

Autores:

Gonçalo Maranhão

Filipe Posio

Pedro Carneiro

Inês Águia

De forma a facilitar a leitura deste documento queremos apresentar uma descrição de como a app está implementada em termos de estrutura. Usamos módulos ES6 JavaScript para os membros não terem que mexer demasiado nos mesmo ficheiros.

No root:

Ficheiro json e script sql → dizem respeito à base de dados sql.

No dir app:

index.html → responsável por apresentar os links para as páginas na app. As duas divs(htmlForAll e JSForAll) presentes vão buscar o seu conteúdo ao index.js que por sua vez chama o conteúdo dos módulos de JavaScript(app/js/pages).

server.js -> responsável pelo funcionamento do servidor, conta com a lógica de sql apropriada, e com a lógica relativa ao upload de ficheiros, que é usado para uma das CWE).

db.js → diz respeito à base de dados.

No dir js(app/js):

index.js → responsável pelo routing e por chamar as funções dos módulos.

No dir pages(app/js/pages):

AbstractPage.js → classe abstracta de onde todos os módulos(que são os restantes ficheiros) herdam as funções que têm de implementar.

Restantes ficheiros.js → módulos ES6 Javascript, responsáveis pelo hml e JavaScript(apropriado) para o funcionamento e aspeto de cada página em particular.

A versão segura em (/app_sec) é em tudo igual, exceto a adição do ficheiro fileCheck.js, que faz a verificação para garantir que o upload de ficheiros é uma imagem(relevante para uma CWE, explicada em mais detalhe abaixo).

CWE-79 : Improper Neutralization of Input During Web Page Generation(‘Cross-site Scripting’)

Na página “My List”(módulo app/js/pages/BookList.js) encontrasse uma caixa de texto para que o utilizador possa fazer uma lista sua de livros em que esteja interessado, esta lista é persistente, ou seja, até o utilizador carregar no botão delete os itens ficam na página. Na perspectiva de um utilizador normal, esta página é útil para ter um sítio para apontar os livros em que tem interesse e remover os que já não têm interesse, ficando assim livre de usar o resto da app sendo sempre possível voltar a esta lista.

Na versão insegura da app esta caixa de texto é vulnerável a injeção de código html. O aspeto normal seria:

- In Search of Lost Time
- One Hundred Years of Solitude
- Hunger
- The Stranger
- Lord of the Flies

Tendo em conta a vulnerabilidade desta caixa, o utilizador poderia , por exemplo, escrever

`ShortcutForSIO`

e teria criado um link para outra página, o que, obviamente, não é o pretendido quando a funcionalidade desta caixa foi implementada.

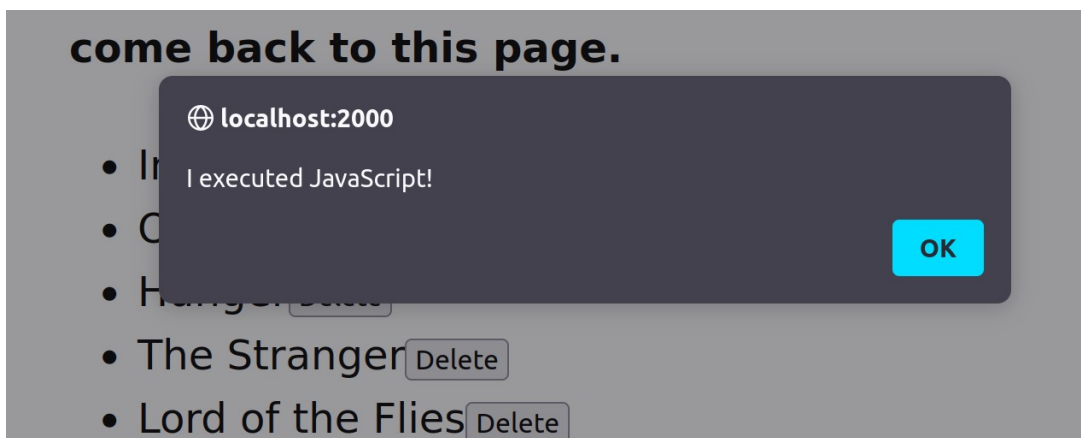
- In Search of Lost Time
- One Hundred Years of Solitude
- Hunger
- The Stranger
- Lord of the Flies
- [ShortcutForSIO](http://sweet.ua.pt/jpbarraca)

É possível injectar pedaços de código mais maliciosos, visto que é possível usar javascript em html, por exemplo, se for introduzido `JavaScript working!`, a página fica com este aspeto.

- In Search of Lost Time
- One Hundred Years of Solitude
- Hunger
- The Stranger
- Lord of the Flies
- [ShortcutForSIO](#)
- [JavaScript working!](#)

Devido à implementação, a tag `<script>` não funciona(por causa da propriedade `innerHTML`, explicada mais abaixo).

Ao carregar no botão criado:



Sendo assim, o utilizador pode injectar nesta página código html e javascript, manipulando a página como queira.

Certamente que, tendo em conta que esta página seria apenas para uso pessoal de cada utilizador neste projeto, não seria o melhor sítio para um atacante, por exemplo, meter um link de forma a redirecionar outros utilizadores, mas é possível.

