Projet informatique CSC 3502

Réalisation d'un *honeypot* SSH

Anthony Vérez
Guilhem Tiennot
Thibault Ingargiola
François Monniot

Version 0.3
Dernière révision : 23 mai 2012

Table des matières

Chapitre 1

Introduction

Le but de ce projet informatique est de réaliser un *honeypot* (pot de miel) SSH en C. Il s'agit d'un service de sécurité informatique volontairement vulnérable qui sert à piéger un attaquant. Dans le cas présent, ce service consiste en un serveur SSH auquel il est possible de se connecter avec des identifiants triviaux (`root/root` par exemple). On désire une fois l'attaquant connecté enregistrer un maximum d'informations sur lui et son interaction avec le serveur. Il est également souhaitable, pour des raisons de sécurité, de pouvoir paramétrer finement le résultat de commandes particulières.
Les intérêts sont multiples :

- Pouvoir savoir si un réseau est souvent attaqué.
- Essayer de comprendre comment un attaquant s'y prend pour compromettre un serveur.
- Récupérer les scripts et logiciels potentiels que l'attaquant aurait chargés pour l'aider à compromettre le serveur, dans le but de les examiner et les comprendre. Cela peut conduire à la découverte de nouvelles failles de sécurité.
- Observer les réactions de l'attaquant face à des comportements volontairement anormaux du serveur SSH.

Au cours de ces derniers mois, nous avons donc du réaliser une architecture réseau complète, à même de répondre aux contraintes du problème, implémenter des commandes bash destinées à leurrer l'attanquant, et à faire en sorte qu'il ne puisse pas outrepasser notre sécurité, ainsi que créer un serveur de log pour récolter le plus d'informations possible sur l'attaquant.

Notre Honeypot est aujourd'hui fonctionnel, bien que légèrement en dessous de nos espérances premières. nous pouvons nous connecter à notre faux serveur SSH, entrer quelques commandes, obtenir des réponse à la fois pour l'attaquant, qui croit donc que le système lui répond directement, et sur le système de log, qui enregistre toutes les commande taper, ainsi que les réponse renvoyées.

Chapitre 2

Cahier des charges

2.1 Contraintes

- Le serveur SSH devra pouvoir tenir la charge contre des attaques par dictionnaire et par force brute.
- Toute exécution d'un programme chargé ou écrit par l'attaquant devra être controlé.
- Tous les moyens d'accès au réseau devront être contrôlés.
- Il faudra gérer l'utilisation des ressources (Bande passante, CPU, mémoire vive, disque dur, ...) par l'attaquant pour éviter qu'il sature la machine.
- Le système devra être très sécurisé au niveau de l'architecture ainsi que de l'implémentation de l'honeypot
- Nettoyer le système à chaque session pour éviter que l'honeypot finisse en plateforme d'échange de fichiers par exemple

2.2 Architecture

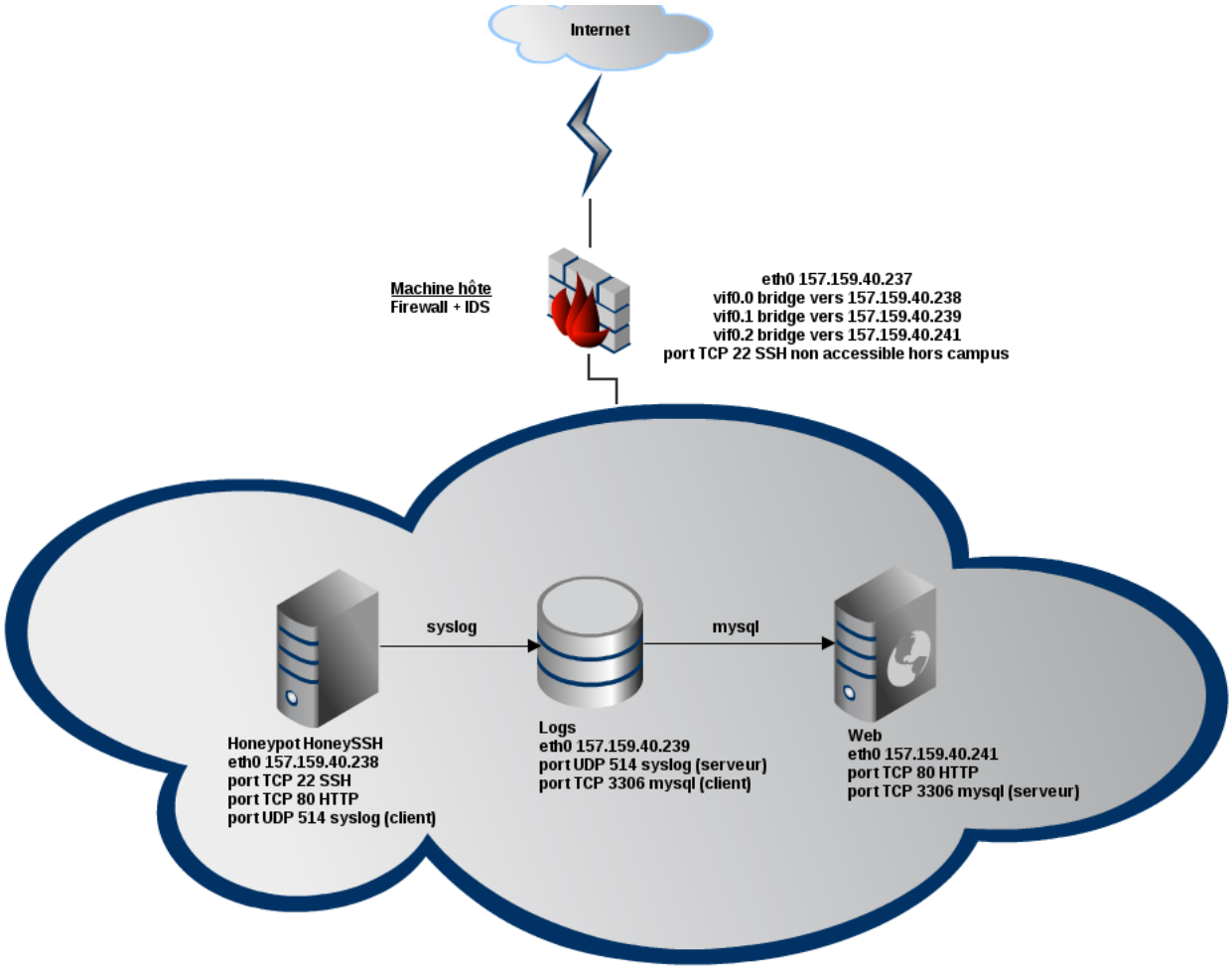
L'architecture réseau est indispensable car c'est un moyen de sécurisation du réseau qui contient notre honeypot. Par exemple, si on place l'honeypot dans un réseau d'entreprise et qu'il permet de passer outre un pare-feu ou un IDS, il peut compromettre l'ensemble du réseau de l'entreprise.

