EA872 Laboratório de Programação de Software Básico
Atividade 12

Vinícius Esperança Mantovani **RA 247395**
Entrega (limite): 22/11/2023, 13hs.

Código-fonte documentado e explicações detalhadas sobre as mudanças introduzidas:

A seguir, são apresentadas e explicadas todas as alterações e adições feitas no código de forma a tratar da existência de arquivos “.htaccess” no webspace:

Inicialmente, vale apresentar o código usado para decodificação de strings em Base64:

```
static const unsigned char pr2six[256] =
{
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 62, 64, 64, 64, 63,
   52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 64, 64, 64, 64, 64, 64,
   64,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
   15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 64, 64, 64, 64, 64,
   64, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
   41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
   64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64
};
int Base64decode( char * bufplain, const char * bufcoded )
{
```

```
   int nbytesdecoded;
   register const unsigned char *bufin;
   register unsigned char *bufout;
   register int nprbytes;
   bufin = (const unsigned char *) bufcoded;
   while (pr2six[*(bufin++)] <= 63);
   nprbytes = (bufin - (const unsigned char *) bufcoded) - 1;
   nbytesdecoded = ((nprbytes + 3) / 4) * 3;
   bufout = (unsigned char *) bufplain;
   bufin = (const unsigned char *) bufcoded;
   while (nprbytes > 4)
   {
             *(bufout++)  =  (unsigned  char)  (pr2six[*bufin]  <<  2  |
pr2six[bufin[1]] >> 4);
            *(bufout++)  =  (unsigned  char)  (pr2six[bufin[1]]  <<  4  |
pr2six[bufin[2]] >> 2);
            *(bufout++)  =  (unsigned  char)  (pr2six[bufin[2]]  <<  6  |
pr2six[bufin[3]]);
       bufin += 4;
       nprbytes -= 4;
   }
   if (nprbytes > 1)
   {
             *(bufout++)  =  (unsigned  char)  (pr2six[*bufin]  <<  2  |
pr2six[bufin[1]] >> 4);
   }
   if (nprbytes > 2)
   {
            *(bufout++)  =  (unsigned  char)  (pr2six[bufin[1]]  <<  4  |
pr2six[bufin[2]] >> 2);
   }
   if (nprbytes > 3)
   {
            *(bufout++)  =  (unsigned  char)  (pr2six[bufin[2]]  <<  6  |
pr2six[bufin[3]]);
   }
   *(bufout++) = '\0';
   nbytesdecoded -= (4 - nprbytes) & 3;
   return nbytesdecoded;
}
```

Este código apresenta uma função de decodificação e uma lista de “char” usada nesta função. Foi obtido da página:
https://copyprogramming.com/howto/como-deserializar-uma-string-base64-em-c

No código do programa, para além do que foi adicionado sobre tratamento de Base64, foi adicionado o tratamento no parser, que identifica a presença do comando “Authorization” e coleta o parâmetro passado após os “:”, guardando-o na variável global “autenticacao", uma string. Assim, essa variável é usada na função InitThread conforme os trechos a seguir:

```
pthread_mutex_lock(&mutex_parse);
yy_scan_string(buf_entrada);
yyparse();
strcpy(requisition, reqs);
strcpy(resource, ress);
strcpy(connection, conn);
strcpy(autent, autenticacao);
memset(ress, 0, sizeof(ress));
memset(reqs, 0, sizeof(reqs));
memset(conn, 0, sizeof(conn));
memset(autenticacao, 0, sizeof(autenticacao));
pthread_mutex_unlock(&mutex_parse);
```

Neste trecho, é possível notar que a variável é copiada para a variável “autent” que é, também, uma string. Isso se faz para que o os dados de autenticação não sejam perdidos com a limpeza de “autenticacao”.

