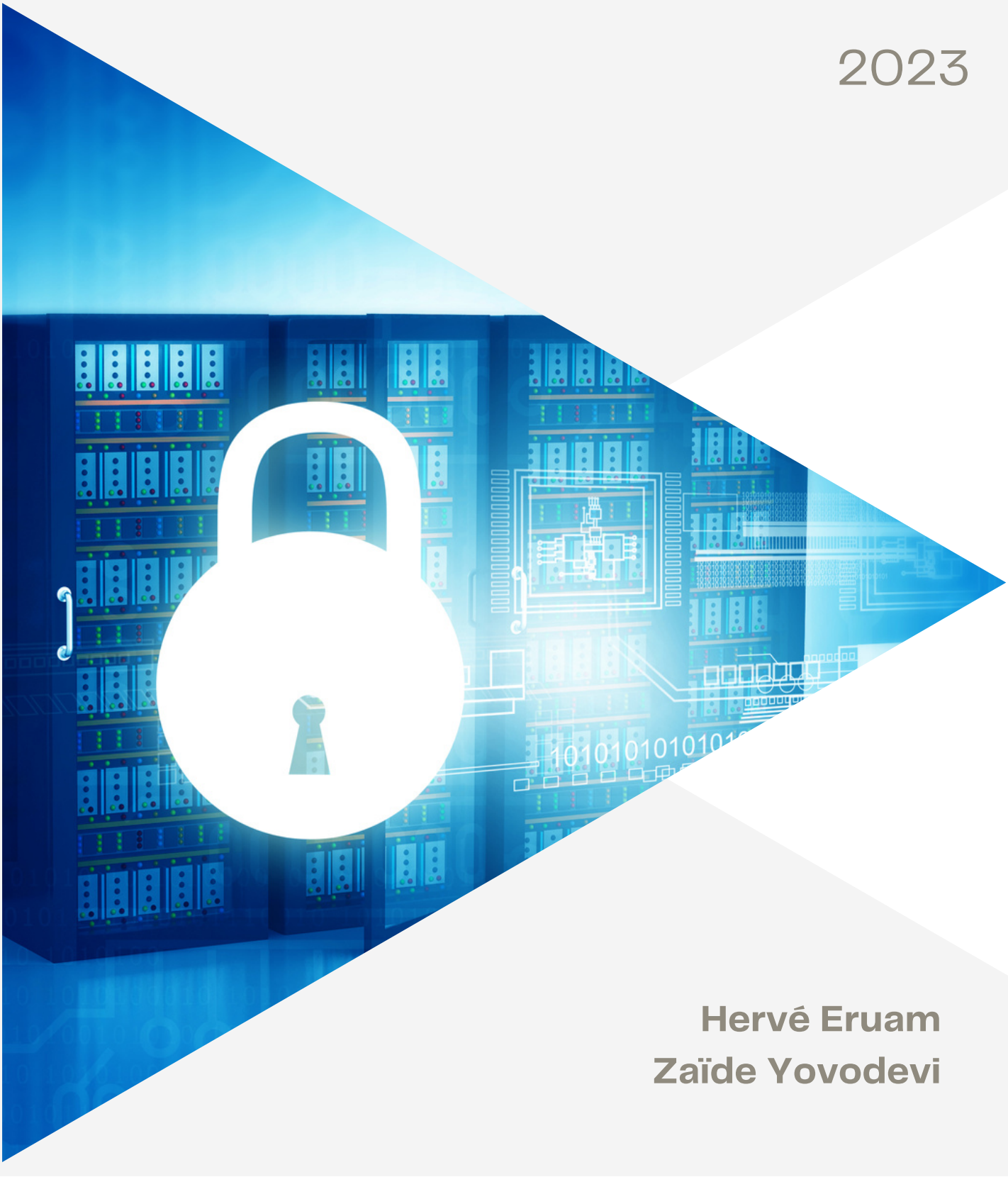
Rapport INFO-731 SERVEUR PROXY

2023



Hervé Eruam
Zaïde Yovodevi

Sommaire

Introduction

À l'issue de notre formation sur la sécurité des systèmes cyber-physiques, nous avons eu l'occasion de travailler sur un Projet. L'objectif de ce dernier étant la réalisation d'un proxy de sécurité Web. Ce projet a été l'occasion de mettre en pratique les connaissances acquises tout en les confrontant aux difficultés dues à leurs mises en place dans un contexte réel.

