Instituto Superior de Engenharia de Lisboa
LEIC, LEIRT, LEIM
Segurança Informática
Segundo Trabalho Prático, Semestre de Inverno de 23/24
Entrega no Moodle até 13 de dezembro de 2023

Entregar:

- *documento PDF com identificação do grupo, respostas às perguntas da parte 1, sumário das configurações realizadas na questão 6 e descrição da política de controlo de acessos da questão 7.*
- *ficheiros em separado com código fonte das questões 6 e 7.*

Parte 1

1. Considere o protocolo TLS:
   (a) Porque motivo a propriedade *perfect forward secrecy* não é garantida usando o processo base com RSA para estabelecimento do `master_secret`?
   (b) Identifique dois possíveis ataques ao *record protocol* e explique as técnicas usadas para os prevenir.
2. Considere uma aplicação *web* que usa como informação de validação das *passwords* a forma $h_u = H(pwd_u||salt_u)$, sendo H uma função de *hash*, pwd_u a *password* do utilizador u e $salt_u$ um número aleatório gerado no momento do registo do utilizador u ($||$ representa a concatenação de bits).
   O uso da técnica conhecida como CAPTCHA (*Completely Automated Public Turing Test to Tell Computers and Humans Apart*) contribui para mitigar ataques de dicionário à informação de validação? Explique.
3. Considere uma aplicação *web* que mantém estado de autenticação entre o *browser* e a aplicação servidor usando *cookies*. No *cookie* é guardado um JSON web token (JWT) com o identificador do utilizador. Como é que a aplicação servidor pode detetar se o conteúdo do *cookie* foi adulterado no *browser*?
4. Considere a norma OAuth 2.0 e OpenID Connect no fluxo *authorization code grant*:
   (a) Em que situação está previsto o uso da estrutura JWT e com que objetivo?
   (b) No OAuth 2.0, após o dono de recursos ter autorizado e consentido o uso de um recurso, descreva as ações da aplicação cliente para conseguir fazer os pedidos ao servidor de recursos.
5. Considere o modelo de controlo de acessos $RBAC_1$. Para realizar o controlo de acessos aos recursos foi definida a seguinte política $RBAC_1$ que inclui os papéis *(M)ember*, *(D)eveloper*, *(T)ester* e *(S)upervisor*.
   - $U = \{u_1, u_2, u_3, u_4\}$
   - $RH = \{M \preceq T, M \preceq D, D \preceq S, T \preceq S, T \preceq T_2, D \preceq D_2\}$
   - $UA = \{(u_1, M), (u_2, T_2), (u_3, D_2), (u_4, S)\}$
   - $PA = \{(M, p1), (D, p2), (T, p3), (D_2, p5), (T_2, p4)\}$

   Justifique qual o conjunto total de permissões que podem existir numa sessão com o utilizador u_4.

Parte 2

6. Configure um servidor HTTPS, **sem** e **com** autenticação de cliente. Tenha por base o ficheiro do servidor no repositório github da disciplina (`HTTPS-server\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 `http-server-base.js`, o servidor é colocado em execução com o comando:

  `node http-server-base.js`

- As chaves e certificados no servidor node.js são configuradas usando o formato PEM. 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` tem a chave privada do servidor e não está protegida por *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 criar tarefas de um utilizador Google através da *Google Tasks API* [1] usando milestones de projetos GitHub [2]. É opcional, mas valorizado, o acesso a projetos GitHub privados.

   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 [3, 4]. 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 a partir de milestones GitHub. Por simplificação do exercício, o papel `admin` não tem funcionalidades específicas atribuídas, mas herda todas as outras. Deve ser visível na interface da aplicação o papel ativo.
   - A política de controlo de acessos é aplicada usando a biblioteca *Casbin* [5]. 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/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 e GitHub para realizar os pedidos aos serviços. Os mesmos têm de ser feitos através de pedidos HTTP construídos pela aplicação *web*, como demonstrado durante as aulas.

   Considere os *endpoints* de registo e autorização de aplicações nos serviços Google e os endpoints equivalentes do GitHub [7]:

   - 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 a política de controlo de acessos.

12 de novembro de 2023

Referências

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

[2] Github Milestones API. https://docs.github.com/en/free-pro-team@latest/rest/reference/issues#milestones

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

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

[5] Biblioteca de autorização Casbin. https://casbin.org/docs/en/overview

[6] Criação de credenciais na Google para as aplicações cliente.

[7] Documentação sobre OAuth2 no GitHub https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps#web-application-flow