Lockatme
A screen lock with facial recognition abilities
Final report

David Anandanadaradja, Sagar Gueye, Bruno Inec,
Matthieu Kirschleger, Pierre-Louis Sergent

26 septembre 2018

Table des matières

1 Antériorité **2**
- 1.1 Présentation des membres du groupe . . . 3
  - 1.1.1 Contexte . . . 3
  - 1.1.2 Organisation et membres . . . 3
  - 1.1.3 Compétences . . . 3
- 1.2 Présentation du projet . . . 4
  - 1.2.1 Buts . . . 4
  - 1.2.2 Motivations . . . 4
  - 1.2.3 Linux . . . 4
  - 1.2.4 Open Source . . . 4

2 Stratégie de développement **5**
- 2.1 Méthodologie voulue . . . 6
- 2.2 Application réelle au projet . . . 6

3 Phase du projet – distribution des tâches **7**
- 3.1 Rétrospective - diagramme de Gantt specification . . . 8
- 3.2 Répartition réalisée . . . 8
  - 3.2.1 Difficultés rencontrées . . . 8
  - 3.2.2 Chronologie . . . 9
  - 3.2.3 Liste des tâches approximative . . . 15

4 Technologies utilisées **16**
- 4.1 Liste des technologies explorées . . . 17
- 4.2 Technologie utilisée – point technique . . . 17
  - 4.2.1 Xlib – python . . . 17
  - 4.2.2 Difficulté rencontrée - multithreading . . . 17

5 Version finale **18**
- 5.1 Présentation . . . 19
- 5.2 Utilisation - mode d'emploi . . . 19

6 Amélioration possible **20**
- 6.1 Interface graphique . . . 21
- 6.2 Le futur de lockatme . . . 21

Chapitre 1

Antériorité

1.1 Présentation des membres du groupe

1.1.1 Contexte

Dans le cadre de notre DUT Informatique à l'IUT Lyon 1, nous sommes tenus de réaliser un projet tuteuré durant le second semestre. Ce projet s'étendant également sur le troisième semestre, il a pour but de répondre à une problématique précise et de mettre en oeuvre les compétences acquises au cours de la formation. Il a aussi vocation à faire découvrir de nouveaux domaines et il nous permettra d'élargir nos savoirs à travers une auto-formation.

Ce projet se découpe en deux axes :

- Rédaction du cahier des charges (second semestre)
- Réalisation du projet en lui même (troisième semestre)

Malgré une liste de sujets proposés, notre groupe a voulu suivre ses propres motivations (présentées plus loin dans ce document) et a choisi de proposer un sujet à M. Vidal. L'intitulé est le suivant : Verrouillage et déverrouillage d'écran par reconnaissance faciale sous Linux.

1.1.2 Organisation et membres

L'équipe chargée de ce projet est constituée

- Tuteur du projet : M. Vincent VIDAL
- Chef du projet : M. Bruno INEC
- Membres : M. David ANANDANADARADJA, Mme Sagar GUEYE, M. Matthieu KIRSCHLEGER et M. Pierre-Louis SERGENT

1.1.3 Compétences

Notre projet comporte deux contraintes principales : il nécessite une bonne connaissance du langage Python et une maîtrise de Linux. L'impulsion de ces choix vient en grande partie du chef de projet qui possède une expérience importante dans ces deux domaines. David et Pierre-Louis possèdent quant à eux une expérience modérée dans l'utilisation de Linux (distribution Arch). L'ensemble des compétences individuelles est résumé ci-après :

Python :

- Confirmé : Bruno INEC
- Intermédiaire : Pierre-Louis SERGENT
- Débutant : Sagar GUEYE, Matthieu KIRSCHLEGER, David ANANDA

Linux :

- Confirmé : Bruno INEC
- Intermédiaire : David ANANDA, Pierre-Louis SERGENT
- Débutant : Sagar GUEYE, Matthieu KIRSCHLEGER

Comme le montre le listing précédent, les compétences du groupe sont très disparates. Cela peut apparaître comme une contrainte, mais en réalité cela constitue une véritable opportunité pour tous les membres. Ils vont ainsi pouvoir se former dans les domaines ci-après. Ils sont essentiels pour la suite des études et pour le milieu professionnel.