Ce rapport a pour but de présenter, les résultats obtenues ainsi que les difficultés rencontrées. Pour les précisions sur la partie code, de nombreux commentaires sont présents directement dans les fichiers .py.

Le proxy est disponible sur le répertoire git suivant :
https://github.com/tweex-creator/projetProxy

Nos objectifs

Le proxy que nous avons réalisé au cours de ce projet doit permettre d'établir une connexion chiffrer entre deux réseaux.
Le proxy est en réalité composé de deux éléments :

- Le **"proxy d'entrée"** auquel se connecte directement le client. Celui-ci peut être directement exécuté sur la machine du client ou sur une autre machine. Dans tous les cas, il faudra s'assurer de la sécurité des données lors de l'échange entre le client et ce proxy d'entrée (non traité ici).
- Le **"Proxy de sortie"** auquel le proxy d'entrée envoie toutes les requêtes via une liaison encrypté et qui fournit les réponses à retransmettre au client.

De cette manière, même sur un réseau publique, les données ne sont pas clairement visibles par un éventuel attaquant.

Ci-dessous, un tableau récapitulant les fonctions implémentée:

Fonction	Descritpion
Encryptions	Une fois la liaison avec le proxy de sortie établie, toutes les données en transit sont cryptées par clé symétrique.
Échange de clé autonome	La clé symétrique est échangée automatiquement sur le réseau en utilisant un cryptage asymétrique (RSA).
Confirmation humaine	Un nouveau client ne peut accéder à la clé symétrique qu'après qu'il a été manuellement accepté sur le proxy de sortie.
HTTP	Le proxy réalisé prend en charge le chargement de page HTTP
HTTPS/TUNNELING	Le proxy réalisé prend en charge l'établissement de Tunnel HTTPS (Requête de type CONNECT)

Fonctionnement du proxy

Gestion des connexions entrantes (client)

Lorsqu'un nouveau client tente de faire une requête sur le proxy (d'entrée), on vérifie avant toute chose que la connexion avec le proxy distant soit toujours valable (connexion + validité des clés). Dans le cas où la connexion n'est plus bonne et ne peut pas être rétablie, on rejette la demande du client (erreur 502).

On récupère ensuite le contenu de la requête afin de pouvoir la traiter de la bonne manière, on s'intéresse notamment à la méthode (HTTP) utilisée.

- Pour les requêtes classiques (GET, POST, PUT, ...), on encrypte le contenu de la requête que l'on envoie au proxy de sortie. Celui-ci décrypte la requête, l'effectue et renvoie la réponse sous forme cryptée. Le proxy d'entrée peut alors la décrypter et envoyer la réponse au client.
- Pour les requêtes de type CONNECT, elles sont cryptées et transmises au proxy distant. Si celui-ci accepte la connexion, une connexion permanente et transparente et mise en place jusqu'à la fermeture par le client. Dans le cas d'une requête CONNECT, il est inutile d'encrypter la communication, car elles le sont déjà (https).

Vérification de la liaison avec le proxy de sortie

Avant de procéder à l'échange de donnée en provenance du client, nous effectuons une vérification sur la liaison entre les deux éléments du proxy.

On commence par vérifier si le proxy de sortie et bien joignable en ouvrant une connexion. Si celle-ci c'est ouvert avec succès, on vérifie la cohérence des clés sur les deux proxys.
Pour cela, on envoie un message crypté dont le contenu est "ping". Quand le proxy de sortie reçoit ce message, il répond par un "pong" (crypté).
Dans le cas où les clés ne seraient pas les mêmes, le proxy de sortie ne pourrait pas déchiffrer le "ping" et n'y répondrait pas. Dans ce cas, la connexion sera considérée comme n'étant pas établie.

