

## Plano de Segurança - conversaFiada

Alunos: Rodrigo Almeida, Rodrigo Teixeira, Eduardo Meneses

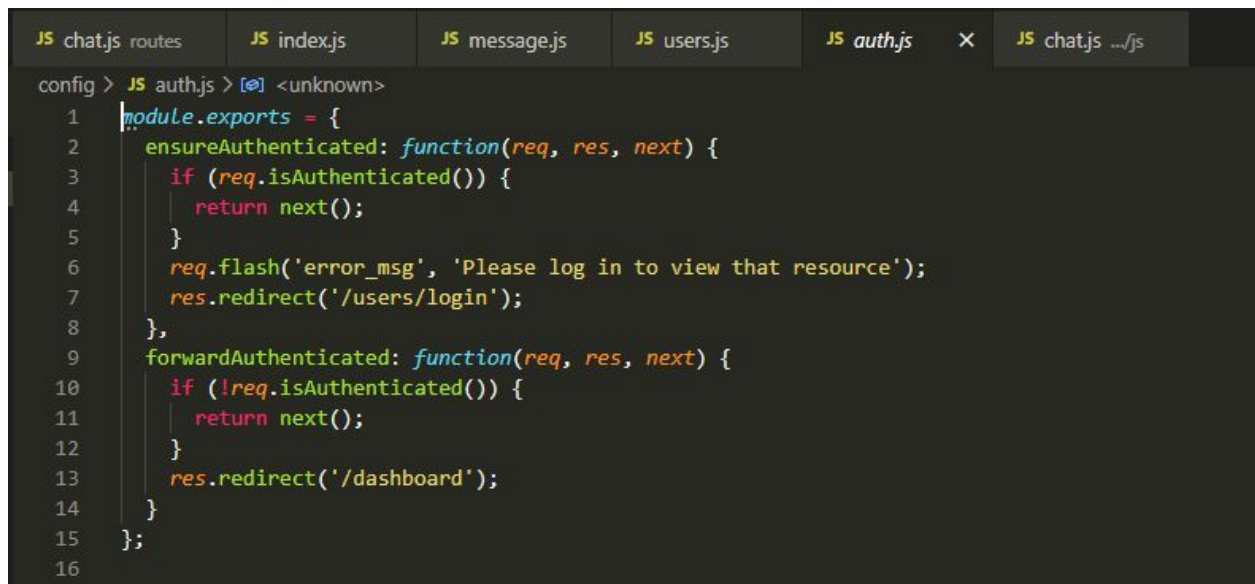
### 1) VULNERABILIDADES

- **Acesso a funcionalidades do site sem autenticação**

Para que os usuários acessem certas áreas do site e realizem certas ações é necessário que eles possuam uma sessão ativa. Caso contrário, poderiam ocorrer eventos não esperados como:

- Usuário não autenticado cria salas de chat
- Usuário não autenticado entra no Dashboard
- Usuário não autenticado utiliza APIs que dependem da sessão, de modo que a chamada delas sem uma sessão ativa causa um erro e/ou queda no servidor
- Usuário não autenticado envia e lê mensagens

Para evitar este tipo de anomalia, podemos utilizar um middleware no Express para verificar se as requisições enviadas estão atreladas a uma sessão ativa:



```
JS chat.js routes JS index.js JS message.js JS users.js JS auth.js X JS chat.js .../js
config > JS auth.js > [?] <unknown>
1  module.exports = {
2    ensureAuthenticated: function(req, res, next) {
3      if (req.isAuthenticated()) {
4        return next();
5      }
6      req.flash('error_msg', 'Please log in to view that resource');
7      res.redirect('/users/login');
8    },
9    forwardAuthenticated: function(req, res, next) {
10     if (!req.isAuthenticated()) {
11       return next();
12     }
13     res.redirect('/dashboard');
14   }
15 };
16
```

Um exemplo deste código sendo utilizado quando é realizado um GET para o Dashboard:

```
16 // Dashboard
17 router.get("/dashboard", ensureAuthenticated, (req, res) => {
```

- **Requisições Cross-Origin**

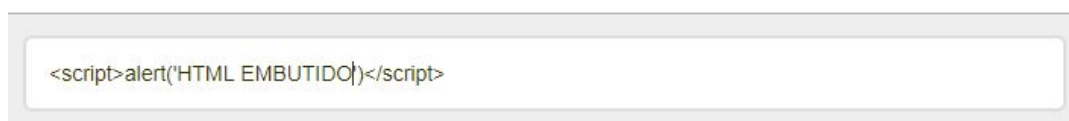
Através das requisições Cross-Origin, é possível que ataques como os de CSRF sejam realizados. Desse modo, não é possível realizar requisições Cross-Origin para a aplicação. Vamos tentar, por exemplo, realizar uma requisição para a aplicação de outro site:

```
> var x = new XMLHttpRequest()
x.open("GET", "http://localhost:5000/teste")
x.send()
< undefined
✖ Access to XMLHttpRequest at 'http://localhost:5000/teste' from origin 'https://sucuri.(index):1 net' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
✖ ▶ GET http://localhost:5000/teste net::ERR_FAILED VM691:3
```

