FACULDADE DE PINDAMONHANGABA

Gabriela Silva Camargo
Verônica P. P. de Toledo

ANÁLISE DAS TÉCNICAS DE CROSS-SITE SCRIPTING:
Compreensão, simulação e prevenção

PINDAMONHANGABA - SP
2017

FACULDADE DE PINDAMONHANGABA

Gabriela Silva Camargo
Verônica P. P. de Toledo

ANÁLISE DAS TÉCNICAS DE CROSS-SITE SCRIPTING:
Compreensão, simulação e prevenção

Trabalho apresentado como parte dos requisitos para obtenção do diploma de graduação pelo curso de Sistemas de Informação da Fundação Universitária Vida Cristã – FUNVIC
Orientador: Prof. Luís Felipe Féres Santos

PINDAMONHANGABA - SP
2017

Camargo, Gabriela Silva; Toledo, Verônica Paloma Priscilla
Analise das técnicas de Cross-site Scripting: compreensão, simulação e prevenção / Gabriela Silva Camargo; Verônica Paloma Priscilla Toledo / Pindamonhangaba – SP : Fundação Universitária Vida Cristã, 2017.
34f.
Artigo (Bacharelado em Sistemas de Informação) FUNVIC-SP.
Orientador: Prof. Luis Felipe Féres Santos.

1 *Cross-site Scripting*. 2 Prevenção. 3 Mitigação. 4 Top 10 OWASP. 5 Segurança Web.
I Analise das técnicas de Cross-site Scripting: compreensão, simulação e prevenção. II Gabriela Silva Camargo; Verônica Paloma Priscilla Toledo.

GABRIELA SILVA CAMARGO
VERÔNICA P. P. DE TOLEDO

ANÁLISE DAS TÉCNICAS DE CROSS-SITE SCRIPTING: Compreensão, simulação e prevenção

Artigo apresentado como parte dos requisitos para obtenção do diploma de graduação pelo curso de Sistemas de Informação da Fundação Universitária Vida Cristã – FUNVIC.

Data:____/____/_____
Resultado:_____

BANCA EXAMINADORA

Prof._____ FUNVIC - Fundação Universitária Vida Cristã
Assinatura _____

Prof._____ FUNVIC - Fundação Universitária Vida Cristã
Assinatura _____

Prof._____ ……
Assinatura _____

PINDAMONHANGABA - SP
2017

Dedico este trabalho primeiramente a Deus por ser essencial em minha vida e me ajudar em todos os momentos, e aos meus pais que não mediram esforços para que eu chegasse até esta etapa da minha vida.

A Deus, o que seria de mim sem a fé que eu tenho nele. À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, Lucia, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, Alcizo, sua presença significou segurança e certeza de que não estou sozinha nessa caminhada. Leandro, obrigada pelo carinho, a paciência e por sua capacidade de me trazer paz na correria de cada semestre.

Agradecimentos

À Fundação Universitária Vida Cristã pela concessão da bolsa de estudo que nos permitiu que atingíssemos nossa graduação com louvor.
Ao Prof. Luís Felipe Féres Santos, pelo empenho dedicado, pela paciência na orientação e incentivo que tornaram possível a conclusão deste trabalho.
À Prof. Luciane Garcia, pela paciência e dedicação na revisão e correção desse trabalho.

Este trabalho foi redigido na forma de artigo, sob as normas da Revista Eletrônica de Ciências Humanas da FUNVIC, as quais estão em anexo.

ANÁLISE DAS TÉCNICAS DE CROSS-SITE SCRIPTING: Compreensão, simulação e prevenção

CROSS-SITE SCRIPTING TECHNIQUES ANALYSIS: Understanding, simulation and prevention

Gabriela Silva Camargo [1] - FUNVIC - Faculdade de Pindamonhangaba
Verônica Paloma Priscilla de Toledo [1] - FUNVIC - Faculdade de Pindamonhangaba
Luis Felipe Féres Santos[2] - FUNVIC - Faculdade de Pindamonhangaba

Resumo. As aplicações *Web* estão cada vez mais interativas e atrativas aos usuários. No entanto, as novas tecnologias que proporcionam estas funcionalidades dinâmicas geram também novas brechas as quais viabilizam novos ataques como o *Cross-site Scripting* (XSS). Apresentado na lista das dez vulnerabilidades mais exploradas, o XSS trata-se da injeção de *scripts* maliciosos em campos de entrada de dados não validados corretamente em aplicações *web*. Felizmente, a *Open Web Application Security Project (*OWASP*)*, uma organização de segurança de aplicações, fornece informações, regras e scanners para a mitigação desta vulnerabilidade. Este estudo apresenta meios de prevenção aos ataques do tipo XSS, segundo as normas da OWASP e diretrizes de outros autores. A fim de garantir a eficácia dos meios de prevenção apresentados, realizou-se um experimento de avaliação onde as técnicas propostas foram implementadas em uma página *web* e escaneada pela ferramenta *Zed Attack Proxy Project* (ZAP) onde pode-se observar a eficácia dos métodos adotados na prevenção das vulnerabilidades. Com isso pretende-se propor métodos a serem adotados por desenvolvedores e empresas, dos quais toma-se como iniciativa o incentivo às boas práticas da segurança da informação.

Palavras chave: *Cross-site Scripting.* Prevenção. Mitigação. Top 10 OWASP. Segurança *web*.

Abstract. *As Web applications are increasingly interactive and appealing to user. However, new technologies that provide these dynamic features also create new gaps that enable new attacks such as Cross-site Scripting (XSS). Presented in the list of the ten most exploited vulnerabilities, XSS is about injecting malicious scripts into data entry fields that are not validated correctly in web applications. Fortunately, an Open Web Application Security Project (OWASP), an application security organization, information, rules, and scanners for mitigation of this vulnerability. This study presents a means to prevent XSS attacks according to OWASP norms and guidelines of other authors. In order to ensure the effectiveness of the presented means of prevention, an evaluation experiment was carried out where the proposed techniques were implemented in a Web page and scanned by the tool Zed Attack Proxy Project (ZAP) where it can be observed the effectiveness of the methods adopted the prevention of vulnerability. With this, it is proposed to propose methods, to be adopted by developers and companies, of which, as an initiative, it encourages good practices of information security.*

Keywords: Cross-site Scripting. Prevention. Mitigation. Top 10 OWASP. Web security

[1] Alunos do curso de graduação Sistemas de Informação, ministrado pela Fundação Universitária Vida Cristã – FUNVIC.
[2] Luis Felipe Féres Santos – Professor Orientador.

1 INTRODUÇÃO

Devido ao avanço da tecnologia, a *internet* está cada vez mais presente na vida das pessoas. A estrutura dinâmica de aplicações *web* proporciona a inserção de novas funcionalidades e criação de serviços interativos como, por exemplo, *sites* para realização de transações bancárias, conferência em vídeo, compra, venda e pesquisas. Com tantas utilidades, a quantidade de informações que circulam por este meio é imensa, logo a informação tem se tornado cada vez mais importante, e é considerada por muitos como um dos patrimônios mais valiosos do século atual (BEAL, 2004).

Juntamente com o aumento no volume de informações na *web* a quantidade de pessoas interessadas em obtê-las de maneira ilegal cresce absurdamente. Em um estudo realizado no ano de 2016 o número médio de incidentes de segurança reportados aumentou 274% em comparação ao ano anterior (PWC, 2016). Neste mesmo ano o Brasil ficou em 9° lugar no ranking dos países que mais receberam ataques no mundo e é o país com o maior número de ataques na América Latina (G1, 2016; ESTADÃO, 2017 apud. KASPERSKY LAB, 2016).

