─ e aplica regras para analisar a presença de vulnerabilidades nas respostas. Este procedimento foi realizado em cada serviço web (69) para os 7 ataques testados, fazendo um total de 483 ataques realizados.

B. Seleção da Amostra

Para determinar quão vulneráveis são os serviços web a Ataques de Injeção e Negação de Serviços, selecionamos uma amostra representativa de 69 serviços web de um conjunto de 22.272 serviços armazenados no Seekda. Seekda é uma UBR (Universal Business Registry) que registra os WSDL dos serviços web e descreve seus parâmetros (operações). A amostra fornece 90% de precisão com uma margem de ±10% de erro para avaliar a presença de vulnerabilidades.

Por questões de segurança, não descrevemos o WSDL de cada serviço web testado, mas detalhamos algumas das características dos mesmos: 46,38% dos serviços web são de uso comercial; 37,68% são repositórios de informação de áreas como música, meteorologia, entre outros; 14,49% apoiam as atividades acadêmicas e de pesquisas; 1,45% são serviços governamentais. Além disso, 86% dos servidores usam SO Microsoft Windows Server 2003/2008 com Internet Information Server (IIS) 6.0 ou superior, 10% usam SO Linux com o Servidor Apache; o 3% também usa SO Linux com o Servidor Jetty V6.1.17 e o 1,45% usa Windows 7 Professional Server com Glassfish V2.1.

C. Configuração e Execução do Add-On Security Testing

O próximo passo é a configuração do add-on Security Testing. Primeiramente criamos uma instancia de teste de segurança para cada serviço web dentro da ferramenta. A seguir selecionamos os ataques a ser injetados no serviço web.

Depois de executar os ataques, a ferramenta analisa os resultados de forma automatizada, retornando o numero de vulnerabilidades por ataque (ver Fig. 3). O usuário tem a possibilidade de criar um reporte de ataques, no entanto, a ferramenta armazena os resultados no seu banco de dados.

Figura 3. Exemplo de análise de resultados do add-on Security Testing.

D. Resultados dos Testes de Penetração

O add-on Security Testing gera um log de resultados que classifica as respostas das requisições em dois tipos: alertas, que detalham as possíveis vulnerabilidades detectadas no serviço web (FAILED), e sem alertas, quando o ataque não encontrou vulnerabilidades (VALID – OK). Esta fase retorna 35.087 respostas (logs), das quais 54,45% pertencem a ataques de Cross-site Scripting.

A ferramenta classificou 54,57% das respostas como alertas e 45,43% sem alertas. Segundo o soapUI, os ataques que têm mais vulnerabilidades detectadas são: 1º XML Bomb com 66,88%; 2º Cross-site Scripting com 65,93%; e 3º Malformed XML com 65,51%. Os demais dados estão descritos na Tabela I abaixo.

TABELA I. RESULTADOS DA INJEÇÃO DE ATAQUES PELO SCANNER DE VULNERABILIDADES SOAPUI.

Ataque	Alertas	%	Sem Alertas	%
XML Bomb	426	66,88%	211	33,12%
Cross-site Script.	12.598	65,93%	6.510	34,07%
Malformed XML	1.094	65,51%	576	34,49%
XPath Injection	920	50%	920	50%
Fuzzing Scan	2.818	47,76%	3.082	52,24%
SQL Injection	1.158	46,01%	1.359	53,99%
Invalid Types	1.363	39,91%	2.052	60,09%
Total = 35.087	**20.377**	**54,57%**	**14.710**	**45,43%**

Segundo o add-on Security Testing, 64 dos 69 (92,75%) serviços web apresentam ao menos uma vulnerabilidade a algum dos ataques emulados. A Tabela II analisa descreve o restante dos ataques.

TABELA II. REQUISIÇÕES POR ATAQUE PARA OS 69 SERVIÇOS WEB.

