Rapport projet
“Traçage de contacts”

Kévin MONVOISIN - Adrien NICOLLE

Année 2020 - 2021

I. But

1.1 But de l'application de traçage de contacts

Dans le contexte de la pandémie mondiale de COVID-19, des solutions ont été imaginées dans le but de freiner la propagation du virus.
L'application de traçage de contacts est l'une de ces nombreuses solutions.

Le but de ce type de solution sous forme d'application mobile est de prévenir les personnes ayant été en contact avec un utilisateur infecté par le virus.
Pour que cela fonctionne, les individus doivent posséder un smartphone avec une application de traçage installée, et doivent laisser (pour la plupart des applications) le protocole Bluetooth activé.

Les smartphones collectent des données utiles pour prendre connaissance des utilisateurs à proximité.
Données qu'ils stockent pendant un certain temps, et c'est grâce à ces dernières que l'on peut déterminer le contact avec un individu infecté.

Une personne certifiée malade, peut valider via son application de traçage qu'elle est infectée, et selon le protocole utilisé, l'application prévient les personnes contacts.

1.2 But de ce projet

L'objectif de ces applications est bon, cependant des questions éthiques, mais aussi liées à la sécurité de ces applications se posent.
TousAntiCovid, la réponse française à ces applications de traçage de contacts se considère sécurisée et anonyme. Mais l'est-elle vraiment ?

Le but de ce projet est donc dans un premier temps de déterminer si des failles dans la sécurité et la confidentialité existent dans ce genre d'applications, et en particulier dans l'application française *TousAntiCovid.*

Dans un deuxième temps, nous tenterons d'établir une liste de contre-mesures pour les failles citées, puis nous développerons un simulateur mettant en application ces contre-mesures.

II. Analyse des failles

En règle générale pour fonctionner, le smartphone de chaque utilisateur d'une application de traçage émet via Bluetooth à intervalle régulier un code généré aléatoirement (valide pendant un certain temps, par exemple 15 minutes), et cela représente le pseudonyme d'un utilisateur.

Le téléphone des utilisateurs aux alentours réceptionne ce pseudonyme, et le stock pendant 14 jours.

Ainsi, nous pourrions penser que le système est bien anonyme, puisque aucune information directement liée à l'utilisateur (Notamment le numéro de téléphone) n'est transmise. Uniquement des codes aléatoires.

Ce qui peut différer d'une application de traçage à une autre, est la manière de signaler à un utilisateur qu'il est cas-contact.

En effet, un protocole peut utiliser un des deux modèles de transmission de données suivants :

- Le modèle décentralisé
- Le modèle centralisé

Dans le modèle décentralisé, l'application ne transmet pas ses pseudonymes à une seule base de données, mais à plusieurs entités. Cela peut être via un principe de pair-à-pair (L'application contacte tous les autres utilisateurs) ou via des intermédiaires comme des hôpitaux.
Les protocoles d'Apple-Google, DP3T et PACT utilisent le modèle décentralisé.

Dans le modèle centralisé, l'application transmet ses pseudonymes à une seule autorité centrale, et donc cette entité regroupe la liste de tous les pseudonymes des utilisateurs de l'application
Le protocole ROBERT implémente ce modèle. C'est donc ce modèle que l'application TousAntiCovid utilise.

Premier problème, dans le modèle centralisé, les données ne sont en réalité pas anonymes, mais plutôt pseudonymes.
Une liste de pseudonymes infectés, rassemblée sous l'autorité d'une seule entité paraît risquée. Il faut pouvoir faire confiance à ce tiers.

Bien que les données ne soient pas nominatives, elles n'en sont pas plus anonymes.
Et cela car via un recoupement d'informations extérieures, il est potentiellement possible de remonter à l'identité de l'utilisateur.

Même en n'ayant pas accès directement à ces données, il est possible dans certains cas d'identifier un individu malade par le biais de l'application.

1er cas :
Une personne ne sortant que rarement, et croise un seul autre individu pendant sa sortie, peut immédiatement faire le lien si elle reçoit une notification l'identifiant comme cas contact.

2e cas :
De la même manière, pour une autre personne ayant été en contact avec d'autres personnes de divers cercles sociaux, il lui est possible de se renseigner auprès de ces groupes pour en identifier plus ou moins précisément le "coupable".

