Instituto Superior de Engenharia de Lisboa

Segurança Informática

Segundo Trabalho Prático, Semestre de Inverno de 22/23

Entregar até 20 de dezembro de 2022

(i) documento com respostas às questões (ii) código fonte desenvolvido nas alíneas 6 e 7

1. Considere o protocolo TLS e as infraestruturas de chave pública:

   (a) De que forma é garantida a autenticidade nas mensagens no *record protocol*?

   (b) O sub-protocolo *handshake* assume que o canal de comunicação não envia/recebe mensagem com confidencialidade nem integridade. Sendo assim, como é que o *handshake* deteta a inserção ou adulteração maliciosa de mensagens?

   (c) Considere a versão do TLS em que o *pre-master secret* é estabelecido usando chaves públicas e privadas. Porque motivo esta forma de estabelecimento de chaves não garante a propriedade *perfect forward secrecy*?

2. Considere uma aplicação *web* onde as *passwords* são armazenadas na forma $h_u = H(pwd_u||salt_u)$, sendo H um função de hash, pwd_u a password do utilizador u e $salt_u$ um número aleatório gerado no momento do registo do utilizador u, em que $||$ indica a concatenação de bits.

   Devido a um erro de programação, a informação sobre os utilizadores, *hashs* e respetivos *salts*, ficou exposta numa página da aplicação web. Discuta se este erro facilita um ataque de dicionário através da interface de autenticação onde o número de tentativas é limitado.

3. Considere uma aplicação web que guarda no *browser cookies* contendo o par $(u, H(u))$, sendo u o identificador de um utilizador e H uma função de *hash*. Assuma que a construção do *cookie* é conhecida. A comunicação entre *browser* e aplicação é feita sobre HTTPS.

   (a) Como poderia um atacante fazer-se passar por outro utilizador para o qual sabe o seu identificador?

   (b) Que alterações propõe para evitar o ataque anterior?

4. Considere a norma OAuth 2.0 e OpenID Connect no fluxo *authorization code grant*:

   (a) O valor indicado no *scope* é determinado pela aplicação cliente ou pelo dono de recursos?

   (b) Em que situações o cliente e o servidor de autorização comunicam indiretamente através do *browser* do dono de recursos?

   (c) Qual a diferença entre o *access_ token* e o *id_ token*?

5. Considere os modelo de controlo de acessos RBAC1.

   (a) Em segurança da informação, o princípio de privilégio mínimo determinada que cada operação (ou conjunto de operações) deve ser realizada com o conjunto mínimo de permissões. De que forma a família de modelos RBAC contribui para a implementação deste princípio?

   (b) Na política $RBAC_1$:
   $U = \{u_1, u_2\}$
   $R = \{r_0, r_1, r_2, r_3, r_4\}$
   $P = \{p_a, p_b, p_c\}$
   $UA = \{(u_1, r_1), (u_2, r_2)\}$
   $PA = \{(r_0, p_a), (r_1, p_b), (r_4, p_c)\}$
   $RH = \{(r_0 \leq r_2), (r_1 \leq r_2), (r_1 \leq r_3), (r_2 \leq r_4), (r_3 \leq r_4)\}$

   Considere a existência da sessão s_2, na qual está o utilizador u_2. Neste contexto, o utilizador pretende aceder ao recurso W que exige a permissão p_c, e p_b. O utilizador u_2 poderá aceder a este recurso?

6. Configure um servidor HTTPS, **sem** e **com** autenticação de cliente. Tenha por base o ficheiro do servidor no repositório github da disciplina (`nodeJS-TLS\http-server-base.js`). Considere o certificado e chave privada do servidor `www.secure-server.edu` em anexo, o qual foi emitido pela `CA1-int` do primeiro trabalho.

   $\Longrightarrow$ No documento de entrega descreva sucintamente as configurações realizadas.

   (a) Usando um *browser*, ligue-se ao servidor HTTPS sem e com autenticação de cliente `Alice_2`.

(b) Usando a JCA, realize uma aplicação para se ligar ao servidor HTTPS, sem autenticação de cliente.

