**I**nstituto **S**uperior de **E**ngenharia de **L**isboa

Licenciatura em Engenharia Informática e de Computadores

Semestre de Inverno 2016/2017

**Segurança Informática**

**Terceira série**

LI51N

**Trabalho elaborado pelo Grupo 1:**

Sara Sobral N.º 40602

Eduardo António N.º 40686

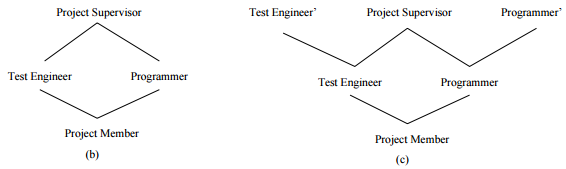
1. **No contexto do OpenID Connect, e usando o implicit flow [1], porque motivo é essencial que o cliente valide a assinatura do ID Token?**

Ao usar o Implícit flow, o cliente deve validar a resposta da seguinte forma:

1. Verifique se a resposta está em conformidade com a Secção 5 de [OAuth.Responses].
2. Seguir as regras de validação de RFC 6749.
3. Seguir as regras de validação do Token ID.
   * Ao usar o implícit flow, o conteúdo do Token de Identificação DEVE ser validado da mesma maneira que para o Fluxo de Código de Autorização.
     1. O cliente deve validar a assinatura do Token ID de acordo com o JWS usando o algoritmo especificado no Alg Header Parameter do JOSE Header.
     2. O valor da reivindicação nonce deve ser verificado para verificar se é o mesmo valor que foi enviado no Pedido de Autenticação. O cliente deve verificar um valor para os ataques de repetição. O método preciso para detetar ataques de repetição é específico do cliente.
4. Seguir as regras de validação do Access Token, a menos que o response\_typevalue usado seja id\_token.
5. **Considere o modelo RBAC1.**
   1. **É possível que a adição de um utilizador a um role diminua as suas permissões?**

Depende do role. Se o role for um role júnior em relação ao já adquirido pelo utilizador, então sim.

* 1. **Apesar de não haver permissões negativas, que mecanismo do modelo pode ser usado para limitar a associação entre roles e permissões? Dê um exemplo.**



Fonte: <http://csrc.nist.gov/rbac/sandhu96.pdf>

Às vezes, é útil nas hierarquias limitar o scope da herança.

No exemplo da hierarquia da figura (b) a função de supervisor de projeto é superior às funções de engenheiro de teste e programador. Supondo que os engenheiros de teste desejam manter algumas permissões privadas para seu papel e impedir a sua herança na hierarquia dos supervisores de projetos.

Pode existir por razões legítimas o acesso a trabalhos incompletos em curso pode não ser apropriado para o papel principal, enquanto o RBAC pode ser útil para permitir esse acesso a engenheiros de teste.

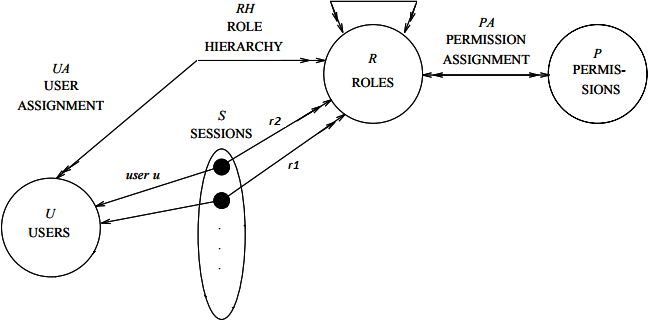
Esta situação pode ser acomodada definindo-se um novo engenheiro de teste de função’ e relacionando-o com o engenheiro de teste, como mostrado na figura (c). As permissões privadas dos engenheiros de teste podem ser atribuídas ao papel de engenheiro de teste’.

Os engenheiros de teste são atribuídos ao engenheiro de teste de função’ e herdam permissões da função de engenheiro de teste, que também são herdados para cima na hierarquia pela função de supervisor de projeto. As permissões do engenheiro de teste’, no entanto, não são herdadas pela função de supervisor de projeto.

Chamamos papéis como engenheiro de teste’ como papéis particulares.

A figura (c) também mostra um programador de função privada’. Em alguns sistemas, o efeito dos papéis privados é alcançado bloqueando a herança para cima de certas permissões. Neste caso, a hierarquia não representa a distribuição de permissão com precisão. É preferível introduzir papéis privados e manter intacto o significado da relação hierárquica entre papéis.

