EA872: Laboratório de Programação de Software Básico

Atividade 12 – Autenticação de Usuários

Eleri Cardozo e Marco Aurélio Amaral Henriques

Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas

1 Assuntos abordados

- Chamada de sistema de criptografia em Unix (crypt)
- Processo de codificação base64 (RFC4648)
- Processo de autenticação/autorização básico do HTTP (RFC7617)

2 Introdução

Frequentemente temos a necessidade de controlar o acesso de terceiros aos recursos (arquivos) armazenados em um servidor. Este controle de acesso pode ser feito de várias formas como, por exemplo, por usuário/senha, por nome de domínio, por endereço IP etc. Nesta atividade nos restringiremos ao controle de acesso baseado na autenticação de um usuário por meio de senha e focaremos na proteção de dados em um servidor web.

Um dos métodos utilizados para controlar o acesso aos arquivos em um servidor web é colocando no diretório dos arquivos que se deseja controlar um arquivo especial (normalmente chamado de .htaccess) que fornece ao servidor informações sobre o controle de acesso aos arquivos daquele diretório. Um arquivo .htaccess protege não só o diretório no qual se encontra, mas também todos os subdiretórios abaixo dele. Se o servidor percebe a existência de um .htaccess em qualquer um dos diretórios que fazem parte do caminho ao recurso solicitado, ele não envia diretamente o recurso ao cliente, mas somente um cabeçalho (de erro 401) acrescido de uma linha indicando a necessidade de autenticação (linha `WWW-Authenticate`). Supõe-se aqui que neste momento o cliente ainda não enviou nenhuma informação de autenticação junto com a requisição do recurso, ou seja, o cliente só enviou uma requisição básica, sem saber que o recurso tinha acesso controlado.

Observe que, caso haja mais de um .htaccess no caminho até o recurso, aquele mais próximo do recurso é que predomina para o controle de acesso. Por exemplo, se o navegador solicitou o recurso /docs/sigiloso/arquivo.html, o servidor deverá verificar a existência de arquivo .htaccess em / (que indica o webspace), em /docs e em /docs/sigiloso e, caso haja mais de um, as regras no que estiver mais próximo do recurso é que devem predominar. Esta necessidade de verificação do .htaccess a cada diretório é uma desvantagem deste esquema, pois traz quedas em desempenho no servidor. Portanto o esquema baseado em .htaccess só deve ser usado em casos específicos.

Observe ainda que, se o recurso solicitado estiver protegido por .htaccess (existente em qualquer parte do caminho ao recurso), o servidor deve solicitar a senha antes mesmo de verificar se o recurso existe. Caso tal recurso não exista ou exista, mas esteja sem permissão de leitura, o servidor não deve deixar que o cliente saiba disso sem ter antes fornecido uma senha correta.

Ao receber do servidor um cabeçalho indicando a necessidade de autenticação (cabeçalho contendo a linha `WWW-Authenticate`), o cliente (navegador) abre automaticamente uma janela de autenticação, solicitando um nome de usuário e uma senha. Após serem digitadas pelo usuário, estas duas informações são agrupadas na forma `usuário:senha` , codificadas segundo o padrão base64 (RFC4648) (para evitar que eventuais caracteres especiais usados no nome ou senha sejam corrompidos no percurso) e enviadas junto com uma nova requisição ao servidor solicitando o mesmo recurso que foi solicitado antes.

Ao receber a nova requisição, agora completa com as informações de autenticação, o servidor deve decodificá-las (usando o mesmo padrão base64) e compará-las com as que possui em seu cadastro (o arquivo `.htaccess` deverá conter o caminho completo até o arquivo com as senhas chamado `.htpassword`). Se as informações de usuário e senha conferirem, então o servidor pode enviar o recurso solicitado pela rede; caso contrário ele deve enviar um novo cabeçalho de erro 401 solicitando ao navegador o envio de senha.