De plus, le logiciel honeypot a besoin de communiquer avec d'autres services disponibles sur le réseau. Chacun des systèmes représentés sera sa machine virtuelle (VM). Ceci permettra de simuler un vrai réseau tout en pouvant utiliser un unique ordinateur.

Pour déployer l'honeypot sur un réseau physique, on pourra utiliser les fichiers de configuration utilisés dans nos VM (sous système Debian Squeeze). Afin d'utiliser notre honeypot, nous aurons donc :

- 1 VM qui servira à sécuriser le réseau grâce à un pare-feu. Elle permettra de filtrer les connexions partants et arrivants à l'honeypot et empêcher l'attaquant de rebondir et d'attaquer d'autres serveurs.
- 1 VM sur laquelle se trouvera notre honeypot HoneySSH qui aura les fonctionnalités décrites plus haut.
- 1 VM qui stockera les logs. Ainsi même si l'honeypot est compromis, l'attaquant ne pourra pas modifier les logs.
- 1 VM qui servira de serveur web pour héberger l'interface web.

FIGURE 2.1 – Architecture du réseau



2.3 Fonctionnalités

2.3.1 Serveur SSH

Le serveur SSH doit pouvoir communiquer avec tout client SSH, plusieurs clients pourront être connectés en même temps.

On devra pouvoir changer les bannières envoyées au client afin qu'il se fasse passer pour un serveur SSH courant.

On pourra paramétrer des identifiants de connexion triviaux pour viser les attaques par dictionnaire et par force brute.

2.3.2 Logs

Un système de gestion des logs sera mis en place.

Il permettra d'enregistrer dans des fichiers les messages suivants avec les dates correspondantes :

- Identifiants de connexion essayés avec réponse du serveur
- Commandes entrées par l'attaquant
- Résultats des commandes
- Informations sur l'état du système hébergeant l'honeypot

– Éléments de debuggage

2.3.3 Identification de l'attaquant

Un système passif de collecte d'informations sera implémenté pour essayer d'identifier les sources des attaques (adresse IP, client utilisé, . . .).

2.3.4 Système de fichiers

Nous allons utiliser un chroot pour isoler le système de fichiers de notre honeypot.

Ce système pourra gérer les commandes de manipulations des fichiers et des dossiers.

On cherche à simuler un système Debian 6 Squeeze. Les fichiers présents par défaut dans cette distribution seront donc visibles par l'attaquant.

2.3.5 Interpréteur de commande

Afin de contrôler le fonctionnement des commandes, un interpréteur de commande sera réalisé. Il sera doté d'un historique.

Les commandes pourront être paramétrées de trois façons :

1. Redirection directe vers une console bash
2. Redirection filtrée vers une console bash
3. Implémentation personnalisée

On implémentera les commandes les plus utilisées selon les études existantes. Le nombre de commandes implémentées dépendra de l'avancée du projet.

Certaines commandes sont néanmoins indispensable :

– ls
– pwd
– ps
– cd
– who
– whoami
– id
– wget

D'autre sont moins critiques :

– echo
– rm
– adduser/useradd
– ...

2.3.6 Interface web

Une interface web sera utilisée pour afficher les données collectées par l'honeypot. Dans la mesure du possible, on adaptera une solution existante afin de réduire la charge de travail.

Les données seront stockées dans une base de données après un transfert comportant un traitement de sécurité des fichiers de log. L'interface web permettra de visualiser les différentes sessions. On pourra voir les différentes commandes entrées par l'attaquant et les résultats qu'il a obtenu. Les informations sur l'identification de l'attaquant seront aussi affichées, ainsi que des données stastistiques (mot de passe les plus essayés, nombre d'attaquant par semaines ...).

Chapitre 3

Dévelopement

3.1 Architecture

La partie architecture a été réalisée sur une lame serveur mise à disposition par l'association MiNET, et remise en marche par nos soins (il a fallut changer quelques pièces). La virtualisation a été faite sous Xen, qui permet une *Full Virtualisation*, on peut donc installer des machines virtuelles avec des noyaux différents de la machine physique, et même des autres machines virtuelles. Un pare-feu Netfilter a été installé sur la machine physique, et configuré avec iptables. La partie honeypot est installé seule sur une machine virtuelle, elle est donc isoler des autres machines. Les logs sont donc récupérés, sur une machine virtuelle différente, puis réinjectés dans la base de données mysql de l'application web, après filtrage de sécurité.