Em seguida, é chamada a função “strtok(autent, “Base ”)” para separarmos a string em dois tokens, um anterior a “Base ” e um posterior, que é o que precisamos. Desse modo, ao ser chamada novamente, a função tem seu retorno, o token posterior a “Base ”, armazenado em uma outra string “us_pa”, esta string, por sua vez, armazena o valor “user:password” em Base64 com o padding composto por caracteres “=”. Por esse motivo, seguindo, é chamada “strtok(us_pa, “=”)” para obtermos o token anterior a “=”, que é a “user:password” em Base64 sem padding. Dando continuidade, tem-se um bloco “if” cuja condição de entrada é que “user_pass” seja diferente de NULL, onde “user_pass” é o retorno de “strtok(us_pa, "=")”, ou seja, o conjunto “user:password”, conforme explicado acima.

Seguindo, caso haja um token anterior a “=”, ou seja, caso a string “us_pa” e, consequentemente, a “autent” não estejam vazias, o fluxo entra no bloco “if”. Isso foi feito, para que a função apenas entrasse no bloco “if” no caso de o recurso ser protegido por “.htaccess”, pois, caso não fosse, a string autent estaria vazia e as demais, originárias de “strtok” com ela, teriam valor “NULL”. Assim, entrando no bloco “if”, a string “user_pass” é decodificada de base64 para sua forma limpa.

```
strtok(autent, "Base ");
char *us_pa = strtok(NULL, "Base ");
char *user_pass = strtok(us_pa, "=");
char decoded[50] = "\0";

if(user_pass != NULL){
```

```
        Base64decode( decoded, user_pass );
        // write(1, user_pass, strlen(user_pass));
        // write(1,"\n", 1);
        // write(1, decoded, strlen(decoded));
        // write(1,"\n", 1);
    }
    char *user = strtok(decoded, ":");
    char *password = strtok(NULL, ":");

    // if(user != NULL && password != NULL){
        // write(1, user, strlen(user));
        // write(1,"\n", 1);
        // write(1, password, strlen(password));
        // write(1,"\n", 1);
    // }
```

Mais a frente, ainda no código acima, são criadas duas strings “user” e “password”, que são o usuário e a senha passados na requisição do cliente para o servidor. Essas duas strings são passadas de argumentos na função “getRes” (também modificada), conforme a linha:

```
            getRes(webspace, resource, connection, fd_reg, ((struct
paramsThread*)soq)->soquete_mensagem, user, password);
```

Agora, analisemos as alterações feitas no código da função “getRes”. Já de início, temos a alteração no conjunto de parâmetros, que ganhou dois novos integrantes, “user” e “password”, que são strings que armazenam o nome de usuário e a senha passados pelo cliente, respectivamente. Quanto ao interior da função, destaquemos o trecho a seguir:

```
while(teste_begin != NULL){
        strcat(caminho_recurso, "/"); //adiciona uma barra ao fim do
caminho
        strcat(caminho_recurso, teste_begin); //adiciona o proximo
diretorio ou arquivo no caminho
        strcpy(caminho_req_ht, caminho_recurso); //copia o caminho do
recurso atual para uma string na qual sera concatenado .htaccess

        int st_ht = stat(caminho_req_ht, &statht); //armazena status do
diretorio/arquivo

         //caso o arquivo nao exista
        if (st_ht == -1 && errno == ENOENT) {
        } else{
            //caso exista e seja diretório:
```

```
            if((statht.st_mode & S_IFMT) == S_IFDIR){
                strcat(caminho_req_ht, "/.htaccess"); //concatena
.htaccess ao caminho do diretorio, para ter caminho do .htaccess

                int st_access = stat(caminho_req_ht, &stataccess);
//armazena status do .htaccess

                if((st_access == -1)){
                } else{
                    strcpy(last_ht, caminho_req_ht); //copia o caminho
do ultimo htaccess encontrado (mais proximo do arquivo requisitado)
encontrado
                }
            }
        }

        teste_begin = strtok(NULL, "/");
    }

    // write(1, "Chega aqui apos uma requsicao\n", 31);

if(user != NULL && password != NULL){
        confere_if = 1;

        int fd_htaccess = open(last_ht, O_RDONLY, 0600);
        // write(1, "chega em 1\n", 11);
        read(fd_htaccess, buf_caminho_users, sizeof(buf_caminho_users));
        // write(1, "chega em 2\n", 11);
        int fd_user_password = open(buf_caminho_users, O_RDONLY, 0600);
//abre o arquivo de senhas!
        // write(1, "chega em 3\n", 11);

        //coloca nesta string o valor ja formatado (usuario:senha
cripto)
        char user_password[50] = "\0";
        // write(1, "chega em 4\n", 11);

        strcpy(user_password, user);
        // write(1, "chega em 5\n", 11);


        char *password_cryptd =  crypt(password, "95");
```

```
        // write(1, "cryptd password: ", 17);
        // write(1, password_cryptd, strlen(password_cryptd));
        // write(1, "\n", 1);
        int rBytes;

        strcat(user_password, ":");
        strcat(user_password, password_cryptd);
        // write(1, "cryptd user/password: ", 22);
        // write(1, user_password, strlen(user_password));
        // write(1, "\n", 1);

        while(rBytes = read(fd_user_password, buf_senhas,
sizeof(buf_senhas)) != 0){
            char *us_pass_document = strtok(buf_senhas, "\n");
            comp = strcmp(us_pass_document, user_password);
            // if(comp == 0){
            //     write(1, "user:password existent!",23);
            // }

            while((us_pass_document = strtok(NULL, "\n")) != NULL){
                comp = strcmp(us_pass_document, user_password);
                // write(1, "user/pass do arquivo: ", 22);
                // write(1, us_pass_document, strlen(us_pass_document));
                // write(1, "\n", 1);
                if(comp == 0){
                    // write(1, "user:password existent!",23);
                    break;
                }
            }
            if(comp == 0){
                break;
            }
        }

    }

    if(strcmp(last_ht, "\0") != 0 && confere_if == 0){
        printsAnswer(401, connection, caminho_recurso,0 , fd_reg, 0,
soquete);
        write(1, "primeira requisicao, nao tem tentativa de login\n",
48);
    } else if(strcmp(last_ht, "\0") != 0 && confere_if == 1 && comp !=
0){
```

```
        printsAnswer(401, connection, caminho_recurso,0 , fd_reg, 0,
soquete);
        write(1, "User or Password doesn't match\n", 31);
    }
```

