Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Segurança Informática

Segunda série de exercícios, Semestre de Inverno de 17/18

Data limite de entrega: 24 de novembro de 2017

---

1. No contexto do protocolo TLS:

   1.1. O protocolo *handshake* pressupõe a existência de um canal seguro ou inseguro? Como é garantida a autenticidade das mensagens nesse canal?

   1.2. Considere que o cliente C e o servidor S usam TLS apenas com autenticação de servidor. Explique que autoridade de certificação teria de ser comprometida para que fosse possível realizar um ataque de *man-in-the-middle*.

   1.3. Atualmente os *browsers* classificam como obsoleto/inseguro o uso de chaves públicas e privadas para troca do *master secret* entre cliente e servidor. Qual a justicação?

   1.4. Considere que um atacante quer conhecer a estrutura dos *cookies* usados por determinada aplicação *web*. A utilização de HTTPS em todas as comunicações dificulta esta ação?

2. Num sistema com autenticação baseada em *passwords* foi usada a seguinte abordagem para guardar a informação de validação de cada utilizador: $H(pwd_u|H^{1000}(pwd_u))$, sendo H uma função de *hash* aplicada repetidamente 1000 vezes, pwd_u a *password* do utilizador u e $'|'$ a concatenação de *bytes*. Comente sobre a resistência a ataques de dicionário.

3. No contexto da *framework* OAuth 2.0, porque motivo o `client_secret` só é usado em comunicações directas entre o *client* e o *authorization server*?

4. No contexto do fluxo *authorization code grant* da norma OpenID Connect.

   4.1. Após o utilizador introduzir as credenciais no formulário do servidor de autenticação, como é entregue o resultado à aplicação web cliente (*relying party*)?

   4.2. Em que situação não é obrigatório o *relying party* validar a assinatura do *id token* [1]?

5. Usando as classes da biblioteca Java para *sockets* [2], e a extensão para SSL/TLS [3], realize duas aplicações, cliente e servidor, com autenticação mútua. Os certificados aceites pelo servidor são os emitidos pela CA1. O certficado e chave do servidor estão em anexo ao enunciado. O servidor recebe pedidos HTTP GET e responde com o *echo* desses pedidos no corpo da resposta. Segundo o RFC 7230 [4] os pedidos GET têm o seguinte formato:

   ```
   GET SP request-target SP HTTP-version CRLF
   *( header-field CRLF )
   CRLF
   ```

   Sendo `SP` um espaço, `request-target` o URI do recurso e `CRLF` a sequência `\r\n`.

   5.1. Teste a aplicação anterior com um *browser*, usando os certificados e chaves privadas de `Alice_1` e `Bob_1`.

6. Realize uma aplicação *Web* com as seguintes funcionalidades:

   - Os utilizadores são autenticados através do fornecedor de identidade social Google, usando o protocolo OpenID Connect [5].
   - Os utilizadores autenticados podem consultar *milestones* de um repositório GitHub [6].
   - Os utilizadores autenticados podem criar *all-day events* no seu Google Calendar [7] a partir de *milestones* do GitHub.

   Valorizam-se soluções que permitam o acesso a repositórios privados.

Referências

[1] http://openid.net/specs/openid-connect-core-1_0.html#IDTokenValidation, visitado a 3 de novembro de 2017.

[2] https://docs.oracle.com/javase/tutorial/networking/sockets/index.html, visitado a 17 de outubro de 2017.

[3] http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html, visitado a 17 de outubro de 2017.

[4] https://tools.ietf.org/html/rfc7230, visitado a 17 de outubro de 2017.

[5] https://developers.google.com/identity/protocols/OpenIDConnect, visitado a 17 de outubro de 2017.

[6] https://developer.github.com/v3/issues/milestones/, visitado a 17 de outubro de 2017.

[7] https://developers.google.com/google-apps/calendar/v3/reference/, visitado a 17 de outubro de 2017.

ISEL, 19 de outubro de 2017.
(editado a 3 de novembro de 2017)