Esta é a forma de autenticação mais simples, conhecida por `Basic Authentication`. Há outra, chamada `Digest Authentication`, que evita que a senha seja enviada de forma aberta pela rede. Esta segunda não será utilizada nesta atividade por restrições de tempo. Maiores detalhes sobre a autenticação básica e digest estão disponíveis nas RFCs 7617 e 7616, respectivamente.

3 Etapas da Basic Authentication

1. O cliente requisita recurso sem enviar informações de autenticação (pois não sabe ainda que o recurso é protegido).

2. Se o recurso está protegido por pelo menos um arquivo `.htaccess`, o servidor responde ao cliente que o recurso possui controle de acesso e solicita o par `usuário:senha` .

```
HTTP/1.1 401 Authorization Required
...
...
WWW-Authenticate: Basic realm="Espaço Protegido 1"
...
...
```

O texto que vem após “`Basic realm=`” é um título que aparecerá na janela de autenticação e serve para que o usuário tenha alguma pista sobre qual contexto ele está entrando e que tipo de senha deve entrar. É apenas um texto de orientação e pode ser qualquer coisa. Uma dica é guardar este `realm`

dentro do arquivo .htaccess e assim, para cada diretório protegido, o realm poderá ser definido adequadamente.

3. O cliente (navegador) abre automaticamente uma janela e requisita usuário e senha.

4. Após a introdução de usuário e senha, o cliente concatena estes dados, separando por ":" e codifica o conjunto (usuário:senha) usando um código chamado base64. Em seguida ele envia para servidor a mesma requisição que enviou anteriormente, só que agora acrescida de uma linha com dados de autenticação:

```
Authorization: Basic ZWNvbXA6bW9kX0FC <-- esta parte final é o par
                                          ecomp:mod_AB codificado em base64.
```

5. O servidor decodifica os dados, faz a autenticação (confere usuário e senha) e envia recurso ao cliente se tudo estiver ok. Caso contrário, servidor envia novamente uma mensagem de erro como na etapa 2.

Exemplo

Pedido do cliente (existe arquivo .htaccess no diretório chamado controlado):

```
GET /ea872/teste/controlado/segredo.html HTTP/1.1
Host:       www.fee.unicamp.br
User-Agent: Firefox
Accept:     text/xml, application/xml, application/xhtml+xml, text/html;q=0.9,
            image/png, image/jpeg,image/gif;q=0.2, text/plain;q=0.8
```

Resposta do servidor:

```
HTTP/1.1 401 Authorization Required
Date: Thu, 03 Nov 2015 11:51:21 GMT
Server: Apache/2.0 (Unix)
WWW-Authenticate: Basic realm="Espaço Protegido 1"
Transfer-Encoding: chunked
Content-Type: text/html
```

Novo pedido do cliente, agora com autorização (após usuário ter digitado nome senha):

```
GET /ea872/teste/controlado/segredo.html HTTP/1.1
Host:       www.fee.unicamp.br
User-Agent: Firefox
Accept:     text/xml, application/xml, application/xhtml+xml, text/html;q=0.9,
            image/png, image/jpeg,image/gif;q=0.2, text/plain;q=0.8
Authorization:    Basic ZWNvbXA6bW9kX0FC
```

4 Geração do arquivo de senha no servidor

Para melhorar a segurança, o cadastro de usuários e senhas deverá ficar em um ou mais arquivos de senhas armazenados fora do webspace. Para simplificar a implementação, o caminho completo até um dos arquivos de senhas será o único conteúdo de cada .htaccess. Para cada usuário deveremos ter no arquivo de senhas especificado por .htaccess uma única

linha contendo usuário:senha , que será usada na comparação dos dados que vierem pela rede.