Todo esse código apresentado acima é novo e foi escrito como forma de tratar da existência de “.htaccess” nos diretórios requisitados ou contenedores do arquivo requisitado conforme se deve tratar a questão. Além disso, vale notar que a parte em que simplesmente se concatenava “caminho” e “recurso”, strings do conjunto de parâmetros, foi substituída por um bloco complexo de código, responsável por verificar a existência de todos os “.htaccess” presentes em diretórios componentes do caminho do recurso requisitado ou, presente no próprio recurso, se este for um diretório.

A seguir, analisemos a primeira parte a qual se refere no parágrafo anterior:

```
while(teste_begin != NULL){
        strcat(caminho_recurso, "/"); //adiciona uma barra ao fim do
caminho
        strcat(caminho_recurso, teste_begin); //adiciona o proximo
diretorio ou arquivo no caminho
        strcpy(caminho_req_ht, caminho_recurso); //copia o caminho do
recurso atual para uma string na qual sera concatenado .htaccess

        int st_ht = stat(caminho_req_ht, &statht); //armazena status do
diretorio/arquivo

         //caso o arquivo nao exista
        if (st_ht == -1 && errno == ENOENT) {
            //imprime a saida necessaria (resposta ao cliente)
            int rBytes = 0;
            char webspace[100];
            strcpy(webspace, caminho);
            strcat(webspace, "/erro404.html");
            strcpy(caminho_recurso, webspace);
            printsAnswer(404, connection, caminho_recurso,0, fd_reg, 0,
soquete);

            fd2 = open(webspace, O_RDONLY, 0600);

            while((rBytes = read(fd2, buf, sizeof(buf))) != 0){
                //write(1, buf, rBytes);
                write(soquete, buf, rBytes);
            }
            //write(1, "\n", 1);
            write(soquete, "\n", 1);
            write(fd_reg, "\n", 1);
```

```
            return 404;

        } else{
            //caso seja diretório:
            if((statht.st_mode & S_IFMT) == S_IFDIR){
                strcat(caminho_req_ht, "/.htaccess"); //concatena
.htaccess ao caminho do diretorio, para ter caminho do .htaccess

                int st_access = stat(caminho_req_ht, &stataccess);
//armazena status do .htaccess

                if((st_access == -1)){
                } else{
                    strcpy(last_ht, caminho_req_ht); //copia o caminho
do ultimo htaccess encontrado (mais proximo do arquivo requisitado)
encontrado
                }
            }
        }

        teste_begin = strtok(NULL, "/");
    }
```

