Respostas da parte teórica

Exercício 1

a) No protocolo Transport Layer Security (TLS), a autenticidade das mensagens no *record protocol* é assegurada por meio do uso de certificados digitais e infraestruturas de chave pública. Quando um cliente e um servidor estabelecem uma conexão TLS, eles trocam certificados digitais que contêm as chaves públicas de cada parte. Esses certificados são emitidos por *trust-anchors* chamados de autoridades de certificação (CA), que verifica a identidade do proprietário do certificado antes de emitir o certificado. O cliente e o servidor usam suas respetivas chaves públicas para estabelecer uma conexão segura e autenticar um ao outro. Eles fazem isso enviando mensagens criptografadas usando a chave pública da outra parte. O destinatário pode desencriptar a mensagem usando sua chave privada, que só ele possui. Esse processo garante que apenas o destinatário pretendido possa ler a mensagem, fornecendo autenticação e confidencialidade.

b) O handshake TLS detecta a inserção ou adulteração maliciosa de mensagens por meio do uso de assinaturas digitais. Durante o processo de handshake, o servidor envia um certificado ao cliente que contém sua chave pública. O certificado é assinado por uma autoridade de certificação (CA), que é um *trust-anchor* que verifica a identidade do servidor e emite o certificado. O cliente verifica a autenticidade do certificado do servidor verificando a assinatura da CA e garantindo que o certificado não foi adulterado. Se a assinatura não for válida ou o certificado for adulterado, o cliente rejeitará o certificado e encerrará a conexão.

c) O *pre-master secret* é estabelecido usando chaves públicas e privadas. Esse processo, conhecido como *key exchange*, permite que o cliente e o servidor negociem uma chave secreta compartilhada que é usada para criptografar e desencriptar os dados transmitidos pela conexão segura. No entanto, esse método de troca de chaves não garante *perfect forward security*. Este é a propriedade do *handshake* que garante que, se a chave privada for comprometida, não é possível decifrar *master secret* anteriores (e consequentemente não é possível decifrar mensagens do *record protocol*). O método de troca de chaves usando chaves públicas e privadas não fornece *perfect forward security* porque a chave privada é usada para desencriptar o *pre-master secret*. Se um invasor conseguir obter a chave privada, poderá usá-la para desencriptar o *pre-master secret* e potencialmente desencriptar as comunicações anteriores.

Exercício 2

Se a informação sobre os utilizadores, *hashs* e respetivos *salts*, ficou exposta numa página da aplicação web, devido a um erro de programação, vai potencialmente facilitar um ataque de dicionário através da interface de autenticação onde o número de tentativas é limitado. Um ataque de dicionário é um tipo de ataque de força bruta

que envolve tentar adivinhar uma senha através de uma lista predefinida de palavras ou frases. Se as informações sobre os *hashes* e *salts* de cada utilizador estiverem disponíveis, um atacante poderá usar essas informações para iniciar um ataque de dicionário. Em um ataque de dicionário, o atacante tenta adivinhar a senha de cada utilizador concatenando o palpite com o valor *salt* conhecido e, em seguida, fazendo o *hash* do resultado. Se o *hash* resultante corresponder ao *hash* exibido na página da Web, o atacante sabe que o palpite foi correto e adivinhou com sucesso a senha desse utilizador.

Exercício 3

a) Um atacante que conhece o identificador de outro utilizador e tem acesso à construção do *cookie* pode representar o utilizador criando um *cookie* com a mesma estrutura do original, mas com o próprio identificador do atacante no lugar da vítima. Para fazer isso, o atacante precisa conhecer a estrutura do *cookie* e ser capaz de criar um cookie com o formato correto. O atacante pode então usar a função *hash* para gerar um *hash* de seu próprio identificador e incluir esse *hash* no novo *cookie*. Depois que o atacante cria o *cookie*, ele pode enviá-lo para o aplicativo da *web* no lugar do *cookie* original. Se o aplicativo da *web* não validar adequadamente a autenticidade do *cookie*, ele pode aceitar o *cookie* do atacante como genuíno e permitir que o atacante faça-se passar pela vítima. *Cross-Site Request Forgery Attack (XSRF)* O principal problema com os cookies é que os sites não conseguem distinguir se as solicitações vêm do utilizador real ou de outra pessoa.