3e cas :
En laissant un téléphone "espion" dans un endroit spécifique, il est possible de déterminer qu'une personne malade était présente à cet endroit précis.
Et même parfois de connaître l'identité précise d'une personne infectée, dans une situation où un téléphone ne serait en contact qu'avec un seul individu.
Admettons qu'un candidat passe un entretien en seul à seul avec un recruteur, si un téléphone utilisé uniquement lors de cet entretien reçoit plus tard une notification, le recruteur pourra alors discriminer le candidat et ne pas le recruter.

En dehors du problème d'anonymat, d'autres soucis se posent en termes de sécurité.
Le fait d'utiliser le protocole Bluetooth pour transmettre les pseudonyme peut être une faille, car cela nécessite de laisser ce protocole toujours activé sur son téléphone pour que l'application puisse fonctionner.

Il y'a quelques années, une faille dans le protocole Bluetooth fut révélée, et cette dernière permettait entre autres de prendre le contrôle du smartphone.
Cette faille a été corrigée depuis, mais nous ne sommes pas à l'abri qu'une nouvelle brèche de sécurité voit le jour.

III. Possibles contre-mesures

L'utilisation d'une solution centralisée propose des avantages comme des inconvénients, il est à la fois responsable de la transmission des données, mais aussi pour le stockage de celles-ci. Ainsi, un serveur centralisé empêche les utilisateurs de communiquer entre eux directement. En revanche, il nécessite une implémentation empêchant quiconque ayant accès au serveur de pouvoir lire et utiliser ces données.
Comme dit plus haut, une solution centralisée ne propose pas l'anonymat, cette solution ne propose qu'une solution "pseudonyme".

Une contre mesure serait de générer ce pseudonyme en utilisant des algorithmes d'aléa définit comme cryptographiquement sûr; comme par exemple via la lecture de la mémoire de l'ordinateur en sélectionnant des bits à des positions aléatoires.
Un autre problème qui pourrait entrer en jeu, serait l'implémentation d'une attaque de type homme du milieu. Le protocole de communication doit être définie de telle sorte à ce que quiconque écoute le réseau ne doit pas être en mesure de comprendre ce qui est échangé.

Dans le cas où une application utiliserait une solution décentralisée, l'anonymat serait plus compliqué, car l'accès aux données serait plus "simple", chaque client serait considéré comme un nœud contenant ses propres données, ainsi que celles des personnes rencontrées. Un attaquant aurait alors le choix, il possède plusieurs cibles potentielles.
À contrario, l'accès aux données serait restreinte, c'est-à-dire qu'un attaquant n'aurait accès qu'aux données contenues sur le nœud qu'il attaque, à l'inverse d'une solution centralisée où il aurait accès à toutes les données.

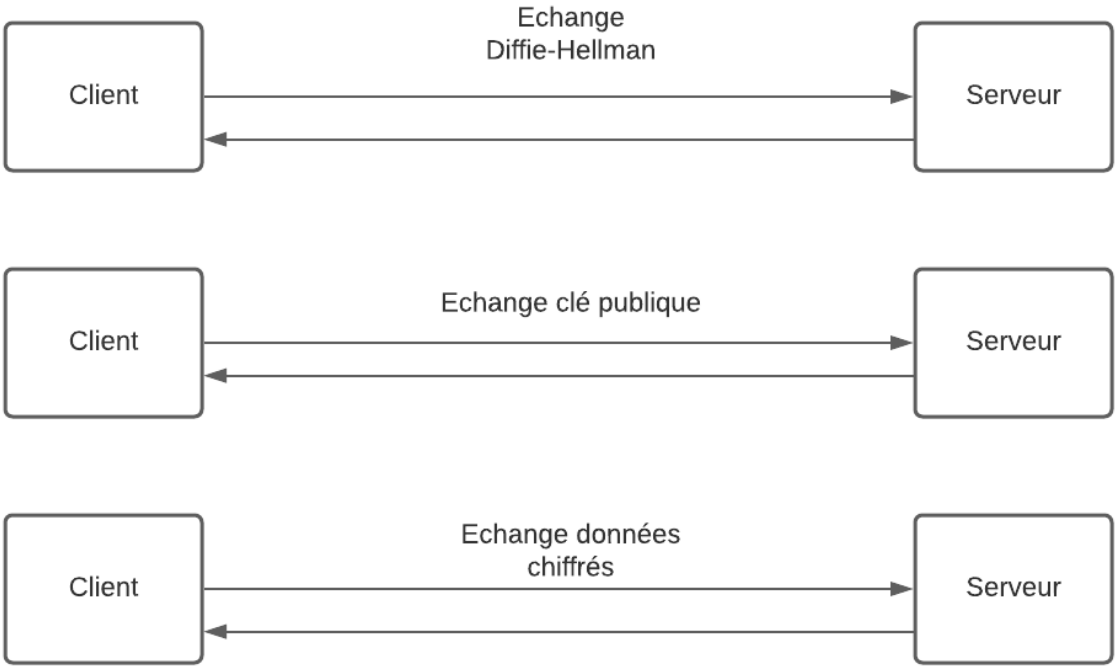
Un autre vecteur d'attaque à l'utilisation d'une solution décentralisée serait l'infection de noeud. Il a déjà été prouvé[1] dans un réseau à nœud (Tor) qu'il était possible d'infecter un nœud et d'injecter du contenu malveillant. Ce vecteur d'attaque pourrait potentiellement être utilisé dans une solution décentralisée et injecter de fausses informations.
En contre mesure, il peut être intéressant d'injecter des nœuds de contrôle. Chaque nœud va communiquer avec ses voisins, et des vérifications peuvent être effectuées par des entités reconnues, afin de vérifier l'authenticité des données.