3.2 Serveur SSH

Le serveur SSH a été réalisé afin de fonctionner sur une machine sous Debian. Nous avons choisi d'utiliser la bibliothèque *libssh* pour implémenter notre serveur en C. Dans un premier temps nous avons du gérer la redirection vers un TTY des requêtes SSH, ce qui était mal documenté sur la bibliothèque, nous y avons donc contribuée. Ce serveur SSH utilise des threads pour permettre de gérer plusieurs utilisateurs connectés simultanément, implémentée grâce à la bibliothèque C libssh. Cependant ladite bibliothèque laisse visible sa bannière (dans les paquets réseau, visualisé par exemple avec l'outil Wireshark), peu crédible du point de vue de l'attaquant. Nous avons dû modifier une nouvelle fois la bibliothèque libssh pour modifier la bannière. Nous réussissons à modifier la bannière d'openssh à un caractère de retour chariot près (il s'agit vraissemblablement d'un bug dans libssh). Malgré nos efforts, nous nous sommes rendus compte que notre serveur SSH peut être utilisé avec le client openssh mais de nombreuses bibliothèques ne parviennent pas à s'y connecter.

3.3 Tests

Il a été choisi d'utiliser la bibliothèque TinyTest pour les tests unitaires. Les tests fonctionnels du pare-feu ont déjà été réalisés avec ScaPy, et passent avec succès.

Chapitre 4

Déploiement

4.1 Introduction

Cet article a pour but de vous expliquer comment mettre en place l'honeypot HoneySSH. Nous allons dans un premier temps visualiser l'infrastructure telle que nous l'avons envisagé lors du développement de l'honeypot. Vous pouvez bien sûr adapter cette infrastructure à vos propres besoins.

Dans ce guide, les nom écrits entre deux underscore (par exemple _exemple_) sont à remplacer. Les expressions entre guillements ‘ sont des commandes shell à exécuter.

N.B. : Pour mettre en place cette architecture réseau nous avons utiliser l'hyperviseur Xen pour lequel vous pouvez trouver un guide d'installation à cette adresse : http :/ /wiki.debian.org/Xen

4.2 Création des machines virtuelles

Configuration par défaut des VM se trouve dans le fichier ‘/etc/xen-tools/xen-tools.conf ‘

Pour créer une VM :

```
# xen-create-image -hostname _hostname_ -role udev -ip _ip_
```

Une fois créée, il faut démarrer la VM :

```
# xm create /etc/xen/_hostname_.cfg
```

4.2.1 Renommage des interfaces bridgées

Dans les fichiers `_/etc/xen/mavm.cfg_` : trouver la ligne ‘vif = [...]‘ et rajouter dans les crochets l'option ‘_vifname=vif0.0_‘ pour renommer par exemple l'interface bridgée en ‘_vif0.0_‘

Dans notre installation, nous avons paramétré :
- vif0.0 : honeyssh
- vif0.1 : logs
- vif0.2 : web

4.3 Configuration générale des machines

- Se connecter en root sur la machine physique et charger le contenu de conf/leffe du projet dans / de la machine
- Se connecter en root sur la VM honeypot et charger le contenu de conf/honeyssh-honeypot du projet dans / de la VM

– Se connecter en root sur la VM de logs et charger le contenu de conf/honeyssh-logs du projet dans / de la VM

4.4 Configuration de la machine physique

Ajouter dans /etc/modprobe.d/options :

```
_options xt_recent ip_pkt_list_tot=100_
```

Pour lancer le pare-feu :

```
# /etc/init.d/iptables.sh start
```

Pour l'arrêter :

```
# /etc/init.d/iptables.sh stop
```

4.5 Création du système de fichiers

Le système de fichiers de l'honeypot est une simple image d'une partition d'un système Debian 6 standard qui sera montée à chaque démarrage de la VM. Le processus Honeyssh pourra alors se chrooter dedans, et les modifications effectuées par l'attaquant restent cantonnées à ce système virtuel.

Pour créer une telle image, il suffit de lancer un média d'installation de Debian Squeeze dans une machine virtuelle sous KVM. Lorsque le processus d'installation arrive à l'étape du partitionnement, il faut bien prendre garde à demander d'installer la totalité du système sur une seule partition.

Une fois l'installation terminée, il faut extraire la partition dans un fichier en *.img. En effet, le fichier disque servant de machine virtuelle à KVM ne peut pas être utilisé directement, car non reconnu par le système comme une partition montable. Il faut donc se munir d'un liveCD de n'importe quelle distribution GNU/Linux, et lancer la commande suivante :

```
# dd if=/dev/sda1 of=~/honeyssh.img
```

On obtient alors un fichier /honeyssh.img contenant l'image de la partition.

4.6 Utilisation de l'image disque

L'image disque se monte à l'aide de la commande suivante :

```
# mount -o loop ~/honeyssh /mnt
```

Pour se simplifier la vie, nous avons choisi de monter la partition automatiquement au démarrage de la VM à l'aide du fichier /etc/fstab. L'image se trouve dans le fichier /home/honeyssh.img, et se monte en /media/honeyssh. Voici la ligne correspondante dans le fstab :

```
/home/honeyssh.img /media/honeyssh ext3
rw,suid,nodev,exec,auto,nouser,async,loop 0 0
```

4.7 Mise en place de l'interface web

Sur la VM de serveur web, nous avons installé un serveur LAMP (Apache + Mysql + PHP).
Pour installer les logiciels nécessaires : `# apt-get install apache2 mysql-server php5 phpmyadmin`
Dans l'interface phpmyadmin accessible à http ://157.159.40.241/phpmyadmin/, créez un utilisateur honeyssh avec sa base de données. Se connecter en root sur la VM de serveur web, `# rm /var/www/index.ht`
Copier le contenu de www/ du projet dans /var/www de la VM.
Nous devons maintenant modifier le mot de passe mysql dans le site web. Pour ce faire éditer le

fichier /var/www/application/config/database.php, à la ligne 53, mettez le mot de passe de l'utilisateur mysql honeyssh entre les guillemets.
Modifiez le mot de passe contenu dans /var/www/.htpasswd.