- Programmation : Linux, Python

- Rédaction cahier des charges : LaTeX
- Travail en équipe : réunion, communication, CI, modèle de développement

Nous étions donc motivés pour nous lancer dans un sujet avec nombre d'inconnus mais qui allait être fort enrichissant.

1.2 Présentation du projet

1.2.1 Buts

Le but premier de l'application est de déverrouiller un écran d'ordinateur, à l'aide d'une caméra, par reconnaissance faciale. Cependant cela implique de mettre en place un verrouillage d'écran sous Linux. Les URS spécifiques seront décrit plus tard dans ce document.

1.2.2 Motivations

Trois membres du groupe utilisent Arch Linux qui est une distribution minimale de Linux. Le fait de quitter Windows leur a permis de pleinement se concentrer sur la machine à un plus bas niveau, avec tous les avantages de liberté qu'offre une plateforme open source, mais aussi toutes les contraintes qui sont très formatrices et qui forcent à se pencher d'avantage sur le fonctionnement de ce système d'exploitation. Les trois utilisateurs cherchaient une manière de verrouiller/déverrouiller leur écran de manière sécurisée. Et l'idée de ce projet a fleuri suite à un article présent dans le magazine Linux Magazine/France n°203 : "Mettez en place un système de reconnaissance faciale".

1.2.3 Linux

Le développement du logiciel se fera sur Linux. Un tel projet sur Windows aurait été bien plus difficile concernant l'implémentation système mais aussi le code de l'application. De plus, l'OS est largement privilégié par les développeurs dans le monde de la programmation. C'est pourquoi nous avons choisi de réaliser notre projet sous Linux, qui s'adressera donc à un public famillié avec la CLI (Command Line Interface) et les autres aspects techniques. Des interfaces seront potentiellement développées à terme pour les utilisateurs de distributions plus user-friendly (comme Ubuntu).

1.2.4 Open Source

Le développement du projet se fera de manière complètement transparente et donc en open source. Ce choix est assez logique lorsque l'on réalise un programme pour Linux, car il s'inscrit exactement dans la politique des développeurs qui ont réalisé ce dernier. Cela possède de nombreux avantages : possibilité pour la communauté de contribuer au projet au travers de modifications du code, commentaires, rapport de bug, ...

Chapitre 2

Stratégie de développement

2.1 Méthodologie voulue

Lors de la réalisation du cahier des charges, nous avions réfléchi à la mise en place d'une méthodologie agile : SCRUM, afin de conserver une bonne visibilité sur le projet. Ainsi nous souhaitions réaliser des cycles ; un cycle correspondant au lapse de temps entre deux réunions avec le tuteur ; pour que ce dernier soit au maximum impliqué dans le projet et puisse nous aiguiller en cas de problème.

La méthodologie SCRUM implique également une intégration continue pour faire paraître à la fin de chaque cycle une nouvelle version. Pour cela l'idée était d'utiliser un service de décentralisation basé sur Git (GitHub), pour pouvoir merged les différentes branches dans la branche master.

2.2 Application réelle au projet

La réalisation du projet en lui même à nécessité de longues heures de recherches individuelles, notamment pour lire de la documentation et faire des "boûts" de code afin de mieux cerner ce que nous allions faire. C'est pourquoi la phase de développement à été plus courte que prévue.

Malgré cela nous avons réalisé de manière régulières des *stand-ups* pour répartir les recherches et pour parler de nos avancements et de nos problèmes. Lorsque nous avions plus de temps (période de vacances), il nous arrivait aussi de travailler quotidiennement en open-space en équipe de 2-3 personnes pour permettre un retour plus rapide sur le travail effectué et une mise en commun.

Pour ce qui est de la méthode SCRUM, nous avons mis en place des sprints de une à deux semaines. À la fin de ces derniers nous nous retrouvions, à l'occasion des stand-ups, afin de fermer les tickets résolus et d'ouvrir de nouveaux tickets pour le sprint d'après. Les outils qui nous ont servi pour exploiter le potentiel de cette méthodologie sont *Taiga* pour le suivi et la gestion des tickets et *Discord* pour la communication par message ou vidéo.

