Projet de programmation système : BotNet

Xavier Gillard, Olivier Choquet, José Vander Meulen, Anthony Legrand

Introduction

Dans ce projet, les étudiants intégreront les connaissances acquises en termes de programmation système au travers du développement d'un botnet assez simple. Un botnet est un réseau d'ordinateurs infectés par des programmes malveillants (botnets ou bots) qui permettent à un attaquant (ou un "contrôleur" du botnet) de prendre le contrôle à distance de ces ordinateurs et de les utiliser pour effectuer des activités malveillantes. Les botnets peuvent être utilisés pour diverses activités malveillantes, telles que le spamming, le vol d'informations personnelles, les attaques par déni de service distribuées (DDoS), les attaques par force brute, les attaques de phishing et autres activités criminelles. Les botnets peuvent être très vastes, avec des milliers d'ordinateurs infectés à travers le monde, et leur utilisation est souvent illégale. Les botnets sont un véritable fléau pour la sécurité informatique et leur détection et leur neutralisation sont des enjeux majeurs pour les professionnels de la sécurité informatique.

L'objectif de ce projet n'est en aucun cas d'inciter les étudiants à programmer de tels botnets pour les utiliser à des fins malveillantes. Ceux-ci s'engageront d'ailleurs à ne pas utiliser leurs nouvelles compétences pour exercer des activités qui sortent du cadre prévu par la loi. Ils profiteront néanmoins des connaissances acquises pour vérifier la vulnérabilité (code et configuration) des programmes dont ils auront la charge.

Dans ce projet, les étudiants seront chargés de concevoir et de programmer deux parties d'un Botnet, à savoir le côté client (aka le "contrôleur") et le côté serveur ("zombie") ainsi qu'un logiciel permettant d'automatiser la validation du botnet. A cet effet, ils programmeront **trois exécutables : `controller`, `zombie` et `labo`**. Le rôle du controller étant de se connecter à distance à un ou plusieurs zombies afin d'y ouvrir une session **bash** sans avoir à s'authentifier.

1. Le programme zombie

Avant toute chose, bien comprendre qu'ici le mot *zombie* désigne un programme déposé sur la machine "victime" qui permettra de contrôler celle-ci. Cela n'a aucun lien avec la notion de processus zombie (process defunct) vue au cours !

Le programme zombie est le point d'entrée qui permet au contrôleur d'exécuter du code sur la machine qu'il a ciblée. Il s'agit d'un serveur qui implémente le paradigme *BLAB* (**B**ind - **L**isten - **A**ccept - **B**egin) pour écouter sur un port TCP. Pour chaque connexion entrante, l'outil va créer un nouveau processus, lequel bénéficiera des droits de l'utilisateur qui a lancé le zombie. Par exemple, si c'est root qui a lancé le zombie, les nouveaux processus doivent avoir les mêmes permissions que root.

Si le programme zombie est lancé sans argument il choisira un port au hasard parmi 10 ports définis dans un fichier header. Le programme zombie peut également être lancé avec un port donné en argument.

1.1. Les sous-processus

Le sous-processus lancera un shell bash. Afin de cacher un peu plus à d'éventuels administrateurs systèmes que le shell bash qui a été lancé est potentiellement dangereux, le nom du shell bash sera "programme_inoffensif" — ce qui est évidemment bien moins suspect. Ce shell bash a pour but d'être utilisé par le contrôleur ce qui veut dire l'entrée, la sortie standard, et l'erreur standard de ce shell doivent être connectés au programme controller.

