BORDEAUX INP Enseirb-Matmeca

Création d'un dispositif d'attaque par usurpation de requêtes DNS

Rapport de projet

Paul BARBARIN

Kenzo MORENO CAPRIO

Promo RSR 2024

Contents

1 Objectifs du projet

Le but du projet est de programmer une application malveillante permettant de rediriger une ou plusieurs victimes, présentes sur le même réseau que l'attaquant, vers de faux sites web dans le but de récupérer des informations personnelles, telles que leurs mots de passe. L'utilisation principale de cette application serait envisagée sur des réseaux publics (comme ceux d'un McDonald's ou d'un aéroport) ou sur des réseaux privés que l'attaquant aurait infiltré (par exemple, dans une entreprise ou une salle de classe disposant d'un réseau privé).

L'objectif final serait de concevoir un système indétectable pour des personnes non initiées ou non sensibilisées à la sécurité informatique, ou alors difficilement détectable par des individus plus avertis.

Ainsi, l'objectif est de manipuler les requêtes des utilisateurs sur un réseau, tout en ayant un contrôle total sur quelles cibles attaquer, et quels sites web remplacer.

2 Attaque Man-In-The-Middle

Nos objectifs impliquent inévitablement la mise en place d'une attaque de type **Man-In-The-Middle**. En effet, nous cherchons à manipuler et modifier les requêtes DNS transitant sur le réseau. Le point d'entrée sera donc de mettre en place un MITM entre le serveur DNS et les différentes victimes.

La manière la plus courante d'effectuer ce type d'attaque est de manipuler les adresses physiques (Ethernet dans notre cas). On utilise donc le protocole ARP à des fins malveillantes, afin d'assurer d'un côté que les victimes considèrent le serveur attaquant comme le serveur DNS, et de l'autre que le serveur DNS considère le serveur attaquant comme les victimes.

Cela s'illustre en pratique par l'émission régulière (toutes les secondes dans le cas de notre programme) de 2 paquets d'annonce ARP par victime :

- Une requête ARP envoyée au serveur DNS par l'attaquant prenant l'IP de la victime (et venant donc de l'addresse MAC de l'attaquant)
- Une requête ARP envoyée à la victime par l'attaquant prenant l'IP du serveur DNS (et venant donc de l'addresse MAC de l'attaquant)

Ici, il est important de comprendre que les requêtes n'ont comme intérêt que de provoquer une mise à jour de la table de cache ARP du serveur DNS et des victimes. Cette mise à jour forcée est illustrée dans la Figure 1 :

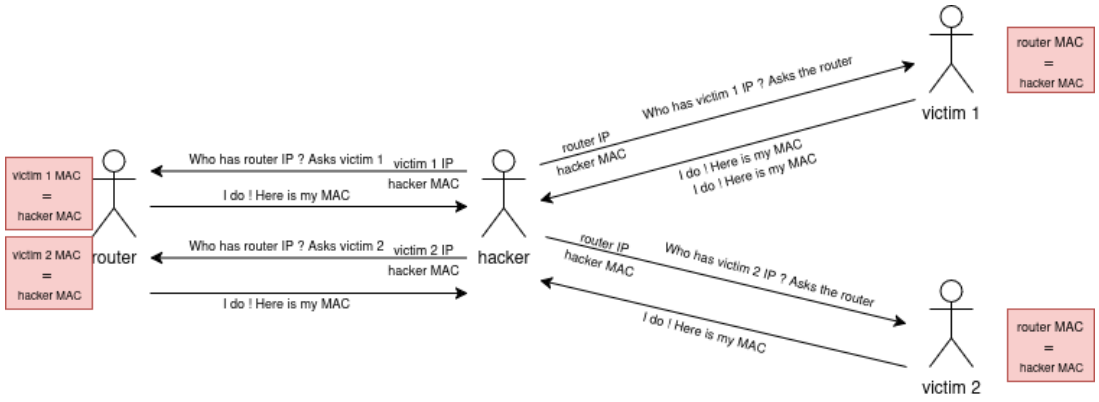

Figure 1: ARP attack.