4.8 Configuration du serveur syslog

Sur la VM de logs, l'installation de syslog-ng se fait par :

```
# apt-get install syslog-ng
```

Éditez le fichier etc/syslog-ng/scripts/web.py, ligne 5, entre les guillements doubles mettez le mot de passe de l'utilisateur mysql honeyssh.

4.9 Installation de l'honeypot

4.9.1 Dépendances

Sur la VM honeypot, exécutez :

```
# apt-get install libssh-dev openssl libpcap-dev cmake
```

Le paquet libssh servira à créer les serveur ssh, le paquet libpcap-dev servira à l'utilisation de p0f (voir 4.9.3), et cmake servira à compiler la libssh que l'on a modifié.

4.9.2 Compilation de libssh

La compilation de notre version de libssh est obligatoire car nous avons dû apporter des modifications à libssh.

Avant, bien penser à désintaller libssh si elle a été installée via le gestionnaire de paquets.
Se placer dans le dossier 3rdparty/libssh-0.5.2/build. Exécuter les commandes suivantes :

```
$ cmake -DCMAKE_INSTALL_PREFIX=/usr/local -DCMAKE_BUILD_TYPE=Debug ..
$ make
```

Puis :

```
# make install
```

4.9.3 Compilation et Utilisation de p0f

Introduction

Le logiciel p0f est un outil qui analyse, de manière passive (c'est-à-dire sans envoie de paquet), la trame des paquets TCP.

Pour plus d'information vous pouvez consulter la page http ://lcamtuf.coredump.cx/p0f3/

Nous réalisons l'installation de p0f sur la machine physique pour des raisons de sécurité et étant donné que les interfaces sont bridgées, tout le trafic passe par cette machine physique ce qui permet un plus grand rayon d'action à p0f.

Nota : si votre système dispose de p0f version 3 dans ses paquets vous pouvez l'installer depuis les paquets, aucune modification n'a été apporté au logiciel. Pour ceux qui, comme nous, utilise la version stable de Debian (à la date de rédaction de ce guide) vous disposez dans vos paquets de la version 2. Il vous faut donc compiler puis installer p0f manuellement.

Compilation

Avant de compiler, il vous faut installer les dépendances (si ce n'est pas déjà fait) :

```
# apt-get install libpcap-dev
```

Exécuter ensuite le script 'build.sh'

Utilisation

Nous allons dans un premier temps déplacer le logiciel ainsi que la base de données qui l'accompagne dans un autre répertoire :

```
$ cp _path/to/p0f_ _/new/path/to/p0f_
$ cp _path/to/p0f.fp_ _/new/path/to/p0f.fp_
```

On va ensuite créer un utilisateur sans aucun droit afin de sécuriser l'exécution de p0f :

```
# mkdir -p -m 755 /var/empty/p0f
# useradd -d /var/empty/p0f -M -r -s /bin/nologin p0f-user
```

On peut ensuite lancer le daemon avec la commande (vous devez ici remplacer le eth0 par le nom de votre interface) :

```
# ./p0f -ivif0.0 -s/var/run/p0f-sock -up0f-user -o /var/log/p0f &
```

Les logs sont enregistrés dans /var/log/p0f

Nota : Toutes les commandes sont recensées dans le fichier README ('3rdparty/p0f-3.03b/README')

4.9.4 Installation d'HoneySSH

Dans le dossier src, exécuter

```
$ make
```

Puis

`# make install` ce qui créera aussi les clés SSH.

Chapitre 5

Profils d'utilisation et tests

5.1 Fonctionnalités

Les tests de fonctionnalités devront se faire au niveau de la couche SSH : on envoit une commande et on regarde si la réponse formulée correspondant aux attentes.

5.2 Point de vue de l'attaquant

Un attaquant utilisant les outils populaires d'audit et d'exploitation de sécurité informatique devra pouvoir se connecter facilement à l'honeypot.

À cet effet, l'utilisation de la distribution backtrack devra suffir à détecter le serveur SSH, trouver des identifiants permettant de s'y connecter après une attaque par dictionnaire, s'y connecter et utiliser les commandes basiques pour identifier le système et exécuter des logiciels malveillants.

Chapitre 6

Interprétation des résultats

6.1 Analyse du problème

Le but de l'honeypot est de récupérer des logs enregistrés lors de l'attaque d'un éventuel attaquant. Les logs devront donc être stockés sur une machine virtuelle différente de celle du serveur, pour ne pas être affectés par l'attaquant. Les programmes de l'attaquant ne seront pas exécutés.

6.2 Conception du problème

Le problème a été traité en couches successives, chacune répondant à une partie du problème, et chacune étant encapsulée dans la suivante.

La première étape a été de penser une architecture réseau. En effet, avant toute chose, il fallait être sûr que l'attaquant serait bloqué, et qu'il ne pourrait donc pas déborder sur la partie du réseau que l'on veut garder saine. L'attaquant sera donc chrooté sur une machine virtuelle, qui servira d'interface avec le serveur de log, en passant à travers un pare-feu.

Ceci fait, il a fallu penser à un moyen de leurrer l'attaquant, lui faire croire qu'il est sur une vraie machine. Il a donc été choisi de créer une image à partir d'une vraie machine, que l'on monte à l'ouverture du serveur. L'attaquant est chrooté à la racine de cette image. Pour plus de réalisme, en guise d'appât, on a simulé un serveur apache. Toujours dans la même optique de leurrer l'attaquant, il a fallu rendre son action sur l'honeypot vraisemblable. Il a donc été choisi de rediriger une partie des commandes vers un vrai bash et de renvoyer les résultats. Cependant, par des soucis de sécurité, toutes les commandes ne pouvaient être implémentées de cette manière, certaine ont été filtrées puis redirigées, d'autres ont tout simplement été réécrites (simulant parfois des erreurs).