Para evitar que as senhas possam ser conhecidas por quem tiver acesso ao arquivo, elas devem ser armazenadas de forma criptografada, usando a chamada de sistema crypt(), exemplificada pelo programa abaixo.

```
/* Compile com: gcc -o cripto cripto.c -lcrypt */
#define _XOPEN_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv){
   if (argc < 3) exit(0);
   printf("\n( Salt = %s ) + ( Password = %s ) ==> (crypt = %s)\n\n",
                              argv[2], argv[1], crypt(argv[1], argv[2]));
   return(0);
}
```

Esta chamada de sistema é a utilizada pelo Unix para proteger suas senhas. Por default ela se baseia em um algoritmo criptográfico obsoleto (DES ou Data Encryption Standard), transformando somente os primeiros oito bytes de argv[1] (a senha) em um conjunto de caracteres imprevisíveis, mas imprimíveis. Isto significa que não adianta usar senhas mais longas que oito caracteres, pois os extras serão ignorados.

Como a saída de crypt() é de caracteres imprimíveis, não é preciso aplicar a codificação base64 no resultado dela antes de armazenar em um arquivo, pois ela não retorna valores binários arbitrários.

No modo default de crypt(), o argumento argv[2] é um parâmetro de dois caracteres chamado salt cujo objetivo é modificar a forma de funcionamento interno do algoritmo de criptografia DES de maneira a tornar crypt() mais robusta contra ataques por hardware especializado. Em geral o salt é baseado em um valor pseudo-aleatório. Em nossa implementação vamos utilizar um valor único para ele em todos os casos: os dois últimos dígitos de seu RA.

Observe que os primeiros 2 caracteres retornados por crypt() são exatamente estes dois caracteres do salt, mas podemos ignorar isso e considerá-los parte da senha cifrada. Por exemplo, se usarmos o programa cripto.c acima na senha **feec** com um salt **99**, obtemos como saída a sequência **99J/wVDwmKC5s**, que será a forma de armazenarmos a versão criptografada da senha **feec** no arquivo de senhas.

Observe ainda que, no arquivo de senhas, a parte **usuário:** da linha **usuário:senha** deverá estar armazenada de forma clara, isto é, não codificada e nem criptografada. Somente a parte senha deverá estar criptografada.

Exemplo: feec:99J/wVDwmKC5s

Para o servidor comparar a senha recebida pela rede com a que está no arquivo, será preciso que ele decodifique (base64) o par **usuário:senha** recebido do navegador, cifre somente a senha usando crypt(), busque o

nome do usuário no arquivo de senhas apontado por `.htaccess` e compare a senha que o acompanha com aquela retornada por `crypt()`.

A comparação se dará, portanto, no domínio das senhas cifradas, porque essa é uma forma mais segura e que deve ser adotada em sistemas que usam autenticação de usuários. Se um mesmo usuário estiver cadastrado mais de uma vez dentro de um arquivo de senhas, deverá ser considerada somente a primeira ocorrência do usuário no arquivo.

Observações:

1. a mesma função `crypt()` pode ser usada para fazer uma codificação mais segura de senhas, podendo trabalhar com senhas de tamanhos maiores que oito caracteres e com valores de salt também mais longos. Para isso o parâmetro `salt` deve ser passado para ela com um formato especial que é o seguinte: `$n$salt$`, onde n é um valor dentre {1, 5, 6} para indicar o uso dos algoritmos {MD5, SHA-256, SHA-512}, respectivamente, e `salt` pode ser uma sequência de caracteres qualquer. O resultado da criptografia da senha será uma sequência de {22, 43, 86} caracteres imprimíveis, dependendo do algoritmo escolhido. O formato da saída será também `salt` seguido da senha criptografada, ou seja, será uma saída do tipo `$n$salt$`**……**, onde os pontos representam aqui os caracteres da senha criptografada. Caso queira experimentar esta forma mais segura no mesmo programa ilustrado acima, basta passar o `salt` com o formato especial, lembrando de fazer o "escape" do caractere $ com a barra invertida (\$) para que o shell não interprete o $ da forma tradicional;

2. dos três algoritmos, o MD5 já é considerado obsoleto e, como o DES, deve ser evitado em aplicações que requerem mais segurança;

3. a função `crypt()` é voltada para criptografia de senhas e não serve como função para cifrar e decifrar dados de um modo geral. Ela é de mão-única, ou seja, só codifica em um sentido, não sendo possível decodificar uma senha criptografada.

