ETSSDetector: uma ferramenta para detecção automática de vulnerabilidades de *Cross-Site Scripting* em aplicações *web*

Thiago de Souza Rocha, Eduardo Souto, Gilbert Breves Martins

Instituto de Computação Universidade Federal do Amazonas (UFAM)
Manaus, AM – Brazil

{thiago.rocha,esouto,gilbert.martins}@icomp.ufam.edu.br

Resumo. O uso inadequado de recursos destinados a melhoria da interatividade e usabilidade das aplicações *web* tem contribuído para o surgimento de diversos ataques, entre eles, o *Cross-Site Scripting* (XSS). Este trabalho apresenta o *scanner* ETSSDetector, uma ferramenta de teste que automaticamente analisa as aplicações *web* com o objetivo de encontrar vulnerabilidades XSS. O ETSSDetector é capaz de identificar e analisar os pontos de entrada de dados da aplicação e gerar testes de injeção de código específico para cada ponto. Os resultados mostram que o preenchimento correto dos campos de entrada apenas com informações válidas garante uma melhor eficácia dos testes, aumentando a taxa de detecção de ataques XSS.

Abstract. *The inappropriate use of features intended to improve usability and interactivity of web applications resulted in the emergence of various threats, including Cross-Site Scripting (XSS) attacks. In this work, we developed ETSSDetector, a generic and modular web vulnerability scanner that automatically analyzes web applications with the aim of find XSS vulnerabilities. ETSSDetector is able to identify and analyze the data entry points of the application and generate specific code injection tests for each point. The results show that the correct filling of the input fields with only valid information ensures a better effectiveness of the tests, increasing the detection rate of XSS attacks.*

1. Introdução

As aplicações *web* têm sido utilizadas como um dos principais canais de comunicação entre os provedores de serviço e seus usuários. Portanto, garantir a segurança dessas aplicações se tornou uma tarefa prioritária e indispensável. Conforme [Shelly 2010], o número de ataques realizados nas aplicações *web* atualmente é maior que o número realizado em sistemas operacionais. O fato das aplicações estarem, normalmente, acessíveis fora do ambiente interno das empresas explica a razão das aplicações *web* serem os alvos preferenciais dos atacantes.

Um dos pontos centrais que possibilitam a existência de vulnerabilidades é a ausência ou falha na validação e interpretação dos campos de entrada das aplicações. A maior parte das linguagens de programação *web* não fornece, por padrão, uma passagem de dados segura para o cliente, conforme destacam [Wasserman e Su 2006]. A falta desse procedimento pode viabilizar uma das vulnerabilidades mais frequentes em aplicações *web*, a injeção de códigos maliciosos ou *Cross-Site Scripting* (XSS), [Grossman et al. 2007].

Para mitigar estes problemas, diferentes abordagens e técnicas têm sido propostas. Entre elas, [Martin et al. 2008] citam a análise estática, análise dinâmica, testes caixa-preta, testes caixa-branca e detecção de anomalias. Independente da abordagem utilizada, os *scanners* de vulnerabilidades *web* são as ferramentas mais empregadas por profissionais de segurança para realizar os testes de vulnerabilidade em aplicações *web*. Tal escolha é justificada por um conjunto de características que, a priori, estas ferramentas deveriam possuir. [Shelly 2010] destaca que essas ferramentas deveriam: *i)* ser fáceis de usar; *ii)* realizar os testes rapidamente; e *iii)* ser capazes de identificar uma grande variedade de vulnerabilidades.

Outra característica que facilita a aceitação dos *scanners* de vulnerabilidade pelos usuários é que a maioria deles emprega técnicas de caixa preta, ou seja, não é necessário ter acesso ao código fonte da aplicação para realizar os testes de vulnerabilidade. Contudo, apesar desta grande aceitação, pesquisas recentes mostram que muitas dessas ferramentas têm dificuldade de lidar com os diferentes níveis de complexidades empregados por diferentes ataques. [Bau et al. 2010], por exemplo, apresentam uma análise sobre um conjunto de *scanners*, concluiu que a detecção de um dos tipos de ataque XSS através dessas ferramentas é de apenas 15%. Conclusões similares sobre a efetividade dessas ferramentas também foram obtidas no trabalho desenvolvido por [Doupé et al. 2010]. [Mcallister et al. 2008] observam que a maioria dos programas falha ao testar parte das aplicações quando se envolve o preenchimento de formulários complexos. Essa falha pode fazer com que a ferramenta automatizada não consiga coletar todas as páginas da aplicação que está sendo avaliada. Tal fato influencia na cobertura dos testes na aplicação, pois as páginas que não foram coletadas não serão testadas, podendo gerar falsos negativos. Outro trabalho, realizado por [Kosuga 2011] observa que a maioria dos *scanners* aplica todos os testes possíveis em todos os campos de entrada da aplicação, gerando um esforço desnecessário.

