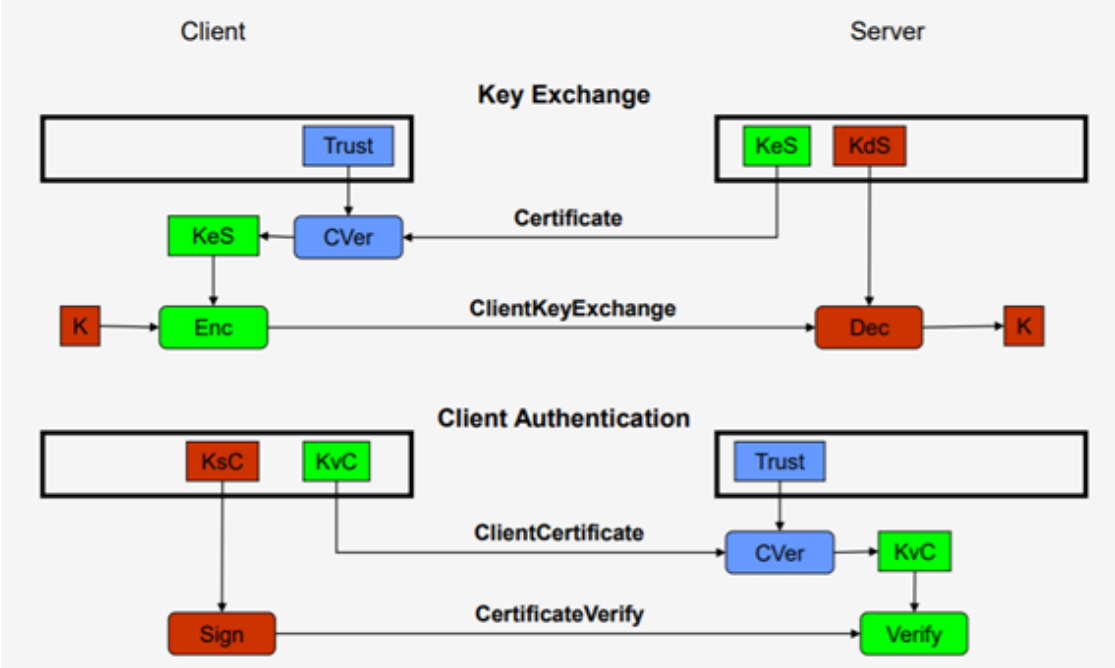
Segurança Informática

Licenciatura em Engenharia Informática, Redes e Telecomunicações

1ª Série de Exercícios



Grupo 4

43874 João Florentino

46435 Mihail Ababii

46919 Bárbara Castro

1º Semestre - Ano letivo 2020/2021

29 de dezembro de 2020

Resumo

Este trabalho incide sobre o funcionamento do protocolo TLS, nomeadamente na forma como este gere a confidencialidade e autenticidade das mensagens e autenticação do servidor e do cliente; o funcionamento do protocolo *handshake* e das mensagens que são trocadas para autenticação do servidor e estabelecimento de chaves simétricas; gestão de identidade e controlo de acessos, nomeadamente a autenticação baseada em *passwords* e ataques de dicionário; aplicações Web; protocolo HTTPS e autenticação no mesmo; autenticadores na forma de *cookie*, o protocolo OpenID Connect e *framework* OAuth 2.0.

Palavras-chave: aplicação *web*, ataque, autenticação, autorização, chave simétrica, cliente, *client_id*, *client_secret*, confidencialidade, controlo de acesso, *cookies*, *framework*, Google, *handshake*, HTTPS, mensagem, OAuth 2.0, OpenID Connect, protocolo, *salt*, servidor, SSL, TLS.

Índice

Lista de Figuras

1. Introdução

Na realização deste trabalho, foi-nos proposto que estudássemos o protocolo TLS, OpenID Connect e a *framework* OAuth2.

Numa primeira parte foi-nos pedido que estudássemos quais os mecanismos que o subprotocolo handshake tem para evitar ataques de replay e de que forma um ataque a uma autoridade de certificação torna possível um ataque man-in-the-middle. De seguida foi-nos questionado como seria possível tornar mais difícil um ataque de dicionário utilizando um salt e como seria possível um atacante fazer-se passar pela vítima utilizando as suas cookies.

Numa terceira fase foi-nos proposto um conjunto de questões sobre o protocolo OpenID Connect e sobre a *framework* OAuth2.

Por último foi-nos pedido que desenvolvêssemos duas aplicações: a primeira seria de testes sobre um servidor web HTTPS em que teríamos de desenvolver um cliente browser que testasse a ligação com e sem autenticação e a segunda seria uma aplicação web que iria adicionar eventos ao calendário Google do utilizador autenticado através de datas de *milestones* presentes num repositório GitHub.

Ferramentas utilizadas para o desenvolvimento do trabalho

No desenvolvimento do trabalho aqui descrito, recorremos à utilização da linguagem de programação JAVA e JavaScript para realizar os programas que se encontram descritos nas alíneas 5 e 6.