O desenvolvimento de aplicações, assim como outras ocupações humanas, está suscetível a erros que ocasionam em falhas, as quais tornam uma aplicação vulnerável deixando brechas por onde as informações podem ser comprometidas. A *Open Web Application Security Project* (OWASP), uma entidade de reconhecimento internacional sem fins lucrativos, que contribui para a melhoria da segurança de *softwares* e aplicativos, apresenta de tempos em tempos uma lista com as dez vulnerabilidades mais críticas encontradas em aplicações *web*. O relatório mais recente foi emitido em 2017 e apresenta o *Cross-Site Scripting* (XSS) que aparece consecutivamente no topo das listas disponibilizadas desde 2004 (OWASP, 2017a, 2017b)

Conforme a OWASP (2016c) apresenta o dano e prejuízo que um ataque XSS pode causar tanto ao usuário quanto ao provedor de serviço é enorme, e infelizmente apesar de haver relatórios que apresentam existência deste risco, o número de vítimas ainda é grande, o que ressalta a necessidade de maior dedicação e conhecimento sobre o assunto tanto por parte dos desenvolvedores quanto pelos usuários, visando melhor controle, mitigação e prevenção do ataque.

1.1 Revisão e Fundamentação Teórica

Para melhor entendimento de como ocorre o ataque XSS e as formas de se prevenir, primeiramente é necessário conhecer o ambiente em que atua, ou seja, a estrutura *web*.

1.1.1 ARQUITETURA WEB

Nunan (2012 apud. W3C, 2010) define a *web* como uma arquitetura fundamentada no conceito cliente-servidor, composta por *softwares*, convenções e protocolos, objetivando disponibilizar serviços, informações e documentos multimídia.

Nunan (2012 apud. W3C, 2010) ainda menciona três tecnologias principais para estrutura de aplicações *web*, são elas:

- ***Hypertext Markup Language* (HTML):** linguagem de marcação que informa ao navegador como exibir o documento *web* para o usuário;
- ***Cascading Style Sheets* (CSS):** tecnologia utilizada para definição da aparência e estilo dos dados exibidos no documento *web*;
- ***Scripts*:** linguagem empregada para realizar ações no navegador do usuário proporcionando um conteúdo dinâmico às páginas *web*. O interpretador no navegador do usuário é responsável por interpretar e executar o código.

As tecnologias apresentadas unidas ao *Document Object Model* (DOM) - Uma interface neutra em linguagem e plataforma que permite aos programas e *scripts* acesso dinâmico ao conteúdo, estrutura e estilo dos documentos (W3C, 2009) - compõem o conceito de HTML dinâmico (*Dynamic* HTML – DHTML), o qual permite que uma página *web* seja modificada dinamicamente no próprio navegador (NUNAN, 2012 apud. W3C, 2010).

Neste ambiente ao utilizar a *web* o usuário está conectado diretamente com o servidor onde a aplicação se encontra, de forma autônoma, sem a necessidade de que outras pessoas o operem, porém, não é possível garantir que essas informações não estão sendo interrompidas e interceptadas por terceiros por meio de vulnerabilidades (DANTAS, 2011).

1.1.2 CROSS-SITE SCRIPTING

Observando os relatórios de vulnerabilidades fornecidos pela OWASP (2017b) o *Cross-Site Scripting* aparece consecutivamente no top 10 de vulnerabilidades desde 2004, variando entre a primeira, segunda, terceira e setima posição, nota-se que o grau de risco diminuiu com o passar dos anos, mas ainda continua em alta.

O ataque XSS é um tipo de "injeção", onde *scripts* maliciosos são injetados em *sites* benignos e confiáveis, e posteriormente executados pelo navegador do usuário sem saber que o *script* malicioso não pertence à página original (OWASP, 2016c). Através do *script*, o atacante adquire acesso a todos os *cookies* (informações de preferências de usuário e visitas a cada *site*), *tokens* de sessão (informações de permissão de acessos) ou outras informações confidenciais mantidas pelo navegador e usadas no *site* acessado (GROSSMAN et. al, 2011). O código *script* é geralmente desenvolvido na linguagem de programação *JavaScript*, mas também pode ser em Java ou VBScript (NUNAN, 2012).

Assim como qualquer outro ataque, o XSS só consegue penetrar na aplicação por meio de brechas, que neste caso, é a falta de validação ou validação inadequada dos campos de entradas da página. Sendo assim um campo de senha, pesquisa, comentário, mensagem e até mesmo áreas de redirecionamento de *link*s são possíveis vulnerabilidade para este ataque (WASSERMANN and SU, 2008).

O *Cross-site Scripting* pode ser dividido em três categorias: *stored* XSS (persistente), *reflected* XSS (refletido) e o DOM *Based* XSS (baseado em DOM). Estas se divergem pela quantidade de vítimas afetadas, o ambiente em que atuam e a permanência da ação (OWASP, 2016c).

O primeiro tipo chamado de *stored* XSS ocorre quando o *script* malicioso é inserido na página e fica permanentemente armazenado no servidor da aplicação, e exposto na página como uma informação comum, assim quando um usuário entra na página infectada o navegador lê o *script* como se fizesse parte da página original e o executa (VISSOTTO et al, 2015; OWASP, 2016c). Este ataque uma vez realizado consegue atingir mais vítimas do que as outras categorias, já que quando armazenado no servidor qualquer usuário que visite a página infectada será afetado sem a chance de se defender, o que torna esta categoria a mais perigosa (GROSSMAN et. al, 2011).

No *Reflected* XSS o código malicioso é inserido no *link* da página vulnerável e executado de maneira refletida como resposta à uma requisição no navegador do usuário alvo, sem a necessidade de estar armazenado (GROSSMAN et. al, 2011). Geralmente neste ataque a pessoa mal intencionada identifica a vulnerabilidade da aplicação, insere o *script* malicioso no *link* e o envia ao alvo, o usuário alvo reconhece o endereço da página original e acreditando que seja confiável clica no *link*, o navegador executa junto o *script* malicioso e como resposta à requisição da página, o

atacante recebe acesso e informações privadas do navegador. Este é o XSS mais frequente, no entanto afeta apenas os usuários que clicam no *link* infectado (OWASP, 2016c).

Por último diferente das outras duas categorias de XSS apresentadas em que o código malicioso é incorporado a página vulnerável, o DOM *Based* XSS interage com o ambiente DOM, ou seja, explora a vulnerabilidade do lado do cliente e não do servidor (GROSSMAN et al, 2011 and NUNAN, 2012).

> Um ataque do tipo DOM 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 d*ocument referrer* (ROCHA, 2013, pág.18).

Assim, na categoria DOM *Based* XSS o *script* malicioso injetado no conteúdo da página *web* ou enviado em um *link* tem acesso ao DOM, e modificam dinamicamente os nós e objetos da estrutura para realizar o ataque desejado, podendo alterar a página exibida a fim de denegrir sua imagem ou roubar informações do usuário (NUNAN, 2012).

Wassermann e Su (2008) afirmam que vários fatores são responsáveis pelas vulnerabilidades XSS, o primeiro é o fato de que os requisitos para um ataque XSS são mínimos, necessitando apenas de uma entrada de dados não confiável, o que ocorre com a maioria das aplicações *web*, em segundo lugar, a maioria das linguagens de programação de aplicações *web* não fornecem por padrão mecanismos para uma entrada segura de dados.