IV. Protocole implémenté

Concernant le protocole utilisé pour la simulation du projet, nous avons décidé d'utiliser une solution centralisée. En utilisant cette solution, nous limitons au maximum les communications vers des pairs non connus et limitons ainsi les connexions non fiables. Il ne sera également pas possible d'accéder de manière directe aux données de l'application.

Nous avons décidé d'utiliser le protocole suivant:

[1] https://www.leviathansecurity.com/blog/the-case-of-the-modified-binaries/



Dans notre simulation, nous ne permettons qu'un seul échange client/serveur; nous ne gérons pas les requêtes simultanés, bien évidemment, dans un véritable environnement où l'application fonctionnerait avec des requêtes concurrentes. Afin de rendre l'application la plus sécurisée possible (autant côté client que serveur) chaque intervenant possède son propre jeu de clé publique/privée, utilisé pour sécuriser les données stockées. Une personne ayant donc accès aux données du client/serveur ne sera donc pas en mesure de lire les informations stockées, ou bien de les modifier manuellement.

Afin de sécuriser l'échange, au début d'une connexion le serveur enverra au client un fichier contenant des paramètres Diffie-Hellman afin de générer une clé de chaque côté, empêchant donc à quiconque qui écoutent le réseau de déchiffrer les messages échangés. À chaque nouvelle connexion, un nouveau jeu de clé sera généré et ne sera utilisé que pour cette connexion. Ensuite, le client et le serveur échangeront chacun leurs clés publiques. Tout échange sera chiffré avec la clé publique du destinataire, assurant alors la confidentialité des données.

Chaque donnée reçu devant être stocké sera alors chiffré en utilisant la clé publique[2] de l'utilisateur (client ou serveur) et pourra être déchiffré uniquement en utilisant la clé privée de celui-ci.
Concernant les permissions et l'accès aux données,

```
serveur@projet-anti-covid:/app# ls -l
drwx------ 4 serveur serveur 4096 Jun  4 09:02 data
-rw-r--r-- 1 serveur serveur 6152 Jun  4 09:08 server.sh
```

[2] Clé utilisateur propre à l'application, et non celle de l'échange.

```
-rw-r--r-- 1 serveur serveur 2048 Jun  4 09:56 server.zip
```

```
client@synchroneyes:~/Cours/SecuReseau# ls -l
-rw-r--r-- 1 client client 9044 Jun  4 11:53 client.sh
drwx------ 2 client client 4096 Jun  4 14:31 data
```

Chaque utilisateur (client/serveur) possède un dossier data, ce dossier ainsi que son contenu n'est visible QUE par l'utilisateur en lui-même.
Pour cette démonstration, les données stockées par le serveur ne sont pas chiffrées, mais elles le seraient de la même manière que le client. En revanche pour le client, tout est chiffré.

