Classificação Automática de *Cross-Site Scripting* em Páginas *Web*

Eduardo Nunan, Eduardo Souto, Eulanda M. dos Santos, Eduardo Feitosa
Instituto de Computação
Universidade Federal do Amazonas - UFAM
Av. Gal. Rodrigo Octávio Jordão Ramos, 3000
CEP 69.077-000 – Manaus – AM – Brasil
eduardonunan@gmail.com {esouto, emsantos, efeitosa}@icomp.ufam.edu.br

Abstract. *The structure of dynamic websites, which comprises a set of objects such as HTML tags, script functions, hyperlinks and advanced features in browsers, provides several resources and interactiveness in services currently offered on the Internet. However, these features also increase the security risks and attacks caused by malicious codes injection or Cross-Site Scripting (XSS). This kind of attack has been at the top of the lists of the greatest threats to web applications in recent years. This paper presents the experimental results achieved on XSS automatic classification in web pages using Machine Learning techniques. The work has been concentrated on features extracted from web document content and URL. The results demonstrate that the proposed features lead to highly accurate classification of malicious page.*

Resumo. *A estrutura de páginas web dinâmicas, constituída de um conjunto de objetos,tais como, tags HTML, funções script, hyperlinks e recursos avançados em navegadores web, propiciaram inúmeras funcionalidades e interatividade dos serviços oferecidos atualmente na Internet. No entanto, esses recursos têm aumentado o potencial de riscos de segurança e ataques resultantes de injeção de código malicioso por Cross-Site Scripting (XSS), que permanece no topo das listas das maiores ameaças para aplicações web nos últimos anos. Este artigo apresenta os resultados experimentais obtidos na classificação automática de XSS em páginas web usando técnicas de aprendizagem de máquina. O trabalho foi concentrado em recursos extraídos do conteúdo do documento web e URL. Os resultados demonstram que as características propostas produzem alta precisão na classificação de páginas maliciosas.*

1. Introdução

As aplicações *web* dinâmicas desempenham um importante papel no provimento e manipulação de recursos e na interação entre clientes e servidores. As funcionalidades atualmente suportadas pelos navegadores *web*, como *tags* HTML, *scripts* e recursos avançados têm aumentado as oportunidades de negócio, propiciando maior interatividade nos serviços oferecidos via *web*, tais como e-*commerce*, *Internet Banking*, redes sociais, *blogs*, fóruns, entre outros. Por outro lado, tais funcionalidades também potencializaram vulnerabilidades e ameaças de exploração maliciosa.

De acordo com Wasserman & Su [2008], a maior parte das linguagens de programação *web* não fornece, por padrão, uma passagem de dados segura para o cliente. A falta desse procedimento pode viabilizar um dos ataques mais freqüentes em aplicações *web*, o *Cross-Site Scripting* (XSS), descrito por Uto & Melo [2009] como um ataque que explora uma aplicação *web* vulnerável, usada para transportar código malicioso, normalmente escrito em *JavaScript*, até o navegador de outro usuário. O ponto central dessa vulnerabilidade está na falta de validação dos dados de entrada do usuário [OWASP, 2008], [Uto & Melo, 2009].

As pesquisas demonstram que XSS se mantém no topo das listas das maiores vulnerabilidades em aplicações *web* nos últimos anos [OWASP, 2010]. Diante do grande volume e abrangência dos ataques de XSS, diferentes abordagens e técnicas têm sido empregadas na elaboração de soluções, dentre as quais se destacam o uso de linguagens formais e autômatos, primitivas de marcação de elementos da estrutura estática do documento *web*, listas negras, listas brancas, soluções baseadas em *proxy*, aprendizagem de máquina, combinação de técnicas, entre outras.

Este trabalho apresenta um método de classificação automática de XSS em páginas *web* baseado em técnicas de aprendizagem de máquina supervisionadas, que apresenta como vantagem a descoberta de padrões a partir de um conjunto de fatos ou observações rotuladas, induzindo a máquina a um processo de aprendizagem e que têm sido empregadas com bons resultados em aplicações *web* [Chandola et al., 2009].

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os conceitos relativos ao XSS e discute alguns trabalhos relacionados; a Seção 3 descreve as características e o método proposto para a classificação automática de ataques de XSS em páginas *web*; a Seção 4 apresenta a análise dos resultados dos experimentos; e, por fim, a Seção 5 apresenta as conclusões e possíveis trabalhos futuros.

2. *Cross-Site Scripting* (XSS)

Grossman [2007] define *Cross-Site Scripting* (XSS) como um vetor de ataque causado por *scripts* maliciosos no lado cliente ou servidor, em que os dados de entrada do usuário não são adequadamente validados, permitindo o roubo de informações confidenciais e sessões do usuário, além de comprometer o navegador *web* cliente e a integridade do sistema em execução. Tipicamente, os códigos *script* usados em XSS são geralmente desenvolvidos na linguagem *JavaScript* e inseridos na estrutura HTML [Grossman, 2007], [Uto & Melo, 2009]. Contudo, tecnologias como *Active* X, *Flash* ou outra qualquer suportada pelo navegador *web* também podem ser empregadas como vetor [Grossman, 2007]. Cabe ressaltar que este trabalho foca na linguagem *JavaScript*.

