Labo Sécurité des Systèmes d'Information

Rendu de projet

Quentin Cassaigne
Mathéo Bayle

Sommaire

Introduction

Pour notre deuxième année dans le labo SSI, on n'avait pas un projet précisément en tête que l'on voulait faire, donc, comme pour l'année dernière on a choisi un projet (peut-être ambitieux) dans lequel on avait aucune connaissance, la création d'un malware, plus spécifiquement un keylogger, dans un langage où l'on en avait encore moins, l'assembleur.

Mais qu'est-ce qu'un keylogger, au juste ?

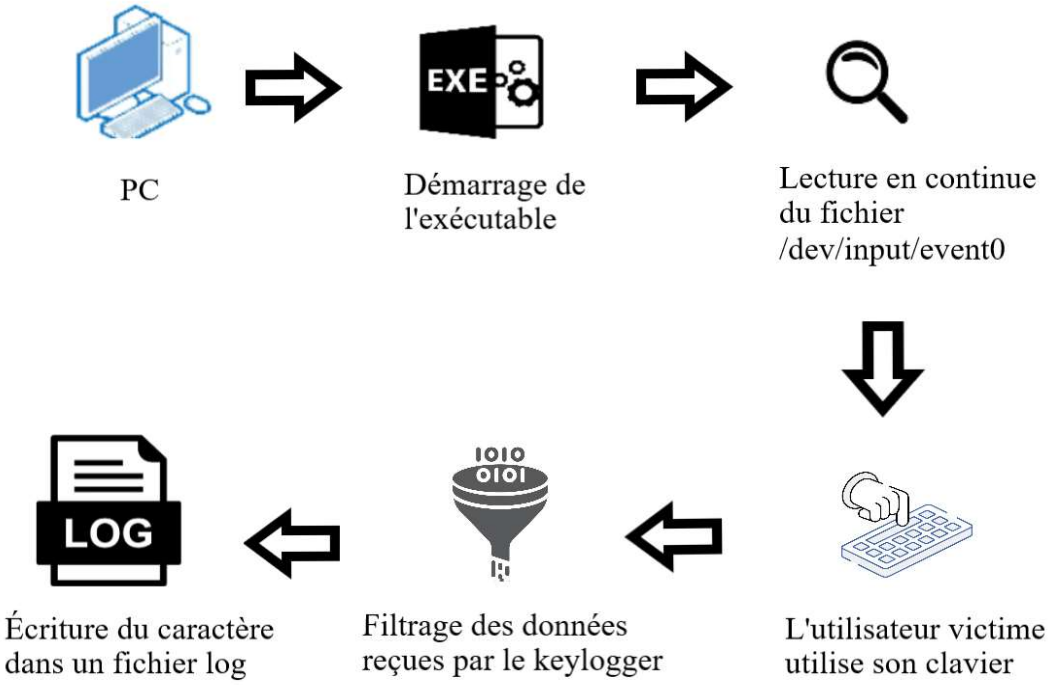
Il s'agit essentiellement d'un programme conçu pour enregistrer chaque frappe de clavier effectuée sur un système. Ce type d'outil est à la fois puissant et malveillant dans le domaine de la cybersécurité, car il peut être utilisé pour capturer des informations sensibles telles que des mots de passe, des numéros de carte de crédit et d'autres données confidentielles.

Le langage de programmation que nous utiliserons est donc l'assembleur plus précisément NASM qui supporte l'architecture 64 bits et x86_32 bits et qui utilise la syntaxe Intel.

Notre projet va donc consister à explorer le fonctionnement des malwares avec une approche en détails grâce à l'utilisation de l'assembleur et la façon dont le système d'exploitation interprète les signaux qu'il reçoit des périphériques, dans notre cas le clavier, pour les transférer ensuite vers les applications qui en ont besoin.

Pour cela, on va notamment utiliser les fichiers 'events' dans le répertoire /dev/input sous Linux qui représentent les différents périphériques d'entrée connectés au système et dont chaque fichier 'event' va correspondre à un périphérique d'entrée spécifique (souris, clavier etc…).

Schéma logique



Justification projet

Pour notre projet de l'année précédente, nous avons utilisé python comme langage de programmation de haut niveau pour recréer des outils existants de pentesting, donc, on avait envie pour cette deuxième année d'explorer une autre partie de la programmation avec un langage de bas niveau, et notre choix s'est porté sur l'assembleur qui est un langage emblématique et comme étant le plus proche du langage machine et que nous allons utiliser pour la création du malware.