L'intégration continue a aussi été plus ou moins mise en place. Nous avons bien utilisé GitHub pour la mise en commun du travail. Chacun travaillant sur sa branche. Cependant, de beaucoup de recherche résultait assez peu de code. Il était donc difficile de séparer les tâches sur plusieurs branches pour ensuite merge sur master. La plupart du temps c'etait donc le chef du projet qui implémentait les features, en se basant sur les recherches de tout le monde.

Chapitre 3

Phase du projet – distribution des tâches

3.1 Rétrospective - diagramme de Gantt specification

Voici ci dessous le diagramme de Gantt que nous avions réalisé lors du cahier des charges pour le S2 :

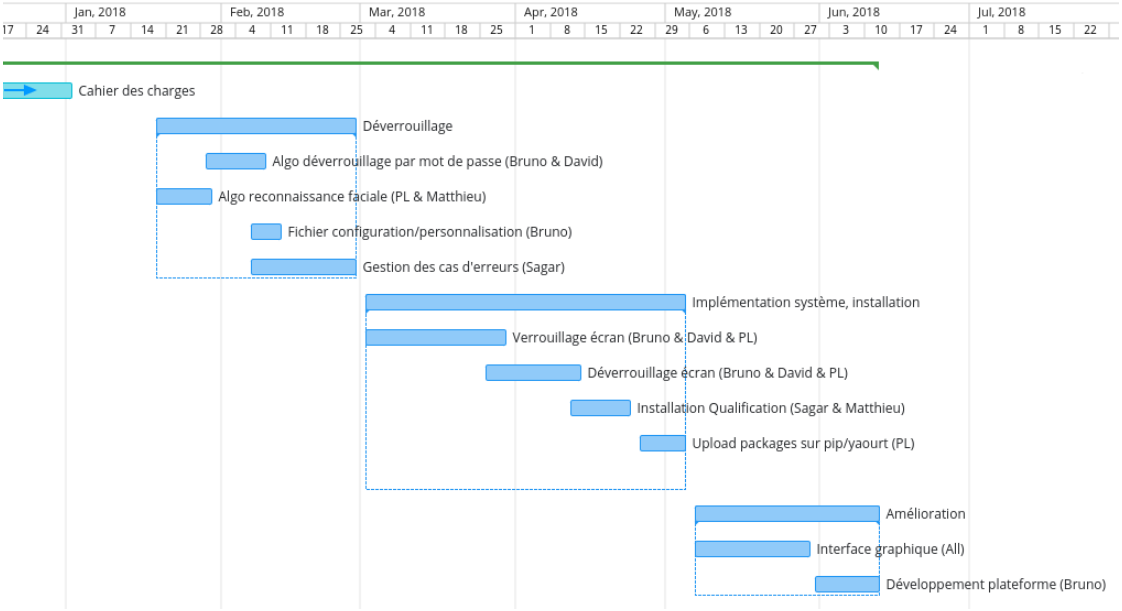

FIGURE 3.1 – diagramme de Gantt

Etant donnée que le sujet était très complexe et comportait de nombreuses zones d'ombres, nous avions fait un diagramme de Gantt, approximatif, quant à la durée des différentes tâches, ainsi qu'à leurs enchainements. Nous allons voir dans la partie qui suit que ce diagramme n'a pas été respecté. Des tâches ont été largement sous-estimé, notamment à cause de l'aspect technique de certains éléments qui ont été bloquant. Certaines tâches ont été supprimé ou requalifié. Suite à nos recherches nous avons changé de stratégie de développement plusieurs fois, il y a donc eu un changement dans les technologies utilisées ainsi que dans les moyens pour arriver à nos fins. Il existe cependant une certaine ressemblance, concernant les grandes parties, entre l'ancien diagramme et le déroulement réel. A savoir :

- **Recherche sur l'algorithme pour la reconnaissance faciale**
- **Verrouillage et déverrouillage de l'écran, implémentation système**
- **Upload pip, amélioration, test**

3.2 Répartition réalisée

3.2.1 Difficultés rencontrées

Comme dit auparavant, de nombreuses difficulté ont été rencontré plus tôt que prévu. Voici une liste exhaustive de celles-ci :

- **Répartition des recherches**
  Le sujet abordé était très technique et précis, il nécessitait donc beaucoup de recherche.

