

APT31 : Pakdoor

SYNTHÈSE TECHNIQUE

TLP:WHITE



Version : 2.1
Date d'enregistrement : 2021-12-15
Nombre de pages : 16

Table des matières

1	Introduction	3
2	Compromission initiale	4
3	Lanceur	5
4	Porte dérobée	6
4.1	Configuration	6
4.2	Communication	7
4.2.1	Initialisation	7
4.2.2	Gestion des pairs	7
4.2.3	Gestion des entrées et sorties	7
4.2.4	Structure des messages	8
4.3	Tâches	9
4.3.1	Découverte des pairs	9
4.3.2	Relai de trafic	10
4.3.3	Administration	13
5	Conclusion	14
A	Empreintes	15
A.1	Porte dérobée	15

1 Introduction

Depuis début 2021, plusieurs rapports ont fait état de la compromission de routeurs par le mode opératoire d'attaque (MOA) APT31, notamment ceux développés par l'entreprise Pakedge¹. Ces routeurs compromis sont utilisés par le MOA pour construire son infrastructure de commande et contrôle (C&C). Plus récemment, il a été observé que d'autres marques d'équipements réseau comme Cyberoam et Cisco sont également ciblées par le MOA.

Afin d'administrer les routeurs compromis et pour les faire communiquer entre eux, APT31 installe une porte dérobée assez sophistiquée sur ces derniers. Comme celle-ci n'était pas connue en source ouverte, nous l'avons nommée **Pakdoor**. La fonction principale de cette porte dérobée est de fournir une infrastructure d'anonymisation dédiée au MOA.

Plusieurs observations ont montré que les attaquants utilisent cette infrastructure à la fois pour faire de la reconnaissance et également pour communiquer avec des implants installés sur des machines victimes. De plus, il a pu être observé que les routeurs compromis sont divisés en différentes grappes, où un routeur appartenant à une grappe peut uniquement communiquer avec des routeurs de la même grappe.

Ce document décrit le fonctionnement de Pakdoor et de ses différents composants, et apporte ainsi une vision plus précise d'une des infrastructures utilisées par ce MOA.

1. <https://www.pakedge.net>

2 Compromission initiale

Lorsqu'un routeur est compromis, les attaquants déposent au moins trois fichiers sur le système de fichiers :

- un exécutable ELF (développé avec le langage C) qui implémente la plupart des fonctionnalités de Pakdoor, ci-après dénommé **porte dérobée**;
- un script Bash qui assure la persistance de la porte dérobée et qui fournit également d'autres fonctionnalités à cette dernière, ci-après dénommé **lanceur**;
- une configuration chiffrée contenant les pairs avec lesquels la porte dérobée peut communiquer.

3 Lanceur

Ce script est exécuté par le script `/etc/rc.local`, qui peut exécuter des commandes après que les services système aient été chargés lors du démarrage de la machine.

L'objectif de ce script est d'assurer la persistance de la porte dérobée, mais également de fournir des fonctionnalités supplémentaires (voir la Table 1) à cette dernière, comme la capacité d'ouvrir des ports sur le pare-feu local (`Netfilter` dans le cas présent). De plus, deux certificats et une clé privée sont stockés à la fin du script. Ces derniers sont utilisés par la porte dérobée pour créer des sockets TLS (voir la Section 4.2 pour plus de détails).

Le script prend un ou plusieurs arguments en ligne de commande. Le premier argument est le nom de la commande à exécuter (six sont implémentées), et dépendamment de celle-ci, un second argument peut être attendu.

Egalement, deux chemins sont écrits au début du script dans les variables suivantes :

- `file_name`, chemin vers la porte dérobée;
- `port_file`, chemin vers un fichier texte contenant les numéros de port utilisés par la porte dérobée pour relayer le trafic.

La Table 1 résume les différentes commandes implémentées dans le script.