Durant tout le temps de l'attaque, l'attaquant se retrouve dans une position intermédiaire entre le serveur DNS et les victimes, comme observable sur la Figure 2. Et c'est justement cette position d'intermédiaire que l'attaquant va exploiter, cette position lui donnant un contrôle total sur les paquets transitant sur le réseau.

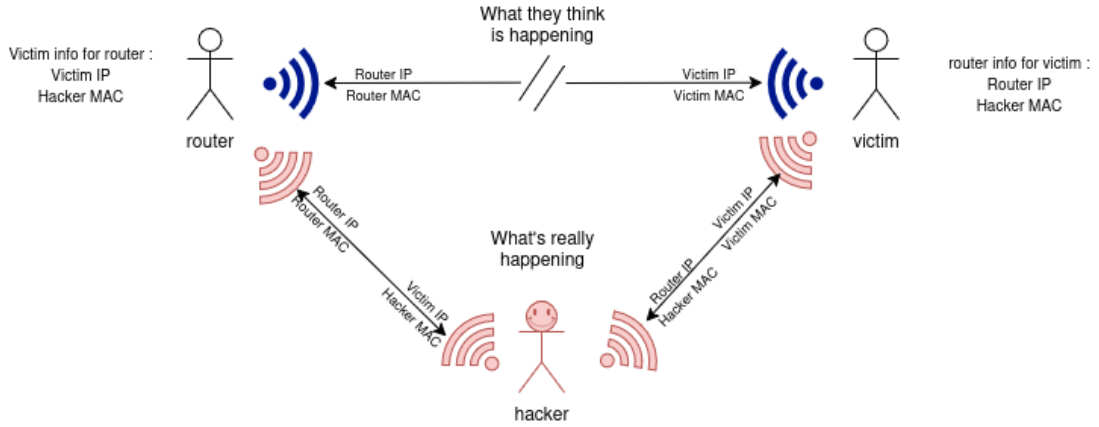

Figure 2: Man In the Middle.

3 Répondre aux requêtes DNS

L'objectif du projet étant de rediriger les victimes vers des faux sites, il apparaît comme indispensable le fait d'envoyer de fausses réponses à certaines requêtes DNS.
Pour cela, il a fallu mettre en place un *sniffer* sur le port 53, usuellement utilisé par DNS. Lorsqu'un paquet était repéré sur ce port, il était alors envoyé à des fonctions d'analyse, lancées sur un thread à part pour chaque paquet reçu, afin de permettre au sniffer de continuer de tourner tout en optimisant la performance.

Une fois le paquet intercepté, l'idée est maintenant de savoir s'il faut créer ou non une réponse falsifiée pour cette requête.
Pour cela, le paquet passe par de nombreuses vérifications :

- L'adresse IP d'origine fait-elle partie des victimes ?
- L'adresse IP de destination est-elle bien celle du serveur DNS ?
- Y a-t-il bien les couches attendues dans ce paquet ?
- Est-ce bien une requête DNS ? Dans le cas d'une réponse, elle est automatiquement forwardée **au niveau logiciel**, car cela veut dire que le programme a accepté de laisser passer la requête correspondante.
- La requête permet-elle de résoudre un nom de domaine que nous voulons usurper ?
- La requête est-elle bien en IPv4 ? Dans le cas d'une requête AAAA (ipv6), elle est automatiquement bloquée par le programme)
- La requête est-elle bien originaire d'une victime dont nous n'avons pas déjà récupéré les informations sur le site demandé ? (afin de permettre à la victime de pouvoir réellement se connecter au site usurper, rendant ainsi l'attaque crédible)