Esta vulnerabilidade é CWE-79 Type 2: Stored XSS, visto que a app guarda na página o código injectado. Devido à grande possibilidade de manipulação da parte do utilizador esta vulnerabilidade é grave.

Esta injeção de código é possível devido à forma como o suposto livro está a ser

inserido. A string está a ser lida como html porque a implementação usa a propriedade `innerHTML` para ler a string. Isto acontece no módulo `bookList.js` na função `addBook`.

```
function addBook(bookText) {  
    books.push(bookText)  
    var li = document.createElement('li')  
    li.innerHTML = bookText
```

Esta é uma forma comum de se implementar uma lista com javascript. Para defender contra esta vulnerabilidade, apenas se tem de fazer com que a string não seja lida como html, o que, em javascript é possível usando a propriedade `innerText` que se encontra no mesmo módulo, na mesma função, mas na versão segura da app.

```
function addBook(bookText) {  
    books.push(bookText)  
    var li = document.createElement('li')  
    li.innerText = bookText
```

Assim, se for introduzido os dois pedaços de código acima descritos na caixa, obtemos este aspeto:

- In Search of Lost Time
- One Hundred Years of Solitude
- Hunger
- The Stranger
- Lord of the Flies
- [ShortcutForSIO](http://sweet.ua.pt/jpbarraca)
- JavaScript working!

Já não é possível injectar código nesta caixa de texto.

CWE-434: Unrestricted Upload of File with Dangerous Type

Na mesma página encontrasse a possibilidade de se fazer upload de imagens, de forma a que a lista do utilizador não seja apenas texto, acrescentando assim um aspeto visualmente mais apelativo.

Upload your image

No files selected.

Na versão insegura da app, este upload apresenta enorme risco devido à sua implementação. Os quatro aspetos cruciais são o facto de ser possível fazer upload de qualquer ficheiro, não apenas de imagens, o ficheiro está a ser guardado num dir que conta com ficheiros relevantes para o funcionamento da app(/app), o ficheiro preserva o nome original e os ficheiros com o mesmo nome são sobrescritos.

De forma a minimizar código foi usado o middleware multer do npm. Em html, o formulário encontrasse em bookList.js na função getHtml().

```
<form action="/upload" method="post" enctype="multipart/form-data">
<input type="file" name="something" multiple/>
<button>Submit</button>
```

O javascript responsável pelo back-end do upload encontrasse em server.js.

```
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, '../app');
  },
  filename: (req, file, cb) => {
    const { originalname } = file;
    cb(null, originalname);
  }
});

const upload = multer({ storage: storage });

app.post('/upload', upload.single('something'), (req, res) => {
  return (res.json({ status: 'File uploaded, please go the previous page' }));
});

app.get("/testImage.png", (req, res) => {
  res.sendFile("/testImage.jpeg");
});
```

O fato de os ficheiros com o mesmo nome serem sobreescritos é default do multer. Este dir conta com os ficheiros index.html, server.js, db.js e fileCheck.js (este último apenas presente na versão segura).

Sendo que ficheiros enviados pelo utilizador sobrescrevem ficheiros com o mesmo nome, se o utilizador fizer upload de um ficheiro com o nome index.html ou server.js pode alterar todas as funcionalidades da app. O melhor cenário seria o de o utilizador fazer upload de um ficheiro (com um desses nomes) em branco (sem conteúdo), sendo que isto faz com que a app não apresente absolutamente nada (ou que o servidor vá abaixo, depende do ficheiro), visto que esse ficheiro iria substituir o ficheiro com o mesmo nome presente no dir.

Sendo o melhor cenário o de toda a app deixar de funcionar, é difícil pensar numa vulnerabilidade que apresente maior risco para a fiabilidade da app. Não sabemos até que ponto o atacante teria facilidade em descobrir que existe um ficheiro chamado index.html ou server.js, mas para além de serem nomes bastante convencionais, se o atacante vir o código fonte do index.html (que seria apenas carregar com o botão direito do rato e “View Page Source” sem necessidade de softwares esquisitos) consegue ver que existe uma pasta chamada index.js (noutro dir), não sendo assim difícil adivinhar que poderá existir um ficheiro chamado index.html. No caso do index.html, existindo esta capacidade de ver o código fonte, um atacante pode fazer o que queira com a app, visto que pode escrever html, javascript, css etc.

A versão segura passa por mitigar os aspetos cruciais descritos em cima. Nesta versão o upload apenas aceita imagens, esta verificação é feita no fileCheck.js.

```
const imageFilter = function(req, file, cb) {
  if (!file.originalname.match(/\.(jpg|JPG|jpeg|JPEG|png|PNG|gif|GIF)$/)) {
    req.fileValidationError = 'Only image files are allowed!';
    return cb(new Error('Only image files are allowed!'), false);
  }
  cb(null, true);
};
exports.imageFilter = imageFilter;
```