Para lidar com estas deficiências, este trabalho apresenta o ETSSDetector, uma ferramenta desenvolvida para detectar vulnerabilidades XSS de forma automática, através da análise de informações contidas nas aplicações *web*. A ferramenta propõe a utilização de técnicas que possibilitam a qualificação e preenchimento dos campos de entrada com valores válidos permitindo a submissão correta da página. A qualificação dos campos de entrada permite a obtenção de todas as páginas da aplicação, possibilitando uma análise mais ampla e o aumento da taxa de detecção de ataques XSS.

O restante deste artigo está organizado da seguinte maneira. A seção 2 apresenta uma breve introdução sobre XSS. A seção 3 mostra uma visão geral sobre alguns trabalhos relacionados ao tema do artigo. A ferramenta proposta para atingir os objetivos do artigo é apresentada na seção 4. A seção 5 apresenta os resultados obtidos através de experimentos realizados em um conjunto de páginas *web* e a seção 6 finaliza o artigo apresentando as considerações finais e trabalhos relacionados.

2. *Cross-Site Scripting*

[Grossman et al. 2007] define *Cross-Site Scripting* (XSS) como um vetor de ataque causado por *scripts* maliciosos no lado cliente ou servidor que, devido à falta de validação adequada dos dados de entrada repassados a aplicação *web,* possibilitam o roubo de informações confidenciais, sequestro de sessões do usuário, comprometimento do navegador cliente e da integridade do sistema sob execução. Falhas XSS podem ser

classificadas em três tipos [Kirda et al. 2006]: *i)* XSS não-persistente; *ii)* XSS persistente; *iii)* XSS *Document Object Model* (DOM). As próximas seções apresentam mais informações a respeito de cada um destes tipos de ataque.

2.1 XSS Não-Persistente

Ataques XSS do tipo não persistente são os mais comuns [Saha 2009]. Nesse tipo de vulnerabilidade o código malicioso não é salvo na aplicação. O atacante analisa a aplicação procurando por campos de entrada que possam ser manipulados. Por exemplo, caixas de texto em aplicações de busca são um dos vetores mais utilizados nesse tipo de falha. Após conseguir manipular a página *web*, o atacante envia a URL maliciosa para o usuário através de e-mails, mensagens instantâneas, ou *links* embutidos em outras páginas [Kirda et al. 2006]. Figura 1 mostra um exemplo do fluxo de um ataque explora uma vulnerabilidade XSS não-persistente.

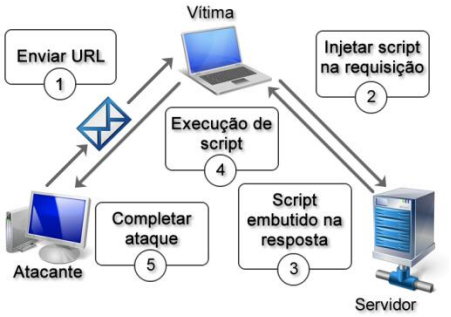

Figura 1. Exemplo de um Ataque XSS não persistente.

No passo 1, o atacante envia um e-mail para vítima contendo um *link* para uma aplicação *web* vulnerável. Quando o usuário acessa o *link* enviado por e-mail (passo 2), uma requisição HTTP é criada. Com isso o servidor *web* recebe essa requisição que gera uma resposta com o código malicioso embutido (passo 3). Em seguida (passo 4), o código é executado enviando as informações confidenciais que o atacante requisitou finalizando o ataque no passo 5.

2.2 XSS Persistente

Ataques XSS persistentes ocorrem, na maioria das vezes, em fóruns ou aplicações de *Webmail* e não precisam de páginas com código embutido para serem executados [Grossman et al. 2007]. O ataque só é completado quando o código malicioso salvo na aplicação está sendo acessado e utilizado [Pietraszek e Berghe 2005].

O atacante acessa a aplicação *web* vulnerável e salva um código malicioso no repositório da aplicação. Quando um usuário acessa a parte vulnerável da aplicação, o *script* malicioso é executado, obtendo informações confidenciais. A Figura 2 mostra um exemplo do fluxo de um ataque que explora uma vulnerabilidade XSS persistente.

O atacante corrompe a aplicação vulnerável adicionando um *script*. Assim, o código malicioso será executado por todo usuário que acessar a página que o contenha. Comentários de blogs, mensagens de fóruns e mensagens em chats são os vetores mais usados nesse tipo de ataque. Por exemplo, ao acessar um *link* disponibilizado em um fórum, o navegador do usuário irá automaticamente executar o código malicioso. Tal fato faz com que o ataque XSS persistente seja mais perigoso que os outros tipos de

ataque XSS, pois o usuário não tem como se defender de um ataque que ele não sabe que foi executado [Grossman et al. 2007].

Figura 2. Procedimento de um ataque XSS persistente.

2.3 XSS DOM

Um ataque que explora uma falha do tipo *Document Object Model* (DOM)[1] requer que a página vulnerável permita que o usuário informe os dados que serão utilizados em *scripts* que utilizam o DOM e consigam modificar algum aspecto na página de forma insegura, como *document.location, document.URL* ou *document.referrer* [Klein 2012].

```
1 <HTML>
2 <TITLE>Welcome!</TITLE>
3 Hi
4 <SCRIPT type="text/javascript">
5 var pos=document.URL.indexOf("name=")+5;
6 document.write(unescape(document.URL.substring(pos,document.URL.length)));
7 </SCRIPT>
8 <BR>
9 Welcome to our system
10 …
11 </HTML>
```

Figura 3. Exemplo de código de página vulnerável a um ataque do tipo DOM.

