C, la vraie vie

Éric Leblond

Freelance/OISF

16 octobre 2013

Introduction

Éric Leblond

- Initiateur du projet NuFW
- Co-fondateur de la société INL/EdenWall
- Membre de la coreteam Netfilter
  - Mainteneur de ulogd2
  - Developpement de nftables
- Core développeur Suricata :
  - Responsable de la partie acquisition
  - optimisation multicore
- Auteur d'articles dans MISC et LinuxMagazine France
- Consultant indépendant Open Source et Sécurité
- ...

Eric Leblond a.k.a Regit

FIGURE : Mon avatar

Eric Leblond a.k.a Regit

FIGURE : Mon avatar

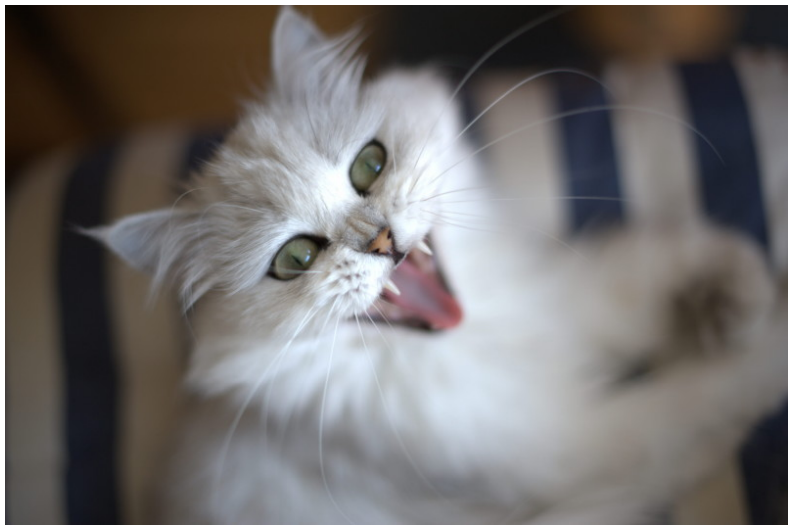


FIGURE : Mon chat, e-cat

Plus sur moi, contacts

- Pour aller plus loin :
  - Mon blog : `http://home.regit.org`
  - Mon Github : `https://github.com/regit/`
  - Site de Suricata : `http://www.suricata-ids.org/`
  - Site développeurs de Suricata : `https://redmine.openinfosecfoundation.org/`
- Me joindre :
  - Courriel : `eric@regit.org`
  - Twitter : Regiteric
  - Google + : Eric Leblond

Objectif de ces 2 jours

- Faire de vrai trucs en C
  - L'existence de bases dans ce langage est assumé
  - Orientation sécurité de la présentation
  - Pratique et exercice sous Linux
  - Inspiration d'un support de Benjamin Caillat
- C'est du C
  - Mais ça peut être amusant
  - Et on peut même en vivre
- Au programme
  - Quelques notions de bases
  - Des exercices pratiques

Mon optique

- Vous faire profiter de mon expérience professionelle
  - Parler de ce que j'utilise
  - De comment je l'utilise