b) Para se proteger contra esse tipo de ataque, é importante que o aplicativo da *web* valide adequadamente a autenticidade dos *cookies*. Isso pode ser feito incluindo uma chave secreta no *cookie* que é conhecida apenas pelo aplicativo da *web* e usando essa chave para verificar a integridade do *cookie* antes de aceitá-lo. Além disso, a comunicação entre o navegador e o aplicativo deve ser protegida usando *HTTPS*, o que ajuda a proteger contra-ataques *man-in-the-middle* que podem intercetar e modificar os *cookies* enviados entre os dois. Também pode ser utilizado a *flag HttpOnly* informa ao navegador para não permitir que o JavaScript tenha acesso ao *cookie*. Essa é a melhor defesa contra ataques *XSS* porque impede que *hackers* consigam recuperar e usar informações em sites.

Exercício 4

a) O valor indicado no *scope* é determinado pela aplicação cliente. O *Auth 2.0* é um *open standard* para autorização que permite que aplicativos de terceiros obtenham acesso limitado aos recursos de um utilizador, como seus dados em outro serviço da *web*, sem exigir que o utilizador compartilhe suas credenciais de login. O *OpenID Connect* é um protocolo desenvolvido com base no *OAuth 2.0* que adiciona uma camada adicional de verificação de identidade e permite que os utilizadores se autentiquem com um único conjunto de credenciais de login. No fluxo *authorization code grant*, o aplicativo cliente inicia o processo direcionando o navegador do usuário para o servidor de autorização com uma solicitação para permitir acesso aos recursos do utilizador. Como parte dessa solicitação, o cliente pode especificar o *scope* do acesso que está a ser solicitado. Em suma, o *scope* no fluxo *authorization code grant* é determinado pelo aplicativo cliente e especifica as permissões que o cliente está a

solicitar ao utilizador. Este utilizador tem a opção de conceder ou negar o acesso pedido.

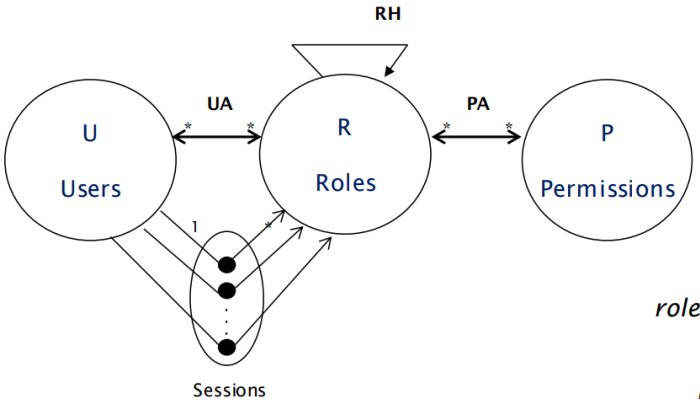
b) As situações em que o cliente e o servidor de autorização comunicam indiretamente através do browser do dono de recursos é:
Quando o cliente inicia o processo de autorização: O cliente direciona o navegador do utilizador para o servidor de autorização com uma solicitação para ter acesso aos recursos do utilizador. O servidor de autorização então apresenta ao utilizador uma tela de login e solicita que ele forneça as suas credenciais.
Quando o utilizador concede acesso ao cliente: Se o utilizador conceder o acesso pedido, o servidor de autorização emite um código de autorização para o cliente e redireciona o navegador do utilizador de volta para o aplicativo do cliente.
Quando o cliente troca o código de autorização por um *token* de acesso: Após receber o código de autorização, o cliente pode utilizá-lo para solicitar um *token* de acesso ao servidor de autorização. Essa solicitação é feita por meio do navegador do utilizador, que atua como intermediário entre o cliente e o servidor de autorização.