Os ataques de XSS são categorizados como Persistentes, Reflexivos [Grossman, 2007] e DOM-*Based* [Klein, 2005]. No Persistente, o código malicioso é permanentemente armazenado em recursos do servidor, configurando-se no tipo mais perigoso [OWASP, 2008]. No Reflexivo, o código é executado de forma reflexiva no navegador *web* cliente, não necessitando estar armazenado no servidor. Este ataque é viabilizado normalmente por meio de *links* com injeção de código malicioso. Segundo o OWASP (*Open Web Application Security Project*) [OWASP, 2008], este é o tipo mais freqüente de ataque de XSS. Diferente dos dois tipos de ataques apresentados, em que o código malicioso está incorporado na página que é retornada para o navegador *web*

cliente, o tipo DOM-*Based* habilita os *scripts* dinâmicos para referenciar componentes do documento, modificando o ambiente DOM (*Document Object Model*) [OWASP, 2008]. De acordo com Klein [2005], a identificação de XSS DOM-*Based* requer a execução ou simulação do preenchimento dos objetos DOM. Ao analisar o conteúdo da URL e do documento *web*, este trabalho abrange a detecção de XSS Reflexivos e Persistentes.

Para melhor exemplificar, a Figura 1 apresenta uma visão geral dos ataques de XSS. Em uma das ações, o atacante (*hacker*) pode inserir um *script* malicioso, que fica armazenado de forma permanente (XSS Persistente) em um servidor *web* vulnerável. Em outro ataque o *hacker* pode enviar uma mensagem de *e-mail* para o usuário alvo, contendo um *link* com o código *script* malicioso (XSS Reflexivo) para a URL do *site* legítimo, hospedado em um servidor *web* vulnerável. Em ambos os ataques, o usuário final recebe a página *web* solicitada e o navegador *web* interpreta e executa o conteúdo da página e o código *script* malicioso que foi injetado.

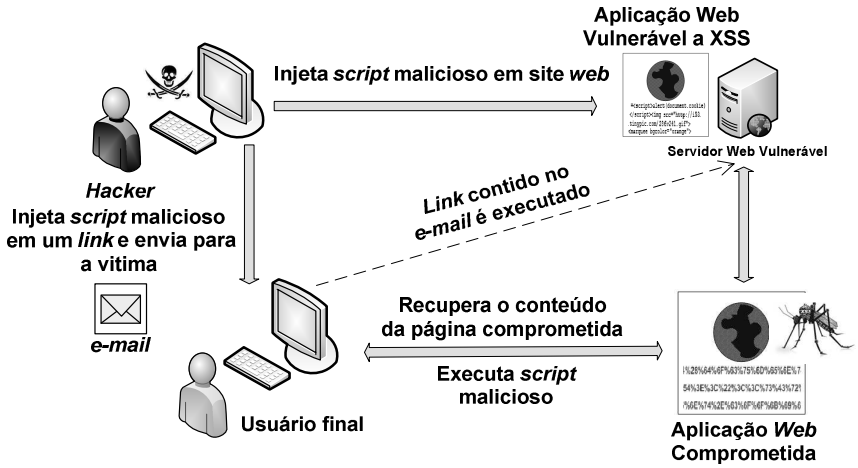

Figura 1. Visão geral de um ataque de XSS

2.1. Trabalhos Relacionados

As abordagens propostas para identificação e classificação de ataques de XSS podem ser categorizadas de acordo com o paradigma empregado na análise [Ernst, 2003], [Saha, 2009]. Na análise estática, o código do programa e as razões sobre todas as possibilidades de comportamento são analisadas antes da execução do mesmo [Ernst, 2003]. A análise dinâmica atua na observação do comportamento pela execução do código do programa. Este trabalho emprega a análise estática, que em termos gerais, possui a vantagem de apresentar o menor tempo de execução na análise quando comparado à análise dinâmica [Ernst, 2003].

Abordagens estáticas

Nadji et al. [2009] propuseram a aplicação da técnica de serialização mínima, em que são empregados sufixos e prefixos pertencentes a uma gramática livre de contexto (CFG - *Context Free Grammar*) para marcar os nós não confiáveis da estrutura estática do documento *web* e prevenir que ações maliciosas nesses nós alterem a estrutura do documento no cliente *web*. A abordagem proposta foi capaz de detectar 98,41% dos 5.353 ataques de XSS avaliados. Wasserman & Su [2008] utilizaram técnicas de

linguagem formal para analisar a sintaxe dos dados de entrada e comparar com uma lista negra contendo uma base de *scripts* não confiáveis que invocam o interpretador *JavaScript*.

Choi et al. [2011] propuseram o uso de técnicas de aprendizagem de máquina e N-*gram* - uma subsequência contígua de *n tokens*, em uma sequência consecutiva de *tokens* [Choi et al., 2011], para extrair características usadas na classificação conjunta de ataques XSS e SQL *Injection*. Usando uma base composta por 283 ataques de XSS e SQL *Injection*, os experimentos obtiveram a taxa de 98,04% de precisão usando o classificador SVM (*Support Vector Machines*). Song et al. [2009] propuseram o uso de modelos de cadeias de Markov com N-*gram*, em que modelos de diferentes tamanhos são submetidos ao algoritmo EM (*Expectation Maximization*) para agrupar padrões que permitam a distinção entre entradas de *script* legítimas na camada *web* e ataques SQL *Injection*, *Buffer Overflows* e XSS. Usando uma base com 130 ataques de XSS, os experimentos obtiveram 99% de taxa de acurácia usando 05 cadeias de Markov e 10-*gram*. Likarish et al. [2009] empregaram técnicas de aprendizagem de máquina para propor modelos para a detecção de código *JavaScript* malicioso em páginas *web*, usado em ataques de XSS e distribuição de *malware* (vírus, *worm*, *trojan*, etc.), com base na extração de *tokens* (*unigrams* ou *bigrams*) que representam características baseadas em ofuscação de código malicioso. Usando uma base composta por 62 *scripts* maliciosos, os experimentos obtiveram a taxa de 92,0% de verdadeiro positivo usando SVM.

