01

Requisição POST

Transcrição

Com o `sqlmap`, foi possível visualizar todas as informações dos usuários cadastrados na aplicação **Alura Shows**. Acessaremos a aplicação novamente em **Usuario > Login** e faremos a autenticação com a conta do **Alex**, no campo **E-mail** colocaremos `alex@gmail.com e no campo **Senha** colocaremos 123`.

Como já mencionamos, os parâmetros de e-mail e senha estão sendo passados pela URL. Com isso, colocamos a URL no `sqlmap` para que fosse injetado diversos códigos SQL, possibilitando vulnerabilidades. O ideal é passarmos esses parâmetros para o servidor de forma escondida no corpo da requisição. Devemos trabalhar com outro tipo de requisição, que é o tipo ***POST***.

No Eclipse pararemos o Tomcat, em seguida usaremos o atalho "Ctrl + Shift + R" e no campo de busca aberto colocaremos "usuario.jsp", a página será aberta. Em `usuario.jsp`, mudaremos o `method="get"` para `method="post"`:

```
<form id="login-form" action="${s:mvcUrl('UC#login').build()}"
    method="post" role="form" style="display: block;">

    // ...

</form>
```

Podemos salvar as modificações da página. Agora devemos acessar a classe `UsuarioController` e dizer que o método `login()` só deve ser acessado por meio do ***POST***. Novamente com o atalho "Ctrl + Shift + R", pesquisaremos a classe `UsuarioController`, e deixaremos explícito que o valor `value = "/login"` só será acessado usando `method = ResquestMethod.POST`:

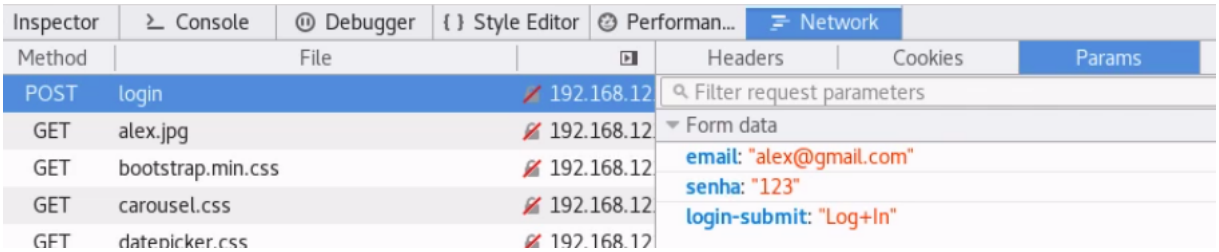
```
@RequestMapping(value = "/login", method = RequestMethod.POST)
public String login(@ModelAttribute("usuario") Usuario usuario,
        RedirectAttributes redirect, Model model, HttpSession session) {

    // ...

}
```

Reiniciaremos o Tomcat e voltaremos para o Linux. Acessaremos a aplicação e faremos o *login* com a conta do Alex. Repare que os parâmetros não estão mais sendo passados pela URL. Mas será que está seguro? No Terminal do Linux, passaremos o comando:

```
sqlmap -u "http://192.168.121.171/alura-shows/login" --dump -T usuario -D owasp
```

Receberemos a mensagem de que os parâmetros testados não podem ser injetados. E faz sentido, não tem parâmetros sendo passado pela URL. Mas não podemos nos iludir e achar que por mudarmos de **GET** para **POST**, a aplicação está segura. Os parâmetros de e-mail e senha continuam sendo enviados, mas no corpo da requisição. Vamos verificar.

Faremos o "Logout" da conta do Alex. De volta na página, faremos o "Login" com a conta do Alex. Após a autenticação, clicaremos com o botão direito do mouse, selecionaremos ***Inspect Element*** e clicaremos na aba ***Network***. Clicaremos na requisição ***POST*** para abrir as informações, em seguida clicaremos na aba "Params". Veremos os parâmetros de e-mail e senha sendo passados no corpo da requisição.



De volta no Terminal do Linux, usaremos o comando `clear` para limpar as informações do terminal. Podemos falar para o `sqlmap` quais são os parâmetros passados na requisição, utilizando o `--data` . O comando completo ficará da seguinte maneira:

```
sqlmap -u "http://192.168.121.171/alura-shows/login" --dump -T usuario -D owasp --data="email=a
```

Executaremos o comando. Ele perguntará se queremos continuar, responderemos "c". Em seguida informará que encontrou um redirecionamento, responderemos "n". Seremos perguntados sobre pular os testes em outros bancos de dados, responderemos "Y". Por fim, seremos perguntados se queremos testes estendidos para o MySQL, responderemos "Y".

Novamente teremos a informação de que o parâmetro `email` é vulnerável e pergunta se queremos testar outros parâmetros, colocaremos "n". Conseguimos acesso a todos os resultados dos usuários cadastrados no banco de dados. O problema está na nossa classe de acesso a dados `UsuarioDaoImpl` . As *querys* ao banco de dados estão concatenadas com códigos Java, além de não ter nenhuma validação.

A nossa próxima etapa, é corrigir a classe `UsuarioDaoImpl` e evitar a injeção de código SQL.