Segurança Informática - Trabalho 2

> Segundo trabalho de Segurança Informática do grupo 03 da turma LEIC51D.

The English version of this document is available here.

Autores

- 48089 André Páscoa
- 48280 André Jesus
- 48287 Nyckollas Brandão

Professor: Eng. José Simão

@ISEL
Licenciatura em Engenharia Informática e de Computadores
Segurança Informática - LEIC51D - Grupo 03
Semestre de Inverno 2022/2023

Exercícios

Exercício 1

a) A autenticidade das mensagens no *record protocol* é garantida através da utilização de um MAC (Message Authentication Code), sendo calculado sobre a mensagem e o seu respetivo segredo partilhado. Este MAC é então enviado juntamente com a mensagem, de forma a que o receptor possa verificar a sua autenticidade.

b) A deteção de inserção ou adulteração maliciosa de mensagens é feita através da mensagem **Finished** que é enviada no fim do handshake, indicando que o handshake foi concluído com sucesso no cliente. Esta mensagem contém um MAC que é calculado sobre o segredo partilhado e o respetivo hash do handshake. O servidor então calcula o seu próprio MAC e compara-o com o MAC recebido, verificando se o handshake foi concluído com sucesso e que não houve alterações maliciosas.

c) A utilização de chaves públicas e privadas para estabelecer o *pre-master secret* não garante a propriedade **perfect forward security**, porque o *pre-master secret* gerado pelo cliente é partilhado com o servidor. Se um atacante encontrar a chave privada do servidor, conseguirá decifrar o *pre-master secret* e, consequentemente, decifrar todas as mensagens trocadas entre o cliente e o servidor.

Exercício 2

O erro de programação em questão é uma vulnerabilidade no sistema de autenticação, visto que o atacante pode realizar um ataque de dicionário para descobrir a password que, juntamente com o salt, gera o hash associado ao utilizador. Como o atacante tem acesso ao salt, pode gerar todos os hashes possíveis para todas as passwords possíveis, sem interagir com a interface de autenticação, e comparar com o hash exposto. Caso o hash gerado seja igual ao hash exposto, o atacante pode então utilizar a password correspondente para se autenticar no servidor, utilizando apenas uma tentativa na interface de autenticação.

Exercício 3

a) Como a estrutura do cookie é conhecida e é constituida pelo identificador do utilizador e o seu hash, se a função de hash não for autenticada (e.g. SHA-256), o atacante pode gerar o cookie e fazer-se passar pelo utilizador.

b) Para evitar este ataque, o cookie deve ser gerado no servidor com uma função de hash autenticada (HMAC), de forma a que o atacante não consiga gerar um cookie válido, pois não tem acesso à chave simétrica que é armazenada no servidor.

Exercício 4

a) O valor indicado no scope representa os recursos a que o cliente pretende ter acesso, estas permissões são fornecidas pelo dono de recursos através do servidor de autorização. Como o cliente é que decide que permissões ele pretende ter acesso, este é que determinado o valor do scope.

b) O cliente e o servidor de autorização comunicam indiretamente através do *browser* do dono de recursos **quando o dono de recursos não tem uma sessão ativa no servidor de autorização**. Neste caso, o cliente redireciona o *browser* do dono de recursos para o servidor de autorização, que pede ao dono de recursos para se autenticar. Após a autenticação, o servidor de autorização redireciona o *browser* do dono de recursos para o cliente, que recebe o *access token* e o *id token*.

c) O *access_token* é um token que permite ao cliente fazer pedidos ao servidor de recursos. Este token é gerado pelo servidor de autorização sendo enviado para o cliente. Um *id_token* é um token que contém informações sobre o utilizador, sendo enviado ao cliente após o processo de autenticação. Este token tem o formato de um *JSON Web Token* (JWT), e apenas existe no protocolo *OpenID Connect*.