Abordagens dinâmicas

Jim et al. [2007] apresentam uma abordagem baseada em políticas, que implementa uma função que analisa cada *script* com base em argumentos de entrada do texto do elemento DOM (*Document Object Model*) em que o *script* está inserido e do evento para o qual este deverá ser instalado como uma função que invoca o interpretador *JavaScript*. Kirda et al. [2006] propuseram uma solução baseada em *proxy*, denominada Noxes, que atua como um *firewall* pessoal para aplicações *web* analisando as requisições HTTP, permitindo as conexões com base na política de segurança definida por meio de regras, lista negra e pela ação do usuário.

2.2. Discussão

Saha [2009] avaliou 10 importantes trabalhos propostos na literatura para detectar *cross-site scripting*, em que 07 metodologias propostas apresentaram baixa capacidade para resolver ou detectar ataques XSS, dentre os quais, o trabalho de Wasserman & Su [2008]. O mesmo estudo avaliou 08 metodologias sob o aspecto de falso positivo, sendo que 06 metodologias apresentaram taxas altas e médias de falso alarme, dentre os quais o trabalho de Kirda et al. [2006]. Além disso, muitas propostas apresentam limitações práticas, por dependerem de forte aderência as políticas de mesma origem entre clientes e servidores ou de alterações significativas no código fonte dos atuais navegadores *web*, como por exemplo, os trabalhos de Nadji et al. [2009] e Jim et al. [2007].

Muitos métodos desconsideram as características de ofuscação, como por exemplo, no trabalho de Choi et al. [2011], em que é demonstrado apenas a extração de *tokens* em ASCII (*American Standard Code for Information Interchange*), o que permite ao invasor evadir muitas técnicas de detecção ao empregar a ofuscação de código. Song et al. [2009] incluem uma etapa de normalização para obtenção dos *tokens*, porém, não

usam as características que podem ser extraídas desse conjunto. Likarish et al. [2009] demonstram em seu trabalho a relevância desse tipo de características, contudo, de forma oposta aos outros trabalhos, esses autores desconsideram muitas características que podem ser obtidas pela decodificação e que podem revelar esquemas potencialmente inseguros empregados em ataques de XSS.

Outro ponto a ser considerado é a variabilidade de ameaças. O OWASP [2010] categoriza várias ameaças em virtude do risco, tais como, XSS, SQL *Injection*, *Buffer Overflow*, LDAP *Injection*, *Drive-by-download*, entre outras, o que revela que abordagens gerais precisam lidar com um alto grau de complexidade.

Em comparação com os trabalhos que empregam técnicas de aprendizagem de máquina, o principal diferencial deste trabalho está no método proposto que permite a extração de características relevantes da URL e documento *web* baseadas na ofuscação de código, comumente desconsiderado por outros métodos, e de características obtidas após a sua decodificação. Além disso, este trabalho explorou toda a base de ataques XSS que se tem conhecimento público (http://www.xssed.com), fornecendo ao classificador mais de 15.000 exemplos de treino, permitindo inferir maior acurácia no resultado.

3. Classificação Automática de Ataques de XSS em Páginas *Web*

Como mencionado anteriormente, o método proposto utiliza a análise estática do conteúdo dos elementos que compõem o documento *web* e URL, a partir da extração de características representativas dos ataques de XSS de maior ocorrência. Nesta seção são apresentadas as características selecionadas para diferenciar as páginas benignas de páginas infectadas por código XSS e o método adotado para classificação automática de ataques de XSS em páginas *web.*

3.1. Características

A seguir, são apresentadas as características propostas para diferenciar o conjunto de instâncias infectadas por código XSS, das instâncias benignas. Estas características são extraídas do documento *web* e URL para um vetor e submetido ao algoritmo de classificação. As características são categorizadas em classes baseadas em ofuscação, em esquemas HTML/*JavaScript* e em padrões suspeitos.

Características baseadas em ofuscação: esta classe é composta por 02 características e 08 atributos que correspondem às codificações alternativas, comumente utilizadas para ofuscar código malicioso e contornar filtros de validação de entrada de dados, restrições de acesso, causar a negação de serviço de uma aplicação ou ainda, viabilizar o acesso a lista de diretórios, a disseminação de *malware*, ou facilitar a exploração de vulnerabilidades e recursos locais. [Likarish et al., 2009], [CAPEC-72, 2011]. Likarish et al. [2009] empregaram características baseadas em ofuscação com bons resultados em seu trabalho. As características desta classe são descritas a seguir:

***Tamanho da URL (URL_Lenght)*:** extrai a quantidade de caracteres presentes na URL. Esta característica foi usada com bons resultados na distinção entre URLs maliciosas e benignas na detecção de *Phishing* por Ma et al. [2009]. Neste trabalho, esta característica foi empregada para determinar as URLs potencialmente ofuscadas, pois diversos ataques de XSS por meio da URL normalmente apresentam códigos maliciosos

ofuscados, que possuem um padrão de tamanho comumente maior que URLs sem codificação, conforme pode ser observado na Figura 2.