Enfin, le but premier de l'honeypot étant de récupérer des informations sur l'attaquant, il a fallu installer un serveur de log pour intercepter toutes les actions faites par l'attaquant. Elles sont ensuite renvoyées au serveur web.

Chapitre 7

Conclusion

Le problème de l'honeypot n'est pas des plus simples, il a fallu se soucier à la fois de l'architecture, pour avoir un interfaçage plus stable entre les différentes couches, de la sécurité, qui a rendu l'implémentation de certaines commandes compliquée (par exemple le chroot, a rendu certaines commandes inexploitables), et de la vraisemblance, il faut que l'attaquant puisse y croire (on retiendra ici le problème de la bannière du serveur ssh, qui n'est toujours pas conforme à nos exigeances). Notre honeypot est fonctionne pour un client openssh, les commandes les plus courantes ont été implementées. Avec davantage de temps, l'idéal aurait été de tester davantage notre honeypot avec des bibliothèques clients et d'implémenter plus de commandes. Un système de gestion d'utilisateurs et l'utilisation d'un IDS auraient aussi été appréciés.

Annexe A

Gestion de projet

A.1 Planning prévisionnel des charges

	Plan de charge prévisionnel					
			Charge en H / Participant			
Description de l'activité	**Charge en %**	**Charge en H**	**Verez**	**Ingargiola**	**Monniot**	**Tiennot**
Total	100%	200	50	50	50	50
Gestion de projet	**25%**	**49**	**13**	**13**	**12**	**11**
Réunion de lancement	2%	4	1	1	1	1
Planning prévisionnel et suivi d'activités	2%	4	1	1	1	1
Réunions de suivi	12%	24	6	6	6	6
Rédaction	7%	14	4	4	3	3
Site web	2%	3	1	1	1	0
Spécification	**4%**	**8**	**2**	**2**	**2**	**2**
Définition des fonctionnalités	4%	8	2	2	2	2
Conception préliminaire	**13%**	**26**	**6**	**6**	**7**	**7**
Définition d'un modèle de données	3%	6	1	1	2	2
Définition d'un format de fichiers associé	3%	5	1	1	1	2
Définition des fonctionnalités	6%	11	3	3	3	2
Définition des différents modules	2%	4	1	1	1	1
Conception détaillée	**23%**	**46**	**11**	**11**	**11**	**13**
Définition des structures de données	4%	7	2	1	2	2
Définition des fonctions	9%	17	4	4	4	5
Définition des tests unitaires	5%	10	3	2	3	2
Auto-formation GTK et Glade	2%	4	1	1	1	1
Maquettage des IHM (ascii, web, ...)	4%	8	1	3	1	3
Codage	**24%**	**47**	**12**	**11**	**12**	**12**
Écriture des interfaces (.h)	3%	5	2	1	1	1
Écriture du makefile	2%	4	1	1	1	1
Écriture des différentes fonctions	11 %	22	5	5	5	7
Tests unitaires	8%	16	4	4	5	3
Intégration	**4%**	**8**	**2**	**2**	**2**	**2**
Intégration des différents modules	2%	4	1	1	1	1
Tests d'intégration	2%	4	1	1	1	1
Soutenance	**8%**	**16**	**4**	**5**	**4**	**3**
Préparation de la soutenance	6%	12	3	4	3	2
Soutenance	2%	4	1	1	1	1

A.2 Répartition réelle des charges

	Suivi d'activités (charge consommée)					
			Charge en H / Participant			
Description de l'activité	**Charge en %**	**Charge en H**	**Verez**	**Ingargiola**	**Monniot**	**Tiennot**
Total	100%	200	53	49	49	49
Gestion de projet	**24%**	**47**	**13**	**12**	**12**	**10**
Réunion de lancement	2%	4	1	1	1	1
Planning prévisionnel et suivi d'activités	2%	4	1	1	1	1
Réunions de suivi	11%	22	6	6	6	4
Rédaction	7%	14	3	4	4	3
Site web	2%	3	2	0	0	1
Spécification	**4%**	**8**	**2**	**2**	**2**	**2**
Définition des fonctionnalités	4%	8	2	2	2	2
Conception préliminaire	**13%**	**26**	**6**	**6**	**7**	**7**
Définition d'un modèle de données	3%	6	1	1	2	2
Définition d'un format de fichiers associé	3%	5	1	2	1	1
Définition des fonctionnalités	6%	11	3	2	3	3
Définition des différents modules	2%	4	1	1	1	1
Conception détaillée	**25%**	**49**	**13**	**13**	**11**	**12**
Définition des structures de données	4%	7	2	1	2	2
Définition des fonctions	9%	18	5	4	4	5
Définition des tests unitaires	6%	11	4	3	3	1
Auto-formation GTK et Glade	2%	4	1	1	1	1
Maquettage des IHM (ascii, web, ...)	5%	9	1	4	1	3
Codage	**31%**	**62**	**17**	**14**	**15**	**16**
Écriture des interfaces (.h)	3%	5	2	1	1	1
Écriture du makefile	2%	4	1	1	1	1
Écriture des différentes fonctions	18%	36	10	8	8	10
Tests unitaires	9%	17	4	4	5	4
Intégration	**4%**	**8**	**2**	**2**	**2**	**2**
Intégration des différents modules	2%	4	1	1	1	1
Tests d'intégration	2%	4	1	1	1	1
Soutenance	**0%**	**0**	**0**	**0**	**0**	**0**
Préparation de la soutenance	0%	0				
Soutenance	0%	0				

Une différence est à noter au niveau du code en lui même, en effet, des difficultés sont apparues avec l'utilisation de libssh. Nous avons du la modifier, et la recompiler plusieurs fois avant de pouvoir reprendre le cours du projet.