Si toutes ces conditions sont remplies pour un paquet, l'attaquant forge une réponse falsifiée de cette requête pour la renvoyer à la victime.
Dans celle-ci, l'attaquant place alors sa propre IP dans la réponse de la requête. De cette manière, la victime, lors de sa future tentative de connexion au site web usurpé, va être redirigée vers la machine de l'attaquant. La suite du rapport explique en détails la manière dont nous avons automatisé la configuration du serveur web, permettant ainsi les connexions HTTP et même HTTPS (attaque évoluée que nous décrirons à la fin)

4 Forwarding et bloquage des paquets non sélectionnés par l'attaque

Pour permettre un bon fonctionnement du réseau, il est nécéssaire de transmettre les requêtes et les réponses DNS non sélectionnées par l'attaque.
Ainsi, lorsqu'une des vérifications énoncées précédemment n'est pas remplies, on envoie le paquet dans une fonction qui va manuellement modifier la paquet pour simuler un *forward*.
Pour cela, on modifie les adresses mac d'origine et de destination pour l'envoyer au vrai destinataire du paquet.

Originellement, on ne bloquait que le port 53 au niveau du pare-feu de la machine attaquante, pour le forward.
Mais suite à de nombreux tests, on se rendait compte que le passage de certains paquets passaient à travers le dispositif. Nous avons donc pris la décision de bloquer totalement le forward au niveau pare-feu, pour ipv4 et ipv6. Le forwarding doit se faire au niveau logiciel, directement sur le programme.

Il a alors fallu ajouter un nouveau *sniffer*, récupérant tous les paquets reçu, sauf ceux du port 53, et les envoyant dans une fonction de forward manuel dans un autre thread. Ainsi, chaque paquet passant par la machine de l'attaquant, qui n'est pas un paquet DNS, est automatiquement forwardé.

Aussi, pour rajouter une sécurité supplémentaire sur les paquets reçu sur le port 53, il a été décidé de bloquer tous les paquets DNS n'étant pas de type A (IPv4).
Ainsi, tous les paquets DNS d'un type autre (comme $AAAA$ (IPv6)), sont simplement bloqués).

Sur la Figure 3, un diagramme du fonctionnement des différents threads liés à la gestion des paquets est représenté. Ainsi, on a 2 threads constamment en cours d'éxecution: un pour sniffer le port 53 et un autre pour tous les autres ports. Lorsqu'un paquet est détecté par l'un de ces deux threads, un nouveau thread temporaire est créé afin d'exécuter les comportements adéquats au paquet.