Código Ofuscado (Encoded): esta característica apresenta 07 atributos que correspondem às cadeias de caracteres ofuscados por codificações alternativas: Hexadecimal, Decimal, Octal, *Unicode*, Base64, caracteres de referência HTML e HTML *Name*. Por exemplo, considera-se uma cadeia de caracteres codificada em Base64, aquela que apresenta uma sequência superior a 40 caracteres na faixa [a-zA-Z0-9], incluindo caracteres "+, /" e um possível sufixo "=" ou "==" no fim do texto. Apesar de ser encontrada também em código *JavaScript* não malicioso, a ofuscação de código malicioso por meio de codificações alternativas é largamente usada em ataques do tipo XSS [CAPEC-72, 2011], [Likarish et al., 2009].

```
1 http://www.siteconfiavel.com/search.html?type=<"<<sCrIpT>alert(document.cookie)</sCrIpT><"<<sCrIpT>
2 alert(document.cookie)</sCrIpT>
3
4 http://www.siteconfiavel.com/search.html?type=%3C%22%3C%3C%73%43%72%49%70%54%3E%61%6C%65
5 %72%74%28%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%29%3C%2F%73%43%72%
6 49%70%54%3E%3C%22%3C%3C%73%43%72%49%70%54%3E%61%6C%65%72%74%28%64%6F%63%75
7 %6D%65%6E%74%2E%63%6F%6F%6B%69%65%29%3C%2F%73%43%72 %49%70%54%3E
```

Figura 2. Exemplo de ataque usando código malicioso ofuscado

A Figura 2 ilustra um ataque via URL que exibe um *popup* de alerta com o *cookie* atual para o site acessado, empregando codificação alternativa [CAPEC-72, 2011]. O código malicioso inserido após **type=** (linhas 1 e 4), em formato ASCII (linhas 1 e 2) é convertido pelo atacante para o formato Hexadecimal (linha 4 a 7).

Características baseadas em padrões suspeitos: esta classe é composta por 03 características e de um conjunto de atributos que compõem padrões suspeitos.

***Quantidade de Domínios (URL_Chain)*:** esta característica corresponde à quantidade de domínios encontrados na URL. Ataques do tipo *Redirect* apresentam URLs em cadeia [Grossman, 2007], que são inseridas para redirecionar a vítima para páginas armazenadas em servidores de atacantes com o objetivo de executar recursos externos com conteúdo malicioso, como por exemplo, arquivos ".js" que podem conter código *JavaScript* malicioso ou para transferir *cookies* ou informações críticas pela manipulação de esquemas potencialmente perigosos. Por exemplo: *www.sitebenigno.com/redir.php?url=http://www.sitemalicioso.com./*. Esta URL apresenta dois domínios ".com" indicando a presença de uma outra URL. Neste caso, inserida via "redir.php?url=" para redirecionar o usuário para o site do atacante.

***Caracteres Especiais Duplicados (Doubled_Char)*:** corresponde à identificação de uma cadeia de caracteres especiais mal formados, inseridos nas aberturas e fechamentos de *tags* [CAPEC-245, 2010], freqüentemente encontrados em ataques que têm por objetivo burlar filtros anti-XSS, pois muitos navegadores *web* ignoram os caracteres extras e corrigem o código de forma automática, permitindo a sua execução. A presença desse recurso indica a potencial existência de código malicioso XSS. Exemplo: *<"<<sCrIpT>alert(document.cookie)</sCrIpT><"<sCrIpT>alert(document.cookie)<" </sCrIpT>*.

Palavras Chaves* (*KeyWords): corresponde as palavras chaves comumente encontradas em redirecionamentos de página, disseminação de *malware* e ataques *phishing* associados a ataques de XSS. [Grossman, 2007].

Características baseadas em esquemas HTML/*JavaScript*: esta classe é composta por uma característica e um conjunto de atributos que são frequentemente usados em associação com esquemas potencialmente perigosos [Yue & Wang 2009].

***Esquemas HTML/JavaScript (Tags_Scheme)*:** esta característica identifica a presença de esquemas potencialmente vulneráveis à execução de código malicioso, como por exemplo, *tags* HTML: <*script*>, <*iframe*>, <*meta*> e <*div*>; propriedades HTML: *href, http-equiv* e *lowsrc; EventHandlers*: *onclick*, *onmouseover* e *onload*; objetos DOM: *Windows*, *Location* e *Document*; propriedades do objeto DOM: *Cookie*, *Referrer* e *InnerHtml;* e métodos *JavaScript: write* ( ), *getElementsByTagName* ( ), *alert* ( ), *eval* ( ), *fromCharCode* ( ), etc.

Os esquemas que executam conteúdos dinâmicos são recursos importantes que provêem funcionalidades e maior interação para o usuário, entretanto, muitos desses esquemas são freqüentemente explorados em ataques de XSS. No trabalho de Yue & Wang [2009] os autores apresentaram um estudo de caracterização de práticas inseguras com o uso da linguagem *JavaScripts* em páginas *web*. Jim et al. [2007], também fazem uso desses esquemas como argumentos de entrada para uma função que analisa a execução de *scripts.* O W3C *Consortium* relaciona em seu site (http://www.w3c.org) uma série de esquemas que invocam o interpretador *Javascript* e por fim, o *Scripting Mapping Project* [Gaucher et al., 2007] relaciona uma série de *EventHandlers* especificados pela combinação de elementos e atributos HTML que podem executar *scripts* sem o uso da *tag* <*script*> </*script*>. Por exemplo, o elemento HTML *img* pode executar um *script* ao ser especificado com o atributo *onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onclick* ou *ondblclick.*

3.2. Método

Para realizar a classificação automática de ataques de XSS em páginas *web*, foi elaborado o método ilustrado na Figura 3, obtido a partir do estudo das características propostas.