5 Atividades em sala de aula

Resolva e entregue, via Moodle e no prazo determinado, as respostas às seguintes questões.

1. (1,0) Explique com suas palavras qual é a ideia básica por trás do algoritmo base64 que permite que o mesmo, na etapa de codificação, transforme qualquer entrada de dados arbitrários em uma saída com dados legíveis e imprimíveis. Não é para explicar o algoritmo passo a passo, mas apenas a ideia básica seguida pelo mesmo para garantir os resultados esperados. Dica: os valores 6 e 4 do nome têm forte relação com a ideia básica.
2. (0,5) Explique por que o algoritmo base64 é necessário dentro do escopo de autenticação do protocolo HTTP.

3. (0,5) Crie, execute e documente exemplos com a função `crypt()` que mostrem que o uso em um modo não-default não sofre das limitações impostas pelo modo default. Dica: será preciso usar `crypt()` pelo menos seis vezes para mostrar a quebra de duas limitações quando se usa os modos não-default.
4. (0,5) Por que não é recomendável que o próprio arquivo `.htaccess` armazene as senhas dos usuários autorizados?
5. (0,5) Por que o esquema Basic Authentication é considerado inseguro?

6 Atividades para relatório

As atividades desta semana se baseiam na adaptação do servidor web em desenvolvimento para que o mesmo seja capaz de decidir se entrega ou não um recurso solicitado com base na existência de arquivos `.htaccess` e nas credenciais (usuário e senha) que lhe forem apresentadas. O servidor deverá seguir todos os passos da autenticação básica descrita anteriormente, de maneira a fazer com que o navegador abra uma janela de coleta de senha, receba a senha enviada por este navegador, decodifique-a e verifique se a mesma é compatível com aquela que foi registrada anteriormente. O recurso só deve ser enviado ao navegador caso haja total coincidência entre o par usuário:senha recebido da rede e aquele que já estava gravado servidor. Caso contrário, mensagens de erros apropriadas deverão ser retornadas ao navegador, conforme determina a RFC.

Um relatório deverá ser preparado e entregue contendo, no mínimo, os seguintes pontos:

- (2,0) código-fonte documentado (só com as funções que mudaram e/ou foram incluídas nesta atividade);
- (2,0) explicações detalhadas sobre as mudanças introduzidas no seu sistema por causa da autenticação, sobre as dificuldades enfrentadas e sobre pontos ainda pendentes;
- (1,5) documentação de três testes com erro, mas causados por situações distintas (usuário errado, senha errada, falta de senha etc); esta documentação deve incluir todas as informações que permitam o completo entendimento das condições em que os testes ocorreram: conteúdos dos arquivos de senhas e dos `.htaccess` envolvidos no teste, estado do webspace no momento dos testes, comandos (URLs) usadas etc;
- (1,5) documentação de três testes com sucesso e em diferentes profundidades do webspace, cada um com um arquivo `.htaccess` diferente; esta documentação deve incluir todas as informações que permitam o completo entendimento das condições em que os testes ocorreram: conteúdos dos arquivos de senhas e dos `.htaccess` envolvidos no teste, estado do webspace no momento dos testes etc. Cada teste bem sucedido, deve mostrar a recuperação do arquivo após a conclusão de uma autenticação diferente (usuário e/ou senha diferente) em cada um de três níveis de diretórios, sendo cada nível pertencente ao

anterior que foi testado.

Obs: apesar de ser recomendável que cada aluno desenvolva seu próprio programa para decodificar dados codificados em base64 e implantá-lo dentro do servidor em construção, isto não será exigido nesta oportunidade. Portanto os alunos podem recorrer a códigos já prontos disponíveis na web, mas devem tomar cuidado de verificar se tais códigos estão funcionando corretamente. Uma forma de fazer esta verificação é comparando os resultados produzidos por eles com aqueles produzidos com o comando base64 disponível no Linux (man base64).