Mais la répartition était plutôt compliqué étant donnée que idéalement il aurait fallu que tout le monde possède une connaissance globale des technologies, qui pouvait théoriquement nous servir pour le développement.

- **Exploitation des recherches, mise en commun**
  Par la suite, les nombreuses heures de recherches ne nous avançaient guère pour la réalisation de notre projet. Il était difficile d'avoir une vision sur le long terme, quand nous stagnions parfois plusieurs jours sur un aspect compliqué. Il n'était pas évident de savoir concrètement à quoi allait nous servir certaines connaissances (d'ailleurs, un bon nombre n'ont finalement pas été utile). La mise en commun prenant beaucoup de temps. Il était parfois nécessaire de rédiger un petit document pour résumer les choses apprises durant une semaine afin de mieux pouvoir les exploiter et de déterminer si oui ou non elles seront utiles.

- **Compréhension, avancement éparse**
  Et donc suite à cela, venait le temps de la compréhension. Certains aspects techniques liés à l'architecture linux bloquaient certains. L'évolution des connaissances de chacun était assez éparpillé. De plus en plus, chacun recherchait de son propre côté en fonction de ses avancements. Cela à mener à une dispersion néfaste pour l'intérêt commun du groupe.

- **Changement fréquent de stratégie suite à de nouvelle découverte**
  Nos avancements nous ont irrémédiablement mené à changer d'idée plusieurs fois pour le développement (specification dans la partie suivante). Il était donc assez frustrant de se rendre compte que des heures de travail ne seront peut être pas utilisés concrètement dans le rendu final. Mais cela fait partie du projet, c'est à dire que pour en arriver au meilleur résultat possible, il était important d'explorer toutes les pistes possibles pour découvrir de nouvelles choses et ainsi réaliser le code le plus pertinent possible, dans le cadre de la philosophie de lockatme.

- **Impliquation de tout le groupe**
  Evidemment vu la liste qui est en train d'être faite, il est assez simple de deviner que cette phase à été bien compliqué pour certains membres du groupe, qui ont pu prendre du retard dans la compréhension de l'avancement. Mais il était quand même important de tenir tout le monde informé, notamment à travers des réunions régulières. Il a également été décidé, que les personnes, pour qui la partie développement pure était trop complexe, se verraient attibuer des tâches lié à la gestion, au déploiement ou à la présentation du projet. Toutefois il est essentiel que chaque membre ait connaissance des aspects techniques de l'application.

- **Répartition des tâches lors du développement**
  En lien avec le point précédent et avec un paragraphe du 2.2, dans le développement final il y avait assez peu de code (même si très complexe). Il n'y avait pas de séparation possible avec differentes couches, comme peu nous offrir le modèle MVC par exemple. C'est donc essentiellement le chef de groupe qui a implémenté la partie finale. Cependant les recherches de chacuns ont été prise en compte.

3.2.2 Chronologie

Quelles recherches ?

L'idée de la reconnaissance faciale nous était venu d'un article de magazine, ainsi la documentation sur cette algorithme est très abondante avec de multiples solutions. Cependant la partie implémentation système et verrouillage sur X11 nous était complètement inconnue il nous fallait donc nous plonger dans la recherche et la documentation pour ces deux aspects.

La première question a été, comment le système gère-t'il le déverrouillage ?

L'ensemble du groupe s'est donc plongé dans cette recherche.

PAM : Pluggable Authentication Modules

(Bruno, David, Pierre-Louis, travail en parallèle de Python into C)

En nous penchant sur des applications de lock sous Linux, tel que i3lock, lightDM, nous nous sommes rendus compte qu'elles utilisent toute PAM. Ce dernier permet à travers de configs files, contenant des modules, de gérer l'authentification. Les applications font appels à PAM en passant en paramètre le nom de la config file, qui elle même contient des modules d'authentification.

