

**2ª Série de Exercícios**

**Licenciatura em Engenharia Informática e de Computadores**

## Segurança Informática

# *Prof. Eng. José Manuel Simão*

# *Grupo 7:*

*Hugo Fora 42121*

*Luana Silva 42189*

*Rui Lima 42200*

*Bruno Lourenço 42400*

*Lisboa, 24 de novembro de 2017*

1. 1. O protocolo *handshake* pressupõe a existência de um canal inseguro logo a autenticidade das mensagens tem de ser garantida através da geração do *Message Authentication Code* *(MAC)* das mensagens anteriores com o valor aleatório, tanto da parte do servidor como do cliente, e caso a igualdade dos *MACs* seja verificada prova-se que o servidor tem a chave privada associada à chave pública confirmando assim que as mensagens são autênticas.
   2. A autoridade a ser comprometida é a autoridade que produz os certificados raiz em que o cliente confia, pois se o atacante conseguir que a autoridade emita um certificado válido para o atacante usar então o cliente ao ver esse certificado como confia na autoridade iria estar a comunicar com o atacante e não com a autoridade.
   3. O uso de chaves públicas e privadas para a troca do *master secret* entre cliente e servidor é considerado obsoleto/inseguro pelos browsers, pois se o atacante descobrir a chave privada do servidor consegue alcançar os *master secrets* usados nas ligações anteriores e consequentemente decifrar as mensagens destas comunicações.
   4. A utilização de HTTPS em todas as comunicações não dificulta o conhecimento da estrutura das *cookies*, uma vez que estas estão guardadas no browser e qualquer utilizador com acesso à máquina que corre o browser consegue ver a estrutura destas.
2. Se não for usado o fator aleatório *salt* então com um ataque de dicionário pode ser descoberta a *password* mais facilmente pois os valores de *hash* podem ser calculados anteriormente.
3. O *client\_secret* só é usado em comunicacões directas entre o *client* e o *authorization server* pois só é necessário para autenticar o cliente (aplicação web) em relação ao *relying party*, para isso teve que haver um registo do cliente na *relying party* e a outra razão é para se evitar que o *client\_secret* seja passado pelo browser de forma a que ninguém tenha acesso a este.
4. 1. Após o utilizador introduzir as credenciais no formulário do servidor de autenticação e este for validado, o servidor de autenticação responde com um *302 redirect* para a App web com o código resultante da validação do utilizador.
   2. Não é obrigatório o *relying party* validar o *id\_token* quando este é directamente obtido do *identity provider*, sendo apenas usado pelo *relying party* uma vez que este confia no *identity provider* e sabe que o valor é válido.
5. Na primeira versão existem duas classes uma que representa a aplicação servidora e outra que representa o cliente. No servidor é criado um *SSL socket* com o seu certificado, a sua chave privada e o certificado raiz que é usado para validar os clientes. No cliente é criado um *SSL socket* com o seu certificado, a sua chave privada e o certificado raiz que é usado para validar o servidor. Uma vez estabelecida a ligação entre o servidor e o cliente e realizado o *handshake* com sucesso é iniciada a comunicação entre os mesmos. Esta consiste no cliente enviar um pedido *HTTP GET* e o servidor responder com o *ECHO* do pedido no corpo da resposta *HTTP.*
   1. Na segunda versão manteve-se o servidor, mas alterou-se o cliente que passou a ser o browser, para tal foi necessário instalar os certificados usados pelo servidor e pelo cliente, sendo esses no contexto desta aplicação o CA1 e o Bob\_1 que poderia ser substituído pelo Alice\_1. Observa-se que para o *browser* *Microsoft Edge* e para o *Curl* (Cliente HTTP) é obtido o resultado esperado, todavia ao utilizar o *Google Chrome* como cliente verifica-se que são realizadas diversas tentativas de conexão com o servidor que falham devido ao *handshake*, no entanto passado algumas tentativas é obtida uma conexão com sucesso.