Commande	Description
<code>add</code>	Ajoute une règle <code>iptables</code> pour autoriser le trafic réseau vers le numéro de port donné en second paramètre.
<code>del</code>	Supprime la règle <code>iptables</code> autorisant le trafic réseau vers le numéro de port donné en second paramètre. Si ce dernier n'est pas spécifié alors toutes les règles correspondant aux ports présents dans le fichier <code>port_file</code> sont supprimées.
<code>wl</code>	Exécute la commande <code>add</code> pour chaque port présent dans le fichier <code>port_file</code> toutes les 20 secondes.
<code>port</code>	Si le port donné en second paramètre n'est pas présent dans le fichier <code>port_file</code> , alors la commande <code>add</code> est exécutée avec ce port en paramètre et ce dernier est ajouté dans le fichier <code>port_file</code> .
<code>ost</code>	Ferme les ports présents dans le fichier <code>port_file</code> avec la commande <code>del</code> , exécute la porte dérobée, exécute la commande <code>wl</code> , attend que l'exécution de la porte dérobée se termine, supprime le répertoire courant si le fichier <code>port_file</code> n'existe pas, termine le processus correspondant à l'exécution du script avec la commande <code>wl</code> , attend cinq secondes et termine l'exécution du script.
<code>st</code>	Exécute la commande <code>ost</code> . Il semblerait qu'il s'agisse du point d'entrée du script puisque c'est cette commande qui est exécutée dans le script <code>/etc/rc.local</code> .

Table 1 – Commandes implémentées par le lanceur

4 Porte dérobée

La porte dérobée prend un paramètre en ligne de commande qui correspond à un numéro de port (voir la Section 4.2 pour son utilisation).

4.1 Configuration

La configuration de la porte dérobée est lue et déchiffrée depuis le fichier `conf` (présent dans le même répertoire que la porte dérobée). Dépendamment de la version de l'implant, la configuration est chiffrée avec l'algorithme ChaCha20 (version 2019) ou ChaCha20-Poly1305 (version 2021). La clé de chiffrement est inscrite dans la porte dérobée.

La configuration chiffrée correspond à la structure détaillée sur la Figure 1. Bien entendu, le champ `tag` est uniquement présent si l'algorithme utilisé est ChaCha20-Poly1305.

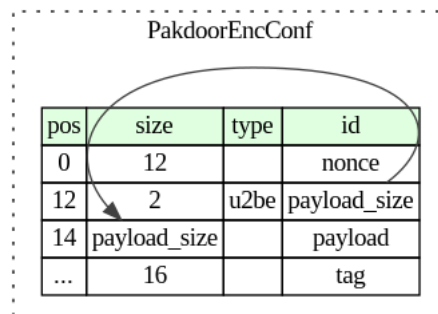


Figure 1 – Structure de la configuration chiffrée

La taille de la charge utile est présente dans la configuration chiffrée (champ `payload_size`) mais elle est également écrite en dur dans la porte dérobée. Dans tous les échantillons analysés, la taille présente dans le code est égale à 372 octets et elle n'est pas comparée à la taille présente dans la configuration chiffrée.

La configuration déchiffrée contient essentiellement les pairs avec lesquels la porte dérobée peut communiquer initialement. Un implant ne peut pas avoir plus de dix pairs à un moment donné.

4.2 Communication

Les noeuds (routeurs compromis) de l'infrastructure communiquent entre eux via des sockets TLS où chaque côté doit s'authentifier avec un certificat valide signé par une autorité de certification créée par le MOA. Chaque noeud ouvre une socket TLS en écoute, afin de recevoir des tâches envoyées par d'autres noeuds de l'infrastructure.

4.2.1 Initialisation

Au début de l'exécution, la porte dérobée lit le lanceur pour extraire les certificats et la clé privée. Il y a un certificat pour l'autorité de certification (AC) et un second (ci-après dénommé **certificat du noeud**) utilisé pour authentifier le noeud courant auprès des autres noeuds de la grappe. Le certificat du noeud est signé par l'autorité de certification et la clé privée est celle correspondant au certificat du noeud. D'après ce qui a été observé, il y a une autorité de certification par grappe de routeurs compromis afin d'assurer la segmentation entre les grappes. Ces dernières sont identifiées par le nom commun de l'émetteur du certificat.