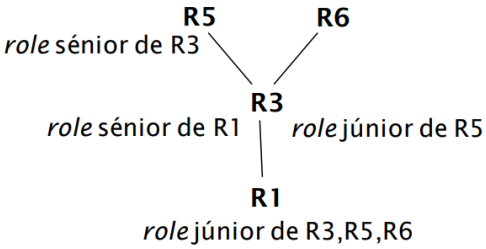
c) O *access_token* é uma *string* que representa a autorização concedida ao cliente pelo dono de recursos. Ele pode ser usado pelo cliente para ter acesso aos recursos protegidos no servidor de recursos. O *access_token* geralmente é um *token* da *web JSON (JWT)* que contém um conjunto de reivindicações sobre a autorização, como o *scope* do acesso e o tempo de validade do *token*. O *id_token*, por outro lado, é um *JWT* que contém informações sobre a autenticação do utilizador. Ele inclui reivindicações como o identificador único do utilizador, o seu nome e outras informações de identificação. Além de ser usado para identificar o utilizador, o *id_token* também pode ser usado para transmitir de forma segura informações sobre o utilizador entre o cliente e o servidor de autorização. Em suma, o *access_token* permite acesso aos recursos protegidos no servidor de recursos, enquanto o id_token permite identificar o utilizador e transmitir informações sobre o utilizador entre o cliente e o servidor de autorização. Ambos os *tokens* são emitidos pelo servidor de autorização e geralmente são do tipo *JWT*.

Exercício 5

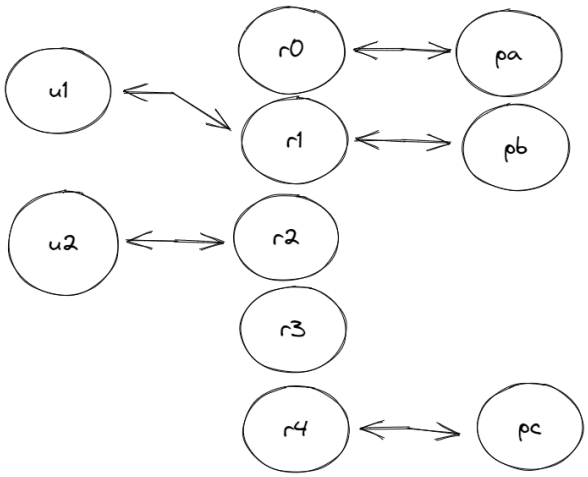
a) O modelo *RBAC* estabelece que cada utilizador de um sistema tem uma determinada função ou papel, e que esses papéis são atribuídos a conjuntos específicos de privilégios e permissões. Por exemplo, um administrador do sistema pode ter privilégios de acesso total enquanto um utilizador comum pode ter privilégios limitados. Isso permite que o sistema conceda automaticamente o conjunto mínimo de privilégios e permissões necessários para realizar uma tarefa específica, de acordo com o papel do utilizador. Dessa forma, a família de modelos *RBAC* contribui para a implementação do princípio de privilégio mínimo, pois permite que o sistema conceda automaticamente apenas os privilégios e permissões necessários para realizar uma tarefa específica, sem a necessidade de conceder privilégios adicionais que não são necessários. Isso ajuda a minimizar os riscos de acesso não autorizado ou de violação de segurança, pois limita o acesso de cada utilizador ao conjunto mínimo de privilégios e permissões necessários para realizar suas tarefas.

b)





Não é possível determinar se o utilizador *u2* poderá aceder ao recurso *R*. O *RBAC* atribui uma ou mais "funções" a cada utilizador e concedendo permissões diferentes a cada função. Neste caso, para determinar se o utilizador *u2* tem acesso ao recurso *R*, é preciso verificar se ele possui a permissão *pc* e *pb*, bem como se ele está associado a uma função que lhe conceda essas permissões. Para verificar se o usuário *u2* tem acesso ao recurso *R*, seria necessário verificar se ele possui as permissões *pc* e *pb* atribuídas diretamente a ele ou se ele está associado a uma função que possua essas permissões.



Referências bibliográficas

https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

https://www.techtarget.com/searchsecurity/definition/salt

https://quadrantsec.com/security-issues-cookies/

https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow

https://oauth.net/2/

https://openid.net/connect/

https://www.oauth.com/oauth2-servers/access-tokens/authorization-code-request/#:~:text=The%20authorization%20code%20grant%20is,to%20request%20an%20access%20token.

https://auth0.com/blog/id-token-access-token-what-is-the-difference/

https://www.ietf.org/archive/id/draft-ietf-oauth-v2-22.html

https://cloud.google.com/docs/authentication/token-types#:~:text=ID%20tokens%20are%20JSON%20Web,and%20used%20by%20the%20application.

https://www.smartspaces.app/blog/role-based-access-control/

https://en.wikipedia.org/wiki/Role-based_access_control

https://www.cloudflare.com/pt-br/learning/access-management/role-based-access-control-rbac/

Realizado por:

Grupo 19

Raul Santos

44806