Ataque	Requisições por Ataque	% de vulnerabil. dos serviços web	%
Cross-site Scripting	276,93	64/69	92,75%
Malformed XML	24,2	57/69	82,61%
XML Bomb	9,23	48/69	69,57%
Invalid Types	49,49	31/69	44,93%
XPath Injection	26,67	30/69	43,48%
Fuzzing Scan	85,51	29/69	42,03%
SQL Injection	36,48	28/69	40,58%
Total	**72,64**	**64/69**	**97,75%**

A continuação, analisamos a presença de falsos positivos e negativos nos resultados obtidos nesta seção.

IV. AVALIAÇÃO DOS TESTES DE PENETRAÇÃO

Para conferir a presença de falsos positivos e negativos, inicialmente analisamos as assertivas do add-on Security Testing, descrita na Tabela III. O add-on analisa o conteúdo das respostas tentando encontrar informações sensíveis que forneçam indícios de vulnerabilidades encontradas, e.g. a assertiva de Sensitive Information Exposure verifica se a resposta contém informações sensíveis do sistema operacional como acesso aos diretórios sem autorização.

TABELA III. RESULTADOS DA INJEÇÃO DE ATAQUES PELO SCANNER DE VULNERABILIDADES SOAPUI.

Assertiva	Descrição
Invalid HTTP Status Codes	Verifica a existência de códigos de status HTTP inválidos nas respostas
Schema Compliance	Valida as respostas das requisições contra o esquema XML definido no WSDL
Simple Contains	Verifica a existência de uma cadeia nas respostas das requisições
Sensitive Information Exposure	Verifica se a última mensagem recebida não contém informação sensível do sistema operacional
SOAP Fault	Verifica se a resposta da requisição não contém uma falha SOAP
XPath Match	Compara o conteúdo das respostas das requisições contra uma expressão XPATH

Utilizamos duas ferramentas fornecidas pelo add-on Security Testing para analisar a presença de vulnerabilidades nos serviços web. A primeira são os logs gerados pelos testes de penetração. Usando o exemplo da injeção de Cross-site Scripting, a Fig. 4 descreve a existência de vulnerabilidades (FAILED) pela inserção do script xss.js (linha 4). O ataque levou 216 ms, retornando a mensagem Can give hackers information about which software or language you are using.

```
1  SecurityTest started at 2012-10-26 11:04:59.503
   Step 1 [ConversionRate - Request 1]
2  SecurityScan 2 [Cross Site Scripting]
   [Cross Site Scripting] Request 3 - FAILED -
3  [FromCurrency=<SCRIPT a=">"
   SRC="http://soapui.org/xss.js"></SCRIPT>]: took 216 ms
4  -> [Stacktrace] Can give hackers information about which software
   or language you are using - Token [(?s).*(s|S)tack ?(t|T)race.*] found
   [stack trace]
   -> [Stacktrace] Can give hackers information about which software or
   language you are using - Token [(?s).*at [\w\$]+(\.[\w\$<>\[\],]+|\.\.…
5
6
7
```

Figura 4. Log gerado pelo VS soapUI pela injeção de XSS.

A segunda ferramenta utilizada foi Message Viewer do soapUI. Esta ferramenta nos permite observar cada uma das 35,087 requisições e suas respostas.

Continuando com o exemplo de Cross-site Scripting, a Fig. 5 apresenta a requisição e a resposta deste ataque. Na linha 6 e 7 da requisição, observamos o script inserido na mensagem SOAP (em negrito). Este script é inserido na etiqueta <web:FromCurrency> com o objetivo que o servidor execute o javascript xss.js armazenado no servidor do soapUI. A linha 1 da resposta apresenta o código de status HTTP 500 Internal Server Error. Isto acontece porque o serviço web enviou um conjunto de etiquetas (da linha 5 à 9) descrevendo falhas no servidor (em negrito), i.e. <soap:Fault>, <faultcode>, <faultstring> ao processar a requisição.