O objeto *document* do DOM representa a maioria das informações de uma página que é utilizada no navegador. Para facilitar o entendimento de um ataque do tipo DOM, considere o trecho de código da Figura 3. Na linha 6, o desenvolvedor utiliza as funções *document.URL* e *document.write* para permitir que a página seja modificada no navegador do cliente, sem realizar nenhuma validação, possibilitando a exploração de uma vulnerabilidade XSS do tipo DOM.

3. Trabalhos Relacionados

Esta seção mostra pesquisas relacionadas a vulnerabilidades XSS. Em geral, as soluções propostas são construídas a partir da combinação de diferentes técnicas de detecção. As soluções existentes podem ser divididas em: técnicas que modificam o lado servidor da aplicação, técnicas que modificam o lado cliente e os *scanners* de vulnerabilidades *web*. Entretanto, existem trabalhos que modificam tanto o lado cliente e o lado servidor.

[Shalini e Usha 2011] afirmam que as soluções aplicadas no lado servidor tem a vantagem de serem capazes de descobrir um número maior de vulnerabilidades e o

[1] DOM trata-se de uma plataforma que possibilita que programas e scripts acessem e modifiquem dinamicamente o conteúdo, estrutura e estilo de documentos. Maiores detalhes em http://www.w3.org/DOM/.

benefício de que, caso uma vulnerabilidade seja resolvida, sua solução seja automaticamente propagada para todos os usuários. A maioria das soluções focalizadas no lado servidor são baseadas na instalação de *proxies,* utilizados para classificar qualquer conteúdo fornecido pelo usuário, como nos trabalhos desenvolvidos por [Pietraszek et al. 2005], [Martin et al. 2008] e [Bisht et al. 2006].

[Pietraszek et al. 2005] desenvolveram uma solução que visa descobrir a causa raiz de ataques XSS através da serialização do conteúdo passado pelos usuários nos campos de entrada. Isto se torna possível através da modificação do interpretador da linguagem de programação utilizada, de forma a que este gere as informações adicionais necessárias para a realização de análises baseadas em contexto. A principal limitação ocorre pelo fato de exigir a alteração do interpretador. Na proposta original, um interpretador de PHP foi alterado para a validação da ferramenta. Entretanto, esta modificação deveria ser feita em qualquer outro interpretador utilizado no processamento de *scripts*, o que não é facilmente replicável em outras linguagens para *web* como, por exemplo, aquelas baseadas em soluções proprietárias.

[Martin et al. 2008] desenvolveram duas abordagens que podem ser utilizadas separadas ou em conjunto. A primeira apresenta um detector para encontrar vulnerabilidades XSS persistentes e a segunda outro detector para XSS não persistentes. Ambos são baseados em monitoramento de tráfego HTTP que dependem da correlação entre os parâmetros que são coletados. A maior limitação é o grande número de falsos positivos gerados, principalmente, pelo segundo detector. Além disso, a utilização dos dois detectores em conjunto pode causar problemas de desempenho e escalabilidade.

As soluções no lado cliente, em sua maioria, também atuam como um *proxy,* criando regras manuais para mitigar as tentativas de ataques e protegendo eficientemente contra o vazamento de informações do ambiente do usuário [Selvamani et al. 2010] [Kirda et al. 2006] [Ismail et al. 2004] [Athanasopoulos et al. 2010].

[Kirda et al. 2006] apresentaram um trabalho que implementa uma ferramenta chamada Noxes, um *web proxy* que atua como um *firewall* pessoal para aplicações *web* analisando as requisições HTTP. O Noxes permite que as conexões do usuário sejam realizadas com base em uma política de segurança definida por meio de regras automáticas ou manuais. O usuário pode criar regras para requisições de páginas *web* e inseri-las de forma interativa, cada vez que um pedido de conexão é realizado. Conforme os próprios autores, um dos grandes problemas dessa abordagem corresponde à geração de um grande número de alertas, exigindo a intervenção do usuário.

Pode-se observar que todos os trabalhos citados até o momento modificam algum aspecto no lado servidor ou no lado cliente da aplicação para combater as falhas XSS. Entretanto, existem ferramentas que não exigem que qualquer modificação seja introduzida na aplicação. Estas ferramentas são os *scanners* de vulnerabilidades *web*.

Atualmente existem muitos *scanners* disponíveis, alguns comercias como [Appscan 2012], [Acunetix 2011], [N-Stalker 2012], [WebCruiser 2012] e outros de código aberto como os desenvolvidos por [Kals et al. 2006], [PowerFuzzer 2012], [WebSecurify 2012] e [Jia 2006].

[Kals et al. 2006], apresentam o SecuBat, uma ferramenta utilizada para descobrir vulnerabilidades XSS. A arquitetura do SecuBat é dividida em três camadas. A primeira camada é responsável pela configuração da varredura de vulnerabilidades.