Mon optique

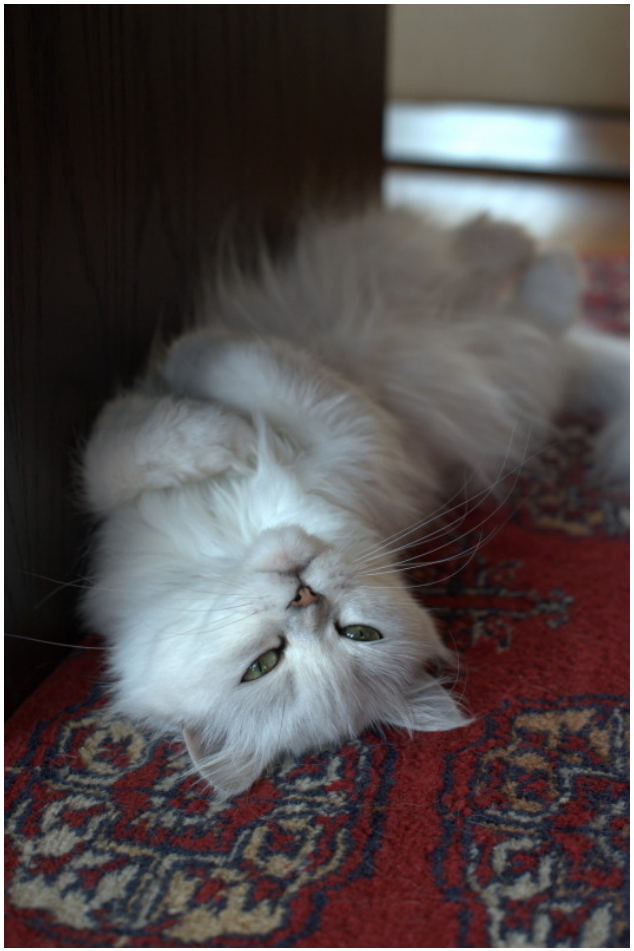
- Vous faire profiter de mon expérience professionelle
  - Parler de ce que j'utilise
  - De comment je l'utilise
- A vous de voir si cela s'applique à vous

Ne sous estimez jamais vos amis

- Read The Fucking Manual

Ne sous estimez jamais vos amis

- Read The Fucking Manual
- Car plus de RTFM, c'est plus de :



Le code c'est de la poésie

- Rusty Russel dans le noyau Linux :

```
#define SEC     * 1
#define MIN     * 60 SEC
#define HOUR    * 60 MIN
#define DAY     * 24 HOUR
```

Le code c'est de la poésie

- Rusty Russel dans le noyau Linux :

```
#define SEC     * 1
#define MIN     * 60 SEC
#define HOUR    * 60 MIN
#define DAY     * 24 HOUR
```

- Ne soyez pas Vogon :

```
for (i = 0; i < runmodes[runmode].no_of_runmodes; i++) {
    if (strcmp(runmodes[runmode].runmodes[i].name, custom_mode) == 0)
        return &runmodes[runmode].runmodes[i];
}
```

Humez l’odeur du bon code

Un mini projet sur Suricata

Socket de commande de Suricata

- Une socket unix de commande a été ajouté
- Elle permet de commander Suricata
- Elle utilise JSON pour son protocole
- Elle permet d'envoyer des fichiers à traiter

Un mini projet sur Suricata

Socket de commande de Suricata

- Une socket unix de commande a été ajouté
- Elle permet de commander Suricata
- Elle utilise JSON pour son protocole
- Elle permet d'envoyer des fichiers à traiter

Objectif

- Ecriture d'un embryon de client
- Ajout de fichier dans la liste des fichiers à traiter
- Récupération de la liste des fichiers en attentes

Humez l’odeur du bon code

Organisation des sources

Organisation du répertoire

```
[FIL] Makefile
[FIL] COPYING
[DIR] src
      [FIL] main.c
      [FIL] Makefile
[DIR] include
```

Exercice

Écriture d'un Hello world

- Écrire un simple programme C
- Affichant un message de test

Compilé depuis un Makefile

- Compilation de tous les fichiers C de src
- Édition liens et création d'un binaire
- Appelé SuricataC

Squelette de la fonction main

```
int main(int argc, char * argv[])
{
return 0;
}
```

Squelette de la fonction main

```
int main(int argc, char * argv[])
{
return 0;
}
```

Description des paramètres

- `argc` représente le nombre d'arguments passés au programme
- `argv` représente un tableau de pointeurs sur les différents arguments

Exercice

Écrire la liste des arguments

- On écrira la liste des arguments
- Un par ligne avec son numéro :

```
Arg1 : '$(argument 1)'
...
ArgN : '$(argument N)'
```

optarg

```
while ((opt = getopt(argc, argv, "nt:")) != -1) {
    switch (opt) {
    case 'n':
        flags = 1;
        break;
    case 't':
        nsecs = atoi(optarg);
        tfnd = 1;
        break;
    default:
        exit(EXIT_FAILURE);
    }
}
```

