Instituto Superior de Engenharia de Lisboa
LEIC, LEIRT, LEIM
Segurança Informática
Segunda série de exercícios, Semestre de Inverno de 20/21
Entregar até 29 de dezembro de 2020

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

   1.1. Quais os mecanismos que o sub-protocolo *handshake* tem para evitar ataques de *replay* (reposição de mensagens de *handshakes* anteriores)?

   1.2. De que forma um ataque a uma autoridade de certificação torna possível um ataque de *man-in-the-middle*?

2. No contexto da autenticação baseada em *passwords*, um ataque de dicionário à interface de autenticação pode ser dificultado usando um *salt* com mais bits na geração da informação de validação?

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.

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

   3.2. Que alterações propõe para evitar o ataque anterior?

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

   4.1. O valor indicado no *scope* é escolhido pelo cliente ou pelo dono de recursos?

   4.2. Em que situações o cliente e o servidor de autorização comunicam indiretamente através do *browser* do dono de recursos?

   4.3. Qual a diferença entre o *access_ token* e o *id_ token*?

5. Pretende-se configurar e testar um servidor web com HTTPS, com e sem autenticação de cliente, usando o *browser* e uma aplicação Java. Considere o certificado e chave privada do servidor `www.secure-server.edu` em anexo, o qual foi emitido pela `CA1-int` da primeira série.

   a) Configure e teste o servidor usando o cliente *browser*, com e sem autenticação de cliente. Tenha por base o ficheiro do servidor em anexo (`server.js`);

   b) Realize a aplicação cliente Java;

   c) Teste o servidor usando o cliente Java, com e sem autenticação de cliente.

   Note que:

   - 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. Pode fazer download da versão mais adequada aqui: `https://nodejs.org/en/download/`. Após a configuração mínima na secção `options` do ficheiro `server.js`, o servidor é colocado em execução com o comando:

     ```
     node server.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 do 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 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)`.

6. Realize uma aplicação *web* para adicionar eventos ao calendário Google do utilizador a partir de datas de *milestones* presentes num repositórios GitHub.

   Requisitos funcionais:

   - Após autenticação do utilizador na aplicação, são apresentados os *milestones* do GitHub [1] para um determinado projecto no qual o utilizador tem acesso;
   - Os utilizadores autenticados podem criar *tasks* Google [2] a partir das datas presentes nos *milestones* GitHub.

   Requisitos não funcionais:

   - A aplicação só pode ser usada por utilizadores autenticados, exceto a rota de autenticação. Os utilizadores são autenticados através do fornecedor de identidade social Google, usando o protocolo OpenID Connect [3, 4];
   - O estado de autenticação entre o *browser* e a aplicação web é mantido através de *cookies*;
   - A aplicação tem de estar preparada para ser usada em simultâneo por utilizadores diferentes;
   - Os dados guardados para cada utilizador podem estar apenas em memória;
   - É valorizado o acesso a repositórios privados do Github.

   Neste exercício não pode usar os SDK da Google/GitHub 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 Github:

   - Endpoints Google
     - 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`
   - Endpoints Github
     - Registo de aplicações: `https://docs.github.com/en/free-pro-team@latest/developers/apps/creating-an-oauth-app`
     - *Authorization endpoint*: `https://github.com/login/oauth/authorize`
     - *Token endpoint*: `https://github.com/login/oauth/access_token`

14 de novembro de 2020

Referências

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

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

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

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