Ces certificats sont ensuite utilisés pour créer une socket TLS en écoute sur le port passé en ligne de commande de la porte dérobée.

4.2.2 Gestion des pairs

Puisque la porte dérobée est susceptible de communiquer avec plusieurs pairs en même temps, une table de hachage est implémentée pour gérer les pairs actifs. Cette même table est également utilisée pour la gestion des processus fils lorsque la porte dérobée est utilisée pour relayer du trafic (voir la Section 4.3.2 pour plus de détails). Une clé dans la table est générée à partir de la clé publique du pair. L'algorithme de hachage utilisé pour calculer l'index à partir de la clé est une variante de l'algorithme djb2².

Afin de gérer les collisions (un même emplacement dans la table de hachage pour deux clés publiques différentes), chaque emplacement de la table est une liste chaînée où chaque noeud est une structure de taille fixe de 482 octets contenant de nombreux détails sur un pair donné (notamment son adresse IP, son port d'écoute, sa clé publique et la dernière fois que ce noeud a été actif).

4.2.3 Gestion des entrées et sorties

La porte dérobée se base sur la bibliothèque `libev` pour gérer les entrées/sorties sur les sockets de communication. La page `man` de `libev`³ est très complète, toutefois les structures `ev_io` et `ev_timer` méritent une attention toute particulière vu leur importante utilisation dans le code.

La structure `ev_io` est utilisée pour enregistrer un *callback* si un ou plusieurs événements se produisent sur un descripteur de fichier donné. Par exemple, une fonction peut créer un *watcher* sur une socket réseau pour l'événement `EV_READ`, ainsi si la socket peut être lue, le *callback* enregistré dans le *watcher* sera appelé. Comme son nom le suggère, la structure

2. <http://www.cse.yorku.ca/~oz/hash.html>

3. <https://linux.die.net/man/3/ev>

ev_timer a un fonctionnement similaire, sauf qu'au lieu d'attendre un événement particulier, le *callback* sera exécuté après une certaine quantité de temps (définie dans le *watcher*). Egalement, un *timer* peut être répété un certain nombre de fois si nécessaire.

La Figure 2 montre comment la porte dérobée utilise les structures ev_io pour enregistrer un *callback* qui lit la socket en écoute (f_read_from_tls_socket_callback()) et une autre qui écrit (f_write_to_tls_socket_callback()) sur cette dernière.

```
// Register write watcher on socket.
p_write_watcher->fd = socket;
p_write_watcher->watcher_list.watcher.pending = 0;
p_write_watcher->watcher_list.watcher.active = 0;
p_write_watcher->watcher_list.watcher.ev_cb_declare = (int)f_write_to_tls_socket_callback;
p_write_watcher->watcher_list.watcher.priority = 0;
p_write_watcher->events = EV_IOFDSET|EV_WRITE;
ev_io_start(p_ev_loop, p_write_watcher);

// Register read watcher on socket.
p_read_watcher = peer_ctx->p_read_watcher;
p_read_watcher->watcher_list.watcher.ev_cb_declare = (int)f_read_from_tls_socket_callback;
p_read_watcher->fd = socket;
p_read_watcher->events = EV_IOFDSET|EV_READ;
p_read_watcher->watcher_list.watcher.pending = 0;
p_read_watcher->watcher_list.watcher.active = 0;
p_read_watcher->watcher_list.watcher.priority = 0;
ev_io_start(p_ev_loop, p_read_watcher);
```

Figure 2 – Vue du décompilateur d'IDA des watchers utilisés pour gérer les entrées/sorties sur les sockets de communication

4.2.4 Structure des messages

Les messages échangés entre les noeuds (via les sockets TLS) respectent une structure spécifique décrite sur la Figure 3.