Exercice

Ajouter les options -h, -d et -v

- -h affichage un message d'aide
- -d [sec] ajoute un délai de [sec] entre chaque affichage
- -v affiche aussi la longueur des arguments

Socket : Une API essentielle

- Capable de gérer toutes les familles (domain) de socket
  - `AF_UNIX, AF_LOCAL` : Local communication
  - `AF_INET` : IPv4 Internet protocols
  - `AF_INET6` : IPv6 Internet protocols
  - `AF_IPX` : IPX - Novell protocols
  - `AF_NETLINK` : Kernel user interface device
  - `AF_X25` : ITU-T X.25 / ISO-8208 protocol
  - `AF_AX25` : Amateur radio AX.25 protocol
  - `AF_ATMPVC` : Access to raw ATM PVCs
  - `AF_APPLETALK` : Appletalk
  - `AF_PACKET` : Low level packet interface

Socket : Une API essentielle

- Capable de gérer toutes les familles (domain) de socket
  - `AF_UNIX, AF_LOCAL` : Local communication
  - `AF_INET` : IPv4 Internet protocols
  - `AF_INET6` : IPv6 Internet protocols
  - `AF_IPX` : IPX - Novell protocols
  - `AF_NETLINK` : Kernel user interface device
  - `AF_X25` : ITU-T X.25 / ISO-8208 protocol
  - `AF_AX25` : Amateur radio AX.25 protocol
  - `AF_ATMPVC` : Access to raw ATM PVCs
  - `AF_APPLETALK` : Appletalk
  - `AF_PACKET` : Low level packet interface
- Et leur variantes (type), par exemple pour `AF_INET` ou `AF_INET6` :
  - `SOCK_STREAM` : Socket TCP
  - `SOCK_DGRAM` : Socket UDP
  - `SOCK_RAW` : Socket IP

Socket :

La fonction `socket`

```
int socket(int domain, int type, int protocol);
```

- `domain` : famille de la socket
- `type` : type dans la famille de socket
- `protocol` : 0 ou protocole particulier si plusieurs protocoles existent pour le couple domain, type.

Socket :

La fonction `socket`

```
int socket(int domain, int type, int protocol);
```

- `domain` : famille de la socket
- `type` : type dans la famille de socket
- `protocol` : 0 ou protocole particulier si plusieurs protocoles existent pour le couple domain, type.

Et maintenant

- Une socket doit permettre de discuter
- On sait avec quel protocole
- La fonction socket ne dit pas avec qui.

Coté client (1/2)

Pour un protocol non connecté

```
ssize_t sendto(int sockfd, const void *buf, size_t len
               const struct sockaddr *dest_addr, sockl
```

La fonction `connect`

```
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

- On est bien avancé
- Un seul prototype pour gérer des IP et des chemins

Les structures sockaddr_*

La structure `struct sockaddr`

```
#define        __SOCKADDR_COMMON(sa_prefix) \
  sa_family_t sa_prefix##family
struct sockaddr {
    sa_family_t sa_family; /* Common data.  */
    char sa_data[14];      /* Address data. */
};
```

Les structures sockaddr_*

La structure `struct sockaddr`

```
#define     __SOCKADDR_COMMON(sa_prefix) \
  sa_family_t sa_prefix##family
struct sockaddr {
    sa_family_t sa_family; /* Common data.  */
    char sa_data[14];      /* Address data. */
};
```

Une structure générique

- Le champ family est accédée identiquement pour toute les familles
- différentiation de traitement possible
  - Le pointeur vers l'adresse
  - Et la taille sont suffisants
- De la donnée pour que les familles puissent ajouter du stockage

Les structures sockaddr_*

La structure `struct sockaddr_in`