Côté serveur, tout est stocké sous forme de fichier

```
serveur@projet-anti-covid:/app $ find data/
data/
data/dhparams.pem
data/positions
data/positions/10
data/positions/10/11
data/positions/12
data/positions/12/13
data/positions/12/13/1478
data/positions/12/13/1473
data/positions/12/13/123
data/keys
data/keys/89.234.183.140.pubsrv
data/keys/89.234.183.140.pubclient
data/keys/89.234.183.140.common
data/keys/89.234.183.140.keysrv
```

Pour stocker et comparer les données, on utilise le format suivant
data/positions/Coordonnée X/Coordonnée Y/UUID Client

On stockera dans le fichier UUID Client l'état du client, c'est-à-dire s' il est malade, et s'il est cas contact. Si une personne se trouve dans les mêmes coordonnées, alors on regardera dans chaque fichier si quelqu'un est malade ou cas contact.

```
root@projet-anti-covid:/app# bash server.sh -d
[-] En attente d'une connexion client
[+] Information client: 89.234.183.140:11223
[+] Envoie du DHPARAMS
[-] En attente de la cle publique du client
[+] Generation de notre cle privee
[+] Generation de notre cle publique
[+] Derivation de la cle
[+] Envoie de notre cle publique au client
[-] En attente des infos clients GPS, ....
[+] Dechiffrement des donnes en cours
[+] Suppression des fichiers de data
[?] 0n4Xqoh7oxiyPsSRt81oxkOkLiDRxoih est malade: false, il est cas contact: false, il se trouve en [174,86]
[-] On verifie si un cas contact/malade est dans la zone
[-] Suppression des anciennes donnees de l'utilisateur
[-] Enregistrement des donnees d'utilisateurs
[-] Verification des cas contacts/malade
[+] Envoie des donnees au client
[-] Nettoyage des fichiers
[#] FIN DU TRAITEMENT CLIENT
--------------------------------
[-] En attente d'une connexion client
```

```
root@synchroneyes:~/Cours/SecuReseau# bash client.sh -d
[+] Génération de la clé privée du client
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................................................+++++
.....................+++++
e is 65537 (0x010001)
[+] Génération de la clé publique du client
writing RSA key
[+] Nouvel utilisateur, on crée son compte, il n'est pas malade ni cas contact
[+] Chiffrements de nos données en cours
[+] Génération du fichier client.txt
[+] Envoie du fichier client.txt au serveur
[-] En attente de DHPARAMS
[+] DHPARAMS reçu
[+] Génération de notre clé privée
[+] Génération de notre clé publique
[+] Envoie de notre clé publique au serveur
[-] En attente de la cle publique du serveur
[+] Dérivation de la clé
[+] Déchiffrement de nos données avant de les envoyer au serveur
[+] Chiffrement de nos données
[+] Suppression du fichier déchiffré
[+] Envoie de nos données en cours
[-] On attend la réponse du serveur avec nos données
[#] Un malade est dans notre zone: false | Un cas contact est dans notre zone: false
[-] Stockage des nouvelles données
[-] Nettoyage des fichiers
root@synchroneyes:~/Cours/SecuReseau#
```

Ci dessous se trouve une photo affichant l'exécution de notre simulation.
Le serveur se trouve sur une machine distante utilisant l'adresse IP: 62.210.125.202 et le port 1001.
Le client se trouve sur une machine distante utilisant l'adresse IP 89.234.183.140.
Dans notre simulation, le client doit avoir un port ouvert et redirigé vers le port qu'il aura défini dans son application. Dans une application réelle, la méthode utilisée pour communiquer ne sera pas la même, ici nous utilisons la commande "nc" pour écouter, et émettre.

Côté client, l'application dispose de paramètre d'exécution:

Paramètre	Explication
-d | --debug	Permet d'afficher toutes les informations concernant les actions en cours

-m | --malade	Permet de se déclarer comme malade, atteint du virus de la covid-19
-s | --soigne	Permet de se déclarer comme soigné. Dans une véritable application, on pourrait déclencher cet événement de manière automatique après X jours.
-i | --info	Permet d'afficher toutes les informations de l'utilisateur: - UUID - Malade - Cas contact - Position X - Position Y
-g | --gps x:y	Permet de définir sa position

Côté serveur, l'application ne dispose que du paramètre "-d" qui permet d'afficher toutes les informations concernant les actions en cours.

Lorsqu'une personne devient cas-contact, c'est-à-dire qu'elle se trouve à la même position qu'une personne malade, on peut imaginer une notification de manière automatique, dans un délai aléatoire compris entre 10 et 12 heures après la détection.

Paramètres à modifier:
Serveur: L4 - PORT=XXXX
Client:

- L3 - HOST=IP_SERVER
- L4 - PORT=PORT_SERVER
- L5 - PORT=PORT_SUR_LEQUEL_ECOUTER # Pour les réponses
- L5 - ip_client= IP du client # Peut être localhost si les deux scripts sont sur la même machine