2. Resolução dos problemas do trabalho

Problema 1:

Considere o protocolo TLS e as infraestruturas de chave pública:

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

O ataque de replay consiste em reenviar a mensagem de handshake com a mensagem alterada. Nas mensagens ClientHello e ServerHello é apresentada uma componente de aleatoriedade, e sendo esta componente diferente a cada handshake, significa que ao reutilizar a mesma mensagem de ClientHello ou de ServerHello as chaves não irão corresponder.

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

Como a validação do certificado de um dado servidor depende de uma raiz de confiança, podemos concluir que um ataque a uma autoridade de certificação torna possível um ataque de man-in-the-midle, isto porque, se este ataque for realizado com sucesso, o atacante pode emitir novos certificados a partir dos certificados de confiança emitidos pela autoridade de certificação. Estes novos certificados podem substituir certificados "originais" conseguindo assim, ver as mensagens sem quebrar o TLS.

Problema 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?

Um ataque de dicionário ou pesquisa exaustiva serve para testar todas as passwords possíveis, com base num ficheiro que contem passwords mais prováveis ou previamente usadas. Uma das formas de proteção contra este tipo de ataques passa por aumentar a incerteza da palavra pass. Uma forma de aumentar esta incerteza é através do mecanismo salt, que consiste em gerar um dado aleatório que serve para adicionar á password antes da geração do seu Hash, de forma a cria desordem na mesma. Por exemplo, para um salt de 4 bit irá existir 2^4= 16 combinações possíveis para cada password.

Problema 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.

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

Uma vez que o atacante sabe o identificador do utilizador e tem conhecimento relativo á construção da cookie, o atacante utilizando o identificador (u), pode assim gerar um hash H(u), e com este construir a cookie que contem o par (u, H(u)). Desta forma o atacante consegue assim fazer-se passar pela vítima.

Que alterações propõe para evitar o ataque anterior?

A alteração a efetuar seria a nível do hash, isto é, em vez de se usar esta função determinística, utilizaríamos assinatura digital ou mac de forma a verificar a integridade da cookie. Uma vez que estes esquemas são mais complexos, aumentaríamos a dificuldade de gerar uma cookie igual.

Problema 4:

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

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

O valor indicado no scope é escolhido pelo cliente, uma vez que este é a entidade que quer obter os dados do servidor de recursos, o dono de recursos é apenas o responsável pelos dados aos quais o cliente está a tentar aceder.

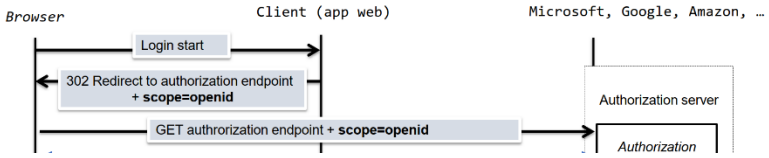


Figura 1 - Code flow

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

Existe comunicação indireta entre o cliente e o servidor de autorização passando pelo browser quando o cliente tenta redirecionar o pedido, para obter um authorization endpoint e durante a respetiva resposta do servidor de autorização para o cliente. Como não é possível conectar o cliente diretamente ao servidor de autorização, o pedido tem de passar por um browser, por forma a ser efetuado com sucesso.

Qual a diferença entre o access_token e o id_token?

O id_token serve apenas como identificador do cliente. Este é fornecido pelo authorization service que associa os dados do utilizador como nome, email a um id token, que é imediatamente consumido pelo cliente. O access_token é apenas o que é atribuído ao cliente, para que este possa ser autorizado a obter os dados requisitados ao servidor de recursos.

3. Conclusão

Na realização desta série de exercícios, estudámos o funcionamento do protocolo TSL, nomeadamente na forma como este gere a confidencialidade e autenticidade das mensagens e autenticação do servidor e do cliente. Numa segunda parte estudamos que o subprotocolo handshake tem um mecanismo de aleatoriedade que permite evitar ataques de replay e ainda que se um atacante conseguir atacar uma entidade de certificação, pode emitir certificados falsos que envia para o cliente. Desta forma o atacante consegue ouvir toda a comunicação entre o cliente e servidor. Este ataque é chamado man-in-the-middle.

Numa terceira fase concluímos que o scope em OpenID Connect e na framework OAuth2 é enviado pelo cliente e conseguimos perceber melhor qual a diferença entre Id_token e Access_token. O primeiro apenas serve como identificador de sessão enquanto que o segundo serve como "chave" para o cliente conseguir obter dados do servidor de recursos.

Por último foram desenvolvidas duas aplicações utilizando as linguagens de programação JAVA e JavaScript. A primeira apenas serve para testar como é feita a conexão em HTTPS através de certificados e a segunda serve para criar eventos no calendário da Google do utilizador autenticado através das datas de *milestones* presentes num repositório GitHub.

Com o desenvolvimento da primeira aplicação aprofundamos melhor o nosso conhecimento sobre certificados de confiança e como estes funcionam e com a segunda aplicação percebemos melhor como acontece o *flow* de um pedido em OAuth 2.0.

Conseguimos ainda aprofundar o nosso conhecimento nas diversas linguagens de programação usadas, neste caso em particular, JAVA e JavaScript.

4. Referências:

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