* 1. **Seja u um utilizador, p uma permissão e dois roles, r1 e r2 http://www.somatematica.com.br/figuras/simbolos/pertence.gif R, tais que r2 >= r1. Se (u, r2) http://www.somatematica.com.br/figuras/simbolos/pertence.gif UA e (p, r1) http://www.somatematica.com.br/figuras/simbolos/pertence.gif PA, isso significa que qualquer sessão s com user(s) = u tem acesso à permissão p?**



Pelo gráfico apresentado acima, sim.

1. **O que é que um atacante consegue obter ao realizar com sucesso um ataque de cross-site scripting? Forneça pelo menos um exemplo concreto.**

Um *script* de exploração de vulnerabilidade *cross-site* pode ser usado pelos atacantes para escapar aos controles de acesso que usam a política de mesma origem.

Através de um XSS, o atacante injeta códigos JavaScript num campo texto de uma página já existente e este JavaScript é apresentado para outros usuários, porque persiste na página.

Exemplo: o atacante insere num fórum de um *website* alvo de ataque, um texto que contenha um trecho de JavaScript que simula a página de *login* do *site*, os valores digitados são capturados e enviados para um *site* que os armazene.

1. **Quais as diferenças entre as soluções “Synchronizer (CSRF) Tokens" e “Double Submit Cookie" propostas pela OWASP [2] para evitar ataques de CSRF?**

Synchronizer (CSRF) Tokens:

* Qualquer operação de alteração de estado requer um token aleatório seguro (por exemplo, token CSRF) para impedir ataques CSRF.

Double Submit Cookie

* Se armazenar o token CSRF na sessão é problemático, uma defesa alternativa é o uso de um cookie de envio duplo. Um cookie de envio duplo é definido como o envio de um valor aleatório em um cookie e como um parâmetro de solicitação, com o servidor verificando se o valor do cookie e o valor da solicitação correspondem.

1. **Implemente um componente (e.g. biblioteca de classes ou módulo NPM) para realizar as funções de Policy Decision Point (PDP) com os seguintes requisitos:**

* **Suporte para o modelo RBAC1, em que os utilizadores, roles e permissões são definidos por strings.**
* **Utilização de ficheiros de configuração ou de outro repositório alternativo (e.g. base de dados) para o armazenamento da política.**
* **Suporte para realização de interrogações sobre a politica (e.g. permissões de um dado utilizador numa sessão).**

Package “serie3.java.connectionToSql”:

As classes Post/GetPermission, Post/GetRole e Post/GetUser utilizam as entidades para se relacionarem com a base de dados (“SegInf”);

Classe SQLConnection estabelece uma convecção à base de dados.

Package “serie3.java.entities”:

As entidades utilzadas: Permission, Role e User.

Package “serie3.sql”:

SQLQuery.sql cria a base de dados e as tabelas

SelectTables.sql mostra os dados de cada tabela

Ficheiro: “CreateDBandTables.bat” para criar a base de dados e as tabelas.

Exemplo utilizado:

U = { Alice, Bob, Charlie, Daniel }

R = { Project Member, Programmer, Test Engineer, Project Supervisor }

RH = { Project Supervisor ≤ Test Engineer, Project Supervisor ≤ Programmer,

Test Engineer ≤ Project Member, Programmer ≤ Project Me+mber }

Nota: O RH está representado na resposta 2.2 igura b).

P = { read, write, execute }

UA = { (Alice, Test Engineer), (Bob, Programmer), (Charlie, Project Supervisor), (Daniel, Project Member)}

PA = { (Test Engineer, execute), (Programmer, write), (Project Member, read) }

1. **Pretende-se desenvolver um Policy Enforcement Point (PEP) para testar o PDP realizado na alínea anterior. Use a componente de autenticação da aplicação web desenvolvida na alínea 7 da segunda série.  
   Acrescente à aplicação um conjunto de rotas cuja decisão de acesso é delegada no PDP, usando uma politica RBAC1 que inclua uma hierarquia de roles com dois ou mais níveis de herança.**

Package “serie3.java.connectionToSql”:

base de dados (“SegInf”).

1. **Elabore um documento, até 2500 palavras, com base num dos seguintes temas:**

* **Passwords gráficas - https://goo.gl/ccPi17, https://goo.gl/Gbx2p2;**
* **Ataques ao OAuth 2.0 - https://goo.gl/Msh20u;**
* **Especificação User-Managed Access (UMA) - https://goo.gl/pbRnPO;**
* **Vulnerabilidades de injeção de comandos em aplicações web - https://goo.gl/wQh45z;**
* **Código malicioso escondido em imagens - https://goo.gl/6AZvtz, https://goo.gl/Q21R5H.**

**Tendo por base o tema escolhido e o(s) artigo(s)/norma a ele associado, organize o documento em quatro secções, com a seguinte estrutura:**

1. **qual a motivação do assunto / o que se pretende resolver;**
2. **o que outros fizeram na área;**
3. **qual a metodologia usada;**
4. **quais os resultados;**