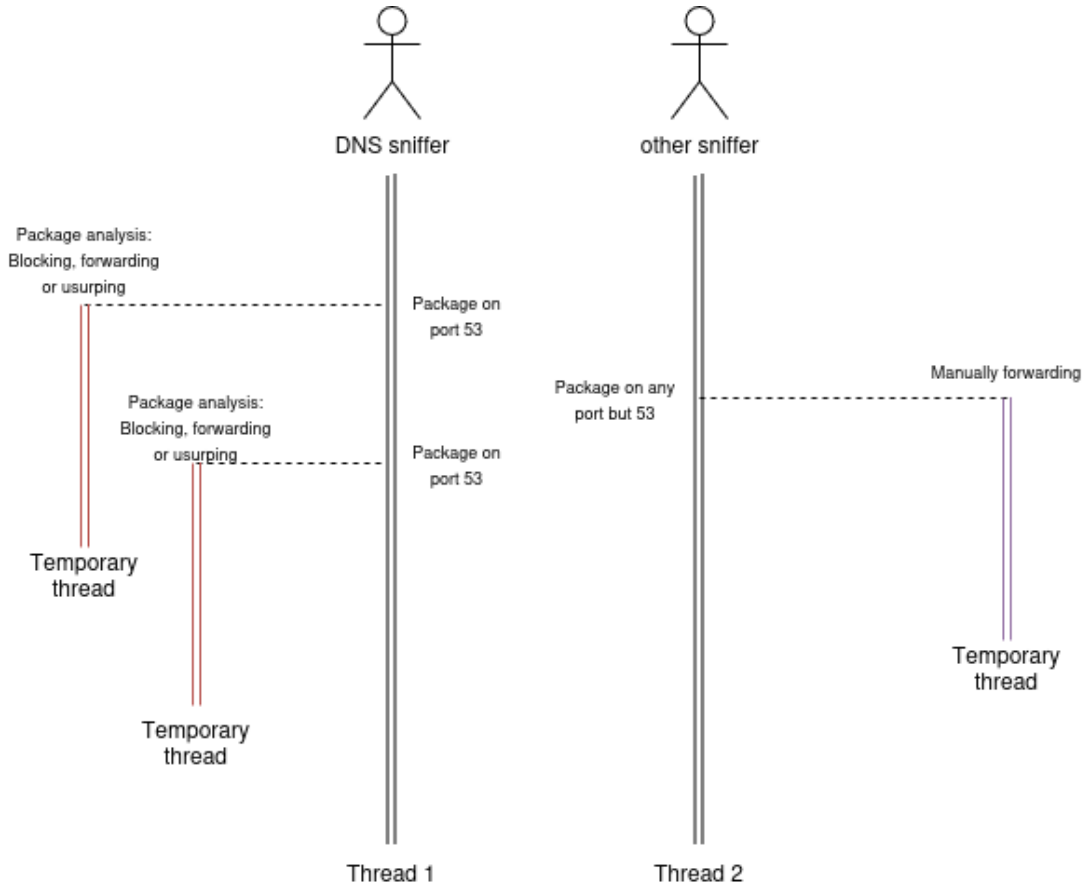

Figure 3: Package threading.

5 Mécanismes et fonctionnement de la CLI et structure du programme

5.1 Fonctionnement de la CLI

L'une des plus grandes plus-value du projet est de fournir une interface client conviviale et simple d'utilisation, permettant à l'attaquant de réaliser les principales actions de manière simple. Le programme prend en effet la forme d'un shell proposant un certain nombre de commandes pour interagir avec l'attaque, comme par exemple les commandes `add_site` et

`add_domain_to_site` pour gérer les sites web et domaines à usurper. Le script principal gère automatiquement le passage des paramètres spécifiés aux commandes. Une librairie permettant l'autocomplétion a également été installée, permettant lors de l'appui sur TAB de suggérer les commandes à l'attaquant.

5.2 Structure du programme

Le programme est articulé autour d'un script principal qui inclue principalement 3 types de fichiers :

- Les fichiers de commande - localisés dans le dossier `commands/`
- Les fichiers de thread - localisés dans le dossier `corethreads/`
- Les fichiers utilitaires - localisés dans le dossier `utils/`

Une liste de toutes les commandes disponibles est affichée au moyen de la commande help. Sur la Figure 4 est résumé la structure globale du programme.