Neste trecho de código apresentado acima, temos um bloco “while” que depende de “teste_begin”, de maneira que, se “teste_begin” é igual a “NULL”, então o “while” é interrompido. Isso é feito como forma de iterar entre os tokens da string que dá o caminho do recurso a partir do webspace, de modo que os tokens são obtidos por “teste_begin = strtok(NULL, "/");” no final do “while”. Assim, a cada iteração concatena-se à string “caminho_recurso” uma “/” e o nome do próximo diretório, que está contido em “teste_begin”. Dessa maneira, a cada iteração, temos o caminho até um dos diretórios superiores ao arquivo requisitado ou mesmo o caminho para o arquivo requisitado (na última iteração) e, podemos usá-los para buscar por “.htaccess” em cada um deles, conforme é explicado adiante.

Ainda no trecho de código apresentado a cima, chama-se a função “stat” para armazenar os status do diretório/arquivo da iteração atual. Em seguida, verifica-se se este diretório/arquivo existe e, em caso negativo, é enviada a resposta com erro “404” para o cliente, mas no caso de ele existir, é verificado também se ele é um diretório e, em caso afirmativo, é concatenado “/.htaccess” à string “caminho_req_ht”, que até o momento tinha o mesmo conteúdo que “caminho_recurso”, pois recebeu uma cópia de “caminho_recurso” no início da iteração. Por fim, após composta a string “caminho_req_ht”, chama-se, novamente, “stat”, agora, para armazenar status de “.htaccess”. Desse modo, caso a função não retorne “-1” (não haja erro), sabemos que “.htaccess” existe e, por consequência, entra-se em um bloco “else”, no qual a string “caminho_req_ht” é copiada para a string “last_ht”, responsável por armazenar o caminho para o último “.htaccess” encontrado pelo bloco “while”, que é, no final das iterações, o “.htaccess” mais próximo do arquivo requisitado.

Em seguida, temos um bloco “if” responsável por tratar do caso em que o cliente já enviou um usuário e uma senha para serem verificados:

```
if(user != NULL && password != NULL){
```

```
        confere_if = 1;

        int fd_htaccess = open(last_ht, O_RDONLY, 0600);
        // write(1, "chega em 1\n", 11);
        read(fd_htaccess, buf_caminho_users, sizeof(buf_caminho_users));
        // write(1, "chega em 2\n", 11);
         int fd_user_password = open(buf_caminho_users, O_RDONLY, 0600);
//abre o arquivo de senhas!
        // write(1, "chega em 3\n", 11);

            //coloca nesta string o valor ja formatado (usuario:senha
cripto)
        char user_password[50] = "\0";
        // write(1, "chega em 4\n", 11);

        strcpy(user_password, user);
        // write(1, "chega em 5\n", 11);



        char *password_cryptd =  crypt(password, "95");

        // write(1, "cryptd password: ", 17);
        // write(1, password_cryptd, strlen(password_cryptd));
        // write(1, "\n", 1);
        int rBytes;

        strcat(user_password, ":");
        strcat(user_password, password_cryptd);
        // write(1, "cryptd user/password: ", 22);
        // write(1, user_password, strlen(user_password));
        // write(1, "\n", 1);

                    while(rBytes   =   read(fd_user_password,   buf_senhas,
sizeof(buf_senhas)) != 0){
            char *us_pass_document = strtok(buf_senhas, "\n");
            comp = strcmp(us_pass_document, user_password);
            // if(comp == 0){
            //     write(1, "user:password existent!",23);
            // }

            while((us_pass_document = strtok(NULL, "\n")) != NULL){
                comp = strcmp(us_pass_document, user_password);
                // write(1, "user/pass do arquivo: ", 22);
```

```
                // write(1, us_pass_document, strlen(us_pass_document));
                // write(1, "\n", 1);
                if(comp == 0){
                    // write(1, "user:password existent!",23);
                    break;
                }
            }
            if(comp == 0){
                break;
            }
        }
    }
```