Adicionalmente, as imagens vão para uma pasta chamada upload que, embora esteja no dir com os ficheiros acima descritos, é outro dir (/app_sec/upload). Todas as imagens têm um nome único e, portanto, não há sobreposição. Para esse efeito é usado o código em baixo.

```

const storage = multer.diskStorage({
  destination: function(req, file, cb) {
    cb(null, 'upload/');
  },

  filename: function(req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname));
  }
});

app.post('/upload', (req, res) => {
  let upload = multer({ storage: storage, fileFilter:
    fileCheck.imageFilter }).single('profile_pic');

  upload(req, res, function(err) {
    if (req.fileValidationError) {
      return res.send(req.fileValidationError);
    }
    else if (!req.file) {
      return res.send('Please select an image to upload');
    }
    else if (err instanceof multer.MulterError) {
      return res.send(err);
    }
    else if (err) {
      return res.send(err);
    }

    res.send(`Upload successful<p><a href="http://localhost:2000/BookList">
      Click here to go the previous page</a></p>`);
  });
});

app.use(express.static(__dirname + '/upload'));

```

Na versão insegura, ao tentar fazer upload de um ficheiro que não é imagem.

```
status: "File uploaded, please go the previous page"
```

Na versão segura:

Only image files are allowed!


O upload destes ficheiros, quando é feito com sucesso, pode ser verificado nas respetivas pastas, versão insegura(/app) e versão insegura(/app_sec/upload).

Já não é possível que o ficheiro(imagem, na versão segura) do qual se fez upload tenha impacto negativo na app.

Infelizmente não conseguimos fazer o back-end de forma a que a imagem, da qual o utilizador fez upload, seja mostrada ao utilizador. Assim, para não descartar por completo esta vulnerabilidade, que existe mesmo sem este back-end, decidimos apresentar outra caixa, caso depois de fazer o upload a imagem não seja mostrada, o que acontece sempre. Esta segunda caixa foi feita tendo em conta a perspetiva do utilizador oferecendo duas possibilidades de na, perspetiva de um utilizador normal, fazer a mesma coisa. Num cenário em que a imagem fosse mostrada após o upload não haveria necessidade desta funcionalidade, visto que não seria relevante nem de

um ponto de vista de segurança nem de um ponto de vista de front-end para utilizador normal. O código para esta funcionalidade encontrasse em BookList.js(nas duas versões) com comentário a separar esta funcionalidade das restantes. Para manter consistência, esta funcionalidade respeita a diferença da versão segura e insegura de só aceitar imagens ou de aceitar qualquer ficheiro e a imagem é também persistente.

If your upload doesn't show your image, click below

No file selected. 

CWE-89 : Improper Neutralization of Special Elements used in SQL Command (‘SQL Injection’)

Files:

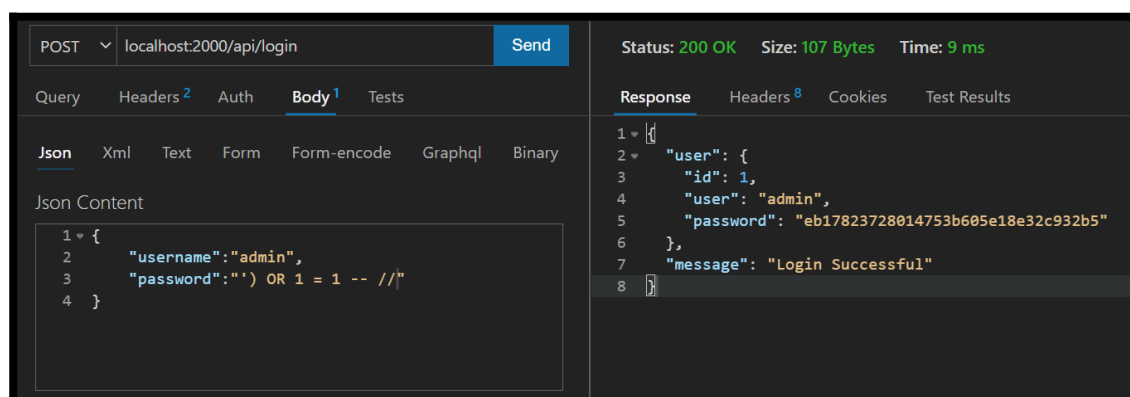
Db.js – método login()

Server.js – POST endpoint api/login

Pages/SqlInjection.js (Menu Login)

