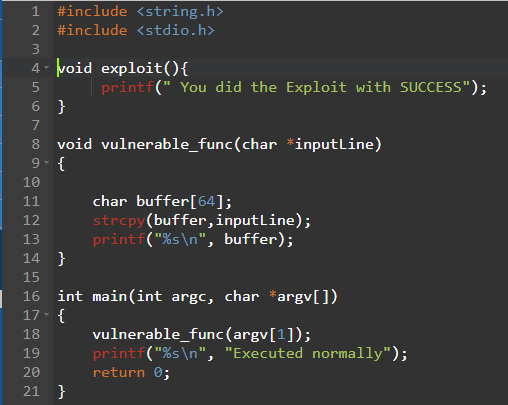
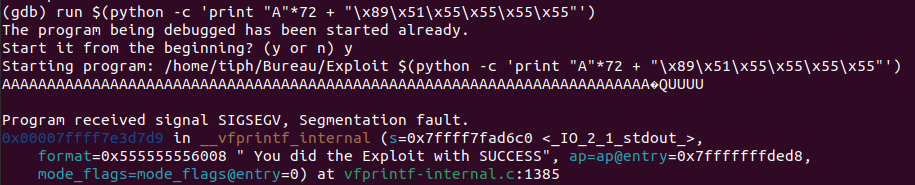
Quand les gens me demandent ce que je fais, bien souvent je leurs dit que je travaille en Cybersécurité plutôt que de dire que je travaille en Informatique, ce qui aurait été beaucoup plus simple pour eux à comprendre. Mais cela me fait tellement plaisir de voir leurs fascination pour ce domaine qui est perçu comme « Wahouuuu, elle doit savoir pirater le Facebook ». Je sais, j’exagère… Mais la réalité en est tout autre quand je travaille. J’ai la chance de toucher un peu à tout et je me dis que c’est essentielle d’élargir ses compétences pour comprendre ce qui nous entoure. Alors j’ai commencé ma carrière dans la Gouvernance. J’étais confronté à un dilemne qui je peux dire était casi quotidien. Mon rôle principale était de vérifier la conformité des projets en regard de la PSSI, alors qu’en face de moi j’avais des interlocuteurs non chalant qui ne voyait que par la validation de leurs projet. Ce que je comprends totalement.. Mais eux ne comprenaient pas que c’était pour le bien de la sécurité de l’entreprise. Aujourd’hui, j’intègre un centre de service dans lequel nous sommes plusieurs auditeurs en charge d’effectuer une revue de code par l’intermédiaire de l’outil CheckMarx. En parallèle, j’effectue des tests d’intrusions, ce qui me permet d’avoir plus de recul sur l’importance de la sécurisation du code à travers la Security By Design. J’ai audité plus de 4000 vulnérabilités dans notamment des programmes en C++/JAVA. Et je dois dire qu’en fonction de l’entreprise où j’audit, je peux faire face à des développeurs qui ont beaucoup plus conscience de l’enjeux qu’il y a derrière par la sécurisation du code et font donc confiance et sont à l’écoute de nous, auditeurs. A contrario, il arrive également d’avoir des interlocuteurs qui sont peu investis et impliqués par ce côté-là du travail, et il faut alors se justifier sans cesse. En même temps, c’est l’enjeu de ce travail aussi. Mais je dois dire que ce côté-là m’enthousiasme peu.

La majeure partie des vulnérabilités que j’ai pu voir sont liées à la confiance que l’on accorde beaucoup trop facilement aux inputs/variables que l’on inclut par la suite dans notre code. Elles peuvent entrainer de nombreuses techniques d’attaques qui peuvent être exploiter pour s’introduire dans le serveur et même voir entrainer une intrusion dans le SI. Je me suis dit que cela serait beaucoup plus simple de comprendre l’enjeux. Pour démontrer mes propos, j’ai décidé de créer un petit programme en C++ pour exploiter une vulnérabilité de type BufferOverflow.