No código acima, é definido o valor de “confere_if” como 1. Esse “int” tem como propósito indicar que o fluxo passou por dentro deste bloco “if”, o que indica, por consequência, que a requisição já não é a primeira, mas a requisição do cliente após o pedido de autenticação. Além disso, já de início é aberto o arquivo “.htaccess” e seu conteúdo é lido para a string “buf_caminho_users”, sendo este conteúdo o caminho para um arquivo “.htpassword” contido no diretório do servidor. Assim, em seguida este arquivo é aberto e o usuário passado pelo cliente, armazenado em “user”, é copiado para a string “user_password”, o que é sucedido pela criptografia da senha, que após criptografada é concatenada à string “user_password”, à qual já foi concatenado também “:”. Com isso feito, temos o conjunto “user/cript password” pronto e seguimos para um “while” em que é lido o conteúdo de “.htpassword” e é procurado linha por linha um conjunto “user:password” que bata com aquele passado pelo cliente. Vale ressaltar que o “while” interno faz o papel de comparador, enquanto que o while externo é responsável pela leitura completa do arquivo e, que o “int” “comp” armazena o resultado da comparação entre arquivo e string e é usado para garantir a parada do while no caso de ser encontrado um par coincidente.

Por fim, temos o código responsável por determinar se a resposta a ser enviada ao cliente é “401” ou não:

```
        if(strcmp(last_ht, "\0") != 0 && confere_if == 0){
            printsAnswer(401, connection, caminho_recurso,0 , fd_reg,
0, soquete);
            // write(1, "primeira requisicao, nao tem tentativa de
login\n", 48);
        } else if(strcmp(last_ht, "\0") != 0 && confere_if == 1 &&
comp != 0){
            printsAnswer(401, connection, caminho_recurso,0 , fd_reg,
0, soquete);
            write(1, "User or Password doesn't match\n", 31);
        }
```

No código acima, temos um bloco “if” que apenas é adentrado se houver algum “.htaccess” no caminho até o arquivo requerido, ou dentro dele próprio, se for um diretório e, se o programa não passou pelo bloco “if” anterior, o que indica que é a primeira requisição (anterior à resposta pedindo

usuário e senha). Desse modo, caso o fluxo passe por dentro deste "if", precisamos enviar o código "401" para o cliente e, é exatamente isso que é feito dentro do bloco. Ademais, tem-se um "else if" que trata, basicamente do caso em que existe ao menos um ".htaccess" no caminho, "confere_if" é 1, o que indica que esta é a segunda resposta à requisição (pedindo usuário e senha) e, é necessário que "comp" seja diferente de 0, ou seja, o usuário ou a senha passados pelo cliente não condizem com nenhum daqueles contidos no arquivo ".htpassword". Assim, ao adentrar no "else if" é enviado "401" ao cliente e é printada uma mensagem de erro de senha ou usuário.

Ademais, no caso de não se passar por nenhum dos blocos apresentados por último, a resposta enviada é a resposta normal que se daria à requisição, considerando, portanto, que o cliente enviou um usuário e uma senha corretos ou que o arquivo requisitado não é protegido por ".htaccess".

Sobre as dificuldades enfrentadas e sobre pontos ainda pendentes:

Quanto às dificuldades enfrentadas, destaco a lógica de acesso ao arquivo de senhas ".htpassword", principalmente no caso em que há mais de um destes arquivos e mais de um ".htaccess". Isso porque, é necessário fazer todo o processo de verificação de existência de ".htaccess" e, é necessário buscar pelo mais próximo ao arquivo requisitado para, por fim, podê-lo ler e procurar pelo ".htpassword". No entanto, apesar da complexidade por trás de toda a lógica da adaptação para proteção por ".htaccess", foi possível fazer um bom trabalho e evoluir, mais uma vez, o servidor.