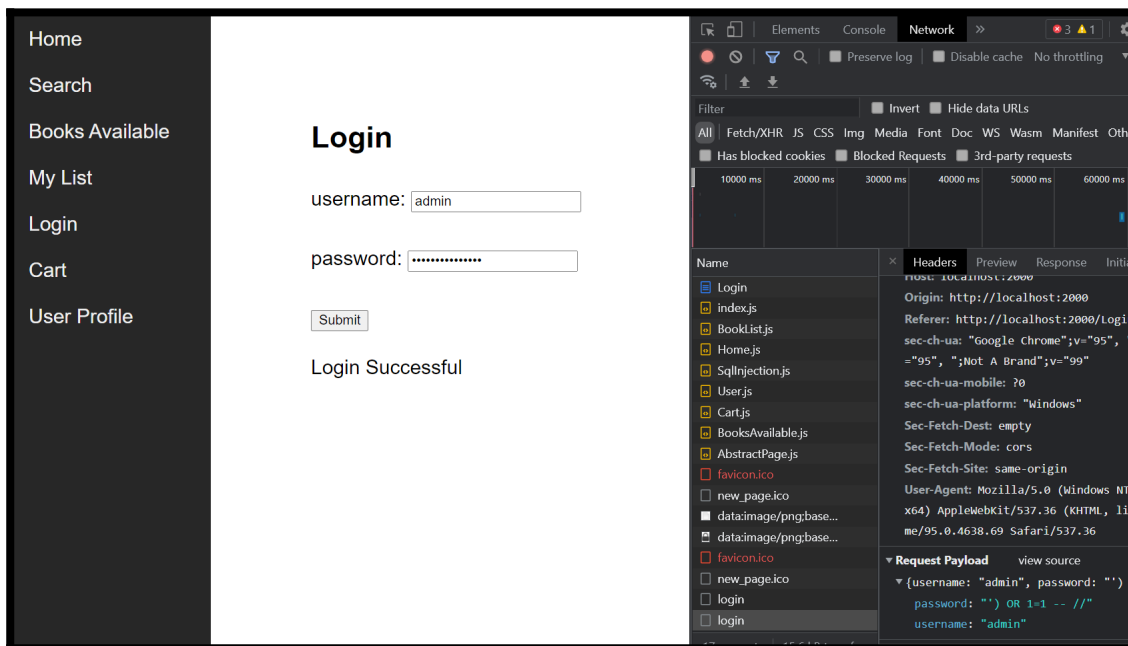
Na página “Login” consta duas caixas de input, uma para username e outra para password.

Na versão insegura, não é feita qualquer validação sobre os dados que são inseridos em ambas as caixas, sendo possível injetar diretamente SQL, tornando possível efetuar login, consoante imagem abaixo



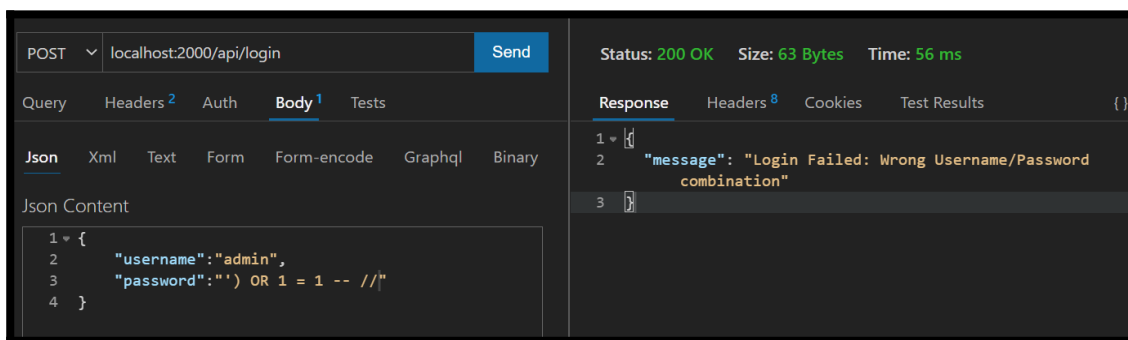
Pedido Login – Not Safe cwe89

*SELECT * FROM User WHERE user = 'admin' AND PASSWORD = MD5('') OR 1 = 1 -- //')*



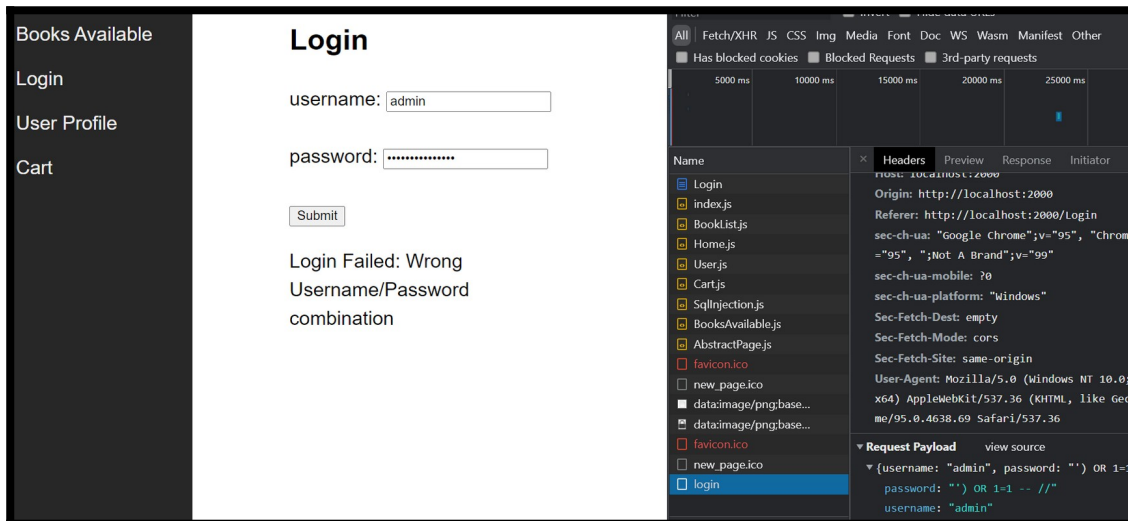
Página Login – Not Safe cwe89

De forma a tornarmos esta funcionalidade segura, optámos por usar placeholders na construção da query que valida o username e password.