1.1.3 PREVENÇÃO DE CROSS-SITE SCRIPTING

No entanto, a OWASP (2017d) apresenta em seu *site* algumas regras de prevenção e tratamento desses fatores de vulnerabilidade, destinadas a impedir ataques das três categorias de XSS. No total são sete regras, sendo que a última referente à prevenção de DOM based XSS que contém outras sete sub regras. A organização também apresenta uma regra zero que especifica a necessidade de negar a entrada de qualquer dado não confiável que não se encaixe no contexto de uma das outras regras apresentadas pela organização, tendo como princípio que não há bons motivos para arriscar colocar dados não confiáveis em outros contextos pois é uma tarefa complexa e perigosa.

A primeira regra propriamente dita, é aplicada na inserção de dados não confiáveis diretamente no corpo HTML, em *tags* (marcadores de interpretação pelo navegador) normais como div, p, b, td, etc. Sendo assim, neste caso os dados não confiáveis, ou seja, dados manipulados pelo

usuário, devem ser codificados para a entidade HTML, de forma a evitar mudanças de interpretação. A codificação deve incluir os seguintes caracteres (OWASP, 2017d):

- **&** - substituído por **&**
- < - substituído por **<**
- > - substituído por **>**
- **“** - substituído por **"**
- **‘** - substituído por **'**
- **/** - substituído por **&#x2F.**

A regra dois determina que a entrada de dados inseguros em atributos comuns de HTML como largura, nome, valor, etc. devem ter todos os caracteres ASCII inferiores a 256 (exceto os alfanuméricos) codificados com entidade HTML no formato &#xHH, onde cada caractere é representado por &#x seguido do seu valor hexadecimal, por exemplo, a palavra “oi” convertida se torna “&#x6f&#x69” (OWASP, 2017d).

A terceira regra define a codificação dos dados não confiáveis inserido dinamicamente nos blocos de script e atributos manipuladores de eventos, ressaltando que o único local seguro para colocar dados não confiáveis é dentro de um valor de atributo citado com aspas, qualquer outro contexto de *javascript* não devem receber dados de entrada. Também não deve se usar “\” como escape, pois pode ser analisado como HTML ou sofrer ataque do tipo “escape-the-escape” onde o invasor envia outro caractere “\” e o código vulnerável se transforma em “\\” que pode ser interpretado como citação. Ainda como uma sub regra é recomendado diretrizes de tratamento de *JavaScript Object Notation* (JSON - formato leve para intercâmbio de dados) em contexto HMTL, como, a definição do atributo Content-Type do cabeçalho JSON para o tipo “application / json” , e a codificação dos dados inseguros inserido no JSON por meio de um script ou bloco HTML em entidade *JavaScript* ou HTML (OWASP, 2017d).

A regra quatro determina a codificação e validação de dados não confiáveis inseridos no valor de uma propriedade CSS. Qualquer outro contexto do CSS que não os valores de propriedades não devem receber dados dinamicamente, pois é perigoso, até mesmo propriedades complexas como de comportamento, personalização, URL e IE’s expression devem ser evitados. Também é necessário certificar que os valores dinâmicos das propriedades nunca comecem com "expression", e especificamente propriedades de URL não devem começar com "*javascript*". A codificação deve incluir todos os caracteres ASCII inferiores a 256 exceto os alfanuméricos, transformando-os no formato \HH, onde “HH” representa o valor hexadecimal do caractere (OWASP, 2017d).

A quinta regra especifica a codificação de dados não confiáveis no valor do parâmetro do método GET pelo Protocolo de Transferência de Hipertexto (HTTP), ou seja, o HTTP GET, a

codificação deve converter os caracteres com valores ASCII inferiores a 256 com o formato %HH para URLS, onde cada caractere é representado com o símbolo de porcentagem seguido do seu valor correspondente em hexadecimal. No entanto a codificação no formato URL não deve ser feita em URLS completas, se esse for o caso, a codificação deve ser realizada de acordo com o contexto, por exemplo, URLs orientadas pelo usuário em *link*s HREF devem ser codificadas conforme a regra 2, por estar no contexto de atributo HTML. Especifica-se também que os atributos devem estar em formato de citação, pois URL com atributos não citados podem ser facilmente quebradas por símbolos como % * + , - / ; < = > ^. A regra seis indica a utilização de bibliotecas de sanitização de HTML como HtmlSanitizer, OWASP Java HTML Sanitizer, SanitizeHelper, Ruby on Rails, entre outras, para aplicações que necessitam receber dados de entrada HTML (OWASP, 2017d).

A sétima e última regra refere-se à prevenção de ataque do tipo DOM *based* XSS, e específica algumas sub regras para a inserção de valores não confiáveis em alguns contextos por meio de *scripts*, onde sua primeira sub regra determina a codificação em entidade HTML dos dados inseguros inseridos no corpo HTML, seguido de codificação em *JavaScript*.em contraste com sub regra anterior, quando os dados são inseridos em atributos HTML é necessário a codificação unicamente em entidade *JavaScript*. Também quando inseridos em tributos CSS devem ser codificados para *JavaScript*. Já os dados inseridos em atributos de URL por sua vez devem ser codificados primeiramente para URL depois para *JavaScript*. A sétima regra também aconselha inserir valores inseguros de definição do DOM apenas em propriedades seguras com o TextContent, por exemplo, se usar a entrada do usuário para escrever em um elemento de *tag div* não use innerHtml, use innerText / textContent (OWASP, 2017e).

Além das regras mencionadas, a OWASP (2017d) também fornece regras complementares as quais aconselham a implementação de uma Content Security Policy(CSP) - políticas de segurança de conteúdo -, o uso do padrão HTTPOnly para *cookies* - um padrão responsável por proibir o acesso aos *cookies* do navegador por meio de *javascript*s -, cabeçalho de resposta X-XSS-Protection - utilizado para auxiliar o navegador na detectação e inibição de ataques XSS- e funções de escape automáticas. A organização também ressalta que não é necessário permitir ou aplicar todas as regras apresentadas, pois cada regra atende um tipo de necessidade e sua aplicação depende das características de cada projeto, por exemplo, uma organização pode definir que apenas a regra 1 e 2 sejam suficientes para suas necessidades.

A fim de alcançar o objetivo principal deste trabalho, o de expor meios de prevenção de ataques XSS eficazes encontrados na literatura e abordar características do ataque *Cross-Site Scripting*, as técnicas apresentadas, até o momento, tiveram sua eficácia e viabilidade avaliadas por

meio de um experimento de aplicação e análise. Sendo assim, enfatiza-se a grande importância de se utilizar métodos de prevenção, uma vez que existem várias ferramentas acessíveis para isso.

2 METODOLOGIA

Para realização do experimento de avaliação deste estudo foi selecionada uma aplicação *web* de código simples e sem implementação de técnicas de seguranças; a aplicação trata-se de um fórum escrito em PHP, formatado com Bootstrap e com armazenamento em banco de dados MySQL. O fórum possui três páginas: a página inicial onde são exibidos os tópicos de discussão já cadastrados, a página de cadastro destes tópicos e a página de visualização onde são apresentados os comentários do tópico escolhido e também a possibilidade de inserção de novos comentários.