**Recomenda-se o uso do template IEEE [3].**

Código malicioso escondido em imagens

Sara Sobral

Segurança Informática

ISEL

Lisboa, Portugal

40602

Eduardo António

Segurança Informática

ISEL

Lisboa, Portugal

40686

16/01/2017

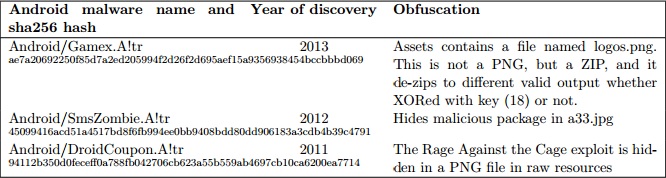
### qual a motivação do assunto / o que se pretende resolver

Nós vamos explicar como é possível embeber *bytecode* válido, indetetável e funcional, contido numa APK encriptado num ficheiro de imagem como, por exemplo, PNG ou JPEG que por sua vez essa imagem está contida numa APK que aparenta ser completamente legitima.

Este *exploit* foi desenvolvido pela Axelle Apvrille uma investigadora de *malware* para antivírus que trabalha para a Fortinet e pelo Ange Albertini um *reverse engineer*, que apresentaram a sua prova de conceito na conferencia de segurança europeia Black Hat em Amesterdão a 16 de outubro de 2014.

Este *exploit* dificulta reverter o código como, por exemplo, um programador de antivírus não consegue detetar o *malware* encriptado dado não ter a AES *key* correta.

### o que outros fizeram na área

Agora podemos ver alguns exemplos de como este *exploit* foi usado na tabela em baixo:[1]

No primeiro foi escondido um zip num ficheiro PNG, no segundo o *malware* é escondido numa imagem JPEG e por fim no terceiro é escondido um *rooting exploit* numa imagem PNG dentro das amostras do diretório *raw resource*.

Este *exploit* também poderá ser usado para esconder uma mensagem numa imagem que só quem tem a chave irá conseguir visualizar.

### qual a metodologia usada

A metodologia usada segue a seguinte hierarquia:

-Escrever uma APK(*Malware*);

-Encriptamos essa APK(*Malware*) usando *AngeCryption* para parecer um PNG normal. Inserimos num PNG genuíno um *chunk* extra (que é ignorado) onde é colocado o *malware*;

-Implementamos uma APK que vai conter o PNG com um *Dummy chunk* com o *Malware*.

A APK(*Malware*) contém:

*Payload APK*

AES−1(CRC32 + IHDR + IDAT + IEND)

CRC32 corresponde à soma dos *chunks* adicionados à imagem

IHDR corresponde ao *header chunk* da imagem

IDAT corresponde aos data chunks da imagem(genuina)

IEND corresponde a um determinado PNG

*Bytes* de *padding* para garantir que a APK tem um tamanho múltiplo de 16, para que se possa encriptar com AES apesar de se ter de usar *Cipher Block Chaning* (CBC) porque o AES tem limite de 16 bytes.

EOCD assinala o fim da APK

De seguida encriptamos esta APK que encripta para um PNG válido porque o IV foi selecionado para que o AES do início da APK fosse igual ao *header* mais o *garbage chunk* e porque a data anexada corresponde a data desencriptada.

E por fim colocamos o nosso PNG com *malware* nos *assets* de uma APK para ser enviada.

A APK que contém o *malware* é obtida ao decifrar o *bitmap* da imagem pois esta imagem decifrada contém o *malware*.

No *Android* será na mesmo pedido ao utilizador para instalar o *malware*, mas este pedido poderá ser escondido usando uma ferramenta como o DexClassLoader para esconder esse pedido.

### quais os resultados

Neste resumo explicamos como era possível esconder um *malware* numa aplicação *Android*, sendo este *malware* indetetável ao sistema mesmo realizando um *disassembly* à APK.

Também conseguimos ver que é possível manipular o *output* encriptado.

A equipa de segurança do *Android* resolveu o problema e ainda não foram detetados problemas relacionados desde a versão 4.4.2

### Bibliografia

- <https://0xicf.wordpress.com/tag/angecryption/>

- <https://www.blackhat.com/docs/eu-14/materials/eu-14-Apvrille-Hide-Android-Applications-In-Images-wp.pdf>

- <http://arstechnica.com/security/2016/12/millions-exposed-to-malvertising-that-hid-attack-code-in-banner-pixels/>

- <https://www.youtube.com/watch?v=hajOlvLhYJY>

- <https://www.quora.com/What-are-the-exact-mechanisms-flaws-exploited-by-the-rage-against-the-cage-and-z4root-Android-exploits>