Quanto a pontos pendentes, destaco a necessidade de implementar a proteção às requisições "HEAD" e, também, a necessidade de corrigir alguns problemas do código que não estão relacionados à proteção por ".htaccess" (relacionados a outras atividades).

Informações gerais sobre os testes:

Inicialmente, deve-se entender quantos são os ".htaccess" e os ".htpassword", seus conteúdos e, onde estão localizados.

".htaccess" e "htpassword":

1) O primeiro ".htaccess" se encontra no diretório "/dir1/dir11" do webspace e seu conteúdo é o caminho até um dos ".htpassword":

/home/pininsu/Documents/EA872/lab12-ea872/servidores_lab11/.htpassword

Este ".htpassword" contém apenas o conjunto "azul:95IooVIGDCLrM", onde a senha criptografada é "azul".

2) O segundo ".htaccess" se encontra no diretório "dir1/dir11/dir20" do webspace e seu conteúdo é o caminho até um segundo ".htpassword":

/home/pininsu/Documents/EA872/lab12-ea872/servidores_lab11/servidor_threads/.htpassword

Este ".htpassword" contém apenas o conjunto "aaa:95NR/AYS4kv/6", onde a senha criptografada é "aaa".

3) O terceiro e último ".htaccess" se encontra no diretório "dir1/dir11/dir20/dir100" do webspace e seu conteúdo é o caminho até o, também, terceiro e último ".htpassword":

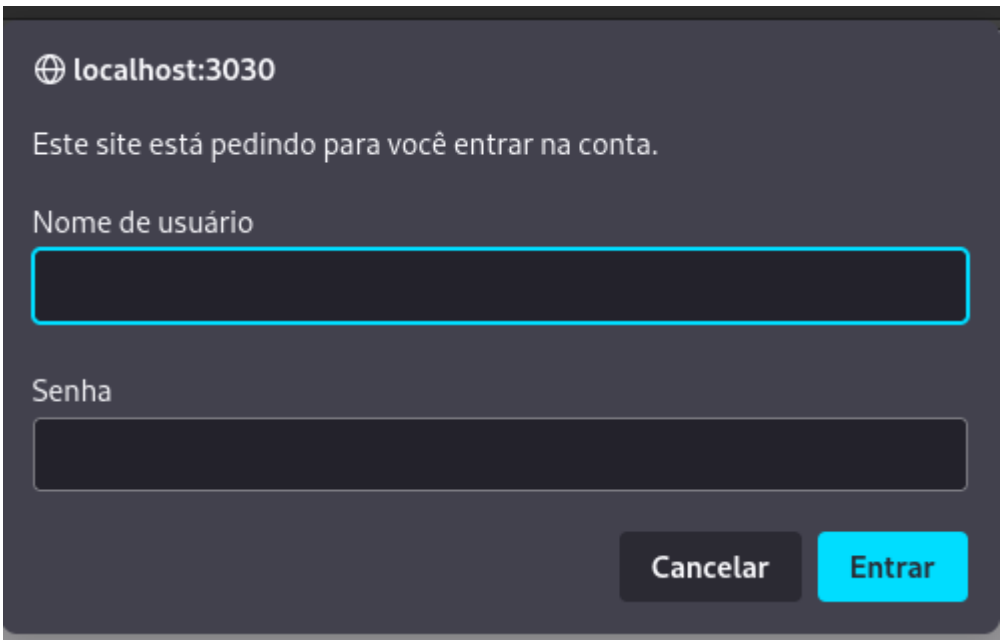
/home/pininsu/Documents/EA872/lab12-ea872/.htpassword

Este ".htpassword" contém apenas o conjunto "chique:95Xs4C2kQX8qA", onde a senha criptografada é "chique".

Documentação de três testes com erro, mas causados por situações distintas:

Agora, sigamos para o primeiro dos testes com erros, o teste com usuário errado:

Para este teste, foi usada a URL “localhost:3030/dir1/dir11/” e o resultado anterior ao preenchimento de usuário e senha foi o aparecimento da caixa abaixo:



Então, ao errar o usuário, não é mostrado nada no navegador, apenas é enviada a mesma resposta do servidor, pedindo para o cliente tentar se autenticar novamente, sem nenhuma mensagem diferente, apenas a mesma caixa para preenchimento dos dados. No entanto, a mando do meu código, o programa imprime no console a mensagem “User or Password doesn't match”. Cabe ressaltar que a tentativa de autenticação foi feita com senha “azul” e usuário “abc”, enquanto o correto seria “azul”.