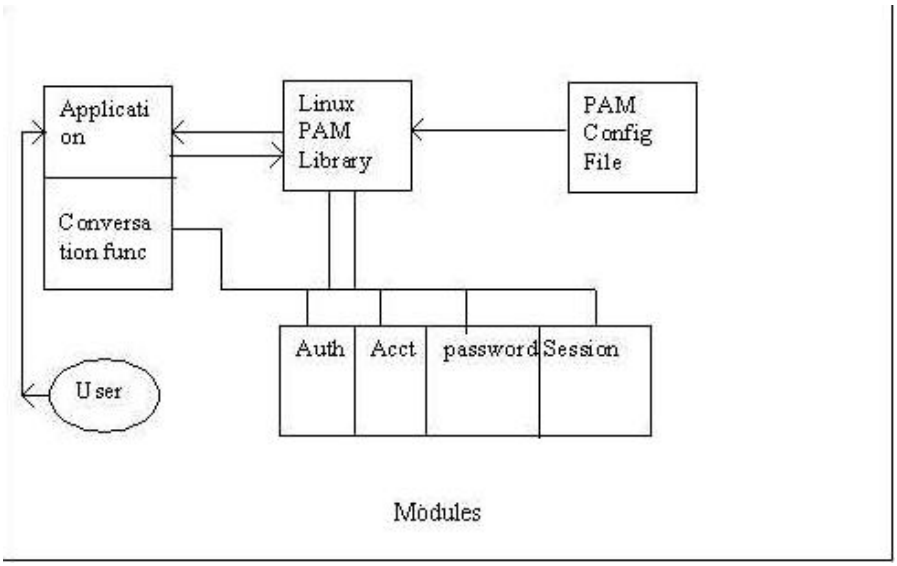

FIGURE 3.2 – schéma explication PAM

Nous nous sommes donc répartis les recherches à partir de la en trois parties (issues de la documentation officielle) :

- **The System Administrators' Guide (Pierre-Louis)**
  Cette partie concerne les PAM config file contenues dans le répertoire `/etc/pam.d`. Elles contiennent les modules PAM qui seront utilisés suite à l'appel de PAM par l'application. Il s'agit d'une syntaxe particulière où les modules sont regroupés en "type group" : account, auth, password et session. Après les types groups, sur la même ligne, vient la "control value", qui va définir le comportement de l'application en fonction des valeurs de retour des modules. Par exemple : required, sufficient, include, optional... Après la control value vient le nom du module PAM utilisé. Ensuite le quatrième argument constitue des options diverses que nous ne détailleront pas ici car inutile pour notre projet.

  Exemple : system-auth

  auth required `pam_unix.so try_first_pass nullok`
  auth optional `pam_permit.so`
  auth required `pam_env.so`

account required `pam_unix.so`
account optional `pam_permit.so`
account required `pam_time.so`

password required `pam_unix.so try_first_pass nullok sha512 shadow`
password optional `pam_permit.so`

session required `pam_limits.so`
session required `pam_unix.so`
session optional `pam_permit.so`

- **The Module Writers' Guide (Bruno)**

  Sur l'exemple du dessus on voit que le troisième argument de chaque ligne est un module PAM. Il est possible d'écrire soit même un module en C. C'est ce qui est traité dans cette partie. Cependant cet aspect est très lourd à comprendre. Nous nous sommes donc reposé sur des exemples tel que le repository simple-pam trouvé sur github. On a donc pu comprendre que chaque type group peut être gérer avec une fonction qui peut retourner `PAM_SUCCESS ou PAM_ERR`.

  Exemple :

```c
#include <security/pam_appl.h>
#include <security/pam_modules.h>

/* expected hook, this is where custom stuff happens */
PAM_EXTERN int pam_sm_authenticate( pam_handle_t *pamh, int flags,int argc,
  const char **argv ) {
        int retval;

        const char* pUsername;
        retval = pam_get_user(pamh, &pUsername, "Username: ");

        if (retval != PAM_SUCCESS) {
                return retval;
        }

        if (strcmp(pUsername, "papalouis") != 0) {
                return PAM_AUTH_ERR;
        }

        return PAM_SUCCESS;
}
```

- **The Application Developers' Guide (David)**

  Cette partie concerne l'appel du module PAM au sein d'une application. Nous avons retenu que toute utilisation de PAM débute avec un `pam_start()` qui contient la config file souhaité. Ensuite si l'appel s'est fait correctement, on peut utiliser la fonction `pam_authenticate()` par exemple pour lancer le processus d'authentification par mot de passe ou autre, en fonction des modules utilisés. A la fin du programme il faut fermer le PAM en utilisant `pam_end()`.