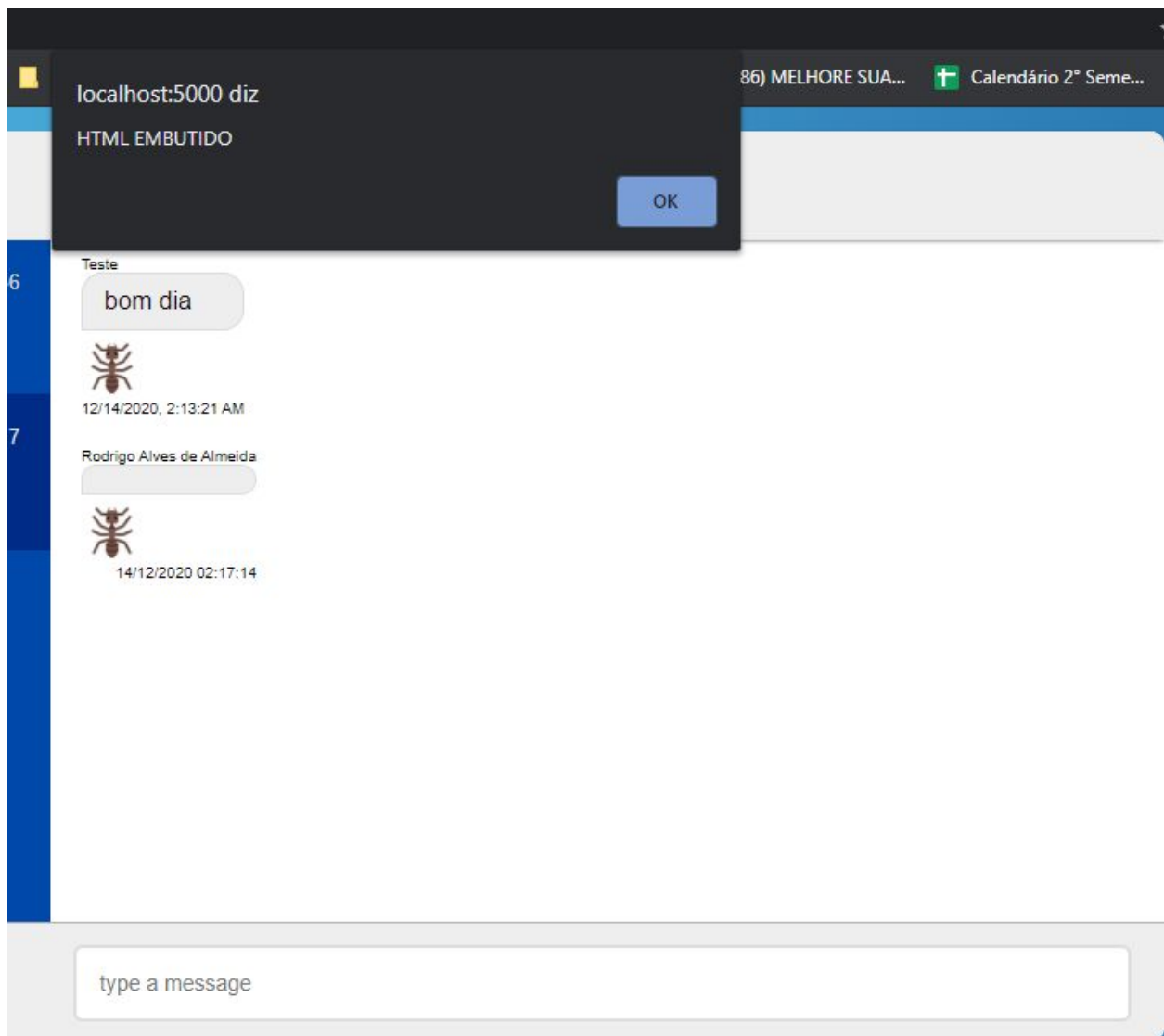
- **Injeção de HTML pelas mensagens**

Como as mensagens enviadas são renderizadas como HTML, é possível que um usuário envie uma mensagem com HTML embutido para que esse código seja executado no client-side do outro usuário.

Vamos tomar um exemplo de script:



Ao enviar essa mensagem, o script é executado na tela do outro usuário:



Para se proteger deste tipo de vulnerabilidade, foi utilizado no back-end um módulo chamado *express-sanitizer*, e toda mensagem enviada é antes passada por revisão:

```
const msg = req.sanitize(req.body.message);
```

Agora, as mensagens são filtradas antes de enviadas:

## Teste

teste

tudo bem?