A.3 Planning prévisionnel

Ce planning prévisionnel ne comprend pas les points rencontre à venir car nous ne pouvons pas donner de dates. Néanmoins nous prévoyons 8h pour ces points rencontre mais ils seront fonction des difficultés rencontrées.
Aussi nous prévoyons de rédiger le rapport et le wiki au fur et à mesure de l'avancement du projet, nous estimons la charge de travail à 15h pour le rapport et 7h pour le wiki.

Tâche	Date de début	Charge de travail	Date de fin	Participants
Réunion de lancement	02/02/12	1h30	02/02/12	Verez, Ingargiola, Monniot, Tiennot
Amphi génie logiciel	07/02/12	3h	07/02/12	Verez, Ingargiola, Monniot, Tiennot
Rédaction du cahier des charges	20/01/12	7h	14/02/12	Verez, Ingargiola, Monniot
Rédaction du planning prévisionnel	20/01/12	3h	14/02/12	Verez, Ingargiola
Rédaction du plan de charges prévisionnel	20/01/12	1h	14/02/12	Verez, Tiennot
Outils d'aide au développement	14/02/12	3h	15/02/12	Verez, Ingargiola, Monniot, Tiennot
Point rencontre	15/02/12	1h	15/02/12	Verez, Ingargiola, Monniot
Architecture : Firewall	15/02/12	8h	23/02/12	Verez, Ingargiola
Serveur SSH	15/02/12	20h	25/02/12	Monniot, Tiennot, Ingargiola
Programme et documentation d'installation	25/02/12	8h	12/03/12	Monniot
Logs (VM et code)	25/02/12	15h	12/03/12	Tiennot, Verez, Ingargiola
Architecture : IDS	12/03/12	8h	26/03/12	Verez
Interpréteur de commandes	12/03/12	15h	26/03/12	Tiennot, Ingargiola
Identification de l'attaquant	12/03/12	4h	26/03/12	Monniot
Livraison du pré-rapport	14/03/12	1h	14/03/12	Verez, Ingargiola, Monniot, Tiennot
Implémentation des commandes	23/03/12	20h	26/04/12	Ingargiola
Système de fichiers	26/03/12	20h	09/04/12	Tiennot, Monniot, Verez
Gestion des utilisateurs	09/04/12	20h	23/04/12	Tiennot, Monniot
Interface Web	09/04/12	15h	01/05/12	Verez, Ingargiola
Packaging du prototype	01/04/12	4h	11/04/12	Verez, Tiennot
Livraison prototype	11/04/12	1h	11/04/12	Verez, Ingargiola, Monniot, Tiennot
Fin des tests unitaires et tests d'intégration	23/04/12	10h	23/05/12	Verez, Monniot
Packaging et documentation des livrables	14/05/12	15h	22/05/12	Verez, Ingargiola, Monniot, Tiennot
Remise des livrables	23/05/12	1h	23/05/12	Verez, Ingargiola, Monniot, Tiennot
Préparation de la soutenance	22/05/12	16h	28/05/12	Verez, Ingargiola, Monniot, Tiennot
Soutenance	29/05/12	1h	30/05/12	Verez, Ingargiola, Monniot, Tiennot

A.4 Planning réalisé

Tâche	Statut	Charge de travail effectuée	Prévue pour le	Participants
Réunion de lancement	Réalisée	1h30	02/02/12	Verez, Ingargiola, Monniot, Tiennot
Amphi génie logiciel	Réalisée	3h	07/02/12	Verez, Ingargiola, Monniot, Tiennot
Rédaction du cahier des charges	Réalisée	7h	14/02/12	Verez, Ingargiola, Monniot
Rédaction du planning prévisionnel	Réalisée	3h	14/02/12	Verez, Ingargiola
Rédaction du plan de charges prévisionnel	Réalisée	1h	14/02/12	Verez, Tiennot
Outils d'aide au développement	Réalisée	3h	15/02/12	Verez, Ingargiola, Monniot, Tiennot
Point rencontre	Réalisée	1h	15/02/12	Verez, Ingargiola, Monniot
Architecture : Firewall	Réalisée	8h	23/02/12	Verez
Serveur SSH	Réalisée	20h	25/02/12	Verez, Monniot, Tiennot
Programme et documentation d'installation	Réalisé	8h	12/03/12	Monniot
Logs (VM et code)	Réalisée	15h	12/03/12	Verez
Architecture : IDS	Abandonnée	8h	26/03/12	Verez
Interpréteur de commandes	Réalisé	15h	26/03/12	Monniot, Verez, Tiennot
Identification de l'attaquant	Réalisée	4h	26/03/12	Monniot
Livraison du pré-rapport	Réalisée	1h	14/03/12	Verez, Ingargiola, Monniot, Tiennot
Implémentation des commandes	Réalisée	20h	26/04/12	Verez, Monniot, Ingargiola
Système de fichiers	Réalisée	20h	09/04/12	Tiennot, Ingargiola
Gestion des utilisateurs	Abandonnée	20h	23/04/12	Tiennot, Monniot
Interface Web	Réalisée	15h	01/05/12	Verez
Packaging du prototype	En attente	4h	11/04/12	Verez, Tiennot
Livraison prototype	Réalisée	1h	11/04/12	Verez, Ingargiola, Monniot, Tiennot
Fin des tests unitaires et tests d'intégration	Réalisée	10h	23/05/12	Verez, Ingargiola, Tiennot, Monniot
Packaging et documentation des livrables	Réalisée	15h	22/05/12	Verez, Ingargiola, Monniot, Tiennot
Remise des livrables	Réalisée	1h	23/05/12	Verez, Ingargiola, Monniot, Tiennot
Préparation de la soutenance	En attente	16h	28/05/12	Verez, Ingargiola, Monniot, Tiennot
Soutenance	En attente	1h	30/05/12	Verez, Ingargiola, Monniot, Tiennot

Certaines tâches ont été abandonnée, notament l'architecture IDS qui n'ai pas indispensable, et longue à mettre en place. La gestion des utilisateurs c'est révélée plus difficile que prévue.

