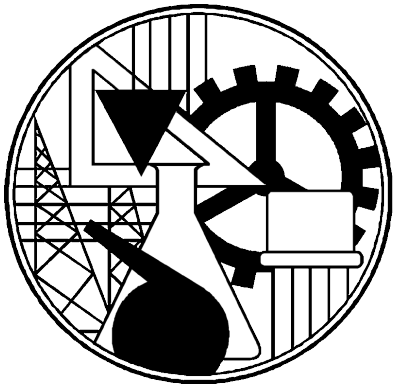

ISEL

Instituto Superior Engenharia de Lisboa

2ª Serie

Engenheiro: José Simão

Engenharia Informática e de Computadores

Semestre de Inverno 2011/2012

Ana Correia - 31831

Diogo Cardoso - 32466

João Silvestre - 32766

29/11/11

Índice

Parte Teórica 4

Exercício 1 4

Alínea 1 4

Alínea 2 4

Alínea 3 4

Alínea 4 5

Alínea 5 5

Exercício 2 5

Exercício 4 6

Parte Prática 7

Exercício 5 7

Alínea 1 8

Exercício 6 9

Exercício 7 10

Parte Teórica

Exercício 1

Alínea 1

O problema tratado no artigo é a segurança das passwords dos utilizadores, nomeadamente a complexidade e o armazenamento destas, este apresenta uma solução possivelmente mais segura e fácil de utilizar.

Alínea 2

Um "*phishing attack*" é uma forma de atacantes obterem as informações pessoais dos utilizadores "fazendo-se passar" por uma terceira entidade, por exemplo bancos, sites de e-commerce, etc. Estes ataques têm diversas maneiras de ser realizados:

- Por *email*, onde o atacante se faz passar por um serviço de um utilizador e utiliza falsos URIs para redireccionar o utilizador para um web site semelhante ao serviço forjado, ai o atacante pede ao utilizador informação confidencial.
- Por *cross-site scripting* onde o atacante utiliza falhas de segurança dos próprios serviços para injectar scripts de forma a obter informações confidenciais do utilizador ou redireccioná-lo para um website forjado.

O serviço proposto pelo artigo resolve este tipo de ataques com a sua geração de password, uma vez que, este utiliza o próprio nome do *host* para gerar a hash que irá ser utilizada pelo utilizador como password.

Outras soluções passariam por criar uma blacklist com os URIs de todos os sites culpados de phishing, assim os browsers poderiam actualizar e verificar os sites que o utilizador tenta aceder de forma a confirmar se são seguros ou não. Autenticação por parte de ambos os intervenientes, cliente e servidor através de uma imagem estabelecida no registo, assim o cliente saberia que apenas deveria confiar as suas credenciais ao possuidor da imagem correcta. Ferramentas anti-phishing que verificam se os sites são fiáveis ou não.

Alínea 3

O uso da concatenação do *username* do utilizador com a *master password* serve para evitar ataques dicionário à *master password*, utilizando o *username* como *salt* um atacante não poderá calcular directamente um conjunto de *hashs* e compará-las simplesmente com o valor da *hash* final.

Alínea 4

A primeira fase representa o valor robusto e computacionalmente demorado de gerar da password, este é gerado através de múltiplas chamadas à função de *hash* e apenas uma única vez para cada máquina. A segunda fase é a geração da password a utilizar em si, esta deve ser rápida de calcular para que seja viável, uma vez que um utilizador não quer esperar muito tempo para que a sua password seja inserida. Concluindo a primeira fase representa a segurança e robustez do método documentado enquanto a segunda além de ressegurar a segurança ainda dispõe de uma interface de utilizador rápida.

Alínea 5

Existem no total quatro tipos de ataques considerados, estes ataques consistem em ataques dicionários com diferente tipo de informação sobre o utilizador. A escolha destes ataques foi realizada com base no tipo de informação utilizado pelo esquema de criação de passwords sendo portanto o ponto provável que um atacante irá utilizar para tentar descobrir a password.

- Ataque sem informação. O atacante terá de atacar directamente um web site para descobrir a palavra-chave do utilizador. Com o esquema apresentado no artigo, este tipo de ataque é bastante limitado uma vez que mesmo que o atacante descubra a palavra-chave do website terá ainda de tentar atacar para descobrir a *master password*. Além disso grande parte das infra-estruturas onde são produzidos os web sites estão protegidas contra tentativas sucessivas de autenticação, limitando a priori um ataque sem qualquer tipo de informação.
- Ataque com a password do site. Este tipo de ataque centra-se na utilização da password de um site como forma de obter a *master password*. Este ataque torna-se custoso para um atacante uma vez que este para gerar o dicionário necessitaria de gastar pelo menos 100 segundos por cada elemento, tornando este ataque demorado.
- Ataque com o valor intermédio, gerado na primeira fase. Este ataque é semelhante ao anterior com a diferença na informação que o atacante tem. Apesar disso o processo de obtenção da *master password* mantêm-se, tornando o ataque igualmente demorado.
- Ataque com o valor intermédio e uma password do website. Neste caso o atacante tem todo a informação necessária para efectuar um ataque eficiente, uma vez que a complexidade de achar a *master password* é a mesma que gerar uma nova password para um website tornando este ataque muito eficiente.