Échange de clé symétrique

Pour encrypter les communications entre nos deux proxys, nous utilisons une clé symétrique qui a pour avantage d'être plus rapide qu'un jeu de clés asymétriques. En revanche, celle-ci soit être connue des deux parties sans qu'elles ne soient divulguées à des tierces . Nous avons donc mis en place un système qui utilise une fois une communication sécurisée par clé asymétrique pour l'échange de la clé clé symétrique. Le fonctionnement est le suivant :

- Le proxy d'entrée envoie un message en claire "START_SECURE_SESSION", le proxy de sortie va alors demander à un utilisateur/administrateur sur le proxy de sortie s'il souhaite autoriser la connexion du proxy d'entrée.
- Une fois que le proxy d'entrée à reçus l'autorisation du proxy de sortie, il va générer un couple de clé publique/privé (RSA) et envoyer la clé publique au proxy de sortie.
- Le proxy de sortie va si besoin générer une nouvelle clé symétrique, il va la crypter à l'aide de la clé publique du proxy d'entrée et lui l'envoyer.
- Le proxy décrypte la clé symétrique et la sauvegarde.
- Pour vérifier que tout fonctionne bien, le proxy d'entrée envoie un message crypté "OK", si il le proxy d'entrée le reçois bien, il renvoie lui-même un "OK" crypté. Et enfin, si le proxy d'entrée reçoit bien le message, la connexion est considérée comme établie.

Les difficultés rencontrées

Tout au long du développement, nous avons fait faces à quelques difficultés. Nous présentons ici les principales et les solutions que nous avons mises en place.

Les requêtes HTTPS :

Lors de la phase de programmation, nous nous sommes heurtés pendant plusieurs jours à un problème majeur, le navigateur client affichait systématiquement "EMPTY_RESPONS" ou "TUNNELING_ERROR" quand bien même le proxy recevait et transmettait bien les requêtes. Après investigation, il s'est avéré que les sites fonctionnant en HTTP uniquement semblait fonctionner partiellement. La contenue était bien chargée, en revanche, la mise en forme était absente. Le problème venant enfaite de la méthode CONNECT de HTTP que nous ne prenions pas en charges dans notre proxy. Celle-ci indique au proxy qu'il doit ouvrir une connexion permanente entre le serveur et le client pour que ces derniers puissent mettre en place leurs propres communications sécurisées (HTTPS). Nous avons donc implémenté cette fonctionnalité, ce qui a permis de résoudre le problème.

Utilisation de la bibliothèque cryptodome:

Si la bibliothèque cryptodome n'a posé aucun problème pour l'usage de la clé symétrique, l'implémentation du mécanisme RSA n'a pas été aussi fluide. En effet, il semble que les évolutions en termes de sécurité aient poussé les développeurs à supprimer l'implémentation de la fonction RSA.decrypt/encrypt. La solution a été d'utiliser la nouvelle version : PKCS1_OAEP.

Comparatif des performances

±20x

Notre proxy rend le chargement d'une page comme httpforever.com 20 fois plus long que sans.

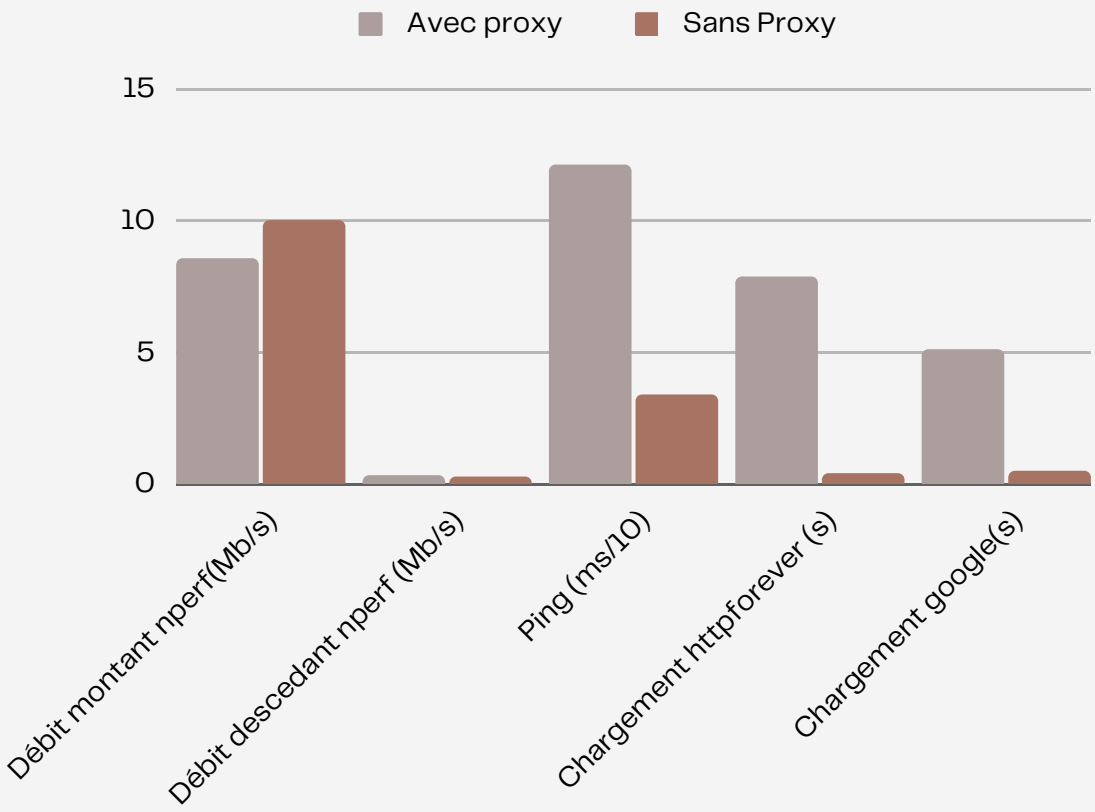
Une fois notre proxy terminé, nous avons réalisé une série de tests afin de déterminer ces performances en termes de vitesse. Celle-ci ce sont avérées particulièrement faible, ce que nous essaierons de comprendre plus loin.