14/12/2020 02:44:12

sim e vc?

14/12/2020 02:44:18

Teste

teste1



14/12/2020 02:44:27

Teste

teste2



14/12/2020 02:44:29

<script>alert('HTML EMBUTIDO')</script> Ola

## Rodrigo Alves de Almeida

14/12/2020 02:44:12

Teste

sim e vc?



14/12/2020 02:44:18

teste1

14/12/2020 02:44:27

teste2

14/12/2020 02:44:29

Teste

Ola



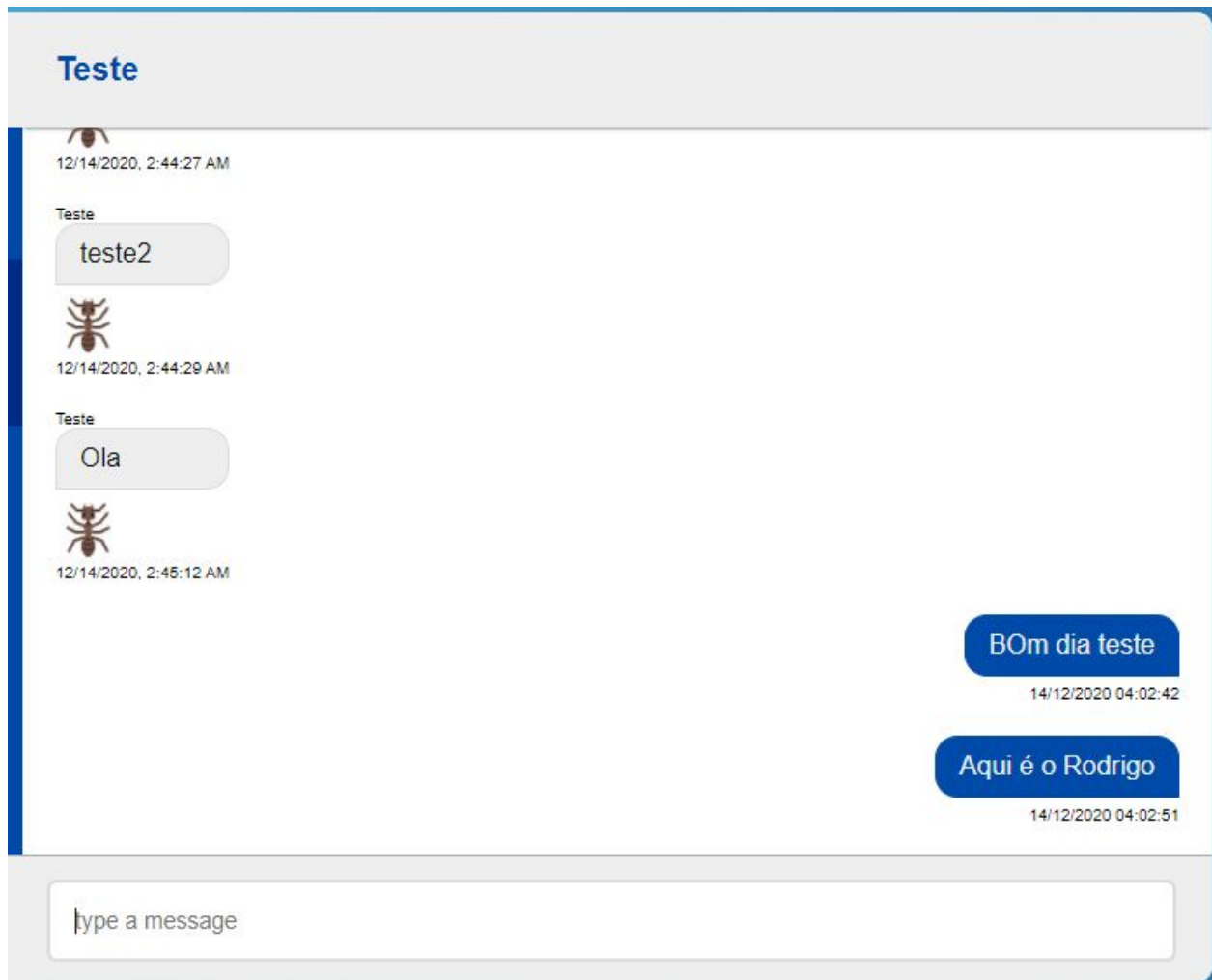
14/12/2020 02:45:12

type a message


- **Obtendo dados que só estão disponíveis a outros usuários**

Outra vulnerabilidade que pode ser explorada no sistema é que os usuários tenham acesso a informações que não deveriam.