On a choisi comme sujet de programmation un malware parce que c'est un vecteur d'attaque qui est de plus en plus présent dans les attaques d'aujourd'hui, avec notamment les ransomwares, et qui est de plus en plus perfectionné.

État de l'art

Des connaissances sur le fonctionnement des malwares, les différentes techniques d'obfuscation pour que le malware soit exécuté sans éveiller de soupçons.

Connaissance sur ce qu'est un langage de bas niveau avec l'assembleur, comment on exécute un code en assembleur, les manières d'optimisation en performances de notre code.

Des connaissances sur la manière dont Linux va récupérer les entrées claviers et communiqué ces événements aux applications qui en ont besoin.

Mise en place du projet

Le début du projet a été plus difficile que prévu, on a commencé à regarder un peu partout des documentations sur Internet pour commencer à apprendre les bases de l'assembleur et on est tombé sur un très bon cours sur HackTheBox (merci Sasha :)), qui reprend toutes les bases à connaître pour débuter la programmer en assembleur.

Cependant, on a rapidement remarqué en parallèle de notre avancement sur l'assembleur que créer un cryptominer exclusivement dans ce langage allait être plus difficile que prévu (même beaucoup plus) donc après quelques idées et essais infructueux, on a décidé de changer de type de malware et de passer vers un keylogger qui nous paraissait plus abordable non sans être facile.

Avant de réellement commencer à créer notre keylogger, on a réalisé plusieurs petits programmes, comme la création et l'écriture dans un fichier, pour nous familiariser avec l'assembleur et le débuggage avec gdb.

Problèmes rencontrés / Solutions apportées

- Manque de connaissances en langage de bas niveau donc un ralentissement en début de projet notamment sur le choix du malware et le fait de ne pas trop savoir par où commencer.

- Notre premier code qui utilisait l'appel système 'read' pour capturer les frappes de clavier, mais ne fonctionnait plus lorsque l'on sortait de l'endroit où s'exécutait le code, donc on a utilisé les fichiers d'événements /dev/input/eventX.

- Le filtrage des données reçues dans notre deuxième code posait un problème parce qu'on les recevait de manière brute et que l'on ne pouvait pas les stockés dans un fichier.

- Quelques autres problèmes mineurs comme la gestion du file descriptor etc...

Ouverture du projet

Notre projet peut avoir plusieurs axes de continuité comme l'intégration du malware dans un fichier exécutable, un PDF ou autre.

On pourrait aussi procéder à différentes méthodes d'obfuscation de code comme rajouter du code qui ferait des opérations sans intérêt pour complexifier sa compréhension mais aussi changer le nom des fonctions ou encore chiffrer les chaînes de caractères en clair comme celle qui indique l'endroit où est sauvegardé le fichier des logs.

Axes d'amélioration

- Traduire toutes les touches du clavier (ne prends pas en compte les touches maj, ctrl ...).

- L'obfuscation du code pour le rendre plus difficile à comprendre pour quelqu'un qui voudrait faire de la rétro-ingénierie dessus.

- Utiliser un autre moyen que le fichier 'event' parce que l'on a besoin des droits du superUtilisateur pour lire ce fichier.

- Quelques changements mineurs comme la possibilité de demander à l'utilisateur quel fichier 'event' correspond à celui qui s'occupe des entrées du clavier parce que le numéro change en fonction des distributions de Linux.

Conclusion

Pour conclure, ce projet nous aura permis de faire un premier pas dans le monde des malwares qui est vaste mais tout aussi intéressants, tout comme pour la programmation en assembleur qui est un langage qui peut paraître austère aux premiers abords mais qui nous montre sa puissance lorsqu'on commence à le comprendre.

Il nous a aussi appris la patience et la persévérance lorsqu'on bloqué sur un problème complexe et que l'on commence à chercher des documentations dans les bas-fonds d'Internet parce que les solutions des problèmes liés à l'assembleur dans la version que l'on utilise n'apparaissent pas souvent en première page.

Et pour finir, ce projet nous a aussi permis d'en apprendre davantage sur la compréhension des langages de bas niveau, leur structure et comment ils vont échanger avec les différents composants de notre ordinateur pour un fonctionnement normal.

Annexes

Github :

https://github.com/quentin-csg/Projet-SSI-B2

Autres liens :

https://academy.hackthebox.com - Intro to Assembly Language

https://x86.syscall.sh

https://www.kernel.org/doc/Documentation/input/input.txt

https://stackoverflow.com/questions/15949163/read-from-dev-input

https://stackoverflow.com/questions/3662368/dev-input-keyboard-format