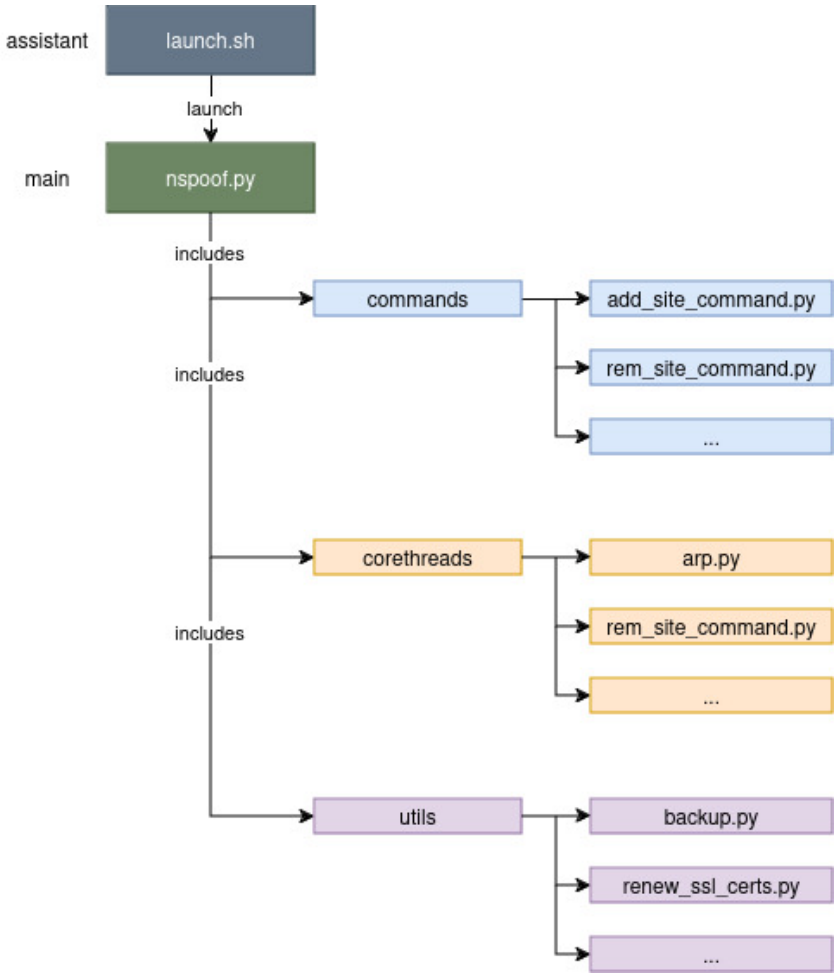

Figure 4: Structure globale du programme.

5.3 Encapsulation de la logique multi-thread

Dans notre programme, nous avons fait le choix d'encapsuler la logique relative aux threads logiciels Python dans une classe *ThreadManager* dédiée à cet effet :

```
1  import threading
2  from corethreads.arp import arp
3  from corethreads.dns_listen import dns_sniffer
4  from corethreads.arptable import arptable
5  from corethreads.other_listen import other_sniffer
6
7  #
8  # Handler of the different threads to launch.
9  #
10 class ThreadManager:
11
12     def __init__(self):
13         self.stopEvent = threading.Event()
14         self.startEvent = threading.Event()
15         self.thread1 = threading.Thread(target=arp, args=(
               self.stopEvent,))
16         self.thread2 = threading.Thread(target=dns_sniffer,
               args=(self.stopEvent,))
17         self.thread3 = threading.Thread(target=other_sniffer,
               args=(self.stopEvent,))
18
19     def start_threads(self):
20         self.startEvent.set()
21         self.thread1.start()
22         self.thread2.start()
23         self.thread3.start()
24
25 manager = ThreadManager()
```

Cette classe embarque nottament un mécanisme de signal permettant d'annoncer aux threads terminer, permettant d'attendre leur terminaison propre avant la fermeture du programme. La méthode `start_threads` quant à elle permet uniquement un lancement des threads ARP et sniffer, qui sont lancés lors du démarrage de l'attaque et sans paramètres, toujours de la même manière.

6 Déploiement des configurations web

Maintenant que l'attaque MITM est en place, et que les paquets DNS transitant entre les victimes et le serveur DNS sont interceptés et manipulés, il est nécéssaire de gérer le déploiement des sites web à proprement parler, ce qui passe donc par l'installation d'un serveur web sur la machine de l'attaquante. Notre programme a été créé pour fonctionner avec un serveur *nginx*, mais on pourrait très bien penser à une modification ou un ajout

de fonctionnalités pour supporter apache, voire donner le choix à l'attaquant. Pour le moment, le script de pré-lancement du programme s'assure que *nginx* est installé sur la machine de l'attaquant et que *apache* est désactivé ou désinstallé.

L'objectif va donc être à chaque ajout de site sur le programme d'automatiser le déploiement du site en question. Pour ajouter un site web, l'attaquant doit fournir une archive au format *.tar.gz* crée au moyen de la commande `zip` du programme. Dans cette archive, l'utilisateur doit fournir ses pages web au format PHP et éventuellement des assets. Tous les formulaires qui doivent être utilisés pour voler des identifiants doivent pointer en POST vers un fichier `login.php` qui est un script de connexion qui sera automatiquement ajouté au dossier du site par le programme. Sur la Figure 5 sont résumées les étapes du processus d'ajout de site web.