Tenha em conta as seguintes notas:

- Comece pelo cenário base: servidor HTTPS testado com cliente *browser* sem autenticação. Para executar o servidor tem de ter instalado o ambiente de execução node.js. Após a configuração mínima na secção `options` do ficheiro `server.js`, o servidor é colocado em execução com o comando:

  `node http-server-base.js`
- Para converter ficheiros CER (certificado) e PFX (chave privada) para PEM use a ferramenta de linha de comandos *OpenSSL*. Existem vários guias na Internet sobre o assunto, tendo todos por base a documentação oficial (`https://www.openssl.org/docs/manmaster/man1/`). Um exemplo de um desses guias pode ser visto aqui:

  `https://www.sslshopper.com/article-most-common-openssl-commands.html`.

  O ficheiro `secure-server.pfx` não tem password.
- O certificado fornecido para configurar o servidor associa o nome `www.secure-server.edu` a uma chave pública. No entanto, o servidor estará a executar localmente, em *localhost*, ou seja, `127.0.0.1`. Para o *browser* aceitar o certificado do servidor, o nome do domínio que consta no URL introduzido na barra de endereços, `https://www.secure-server.edu:4433`, tem de coincidir com o nome do certificado. Para que o endereço `www.secure-server.edu` seja resolvido para `localhost`, terá de fazer a configuração adequada no ficheiro `hosts`, cuja localização varia entre diferentes sistemas operativos: `https://en.wikipedia.org/wiki/Hosts_(file)`.

7. Realize uma aplicação *web* para ver e adicionar tarefas de um utilizador Google através da *Google Tasks API* [1].

   Requisitos da aplicação:

   - A aplicação só aceita pedidos de utilizadores autenticados, exceto na rota de autenticação. Os utilizadores são autenticados através do fornecedor de identidade social Google, usando o protocolo OpenID Connect [2, 3]. Após autenticação do utilizador na aplicação é dado acesso aos serviços, em função do papel atribuído ao utilizador pela política de segurança;
   - Política de controlo de acessos: A aplicação usa o modelo RBAC1 para gerir 3 papéis (*roles*): `free`, `premium` e `admin`. No papel `free` os utilizadores apenas podem ver as suas tarefas, no papel `premium` podem ver e adicionar tarefas. Por simplificação do exercício, o papel `admin` não tem funcionalidades específicas atribuídas, mas herda todas as outras.
   - A política de controlo de acessos é aplicada usando a biblioteca *Casbin* [4]. A política é carregada no início da aplicação e usada pela biblioteca quando necessário nos *policy enforcement points* (PEP) da aplicação. Pode testar diferentes tipos de políticas usando o editor web do *Casbin*: `https://casbin.org/en/editor/`
   - O estado de autenticação entre o *browser* e a aplicação web é mantido através de *cookies*;
   - Os dados guardados para cada utilizador podem estar apenas em memória.

   Não pode usar os SDK da Google para realizar os pedidos aos serviços. Os mesmos têm de ser feitos através de pedidos HTTP construídos pela aplicação *web*.

   Considere os *endpoints* de registo e autorização de aplicações nos serviços Google, e as instruções para criação de credenciais para a aplicação [5]:

   - Registo de aplicações: `https://console.developers.google.com/apis/credentials`
   - *Authorization endpoint*: `https://accounts.google.com/o/oauth2/v2/auth`
   - *Token endpoint*: `https://oauth2.googleapis.com/token`
   - *UserInfo endpoint*: `https://openidconnect.googleapis.com/v1/userinfo`

   $\Longrightarrow$ No documento de entrega do trabalho deve explicar o modelo de política que usou e as diferentes políticas que testou.

5 de novembro de 2022 (editado a 25 de novembro)

Referências

[1] Google Tasks API. https://developers.google.com/tasks/reference/rest

[2] Google OpenID Connect. https://developers.google.com/identity/protocols/oauth2/openid-connect

[3] Especificação *core* do OpenID Connect. https://openid.net/specs/openid-connect-core-1_0.html

[4] Casbin. https://casbin.org/docs/en/overview

[5] Criação de credenciais para as aplicações cliente. https://console.cloud.google.com/apis/credentials