Nela o usuário fornece as informações necessárias para o SecuBat iniciar a sua execução, como o endereço da aplicação que será testada. A segunda camada consiste de módulos para realizar a extração de informações da aplicação, realizar os testes e configurar os *plug-ins* de ataques. Por fim, a última camada é responsável por armazenar todas as informações que são necessárias durante a execução do SecuBat. O SecuBat foi projetado para focar apenas na descoberta de ataques XSS não persistentes.

O estudo bibliográfico realizado neste trabalho demonstra que, apesar de existirem muitas ferramentas propostas para mitigar o problema de XSS, muito ainda deve ser feito para que este problema possa ser tratado de forma mais efetiva.

4. ETSSDetector

Ao se conduzir testes de vulnerabilidades XSS em aplicações *web*, os seguintes problemas devem ser levados em consideração:

1. Como identificar todos os pontos de vulnerabilidade?
2. Como acessar os formulários da página e preenche-los?
3. Como coletar todas as páginas da aplicação?
4. Quais ataques utilizar em cada ponto de vulnerabilidade?
5. Como identificar se o ataque foi executado corretamente?

Para lidar com estas questões, este trabalho propõe uma ferramenta de teste de vulnerabilidades em páginas *web*, denominada, ETSSDetector. A ferramenta propõe a utilização de técnicas (descritas nas seções seguintes) que possibilitam a qualificação e preenchimento dos campos de entrada com valores válidos permitindo a submissão correta da página. A ferramenta ETSSDetector é organizada em um conjunto de módulos especializados como mostra a Figura 4. Cada módulo é listado a seguir:

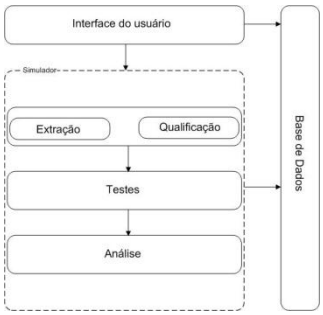

Figura 4. Arquitetura do *scanner* ETSSDetector.

- Interface do usuário: utilizada para que o usuário preencha as informações necessárias para o inicio da execução da ferramenta.
- Simulador: responsável por reproduzir a utilização de um navegador.
- Módulo de extração: responsável por coletar as páginas da aplicação.
- Módulo de qualificação: utilizado para fazer a análise DOM das páginas *web* e minimizar a submissão de formulários complexos.
- Módulo de testes: responsável por simular os ataques gerados pelo componente de geração de código.
- Módulo de análise: usado para determinar se os ataques foram executados com sucesso.
- Base de dados: módulo responsável por armazenar as informações obtidas sobre as páginas da aplicação testada.

4.1. Simulador

Para detectar as vulnerabilidades XSS persistentes e não persistentes, o ETSSDetector deve ser capaz de simular a utilização da aplicação *web* que estará sendo testada e realizar as mesmas ações de um usuário.

Para emular esse comportamento um simulador de navegador pode ser utilizado, no caso do ETSSDetector foi utilizado o HtmlUnit [Gargoyle 2012]. Através desse simulador é possível modelar todo o documento HTML e interagir com as páginas, preenchendo formulários e acessando *links* de forma a simular o comportamento de um navegador real.

4.2. Módulo de Extração

O módulo de extração é responsável pela identificação, coleta e análise das informações necessárias à avaliação da aplicação *web*. O processo de extração coleta informações das páginas que serão testadas, tais como seus *links* e parâmetros de seus formulários, bem como identifica quais métodos o formulário irá executar.

O processo começa com uma página inicial para ser visitada. O extrator então faz uma requisição a um servidor HTTP, baixa o documento e dele extrai os *links*, conteúdo e outras informações. Os endereços dos *links* extraídos são guardados para que se possa acessá-los posteriormente.

Como a maioria dos coletores de página existentes, o processo prevê dois tipos de requisições HTTP, via os métodos POST e GET. Caso a requisição inicial seja realizada via o método GET, os parâmetros do endereço da página são extraídos e inseridos na base de dados. Caso a requisição seja do tipo POST, os parâmetros dos formulários são recuperados e também armazenados na base de dados. Durante o processo de extração de informações, novas páginas da aplicação *web* avaliada poderão ser encontradas e deverão passar pelo mesmo procedimento.

4.3. Módulo de Qualificação

O módulo de qualificação de entradas é responsável por fazer uma análise das páginas no navegador do cliente, identificando cada ponto de vulnerabilidade e seus possíveis valores para qualificar o seu preenchimento durante o processo de teste.

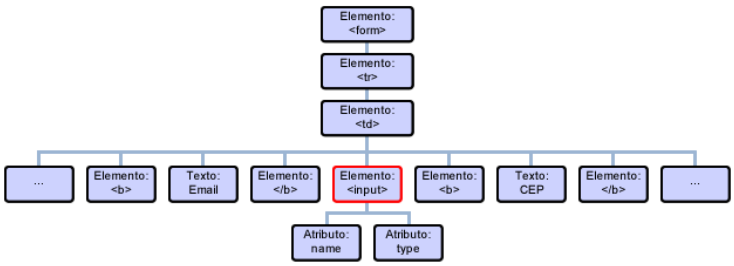

Figura 5. Análise do elemento *input*.