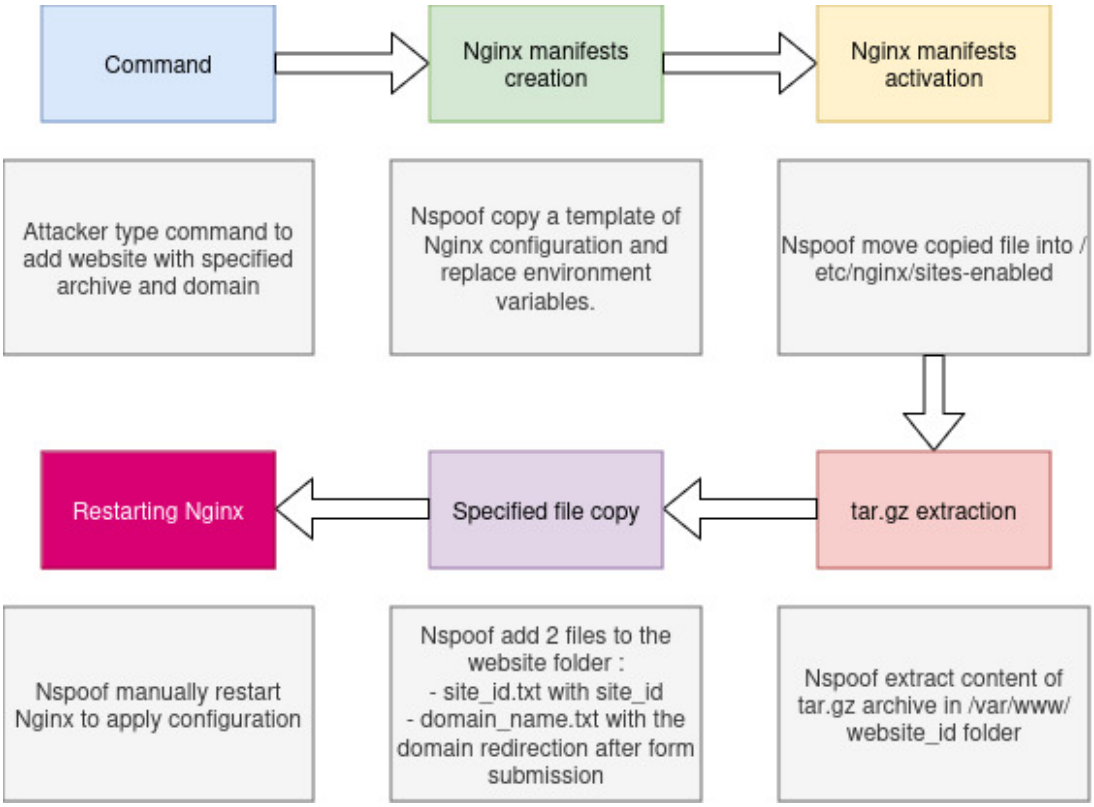

Figure 5: Processus d'ajout de site web

7 Récupération des identifiants soumis

Notre programme embarque également un second serveur web en charge de la récupération des identifiants. En réalité, la récupération des identifiants est articulée en 2 modules distincts : Le fichier `login.php` présent dans le dossier de chaque site internet, contenant le script de traitement des données du formulaire, et le serveur web de catching qui est conçu pour recevoir un triplet `site_id, ip_victim, credentials`.

En réalité, c'est le script `login.php` qui contient toute la logique de traitement des données envoyées en POST depuis le formulaire. Et c'est ce script de traitement qui se charge de formaliser les informations recues pour les intégrer au triplet vu précédemment et les envoyer au catcher.