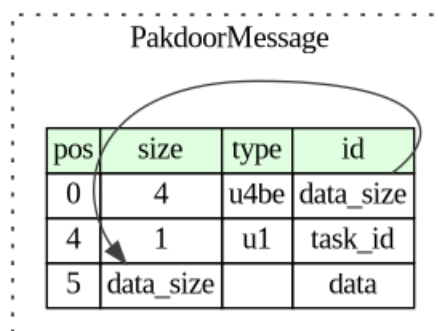


Figure 3 – Structure des messages échangés entre les noeuds

Chaque message débute par un en-tête de cinq octets contenant l'identifiant de la tâche que le noeud destinataire doit exécuter ainsi que la taille des données additionnelles si nécessaire. Ces dernières sont envoyées au noeud destinataire immédiatement après l'en-tête.

4.3 Tâches

La porte dérobée peut exécuter différentes tâches pour découvrir et gérer ses pairs, relayer les communications des opérateurs et pour administrer le routeur compromis.

4.3.1 Découverte des pairs

Avant d'être capable d'exécuter toute autre tâche, la porte dérobée initie une communication vers le premier pair valide du fichier de configuration. Ce processus de découverte des pairs a deux fonctions principales :

- être reconnu comme un pair valide par les noeuds présents dans la configuration ;
- récupérer les pairs actifs des noeuds existants.



Comme la porte dérobée accepte uniquement les tâches envoyées par les noeuds présents dans sa liste de pairs actifs, ce processus est crucial pour que l'implant fonctionne correctement.

La Figure 4 illustre le déroulement d'un processus de découverte des pairs réussi.

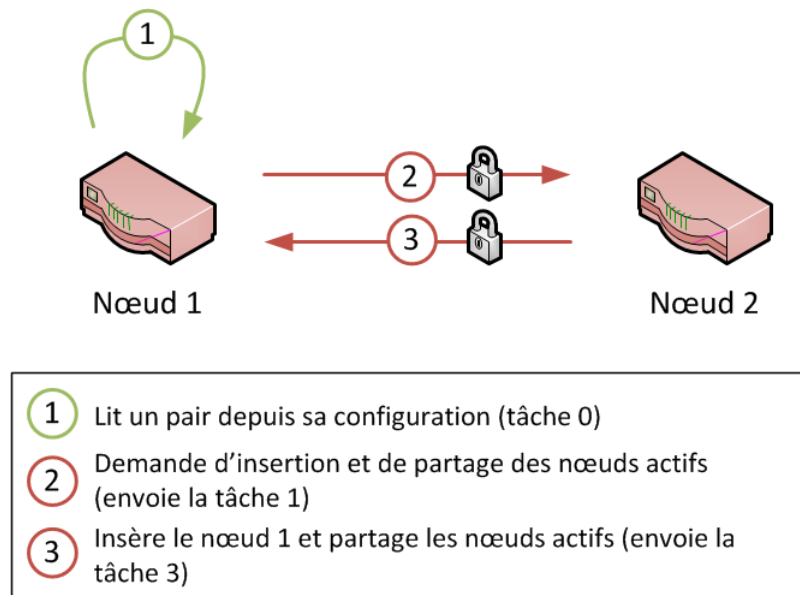
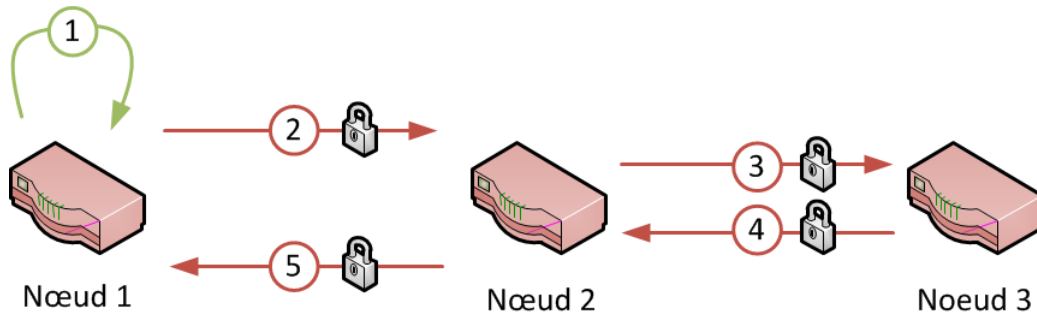


Figure 4 – Processus de découverte des pairs réussi

Quand le noeud 2 (voir Figure 4) exécute la tâche #1, il effectue plusieurs vérifications avant d'insérer le noeud 1 dans sa liste de pairs actifs.