	Requisição
1:	POST … HTTP/1.1
2:	Content-Type: text/xml;charset=UTF-8
3:	SOAPAction: "…"
4:	<soapenv:Envelope xmlns: "… ">
5:	<soapenv:Header/> <soapenv:Body> <web:ConversionRate>
6:	**<web:FromCurrency><SCRIPTa=">'>"SRC="http://soapui.o**
7:	**rg/xss.js"></SCRIPT></web:FromCurrency>**
8:	<web:ToCurrency>BOB</web:ToCurrency>
9:	</web:ConversionRate></soapenv:Body></soapenv:Envelope>
	Resposta
1:	**HTTP/1.1 500 Internal Server Error**
2:	<?xml version="1.0" encoding="utf-8"?><soap:Envelope
3:	xmlns:soap="...">
4:	<soap:Body>
5:	**<soap:Fault> <faultcode>soap:Client</faultcode>**
6:	**<faultstring>System.Web.Services.Protocols.SoapException**
7:	**: Server was unable to read request.**
8:	**...**
9:	**</faultstring> </soap:Fault>**
10:	</soap:Body></soap:Envelope>

Figura 5. Exemplo de Message Viewer gerado pela soapUI.

Na Tabela IV são descritos o comportamento dos códigos de status para ataques contra serviços web.

TABELA IV. LISTA DE CÓDIGOS DE STATUS HTTP PARA ATAQUES A SERVIÇOS WEB.

Códigos HTTP	Descrição
200	Padrão de resposta para requisições HTTP bem sucedida (Ok), ajudará a identificar a existência de uma **possível vulnerabilidade encontrada** quando o sistema executou a requisição sem detectar o script e, **ataque detectado** quando o sistema responde com uma mensagem robusta, descrevendo a existência de erro na requisição.
400	A requisição não pode ser cumprida devido a sintaxe ruim (Bad Request). Já que o servidor detectou um ataque, consideramos como uma **resposta robusta**.
500	O servidor não cumpriu com uma requisição aparentemente válida ou encontrou uma condição inesperada, impedindo de executar a requisição com êxito. Neste caso, o servidor responde com Internal Server Error. Analisamos a resposta do servidor usando o etiqueta <soap:Fault> dentro do corpo (*body*) da mensagem SOAP que fornece os erros e a informação de status da mensagem, contendo os seguintes elementos: • <faultcode> Código de identificação da falha. • <faultstring> Explicação descritiva da falha. • <faultactor> Informação de que/quem fez acontecer a falha. • <details> Informação que descreve o erro no servidor. Além disso, os valores de *faultcode* podem ser classificados em 4 tipos: • VersionMismatch: O servidor encontrou um *namespace* inválido no envelope da mensagem SOAP. • MustUnderstand: ausência de um elemento obrigatório no cabeçalho da mensagem SOAP. • Client: A mensagem enviada foi estruturada de forma incorreta ou contém informações incorretas para sua autenticação. • Server: A mensagem não é processada por um problema no servidor.

Através da utilização dos logs, requisições e respostas armazenados no add-on Security Testing, identificamos possíveis padrões de comportamentos dos serviços web que apresentam vulnerabilidades para os 7 ataques emulados. Este comportamento foi refletido em um conjunto de regras, as quais com base em diversas pesquisas [3], [11], [12], permitem classificar os serviços web em função da sua resposta a um determinado tipo de ataque (Ataque de Injeção e Negação de Serviços) utilizando seu código de status HTTP. As regras são descritas a seguir:

Regra 1. SE o cabeçalho contém o código "200 OK" E respondeu com uma mensagem SOAP descrevendo a existência de erro de sintaxe na requisição, ENTÃO não existe vulnerabilidade encontrada (VNE). CASO CONTRÁRIO, o usuário tem acesso a páginas não autorizadas OU é redirecionado para outro serviço web OU executou trechos de código no servidor (e.g. Java script), ENTÃO existe uma vulnerabilidade encontrada (VE).

Regra 2. SE o cabeçalho contém o código "400 Bad request message", e.g. request format is invalid: Missing required soap: Body element ENTÃO não há vulnerabilidades encontradas (VNE) no serviço web.