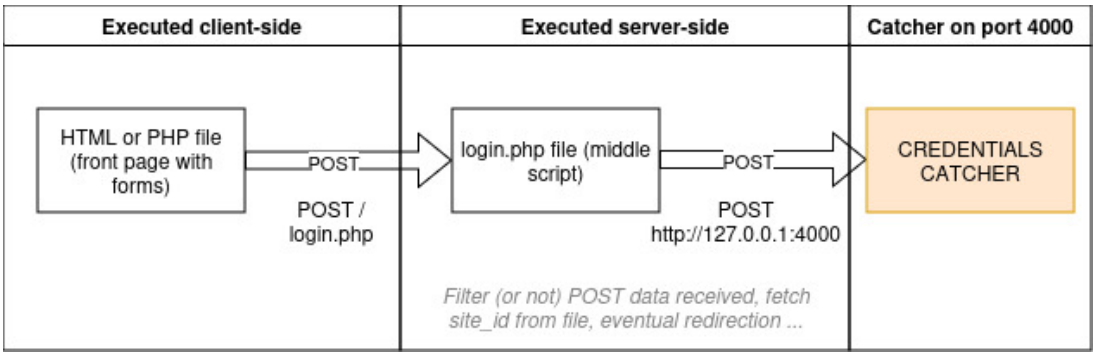

Figure 6: Processus de récupération des identifiants

8 Contre-mesures actuelles et ouverture sur des attaques plus efficaces

8.1 Contre-mesure traitée : Certificats SSL

8.1.1 Contre-mesure

Une grande contre-mesure de ce type d'attaque reste l'arrivée des certificats SSL, authentifiant (pour certains d'entre-eux, nous en reparlerons après) le serveur visité par le Client, que cela soit un site internet ou un autre service.

Pour authentifier un serveur auprès d'un Client, un certificat de sécurité doit être **signé** par une **authorité de certification** de confiance pour le Client. D'un point de vue du web, la majorité des navigateurs comme Chrome embarque une liste par défaut d'autorités de certifications auquel ils font confiance. Un certificat SSL correct et signé par une autorité de confiance se traduit visuellement par l'apparition d'un petit cadenas vert pour le Client.

Au contraire, lorsque le navigateur reçoit un certificat SSL auto-signé ou signé par une autorité auquel le navigateur ne fait pas confiance, cela se traduit bien souvent par des alertes, car bien qu'un certificat comme celui-ci, même correctement généré, puisse assurer une couche de **chiffrement**, il n'assure absolument pas l'authenticité du serveur.

Il va de soit que lors d'une connexion HTTP (sans surcouche TLS), un avertissement apparaît également sur le navigateur du client. Nous avions initialement pensé générer un certificat SSL auto-signé pour chaque site à usurper, mais après différents tests, nous nous sommes rendus-comptes que les avertissements en cas de certificat auto-signé était plus dissuasifs que ceux affichés dans le cas d'une connexion HTTP en clair.

8.1.2 Action de réponse

Pour réponse à cette problématique, nous avons nous-même développé un système de génération de certificats SSL et d'autorité de certification. De cette manière, la commande `gen_ca` permet de générer une autorité de certification Nspoof ROOT, autorité qui sera utilisée pour signer les certificats générés à chaque déploiement de site en HTTPS activé.

Lors de l'activation du HTTPS pour le déploiement d'un site (flag 1 sur la commande), le programme utilise un template de configuration nginx adapté au HTTPS, et spécifie les informations du certificat généré. Toutes les connexions HTTP sont également redirigées vers du HTTPS.

