ISEL

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Segurança Informática

Trabalho 2

48292 - Tiago Neves 48355 -Filipa Machado 48927 - Iara Costa

Parte 1

Exercício 1

a)

Perfect forward secrecy(PFS) é a propriedade do *protocolo handshake* que garante que, se a chave privada for comprometida, não é possível decifrar *master secret* anteriores. Contudo, ao usar o esquema de cifra assimétrica *RSA* o *perfect forward secrecy*(PFS) não é garantido devido à forma como as chaves são usadas durante o processo de aperto de mão.

Quando o RSA é usado no TLS para troca de chaves, o cliente gera uma *pre master key*, criptografa essa chave com a chave pública do servidor e a envia. O servidor, então, usa sua chave privada para desencriptar a chave *pre master key*.

O problema com essa abordagem em termos de *perfect forward secrecy*(PFS) é que, se um atacante em algum momento conseguir obter a chave privada do servidor, ele pode retroativamente desencriptar todas as sessões passadas que foram protegidas com a *pre master key* compartilhada. Isso comprometeria a confidencialidade dessas comunicações antigas.

b)

Dois possíveis ataques ao *record protocol*:

1. Ataque de Falsificação de Registos (*Record Forgery Attack*):

   Ataque: Um atacante tenta forjar registos TLS/SSL para enviar informações falsas.

   Prevenção: Uso de algoritmos de assinatura digital para verificar a autenticidade dos registos. Certificados digitais e cadeias de certificados são usados para autenticação mútua.

2. Ataque de Interceção (*Man-in-the-Middle - MitM*):

   Ataque: Um atacante intercepta as comunicações entre o cliente e o servidor.

Prevenção: Autenticação mútua através de certificados digitais, onde o cliente e o servidor autenticam um ao outro, ajuda a garantir que a comunicação seja estabelecida com entidades confiáveis.

Exercício 2

Tendo em conta o contexto dado e a informação de validação das *passwords* descritas no exercício o uso de *CAPTCHA* pode sim ajudar a mitigar ataques de dicionário à informação de validação. Isso acontece, pois, o *CAPTCHA* tem como objetivo distinguir entre humanos e computadores, dando tarefas fáceis para diferenciá-los, dessa maneira o mesmo torna mais difícil com que programas automatizados submetam várias tentativas de senha, por exigir interação humana. *CAPTCHA* adiciona também uma camada de proteção contra ataques de dicionário, porque mesmo obtendo o *hash*, a *password* e o número aleatório o usuário continuaria a ter de resolver a tarefa do *CAPTCHA*.

Exercício 3

O servidor pode detetar se o conteúdo do *cookie* foi adulterado no navegador verificando a assinatura do *JSON Web Token* (JWT). O servidor, ao criar o JWT, assina digitalmente usando uma chave secreta. Ao receber o *cookie* de volta, o servidor verifica a assinatura para garantir que o *token* não foi alterado. Se a assinatura não corresponder, quer dizer que o conteúdo do cookie foi modificado.

Exercício 4

a)

No fluxo *authorization code Grant* a assinatura do *JSON Web Token* (JWT) é utilizada para o fornecedor de identidade fornecer informações sobre a autentificação do usuário entre ao cliente. Tendo como objetivo fornecer essas informações de maneira segura e eficiente, permitindo que o cliente confirme a autenticidade do que recebeu, isso no contexto do *OpenID*.

Já tendo em conta o *OAuth 2.0* JWT tem outro propósito, pois os *ID tokens* são utilizados para acessar recursos protegidos.

b)

Após o dono de recursos ter autorizado e consentido o uso de um recurso a aplicação cliente solicita uma autorização dada pelo servidor de recursos autentificando o cliente no servidor de autorização e dando permissão à aplicação cliente. Após isso o servidor de autorização retorna um código de autorização para a mesma, que usando esse código, faz solicitação para trocar o mesmo por um *access token*, se o código for válido emite esse *access token*. Com o token de acesso a aplicação cliente pode fazer pedidos ao servidor de recursos para acessar os recursos.

Exercício 5

O conjunto total de permissões que pode existir numa sessão com o utilizador u4 seria {p1,p2,p3,p4,p5}. Estas são as permissões associadas ao papel 'Supervisor' e aos papéis hierarquicamente inferiores a 'Supervisor' na política RBAC1.

Parte 2

Exercício 6

Para a realização do exercício 6 do trabalho foram necessárias algumas configurações:

- Para realizar uma ligação ao servidor HTTPS sem autenticação, foi necessário importar para o browser o certificado secure-server que foi emitido por CA1-Int.
- Para realizar a ligação ao servidor HTTPS com autenticação, além de fornecermos o certificado da raiz de autenticação do servidor CA2, foi também necessário importar para o browser o certificado Alice_2 que foi emitido por CA2-Int, este último emitido por CA2. Desta forma, quando tentamos aceder ao servidor é nos solicitado o certificado com o qual pretendemos realizar a autenticação.
- Na realização de uma aplicação recorrendo a JCA foi necessário criar um ficheiro *.jks* no qual é armazenado o certificado da raiz autenticadora CA2.

Exercício 7

Na realização do exercício 7 foi necessário configurar a política de controlo de acessos. Para tal, e como pretendido, utilizámos o modelo RBAC1 o qual foi integrado no código da aplicação com o uso da biblioteca *Casbin*. Para configurar o modelo RBAC e posterior utilização com o *Casbin* criámos 2 ficheiros: *rbac_model.conf* e *rbac_policy_app.csv*. O primeiro ficheiro é onde configuramos o modelo de permissões e de políticas. No segundo ficheiro é onde atribuímos aos diferentes utilizadores os seus papéis. Neste ficheiro também fica definido as permissões de cada papel assim como a sua hierarquia.