Regra 3. SE o cabeçalho contém o código "500 Internal Server Error" E houve divulgação de informação na mensagem SOAP (e.g. rotas de diretórios do servidor e do banco de dados, bibliotecas de funções e objetos, acesso a arquivos XML com usuários e senhas, entre outros), ENTÃO existe uma vulnerabilidade encontrada (VE), CASO CONTRÁRIO, se não houve divulgação de informação não há vulnerabilidades encontradas (VNE) no serviço web.

Regra 4. SE na ausência de ataques, o cabeçalho contém o código "500 Internal Server Error" E SE na presença de ataques, o cabeçalho contém o código "200 OK", ENTÃO existe uma vulnerabilidade encontrada (VE).

Regra 5. SE na ausência de ataques, o cabeçalho contém o código "500 Internal Server Error" E SE na presença de ataques, o cabeçalho contém o código "400 Bad request message", ENTÃO há vulnerabilidades encontradas (VE).

Regra 6. SE na ausência de ataques, o cabeçalho contém o código "500 Internal Server Error" E SE na presença de ataques, o cabeçalho contém o código "500 Internal Server Error" também, ENTÃO há vulnerabilidades encontradas (VE).

Regra 7. SE o servidor não responde (se considera como colapso ou crash), por consequência de um ataque de Negação de Serviços, ENTÃO é vulnerabilidade encontrada (VE), CASO CONTRÁRIO é uma falha de software (FS), porque não foi provocada pelo ataque, se não por um erro de software do sistema.

Regra 8. SE nenhuma das regras acima pode ser aplicada, ENTÃO o resultado é tido como inconclusivo, pois não há forma de confirmar se realmente existe uma vulnerabilidade no serviço web.

A facilidade para aplicar as regras de análise de vulnerabilidades –para Ataques de Injeção e Negação de Serviço– permite a qualquer usuário analisar de forma rápida e fácil a presença de vulnerabilidades nos serviços web. As regras 4, 5 e 6 analisam a seguinte possibilidade: em ausência do envio do ataque apresenta no cabeçalho o código de status HTTP "500 Internal Server Error", no entanto, quando enviamos as mensagens SOAP com ataques, o serviços web gera novas respostas, as quais são analisadas pelas regras citadas.

Na regra 7, classificamos a resposta segundo o tipo de ataque. Se for um ataque de Negação de Serviços, o atacante conseguiu seu objetivo (VE). Caso contrário, o ataque é uma consequência dos problemas do serviço web (inconclusivo). A regra 8 é uma exceção ao restante das regras, para o caso em que nenhuma das outras regras possa classificar o resposta, classificamos como inconclusivo. Se o testador tivesse uma abordagem de caixa branca, as presentes regras poderiam ser refinadas para determinar melhor o acontecido no servidor ao ser recebido uma mensagem SOAP corrompida.

V. APLICAÇÃO DAS REGRAS AOS TESTES DE PENETRAÇÃO

Inicialmente analisamos os logs e as respostas para verificar se soapUI identificou corretamente as vulnerabilidades no serviço web. A partir da utilização das regras desenvolvidas na anterior seção, classificamos as respostas em 6 categorias:

- Vulnerabilidade encontrada (VE): Detecção correta de uma vulnerabilidade (FAILED) no serviço web por parte do add-on Security Testing.
- Vulnerabilidade não encontrada (VNE): detecção correta de uma resposta robusta (VALID – OK).
- Falso positivo (FP): Detecção incorreta de uma vulnerabilidade que realmente não aconteceu.
- Falso negativo (FN): A não detecção de uma vulnerabilidade, que realmente aconteceu.
- Falha de software (FS): Quando o erro não foi provocado pelo ataque, senão por um erro de software do servidor.
- Inconclusivo: resposta não possível de ser classificada.

A continuação, apresentamos os resultados utilizando o procedimento descrito na seção anterior.

Os resultados descritos na Tabela V registram 35.087 respostas classificadas por ataques e por: vulnerabilidades encontradas (VE), vulnerabilidades não encontradas (VNE), falsos positivos (FP) e falsos negativos (FN). Não se apresentaram Falhas de Software (FP) nem respostas inconclusivas.