Por exemplo, vamos supor que existe um chat particular entre os usuários *Teste* e *Rodrigo*:



Agora, vamos supor que um usuário terceiro *Teste2* descubra o id desse chat particular entre os dois usuários. Pelo seguinte link ele pode acessar as mensagens do site:

 [localhost:5000/message/get?val=5fd6efee943c605ba9e2500d](http://localhost:5000/message/get?val=5fd6efee943c605ba9e2500d)

```
[{"author": "Teste", "content": "bom dia", "date": "12/14/2020, 2:13:21 AM", "icon": 1, "is_self": false}, {"author": "Rodrigo Alves de Almeida", "content": "<script>alert('HTML EMBUTIDO')</script>", "date": "12/14/2020, 2:17:14 AM", "icon": 1, "is_self": false}, {"author": "Rodrigo Alves de Almeida", "content": "<pre><script>alert('a')</script></pre>", "date": "12/14/2020, 2:20:27 AM", "icon": 1, "is_self": false}, {"author": "Rodrigo Alves de Almeida", "content": "<pre>aa</pre>", "date": "12/14/2020, 2:24:12 AM", "icon": 1, "is_self": false}, {"author": "Rodrigo Alves de Almeida", "content": "<div>asokdpaksd</div>", "date": "12/14/2020, 2:24:44 AM", "icon": 1, "is_self": false}, {"author": "Rodrigo Alves de Almeida", "content": "<iframe>asdkoads</iframe>", "date": "12/14/2020, 2:25:06 AM", "icon": 1, "is_self": false}, {"author": "Teste", "content": "salve", "date": "12/14/2020, 2:43:58 AM", "icon": 1, "is_self": false}, {"author": "Teste", "content": "tudo bem?", "date": "12/14/2020, 2:44:12 AM", "icon": 1, "is_self": false}, {"author": "Teste", "content": "sim e vc?", "date": "12/14/2020, 2:44:18 AM", "icon": 1, "is_self": false}, {"author": "Teste", "content": "teste1", "date": "12/14/2020, 2:44:27 AM", "icon": 1, "is_self": false}, {"author": "Teste", "content": "teste2", "date": "12/14/2020, 2:44:29 AM", "icon": 1, "is_self": false}, {"author": "Teste", "content": "Ola", "date": "12/14/2020, 2:45:12 AM", "icon": 1, "is_self": false}, {"author": "Rodrigo Alves de Almeida", "content": "Bom dia teste", "date": "12/14/2020, 4:02:42 AM", "icon": 1, "is_self": false}, {"author": "Rodrigo Alves de Almeida", "content": "Aqui Ã© o Rodrigo", "date": "12/14/2020, 4:02:51 AM", "icon": 1, "is_self": false}]
```

Ele pode, inclusive, invadir essa conversa particular e enviar uma mensagem:

```
> var data = {
  chat_id: '5fd6efee943c605ba9e2500d',
  message: 'Estou logado na Teste2 e enviando esta mensagem para o chat entre o Rodrigo
e o Teste 1'
}

$.post('/message/send', data)
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMime
Type: f, ...}
>
```

De modo que a mensagem realmente aparece no chat da conversa particular entre o *Rodrigo* e o *Teste* (o print a seguir foi tirado logado na *Teste*):



Para se proteger desta vulnerabilidade, é necessário realizar checagens em todas as requisições:

```
31 | router.get("/get", ensureAuthenticated, (req, res) => {
32 |     var chat_id = req.url.substring(req.url.indexOf('=') + 1)
33 |
34 |     Chat.findById(chat_id).then(chat => {
35 |         //chat nao existe
36 |         if (!chat) return res.end()
37 |         //chat privado
38 |         if (chat.is_pvt){
39 |             //se o chat é privado, deve conter o email de quem faz a requisição
40 |             if (!(chat.pvt1 === req.user.email || chat.pvt2 === req.user.email))
41 |                 return res.end()
42 |             Messages.find({ chat_id: chat_id }).sort({date: "asc"})
43 |                 .populate('user_id')
44 |                 .then((messages) => {
```

Desse modo, toda vez que um usuário realizar alguma operação sobre algum chat, é verificado se esse usuário realmente participa do chat.

## 2) AUTENTICAÇÃO

Foram tomados dois modos de realizar autenticação para o programa:

- **Autenticação local**

O usuário cadastra seu email e seleciona uma senha, a qual é criptografada e guardada no banco de dados.

- **OAuth Google**

Aqui o usuário utiliza sua conta Google para realizar a autenticação. Foi criado um aplicativo pelo Google que realiza a autenticação e geração de um token, o qual gera a sessão do usuário. Vale notar que não é possível possuir uma conta local e uma Google com o mesmo email.