No teste com senha errada, temos:

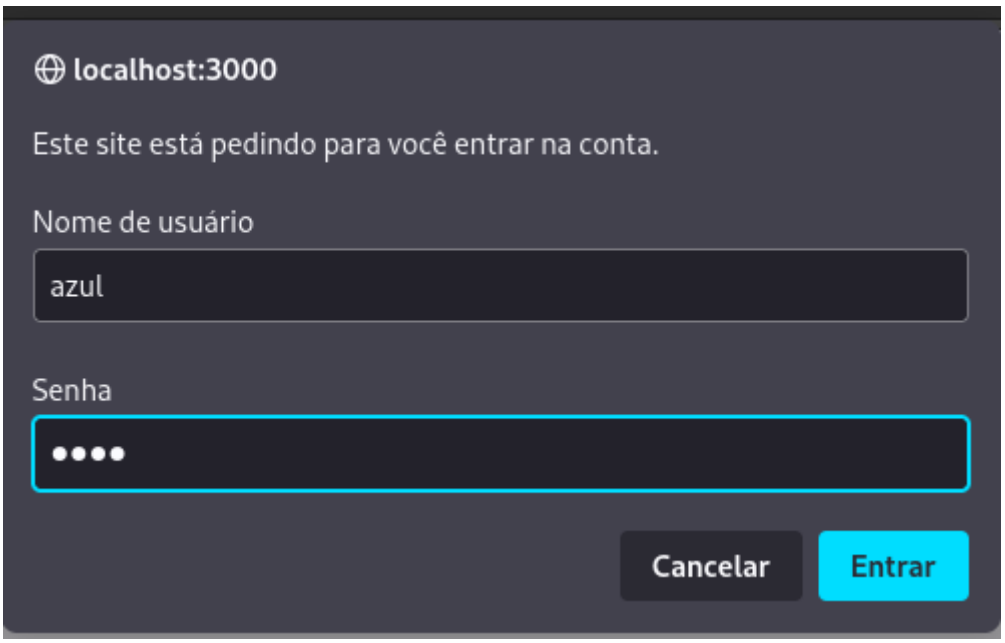
Da mesma forma como no caso em que o erro estava no usuário, ao errar a senha usando o mesmo URL, temos exatamente a mesma mensagem citada anteriormente e, conforme o caso de erro no usuário, aqui também não é mostrado nada no navegador, apenas a mesma página de autenticação e, no console temos a mensagem “User or Password doesn't match”. Usou-se usuário “azul” e senha “abc”, enquanto o correto seria “azul”.

No teste com senha não preenchida:

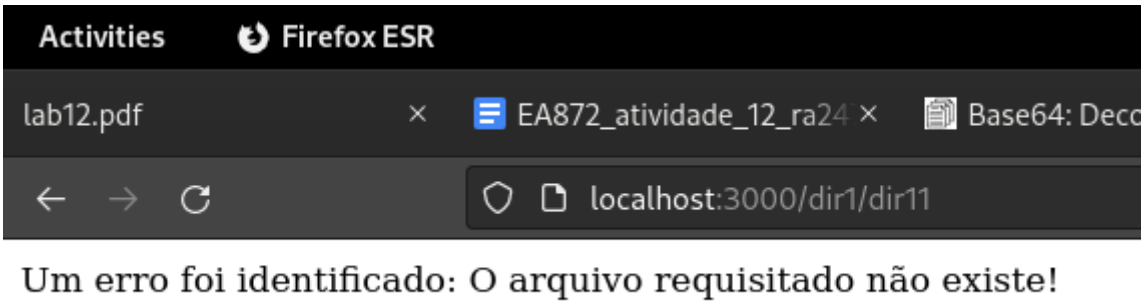
Não ocorre nada nem no navegador nem no console, uma vez que no caso de a senha ou o usuário não serem preenchidos, o fluxo do código não passa pelo bloco “if” principal, logo, apenas é reenviada a resposta pedindo autenticação como se não tivesse sido tentada uma autenticação.