Pedido Login – Safe cwe89

SELECT `user` FROM `User` WHERE user = 'admin' AND password = MD5('') OR 1 = 1 -- //')



4. Página Login – Safe cwe89

Files:

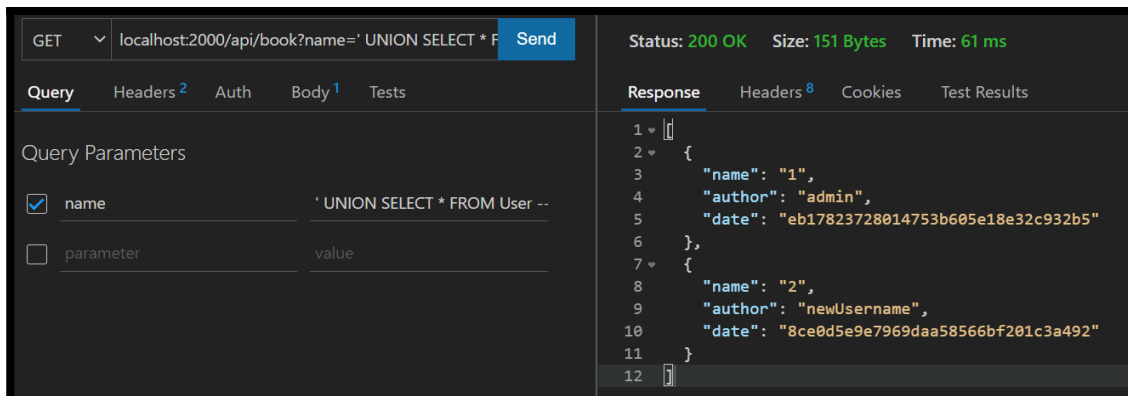
Db.js – método getBookByName()

Server.js – GET endpoint api/book

Pages/BooksAvailable.js (Menu Books Available)

Agora, na página Books Available, é possível fazer uma pesquisa dos livros pelo nome, no entanto na app insegura, é possível visualizar dados de outra tabela

carregando os dados na página.

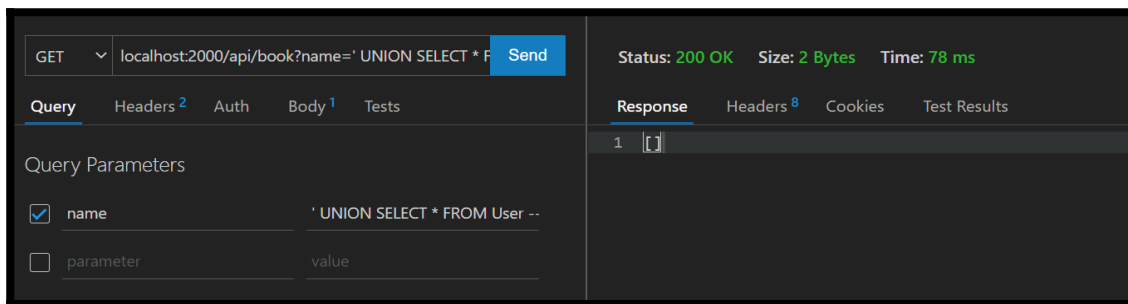


Pedido BooksName – Not Safe cwe89

*SELECT name,author,date FROM Book WHERE name LIKE " UNION SELECT * FROM User -- //%"*

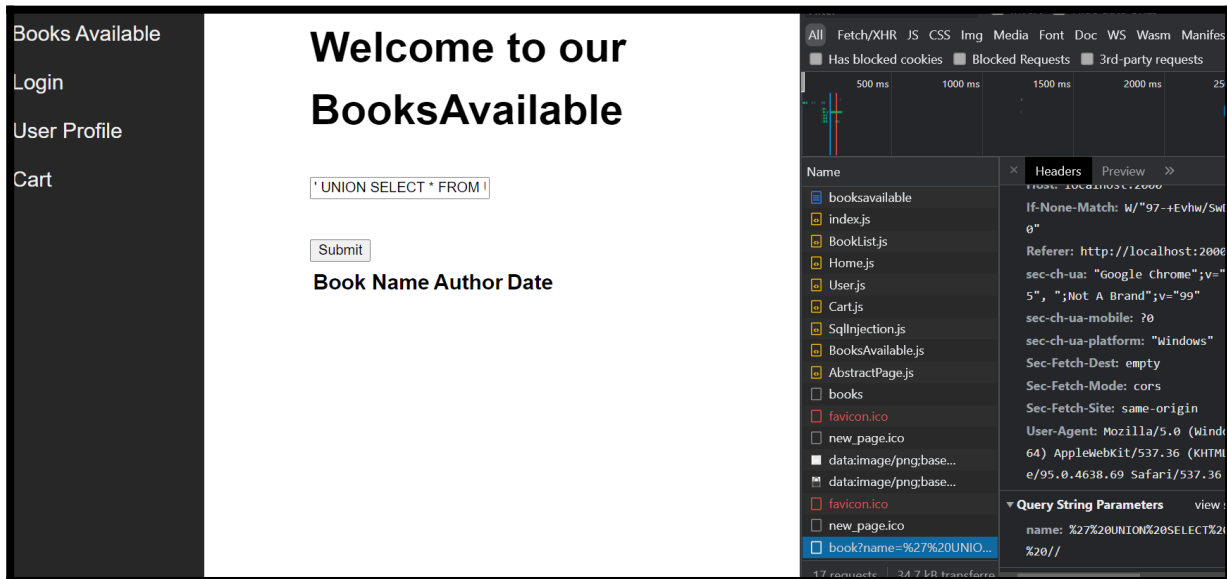
Página BooksName – Not Safe cwe89

Já na app segura, com os placeholders na construção da query conseguimos controlar o output para não aparecer dados indesejados.