A partir da aplicação original foi feito uma cópia, a qual foi revisada e corrigida conforme as técnica e regras de prevenção XSS apresentadas. Devido ao tamanho da aplicação e a quantidade de ações que realiza foi necessária a aplicação das regras 1, 2 e 5, além das regras complementares de uso do padrão HTTPOnly para *cookies* e cabeçalho de resposta X-XSS-Protection, da OWASP,

Obtendo as duas versões da página, uma original sem técnica de segurança intitulada aplicação “A” e a outra cópia devidamente alterada e corrigida referida como aplicação “B”, iniciou-se a avaliação, onde foi aplicado um escaneamento, em ambas as versões, com o scanner de detectação e exploração de vulnerabilidades OWASP Zed Attack Proxy Project (ZAP), que conta com ferramentas para a detectação automática e manual de diversos tipos de ataque, incluindo tipo XSS em suas três categorias (OWASP, 2017f).

O escaneamento de cada aplicação foi feito de modo automático utilizando inicialmente com configuração padrão o escaner *spider*, uma função oferecida pela ferramenta ZAP para identificação automatica dos recursos (URLs) e a estrutura de funcionamento da aplicação (OWASP, 2017f), alem disso, este escaneamento também identifica algumas falhas referentes a não implementação de boas práticas de programação.

Posteriormente, com a estrutura da aplicação devidamente identificada, foi realizado o escaneamento de varredura ativa responsável por encontrar possíveis vulnerabilidades usando ataques conhecidos pela OWASP. Para obter resultados específicos a varredura ativa foi configurada para alertar apenas vulnerabilidade apresentadas como XSS além do CRLF *injection* (***C**arriage **R**eturn **L**ine **F**eed*), vulnerabilidade que permite a inserção caracteres de finalização de instrução em um campo de entrada de dados, que a apesar ser considerada uma vulnerabilidade à

parte do *Cross-site Scripting*, possibilita a iniciação de um script malicioso e pode ser utilizada para realizar um ataque XSS (OWASP, 2017g).

3 RESULTADOS

A partir do escaneamento realizado nas aplicações “A” e “B” foi emitido um relatório contendo o número de falhas encontradas, local vulnerável, grau de risco, descrição, solução e outros detalhes. Como a ferramenta *spider* busca diversas falhas acrescentando vulnerabilidades além do tipo *Cross-site Scripting* no relatório final, este trabalho apresenta apenas os resultados relevantes, ou seja, os resultados referentes à XSS, o relatório completo de cada aplicação pode ser conferido em ANEXO A – Relatório ZAP aplicação “A” e ANEXO B – Relatório ZAP aplicação “B”.

O escaneamento *spider* realizado na aplicação “A” resultou em alguns alertas de vulnerabilidades, sendo dois referentes a ataques *Cross-sites Scripting*, classificados pela ferramenta como risco baixo. Estas vulnerabilidades são apresentadas na Tabela 1 com suas respectivas descrições e o número de instâncias afetadas.

Tabela 1 - Vulnerabilidades XSS Encontradas Na Aplicação “A” Pelo Scanner *Spider*.

Tipo de vulnerabilidade	Descrição	Instâncias afetadas
Web Browser XSS Protection Not Enabled.	A proteção XSS do navegador da *Web* não está habilitada pela configuração do cabeçalho de resposta HTTP 'X-XSS-Protection'.	5
Cookie No HttpOnly Flag.	O *cookie* foi configurado sem o sinalizador HttpOnly.	6
TOTAL		11

FONTE: Adaptado de Zed Attack Proxy Project (OWASP, 2017f).

Por sua vez, no escaneamento de varredura ativa feito na aplicação “A” foram explorados cinco tipos de vulnerabilidades. No entanto, apenas duas se mostraram presentes como conforme apresenta a Tabela 2.

Tabela 2 - Vulnerabilidades exploradas na aplicação "A" com a varredura ativa.

Vulnerabilidade explorada	Quantidade de ataques realizados	Quantidade de Instancias afetadas	Porcentagem de Sucesso dos ataques
Reflected XSS	282	9	0,03%
CRLF injection	637	0	0,00%
Stored XSS (*prime*)	91	0	0,00%
Stored XSS (*spider*)	14	0	0,00%
Stored XSS	8	2	0,25%
TOTAL	1032	11	0,01%

FONTE: Adaptado de Zed Attack Proxy Project (OWASP, 2017f).

Vale destacar que conforme pode ser observado na Tabela 1 e 2, as vulnerabilidades XSS encontradas pelo *Spider* referem-se a má configuração do protocolo HTTP na comunicação da aplicação; já as encontradas na varredura ativa são consequências de falta de validação e tratamento dos dados de entrada. No total, foram quatro tipos de vulnerabilidades XSS e 22 instâncias vulneráveis encontradas na aplicação "A".

Já na aplicação "B" tanto o scanner *Spider* quanto avarredura ativa não encontraram nenhuma vulnerabilidade XSS, apenas as outras vulnerabilidades não tratadas neste estudo. A comparação entre os resultados obtidos nas aplicações "A" e "B" podem ser observados na Tabela 3.

Tabela 3 - Comparação de resultados das aplicações "A" e "B".

Vulnerabilidades	Instância afetadas na aplicação A	Instância afetadas na aplicação B
Reflected XSS	9	0
Stored XSS	2	0
Cookie No HttpOnly Flag	6	0
Web Browser XSS Protection Not Enabled	5	0
TOTAL	22	0

FONTE: Adaptado de Zed Attack Proxy Project (OWASP, 2017f).

Para melhor exemplificar a correção e prevenção das vulnerabilidades que foram realizadas no experimento deste estudo, o Quadro 1 apresenta para cada vulnerabilidade encontrada na aplicação "A" um trecho de código vulnerável e sua respectiva correção implementada na aplicação "B".

Ressalta-se que a aplicação web escolhida para o experimento foi desenvolvida em linguagem PHP, portanto os trechos de códigos apresentados no Quadro 1 aparecem parte em PHP e parte em HTML.

Quadro 1 - Exemplos de código vulnerável encontrados na aplicação "A" e soluções utilizadas na aplicação "B".

Vulnerabilidades	**Código vulnerável (Aplicação A)**	**Código corrigido (Aplicação B)**
Reflected XSS	<input type="hidden" class="form-control" name="id" id="inputid" value="<?php echo $_GET["id"]; ?>">	[...] value="<?php echo **htmlentities**($_GET["titulo"], ENT_QUOTES | ENT_SUBSTITUTE | ENT_HTML401, 'UTF-8', false); ?>">
Stored XSS	echo "<td>" . $topico->texto . "</td>";	echo "<td>" . **htmlspecialchars**($topico->texto, ENT_QUOTES | ENT_SUBSTITUTE | ENT_HTML401, 'UTF-8', false) . "</td>";
Cookie No HttpOnly Flag	<?php session_start(); ?>	<?php **ini_set('session.cookie_httponly',1);** session_start(); ?>
Web Browser XSS Protection Not Enabled	<?php header('Location: index.php'); ?>	<?php **header("X-XSS-Protection: 1");** header('Location: index.php'); ?>

FONTE: Adaptado de Zed Attack Proxy Project (OWASP, 2017f).

4 DISCUSSÃO

Como pode ser observado nos resultados, a vulnerabilidade XSS com maior quantidade de instâncias afetadas na aplicação "A" foi a *Reflected* XSS, em contrapartida a vulnerabilidade de menor ocorrência foi o *Stored* XSS, resultando em dois extremos de incidências de vulnerabilidades de alto risco. No entanto, a aplicação de três regras fornecidas pela OWASP foi suficiente para evitar todas as ocorrências de ambos os ataques, mostrando não ser uma tarefa complexa realizar a prevenção. Já as vulnerabilidades *Cookie No HttpOnly Flag* e *Web Browser XSS Protection Not Enabled*, classificadas como baixo risco, foram mitigadas por meio de duas regras complementares apresentadas.