Annexe B

Manuel utilisateur

B.1 Le honeypot

L'exécutable est présent dans /etc/init.d/honeyssh.
Il prend pour argument obligatoire l'ip du serveur.
Une option argument facultatif mais utile est -vv pour passer en mode verbeux.

B.2 L'interface graphique

L'interface web est disponible à l'ip 157.159.40.241.
Les identifiants sur nos serveurs sont root/honeyhoney.
On arrive dans l'aperçu qui affiche des graphs de stats, les dernières sessions SSH et les derniers événements d'état.
En allant dans les session, on peut voir l'ID mysql, la date, la durée, l'honey id (identifiant unique d'une session sur notre système), l'IP du client, le nombre d'entrées (tentatives de connexion + commandes).
En cliquant sur une session on observe le détail d'une session : on a l'ID mysql de l'entrée, l'hôte, le niveau de log, date, programme, l'honey ID, IP du client, Message.
Dans les deux cas on peut modifier trier selon les colonnes voulues et afficher le nombre de lignes voulues.
Dans les statistiques, On peut éditer ou ajouter un graph, on précisant le type de représentation graphique et en écrivant la requête SQL donnant le graph. La première colonne de résultat doit être l'abscisse ou le nom de la donnée et la deuxième l'ordonnée ou la quantité.
Dans l'onglet état, on a une interface similaire à celle des session. On observe les messages d'état non lié à une session particulière.

Annexe C

Tests unitaires

Des tests unitaires permettent de vérifier le bon fonctionnement de chaque module.
Pour lancer ces tests unitaires, il faut se placer dans le dossier ‘test‘ en ligne commande, puis exécuter ‘make‘. Le programme permettant de voir le résultat se trouve dans le dossier racine du projet, il se nomme ‘test_honeyssh‘

Exemple de test : ’test_bash.c’

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <string.h>

#include "bash.h"
#include "cmd.h"

#include "test_bash.h"

int bash_1(void)
{
  Bash_session *bash_session = malloc(sizeof(Bash_session));
  TINYTEST_ASSERT_MSG(bash_initialize(bash_session), "\nAssert bash_initialize\n");
  return 1;
}

int test_command(char *cmd, char *expected_answer, int expected_return)
{
  char *answer = malloc(100);
  int *size = malloc(sizeof(int));
  char *text_assert = malloc(100);
  char *text_str_equal = malloc(100);
  char *text_equal = malloc(100);
  session_identification session_identification = malloc(sizeof(Session_identification))
  session_identification->id = 0;
  session_identification->ip = "0.0.0.0";

  sprintf(text_assert, "\nAssert bash_exec commande %s\n", cmd);
  sprintf(text_str_equal, "\nAssert bash_exec commande %s retour\n", cmd);
  sprintf(text_equal, "\nAssert bash_exec commande %s size\n", cmd);

  if (expected_return)
  {
    TINYTEST_ASSERT_MSG(bash_exec(cmd, answer, size, session_identification), text_asser
```

```
  }
  else
  {
    TINYTEST_ASSERT_MSG(!bash_exec(cmd, answer, size, session_identification), text_asse
  }
  TINYTEST_STR_EQUAL_MSG(expected_answer, answer, text_str_equal);
  TINYTEST_EQUAL_MSG(strlen(expected_answer), *size, text_equal);

  free(answer);
  free(size);
  free(text_assert);
  free(text_str_equal);
  free(text_equal);

  return 1;
}

int bash_2(void)
{
  return test_command("ls", "\r\nREADME.txt\r\n", 1);
}

int bash_3(void)
{
  return test_command("pwd", "\r\n/root\r\n", 1);
}

int bash_4(void)
{
  return test_command("blablabla", "\r\n", 1);
}

int bash_5(void)
{
  return test_command("lsls", "\r\n", 1);
}

int bash_5b(void)
{
  return test_command("exit", "", 0);
}

int test_bash_shell(char *buffer, char *expected_buffer, int expected_size, int expected
{
  Bash_session *bash_session = malloc(sizeof(Bash_session));
  session_identification session_identification = malloc(sizeof(Session_identification))

  bash_initialize(bash_session);

  session_identification->id = 0;
  session_identification->ip = "0.0.0.0";
  int *size = malloc(sizeof(int));
  char *text_assert = malloc(100);
  char *text_str_equal = malloc(100);
  char *text_equal = malloc(100);
  char *buffer_tmp = malloc(100);
```

```
  strcpy(buffer_tmp, buffer);

  sprintf(text_assert, "\nAssert bash_shell buffer %s return\n", buffer);
  sprintf(text_str_equal, "\nAssert bash_shell buffer %s expected buffer\n", buffer);
  sprintf(text_equal, "\nAssert bash_shell buffer %s size\n", buffer);

  if (expected_return)
  {
    TINYTEST_ASSERT_MSG(bash_shell(buffer_tmp, size, bash_session, session_identificatio
  }
  else
  {
    TINYTEST_ASSERT_MSG(!bash_shell(buffer_tmp, size, bash_session, session_identificati
  }
  TINYTEST_STR_EQUAL_MSG(expected_buffer, buffer_tmp, text_str_equal);
  TINYTEST_EQUAL_MSG(expected_size, *size, text_equal);

  free(buffer_tmp);
  free(bash_session);
  free(size);
  free(text_assert);
  free(text_str_equal);
  free(text_equal);

  return 1;
}

int bash_6(void)
{
  return test_bash_shell("\x03", "^C\r\n", 4, 0);
}

int bash_7(void)
{
  return test_bash_shell("\x04", "^D\r\n", 4, 0);
}

int bash_8(void)
{
  return test_bash_shell("\x09", "\x09", 0, 1);
}

int bash_9(void)
{
  return test_bash_shell("\x0d ls", "\r\n# ", 4, 1);
}

int bash_10(void)
{
  return test_bash_shell("\x0d exit", "\r\n# ", 4, 1);
}

int bash_11(void)
{
  return test_bash_shell("\x1b\x5b\x41", "\x1b\x5b\x41", 0, 1);
}
```