Protocol de test:

Pour tester notre proxy, nous avons :

- Réalisé un test de débit **nperf** (avec et sans proxy)
- Réaliser un programme python (speedTest.py) qui mesure le temps mis pour charger un site web dont l'adresse est donnée (moyenne sur 10 essais)
  - Nous l'avons utilisé sur **http://httpforever.com** avec et sans le proxy
  - Nous l'avons utilisé sur **https://google.com** avec et sans le proxy

Resultats:



-15 %

De débit descendant lorsque le proxy est activé

18,5X

des lecteurs apprécient les informations précises

On observe sur le graphique ci-dessus trois éléments clés :

- Les tests de débits ne montrent par un écart très important (15 %).
  - Ceci est dû au fait que la plateforme utilisée pour faire le test est en HTTPS, le proxy n'encrypte donc pas la connexion et sert simplement de relais (tunnel). On note malgré tout une petite perte de débit qui vient du temps que met le proxy à transmettre les données.
- Le chargement de https://google.com est 10 fois plus lent avec le proxy :
  - Tout comme le test de débit, google.com est chargé via un tunnel, pourtant, celui-ci ne met pas 15 % de temps en plus à charger, mais 1 000%. Cet écart s'explique par le fait que le lors du test de débit, le temps d'ouverture du tunnel n'est pas pris en compte. Dans le cas du chargement d'une page comme Google, il est pris en compte et il est significatif. L'ouverture d'un tunnel implique en effet de vérifier que la connexion soit toujours ouverte, et d'échanger des messages qui eux sont cryptés par le proxy.
- Le chargement de httpforever.com est 18 fois plus long que sans le proxy :
  - En toute logique, les sites de type HTTP sont les plus longs à charger avec notre proxy, il est en effet nécessaire d'encrypter et de décrypter l'intégralité de la requête et de la réponse.