2. Le programme controller

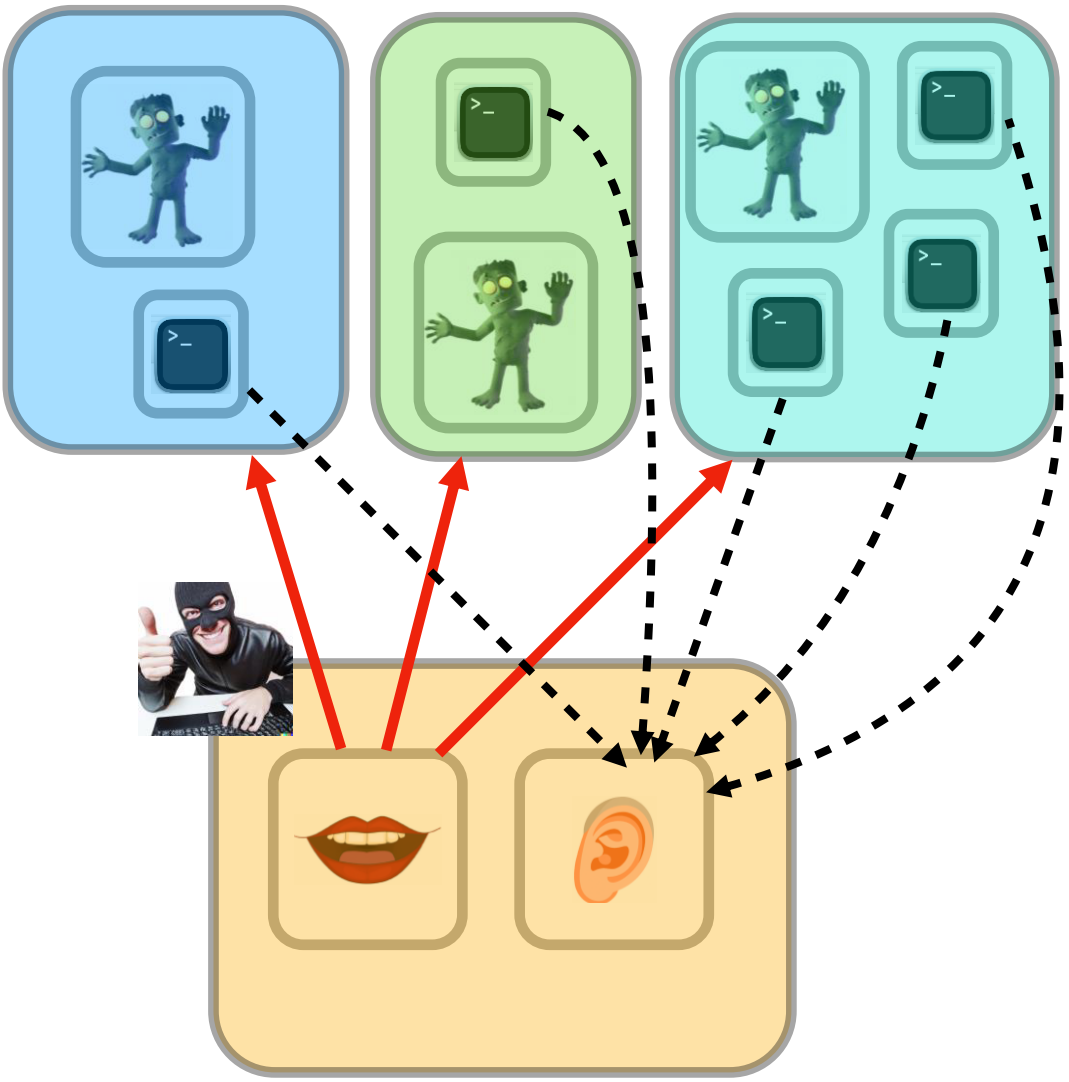
Le programme controller se comporte comme un **client** TCP qui se connecte en parallèle à plusieurs zombies dont les noms ou adresses ip sont passés comme arguments de la ligne de commande. Notez que puisqu'il est possible que les zombies écoutent sur des ports différents (aléatoires), votre implémentation de controller devra tester les différents ports possibles.

Après s'être connecté aux zombies, il sera possible via le contrôleur de transmettre à chaque zombie des commandes lues sur l'entrée standard afin de permettre à l'attaquant d'exécuter à distance le code voulu.

De la même façon, le contrôleur répétera sur sa sortie standard, tout le contenu qu'il a pu lire sur les connexions qu'il a ouvertes. De cette amnière le résultat des commandes lancées sur les zombies sera visible chez le controller. Vous devez éviter les attentes bloquantes dans le programme controller. En particulier, la lecture des réponses des zombies se fera de manière asynchrone.

Afin que l'attaquant puisse lire les réponses de ses victimes ET leur envoyer de nouveaux ordres en parallèle, l'écoute et la répétition des réponses des zombies se fera depuis un processus dédié.

Interaction entre les processus du BotNet



Sur ce schéma nous voyons 3 programmes zombies lançant respectivement un, deux et trois terminaux bash. Le controller se compose de deux processus, un permettant d'envoyer des commandes(parler) aux zombies, l'autre écoutant les réponses.