O objetivo desse processo é identificar quais entradas são mais apropriadas no preenchimento de um formulário. Por exemplo, o processo de qualificação do elemento *input*, mostrada na árvore DOM da Figura 5 se inicia com a recuperação do valor do campo localizado a esquerda. Neste caso, o valor do campo de texto "email". Após a recuperação do valor do campo é checado se existe algum qualificador correspondente a este valor na base de dados.

Caso exista, o valor do qualificador é recuperado e atribuído ao valor do elemento que esta sendo qualificado. Caso contrário, a análise continua verificando os valores dos atributos do elemento seguinte. Nesse caso, os valores dos atributos *name* e *type* do elemento *input*. Caso não exista um qualificador correspondente aos valores desses atributos, a análise continua verificando os campos a direita do elemento *input*. Por fim, caso não exista um qualificador correspondente a nenhum dos valores testados o campo recebe como entrada um valor padrão.

No caso da Figura 5, o texto *email* seria utilizado para análise e o qualificador de e-mail correspondente seria encontrado, fazendo com que o elemento *input* recebesse o valor do qualificador, por exemplo, "teste@gmail.com".

Após qualificar todos os campos do formulário, é realizada uma submissão do mesmo, para identificar se a página resposta foi submetida ao extrator de informações. Caso isso não tenha ocorrido, a página é marcada para passar pelo mesmo processo.

Esse módulo apresenta um diferencial com relação às outras ferramentas analisadas, pois a qualificação visa preencher cada campo com apenas informações válidas para garantir a submissão de formulários complexos, mitigando o problema encontrado por [Mcallister et al. 2008].

4.4. Módulo de Testes

O módulo de testes é responsável pela injeção e execução dos códigos de ataques XSS que serão inseridos em cada ponto de vulnerabilidade da página *web* avaliada. Este módulo é composto pelos componentes de injeção e geração de código.

O processo de injeção de código gerado é utilizado para ajudar na análise do comportamento da página *web* na presença de ataques XSS. Nesta etapa, o código gerado é inserido na página *web* e consequentemente executado. Os pontos de vulnerabilidades onde serão inseridos os códigos em cada página foram identificados no módulo de extração e armazenados na base de dados.

Diferentemente da maioria das ferramentas de teste de vulnerabilidade em aplicações *web*, o ETSSDetector propõe o uso de um processo de seleção ataques. Os testes são feitos de acordo com o tipo de ponto de vulnerabilidade que será testado. Por exemplo, caso o ponto de vulnerabilidade encontre-se no título da página, apenas ataques referentes ao título serão utilizados. O objetivo dessa abordagem é evitar que um grande número de ataques inapropriados e desnecessários seja executado durante o processo de avaliação, problema identificado por [Kosuga 2011].

Da mesma forma que o processo de extração, o processo de injeção considera os diferentes métodos de requisição HTTP (métodos POST e GET). Os parâmetros do endereço da página (método GET) ou os campos de formulários (método POST) são recuperados da base de dados. O próximo passo é injetar um código malicioso nos pontos de vulnerabilidade recuperados da página *web* avaliada.

Antes de inserir o código propriamente dito, um teste de injeção simples (por exemplo, um *alert*) é adicionado para cada ponto de entrada para determinar a posição onde o código irá aparecer na página de resposta. Esta informação possibilitará ao gerador de código criar os ataques mais apropriados e ao componente de detecção saber a possível posição de um código malicioso na página. Tal abordagem, conhecida como "*dummy injection code*", é empregada na ferramenta WebXSSDetector [Jia 2006].

A etapa de geração automática de código malicioso é responsável pela criação dos códigos que serão inseridos nos pontos de vulnerabilidade. Como os ataques XSS possuem uma enorme variedade de formas de serem executados, um gerador de código é necessário para simular o maior número de variações de ataques possíveis.

O processo de geração de código é simples e funciona da seguinte maneira: primeiro, é obtida uma lista inicial de ataques XSS conhecidos. Essa lista pode ser recuperada a partir de repositórios mantidos por grupos de segurança como [OWASP 2011]. Em seguida, variações dos códigos maliciosos iniciais são geradas de acordo com as informações de posicionamento coletadas durante o processo de teste de injeção.

O gerador de código malicioso identifica cada ataque gerado através de um contador (ID Ataque). Esse identificador é utilizado para auxiliar o módulo de análise na procura dos ataques XSS que foram bem sucedidos.

As variações de ataques são geradas pelo fato dos atacantes empregarem diferentes formas de codificações para burlar os filtros que são desenvolvidos pelos programadores. Na tentativa de simular o comportamento real de um atacante, o gerador de código gera diferentes variações de um mesmo ataque.

4.5. Módulo de Análise

O módulo de Análise é responsável por realizar a detecção dos ataques XSS através da análise dos resultados dos módulos anteriores e posteriormente gerar relatórios para que os dados possam ser consultados.