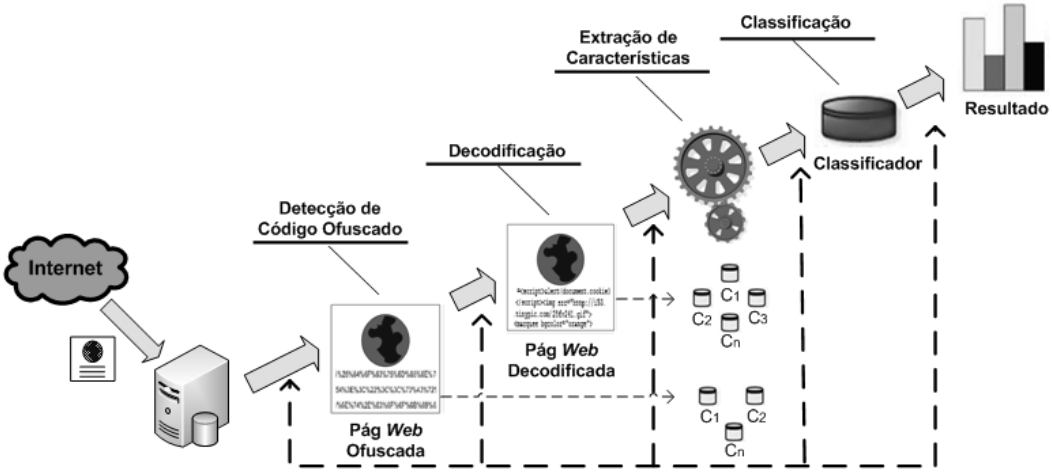

Figura 3. Método empregado na classificação automática de XSS

O método está estruturado nas seguintes etapas: detecção e extração de características de código ofuscado, decodificação da página *web*, extração de características decodificadas e classificação de páginas *web*. O método é executado, em um servidor, em nível de aplicação (*application level gateway*). A evolução do conteúdo das páginas *web* e o tratamento de novas codificações são alcançados a partir de novos treinamentos do classificador.

Etapa de Detecção e Extração de Características de Código Ofuscado: Após a obtenção da página *web* é realizada a detecção de ofuscação de código em seu conteúdo, nas codificações Hexadecimal, Decimal, Octal, *Unicode*, Base64, caracteres de referência HTML e HTML *Name*. Após a detecção das cadeias de caracteres ofuscados são extraídas as características baseadas em ofuscação, descritas na Seção 3.1. Neste trabalho são consideradas somente as codificações alternativas equivalentes à faixa de caracteres abrangida pela codificação ASCII e ASCII estendido.

Etapa de Decodificação da Página *Web*: após a execução da etapa anterior, a página *web* é então decodificada por rotinas que realizam a conversão de codificações alternativas para o formato ASCII, a fim de tornar o texto inteligível e viabilizar a análise e extração das demais características. Muitos métodos desconsideram o emprego de técnicas de ofuscação no código malicioso, o que permite a efetividade de muitos ataques XSS. Esta etapa visa viabilizar a extração das características que dificilmente seriam detectadas em seu estado ofuscado.

Etapa de Extração de Características Decodificadas: Após a decodificação da página *web* ocorre a extração das características decodificadas. Nessa etapa são extraídas as características baseadas em esquemas HTML/*JavaScript* e em padrões suspeitos, descritas na Seção 3.1. As características relativas à ofuscação de código não podem determinar, isoladamente, se o código é malicioso, visto que as codificações alternativas também são empregadas em código não malicioso. Desta forma, esta etapa extrai as demais características que reforçam o aprendizado da máquina e melhoram a acurácia no processo de classificação.

Etapa de Classificação: esta etapa tem por objetivo distinguir se uma instância *web* pertence à classe XSS ou *Non*-XSS, ou seja, se é uma página *web* infectada por vetor XSS empregando código *JavaScript* ou se é uma página benigna. Para isso, um conjunto de instâncias *web* de treino é fornecido ao método classificador para a obtenção de um modelo preditivo.

4. Experimentos e Avaliação de Resultados

Esta seção descreve os classificadores empregados nos experimentos, a base de dados utilizada e a análise dos resultados experimentais na classificação automática de XSS em páginas *web*. Para a análise do conhecimento foi utilizada a ferramenta de mineração de dados *Weka* (http://www.cs.waikato.ac.nz/ml/weka).

4.1. Métodos de Aprendizagem de Máquina

Os experimentos foram realizados com dois métodos de aprendizagem de máquina: *Naive Bayes* e *Support Vector Machines* (SVM).

Naive Bayes: é um método estatístico baseado na regra de *Bayes* [Alpaydin, 2004]. O método busca quantificar as decisões de classificação através do cálculo de

probabilidades e os custos que acompanham as decisões. Os classificadores *Bayesianos* adotam o conceito de independência condicional, assumindo que o valor de um atributo sobre uma determinada classe independe dos valores dos demais atributos. A regra de *Bayes*, basicamente seleciona a classe mais provável para um atributo "x" por meio do cálculo da probabilidade a *posteriori* de cada classe C_i, dado "x" [Alpaydin, 2004], [Witten et al., 2005]. Este método de classificação normalmente apresenta uma elevada taxa de acerto a um custo computacional reduzido. É amplamente utilizado em aplicações *web* [Chakrabarti, 2003] e em detecção de anomalias [Chandola et al., 2009].