1. La somme de contrôle de la clé publique dans les informations (envoyées avec la tâche #1) du noeud 1 doit correspondre à la somme de contrôle calculé à partir de la clé publique présente dans ces mêmes informations. Si ce n'est pas le cas, le noeud 1 est simplement ignoré.
2. La date de dernière activité du noeud présente dans les informations du noeud 1 ne doit pas être plus vieille que trois heures. Sinon, le noeud 2 envoie la tâche #2 au noeud 1 pour s'assurer qu'il est toujours actif. Si c'est le cas, ce dernier répondrait avec la tâche #4 pour indiquer qu'il est toujours actif et il serait finalement accepté comme un pair valide.

3. La liste des pairs actifs ne doit pas contenir plus de dix noeuds. Auquel cas, le noeud 2 (voir la Figure 5) envoie la tâche #2 au premier pair de sa liste (noeud 3) pour vérifier s'il est toujours actif. Si ce dernier répond avec la tâche #4, alors le noeud 1 est ignoré, sinon il est accepté comme pair valide et inséré dans la liste. Dans les deux cas, le noeud 2 partage sa liste de pairs actifs avec le noeud 1.



- | | |
|---|---|
| 1 | Lit un pair depuis sa configuration (tâche 0) |
| 2 | Demande d'insertion et de partage des noeuds actifs (envoie la tâche 1) |
| 3 | Vérifie si le noeud est toujours actif (envoie la tâche 2) |
| 4 | Indique que le noeud est toujours actif (envoie la tâche 4) |
| 5 | Partage ses noeuds actifs (envoie la tâche 3) |

Figure 5 – Processus de découverte des pairs si le pair a déjà dix noeuds actifs

Pour les cas 2 et 3, la porte dérobée continue son exécution en lisant le pair suivant dans la configuration et répète le processus jusqu'à ce que sa liste de pairs actifs contienne dix noeuds.

4.3.2 Relai de trafic

La fonctionnalité principale de la porte dérobée est de pouvoir fournir la capacité de relayer du trafic d'un noeud à un autre. Les opérateurs peuvent définir n'importe quelle chaîne de relai à partir du moment où deux noeuds consécutifs se sont acceptés mutuellement.

Une chaîne de relai de trafic est composée de deux types de noeuds :

- un ou plusieurs noeuds de relai qui transmettent le trafic de manière transparente ;
- un seul noeud de sortie qui chiffre ou déchiffre le trafic avec une clé de session et le transmet.

Le nombre de noeuds dans une chaîne de relai peut varier d'un noeud (un seul noeud de sortie) à quatre noeuds (trois noeuds de relai et un noeud de sortie).

Le cycle de vie d'une session de relai de trafic peut se diviser en trois phases distinctes. Chacune correspond respectivement aux couleurs rouge, bleu et noir sur la Figure 6.

1. Pour créer une session de relai de trafic, les opérateurs envoient la tâche #5 au premier noeud. Les données additionnelles pour cette tâche correspondent aux configurations des différents noeuds de la chaîne. Si plusieurs configurations sont présentes, le premier noeud retire sa propre configuration de la liste et envoie les configurations restantes au noeud suivant, et ainsi de suite jusqu'au dernier noeud. Chaque configuration est chiffrée avec la clé publique du noeud destinataire. Une fois la configuration reçue, chaque noeud crée une socket UDP en écoute sur un port arbitrairement choisi par le système d'exploitation pour gérer la session de relai (le port est ouvert sur le pare-feu local avec la commande `port` du lanceur). La porte dérobée répond avec la tâche #6 pour indiquer le port d'écoute de la socket UDP au noeud précédent.
2. Les opérateurs initient la session de relai en envoyant un message spécifique sur la socket UDP en écoute du premier noeud de la chaîne. Ce dernier transmet le message au noeud suivant, et ainsi de suite jusqu'au noeud de sortie. Une fois que ce processus est terminé, le noeud de sortie répond avec un message spécifique pour indiquer qu'il est prêt à relayer le trafic.
3. Les opérateurs envoient du trafic chiffré au premier noeud de la chaîne. Dépendamment de la version de la porte dérobée, le trafic est soit chiffré avec l'algorithme ChaCha20 (version 2019) ou ChaCha20-Poly1305 (version 2021). La clé de session utilisée pour chiffrer le trafic est uniquement détenue par les opérateurs et le noeud de sortie.