  Exemple :

```c
#include <security/pam_appl.h>
#include <security/pam_misc.h>
#include <stdio.h>

const struct pam_conv conv = {
        misc_conv,
        NULL
};

int main(int argc, char *argv[]) {
        pam_handle_t* pamh = NULL;
        int retval;
        const char* user = "nobody";

        if(argc != 2) {
                printf("Usage: app [username]\n");
                exit(1);
        }

        user = argv[1];

        retval = pam_start("face-auth-test", user, &conv, &pamh);

        // Are the credentials correct?
        if (retval == PAM_SUCCESS) {
                printf("Credentials accepted.\n");
                retval = pam_authenticate(pamh, 0);
        }

        // Can the accound be used at this time?
        if (retval == PAM_SUCCESS) {
                printf("Account is valid.\n");
                retval = pam_acct_mgmt(pamh, 0);
        }

        // Did everything work?
        if (retval == PAM_SUCCESS) {
                printf("Authenticated\n");
        } else {
                printf("Not Authenticated\n");
        }

        // close PAM (end session)
        if (pam_end(pamh, retval) != PAM_SUCCESS) {
                pamh = NULL;
                printf("check_user: failed to release authenticator\n");
                exit(1);
        }

        return retval == PAM_SUCCESS ? 0 : 1;
}
```

Utilisation

Suite à cette phase de recherche, nous commençons à tester un programme en C tout simple. Il prend en argument un nom d'utilisateur, le programme fait appel à PAM avec une config file contenant un module d'authentification écrit par nos soins. Il permet simplement de vérifier si le nom d'utilisateur passé en paramètre est bien celui de la session active. Nous utilisons par la suite le module `pam_unix.so` qui permet également de faire une vérification par mot de passe.

Mais nous nous heurtons à un problème, tous les modules PAM sont écrit en C et notre algorithme de reconnaissance faciale est en Python. Il nous fallait donc un module en C qui permettrait de demander une reconnaissance faciale en guise d'authentification.

Suite à des recherches nous trouvons un module déjà écrit en C : `pam_authentication`. Ce module très bien fait possède une interface graphique pour "entraîner" le programme avec des visages capturés avec la webcam. Nous utilisons donc ce module pour des programmes simples qui simulent un verrouillage dans la CLI, une fois lancé il demandent une reconnaissance faciale, et en cas d'échec demandent un mot de passe. Ensuite nous avons pu modifier un locker déjà existant, en implémentant deux `pam_start` différent, chacun se lançant indépendamment en fonction du choix de l'utilisateur : reconnaissance faciale ou mot de passe. Nous avions donc un locker fonctionnel, qui laisse le choix à l'utilisateur de déverrouiller par reconnaissance faciale ou par mot de passe.

Problèmes

Cependant cela n'était pas satisfaisant, malgré les nombreuses heures pour arriver à un tel résultat. Nous n'utilisions pas notre algorithme de reconnaissance faciale en Python. Le module écrit en C `pam_authenticate` n'était pas de nous. L'utilisation d'un autre locker nous a montré que l'écriture d'une solution pour verrouiller l'écran est loin d'être simple, et nous avions, à ce moment la, un peu négligé ce point, mais nous souhaitions écrire un locker nous même.

C'est pourquoi nous avons assigné de nouvelles tâches à chacun.

- Un duo faisant des recherches complémentaires sur une solution pour implémenter du python dans du code en C afin de faire appel à notre algorithme de reconnaissance faciale dans un module PAM (Matthieu et Sagar). Nous avions déjà assigné cette tâche lors de la découverte de PAM.
- Le reste du groupe travaillant à l'écriture d'un locker en C.

Python into C

Il existe effectivement une bibliothèque en C qui permet d'intégrer du Python dans le code : Python.h.