O diferencial deste módulo encontra-se no fato do ETSSDetector realizar a detecção de ataques em toda a aplicação. A maioria das ferramentas citadas na seção 3 verifica apenas a resposta direta gerada pela requisição. Com essa abordagem o ETSSDetector visa melhorar o nível de detecção de ataques XSS persistentes, problema encontrado por [Bau et al. 2010].

O processo se inicia através da obtenção da lista de testes gerada na fase anterior. De posse dessa lista ocorre uma iteração onde os valores dos testes serão inseridos em cada ponto de vulnerabilidade.

O próximo passo é executar o teste para verificar o seu resultado. Caso o resultado seja positivo ocorre uma atualização na base de dados e o processo continua no próximo ponto de vulnerabilidade, caso contrário, o processo continua até que a lista de testes fique vazia.

4.6. Base de Dados

Todas as informações coletadas durante as etapas de processamento de uma página *web* são armazenadas em uma base de dados. O uso de um repositório persistente possibilita a ferramenta acesso aos recursos em análise (*links*, formulários, parâmetros, e outros) em qualquer instante de tempo. Além de salvar as informações a utilização de uma base de dados possui outros benefícios tais como:

- Os dados podem ser facilmente recuperados através de consultas SQL.
- Depois da condução de um teste, os ataques podem ser recuperados da base de dados a qualquer momento e reconstruídos. Com isso os profissionais de segurança podem entender o ataque e tentar resolve-lo.
- Relatórios podem ser gerados com facilidade através da obtenção dos dados.

5. Avaliação e Resultados

Nessa seção são apresentados os resultados obtidos através da avaliação de desempenho do ETSSDetector. Para validar e comparar o protótipo desenvolvido, foram realizados vários testes sobre cenários experimentais e cenários reais. Por limitação de espaço serão apresentados apenas 03 cenários de testes.

Os resultados da avaliação do ETSSDetector são comparados com as ferramentas [Acunetix 2011], [PowerFuzzer 2012], [WebSecurify 2012], [NStalker 2012], [Jia 2006] e [WebCruiser 2012]. Os critérios para escolha foram a disponibilidade de acesso à ferramenta de testes de vulnerabilidade (*scanners*), a similaridade de objetivos e a percepção de utilização da ferramenta no mercado. No conjunto das ferramentas o Acunetix é considerado um dos melhores *scanners* de vulnerabilidades do mercado conforme os estudos citados anteriormente neste trabalho. Entretanto a versão gratuita do Acunetix, que esta disponível para testes, possui limitações que só permitiram a utilização da ferramenta no primeiro cenário de testes.

As métricas utilizadas para avaliar as ferramentas nos experimentos foram: *i)* quantidade de ataques XSS detectados; *ii)* quantidade de testes gerados na aplicação; e *iii)* tempo de execução das ferramentas. Tais métricas foram escolhidas por serem utilizadas em pesquisas anteriores e por verificarem a resolução dos problemas citados na introdução deste trabalho.

A coleta de informações para análise ocorreu através da análise dos comportamentos das ferramentas, dos relatórios gerados e, quando possível, dos *logs* obtidos no servidor onde se encontra a aplicação que esta sendo testada.

5.1. Primeiro Cenário

No primeiro cenário de teste foi utilizada uma aplicação, disponível em *testphp.vulnweb.com,* desenvolvida pelos proprietários da ferramenta Acunetix. A Tabela 1 mostra os resultados da quantidade de ataques encontrados neste teste.

O primeiro ponto analisado neste cenário de teste é que a maioria dos *scanners*, entre os quais o ETSSDetector, tiveram o seu tempo de execução entre 2 e 5 minutos, com exceção da ferramenta PowerFuzzer que terminou a sua execução em 14 minutos e 51 segundos e o WebSecurify que foi executado em apenas 37 segundos.

Tabela 1. Resultados do primeiro cenário de teste.

Ferramenta	*Ataques XSS encontrados*	*Tempo*
Acunetix	17 ataques	2 minutos e 2 segundos
PowerFuzzer	12 ataques	14 minutos e 51 segundos
N-Stalker	9 ataques	5 minutos
WebSecurify	4 ataques	37 segundos
WebXSSDetector	15 ataques	4 minutos e 9 segundos
WebCruiser	8 ataques	2 minutos e 30 segundos
ETSSDetector	17 ataques	2 minutos e 37 segundos

Com relação aos ataques de XSS, o Acunetix e o ETSSDetector detectaram o maior número de vulnerabilidades, 17 ataques cada, sendo 15 deles os mesmos. O Acunetix encontrou dois ataques XSS que foram executados através de *cookies,* funcionalidade que o ETSSDetector ainda não possui. Contudo, o ETSSDetector

encontrou dois outros ataques não detectados pelo Acunetix. As únicas ferramentas capazes de encontrar esses dois ataques foram o ETSSDetector e o WebXSSDetector.