***Support Vector Machines (SVM)*:** é um método baseado na teoria de aprendizagem estatística e otimização matemática. A técnica consiste em selecionar um hiperplano ótimo, que maximiza a margem de separação das classes, para separar os padrões de treinamento em diferentes classes. A idéia básica de SVM é mapear não linearmente os vetores do espaço de entrada em um espaço de características de maior dimensão, em que os dados podem ser separados linearmente. SVM associa uma função de núcleo (*Kernel*) que permite calcular o hiperplano sem necessariamente mapeá-lo no espaço de características e possui hiperparâmetros ajustáveis, permitindo otimizar a tarefa de classificação. SVM é um método estável e apresenta forte fundamentação matemática. Estas características garantem a SVM uma boa capacidade de generalização [Alpaydin, 2004], [Witten et al., 2005].

4.2. Base de Dados

Os algoritmos classificadores do método proposto precisam de um conjunto de dados de exemplo rotulados, que compreendem as instâncias positivas, que representam as páginas infectadas com código XSS, e as instâncias negativas, que representam as páginas benignas. Para o conjunto de instâncias negativas foram coletadas 57.207 amostras de páginas *web* selecionadas da base *Dmoz* (http://www.dmoz.org) e 158.847 amostras de páginas *web* selecionadas da base *ClueWeb09* (http://www.lemur project.org). As bases negativas foram selecionadas aleatoriamente de suas bases completas e são páginas com conteúdo na língua inglesa. Para a base positiva foram empregadas 15.366 páginas *web* infectadas com código XSS da base *XSSed*, (http://www.xssed.com) referentes a ataques ocorridos de 23 de junho de 2008 a 02 de agosto de 2011.

4.3. Medidas de Desempenho

A métrica empregada para a avaliação dos resultados foi validação cruzada [Chakrabarti, 2003], técnica que tem por objetivo a predição e a estimativa de quão correto um modelo irá ser executado na prática. Foi empregada a validação cruzada com 10 partes (*folds*) para os dois classificadores, mantendo-se a mesma proporção em todos os experimentos, a fim de permitir a proporcionalidade no treinamento e a comparação dos resultados obtidos. Essa estratégia divide o conjunto de treinamento em n subgrupos de igual tamanho, onde $n = 10$. A cada iteração, $n-1$ conjuntos são combinados para compor o conjunto de treino, enquanto que o subgrupo restante é usado como base de teste. Esse processo é repetido n vezes, sendo que cada um dos subgrupos deverá ser escolhido como base de teste uma vez. No final, a média de classificação obtida nas n bases de teste é calculada.

Como métrica para a análise de desempenho foi selecionada a “Matriz de Confusão”, que tem por propósito analisar os resultados com base nas perdas causadas, permitindo avaliar o resultado de falsos positivos e falsos negativos, conforme mostrado na Tabela 1.

As medidas empregadas foram:

- **Taxa de detecção = VP/(VP+FN)**;
- **Taxa de precisão = (VP+VN) / (VP+VN+FP+FN)**;
- **Taxa de falso alarme = FP/ (FP+VN)**.

Tabela 1. Matriz de Confusão

Classe Real	**Classificação**	
	Ataque (XSS)	**Normal (Non-XSS)**
Ataque (XSS)	**VP**	**FN**
Normal (Non-XSS)	**FP**	**VN**

Onde: VN (Verdadeiro Negativo) indica instâncias normais classificadas corretamente; FN (Falso Negativo) indica instâncias maliciosas classificadas como normais; FP (Falso Positivo) indica instâncias normais classificadas como maliciosas; e VP (Verdadeiro Positivo) indica instâncias maliciosas classificadas corretamente.

4.4. Análise de Desempenho na Classificação Automática de XSS

Os métodos de classificação foram avaliados com diferentes valores de configuração e hiperparâmetros. O método *Naïve Bayes* foi empregado com a sua configuração original. O método SVM obteve o melhor resultado com o *Kernel Polynomial* com expoente de grau 1.0 (um) e com o parâmetro de regularização “C” com valor igual a 1.0 (um). Para avaliar a eficiência do método proposto nos resultados obtidos na classificação, foram usadas todas as características selecionadas, a fim de treinar os classificadores empregados. Nesta série de experimentos foram usadas bases negativas (benignas) diferentes, porém com a mesma base positiva (maliciosa).

O experimento com a base *ClueWeb09* teve por objetivo validar a efetividade do método em uma base distinta, a partir da comparação dos resultados obtidos com o experimento empregando a base *Dmoz*, tendo em vista que não existem bases de dados públicas ou de trabalhos relacionados com foco em XSS, empregando técnicas de aprendizagem de máquina, disponíveis para pesquisas.

Os resultados apresentados na Tabela 2 demonstram que as taxas de classificação do experimento com a base *ClueWeb09* (experimento 2) apresentaram um melhor desempenho quando comparado ao experimento empregando a base *Dmoz* (experimento 1). A elevação das taxas de detecção e precisão e a diminuição das taxas de falso alarme confirmam a estabilidade do método e o potencial das características propostas na aprendizagem dos algoritmos classificadores empregados e no resultado da classificação automática de XSS em páginas *web*, considerando o treinamento com uma base distinta e com um número maior de instâncias de treino.