Exemple :

```c
#include <Python.h>

int main () {
    // PyObject est un wrapper Python autour des objets qu'on
    // va échanger entre le C et Python.
    PyObject *retour, *module, *fonction, *arguments;
    char *resultat;

    // Initialisation de l'interpréteur. A cause du GIL, on ne peut
    // avoir qu'une instance de celui-ci à la fois.
    Py_Initialize();
```

```
    // Import du script.
    PySys_SetPath(".");
    module = PyImport_ImportModule("biblio");

    // Récupération de la fonction
    fonction = PyObject_GetAttrString(module, "yolo");

    // Création d'un PyObject de type string. Py_BuildValue peut créer
    // tous les types de base Python.
    arguments = Py_BuildValue("(s)","Leroy Jenkins");
    // Appel de la fonction.
    retour = PyEval_CallObject(fonction, arguments);

    // Conversion du PyObject obtenu en string C
    PyArg_Parse(retour, "s", &resultat);

    printf("Resultat: %s\n", resultat);

    // On ferme cet interpréteur.
    Py_Finalize();
    return 0;
}
```

Locker

Lors des recherches pour écrire le locker, nous nous sommes inspiré de la même application que celle utilisée auparavant, pour ajouter la solution de reconnaissance faciale. Il s'agit de sxlock. C'est un locker qui se veut simple et qui fait appel à PAM. L'écriture d'un programme de verrouillage est très complexe et nécessite de se pencher sur la documentation de X11 et plus particulièrement sur Xlib qui est la bibliothèque interface pour l'implémentation en C du protocole X. Il se trouve qu'il existe des bindings Python pour cette bibliothèque. De plus nous nous sommes rendus compte que sxlock est en fait un fork du projet slock, un locker assez basique qui n'utilise pas PAM.
En parallèle les recherches sur Python.h montre qu'il est difficile d'utiliser cette bibliothèque pour écrire un module PAM.

Abandon de PAM

Notre but étant de faire une application en Python nous avons donc pris à ce moment un virage très serré en abandonnant l'idée d'utiliser PAM pour le déverrouillage de lockatme. En effet, à ce moment là il fallait soit écrire un module PAM en C soit utiliser le module `pam_authenticate`. Nous manquions de temps et l'utilisation de bindings Xlib semblait donc être plus simple. Cette solution correspondait mieux à l'esprit du projet.

Xlib

Suite à ce virage compliqué il était difficile de bien répartir les tâches. Chacun à donc réalisé des recherches sur Xlib et sur les bindings mais la répartition compliquée et la difficulté de la tâche restante c'est Bruno notre chef de projet qui a implémenté la version finale de lockatme en rassemblant les connaissances réunis par le groupe et en utilisant l'algorithme de reconnaissance faciale du S2. Ce point sera développé dans le 4.

3.2.3 Liste des tâches approximative

Liste tâches :

- Recherche PAM : Bruno, David et Pierre-Louis (10h chacun)
- Recherche Python into C : Sagar et Matthieu (10h)
- Compréhension PAM avec programme et écriture module de test : David et Pierre-Louis (15h)
- Programme test python into C : Sagar et Matthieu (8h)
- Travail compréhension locker : Bruno, David et Pierre-Louis (10h chacun)
- Travail écriture module Python into C : Bruno et Sagar (5h)
- Modification locker pour reconnaissance faciale : Pierre-Louis et Matthieu (10h environ)
- Recherche Xlib : Bruno et Pierre-Louis (10h et 3h)
- Implémentation finale : Bruno (20h)
- Rédaction README : Sagar (4h)
- Rédaction compte rendu final : Pierre-Louis et Bruno (10h)
- Slides présentation finale : Sagar et Matthieu (6h)

Chapitre 4

Technologies utilisées

4.1 Liste des technologies explorées

- PAM : Pluggable Authentication Modules
  - Compréhension globale
  - Config files
  - Ecriture module
  - Implémentation lors du développement
  - Module pam authentication
- C
  - Python.c
  - Ecriture locker et module
- Python
  - Algorithme reconnaissance faciale
  - Bindings Xlib
- Xlib

4.2 Technologie utilisée – point technique

4.2.1 Xlib – python

4.2.2 Difficulté rencontrée - multithreading

Chapitre 5

Version finale

5.1 Présentation

5.2 Utilisation - mode d'emploi

Chapitre 6

Amélioration possible

6.1 Interface graphique

6.2 Le futur de lockatme