Esses ataques ocorrem na página da aplicação que serve para o usuário criar contas http://testphp.vulnweb.com/signup.php, em um dos ataques possíveis nessa página o atacante pode utilizar os campos "*password*" e "*retype password*" para inserir *scripts* maliciosos. O *script* inserido deve ser o mesmo nos dois campos, como ambos são vulneráveis a ferramenta identificou como pontos de vulnerabilidade. Para provar que esses ataques não são falsos positivos os mesmos foram reproduzidos manualmente.

5.2. Segundo Cenário

No segundo cenário de teste foi desenvolvida uma página com vulnerabilidades para testar o processo de qualificação e a geração de ataques. Essa página possui quatro campos como mostra a Figura 6.

Entre esses campos dois possuem vulnerabilidades (nome e sobrenome) e dois precisam de validação para o formulário ser submetido (Email e CPF). Propositalmente o campo nome é exibido no título da página de resposta e o campo sobrenome é exibido como atributo de uma *tag* HTML.

Esse procedimento foi feito no intuito de testar se o ETSSDetector gera menos ataques (menos requisições) para cada ponto de vulnerabilidade através da aplicação da técnica de testes de injeção. A Tabela 2 exibe os resultados do segundo cenário.

Através da Tabela 2 é possível verificar que entre os *scanners* que detectaram os ataques, o WebCruiser gerou mais de quatro vezes a quantidade de requisições geradas pelo ETSSDetector e o WebSecurify gerou mais que o dobro. Com isso, pode-se afirmar que o ETSSDetector gerou menos ataques em cada ponto de vulnerabilidade.

Figura 6. Campos da aplicação para teste do processo de qualificação.

Tabela 2. Resultados do segundo cenário de teste.

Ferramenta	*Nº. de requisições*	*Nº. de ataques detectados*
PowerFuzzer	64 requisições	Nenhum
WebSecurify	37 requisições	Todos
WebXSSDetector	Erro em tempo de execução	Nenhum
WebCruiser	60 requisições	Todos
ETSSDetector	15 requisições	Todos

O PowerFuzzer foi a ferramenta que efetuou o maior número de requisições, porém passou valores inválidos em todas. Analisando a Figura 7, é possível verificar que o PowerFuzzer passa como parâmetro para o campo de *email* o endereço do Google e passa como parâmetro para o campo CPF o texto "*on*" fazendo com que o formulário não fosse submetido.

Figura 7: Exemplo de ataque do PowerFuzzer.

5.3. Terceiro Cenário

No terceiro cenário foram conduzidos testes em uma aplicação que efetua reserva de salas, disponibilizada em http://www.webscantest.com/crosstraining /reservation.php. Essa página possui duas vulnerabilidades XSS, ambas não persistentes. Porém, uma delas se comporta como se fosse persistente, pois não é imediatamente refletida ao usuário, a Tabela 3 mostra os resultados.

Tabela 3. Resultados do terceiro cenário de teste.

Ferramenta	*Nº. de ataques detectados*
PowerFuzzer	Nenhum
WebSecurify	Nenhum
WebXSSDetector	Nenhum
WebCruiser	Um
ETSSDetector	Duas

Através da Tabela 3 é possível verificar que apenas o ETSSDetector foi capaz de detectar as duas vulnerabilidades. O WebCruiser detectou apenas uma das vulnerabilidades e as demais ferramentas não encontraram nenhuma. No intuito de verificar se o ataque que apenas o ETSSDetector encontrou tratava-se de um falso positivo o mesmo foi reproduzido manualmente.

Os testes que são realizados pelo ETSSDetector não são baseados apenas na resposta direta gerada pela requisição. Uma análise é realizada nas outras páginas da aplicação. Por esse motivo foi possível detectar os dois ataques que a aplicação de reserva de salas possui.

6. Conclusões

Este artigo apresentou uma ferramenta, denominada ETSSDetector, para detecção automática de vulnerabilidades de *Cross-Site Scripting* em páginas *web* através da análise das informações contidas nas aplicações.

Através da análise dos resultados obtidos foi possível verificar que a aplicação da técnica de testes de injeção reduziu a quantidade de testes gerados nos pontos de vulnerabilidades e a utilização de qualificadores nos pontos de vulnerabilidade proporcionou o aumento da garantia de submissão de formulários complexos.

A verificação dos testes em todas as páginas da aplicação, em conjunto com as técnicas de testes de injeção e qualificação fez com que a taxa de detecção dos ataques XSS fosse mais eficiente, conforme demonstrado nas comparações realizadas com outras ferramentas de testes. Assim, o *scanner* ETSSDetector apresenta-se como uma alternativa para auxiliar os profissionais de segurança da informação na luta contra as vulnerabilidades que estão presentes em sistemas *web*.

Como trabalho futuro pretende-se alterar o módulo de testes para possibilitar que ataques sejam adicionados pelos usuários. Outra característica importante seria a avaliação do ETSSDetector para adicionar mecanismos para detectar falhas XSS do tipo DOM. Por fim, ampliar a utilização da ferramenta proposta adicionando mecanismos para a detecção de outras vulnerabilidades em aplicações *web* como SQL *Injection*.