```
struct sockaddr_in {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;             /* Port number. */
    struct in_addr sin_addr; /* Internet address. */
    /* Pad to size of 'struct sockaddr'. */
    unsigned char sin_zero[sizeof (struct sockaddr) -
                           __SOCKADDR_COMMON_SIZE -
                           sizeof (in_port_t) -
                           sizeof (struct in_addr)];
};
```

Une version appliquée

- Paramètres IP
- Un peu de bourrage

Les structures sockaddr_*

La structure `struct sockaddr_unix`

```
#define UNIX_PATH_MAX    108
struct sockaddr_un {
    sa_family_t sun_family; /* AF_UNIX */
    char sun_path[UNIX_PATH_MAX]; /* Access path */
};
```

La suite logique

- Un chemin comme data
- De longueur finie

Coté client (2/2)

La fonction `bind`

```
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

Coté serveur 1/3

Cinématique

1. Création de la socket avec l'appel à socket(2)
2. La socket est liée à une adresse avec bind(2)
3. Le souhait d'acceptation des connexions est signalé par l'appel à listen(2)
4. Les connexions sont acceptées avec accept(2)

Coté serveur 1/3

Cinématique

1. Création de la socket avec l'appel à socket(2)
2. La socket est liée à une adresse avec bind(2)
3. Le souhait d'acceptation des connexions est signalé par l'appel à listen(2)
4. Les connexions sont acceptées avec accept(2)

La fonction `bind`

```
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

Coté serveur 2/3

La fonction `listen`

```
int listen(int sockfd, int backlog);
```

La fonction `accept`

```
int accept(int sockfd, struct sockaddr *addr,
           socklen_t *addrlen);
```

- accept dépile les clients en attente
- une nouvelle socket est créée à chaque appel

Coté serveur 3/3

Attendre les clients

- select() la fonction historique
  - Limité en nombre de fd monitorés
  - Performances s'écroulent quand le nombre de fd augmentent
- poll() / epoll() les successeurs

Les bibliothèques

- Implémente les différents systèmes
- Choisisse le plus performant suivant l'OS
- Exemple : libev

La gestion des options

```
int getsockopt(int sockfd, int level, int optname,
               void *optval, socklen_t *optlen);
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```

Réutilisation de socket

```
int on = 1;
r = setsockopt(socket, SOL_SOCKET, SO_REUSEADDR,
               (void *) &on, sizeof(on));
```

Exercice

Compiler Suricata

- Installation des prérequis :

```
sudo apt-get -y install libpcre3 libpcre3-dbg libpcre3-dev \
build-essential autoconf automake libtool libpcap-dev libnet1-dev \
libyaml-0-2 libyaml-dev zlib1g zlib1g-dev libcap-ng-dev libcap-ng0 \
make flex bison git libmagic-dev
```

- Installation de libJANSSON
- Récupération des sources

```
git clone https://github.com/inliniac/suricata.git
```

- Compiler et installer

```
./configure
make
make install-full
```

Vérifier le fonctionnement

Vérifier le fonctionnement du mode socket

- Lancer Suricata avec l'option `-unix-socket`

  `suricata --unix-socket`
- Lancer suricatasc et vérifier que la connexion s'effectue bien.

Un protocole basé sur JSON

- JSON
  - Un protocole formatté
  - mais lisible
- Des messages structurés
  - Commande envoyée par le client

```
{
    "command": "$COMMAND_NAME",
    "arguments": { $KEY1: $VAL1, ..., $KEYN: $VALN }
}
```

  - Retour du serveur

```
{
  "return": "OK|NOK",
  "message": JSON_OBJECT or information string
}
```

Les messages de base

Échange initial

1. Message client (version actuelle est la 0.1)

```
{ "version": "$VERSION_ID" }
```

2. Réponse du serveur

```
{ "return": "OK|NOK" }
```

Demande de traitement de fichiers

```
{
  "command": "pcap-file",
  "arguments": { "filename": "smtp-clean.pcap",
                               "output-dir": "/tmp/out"
}
```

