Dans un sous-repertoire nommé **projet** à la racine du dépôt du projet.

Notions évaluées	Xp à gagner
Compiler un kernel linux et customisation d'un LFS	20
Détournement d’appel système/devtmpfs/autre dans le kernel linux	20
Création d'un module kernel de type 'rootkit'	20

Fichier à rendre:

```
                  .

0 directories, 0 files
```

Sujet:

Système d’exploitation et programmation C - Projet

Rootkit

Vous devez réaliser un rootkit linux par groupe de 4. La composition aléatoire des groupes sera envoyé sur la classroom. La date de rendu est présente dans votre planning et sera programmé sur la classroom.

Le contexte d’exécution du rootkit:

- Accès initiale sur la machine.
- Le Kernel Linux accepte le chargement de LKM non signé.

1 - Rendu

- Vous aurez à scripter la fabrication d’un système Linux de Test pour votre démonstration.
- Vous rendrez une image QCOW2 de votre disque contenant votre LFS (Linux From scratch). Le tp01 vous conduit à la création d’une image de type “raw”, utilisez “qemu-img convert -c …” pour obtenir le fichier en QCow2 compressé. La taille du fichier sera grandement réduite (dizaines de méga). Vous serez pénalisé si le test de type de votre rendu n’est pas le bon.
- Fichiers supplémentaires à rendre:
    - PDF de votre présentation lors de la soutenance
    - Documentation utilisateur
    - Rapport de conception de vos programmes. Votre rapport contiendra la répartition des tâches et le rôle de chacun des membres du groupe. Vous

mettrez un tableau de ratio (en pourcentage) du travail fournis sur l'ensemble du projet. Ce tableau sera utilisé pour la notation. Un travail normal d'un étudiant normal aura un ratio normal (25%). Les fantômes à 0% auront 0 sur le projet.

    - En bonus, votre rapport de conception peut prévoir et présenter un/des scénario(s) crédible(s) d'accès initiale sur un système linux sans modules signés. L'idée est d'identifier les domaines industriels dans lequel linux est utilisé et que pour certaines raisons (à préciser) le kernel n'est pas signé et/ou les kernels modules sont chargeables sans être signés.

    - Sources codes:

        - source du rootkit et "build system" (Makefiles, autres…)
        - source du/des programme(s) compagnon(s) et "build system"
        - source des scripts de fabrication de la LFS

- Votre LFS contiendra 2 comptes:

    - un compte root permettant de charger votre rootkit sous forme de fichier *.ko*
    - un compte user permettant de démontrer l'usage de votre rootkit via un/des programme(s) compagnon(s)

- Les login/mdp des différents comptes seront donnés dans la documentation.
- Le code de votre rootkit devra respecter au maximum les consignes de l'ANSSI concernant le développement sécurisé C Coding Standards concernant l'écriture du code C en terme de coding style (ou équivalent si votre rootkit n'est pas en C). De plus, l'usage d'une langue différente de l'anglais pour les identifiants est proscrite.
- Vous êtes libres sur le langage de programmation utilisé pour votre/vos programme(s) compagnon(s), toutefois le respect de la coding style usuelle pour ce langage devra être respecté.
- Vous rendrez un rapport sur la conception et la mise en oeuvre de votre rootkit, détaillant vos choix et votre mise en oeuvre.
- Il est *INTERDIT* de détourner l'appel système "kill", soyez plus créatif.
- Il est *INTERDIT* de rendre un rootkit dont vous ne serez pas les auteurs.

2 - Objectifs

L'objectif de ce projet est de parfaire votre compréhension des domaines suivants:

- Système d'exploitation Linux et Kernel Linux,
- Mécanisme d'appel système,

- Pilote de périphérique,
- Rootkit.

3 - Fonctionnalités de votre projet

L'objectif principal d'un rootkit est de fournir une porte d'accès aux programmes malveillants compagnons afin d'obtenir des privilèges élevés, tout en se protégeant contre les éléments de défense du système.

Ainsi, votre rootkit devra :

- Permettre à un programme malveillant compagnon de lui envoyer des requêtes par différents moyens : appels système, pseudo-système de fichiers "dev", réseaux, ou d'autres méthodes.
- Si le rootkit peut être contacté par des appels système, il le fera en remplaçant ces appels existants plutôt qu'en en ajoutant de nouveaux (hooking des appels système).
- Camoufler au maximum sa présence pour tous les programmes de l'espace utilisateur qui n'ont pas connaissance des méthodes nécessaires pour accéder aux fonctionnalités du rootkit.
- Fournir un ensemble de fonctionnalités malveillantes pour le programme compagnon.
- Être discret en ne contenant aucun bogue susceptible de provoquer un plantage du système.
- Être persistant et donc se charger automatiquement à chaque redémarrage de la machine.

N'oubliez pas que votre rootkit s'exécute dans l'espace noyau (kernel) et peut vous offrir une certaine marge de manœuvre pour atteindre vos objectifs.

Par exemple, si vous modifiez un script exécuté au chargement du système pour forcer le chargement de votre module *.ko* et ainsi être persistant, n'oubliez pas qu'un utilisateur éditant ce fichier pourra voir cette modification. Dans cet exemple, la surcharge des appels système *read* et *write* devrait être capable de détecter l'ouverture spécifique de ce fichier et de camoufler la ligne incriminée. De même, la surcharge de l'appel système *getdents* devrait vous permettre de dissimuler vos fichiers pour n'importe quel programme utilisateur.

4 - Soutenance

- Le projet sera évalué lors d'une soutenance d'une durée de 20 minutes.
- Vous aurez préalablement rendu votre projet ainsi que votre rapport sur la plateforme de la classe.
- Lors de la soutenance, vous réaliserez une présentation et une démonstration de votre rootkit.

- Une revue de code sera effectuée, et vous serez évalué sur la qualité de votre code, tant du point de vue esthétique que fonctionnel.
- La qualité de vos livrables (code et rapport), de votre présentation, ainsi que la richesse des fonctionnalités proposées (malveillantes et de discrétion) seront les principaux critères de votre évaluation.