Documentação de três testes com sucesso:

Iniciando pelo teste na menor profundidade, temos o “.htaccess” presente em “/dir1/dir11”, testado com a URL “localhost:3000/dir1/dir11”. Disso, temos a tela de preenchimento de usuário e senha e a preenchemos com “azul” e “azul”, usuário e senha para o “.htpassword” atualmente usado:

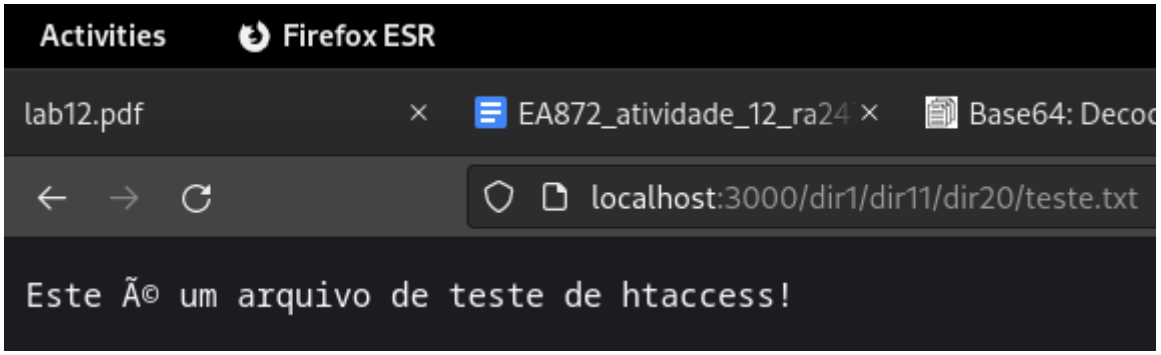


Desse modo, ao pressionar “Entrar”, obtemos o seguinte:



Esta é a página mostrada, pois não existe “index.html” nem “welcome.html” neste diretório.

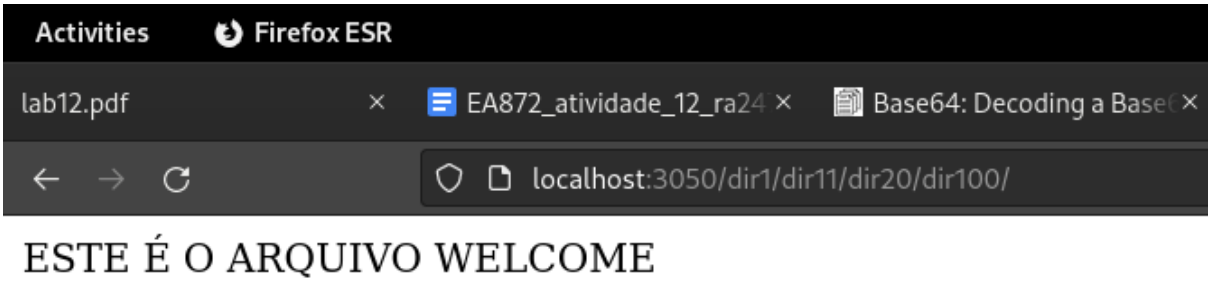
Seguindo para um nível mais profundo, testou-se no html “localhost:3000/dir1/dir11/dir22/teste.txt” e, novamente, tivemos a caixa de autenticação para ser preenchida, mas desta vez com “aaa” e “aaa”, usuário e senha salvos no “.htpassword” atualmente usado. Assim, obtemos:



Por fim, testou-se o nível mais profundo, de URL:

"localhost:3030/dir1/dir11/dir20/dir100/"

Neste teste, recebe-se a mesma caixa de autenticação, e o resultado após autenticação com "chique" de usuário e senha (contido no ".htpassword" usado atualmente), foi:



ESTE É O ARQUIVO WELCOME

Esta resposta se dá, por conta do fato de que há um arquivo welcome dentro do diretório "dir100", logo, após a autenticação, como não existe um "index.html" no diretório, é aberto "welcome.html".