No que diz respeito à complexidade e limitações do experimento não foram encontrados muitos obstáculos uma vez que a linguagem PHP, a qual a aplicação foi desenvolvida, já oferece, por padrão, funções de codificação. Sendo assim, a escolha da função e configuração necessária fica a critério do desenvolvedor baseando-se na necessidade da aplicação a ser desenvolvida. Wehrmann (2004) consente que a proteção e validação das entradas do usuário é totalmente proporcional à complexidade dos requisitos da aplicação *Web*. Logo ele complementa que as aplicações resistentes a ataques possuem, principalmente, filtros de caracteres (que são demasiadamente utilizados por *scripts*), como mencionado nesse estudo.

Ouro fato que favorece a aplicação das regras de segurança e que a organização OWASP, ao mesmo tempo em que menciona as regras necessárias também oferece bibliotecas de sanitização e páginas que auxiliam a implementação da prevenção para cada tipo de projeto. Constatando que a proteção e mitigação de ataques do tipo XSS pode ser facilmente iniciada - e futuramente aprimorada - com a utilização de *softwares* livres e com um pouco de conhecimento na área, justifica que a grande preocupação da abordagem sobre as técnicas de mitigação desse tipo de ataque, está, especificamente, na falta de interesse ou no pouco conhecimento que se tem sobre o assunto, consentido com o que Wehrmann (2004) e de Carvalho (2016) constatam em seus estudos que a maioria dos profissionais responsáveis pelo desenvolvimento dessas aplicações estão pouco informados sobre o quão perigoso é o XSS, e justificam dizendo que, por conta disso, eles não se preocupam em tratar as vulnerabilidades adequadamente.

Estudos semelhantes a este como de Barbosa e Reinaldo (2015), Oliveira (2017) e Endler (2002) também mencionam as regras da OWASP como método de prevenção, no entanto, não apresentam sua aplicação tampouco o desempenho na solução do problema. Em contrapartida, os resultados encontrados neste estudo mostraram-se muito satisfatórios, no que se refere à implementação das técnicas descritas para a mitigação do ataque XSS. De fato, ao aplicá-las e compará-las pode-se observar a redução de 100% de ocorrências dos ataques em ambos os graus.

Outro resultado satisfatório, em relação a prevenção *Cross-site scripting*, foi mencionado por Rao et al. (2017). No entanto, ao invés de usar funções padrões ou biblioteca para a codificação e sanitização dos dados de entrada não confiáveis, como este estudo, o autor desenvolveu sua própria função para a tarefa. Estes dados são muito semelhantes aos encontrados por Vissotto et al. (2015) que obteve sucesso em desenvolver também uma função de codificação no próprio banco de dados. Ao comparar os resultados destes trabalhos com aqueles obtidos nesse estudo pode-se concluir que independente da forma adotada para codificação de dados seguros o resultado é satisfatório. No entanto, neste trabalho pode ser observado que apenas a função de validação de entradas não é suficiente para a total mitigação do ataque; outras técnicas e métodos apresentados

foram necessários para corrigir outras vulnerabilidades de ataques XSS, como *Cookie No HttpOnly Flag e Web Browser XSS Protection Not Enabled.*

Vale observar que as principais técnicas apresentadas e implementadas neste estudo são concentradas na codificação e sanitização das saídas de dados não confiáveis uma vez já inseridos na aplicação, trabalhando com uma "lista branca" que permite a inserção de dados não confiáveis apenas em contextos específicos tratados. Este modelo de prevenção, apesar de avaliado por Nadji (2009) como apenas um tratamento inicial, suscetível a erros de inconsistência do navegador-servidor e a ataques mais complexos, neste estudo, especificamente, se mostrou totalmente eficaz, levantando a ideia de que a prevenção apresentada tem o desempenho relacionado ao tipo do projeto em que é aplicada.

A OWASP (2017d) ressalta que embora as regras oferecidas não tornem a aplicação totalmente imune a dados não confiáveis, elas previnem a grande maioria dos ataques conhecidos, como constatado neste estudo, onde a ferramenta ZAP realizou 1035 ataques XSS conhecidos e não obteve nenhum sucesso na aplicação que implementou as regras aqui apresentadas. No entanto apesar do resultado satisfatório é sempre necessária a busca por mais conhecimento sobre o assunto e sobre novas forma de efetivar a segurança nas aplicações, assim como Grossman (2011) concorda e afirma que mesmo que existam formas de evitar que os problemas XSS mais óbvios ocorram, é impossível proteger seus recursos da *web* completamente, portanto, *webmasters* e desenvolvedores precisam estar sempre atualizados com as mais recentes vulnerabilidades e estratégias de ataque, a fim de contorná-las e resolvê-las.

5 CONCLUSÃO

As vulnerabilidades *Web* são, em suma, uma das principais e mais temidas falhas que uma empresa de TI pode enfrentar em todo o ciclo de vida de suas aplicações *web*, visto que, essas vulnerabilidades ao serem exploradas maliciosamente, podem denegrir a imagem da mesma, prejudicá-las judicial e financeiramente ou até mesmo às falir.

Este trabalho, portanto, apresentou e avaliou meios de prevenção de vulnerabilidade *Cross-site scripting* em aplicações *web* com base nas diretrizes fornecidas pela organização OWASP. A partir do experimento de implementação dos meios de prevenção em uma aplicação *web*, que simula um fórum, e com escaneamento realizado com a ferramenta ZAP, foi possível observar a redução total das vulnerabilidades XSS, constatando a eficácia das técnicas apresentadas.

Além disso, o trabalho também abordou características e o funcionamento de ataques *Cross-site Scripting* em suas três categorias a fim de gerar conhecimento a desenvolvedores e usuários sobre este ataque pouco abordado, porém, muito recorrente.

Futuramente, sugere-se a extensão do presente estudo para outras plataformas de mitigação, bem como a exploração e prevenção de outros tipos de ataques críticos apresentados no relatório "Top Ten", além disso, pode se abordar um modelo de ciclo de desenvolvimento seguro para evitar estes ataques.

REFERÊNCIAS

BARBOSA, Eber Donizeti; CASTRO, Reinaldo de oliveira. Desenvolvimento de Software Seguro: Conhecendo e Prevenindo Ataques SQL Injection e Cross-site Scripting (XSS). **Revista T.I.S**, v. 4, n. 1, 2015.

BEAL, Adriana. Gestão estratégica da informação: como transformar a informação e a tecnologia da informação em fatores de crescimento e alto desempenho nas organizações. **São Paulo: Atlas**, 2004.

DANTAS, Marcus Leal. Segurança da informação: uma abordagem focada em gestão de riscos. **Recife: Livro Rápido-Elogica**, 2011.

DE CARVALHO, Alan Henrique Pardo. Segurança de aplicações *web* e os dez anos do relatório OWASP Top Ten: o que mudou?. **FaSCi-Tech**, v. 1, n. 8, 2016.

ENDLER, David. **The evolution of cross site scripting attacks**. Technical report, iDEFENSE Labs, 2002.

ESTADÃO, **65% dos ataques de hackers miram pequenas empresas, diz estudo**. 2017. Disponível em: <http://pme.estadao.com.br/noticias/pme,65-dos-ataques-de-hackers-miram-pequenas-empresas-diz-estudo,70001746157,0.htm>. Acesso em: 16 de abr. 2017.

