UNIVERSITÉ
CAEN
NORMANDIE

Rapport de stage

Projet IANEC

Analyse forensique et traitement d'archives numériques

Titouan LE BRET

Année universitaire 2024 – 2025

Stage de 1ère année de Master Cybersécurité

Université de Caen Normandie & ENSICAEN

Encadrants :
Tanguy GERNOT, Emanuel GIGUET, Christophe ROSENBERGER

Organisme d'accueil : Laboratoire GREYC

Projet soutenu par le ministère de la Culture

Sommaire

1 Introduction

L'arrivée des ordinateurs comme outils de création et de traitement de textes constitue un défi majeur pour la préservation et l'accès à ces documents, autrefois produits de manière manuscrite. En effet, de plus en plus d'archives numériques commencent à vieillir, tandis que les formats et les normes évoluent rapidement, laissant certaines données dans l'oubli. Se pose alors le problème de la récupération et de l'analyse de fichiers ou de disques pour lesquels nous ne disposons plus nécessairement des logiciels ou des systèmes d'exploitation d'origine.

1.1 Contexte et objectifs du stage

Ce stage s'inscrit dans le cadre d'un projet financé par le Ministère de la Culture, dont l'objectif est de fournir un service numérique innovant à l'Institut Mémoires de l'Édition Contemporaine (IMEC). Cette association conserve les archives de maisons d'édition, d'auteurs contemporains et, plus largement, d'acteurs de la vie littéraire. Si la majorité de ces fonds est encore constituée de manuscrits, l'IMEC reçoit depuis plusieurs années des archives nativement numériques, regroupant une grande quantité de données aux formats variés et couvrant la période de 1986 à 2019.

L'objectif principal du stage est d'automatiser l'analyse des images disque constituant ces fonds, en définissant une organisation normalisée des données afin d'assurer un traitement uniforme. Il s'agit d'identifier automatiquement les systèmes d'exploitation, les applications installées, les fichiers utilisateurs et leurs métadonnées, ainsi que de qualifier les fichiers selon leur type et d'identifier les logiciels et versions ayant permis leur création. La virtualisation des environnements originaux est également envisagée afin de permettre la visualisation des documents dans leur contexte natif. Le stage prévoit également la génération de rapports d'analyse des fonds intégrant des représentations graphiques et des tableaux, pour faciliter l'exploitation et l'interprétation des données.

Source : Convention de stage.

1.2 Le GREYC et le projet IANEC

Le stage s'est déroulé au sein du GREYC (Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen), un laboratoire rattaché à l'Université de Caen Normandie, au CNRS et à l'ENSICAEN. Les recherches qui y sont menées couvrent de nombreuses thématiques liées aux sciences du numérique. Ce laboratoire est composé de 7 équipes différentes, chacune ayant sa spécialité. J'ai pour ma part effectué mon stage au sein de l'équipe SAFE, qui mène des activités de recherche en sécurité informatique, notamment en biométrie, architecture et modèles de sécurité ainsi qu'en forensique.

Plus spécifiquement, ce travail s'inscrit dans le projet IANEC (Informatique et Archives Numériques de l'Édition Contemporaine), qui vise à développer des méthodes et des outils pour préserver et exploiter les archives numériques de l'IMEC. Ce projet cherche ainsi à répondre à un enjeu croissant : traiter et récupérer des données de types variés sur des fonds d'archives de différents formats et époques. Le projet est co-encadré par Tanguy GERNOT, Emanuel GIGUET et Christophe ROSENBERGER.

Dans ce cadre, mon stage a porté principalement sur deux aspects : la mise en place de méthodes d'analyse forensique pour identifier et qualifier automatiquement les fichiers, et l'exploration de solutions de virtualisation permettant de reconstituer les environnements d'origine des fonds étu-

diés. D'autres aspects, tels que le montage de disques ou l'étude des types MIME, ont également été brièvement abordés au cours du stage.

1.3 Organisation du rapport

Ce rapport s'organise en huit grandes parties, la première étant cette introduction. Il se poursuit par une présentation de l'organisation du projet et de l'état de l'art de celui-ci au moment de mon arrivée, en expliquant les travaux précédemment réalisés par d'autres stagiaires ainsi que par mes maîtres de stage.

La partie suivante décrit brièvement les choix techniques réalisés, mon placement précis dans ce projet, ainsi que quelques mots sur les différentes collaborations que j'ai pu avoir avec d'autres stagiaires.

Les quatrième et cinquième parties sont consacrées aux deux principaux projets menés durant mon stage. La première porte sur la conception d'une base de données destinée à stocker les empreintes de fichiers issus d'anciens systèmes Mac OS. La seconde traite de l'exploitation de cette base, afin de faciliter le travail des archivistes dans l'analyse des fonds numériques de l'IMEC.

Je consacrerai ensuite une partie aux travaux complémentaires que j'ai pu réaliser, afin de mettre en avant les projets secondaires que j'ai développés en parallèle des principaux ou au cours de mes recherches.

Pour finir, je présenterai certains résultats obtenus au cours des analyses, afin d'illustrer concrètement l'apport des outils développés. Enfin, la dernière partie sera consacrée à la conclusion du rapport et à l'exposé de quelques perspectives d'amélioration.