Tabela 2. Comparação dos resultados obtidos pelos classificadores *Naive Bayes* e SVM

Classificador	***NaiveBayes***		**SVM**	
Bases de Treinamento	***XSSed***		***XSSed***	
	Dmoz* *(experimento 1)	***ClueWeb09* *(experimento 2)***	***Dmoz* *(experimento 1)***	***ClueWeb09* *(experimento 2)***
Taxa de detecção	**95,02%**	**99,00%**	**94,07%**	**98,86%**
Taxa de precisão	**98,54%**	**99,70%**	**98,58%**	**99,89%**
Falso alarme	**0,51%**	**0,22%**	**0,20%**	**0,02%**
Total de instâncias	**72.573**	**174.213**	**72.377**	**174.213**

Apesar do melhor desempenho geral para o classificador SVM, pode ser observado que o desempenho do classificador *Naive Bayes* foi próximo ao de SVM. Isso se deve ao fato de que as características propostas são aderentes ao conceito de independência condicional, ou seja, o valor de um atributo para uma classe independe dos valores dos outros atributos, o que torna *Naive Bayes* um método de classificação ajustável ao problema pesquisado, com a vantagem de apresentar complexidade reduzida quando comparado ao SVM [Wu et al., 2007].

4.5. Comparação dos Resultados

Como forma de validar as taxas alcançadas em nossos experimentos, os resultados foram comparados com o trabalho de Likarish et al. [2009], que empregaram características baseadas em ofuscação, como por exemplo, a contabilização de caracteres *Unicode*, Hexadecimal, quantidade de caracteres presentes no *script*, entre outras relacionadas no trabalho pesquisado. Essas características foram empregadas em um novo experimento com as bases de treinamento *Dmoz* e *XSSed* e comparadas com os resultados obtidos com as características propostas neste trabalho, conforme pode ser observado na Tabela 3. As métricas adotadas pelos autores foram: PPP (*Positive Predictive Power*), equivalente a equação VP/(VP+FN), e NPP (*Negative Predicitve Power*), correspondente a equação VN/(VN+FP), que foram mantidas para fins de comparação.

Tabela 3. Comparação das taxas obtidas usando o classificador SVM

Experimentos	**[Likarish et al., 2009]** (base original)	**[Likarish et al., 2009]** (base *Dmoz* e *XSSed*)	**(XSS)** (*Dmoz* e *XSSed*)
PPP	**92,0 %**	**91,3%**	**94,0%**
NPP	**99,7%**	**99,1%**	**99,4%**

(XSS) - Valores obtidos com as características propostas neste trabalho.

Analisando a Tabela 3 é possível observar que as características propostas neste trabalho apresentaram melhores resultados na classificação de XSS, quando comparadas

com as características utilizadas por Likarish et al. [2009] em nossa base de dados. Esse ganho é atribuído a agregação das classes de características baseadas em esquemas HTML/*JavaScript* e padrões suspeitos, obtidas após o processo de decodificação. O impacto das características baseadas em ofuscação sobre o resultado geral na classificação de XSS pode ser avaliado na Tabela 4.

Tabela 4. Análise do impacto da classe de características baseada em ofuscação utilizando o classificador SVM

Bases de treinamento		***Dmoz***	***ClueWeb09***
Métrica de Desempenho		**Precisão**	**Precisão**
Características Usadas	**Baseadas em ofuscação**	**97,09%**	**97,36%**
	Baseadas em esquemas e padrões suspeitos	**98,17%**	**98,90%**
	Todas	**98,58%**	**99,89%**

Ao avaliar a Tabela 4, percebe-se que a classe baseada em ofuscação também se confirma neste trabalho como um conjunto de características de alta relevância. Entretanto, a adição das classes de características baseadas em esquemas HTML/*JavaScript* e padrões suspeitos adicionam um ganho no resultado final da classificação de XSS, confirmando o estudo de Yue & Wang [2009] sobre caracterização de práticas inseguras com o uso da linguagem *JavaScript*.

5. Conclusões

Este artigo apresentou um método de classificação automática de ataques de XSS em páginas *web*, baseado na extração e análise de características relevantes do conteúdo do documento *web* e URL, empregando uma abordagem apoiada nos classificadores *Naive Bayes* e SVM. Para tanto, foram realizados experimentos empregando as bases *Dmoz* e *ClueWeb09*, para páginas *web* não maliciosas (benignas), e *XSSed*, para páginas rotuladas como maliciosas e uma breve comparação dos resultados obtidos com as características descritas por Likarish et al. [2009]. A taxa de classificação alcançada pelo método proposto nos experimentos realizados apresentou valores estáveis e desempenho relativamente crescente de aprendizagem e classificação, o que indica que as características definidas são potencialmente representativas e os métodos de classificação aderentes ao problema estudado.

Como contribuições, este trabalho apresentou:

- A definição e análise de características relevantes para discriminar padrões de ataques de XSS, em páginas *web*, representadas por um vetor com baixa dimensionalidade do espaço de características;
- A apresentação de um método baseado na utilização de técnicas de aprendizagem de máquina para detectar ataques de XSS em páginas *web* e;
- A análise comparativa entre os métodos de classificação *Naive Bayes* e SVM, a fim de mostrar o desempenho geral na classificação de ataques de XSS.

Os resultados obtidos neste trabalho podem ser aplicados também em um *framework* para detecção de anomalias na *web* ou ainda, como solução complementar a outras técnicas atualmente usadas, como por exemplo, na aplicação de regras, extraídas do modelo preditivo gerado, no *webproxy* Noxes, proposto por Kirda et al. [2006], pois se um classificador pode detectar ataques de XSS com sucesso, ele pode então simplificar o desenvolvimento de políticas, regras, filtros ou sistemas de detecção.