G1, **Brasil é país que recebe mais ataques cibernéticos da América Latina**. 2016. Disponível em: <http://g1.globo.com/bom-dia-brasil/noticia/2016/12/brasil-e-pais-que-recebe-mais-ataques-ciberneticos-da-america-latina.html>. Acesso em: 20 de abr. 2017.

GROSSMAN, Jeremiah. **XSS Attacks: Cross-site scripting exploits and defense**. Syngress, 2007.

NADJI, Yacin; SAXENA, Prateek; SONG, Dawn. Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense. In: **NDSS**. 2009. p. 20.

NUNAN, Angelo Eduardo. **Detecção de Cross-Site Scripting em páginas *Web***. 2012. 104 f. Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas, Manaus, 2012.

OLIVEIRA, João Marcos Barbosa. Auditoria de Segurança da Informação em Aplicações *Web*: Estudo de caso sobre o ataque Cross-Site Scripting. **Gestão da segurança da Informação-Unisul Virtual**, 2017.

OWASP. **About The Open *Web* Application Security Project**. 2017. Disponível em: <https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project>. Acesso: 20 set. 2017a.

OWASP. **Category: OWASP Top Ten Project**. Disponível em: <https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project>. Acesso: 20 set. 2017b.

OWASP. **Cross-site Scripting (XSS)**. 2016. Disponível em: <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>. Acesso: 10 out. 2017c.

OWASP. **XSS (Cross Site Scripting) Prevention Cheat Sheet**. 2017. Disponível em: <https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet>. Acesso em: 17 out. 2017d.

OWASP. **DOM based XSS Prevention Cheat Sheet**. 2017. Disponível em: <https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet>. Acesso em: 17 out. 2017e.

OWASP. **OWASP Zed Attack Proxy Project**. Disponível em: <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>. Acesso em: 15 out. 2017f.

OWASP. **CRLF Injection**. 2016. Disponível em: <https://www.owasp.org/index.php/CRLF_Injection>. Acesso em: 15 out. 2017g.

PWC. **Pesquisa Global de Segurança da Informação 2016**. Disponível em: <http://www.pwc.com.br/pt/estudos/servicos/consultoria-negocios/2016/giss-pesquisa-global-seguranca-informacao-2016.html>. Acesso em: 19 abr. 2017.

RAO, G. Rama Koteswara et al. CROSS SITE SCRIPTING ATTACKS AND PREVENTIVE MEASURES. 2017.

ROCHA, T. Detecção Automática de Ataques de Cross-Site Scripting em Páginas. 2013. 66 f. Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas, Manaus, 2013.

VISSOTTO, A. C. et al. **Prevenção de Ataques: XSS Residente e SQL Injection em Banco de Dados**. Caderno de Estudos Tecnológicos FATEC, Bauru, v. 3, n. 1, 2015. Disponível em: <http://www.fatecbauru.edu.br/ojs/index.php/CET/article/view/193>. Acesso em: 15 out. 2017.

W3C. **Document Object Model (DOM)**. 2005. Disponível em: <https://www.w3.org/DOM/>. Acesso em: 15 set. 2017.

WASSERMANN, Gary; SU, Zhendong. Static detection of cross-site scripting vulnerabilities. In: **Proceedings of the 30th international conference on Software engineering**. ACM, 2008. p. 171-180.

WEHRMANN, Christoph. Cross Site Scripting. In: **Seminar: Security in Communication Systems.** p. 2004.

ANEXO A - Relatório ZAP aplicação “A”.

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	2
Medium	2
Low	3
Informational	0

Alert Detail

High (Medium)	Cross Site Scripting (Reflected)
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/*JavaScript*, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted *link* laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://localhost/forum_vuneravel/cadtopico.php?id=%3Cbr+%2F%3E%0A%3Cb%3ENotice%3C%2Fb%3E%3A++Undefined+index%3A+id+in+%3Cb%3EC%3A%5Cxampp%5Chtdocs%5Cforum_vuneravel%5Ccadtopico.php%3C%2Fb%3E+on+line+%3Cb%3E65%3C%2Fb%3E%3Cbr+%2F%3E%0A&texto&titulo=ZAP
Method	GET
Parameter	Referer
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/cadtopico.php?id=%3Cbr+%2F%3E%0A%3Cb%3ENotice%3C%2Fb%3E%3A++Undefined+index%3A+id+in+%3Cb%3EC%3A%5Cxampp%5Chtdocs%5Cforum_vuneravel%5Ccadtopico.php%3C%2Fb%3E+on+line+%3Cb%3E65%3C%2Fb%3E%3Cbr+%2F%3E%0A&texto=&titulo=%3C%2Ftd%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Ctd%3E
Method	GET
Parameter	titulo
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/cadtopico.php?id=%3Cbr+%2F%3E%0A%3Cb%3ENotice%3C%2Fb%3E%3A++Undefined+index%3A+id+in+%3Cb%3EC%3A%5Cxampp%5Chtdocs%5Cforum_vuneravel%5Ccadtopico.php%3C%2Fb%3E+on+line+%3Cb%3E65%3C%2Fb%3E%3Cbr+%2F%3E%0A&texto&titulo=ZAP

Method	GET
Parameter	User-Agent
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/cadtopico.php?id=%3Cbr+%2F%3E%0A%3Cb%3ENotice%3C%2Fb%3E%3A++Undefined+index%3A+id+in+%3Cb%3EC%3A%5Cxampp%5Chtdocs%5Cforum_vuneravel%5Ccadtopico.php%3C%2Fb%3E+on+line+%3Cb%3E65%3C%2Fb%3E%3Cbr+%2F%3E%0A&texto&titulo=ZAP
Method	GET
Parameter	Host
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	User-Agent
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/index.php?query=%3C%2Ftd%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Ctd%3E
Method	GET
Parameter	query
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/cadtopico.php?id=%3Cbr+%2F%3E%0A%3Cb%3ENotice%3C%2Fb%3E%3A++Undefined+index%3A+id+in+%3Cb%3EC%3A%5Cxampp%5Chtdocs%5Cforum_vuneravel%5Ccadtopico.php%3C%2Fb%3E+on+line+%3Cb%3E65%3C%2Fb%3E%3Cbr+%2F%3E%0A&texto=%3C%2Ftd%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Ctd%3E&titulo=ZAP
Method	GET
Parameter	texto
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	Referer
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	Host
Attack	</td><script>alert(1);</script><td>
Evidence	</td><script>alert(1);</script><td>
Instances	9
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.

	Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Source ID	1
High (Medium)	**Cross Site Scripting (Persistent)**
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST

	requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	titulo
Attack	</td><script>alert(1);</script><td>
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	texto
Attack	</td><script>alert(1);</script><td>
Instances	2
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable

	inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Other information	Source URL: http://localhost/forum_vuneravel/cadtopico.php?id=%3Cbr+%2F%3E%0A%3Cb%3ENotice%3C%2Fb%3E%3A++Undefined+index%3A+id+in+%3Cb%3EC%3A%5Cxampp%5Chtdocs%5Cforum_vuneravel%5Ccadtopico.php%3C%2Fb%3E+on+line+%3Cb%3E65%3C%2Fb%3E%3Cbr+%2F%3E%0A&texto&titulo=ZAP
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Source ID	1
Medium (Medium)	**X-Frame-Options Header Not Set**
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://localhost/forum_vuneravel/
Method	GET
Parameter	X-Frame-Options
URL	http://localhost/forum_vuneravel/cadtopico.php
Method	GET
Parameter	X-Frame-Options
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	X-Frame-Options
URL	http://localhost/forum_vuneravel/vistopico.php
Method	GET
Parameter	X-Frame-Options
URL	http://localhost/forum_vuneravel/vistopico.php?id=0
Method	GET
Parameter	X-Frame-Options
Instances	5
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3
Medium (Medium)	**Application Error Disclosure**
Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	http://localhost/forum_vuneravel/vistopico.php?id=0
Method	GET
Evidence	You have an error in your SQL syntax
URL	http://localhost/forum_vuneravel/vistopico.php
Method	GET
Evidence	You have an error in your SQL syntax

Instances	2
Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	200
WASC Id	13
Source ID	3
Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://localhost/forum_vuneravel/js/jquery.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/vistopico.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/vistopico.php?id=0
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/css/dataTables.bootstrap.min.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/css/bootstrap.min.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/cadtopico.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/js/dataTables.bootstrap.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/js/jquery.dataTables.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_vuneravel/js/bootstrap.min.js
Method	GET
Parameter	X-Content-Type-Options
Instances	11
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx

	https://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3
Low (Medium)	**Cookie No HttpOnly Flag**
Description	A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
URL	http://localhost/forum_vuneravel/index.
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
URL	http://localhost/forum_vuneravel/
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
URL	http://localhost/forum_vuneravel/cadtopico.php
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
URL	http://localhost/forum_vuneravel/vistopico.php?id=0
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
URL	http://localhost/forum_vuneravel/cadtopico.php?id=%3Cbr+%2F%3E%0A%3Cb%3ENotice%3C%2Fb%3E%3A++Undefined+index%3A+id+in+%3Cb%3EC%3A%5Cxampp%5Chtdocs%5Cforum_vuneravel%5Ccadtopico.php%3C%2Fb%3E+on+line+%3Cb%3E65%3C%2Fb%3E%3Cbr+%2F%3E%0A&texto&titulo=ZAP
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
URL	http://localhost/forum_vuneravel/vistopico.php
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
Instances	6
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
CWE Id	16
WASC Id	13
Source ID	3
Low (Medium)	**Web Browser XSS Protection Not Enabled**
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	http://localhost/forum_vuneravel/vistopico.php
Method	GET
Parameter	X-XSS-Protection
URL	http://localhost/forum_vuneravel/index.php
Method	GET
Parameter	X-XSS-Protection
URL	http://localhost/forum_vuneravel/vistopico.php?id=0
Method	GET
Parameter	X-XSS-Protection
URL	http://localhost/forum_vuneravel/
Method	GET
Parameter	X-XSS-Protection
URL	http://localhost/forum_vuneravel/cadtopico.php

Method	GET
Parameter	X-XSS-Protection
Instances	5
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it: X-XSS-Protection: 1; mode=block X-XSS-Protection: 1; report=http://www.example.com/xss The following values would disable it: X-XSS-Protection: 0 The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit). Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14
Source ID	3

ANEXO B – Relatório ZAP aplicação “B”.

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	1
Informational	0

Alert Detail

Medium (Medium)	**X-Frame-Options Header Not Set**
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://localhost/forum_protegido/
Method	GET
Parameter	X-Frame-Options
URL	http://localhost/forum_protegido/index.php
Method	GET
Parameter	X-Frame-Options
Instances	2
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3
Medium (Medium)	**Application Error Disclosure**
Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	http://localhost/forum_protegido/vistopico.php?id=0
Method	GET
Evidence	You have an error in your SQL syntax
URL	http://localhost/forum_protegido/vistopico.php
Method	GET
Evidence	You have an error in your SQL syntax
Instances	2
Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	200
WASC Id	13
Source ID	3
Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://localhost/forum_protegido/css/dataTables.bootstrap.min.css
Method	GET

Parameter	X-Content-Type-Options
URL	http://localhost/forum_protegido/js/jquery.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_protegido/css/bootstrap.min.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_protegido/js/bootstrap.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_protegido/
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_protegido/js/dataTables.bootstrap.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_protegido/index.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://localhost/forum_protegido/js/jquery.dataTables.min.js
Method	GET
Parameter	X-Content-Type-Options
Instances	8
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3

ANEXO C - Diretrizes Para Autores

Os trabalhos devem ser redigidos em português, com uso obrigatório da norma culta. Os nomes dos autores, bem como a afiliação institucional de cada um, devem ser inseridos nos campos adequados a serem preenchidos durante a submissão e devem aparecer no arquivo. A Revista Eletrônica de Ciências Humanas sugere que o número máximo de autores por artigo seja 6 (seis). Artigos com número superior a 6 (seis) serão considerados exceções e avaliados pelo Conselho Editorial que poderá solicitar a adequação. Pesquisas feitas com seres humanos e animais devem, obrigatoriamente, citar a aprovação da pesquisa pelo respectivo Comitê de Ética, citando o protocolo de aprovação. O não atendimento de tal proposta pode implicar em recusa de sua publicação. Da mesma forma, o plágio implicará na recusa do trabalho.
Os autores dos artigos aceitos poderão solicitar a tradução do artigo para língua inglesa aos tradutores indicados pela revista e reenviar. Os custos com a tradução serão de responsabilidade dos autores.
O periódico disponibilizará aos leitores o conteúdo digital em ambos os idiomas, português e inglês.
O uso da norma culta da Língua Portuguesa e a obediência às normas da Revista são de total responsabilidade dos autores. A não obediência a esses critérios implicará na recusa imediata do trabalho.

Apresentação do Material

Sugere-se um número máximo de 20 páginas, incluindo referências, figuras, tabelas e quadros. Os textos devem ser digitados em Fonte Times New Roman, tamanho 12, espacejamento 1,5, justificado, exceto Resumo e Abstract. Devem ser colocadas margens de 2 cm em cada lado.
As Figuras: gráficos, imagens, desenhos e esquemas deverão estar inseridas no texto, apresentar boa qualidade, estar em formato JPEG, com resolução de 300dpi com 15cm x 10cm. O número de figuras deve ser apenas o necessário à compreensão do trabalho. Não serão aceitas imagens digitais artificialmente 'aumentadas' em programas computacionais de edição de imagens. As figuras devem ser numeradas em algarismos arábicos segundo a ordem em que aparecem e suas legendas devem estar logo abaixo.
Tabelas e Quadros: deverão ser numerados consecutivamente com algarismos arábicos e encabeçados pelo título. As tabelas e os quadros devem estar inseridos no texto. Não serão admitidas as tabelas e quadros inseridos como Figuras.

Títulos de tabelas e quadro e legendas de figuras deverão ser escritos em tamanho 11 e com espaço simples entre linhas.

Citação no texto: deve-se seguir as Normas da ABNT (NBR 10520, 2003). As citações deverão aparecer no texto, seguidas pelo ano de publicação. As chamadas pelo sobrenome do autor, pela instituição responsável ou título podem ser: a) incluídas na sentença: sobrenome (ano). Ex.: Gomes, Faria e Esper (2006) ou b) entre parênteses: (SOBRENOME, ano). Ex.: (GOMES; FARIA; ESPER, 2006). Quando se tratar de citação direta (transcrição literal), indicar, após o ano, a página de onde o texto foi extraído. O trecho transcrito deverá estar entre aspas quando ocupar até três linhas. As citações diretas com mais de três linhas devem ser destacadas com recuo de 4 cm da margem esquerda, ser escritas com letra menor que a do texto utilizado, com espaçamento entre linhas menor do que o utilizado no texto e sem as aspas. Citações indiretas de vários documentos simultaneamente devem constar em ordem alfabética (como nas referências). Citação de citação: autor citado (ano apud AUTOR, ano). Deve-se fazer a referência do autor lido. Ex.: Pádua (1996

apud FERNANDES, 2012, p. 5) salienta que “[...] pesquisa é toda atividade voltada para a solução de problemas [...]”.

