Relatório S2

Docente: José Simão

- LI51_52D – LEIRT51D

- Grupo 6:

Ana Gaspar: 43877
André Martins: 43562
José Pedro Rodrigues: 43596

Síntese: Este segundo trabalho prático teve como objetivo consolidar a matéria no domínio do protocolo TLS, a autenticação baseada em passwords e do protocolo HTTP.

Índice

Introdução

Neste segundo trabalho prático abordam-se os protocolos TLS e HTTP estando estes ligados uma vez que o primeiro protocolo é um protocolo de segurança que protege as telecomunicações via internet para serviços como o SMTP, HTTPS, ou seja, o protocolo TLS encapsula os protocolos de aplicação como o HTTP. Para além disso, também se aborda a autenticação baseada em *passwords*.

Como o primeiro trabalho prático, este também está divido em duas partes sendo que a primeira parte testa os conhecimentos teóricos das matérias e a segunda parte testa a componente prática, sendo realizada com a ajuda de *Java Cryptography Architecture*, porém, no último exercício recorre-se também à linguagem *javascript* uma vez que é necessário realizar uma aplicação *web*.

Resolução

Pergunta 1

Pergunta 1.1

Na autenticação entre cliente e servidor, o material criptográfico que têm de ser configurados no cliente é a autenticação dos endpoints, as chaves e outros parâmetros estabelecidos.

Para além disso, o cliente apesar de ter uma chave privada tem de ter também instalado o certificado do servidor.

Pergunta 1.2

No *handshake* do TLS é usado o esquema simétrico MAC uma vez que permite validar que o servidor é o dono do certificado pois tem a chave privada do mesmo, garantindo que a chave foi trocada com sucesso.

A sua utilização tem como objetivos lidar com a criação e gerir ligações seguras (estabelecimento seguro dos parâmetros de protocolos criptográficos), ou seja, garante que uma mensagem que esteja a ser transmitida entre o cliente e o servidor não seja alterada, mantendo, assim, a sua autenticidade.

Pergunta 1.3

A característica que torna o *record protocol* suscetível a ataques baseados em *Vaudenay* é o uso de *padding* no modelo de operação em bloco.

Como o *record protocol* autêntica e a seguir é que encripta isso pode causar possíveis ataques do *Vaudenay*, porém se acontecesse o contrário, ou seja, se encriptasse e depois autenticasse não seria possível o ataque, visto que o atacante estaria a alterar os dados e daria sempre falso na verificação.

Pergunta 2

Pergunta 2.1

O *OAuth* fornece um acesso seguro aos recursos do servidor em nome do proprietário dos mesmos.

É especificado um processo para proprietários de recursos no intuito de autorizar o acesso de terceiros aos recursos de servidor sem que com isto sejam partilhadas as suas credenciais.

O cliente/*relying party* irá pedir uma autorização com *openID scope*[1] ao *Resource Owner* para aceder ao conjunto de recursos. No caso deste aceitar, o "pedinte" recebe uma garantia de autorização e na resposta virá um *ID token* (contém as *claims* sobre a identidade do *Resource Owner*, podendo ser, estes, visualizados através do *JSON Web Token*).

De seguida, o cliente/*relying party* irá fazer um pedido ao servidor de autorização enviando então o código de garantia, de forma a obter o *acess token.* Se a garantia de autorização for válida, o *Authorization Server* gera um *acess token* e envia ao cliente, com este *acess token*, caso contrário, irá produzir um erro.

Por fim, o cliente/*relying party* fica apto para pedir recursos protegidos pelo *Resource Server*, se este *acess token* for realmente válido o servidor cede o recurso solicitado.

[1] – Scope indica o conjunto de permissões a que o cliente/*relying party* terá acesso.

Pergunta 2.2

O uso do protocolo *OAuth* 2.0 irá fornecer vulnerabilidade a sistema uma vez que deixará que o utilizador consiga aceder a uma página cópia da página original, porém essa página cópia foi criada pelo atacante com o objetivo de conseguir aceder aos dados pessoais do utilizador. Assim, este protocolo irá fornecer algumas limitações.

Em primeiro lugar, o protocolo pode adicionar excessivas extensões no *spec* (especificações) de modo que irá criar implementações incompatíveis entre elas. Como sequência, leva igualmente a que o código tenha de ser diferente para cada uma das implementações.

Em segundo lugar, cada aplicação que use este tipo de autenticação ganha um acesso amplo aos recursos protegidos do utilizador sem que este possa restringir a duração desse mesmo acesso ou limitá-lo apenas para alguns recursos (toda a informação ficará disponível).

Por fim, a utilização do SSL/TLS enquanto os *tokens* são enviados sem qualquer encriptação poderá influenciar alguma falha de segurança porque poderá ser descoberta por um atacante por diversas formas e, assim, o atacante poderá aceder a informações pessoais.

Pergunta 3

Pergunta 3.1

O *OpenID Connect* tem uma forma fácil de consumir *tokens* de identidade, os clientes recebem uma identidade do utilizador codificada em JSON seguro, sendo isto chamado de *ID token*.

Este *token* é apreciado pela sua elegância e portabilidade, tendo suporte para vários algoritmos de assinatura e criptografia.

O *ID token* é obtido através de um fluxo padrão do protocolo OAuth2.0 e lembra um pouco o conceito de identidade, assinado pelo *OpenID Provider*.

Tendo, assim, as seguintes utilidades:

1) Afirma a identidade do utilizador referenciado em OpenID.
2) Especifica quando e como, em termos de força de ligação, o utilizador foi autenticado.
3) Pode incluir detalhes adicionais solicitados sobre um determinado assunto, como nome, mail, etc...
4) É assinado digitalmente pelo que pode ser verificado pelos destinatários pretendidos.
5) Pode ser criptografado para confidencialidade.
6) Pode ser passado como identidade para terceiros, ou seja, podemos enviá-lo para outros componentes de aplicação ou serviçoes de bak-end quando quando a identidade do utilizador é necessária.
7) Este token pode ser trocado por um token de acesso no "endpoint" de token de autorização do OAuth 2.0.
8) Se colocarmos num "cookie" do navegador o ID token, podemos implementar uma sessão sem estado leve (SESSÕES SEM ESTADO). Este comportamento elimina a necessidade de armazenar sessões no lado do servidor (na memória ou no disco), a sessão é verificada avaliando o token de identificação.

Em suma, o *ID token* tem como finalidade a proteção contra-ataques e transporte de informações sobre o utilizador e o *relying party* que requisitou os dados do utilizador.

Pergunta 3.2

Um *relying party* (RP) é um termo usado para referir uma entidade que depende de um provedor de identidade de terceiros para identificar e autenticar um utilizador que está a tentar aceder a conteúdos.

Neste caso, é a aplicação cliente que tem o papel *relying party* visto ser esta que irá requisitar a autenticação do utilizador de forma externa.

Pergunta 5

Resolução em anexo.

Pergunta 6

Tanto o salt e o número de iterações servem para dificultar os ataques do tipo 'dicionário'.

O *salt* é um dado aleatório que serve para adicionar à password antes da geração do seu hash, de forma a criar entropia aquando da tentativa de ataques. Por exemplo, para um *salt* de 64 bits irá existir 2^{64} combinações para cada password.

Já o número de iterações é a quantidade de vezes que aplicamos a função para a geração do hash da password (em cada iteração obtemos a anterior e calculamos o hash dela).

Resolução em anexo.

Pergunta 7

Resolução em anexo.

Conclusão

Este trabalho permitiu melhorar a compreensão dos protocolos HTTP, TLS/SSL e a autenticação baseada em *passwords*.