3. Labo

Le programme labo lancera deux zombies sur la même machine afin d'installer un petit laboratoire de test qui puisse servir à valider les implémentations du controller et du zombie. Chacun des zombies sera lancé sur un port choisi au hasard parmi une liste de 10 ports hardcodés dans un header.

Lorsque le programme labo se termine (avec un 'Ctrl+D'), il doit envoyer un signal à tous les zombies qui ont été lancés afin de s'assurer qu'eux aussi se terminent.

4. Gestion des arrêts et des situations exceptionnelles

De manière générale, ni les zombies, ni le controller ne doivent gérer les arrêts brutaux. En particulier, on fait l'hypothèse que personne n'effectuera jamais un `kill -9` pour tuer un de vos

programmes. Vous ne devez donc pas gérer ces situations d'arrêts brutaux. Par contre, il est prévu que les zombies se terminent lorsque quelqu'un appuie sur les touches "Ctrl+C". De la même façon, les sous processus du zombie doivent se terminer lorsque le contrôleur ferme sa connexion (fin de fichier si l'entrée standard est redirigée ou "Ctrl+D" au clavier).

Il n'est pas non plus attendu que les programmes aient un traitement avancé des situations exceptionnelles (par ex. impossibilité d'ouvrir un socket, erreur d'écriture, etc..). Dans de tels cas, le programme concerné pourra s'arrêter en affichant un message d'erreur avec la fonction `perror`.

5. Indications utiles

Commencez par développer le programme zombie et controller en ayant uniquement un seul zombie (un seul port).

Ensuite généraliser pour avoir plusieurs zombies. Un port sur une machine ne peut être attribué ("bindé") qu'à un seul processus d'où l'existence du programme labo pour tester votre solution sur une même machine (la victime et l'attaquant étant confondu).

6. Délivrables

6.1. L'analyse

Nous vous demandons de réaliser l'analyse de l'application. Il s'agira de réfléchir à la découpe de votre application. Nous vous demandons au terme de votre analyse le Makefile ainsi qu'un document indiquant la découpe en fonctions de votre solution. Voir fichier canevas analyse_application.docx. L'analyse sera à remettre sur MooVin au terme de la première semaine pour validation (Voir Planning ci-dessous). Une validation du Makefile et de la découpe pourra également être faite en séance avant la remise pour les groupes le souhaitant.

6.2. Le code final

Nous vous demandons de remettre sur MooVin l'ensemble de votre code source (documenté !), ainsi qu'un Makefile permettant la compilation des différents programmes exécutables au terme des 2 semaines du projet (voir Planning ci-dessous). Nous vous rappelons que la politique de la Haute Ecole en matière de plagiat est très stricte. Nous vous signalons qu'un système automatique de détection de plagiat sera appliqué à votre code.

6.3. Défense orale

Vos projets seront testés en votre présence le lundi 15 mai. Ces tests seront effectués sur les machines de l'école **OU** vos machines au choix. Nous vous conseillons cependant plutôt l'emploi de vos machines pour éviter des soucis de comptabilité. Le code qui sera exécuté sera celui soumis sur MooVin et pas un autre ! Nous vous le (re)fournirons. Le scénario de test vous sera communiqué durant le projet.

7. Planning

Date	Action
20-28 avril 2023	Choix des groupes
28 avril 2023 à 16h	Présentation du projet
05 mai 2023 à 20h	Remise de la découpe en modules
14 mai 2023 à 20h	Remise du code source de vos programmes
15 mai 2023	Défense orale du projet (un horaire précis + scénario de tests vous sera communiqué en temps voulu)

8. Evaluation

La note du projet est répartie de la sorte :

- 10% : validation de l'analyse
- 90% : code et défense du projet (qualité du code + exécutions des tests lors de la défense)

9. BONUS

Connaissez-vous les uid ainsi que l'appel système setreuid ?
Voici une petite vidéo explicative :
https://www.youtube.com/watch?v=X-hnHmr8WJs

En quoi ceci pourrait être utile dans notre projet ?

Le controller peut envoyer un user id aux zombies. Le programme shell lancé par le zombie peut alors se cacher en se faisant passer pour un utilisateur légitime du système tout en gardant ses vraies permissions. En fait, je dis que je suis l'utilisateur olivier (beaucoup moins suspect) mais en fait je suis root !