7. Agradecimentos

Este trabalho foi parcialmente desenvolvido com o apoio da Fundação de Amparo à Pesquisa do Estado do Amazonas (FAPEAM).

Referências

Shelly, D. (2010) "*Using a Web Server Test Bed to Analyze the Limitations of Web"* http://scholar.lib.vt.edu/theses/available/etd-08102010-184408/unrestricted/Shelly_DA_T_2010.pdf, Julho.

Su, Z., e Wassermann, G. (2006) "The Essence of Command Injection Attacks in Web Applications", *33rd ACM Symposium on Principles of Programming Languages*, páginas 1-6.

Grossman, J., Hansen, R., Petkov, P., Rager, A. e Fogie, S. (2007) "Cross site scripting attacks:XSS Exploits and defense.", Syngress,Elsevier, páginas 67-179.

Martin, J., Bjorn, E. e Joachim, P. (2008) "XSSDS: Server-side Detection of Cross-Site Scripting Attacks," *IEEE Computer Security Applications Conference,* páginas 1-10.

Bau, J., Bursztein, E., Gupta, D. e Mitchell, J. (2010) "State of the Art: Automated Black-Box Web Application," *IEEE Symposium on Security and Privacy Vulnerability Testing,* páginas 2-5.

Doupe, A., Cova, M. e Vigna, G. (2010) "Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners," *Seventh Conference on Detection of Intrusions and Malware and Vulnerability Assessment,* páginas 1-10.

McAllister, S., Kirda, E. e Kruegel, C. (2008) "Leveraging User Interactions for In-Depth Testing of Web Applications," *ACM international symposium on Recent Advances in Intrusion Detection,* páginas 1-2.

Kosuga, Y. (2011) "*A Study on Dynamic Detection of Web Application Vulnerabilities"* http://iroha.scitech.lib.keio.ac.jp:8080/sigma/bitstream/handle/10721/43/document.pdf?sequence=1, Janeiro.

Saha, S. (2009) "Consideration Points: Detecting Cross-Site Scripting," *International Journal of Computer Science and Information Security,* páginas 1-8.

Kirda, E., Kruegel, C., Vigna, G. e Jovanovic, N. (2006) "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks," *ACM symposium on Applied computing,* páginas 1-8.

Pietraszek, T. e Berghe, C. V. (2005) "Defending against Injection Attacks through Context-Sensitive String Evaluation," *Proceedings of Recent Advances in Intrusion Detection,* páginas 1-15.

Klein, A. (2012) "DOM Based Cross Site Scripting or XSS of the Third Kind", http://www.webappsec.org/projects/articles/071105.shtml, Outubro.

Shalini, S. e Usha, S. (2011) "Prevention Of Cross-Site Scripting Attacks (XSS) On Web Applications In The Client Side," *IJCSI International Journal of Computer Science Issues,* páginas 1-5.

Bisht, P. e Venkatakrishnan, V.N. (2006) "XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks," *ACM Symposium on Applied Computing,* páginas 1-10.

Selvamani, K., Duraisamy, A. e Kannan, A. (2010) "Protection of Web Applications from Cross-Site Scripting Attacks in Browser Side," *(IJCSIS) International Journal of Computer Science and Information Security,* páginas 1-8.

Ismail, O., Eto, M., Kadobayashi, Y. e Yamaguchi, S. (2004) "A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability," *International Conference on Advanced Information Networking and Applications (AINA04),* páginas 1-7.

Athanasopoulos, E., Pappas, V., Krithinakis, A., Ligouras, S. e Markatos, E. P. (2010) "xJS: Practical XSS Prevention for Web Application Development," *In: Proceedings of the 2010 USENIX Conference on Web Application Development,* páginas 1-8.

AppScan. (2012) "Download IBM AppScan," http://www.ibm.com/developerworks/downloads/r/appscan/, Junho.

Acunetix. (2011) "Website Security with Acunetix Web Vulnerability Scanner," http://www.acunetix.com/, Agosto.

N-Stalker. (2012) "N-Stalker The Web Security Specialists," http://www.nstalker.com/, Abril.

WebCruiser. (2012) "Download WebCruiser," http://sec4app.com/download.htm, Fevereiro.

Kals, S., Kirda, E., Kruegel, C. e Jovanovic, N. (2006) "SecuBat: A Web Vulnerability Scanner," *International World Wide Web Conference (WWW2006),* páginas 1-10.

PowerFuzzer. (2012) "PowerFuzzer - a fuzzer that introduces powerfull and easy web fuzzing," http://www.powerfuzzer.com/, Julho.

WebSecurify. (2012) "WebSecurify Online Web Application Security Scanner and Web Security Testing Tool," http://www.websecurify.com/, Julho.

Jia, X. (2006) "*Design, Implementation and Evaluation of an Automated Testing Tool for Cross-Site Scripting Vulnerabilities,"* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.1460&rep=rep1&type=pdf, Março.

Gargoyle. (2012) "HtmlUnit - Welcome to HtmlUnit," Gargoyle, http://htmlunit.sourceforge.net/, Janeiro.

OWASP. (2011) "XSS Filter Evasion Cheat Sheet," https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet, Outubro.