Teses, dissertações e monografias, solicitamos que sejam utilizados apenas documentos dos últimos três anos e quando não houver o respectivo artigo científico publicado em periódico. Esse tipo de referência deve, obrigatoriamente, apresentar o *link* que remeta ao cadastro nacional de teses da CAPES e aos bancos locais das universidades que publicam esses documentos no formato pdf.
Grafia de termos científicos, comerciais, unidades de medida e palavras estrangeiras: os termos científicos devem ser grafados por extenso, em vez de seus correspondentes simbólicos abreviados. Para unidades de medida, deve-se utilizar o Sistema Internacional de Unidades. Palavras em outras línguas devem ser evitadas nos textos em português, utilizar preferentemente a sua tradução. Na impossibilidade, os termos estrangeiros devem ser grafados em itálico. Toda abreviatura ou sigla deve ser escrita por extenso na primeira vez em que aparecer no texto.

Estrutura do Artigo

PESQUISAS ORIGINAIS devem ter no máximo 20 páginas com até 40 citações; organizar da seguinte forma:

Título em português: caixa alta, centrado, negrito, conciso, com um máximo de 25 palavras;

Título em inglês (obrigatório): caixa alta, centrado. Versão do título em português;

Autor(es): O(s) nome(s) completo(s) do(s) autor(es) e seus títulos e afiliações à Sociedade ou Instituições. Indicar com asterisco o autor de correspondência. Ao final das afiliações fornecer o e-mail do autor de correspondência.

Resumo: parágrafo único sem deslocamento, fonte tamanho 11, espaço 1, justificado, contendo entre 150 e 250 palavras. Deve conter a apresentação concisa de cada parte do trabalho, abordando objetivo(s), método, resultados e conclusões. Deve ser escrito sequencialmente, sem subdivisões. Não deve conter símbolos e contrações que não sejam de uso corrente nem fórmulas, equações, diagramas;

Palavras-chave: de 3 a 5 palavras-chave, iniciadas por letra maiúscula, separadas e finalizadas por ponto.

Abstract (obrigatório): fonte tamanho 11, espaço 1, justificado, deve ser a tradução literal do resumo;

Keywords: a apresentação deverá ser a mesma das Palavras-chave em Português.

Introdução: deve apresentar o assunto a ser tratado, fornecer ao leitor os antecedentes que justificam o trabalho, incluir informações sobre a natureza e importância do problema, sua relação com outros estudos sobre o mesmo assunto, suas limitações. Essa seção deve representar a essência do pensamento do pesquisador em relação ao assunto estudado e apresentar o que existe de mais significante na literatura científica. Os objetivos da pesquisa devem figurar como o último parágrafo desse item.

Método: destina-se a expor os meios dos quais o autor se valeu para a execução do trabalho. Pode ser redigido em corpo único ou dividido em subseções. Especificar tipo e origem de produtos e equipamentos utilizados. Citar as fontes que serviram como referência para o método escolhido.

Pesquisas feitas com seres humanos e animais devem, obrigatoriamente, citar a aprovação da pesquisa pelo respectivo Comitê de Ética, citando o protocolo de aprovação.

Resultados: Nesta seção o autor irá expor o obtido em suas observações. Os resultados poderão estar expressos em quadros, tabelas, figuras (gráficos e imagens). Os dados expressos não devem ser repetidos em mais de um tipo de ilustração.

Discussão: O autor, ao tempo que justifica os meios que usou para a obtenção dos resultados, deve contrastar esses com os constantes da literatura pertinente; estabelecer relações entre causas e efeitos; apontar as generalizações e os princípios básicos, que tenham comprovações

nas observações experimentais; esclarecer as exceções, modificações e contradições das hipóteses, teorias e princípios diretamente relacionados com o trabalho realizado; indicar as aplicações teóricas ou práticas dos resultados obtidos, bem como, suas limitações; elaborar, quando possível, uma teoria para explicar certas observações ou resultados obtidos; sugerir, quando for o caso, novas pesquisas, tendo em vista a experiência adquirida no desenvolvimento do trabalho e visando a sua complementação.

Conclusões: Devem ter por base o texto e expressar com lógica e simplicidade o que foi demonstrado com a pesquisa, não se permitindo deduções. Devem responder à proposição.

Agradecimentos (opcionais): O autor deve agradecer às fontes de fomentos e àqueles que contribuíram efetivamente para a realização do trabalho. Agradecimento a suporte técnico deve ser feito em parágrafo separado.

Referências (e não bibliografia): Espaço simples entre linhas e duplo entre uma referência e a próxima. As referências devem ser numeradas na ordem em que aparecem no texto. A lista completa de referências, no final do artigo, deve estar de acordo com as normas da ABNT (NBR 6023, 2003). Quando a obra tiver até três autores, todos devem ser citados. Mais de três autores, indicar o primeiro, seguido de et al. Alguns exemplos:

Artigo publicado em periódico:
LUDKE, M.; CRUZ, G. B. dos. Aproximando universidade e escola de educação básica pela pesquisa. Caderno de pesquisa, São Paulo, v. 35, n. 125, p. 81-109, maio/ago. 2005.

Artigo publicado em periódico em formato eletrônico:
SILVA JUNIOR, N. A. da. Satisfação no trabalho: um estudo entre os funcionários dos hotéis de João Pessoa. Psico-USF, Itatiba, v. 6, n. 1, p. 47-57, jun. 2001. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1413-82712001000100007&lng=pt&nrm=iso>. Acesso em: 13 jul. 2015.

Livro (como um todo)
MENDONÇA, L. G. et al. Matemática financeira. 10. ed. Rio de Janeiro: Editora FGV, 2010.

Capítulo de livro
MARTÍN. E.; SOLÉ, I. A aprendizagem significativa e a teoria da assimilação. In: COLL, C.; MARCHESI, A.; PALACIOS, J. (Org.). Desenvolvimento psicológico e educação: psicologia da educação escolar. 2. ed. Porto Alegre: Artmed, 2008. cap. 3, p. 60-80.

Artigos de Revisão

Poderão ser aceitos para submissão, desde que abordem temas de interesse, atualizados. Devem ser elaborados por pesquisadores com experiência no campo em questão ou por especialistas de reconhecido saber. Devem ter até 20 páginas, incluindo resumos, tabelas, quadros, figuras e referências. As tabelas, quadros e figuras limitadas a 06 no conjunto, devem incluir apenas os dados imprescindíveis. As figuras não devem repetir dados já descritos em tabelas. As referências bibliográficas devem ser limitadas a 60. Deve-se evitar a inclusão de número excessivo de referências numa mesma citação.

Devem conter: título em português e inglês, autores e afiliações, resumo e abstract (de 150 a 250 palavras), palavras-chave/keywords, introdução, método (como nos artigos de pesquisas originais) considerações finais (neste item serão retomadas as diferentes colocações dos autores estudados de maneira a conduzir a um fechamento, porém, não havendo conclusões definitivas), agradecimentos (caso necessário), referências.

Ou, em caso de artigos de revisão de literatura contendo metanálise, depois do item método deverá ser apresentado o item resultados (contendo a metanálise) e as conclusões.