Pedido BooksName – Safe cwe89

*SELECT * FROM `Book` WHERE `name` LIKE '\ ' UNION SELECT * FROM User --
//%'*



Página BooksName – Safe cwe89

CWE-200 : Exposure of Sensitive Information to an Unauthorized Actor

Files:

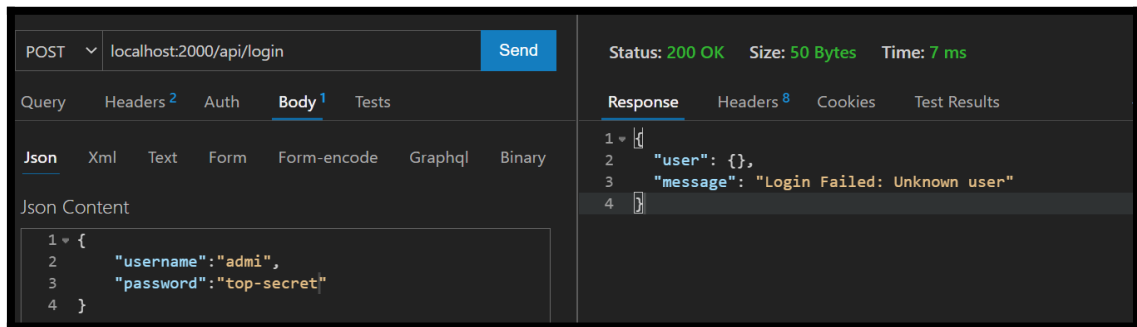
Db.js – método login()

Server.js – POST endpoint api/login

Pages/SqlInjection.js (Menu Login)

Na app insegura na funcionalidade login, quando é feito o pedido, se inserirmos o username errado ou a password errada, é confirmado ao utilizador qual dos campos é que está errado. Assim, um atacante pode tentar através de brute-force misturado com alguma experiência e saber, acertar num dos campos, sendo que seria mais provável

no campo do username , visto que admin ou root são bastante convencionais de se usar.



Pedido Login – Not safe cwe200

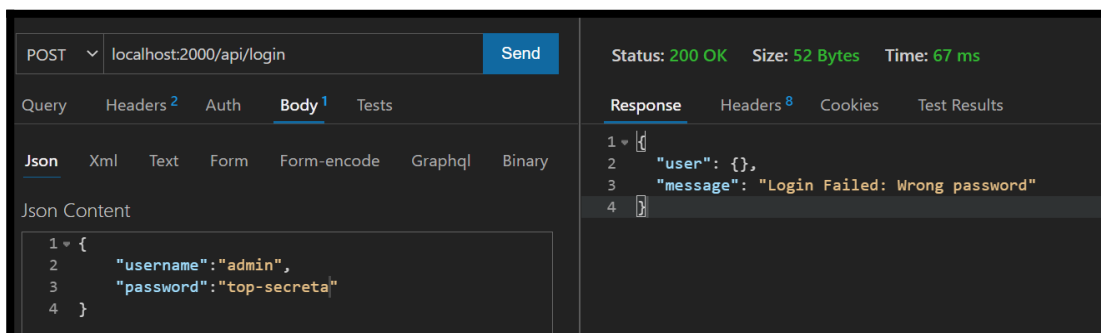
Login

username:

password:

Login Failed: Unknown user

página Login – Not safe cwe200



Pedido Login – Not safe cwe200

Login

username:

password:

Submit

Login Failed: Wrong password

Página Login – Not safe cwe20

Já na app segura, a mensagem ou é de sucesso ou de insucesso, não confirmando qual dos campos está incorreto

POST localhost:2000/api/login Send

Query Headers 2 Auth Body 1 Tests

Json Xml Text Form Form-encode GraphQL Binary

Json Content

```
1 {
2   "username": "admin",
3   "password": "top-secreta"
4 }
```

Status: 200 OK Size: 63 Bytes Time: 6 ms

Response Headers 8 Cookies Test Results {}

```
1 {
2   "message": "Login Failed: Wrong Username/Password combination"
3 }
```

Pedido Login – safe cwe200

My List

Books Available

Login

User Profile

Cart

Login

username:

password:

Submit

Login Failed: Wrong Username/Password combination

Página Login – safe cwe200

CWE-215 : Sensitive information into debugging

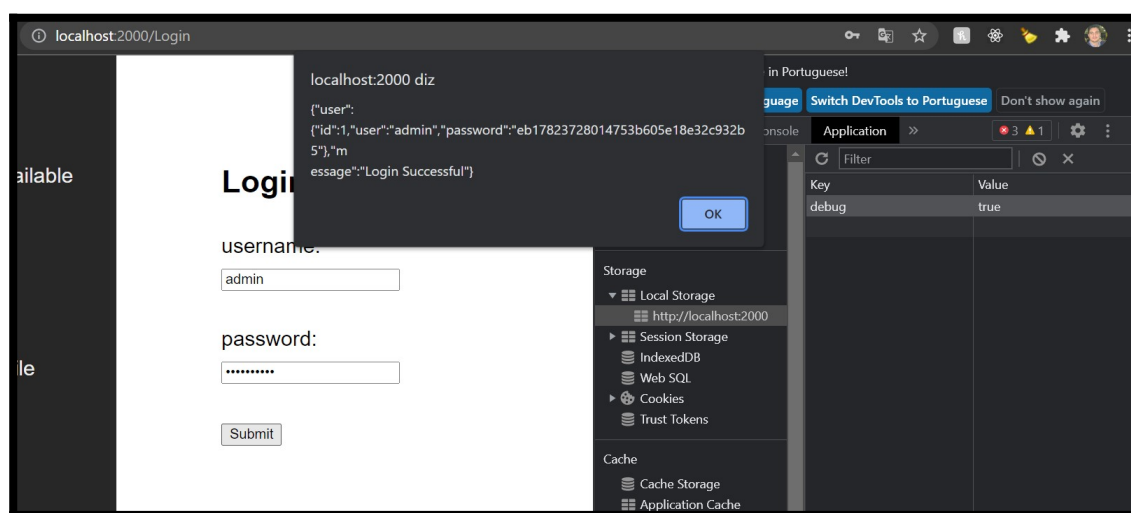
Files:

Db.js – método login()

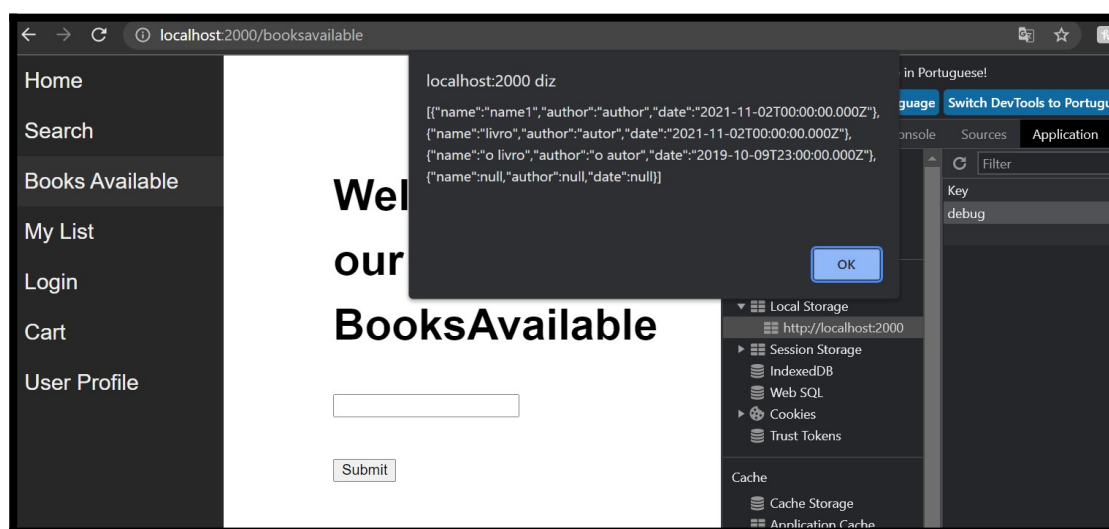
Server.js – POST endpoint api/login

Pages/SqlInjection.js (Menu Login)

Para esta vulnerabilidade, é usada uma variável no local storage para depuração, que é acessível em todas as páginas podendo então executar código específico o que pode mostrar informações que não eram supostas. Como podemos ver numa das imagens abaixo, é exposta a hash da password.



