Durant l'implémentation de notre malware on a mis en places plusieurs principes:

Stockage du hashé de la clé:

L'algorithme de hashé qu'on a choisi est l'algorithme SHA-256 qu'on a implémenté.
On a d'abord regardé comment on pourrait faire pour stocker notre hashé de notre clé et la cacher suffisamment pour qu'on ne puisse pas la trouver juste en regardant les différentes data. On s'est donc dit qu'on allait le cacher au milieu d'autres hashé du même type et qui sont valides.
On a donc 100 hashé de clé qui sont stockés dans notre programme et notre clé hashé est l'une d'entre elles.
On s'est ensuite dit que même si elle est au milieu de plein de hashé ce serait bien de la chiffrer, de la modifier un peu de manière réversible dans le programme et de sorte qu'elle ressemble encore un peu à un hash de SHA-256 pour pas qu'elle soit visible dans le paquet de hash.

Algorithme de chiffrement du hashé de la clé:

Le hashé de la clé est donc stocké après avoir été passé par ces instruction:
for(int i=0; i<64; i++) cle[i] += 1;
cle[30] -= 1; // -1 car sinon notre hashé modifié contient un caractère spéciale et c'est pas très discret
Dans le malware on a cette suite d'instruction inverse pour récupérer notre hashé de la clé non modifié:
for(int i=0; i<64; i++) cleMod[i] -= 1;
cleMod[30] += 1;

Anti-débogage:

On a utilisé l'octet 140 dans ppeb comme méthode d'anti débogage cela a pour avantage d'être moins voyant qu'un IsDebuggerPresent() ou autre fonction pour vérifier qu'un débogueur soit présent. Ou encore mieux qu'un mov eax, fs:[0x30] que certains débogueur reconnaissent directement.

Par exemple:

```
PEB *ppeb;
char *p = (char *) ppeb;
if(p[104] & 0x70){
        printf("%s", arg);
}
else{
        …
```

Attraper un pointeur sur différentes fonctions utiles:

En partant de l'adresse de printf on parcourt le dll jusqu'à ce qu'on trouve un endroit qui correspond au début de scanf. Cela nous permet d'avoir un pointeur sur scanf sans utiliser scanf.
On utilise la même méthode en partant de notre nouveau pointeur sur scanf pour récupérer un pointeur sur strcmp. C'est ce pointeur qu'on utilisera pour comparer l'arg hashé et notre clé hashé.

Automodification de scanf (en passant par notre pointeur) en printf:

Le but ici est de modifier l'instruction du début de scanf pour nous faire sauter sur printf ainsi quand on appelle scanf c'est printf qui est exécuté.
On utilise donc notre pointeur sur scanf qu'on a trouvé avant afin de brouiller un peu les piste.
Le résultat de ce principe et de celui d'avant est qu'on utilise notre pointeur sur scanf au lieu d'un appel à printf pour afficher sur la console !

Utilisation de leurres:

On utilise plusieurs leurres autour de notre vrai chemin qui calcul et compare à notre clé hashé. Il y a des leurres qui comparent avec les mauvais hashé et qui font bien appelle au string "Bravo !" mais renvoie juste l'argument si on arrive à trouver le claire du hashé.
On a d'autres leurres qui pourraient bien renvoyer "Bravo !" si on trouvait le claire du faux hashé mais on a utilisé du call stack tempering pour que l'adresse de retour saute ces instructions.

Call stack tempering:

On a donc des fonctions qui ont eu leur adresse de retour modifier avec le ret afin d'esquiver certaines instructions comme un printf("Bravo !") avec while(1) et return 0;

Probleme de groupe:

Christian VASAUNE et moi Thomas BIGEL avons travaillé du mieux qu'on pouvait mais Matthieu PERRIGOT n'as rien fait du tout concernant la création du malware et donc cette partie du projet.