Avant de créer une session de relai, la porte dérobée se duplique avec l'appel système `fork` afin que le processus parent puisse continuer à exécuter d'autres tâches. Le PID du processus fils est stocké dans la table de hash utilisée pour la liste des pairs actifs. La clé correspondant à cet emplacement est le port d'écoute de la socket UDP utilisée pour gérer la session de relai. En raison de cette parallélisation, un noeud donné peut être utilisé dans plusieurs chaînes de relai et peut avoir différents rôles. Par exemple, un noeud donné peut être un noeud de relai dans une chaîne A et un noeud de sortie dans une chaîne B.

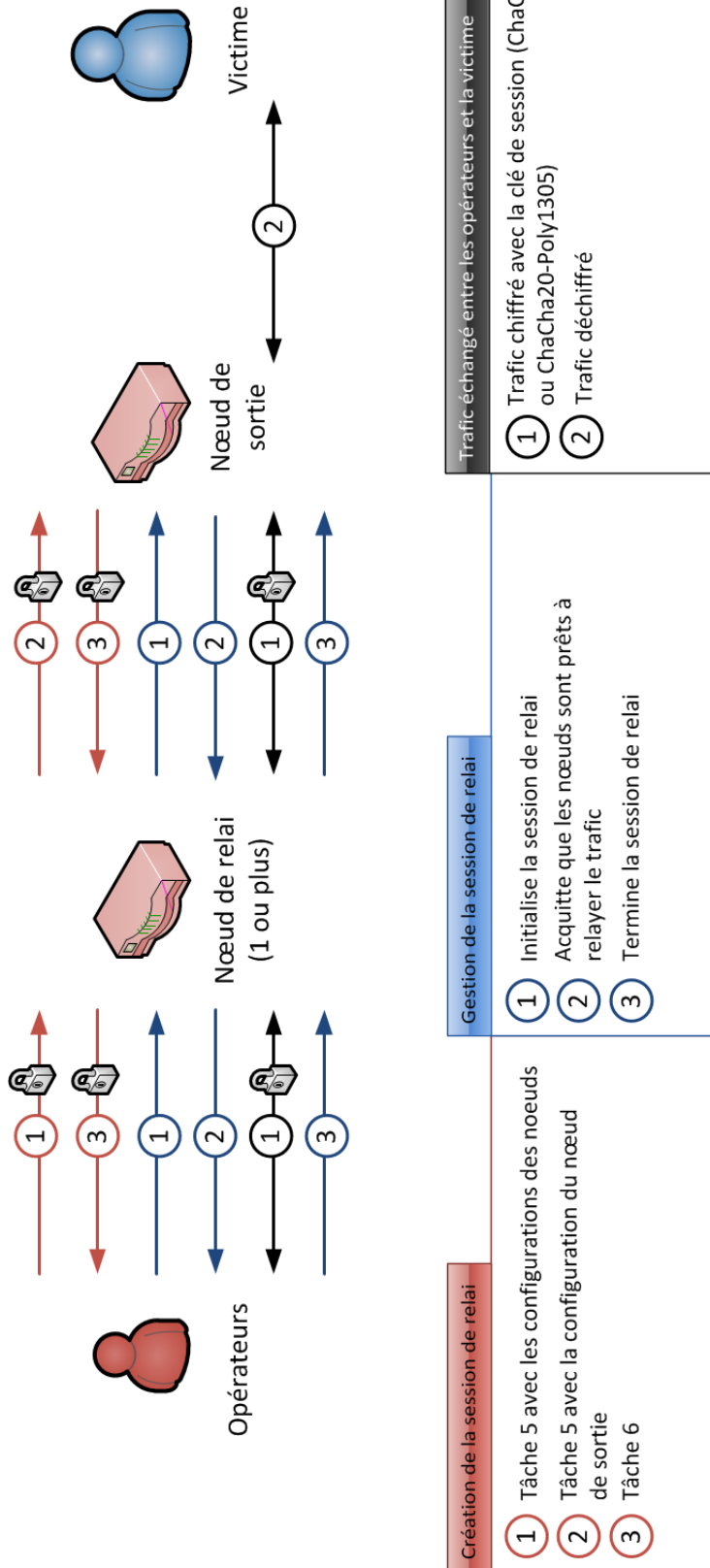


Figure 6 – Vue d’ensemble du cycle de vie d’une session de relais

4.3.3 Administration

Les opérateurs peuvent gérer les différents noeuds de l'infrastructure en utilisant les tâches #7 et #8. La première permet d'exécuter des commandes de porte dérobée assez standard comme :

- lire ou écrire un fichier;
- exécuter une commande *shell*;
- récupérer la configuration active.

Afin d'assurer l'intégrité du contenu de la charge utile, le CRC-64 de cette dernière est vérifié avant d'exécuter la commande. De plus, cette somme de contrôle est signée avec la clé privée de l'autorité de certification (voir la Figure 7), ainsi, même si une personne est capable de contrôler un noeud de l'infrastructure, il ne peut pas envoyer de commandes à un autre noeud sans connaître la clé privée des opérateurs.

```
// Verify checksum signature.
if ( mbedtls_pk_verify(
    &p_task->p_main->p_cert_chain->p_ca_cert->pk,
    0,
    p_cmd_data->payload_checksum,
    15,
    p_cmd_data->checksum_sig,
    256) )
{
    return -1;
}
f_inverse_data_endianness(p_cmd_data);

// Verify checksum.
checksum = f_compute_crc64_checksum(p_cmd_data->payload, p_cmd_data->payload_size);