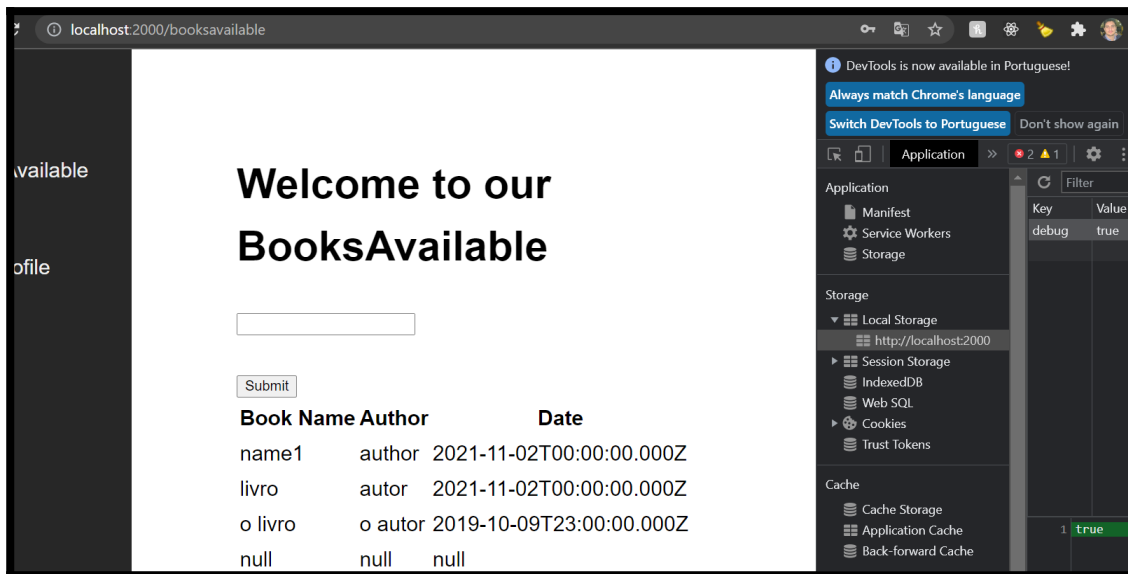
Página Login – not safe cwe215



Página Books Available – not safe cwe215

Já na aplicação segura, foi removido o código extra para quando esta opção está ativa. Talvez seja de espantar que a solução seja esta, mas em <https://cwe.mitre.org/data/definitions/215.html> em potential mitigations, temos que “Do not leave debug statements that could be executed in the source code. Ensure that all debug information is eradicated before releasing the software.” De realçar que um utilizador normal não teria conhecimento para ir seguir os passos que dessem

acesso a esta informação. Também de realçar que existem maneiras de inverter a hash(forá desta app).Página Login – safe cwe215

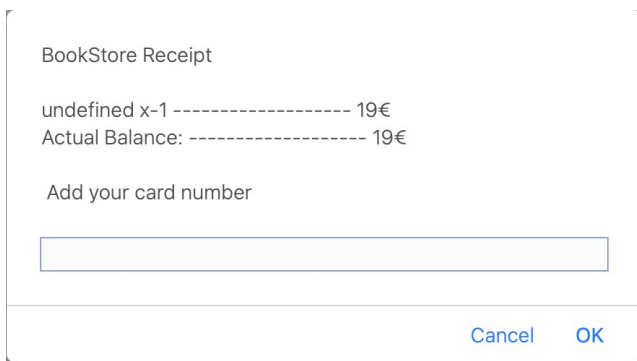


Página Books Available – safe cwe215

CWE-20: Improper Input Validation

Esta vulnerabilidade consiste na receção de dados que, ou não são validados ou são incorretamente validados assumindo que os dados são corretos e respeitam os parâmetros.

CWE-20: Insecure version



Aplicado ao nosso caso, pode ser considerada grave, dado que surge no âmbito de uma compra onde o total está a ser calculado. Como seria de esperar o utilizador não tem qualquer tipo de acesso à variável do preço, ainda assim não é acautelado que a quantidade de um artigo possa ser erradamente assumida como negativa.

Assim sendo, deixa a porta aberta para que um atacante possa adulterar esse fator, tendo como consequência o utilizador ser erradamente creditado valor quando na realidade deveria ter sido cobrado.

CWE-20: Secure version

Shoping Basket

Book Name	Author	Price	Quantity	
A Gentleman in Moscow	Amor Tobles	19€	<input type="text" value="-20"/>	Proceed to Checkout

Como tal, a solução aplicada ao contexto da vulnerabilidade passou por fazer uma validação dos valores recebidos do utilizador forçando uma quantidade positiva.

Shoping Basket

Book Name	Author	Price	Quantity	
A Gentleman in Moscow	Amor Tobles	19€	<input type="text" value="-1"/>	Proceed to Checkout

Please enter a positive quantity!!

[Close](#)

CWE-601: URL Redirection to Untrusted Site (‘Open Redirect’)

Neste caso, a aplicação web aceita uma entrada que especifica um link para um site externo e usa esse link num redireccionamento.

No decorrer do desenvolvimento do perfil do nosso utilizador pode ser introduzido um link para uma página de forma a completar a sua biografia, num caso não

seguro basta a omissão de uma tática de segurança, como por exemplo um pequeno filtro para os links considerados aceitáveis.

CWE-601: Insecure version

User Profile Card

Pedro Carneiro

27years

Aveiro, Portugal

50 books read

google.com

Update Info

Pedro Carneiro 31/01/1994 google.com Update Info

No fundo foi essa abordagem que escolhemos, temos uma lista dos links(allowList) que consideramos aceitáveis e tudo que saia desse padrão não será aceite e o utilizador é alertado.

CWE-601: Secure version

User Profile Card

Pedro Carneiro

2017years

Aveiro, Portugal

50 books read

fb.com/pedrocarneiro

Update Info

Pedro Carneiro 01/05/0004 fb.com/pedrocarneiro Update Info

Esta vulnerabilidade do nosso ponto de vista, apesar de menos passível de ser explorada classifica-se como muito grave, pois pode ser usada de modo a redirecionar os utilizadores que carregarem no link da homepage para um url malicioso com um nome de servidor idêntico ao da pagina da app, onde o utilizador possa ser alvo de ataques de phishing , perdendo os seus dados.