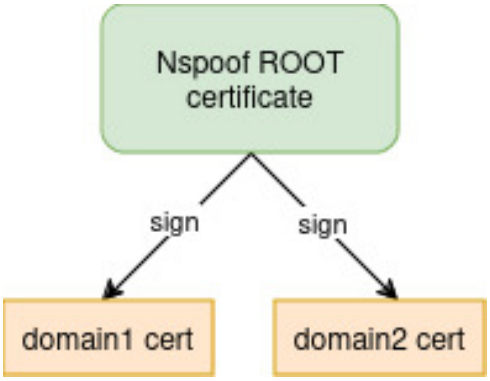

Figure 7: Chaîne de confiance

8.1.3 Limite de la solution proposée

La plus grande limite de cette solution est la nécéssité absolue d'avoir un accès administrateur aux machines des victimes, pour installer l'autorité de certification root en tant qu'autorité de confiance. Cette contrainte est très grande, et on peut préconiser qu'avoir un tel accès est très complexe. Mais dans le cas ou cet accès est possible, l'attaque devient redoutable. Nous avons fait l'expérience (**avec le consentement de participants**) : Nous avons installé l'autorité de certification sur les navigateurs d'étudiants en RSR qui avaient laissé leur ordinateur dévérouillé. Le résultat est très impressionnant. En usurpant le CAS, les étudiants se connectent et l'illusion est parfaite, aucun système ne permet de détecter la fraude, sauf en allant explicitement inspecter le certificat. De cette manière, nous avons pu récupérer un très grand nombres d'identifiants (factices).

8.2 Contre-mesure non traitées : Mécanismes d'isolation réseau

Concernant la sécurité des réseaux en eux-mêmes, là encore plusieurs mécanismes de sécurité peuvent intervenir comme contre-mesures à l'attaque mise en oeuvre, sur certaines installations. Dans le cas des réseaux câblés, on peut observer la mise en place de VLans, réseaux virtuels, permettant de compartimenter le réseau. Ce type de mesure réduit donc la portée de l'attaque, la rendant voire même impossible (à moins d'avoir un accès physique à certains switchs).

Pour les réseaux Wifi, un mécanisme similaire peut être observé, et il est malheureusement (pour notre projet bien-sûr) beaucoup plus courant : L'isolation des pairs. En effet,

un certain nombre de réseaux Wifi et une très grande majorité de réseaux publics isolent chaque pair qui se connecte au point d'accès. Une attaque de ce type est donc totalement impossible dans ce cas.

8.3 Des attaques plus efficaces

Une attaque plus efficace dans le cas des réseaux Wifi pourrait être la création d'un faux point d'accès avec portail captif. Cette attaque a il nous semble été implémentée par un second groupe de projet dans la promo RSR. Cette solution permet de contrer l'isolation des pairs mentionnée plus haut, trompant les Clients et les incitant à se connecter eux-mêmes sur un réseau que l'on contrôle. Une variante de cette attaque, l'attaque *Evil Twin*, désigne la réalisation d'un faux réseau Wifi, mais reprenant les mêmes informations SSID qu'un réseau authentique à proximité.

Pour conclure cette partie, une amélioration de l'attaque implémentée dans ce projet est de trouver un moyen d'installer une autorité de certification de confiance sur nos victimes, et d'usurper nos sites en générant et signant nos certificats au moyen de cette CA. Cependant, cela nécéssite un accès aux machines de nos victimes ce qui rend alors l'attaque beaucoup plus complexe à mettre en oeuvre. Ceci étant dit, l'attaque est alors beaucoup plus efficace et il est impossible pour un individu n'ayant pas de connaissances en sécurité informatique de détecter l'attaque. Il faut aller inspecter manuellement le certificat pour la détection.

9 Conclusion

En conclusion, ce projet nous a permis de réaliser une attaque, mais surtout de comprendre plus en profondeur le fonctionnement du protocole DNS et des certificats de sécurité en général. Cette démarche nous a également sensibilisés aux vulnérabilités au niveau DNS et à l'importance d'appliquer des mesures de sécurité robustes pour prévenir ce type d'attaques. Elle souligne également le rôle crucial de l'éducation et de la recherche en cybersécurité pour anticiper les menaces et développer des solutions innovantes, car nous voyons que des mécaismes de sécurité sont inventés chaque année pour prévenir ce type de menace. Ce projet, bien que réalisé dans un cadre expérimental, renforce notre compréhension des enjeux actuels liés à la protection des données et à la sécurisation des communications sur internet.