TP secure chat

Ce TP consiste à sécuriser un chat et à se prémunir de certaines attaques en utilisant des outils/methodes adaptées.

Les fichiers du TP s'organise comme ceci :

- source
    - chat_client.py : classe servant le client au chat
    - chat_server.py : classe servant de serveur au chat
    - generic_callback.py : classe utilisée par le client et envoyé au serveur permettant de récupéré les messages
    - basic_gui.py : client graphique du chat, repose du ChatClient
- ANSWERS.md : contient vos réponses aux questions de TP
- README.md : donnes les consignes

Même si la base du TP se fera à partir de source/basic_gui.py, soyez curieux !

Coding rules

Le code devra respecter un certain nombre de règles (arbitraires) pour ne pas perdre de point :

- Le code doit être commenter
- Les noms des fonctions et de variables s'écrivent en en snake case (nom_de_fonction)
- Les noms de constantes s'écrivent en snake case majuscule (NOM_DE_CONSTANTES)
- Les noms de classe s'écrivent en camel case (MaClass)
- Les noms de fichiers reprennent le nom de la classe en snake case minuscule (ma_class)
- Pas de valeurs littérales dans le code, utilisez toujours des constantes pour les expliciter
- Une classe, un fichier
- Une fonction ne doit pas dépasser 20 lignes
- Pas plus de deux imbrications de bloc
- Utilisez des variable intermédiaires lors d'enchainement de fonction. Exemple à ne pas faire : x = y(z(w(g,h,j,k)))
- Utilisez les log (self._log) plutôt que les print()
- Utilisez Github pour y mettre votre projet

Globalement : Plus je comprends ce que vous avez fait, plus j'ai envie de mettre des points ... et réciproquement.

Prérequis

Vous devez être sous un dérivé de Debian (Ubuntu, xubuntu, etc), avoir python3 à jours et avoir installer avec pip3 les bibliothèques suivantes :

- pyro5
- cryptography

Il est recommandé d'utiliser un IDE comme Code avec de l'autocomplétion; Si toutefois vous être plus à l'aise avec d'autres outils (vim, emacs, etc), vous pouvez parfaitement les utiliser.

Prise en main

Ouvrez trois terminaux et lancez les commandes suivantes :

- python3 source/chat_server.py
- python3 source/basic_gui.py
- python3 source/basic_gui.py

Vous aurez alors un serveur fonctionnel et deux clients graphiques. Connectez vous (file->connect) en changeant le nom dans les deux clients (défaut : foo) et échangez quelques messages.

1. Comment s'appelle cette topology ?
2. Que remarquez vous dans les logs ?
3. Pourquoi est-ce un problème et quel principe cela viole t-il ?
4. Quelle solution la plus **simple** pouvez-vous mettre en place pour éviter cela ? Détaillez votre réponse.

Chiffrement

Dans cette partie, vous allez mettre en place l'algorithme AES (CTR) pour chiffrer la communication entre les différents interlocuteurs. Pour réaliser cela, vous allez vous baser sur la bibliothèque *cryptography*. Elle contients tous les éléments donc vous allez avoir besoins : AES, pkcs7, PBKDF2HMAC. La documentation dispose de toutes les informations nécessaires ainsi que des exemples pour vous aider.

Vous aurez à dériver la classe BasicGUI pour créer la classe CipheredGUI. Il vous faudra faire les modifications suivantes :

- Surcharger le constructeur pour y inclure le champ self._key qui contiendra la clef de chiffrement (default : None)
- Surcharger la fonction _create_connection_window() pour y inclure un champ password
- Surcharger la fonction run_chat() pour y inclure la récupération du password et faire la dérivation de clef (self._key)
- Créer une fonction encrypt(), prenant une string et retournant un typle de bytes (iv, encrypted)
- Créer une fonction decrypt(), prenant un tuple en paramètre et retournant une string utf8
- Surcharger les fonctions send()/recv() pour y faire intervenir encrypt() et decrypt(), respectivement.

Points d'attention :

- Le chiffrement par bloc implique une taille multiple de la taille du bloc (symetrical padding, pkcs7)
- La taille de la clef doit être de 16 bytes (Key stretching, key derivations functions, PBKDF2HMAC)
- Toutes ces opérations se font sur des octets; par défaut, les strings en python sont en utf8 : bytes(x, "utf8") pour avoir des octets
- Pour passer d'un type bytes à une string il faut appeler str(x, "utf8")
- Serpent, la bibliothèque de serialisation ne transmets pas directement les bytes mais un dict contenant une clef data et une clef encoding. Pour transformer cela en bytes, il faut utiliser serpent.tobytes (avec l'import adapté).
- L'aes ici à une taille de clef et de bloc 128 bits (16 bytes).
- Pour la dérivation de la clef, un salt constant sera suffisant pour le TP

1. Est ce que urandom est un bon choix pour de la cryptographie ? Pourquoi ?

2. Pourquoi utiliser ses primitives cryptographiques peut être dangereux ?
3. Pourquoi malgré le chiffrement un serveur malveillant peut il nous nuire encore ?
4. Quelle propriété manque t-il ici ?

Authenticated Symetric Encryption

Pour être sûre que le message n'a pas été altéré, il faut mettre en place un chiffrement avec un HMAC. La bibliothèque *cryptography* contient une solution adaptée : le module Fernet. A partir d'une clef de 32 octets (format base64) il permet de chiffrer/et dechiffrer tout en étant sûr de l'intégrité du message. Pour généré la clef de 32 bits à partir du password, vous pouvez utiliser dans le cadre du TP la fonction de hashage SHA256.

Dérivé la classe CipheredGUI pour créer FernetGUI. La seule modification portera sur la fonction encrypt/decrypt, ainsi que run_chat qui utilisera sha256().digest() + base64.b64encode() au lieu de PBKDF2HMAC.

1. Pourquoi Fernet est moins risqué que le précédent chapitre en terme d'implémentation ?
2. Un serveur malveillant peut néanmoins attaqué avec des faux messages, déjà utilisé dans le passé. Comment appel t-on cette attaque ?
3. Quelle méthode **simple** permet de s'en affranchir ?

TTL

Vous avez utilisez deux methodes fournie par la classe Fernet : encrypt et decrypt. Un autre jeu de methode permet d'améliorer la robustesse en donnant une durée de vie au message (Time To Live, TTL). Si un message dépasse la durée de vie, il est ignoré au moment du décodage.

Dérivé la classe FernetGUI en TimeFernetGUI et utiliser les fonctions encrypt_at_time et decrypt_at_time dans les fonctions encrypt et decrypt, respectivement. Utilisez un TTL de 30 secondes et capturez l'exception potentielle (try/except InvalidToken as e) avec un log d'erreur en cas d'exception. Pour le temps, on utilisera int(time.time()).

1. Remarquez vous une différence avec le chapitre précédent ?
2. Maintenant soustrayez 45 au temps lors de l'émission. Que se passe t-il et pourquoi ?
3. Est-ce efficace pour se protéger de l'attaque du précédent chapitre ?
4. Quelle(s) limite(s) cette solution peut rencontrer dans la pratique ?

Regard critique

J'ai pris des raccourcis, pris des décisions arbitraires et utilisé des bibliothèques tiers. Ai-je laissé des vulnérabilités ?

A vous maintenant de trouver ce qui ne va pas, de justifier votre propos et de proposer une alternative. Ce n'est pas tant la quantité de point que vous trouverez que la pertinence de votre analyse qui vous permettera de décrocher des points.