Apesar do modelo preditivo gerado apresentar boa capacidade de generalização e permitir a classificação de ataques com o mesmo padrão modelado, este trabalho não é exaustivo. Os experimentos conduzidos foram limitados aos classificadores selecionados e as manipulações de seus parâmetros. Assim, existem outros classificadores com bom desempenho que podem se ajustar ao problema e obter bons resultados.

Por fim, considerando a amplitude e evolução dinâmica das técnicas empregadas nos ataques de XSS, um bom indicador para trabalhos futuros é a pesquisa por características que representem novos tipos e dimensões de ataques de XSS. A avaliação de uma etapa de análise dinâmica para detecção de XSS DOM-*Based* é uma boa oportunidade para novas pesquisas.

Referências

Alpaydin, E. (2004). "*Introduction to Machine Learning*". Cambridge, Massachusetts, London, England, The MIT Press.

CAPEC-72 (2011). "*CAPEC-72: URL Encoding*". CAPEC. Common Attack Pattern Enumeration and Classification. http://capec.mitre.org/data/definitions/72.html, Janeiro, 2011, junho 2011.

CAPEC-245 (2010). "*CAPEC-245: Cross-Site Scripting Using Doubled Characters*". http://capec.mitre.org/data/definitions/245.html, julho 2011.

Chakrabarti, S. (2003). "*Mining The Web: Discovering Knowledge from Hypertext Data*". Morgan Kaufmann Publishers.

Chandola, V., Banerjee, A. e Kumar, V. (2009). "*Anomaly Detection: Survey. A Modified Version of this Technical Report Will Appear*". In: ACM Computing Survey, ACM, September, 2009.

Choi, J., Kim, H. (2011). "*Efficient Malicious Code Detection Using N-Gram Analysis and SVM*". In: 14th International Conference on Network-Based Information Systems, Tirana, Albania, 2011.

Ernst, M. (2003). "*Static and dynamic analysis: synergy and duality*". In Proceedings of WODA'2003 (ICSE Work-shop on Dynamic Analysis), Portland, May 2003.

Gaucher, R., Grutzmacher, K., McReynolds, J., Naumann, M. e Pal, S. (2007). "*WASC ScriptMappingProject*".https://files.pbworks.com/download/6JDcQxyroz/webappsec/13247042/ScriptMapping_Release_26Nov2007.html, abril 2011.

Grossman, J., Hansen R., Petkov, D.P., Rager, A. e Fogie, S. (2007) "*Cross Site Scripting Attacks*: *XSS Exploits and Defense*". Burlington, MA, EUA, Syngress Publishing Inc.

Jim, T., Swamy, N. e Hicks, M. (2007). "*Defeating Script Injection Attacks With Browser-Enforced Embedded Policies (BEEP)*". In:16th International World Wide Web Conference, ACM, 2007.

Kirda, E., Kruegel, C., Vigna, G. e Jovanovic, N. (2006). "*Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks*". In: 21th ACM Symposium on Applied Computing, ACM, 2006.

Klein, A. (2005). "*DOM Based Cross Site Scripting or XSS of the Third Kind: A look at On Overlooked Flavor of XSS*". http://www.webappsec.org/projects/articles/071105.html, julho, 2011.

Likarish, P., Jung, E. e Jo, I. (2009). "*Obfuscated Malicious Javascript Detection using Classification Techniques*". In: 4th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 2009.

Ma, Justin., Saul, L., Savage, S. e Voelker, G. (2009). "*Identifying Suspicious URLs: An Application of Large-Scale Online Learning*". In: Proceedings of the 26th International Conference on Machine Learning, Montreal, Canadá, 2009.

Nadji, Y., Saxena, P., Song, D. (2009). "*Document Structure Integrity: A Robust Basis for Cross-Site Scripting Defense*". In: 16th Annual Network & Distributed System Security Symposium (NDSS 2009), San Diego, California, EUA.

OWASP, Foundation. (2008). "*OWASP Testing Guide*", 2008. V3.0. http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf, outubro 2010.

OWASP, Foundation. (2010). "*OWASP Top 10 – 2010. The Ten Most Critical Web Application Security Risks*", release 2010. http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf, outubro, 2010.

Saha, S. (2009). "*Consideration Points: Detecting Cross-Site Scripting*". Dep. Of Computer Science and Engineering. Hanyang University. Ansan, South Korea. (IJCSIS) International Journal of Computer Science and Information Security, Vol. 4, No. 1 & 2, 2009.

Song, Y., Keromytis, A., Stolfo, S. (2009). "Spectrogram: *A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic*". In: 16th Annual Network & Distributed System Security Symposium (NDSS 2009), San Diego, California, EUA.

Uto, N., Melo, S.P. (2009). "*Vulnerabilidades em Aplicações Web e Mecanismos de Proteção*". Minicursos SBSeg 2009. IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2009.

Wasserman, G. & Su, Z. (2008) "*Static Detection of Cross-Site Scripting Vulnerabilities*". In: 30th Internation Conference on Software Engineering, 2008.

Witten, I. e Frank, E. (2005). "*Data Mining: Practical Machine Learning Tools and Techniques*", 2ed. Elsevier.

Wu, X., Kumar, V., Quinlan, J., et al. (2007). "*Top 10 Algorithms in Data Mining*" Knowledge And Information Systems, Survey, vol 14, pp 1-37, London, Springer.

Yue, C. e Wang, H. (2009) "*Charactering Insecure JavaScript Practice on the Web*". 18th International Conference on the World Wide Web, 2005. Madri. Spain.