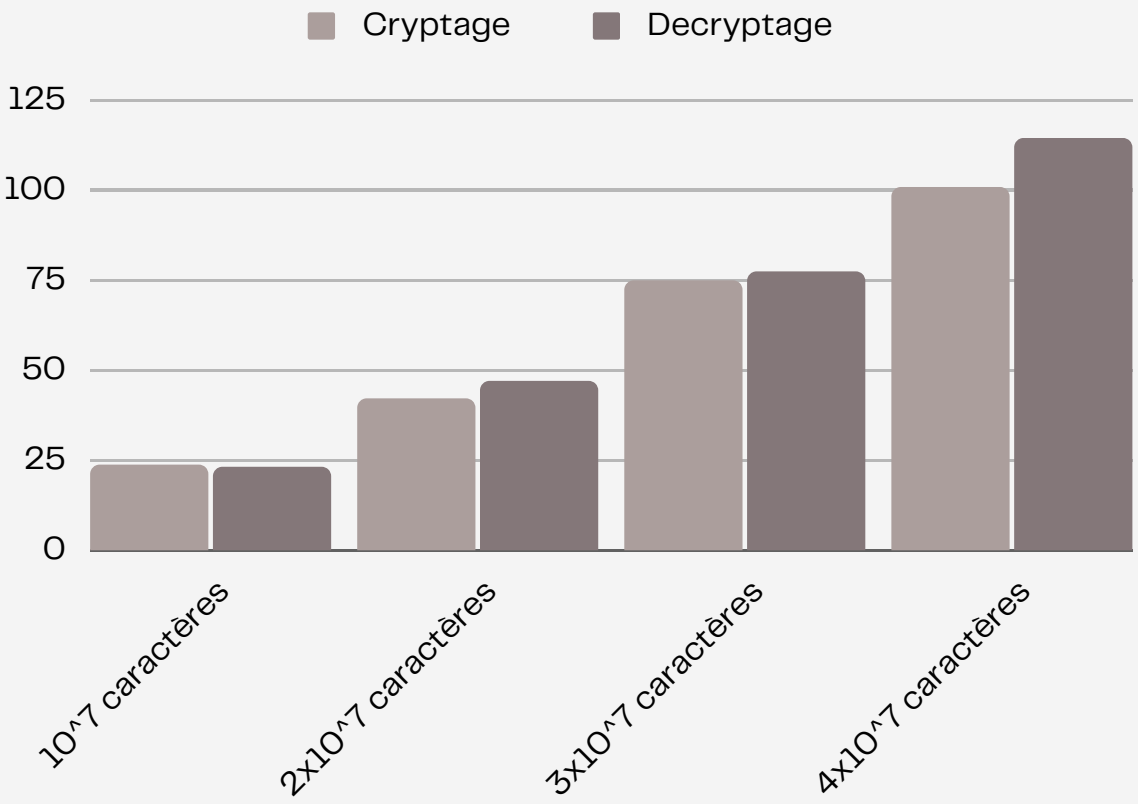
Hormis les performances global de notre proxy, nous avons aussi souhaité tester les performances en cryptage/décryptage de façons individuelles.

Protocol de test:

Pour tester les performances en cryptage/décryptage, nous avons :

- Réalisé des mesures du temps de mis pour crypter puis décrypter des messages de longueur variables.

Resultats:



Ces observations nous permettent de montrer que le cryptage/décryptage des pages est relativement rapide en comparaison des temps de chargement constaté plus tôt. En effet, une page web pèse en moyenne 2,6 Mo (source : Clubic) soit l'équivalent de 2,6 * 10^7 caractères. Le cryptage s'opère donc normalement en ± 50 ms, ce qui est négligeable. De telle performance sont en autre dû à l'utilisation d'une librairie très utilisées et développées depuis plusieurs années, permettant ainsi une bonne optimisation. Ceci mets en évidence le fait que les ralentissement de notre proxy sont principalement du à son fonctionnement plutôt qu'au cryptage/décryptage.

Pistes d'amélioration

Nous avons au cours de ce projet atteint le stade d'un proxy fonctionnel, cependant il y a plusieurs axes qui peuvent être amélioré autant sur l'aspect performance que sur l'aspect sécurité.

Sécurité:

- Sauvegarde de la clé symétrique :
  - Aux lieux de générer une nouvelle clé symétrique et devoir la retransmettre à chaque démarrage, il pourrait être intéressant de la sauvegarder de façons sécurisée sur les machines.
- Plusieurs clés symétriques :
  - Si on veut que plusieurs proxys d'entrées utilisent le même proxy de sortie, il peut être intéressant que chaque proxy d'entrée est une clé symétrique différentes. Comme cela, n'importe quel proxy d'entrée ne peux pas lire les messages des autre proxy d'entrées.

Performances:

- Ne pas ouvrir une connexion juste pour vérifier que la connexion est établie
  - On peut directement utiliser la connexion utilisée pour la transmission des données
- Utiliser un langage plus rapide que python.
- Réaliser des mesures de temps pour chaque opération affin de déterminer les plus lentes et agir dessus si possible.