CY Tech

Ingénieur 3 Cybersécurité

Rapport

Sécurité Shellcode

Étudiant : David KUSMIDER
Enseignant responsable : Maxime BOURY

Date de soumission : 17 Décembre 2024

1. Les étapes réalisées pour créer l'Infecteur ELF.

1. Vérification si le fichier est bien un exécutable ELF et Ouverture du fichier à infecter.
2. Enregistrement de champs important :
    a. e_entry
    b. e_phoff
    c. e_phentsize
    d. e_phnum
3. À l'aide des informations précédentes, parcours du fichier à la recherche d'un segment PT_NOTE.
4. Conversion du segment PT_NOTE en segment PT_LOAD.
5. Modification de l'adresse de e_entry dans une zone qui n'entrera pas en conflit avec l'exécution du programme d'origine.
6. Changement des droits de la protection mémoire pour le segment afin d'autoriser l'exécution d'instructions.
7. Ajout du payload à la fin du fichier et ajustement de la taille du disque et la taille de la mémoire virtuelle afin de tenir compte du payload injecté.
8. Retour à l'adresse de e_entry d'origine afin de poursuivre l'exécution de l'exécutable d'origine. (Non fonctionnel)

2. Les problèmes rencontrés.

Assembleur

Le plus gros problème que j'ai rencontré était mon manque de connaissances et de maîtrise de l'assembleur. C'est un problème très logique puisque le projet devait être réalisé en assembleur. Initialement, j'avais beaucoup de mal à utiliser l'assembleur pour mener à bien ce projet pour plusieurs raisons. Il y avait mon manque de connaissances en assembleur ainsi qu'un manque de motivation à programmer en assembleur à cause des personnes qui disaient que c'était très compliqué. Donc, je me suis dit que j'allais commencer le projet et apprendre sur le tas. Mais après avoir débuté, sans obtenir de résultats et surtout sans aucune compréhension de ce que je faisais, j'ai décidé de dédier un week-end à l'objectif d'apprendre les bases de l'assembleur et surtout de comprendre à minima ce que je faisais. Ce fut un très bon choix puisque cela m'a permis d'avoir enfin une réflexion sur ce que je faisais, d'identifier des erreurs logiques et stupides. Ayant enfin compris comment programmer en assembleur, je pouvais me lancer pleinement dans le projet. Pour ne pas oublier pas la suite j'ai rédigé un cours structuré et résumé des notions les plus importantes avec exemple pour des projet futur en assembleur.

La compréhension de la structure d'un exécutable ELF.

J'ai mis cela en tant que problème, car je n'arrivais pas vraiment à voir et comprendre comment un exécutable ELF était parsé et comment on pouvait l'utiliser à notre avantage pour réaliser le projet. Après m'être renseigné en profondeur en relisant le cours et en consultant le lien suivant : https://gist.github.com/x0nu11byt3/bcb35c3de461e5fb66173071a2379779, j'ai pu mieux comprendre cette structure. Comme pour l'assembleur, une fois que j'ai eu le déclic, tout s'est agencé logiquement. Cela m'a permis de réaliser le commit "Identifying PT_NOTE segment", qui représentait une grosse avancée pour moi. En effet, ce commit signifiait que mon apprentissage de l'assembleur commençait à porter ses fruits et que je pouvais enfin manipuler les segments PT_NOTE et les program headers, ce qui constitue une étape majeure pour le projet. Par ailleurs, tout comme pour l'assembleur, j'ai également rédigé des notes importantes sur ce que je devais utiliser et comment manipuler ces éléments pour mener à bien le projet.

Écriture du shell code

Un problème simple que j'aurais pu éviter, mais sur lequel j'ai quand même passé beaucoup de temps, était l'écriture du payload dans l'exécutable ELF. Je ne sais pas pourquoi, mais je pensais qu'il fallait écrire du code assembleur directement dans le fichier. À ce stade, j'avais réussi à changer le segment PT_NOTE en segment PT_LOAD. J'avais également modifié les champs nécessaires comme p_filesz, p_memsz et e_entry, et ajouté le payload à la fin du fichier. Je pense qu'à ce moment-là, je devais être fatigué d'avoir programmé uniquement en assembleur pendant plusieurs jours d'affilée. Bloqué à ce stade, je me suis dit qu'à l'école, je demanderais comment se passaient les projets des autres. En discutant avec mes camarades, ils m'ont fait remarquer mon erreur donc il fallait écrire le code machine au lieu

de code assembleur. Grâce à eux, j'ai pu reprendre ma progression après avoir pris une pause de quelques jours.

Phases de debugging

Les phases de debugging représentaient un problème que je redoutais le plus, surtout au début, car je ne comprenais pas grand chose à l'assembleur. Une fois l'apprentissage des bases terminé, j'étais plus confiant, d'autant plus qu'au départ, le projet ne contenait pas beaucoup de lignes de code, ce qui le rendait relativement simple à debugguer. Cependant, au fur et à mesure que je dépassais les centaines de lignes de code, je redoutais de plus en plus les erreurs, surtout que je n'utilisais pas les outils nécessaires. J'essayais de résoudre les problèmes en ajoutant des affichages dans la console comme "PT_NOTE segment FOUND", etc. Cela m'a valu de tomber dans une impasse pendant des jours, car l'un des messages d'affichage modifiait un des registres utilisés lors de la recherche du PT_NOTE.

Pour le debugging, j'ai principalement utilisé au début le syscall debugging, puis je l'ai abandonné par la suite. J'ai également utilisé la commande readelf -h afin d'inspecter l'exécutable cible pour identifier e_entry, vérifier si le segment PT_NOTE avait bien été modifié en PT_LOAD, et inspecter les champs p_filesz, p_memsz et p_flags. Enfin, ma révélation dans ce projet a été l'outil gdb avec le plug-in pwndbg. Cet outil m'a permis de parcourir le code ligne par ligne et de vérifier les valeurs des registres. C'est d'ailleurs grâce à gdb que j'ai pu identifier mon problème lié aux messages d'affichage qui modifiaient la valeur de certains registres. Sans cet outil, je serais probablement encore en train d'essayer de debugger ce problème.

Poursuivre l'exécution de l'exécutable d'origine

Enfin il y a la dernière étape que je n'ai pas réussi. J'ai essayé de le faire avec la fonction write_patched_jmp. Cette dernière va calculer l'offset relative vers le point d'entrée initial du fichier à l'aide de la formule suivate : newEntryPoint = originalEntryPoint - (phdr.vaddr + 5) - virus_size. +5 equivaut à la taille de l'instruction jmp. Ensuite la fonction va écrire l'instruction jmp vers l'adresse originale de e_entry. Mais pour certaines raisons, cela ne veut pas fonctionner et apres avoir quitter le shell rien ne se passe.

3. Conclusion

Ce projet m'a permis de mieux comprendre le format ELF, qui bien que complexe, reste très structuré et puissant pour la manipulation de fichiers exécutables. J'ai également appris à programmer en assembleur, ce qui m'a demandé une grande rigueur, notamment dans la gestion des registres et des syscalls. Les phases de debugging ont été particulièrement formatrices grâce à des outils comme gdb, pwndbg et readelf, qui m'ont permis d'inspecter et de comprendre en profondeur le comportement de mon code. Ce projet m'a apporté des connaissances techniques solides et m'a appris à persévérer face aux difficultés, tout en développant une méthodologie de travail claire et structurée.