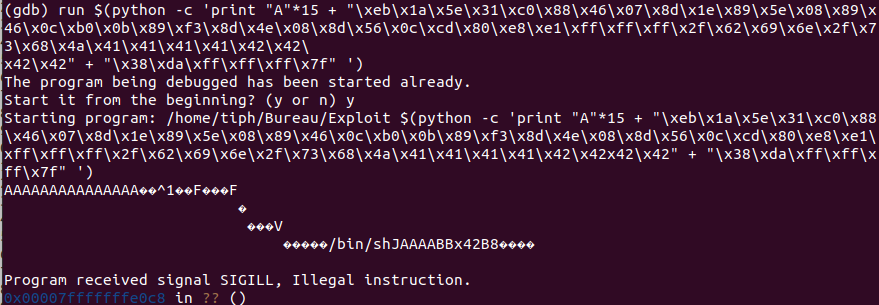
Dans le code ci-contre, je n’effectue aucune vérification sur l’argument argv[1]. Ce qui me laisse la possibilité d’injecter n’importe quelle commande. J’utilise ensuite cette argument comme variable dans la fonction vulnerable\_func que je prends soin de copier dans un buffer de taille 64 à l’aide de la fonction strcpy. Volontairement, j’utilise cette fonction car elle ne permet pas d’évaluer la taille de la variable qui est copier dans buffer. Vous l’aurez compris c’est grâce à cette vulnérabilité que je pourrai générer un dépassement de tampon. Un dépassement de mémoire tampon est provoqué par certaines conditions dans lesquelles un programme en cours d’exécution écrit des données en dehors de la mémoire tampon. En injectant du code (shell) et en redirigeant le flux d'exécution d'un programme en cours d'exécution vers ce code, un attaquant est capable d'exécuter ce code. C'est ce qu'on appelle l'exécution de code arbitraire. Avec l'exécution de code arbitraire, un attaquant peut obtenir le contrôle (à distance) d'une cible spécifique, élever des privilèges ou provoquer un déni de service sur la cible.

Pour exploiter cette vulnérabilité, l’idée est de dépasser la taille alloué pour « char buffer[64] ». Je répète donc 72 fois le caractère A et je concatène à ce pattern l’adresse en hexadecimal de la function Exploit() afin de forcer la pile à l’exécuter. Comme on peut le voir en dessous le programme a fait appel à la fonction Exploit et à afficher « You did the Exploit with SUCCESS ».



Peu importe si vous ne comprenez pas forcément pourquoi j’ai réussi à executer une function que initialement n’était pas sencer être exécuter. L’essentiel est que vous comprenez qu’en tant qu’attaquant, si vous me laissez cette vulnérabilité, je saurai l’exploiter et exécuter tout ce que je souhaite ! Je vais même aller plus loin dans ma démonstration, je vais exécuter un shell afin de prendre la main sur le serveur.

Bien entendue, ce type de cas est light/simple par rapport à ce qu’un attaquant peut lui-même engendrer à partir de ces vulnérabilités. J’ai poussé mon scénario un peu plus loin pour vous le démontrer. L’exploitation de ce bufferoverflow me permet d’injecter un certain nombre de pattern pour remplir la pile et forcer la pile (stack) à executer ce que je souhaite. Ici, mon shellcode qui n’est en autre que la traduction d’une simple commande « execve(‘/bin/sh’) » me permettant d’obtenir un bash par l’intermédiaire d’un input.



Les conséquences sont telles que l’attaquant a libre accès sur le serveur et peut continuer le déroulement de son scénario d’intrusion allant même jusqu’à intrusion généraliser dans le SI sans que nous puissions être en mesure d’arrêter ses mouvements.

En tant qu’auditeur, la première chose que je regarde c’est la validation des entrées car elle m’a permis durant plusieurs tests d’intrusions d’obtenir des informations sur le systèmes que je ciblais. Pour ce type de cas, il s’agit principalement d’effectuer les vérifications ci-dessous :

# Notes

*Comme notamment la version du serveurs, l’exécution de commandes, la réalisation de différentes techniques me permettant de m’introduire dans le système. Mes premiers reflexes lors d’audit, c’est d’effectuer une phase de reconnaissance, ce qui me permet d’identifier les technologies utilisées. Après avoir identifié ces technos, je vais identifier si ces technologies sont obsolètes et par conséquence comportent des vulnérabilités. Ces vulnérabilités bien entendue peuvent provenir du code qui a permis la réalisation de celles-ci, entrainant par exemple des exploits que je testerai bien entendue.*

*Imaginons que votre code s’appuie sur des technologies obsolètes ou bien des fonctions qui sont précaires comme par exemples strcpy.*

A buffer overflow is caused by certain conditions where a running program is writing data outside the memory buffer. By injecting (shell)code and redirecting the execution flow of a running program to that code, an attacker is able to execute that code. This is called arbitrary code execution. With arbitrary code execution an attacker is able to gain (remote) control of a specific target, elevate privileges or cause a denial of service on the target. Buffer overflows can be proactively prevented and mitigated with several techniques. Programmers should write secure code and test it for buffer overflows. When a buffer overflow is not prevented from happening it can still be mitigated with reactive methods like protecting memory from being written to.