if ( memcmp(p_cmd_data->payload_checksum, &checksum, 8) )
    return -1;
```

Figure 7 – Vue du décompilateur IDA de la vérification de la signature de la somme de contrôle de la charge utile

5 Conclusion

Voici les points essentiels résumant l'analyse détaillée dans ce rapport.

- Cet implant est utilisé depuis au moins 2019 (date de la soumission d'un échantillon sur VirusTotal) par APT31.
- Le but de code est de construire une infrastructure pair à pair où chaque pair doit s'authentifier avec un certificat TLS pour communiquer avec un autre pair.
- Le cycle de vie de l'infrastructure est automatisé via l'utilisation de tâches planifiées, que ce soit pour la découverte des pairs ou pour s'assurer que les pairs sont toujours actifs (si ce n'est pas le cas, la porte dérobée s'auto-supprime).
- Les opérateurs utilisent l'infrastructure résultante pour établir des chaînes de *proxy* (de un à quatre noeuds) qui supportent les protocoles TCP, UDP et *raw* pour communiquer avec la victime. De plus, le *fork* de processus permet à la porte dérobée de gérer plusieurs chaînes de *proxy* en parallèle.
- L'implémentation de commandes d'administration (lecture/écriture de fichiers et exécution de commandes *shell* par exemple) permet aux opérateurs de gérer facilement les routeurs compromis. De plus, les opérateurs authentifient leurs commandes en signant la somme de contrôle de la charge utile de la commande avec la clé privée de l'autorité de certification.

A Empreintes

A.1 Porte dérobée

MD5	77c73b8b1846652307862dd66ec09ebf
SHA1	cadf644815f758b78774c1285245e9be13b098fe
SHA256	1d60edb577641ce47dc2a8299f8b7f878e37120b192655aaf80d1cde5ee482d2
Size in bytes	509952
Commentaire	Disponible sur VirusTotal

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

ANSSI - 51, boulevard de la Tour-Maubourg - 75700 PARIS 07 SP
www.ssi.gov.fr



Premier ministre