TABELA V. CLASSIFICAÇÃO DAS RESPOSTAS POR ATAQUES.

Ataque	Total	VE	VNE	FP	FN
Cross-Site Scr.	**19.108**	19,5%	8,03%	46,43%	26,04%
Fuzzing Scan	**5.900**	42,56%	33,98%	5,2%	18,25%
Invalid Types	**3.415**	35,78%	30,31%	4,13%	29,78%
SQL Injection	**2.517**	42,15%	27,81%	3,85%	26,18%
XPath Inject.	**1.840**	45,71%	28,26%	4,29%	21,74%
Malfor. XML	**1.670**	19,64%	10,36%	45,87%	24,13%
XML Bomb	**637**	26,22%	7,54%	40,66%	25,59%
TOTAL	**35.087**	**28,09%**	**17,14%**	**29,98%**	**24,78%**

Os resultados sugerem que o SoapUI apresente uma grande porcentagem de falsos positivos (29,98%) e falsos negativos

(24,78%). Somando ambos, representam quase dois terços (57,76%) do total dos testes. Os ataques com maior número de falsos positivos são Cross-Site Scripting (46,43%), Malformed XML (45,87%) e XML Bomb (40,66%).

Um serviço web é declarado vulnerável se ao menos apresenta uma vulnerabilidade (ataque bem sucedido) ao determinado ataque. A Tabela VI, apresenta o nível de vulnerabilidade dos serviços web frente a cada ataque injetado. Para isso, coletamos as respostas com vulnerabilidades encontradas (VE) e falsos negativos (FN) em cada serviço. Lembramos que ±10% representa o nível de confiabilidade dos resultados.

TABELA VI. PRINCIPAIS ATAQUES QUE ATINGEM OS SERVIÇOS WEB.

Ataques	Web Services Vulneráveis (VE+FN)/69	Porcentagem de vulnerabilidade
Cross-site Script.	58/69	84,06% ± 10%
Malformed XML	54/69	78,26% ± 10%
Invalid Types	48/69	69,57% ± 10%
SQL Injection	40/69	57,97% ± 10%
XPath Injection	40/69	57,97% ± 10%
Fuzzing Scan	38/69	55,07% ± 10%
XML Bomb	29/69	42,03% ± 10%
TOTAL	**67/69**	**97,10%** ± 10%

Os resultados sugerem que o 97,10% dos serviços web testados apresentam ao menos uma vulnerabilidade ao algum tipo de ataque emulado, i.e. 67 dos 69 serviços têm vulnerabilidades. Comparado com o resultado do soapUI (92,75%), nossa abordagem aperfeiçoa a procura de vulnerabilidades nos serviços web. Por último, as vulnerabilidades ao ataque de Cross-Site Scripting estão presentes em 58 dos 69 serviços (84,06% ±10%). No entanto, o ataque de Malformed XML pode estar presente até em 88,26% dos serviços web testados, i.e. 78,26% +10%.

VI. CONCLUSÕES E TRABALHOS FUTUROS

Os testes de penetração podem ser classificados como um método de auditoria de segurança, já que simulam ataques com o intuito de mensurar o impacto da varredura caso seja bem sucedido e seja descoberta a falha (bug).

Nossa abordagem teve o objetivo avaliar os resultados do scanner de vulnerabilidades soapUI com o add-on Security Testing através da injeção de 7 ataques contra 69 serviços.

Cada resposta foi avaliada com um conjunto de regras de análise e detecção de vulnerabilidades para os Ataques de Injeção (Cross-site Scripting, Fuzzing Scan, Invalid Types, Malformed XML, SQL Injection, XPath Injection) e Negação de Serviços (XML Bomb).

Os resultados sugerem que o scanner de vulnerabilidades soapUI apresenta alta porcentagem de falsos positivos, falsos negativos e uma baixa cobertura de vulnerabilidades existentes, as quais podem ser melhoradas utilizando nossa abordagem. Além, 97,10% dos serviços web testados apresentam vulnerabilidades ao menos a um dos tipos de ataques emulados.