Exercício 2

O controlo de acessos mandatário contem *labels* de segurança para cada recurso(Objecto), classificação e categoria. Para que um *user* possa ler ou escrever necessita de conter a mesma classificação e categoria do recurso. Este controlador também não autoriza a que qualquer *user* modifique as *labels* de segurança de recursos. Por isso, nenhum atacante pode modificar as restrições dos recursos.

Por exemplo, o modelo *biba* contem um controlo de acessos mandatário, se este fosse atacado por um cavalo de trojan, este apenas permitiria que o programa escreve-se recursos do mesmo nível ou níveis abaixo. O *biba* não permite que *users* leiam informação de níveis inferiores para não sejam contaminados por objectos de confiança inferior.

Exercício 3

Utilizadores = { u0, u1, u2, u3 }

Permissões = { le0, ll0, le1, ll1, le2, ll2, le3, ll3 }

Roles = { r0, r1, r2, r3 }

RH = { }

PA = { (r0, le0), (r0, le1), (r0, le2) , (r0, le3) , (r0, ll0), (r1, le1), (r1, le2) , (r1, le3) , (r1, ll0), (r1, ll1), (r2, le2) , (r2, le3) , (r2, ll0), (r2, ll1), (r2, ll2), (r3, le3) , (r3, ll0), (r3, ll1) , (r3, ll2), (r3,ll3) }

UA = { (u0, r0), (u1, r1), (u2,r2), (u3,r3) }

le = escrever, ll = ler

Exercício 4

As *certification rules* têm como objectivo especificar a certificação de um objecto ou seja se este é válido, enquanto as *enforcement rules* têm como objectivo garantir a integridade dos objectos já certificados, isto é, as *certification rules* especificam a forma de garantir que uma CDI foi submetida ao procedimento de certificação, já as *enforcement rules* especificam como garantir a integridade das CDI certificadas nas transacções que puderam ocorrer.

Parte Prática

Exercício 5

Esta aplicação, realizada em ASP.NET MVC 3, irá mostrar contactos de um determinado utilizador do Google. Como input a aplicação oferece uma caixa de texto para que o utilizador indique o email a qual pretende ver os contactos, é **necessário** que este email seja o mesmo da conta que está autenticada no momento da inserção uma vez que este irá autorizar a visualização dos contactos.

A aplicação usa OAuth2 na comunicação com a API da Google para obter autorização de forma a aceder ao recurso, o processo de autorização segue o seguinte esquema:

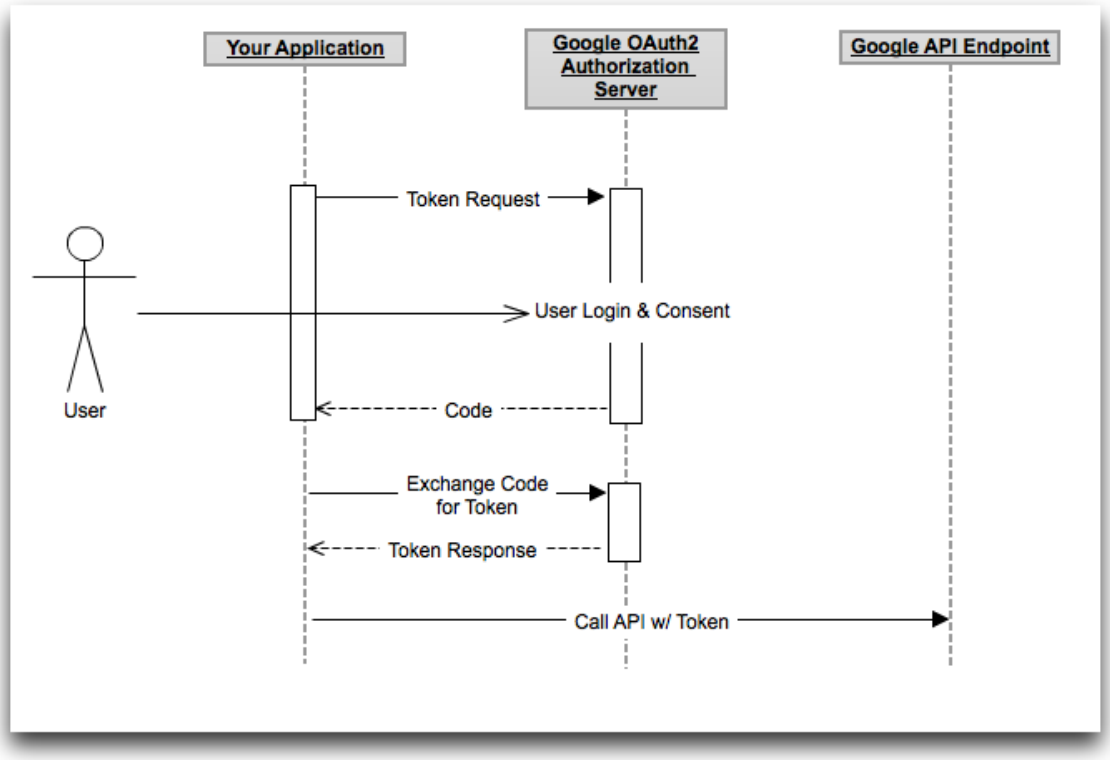

Figura 1 - Processo de autorização utilizando OAuth2.

Quando o utilizador submete o email para a aplicação esta irá redireccionar o utilizador para uma página da Google para que este autorize o acesso da aplicação a informação do utilizador.

Se o utilizador der autorização a Google irá redireccionar de volta para a aplicação, para um URI que é indicado no redireccionamento realizado anteriormente pela aplicação. Com o redireccionamento um *Autorization Grant* irá ser enviado por parte da Google, este valor irá ser usado para obter um *Access Token*.

falta. Este irá ser obtido através de um novo pedido à Google, neste pedido é necessário indicar um redirect_uri (embora não seja usado para fazer redirect) que tem que ser o mesmo que o indicado no primeiro pedido.

Obtendo o Access Token podemos então fazer o pedido para obter os contactos, enviando também o Access Token, sendo que a resposta irá vir em XML bastando então fazer o parsing destes dados e mostrar ao utilizador.

Alínea 1

Para facilitar o processo foi trocado o servidor de testes pelo IIS Express. Começou-se então por configurar o servidor para que este aceite ligações *SSL* e para tal abre-se as propriedades do projecto e na opção "*SSL Enabled*" selecciona-se *true*.

Foi necessário instalar o certificado na máquina, garantindo que temos a chave primária deste, e adicionar o certificado *root* como *trusted*. Estando os certificados instalados foi necessário escolher o certificado que o IIS Express vai usar, para isso usamos os seguinte comandos:

```
netsh http delete sslcert ipport=0.0.0.0:<SSLPort>
```

Este comando irá apagar o certificado que se encontra de momento na porta *<SSLPort>*.

```
netsh http add sslcert ipport=0.0.0.0:44300 certstorename=MY certhash=<certificate hash> appid=<appid>
```

Este comando irá adicionar um novo certificado a essa porta, o valor *appid* pode ser um qualquer identificador único (por ex, {214124cd-d05b-4309-9af9-9cac44b2b74a}).

Por fim, foi necessário acrescentar o *callback* para *https* no painel de controlo das APIs do Google e alterar a aplicação de forma a utilizar este novo *callback* sendo assim possível que a aplicação comunicasse sobre um canal seguro.

Exercício 6

No desenvolvimento deste componente .NET foi definido um modelo interno para representar os diversos intermediários:

- User, implementa interface IPrincipal
- Role
- Permission, implementa interface IPermission

A classe PolicyDecisionPoint é então onde tudo acontece, esta tem dois construtores, um sem parâmetros e outro que recebe 6 enumeráveis de string que representam a seguinte informação:

- Users
- Roles
- Permissions
- Roles Heritage
- User Assignment
- Permission Assignment

No construtor sem parâmetros esta irá carregar a informação presente no ficheiro de configuração da aplicação, secção PDPPolicy, no construtor com parâmetros ele irá obter a informação que precisa dos enumeráveis de string.

Para interagir com o *PolicyDecisionPoint* existe o método *HasPermission*, que recebe um IPrincipal e as permissões requeridas e indica se este tem as permissões necessárias. Para saber quais as permissões activas é feito, por cada role associado ao User, uma pergunta ao principal se este tem o role e são apenas adicionados para os roles que estiverem activos de momento, se não existir nenhum role activo são então adicionados todos. Depois de se saber quais as permissões activas ir-se-á então ver se existem suficientes para aceder ao recurso.

Exercício 7

Neste exercício foi escolhida a opção 2, uma aplicação em MVC, foi criado um filtro de autorização para validar se o utilizador tem permissões, este filtro tem como parâmetro do atributo um array de strings com o nome das permissões. A cada pedido se o utilizador estiver autenticado ele irá fazer *demand* de cada uma das permissões, se alguma falhar irá lançar uma *SecurityException* que não é capturada para que o MVC saiba que deve negar o acesso ao recurso, caso o utilizador não esteja autenticado também é lançada uma *SecurityException*.