Exercice

- Ecrire un client se connectant à la socket de suricata
- Envoyant le message de version
- Et récupérant le retour du serveuro
- Afficher le retour sur la sortie standard
- On veillera à traiter les cas d'erreur du retour de chaque fonction appelée
- Les cas d'erreurs seront provoqués lorsque cela est possible

Exercice

- Générer un message d'ajout de fichiers
  - Le premier argument est le fichier à traiter
  - Le deuxième argument est le répertoire de sortie
- Valider l'exécution en observant le répertoire de sortie

API de manipulation de fichiers

Vérification d'états

- `access()` Vérifie les permissions d'accès à un fichier
- `stat()` Récpupère les informations sur un fichier

Accéder les fichiers

- `fopen()` = ouvre un fihier et récupère un pointeur sur un flux
- `fclose()` = ferme un flux associé à un fichier
- `fread()` = lit un bloc de données à partir d'un flux
- `fwrite()` = écrit un bloc de données dans un flux
- `fscanf()` = lit des données formatées à partir d'un flux
- `fprintf()` = écrit des données formatées dans un flux

Exercice

- Vérifier que le fichier en argument est lisible
- On vérifiera que le répertoire en est bien
- On utilisera notamment access() et stat()

Objectif

Liste chaînée est un méthode de gestion d'un ensemble d'objet permettant d'effectuer facilement des opérations d'ajout, suppression, tri, ...

Principe

Chaque objet en mémoire est encapsulé dans une structure contenant également un pointeur vers la structure suivante

Les différents types de listes chaînées

On distingue :

- Liste simplement chaînée : un élément contient un pointeur pointant sur l'élément suivant
- Liste doublement chainée : un élément contient deux pointeurs, un vers l'élément suivant, l'autre vers l'élément précédent

Exercice

- Lire la liste des fichiers à traiter depuis un fichier
  - On pourra utiliser le format `file;directory`
  - On utilisera une fonction dédiée pour la lecture du fichier
  - On stockera la liste des fichiers dans une liste chainée
- On ajoutera un flag permettant d'activer le choix du fichier
  - -f est un bon choix
  - Le mode par défaut reste le cas du fichier et du répertoire en argument
  - Les deux modes sont exclusifs
- Effectuer la validation des fichiers et répertoires depuis la structure de liste chainée
- On enverra les requêtes à suricata si tout est valide

Objectifs et intérêts

Objectifs

- Implémenter un ensemble de fonction
- Facilitation de l'utilisation
- API fixe

Intérêts

- Suivre les évolutions du protocole
- Déléguer son travail

Cinématique

1. Initialisation globale de la bibliothèque
2. Initialisation d'un contexte
3. Utilisation
4. Libération du contexte
5. Libération globale

Création et envoi d'objets JSON

```
json_t *server_msg = json_object();
json_object_set_new(server_msg, "return",
                    json_string("NOK"));
if (json_dump_callback(server_msg,
                       Callback,
                       this, 0) == -1) {
        SCLogWarning(SC_ERR_SOCKET,
                     "Unable to send command");
        goto error_cmd;
}
```

Avec

```
int Callback(const char *buffer,
             size_t size, void *data);
```

Lecture de message JSON

```
client_msg = json_loads(buffer, 0, &jerror);
version = json_object_get(client_msg, "version");
if(!json_is_string(version)) {
        return 0;
}
if (strcmp(json_string_value(version), "0.1") != 0)
        return 0;
```

Exercice

- Utiliser libJANSSON pour formater les messages.
- On pourra s'inspirer du code de Suricata
  - Dans le fichier `src/unix-manager.c`
  - et en particulier la fonction `UnixCommandFileList()`

Exercice

Ajouter une option pour ajouter un mode console comparable à suricatasc

Exercice

Utiliser readline pour la gestion de la ligne de commande

Exercice

Utiliser un thread pour poller l'avancée du traitement et avertir quand un fichier a été traité.