Em trabalhos futuros analisaremos os ataques de Negação de Serviços contra serviços web e automatizaremos a análise através de técnicas de aprendizagem de máquinas para determinar se um serviço web tem vulnerabilidades ou não.

AGRADECIMIENTOS

Primeiramente, desejo agradecer ao CNPq, Samsung e ao Instituto de Computação (IC) da Universidade Estadual de Campinas (UNICAMP) por financiar e apoiar a pesquisa. Também, reconhecer ao Prof. Paulo Lício Geus do IC-UNICAMP, por sua colaboração ao longo do presente trabalho.

REFERÊNCIAS

[1] J. Holgersson, E. Soderstrom. Web Service Security-Vulnerabilities and Threats within the Context of WS-Security. SIIT 2005, ITU.
[2] J. Williams, D. Wichers. OWASP Top 10 – 2013. OWASP Foundation. Disp. em: https://www.owasp.org/index.php/Top_10_2013.
[3] J. Meiko et al. A Survey of Attacks on Web Services. Computer Science - Research and Development, Springer Berlin; 2009.
[4] D. Rodrigues, J.C. Estrella, Branco, M. Vieira. Engineering Secure Web Services. In: Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions. IGI Global. Jul 2011.
[5] N. Antunes, M. Vieira. Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services. Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium.
[6] MI. Ladan. Web Services: Security Challenges. In: Proceedings of the World Congress on Internet Security, 2011. WorldCIS'11. IEEE Press; Londres, Reino Unido, 21-23 Feb. 2011.
[7] C. Mainka. Penetration Testing Tool for Web Services Security. Services (SERVICES), 2012 IEEE Eighth World Congress on Date of Conference: 24-29 June 2012.
[8] M. Vieira et al. Using Web Security Scanners to Detect Vulnerabilities in Web Services. In: Proc. IEEE/IFIP DSN '09. Lisbon, Portugal, 2009.
[9] R.W. Scherer, J.H. Kleinschmidt. A Middleware Architecture for Wireless Sensor Networks Using Secure Web Services, Latin America Transactions, IEEE (Revista IEEE America Latina), Sept. 2011.
[10] J. Fonseca et al. Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks. Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium.
[11] A. Avizienis et al. Dependability and ist Threats: A Taxonomy. In: Jacquart, R.l eds. Proceedings IFIP 18th World Computer Congress.
[12] Eviware soapUI; Web Services Testing tool – Security Testing V4.5.
[13] YS. Loh et al. Design and Implementation of an XML Firewall. In: Proc. of the International Conference on Computational Intelligence and Security. IEEE Press: Guangzhou, China, 3-6 Nov. 2006.

Marcelo Invert Palma Salas possui uma maestria em Ciências da Computação pela Universidade Estadual de Campinas (2012), graduação em Engenharia de Sistemas pela Escuela Militar de Ingenieria (2005). Trabalhou na Price Water House Coopers (2005) como auditor de Segurança da Informação e Projeto de Analise de Comportamento de Malware em Móbiles para a Samsung. Fundador e Presidente do Capítulo Profesional da IEEE Computer Society na Seção Bolivia. Foi Chefe das Atividades Estudianties para a IEEE Região 9. Professor e Conselheiro do Departamento de Sistemas da Escuela Militar de Ingenieria (EMI). Atualmente, aluno de doutorado do Instituto de Computação da Universidade Estadual de Campinas.

Eliane Martins possui graduação em Matemática Modalidade Informática pela Universidade Federal do Rio de Janeiro (1976), mestrado em Engenharia de Sistemas e Computação pela Coordenação dos Programas de Pós-Graduação em Engenharia da Universidade Federal do Rio de Janeiro (1982) e doutorado em Informática - Ecole Nationale Supérieure de Aeronautique et de Espace (1992). Atualmente é professor associado da Universidade Estadual de Campinas. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: injeção de falhas por software, testes baseados em modelos; ferramentas de teste de software.