Exercício 5

a) A família de modelos RBAC contribui para a implementação deste princípio, pois com a utilização destes modelos, é possível limitar as permissões a partir do mecanismo de roles, de forma a que um utilizador apenas tenha as permissões necessárias para executar as suas tarefas. O princípio de *least privilege* indica que um utilizador deve ter apenas as permissões necessárias para executar as suas tarefas, e não mais do que isso. Este princípio é importante para garantir que um utilizador não tenha acesso a informações que não lhe são necessárias, e que não possa aceder a recursos que não lhe são permitidos. Se averiguar-se que um role tem mais permissões do que as necessárias, deve ser criado um novo role com apenas as permissões necessárias, e atribuir este novo role ao utilizador.

b) O utilizador `u2` não poderá aceder ao recurso `R`, pois o role do utilizador `u2` é `r2`, que tem as permissões `pa` e `pb`, herdadas dos roles `r0` e `r1`, respetivamente. Como o role `r2` não herda a permissão `pc` de `r4`, o utilizador `u2` não poderá aceder ao recurso `R`.

Exercício 6

a)

O servidor HTTPS implementado no exercício 6 encontra-se no ficheiro `https.server.js`.

Para executar o servidor, foi necessário realizar as seguintes configurações:

1. Fazer a configuração adequada do ficheiro `hosts` do sistema operativo, para que o endereço `www.secure-server.edu` seja resolvido para `localhost`;
2. Gerar ficheiros `PEM` para a chave privada e certificado do servidor, com recurso ao comando `openssl`, para que o servidor possa ser executado com sucesso, sem autenticação do cliente:

   ```
   # Gerar certificado
   openssl pkcs12 -in secure-server.pfx -nokeys -out certificate.pem -password pass:

   # Gerar chave privada não encriptada (--nodes)
   openssl pkcs12 -in secure-server.pfx -nocerts -out privatekey.pem --nodes -password pass:
   ```

3. Instalar os certificados necessários para o servidor HTTPS:
   - `CA1-int.cer` e `CA2-int.cer` nas *Intermediate Certification Authorities*;
   - `CA1.cer` e `CA2.cer` nas *Trusted Root Certification Authorities*.
4. Gerar o ficheiro `CA2.pem` que contém o certificado da root certificate authority `CA2`, usado para validar o certificado do cliente:

```
openssl x509 -inform der -in CA2.cer -out CA2.pem
```

Para realizar a conexão através do browser com a autenticação do cliente `Alice_2`, foi necessário:

1. Instalar o certificados:
    - `Alice_2.pfx` no *Personal*;
    - `CA1-int.cer` e `CA2-int.cer` nas *Intermediate Certification Authorities*;
    - `CA1.cer` e `CA2.cer` nas *Trusted Root Certification Authorities*.

b)

Para realizar a conexão entre o cliente implementado, foi necessário gerar a truststore `truststore.jks` com o certificado da root certificate authority `CA2` e o intermediário `CA2-int`, usado para validar o certificado do servidor:

```
keytool -importcert -file "CA2.cer" -keystore truststore.jks -alias "CA2"
keytool -importcert -file "CA2-int.cer" -keystore truststore.jks -alias "CA2-int"
```

Esta truststore é então colocada na propriedade `javax.net.ssl.trustStore` do sistema.

Exercício 7

O exercício 7 foi implementado em dois diretórios distintos, `client` e `server`.

O nosso modelo de políticas de acesso foi implementado conforme o modelo RBAC1, contendo os seguintes roles:

- `free`: utilizador apenas pode ver tasks;
- `premium`: utilizador pode ver e editar tasks;
- `admin`: utilizador pode ver e editar tasks.

O role `free` tem uma permissão de leitura sobre o recurso `tasks`. O role `premium` herda as premissões do role `free`, e adiciona uma permissão de escrita sobre o recurso `tasks`. O role `admin` herda as premissões do role `premium`.

Para executar o servidor, é necessário executar o comando `npm start` no diretório `server`.