

Considerações de segurança

Os aplicativos da Web geralmente enfrentam todos os tipos de problemas de segurança e é muito difícil acertar tudo. O Flask tenta resolver algumas dessas coisas para você, mas há mais alguns que você tem que cuidar de si mesmo.

Script entre sites (XSS)

Cross site scripting é o conceito de injetar HTML arbitrário (e com JavaScript) no contexto de um site. Para remediar isso, os desenvolvedores tem que escapar corretamente o texto para que ele não possa incluir HTML arbitrário Tags. Para mais informações sobre isso, dê uma olhada no artigo da Wikipedia em [Cross-Site Scripting](#).

O Flask configura Jinja2 para escapar automaticamente de todos os valores, a menos que explicitamente dito o contrário. Isso deve descartar todos os problemas de XSS causados em modelos, mas ainda existem outros lugares onde você tem que estar cuidadoso:



- gerando HTML sem a ajuda de Jinja2
- Recorrer aos dados enviados pelos utilizadores **Markup**
- enviando HTML de arquivos carregados, nunca faça isso, use o cabeçalho para evitar esse problema. **Content-Disposition: attachment**
- enviando arquivos de texto de arquivos carregados. Alguns navegadores estão usando content-type com base nos primeiros bytes para que os usuários pudessem enganar um navegador para executar HTML.

Outra coisa que é muito importante são os atributos sem aspas. Enquanto Jinja2 pode protegê-lo de problemas de XSS escapando HTML, há um coisa da qual não pode protegê-lo: XSS por injeção de atributo. Para contra-atacar possível vetor de ataque, certifique-se de sempre citar seus atributos com aspas duplas ou simples ao usar expressões Jinja nelas:

```
<input value="{{ value }}">
```

Por que isso é necessário? Porque se você não estivesse fazendo isso, um invasor poderia facilmente injetar manipuladores JavaScript personalizados. Por exemplo, um invasor pode injetar este pedaço de HTML+JavaScript:

```
onmouseover=alert(document.cookie)
```

Quando o usuário se move com o mouse sobre a entrada, o cookie seria apresentado ao usuário em uma janela de alerta. Mas em vez de mostrar cookie para o usuário, um bom invasor também pode executar qualquer outro Código JavaScript. Em combinação com injeções:  **en**  **estábulo** até mesmo fazer o elemento preencher a página inteira para que o usuário ~~passa~~ **passa** o mouse em qualquer lugar da página para acionar o ataque.

Há uma classe de problemas de XSS que a fuga de Jinja não protege contra. O atributo da tag pode conter um URI, que o navegador executará quando clicado se não for protegido corretamente. `hrefjavascript:`

```
<a href="{{ value }}">click here</a>
<a href="javascript:alert('unsafe');">click here</a>
```

Para evitar isso, você precisará definir o cabeçalho de resposta da [Política de Segurança de Conteúdo \(CSP\)](#).

Falsificação de solicitação entre sites (CSRF)

Outro grande problema é o CSRF. Este é um tópico muito complexo e não vou Descreva-o aqui em detalhes, apenas mencione o que é e como teoricamente evitá-lo.

Se suas informações de autenticação estiverem armazenadas em cookies, você terá implícito gestão do estado. O estado de "estar conectado" é controlado por um cookie, e esse cookie é enviado com cada solicitação para uma página. Infelizmente, isso inclui solicitações acionadas por sites de terceiros. Se você Não tenha isso em mente, algumas pessoas podem enganar seu usuários do aplicativo com engenharia social para fazer coisas estúpidas sem eles sabendo.











Digamos que você tenha um URL específico que, quando você enviou solicitações para Excluir o perfil de um usuário (digamos). Se um O invasor agora cria uma página que envia uma solicitação POST para essa página com alguns JavaScript eles só têm que enganar alguns usuários para carregar essa página e seus perfis acabarão sendo excluídos. `POSThttp://example.com/user/delete`

Imagine que você administrasse o Facebook com milhões de usuários simultâneos e alguém enviaria links para imagens de gatinhos. Quando os usuários iriam para essa página, seus perfis seriam excluídos enquanto eles são olhando para imagens de gatos fofos.