```
int bash_12(void)
{
  return test_bash_shell("\x1b\x5b\x42", "\x1b\x5b\x42", 0, 1);
}

int bash_13(void)
{
  return test_bash_shell("\x1b\x5b\x43", "\x1b\x5b\x43", 0, 1);
}

int bash_14(void)
{
  return test_bash_shell("\x1b\x5b\x44", "\x1b\x5b\x44", 0, 1);
}

int bash_15(void)
{
  return test_bash_shell("\x7f", "\x7f", 0, 1);
}

/* TODO: Faire le cas où buffer = "\x7f" et bash_session->cursor_pos > 0 */

TINYTEST_START_SUITE(Bash);
  TINYTEST_ADD_TEST(bash_1);
  TINYTEST_ADD_TEST(bash_2);
  TINYTEST_ADD_TEST(bash_3);
  TINYTEST_ADD_TEST(bash_4);
  TINYTEST_ADD_TEST(bash_5);
  TINYTEST_ADD_TEST(bash_5b);
  TINYTEST_ADD_TEST(bash_6);
  TINYTEST_ADD_TEST(bash_7);
  TINYTEST_ADD_TEST(bash_8);
  TINYTEST_ADD_TEST(bash_9);
  TINYTEST_ADD_TEST(bash_10);
  TINYTEST_ADD_TEST(bash_11);
  TINYTEST_ADD_TEST(bash_12);
  TINYTEST_ADD_TEST(bash_13);
  TINYTEST_ADD_TEST(bash_14);
  TINYTEST_ADD_TEST(bash_15);
TINYTEST_END_SUITE();
```

Annexe D

Tests d'intégration

Nos tests d'intégration consiste en
- Passer l'ensemble des tests unitaires, qui permettent de vérifier chaque module
- Le succès de l'installation de l'ensemble des composants décrite dans le chapitre déploiement

Annexe E

Tests fonctionnels

Une fois le logiciel installé, pour le lancer il faut lancer l'exécutable 'honeyssh' présent dans /etc/init.d, suivi de l'adresse du serveur.
'# ./honeyssh 157.159.40.238'

E.1 Recette 1 : Enregistrement des logs

- '$ ssh root@157.159.40.238'
  The authenticity of host '157.159.40.238 (157.159.40.238)' can't be established.
  RSA key fingerprint is c0 :b6 :6e :f6 :a8 :d9 :51 :7e :02 :e6 :3b :e9 :77 :9f :05 :9a.
  Are you sure you want to continue connecting (yes/no) ?

- yes
  Warning : Permanently added '157.159.40.238' (RSA) to the list of known hosts.
  root@157.159.40.238's password : #
- ls
  bin
  boot
  debug.server.pcap
  dev
  etc
  home
  initrd.img
  lib
  lib64
  lost+found
  media
  mnt
  opt
  proc
  root
  sbin
  selinux
  srv
  sys
  tmp
  usr
  var
  vmlinuz
- Se connecter via Xen sur l'honeypot

– # tail /var/log/messages
  May 21 22 :05 :39 honeyssh-honeypot honeyssh[2094] : 1460253435 157.159.46.34 User root AUTHENTIFIED with pass root
  May 21 22 :05 :41 honeyssh-honeypot honeyssh[2094] : 1460253435 157.159.46.34 ls
– Se connecter via Xen sur le serveur de log
– # tail /var/log/honeypot/honeyssh.log
  May 21 22 :05 :39 157.159.40.238 honeyssh[2094] : 1460253435 157.159.46.34 User root AUTHENTIFIED with pass root
  May 21 22 :05 :41 157.159.40.238 honeyssh[2094] : 1460253435 157.159.46.34 ls

E.2 Recette 2 : Jusqu'à l'interface graphique

– '$ ssh root@157.159.40.238'
  The authenticity of host '157.159.40.238 (157.159.40.238)' can't be established.
  RSA key fingerprint is c0 :b6 :6e :f6 :a8 :d9 :51 :7e :02 :e6 :3b :e9 :77 :9f :05 :9a.
  Are you sure you want to continue connecting (yes/no) ?

– yes
  Warning : Permanently added '157.159.40.238' (RSA) to the list of known hosts.
  root@157.159.40.238's password : #
– pwd
  /

– Se connecter via Xen sur l'honeypot
– # tail /var/log/messages
  May 21 22 :28 :14 honeyssh-honeypot honeyssh[2107] : 303206803 157.159.46.34 User root AUTHENTIFIED with pass root
  May 21 22 :28 :25 honeyssh-honeypot honeyssh[2107] : 303206803 157.159.46.34 pwd
– Se connecter via Xen sur le serveur de log
– # tail /var/log/honeypot/honeyssh.log
  May 21 22 :28 :14 honeyssh-honeypot honeyssh[2107] : 303206803 157.159.46.34 User root AUTHENTIFIED with pass root
  May 21 22 :28 :25 honeyssh-honeypot honeyssh[2107] : 303206803 157.159.46.34 pwd
– Se connecter avec un navigateur web sur 157.159.40.241. Dans les sessions, une session datée du 2012-05-21 22 :28 :14 apparait.
  En cliquant dessus, on observe les lignes :
  – 130 157.159.40.238 warning 2012-05-21 22 :28 :14 honeyssh 303206803 157.159.46.34 User root AUTHENTIFIED with pass root
  – 131 157.159.40.238 notice 2012-05-21 22 :28 :31 honeyssh 303206803 157.159.46.34 pwd