Como você pode evitar isso? Basicamente para cada solicitação que modifica conteúdo no servidor, você teria que usar um token único e armazená-lo no cookie e também transmiti-lo com os dados do formulário. Depois de receber os dados no servidor novamente, você teria que Compare os dois tokens e certifique-se de que sejam iguais.

Por que o Flask não faz isso por você? O lugar ideal para que isso aconteça é a estrutura de validação de formulário, que não existe no Flask.

Segurança JSON

No Flask 0.10 e inferior, não serializou o nível superior arrays para JSON. Isso ocorreu devido a uma vulnerabilidade de segurança no ECMAScript 4. `jsonify()`   en        

comportamento foi alterado e agora suporta serialização Matrices. `jsonify()`

Cabeçalhos de segurança

Os navegadores reconhecem vários cabeçalhos de resposta para controlar a segurança. Nós recomendamos revisar cada um dos cabeçalhos abaixo para uso em seu aplicativo. A extensão [Flask-Talisman](#) pode ser usada para gerenciar HTTPS e a segurança cabeçalhos para você.

Segurança de transporte estrita HTTP (HSTS)

Informa ao navegador para converter todas as solicitações HTTP em HTTPS, evitando ataques man-in-the-middle (MITM).

```
response.headers['Strict-Transport-Security'] = 'max-age=31536000; includeSubDomains'
```

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

Política de segurança de conteúdo (CSP)

Informe ao navegador de onde ele pode carregar vários tipos de recursos. Este cabeçalho deve ser usado sempre que possível, mas requer algum trabalho para definir o correto política para o seu site. Uma política muito rígida seria:

```
response.headers['Content-Security-Policy'] = "default-src 'self'"
```

- <https://csp.withgoogle.com/docs/index.html>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>

Opções de tipo de conteúdo X

Força o navegador a respeitar o tipo de conteúdo de resposta em vez de tentar detectá-lo, que pode ser abusado para gerar um script entre sites (XSS) atacar.

```
response.headers['X-Content-Type-Options'] = 'nosniff'
```

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>

Opções de quadro X

Impede que sites externos incorporem seu site em um `<iframe>`. Este impede uma `<script>` em um `<iframe>` de executar código. Isso também impede que os cliques no quadro externo podem ser convertidos invisível aos cliques nos elementos da sua

página. Isso também é conhecido como "clickjacking".`iframe`

```
response.headers['X-Frame-Options'] = 'SAMEORIGIN'
```

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

Opções de Definir cookies

Essas opções podem ser adicionadas a um cabeçalho para melhorar sua segurança. O Flask tem opções de configuração para defini-las no cookie de sessão. Eles também podem ser definidos em outros cookies.`Set-Cookie`

- `Secure` limita os cookies apenas ao tráfego HTTPS.
- `HttpOnly` protege o conteúdo dos cookies de ser lido com JavaScript.
- `SameSite` restringe como os cookies são enviados com solicitações de sites externos. Pode ser definido como (recomendado) ou . impede o envio de cookies com solicitações propensas a CSRF de sites externos, como o envio de um formulário. impede o envio cookies com todas as solicitações externas, incluindo links regulares. `'Lax'` `'Strict'` `LaxStrict`

```
app.config.update(
    SESSION_COOKIE_SECURE=True,
    SESSION_COOKIE_HTTPONLY=True,
    SESSION_COOKIE_SAMESITE='Lax',
)
```

```
response.set_cookie('username', 'flask', secure=True, httponly=True, samesite=
```

Especificando ou opções, removerá o cookie após a hora dada, ou a hora atual mais a idade, respectivamente. Se nenhum dos dois estiver definida, o cookie será removido quando o navegador for fechado.`ExpiresMax-Age`

```
# cookie expires after 10 minutes
response.set_cookie('snakes', '3', max_age=600)
```

Para o cookie de sessão, `if` é definido, então é usado para definir a expiração. A implementação de cookie padrão do Flask valida que o cookie criptográfico `signature` não é mais antiga do que esse valor. Reduzir esse valor pode ajudar a mitigar ataques de repetição, onde os cookies interceptados podem ser enviados posteriormente.

```
app.config.update(
    PERMANENT_SESSION_LIFETIME=600
)
```

```
@app.route('/login', methods=['POST'])
def login():
```

  en  **estábulo**