2 État de l'art et description du projet

Cette première partie pose le cadre dans lequel s'inscrit mon stage, elle se compose de trois sous-parties. Tout d'abord, l'organisation du projet et les différents axes de travail répartis entre les stagiaires seront décrits afin d'avoir une vue globale du projet avant de décrire plus précisément le rôle qui a été le mien. Ensuite, j'expliciterai certains concepts et notions qui seront utiles tout au long du rapport afin de faciliter la compréhension de chacun et de m'assurer que les termes utilisés soient compris de la bonne manière. Pour finir, je parlerai brièvement des outils existants et des différents travaux que nous avons pu réutiliser dans notre stage.

2.1 Organisation du projet et répartition des axes

Pour ce projet, qui s'étend sur plusieurs années, différents stages ont été organisés. Lors de ma période, nous étions six stagiaires répartis sur deux grandes thématiques. La première, sur laquelle je n'ai pas travaillé, visait à aider les archivistes dans leurs analyses grâce à l'implémentation de fonctionnalités liées à l'intelligence artificielle, pour faciliter le tri des fichiers et tenter de déterminer un critère de communicabilité, en excluant par exemple les fichiers à caractère obscène, les dossiers privés ou des informations sensibles. Pour ma part, j'ai travaillé sur la seconde thématique, centrée sur l'analyse forensique des fonds d'archives, la récupération de fichiers obsolètes ainsi que la virtualisation de vieux systèmes d'exploitation. J'ai été en étroite collaboration avec Guillaume HAUTOT et Matthias DAVID sur ces travaux.

Dans ce cadre, nous avions de nombreuses interactions avec Guillaume et Matthias, qui ont parfois contribué aux travaux que je présente ici. Nous échangions également régulièrement avec

les stagiaires travaillant sur l'intelligence artificielle ainsi qu'avec nos maîtres de stage, dont les conseils ont grandement orienté nos développements.

Différentes présentations ont été organisées tout au long du stage pour rendre compte de l'avancée de nos travaux à l'IMEC. La préparation et la réalisation de ces échanges nous ont permis de structurer nos résultats, de suivre l'évolution des travaux des autres et de mieux comprendre l'avancée globale du projet, tout en ayant la possibilité de réorienter nos travaux en fonction des besoins réels de l'IMEC.

2.2 Définitions et notions importantes

Afin de faciliter la lecture du rapport, voici certaines notions techniques utilisées au cours du stage :

- **Analyse forensique :** C'est un domaine scientifique qui consiste à récupérer des informations, des fichiers ou encore des métadonnées à partir de supports numériques variés. Dans le cadre de ce stage, l'objectif est d'analyser des supports ayant appartenu à des auteurs contemporains et conservés à l'IMEC, afin d'en extraire et de classer des éléments qui peuvent être importants. Il s'agira donc par exemple de retrouver des ébauches de textes, des versions anciennes d'écrits ou d'autres documents créés avec des logiciels ou éditeurs aujourd'hui disparus dans le but de reconstituer et d'avoir accès à l'information initiale.

- **Image disque :** Fichier représentant la copie d'une partition ou d'un système de fichiers complet. Dans notre cas, ces images proviennent principalement de clés USB, disquettes ou disques durs. Les formats utilisés varient fortement : *dmg, raw, iso*, etc. Ces copies permettent d'analyser et de travailler sur les fonds sans risquer d'altérer le support initial, tout en simplifiant l'accès aux données, qui sont parfois stockées sur de vieux supports difficiles d'accès (exemple : disquettes ZIP).

FIGURE 1 – Exemple de disquette ZIP utilisée dans l'un des fonds.

- **Fonctions de hachage :** Les fonctions de hachage sont des algorithmes qui associent à des données de différentes tailles, une chaîne de longueur fixe que l'on appelle *hash* ou *empreinte*. Ces fonctions tentent de minimiser la probabilité de collision, c'est à dire le fait que deux données différentes soient associés à la même empreinte. Cette caractéristique est particulièrement utile dans notre cas, car elle permet de garantir qu'à chaque fichier est attribuée une empreinte unique, évitant ainsi de stocker et comparer les fichiers a partir de leur contenu, qui peut être volumineux.

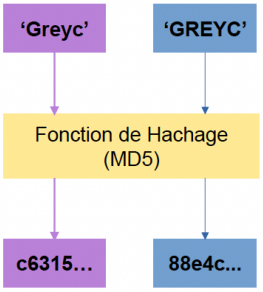


FIGURE 2 – Exemple fonctionnement fonction de hachage (MD5).

- **Virtualisation :** La virtualisation permet l'exécution de différents systèmes d'exploitation sur une seule machine grâce à un logiciel de virtualisation. Les systèmes d'exploitation virtualisés doivent cependant avoir la même architecture que le système hôte. Il existe plusieurs virtualiseurs, mais l'un des plus connus est *VirtualBox*.
- **Émulation :** Contrairement à la virtualisation, l'émulation permet l'exécution d'un plus grand nombre de systèmes d'exploitation. Il n'y a ici plus de contrainte sur l'architecture : l'émulateur traduira les instructions de l'architecture émulée vers l'architecture hôte pour les requêtes. Ce procédé est toutefois plus lent que la virtualisation. Nous avons utilisé pour notre part **QEMU**, **SheepShaver** et **Mini vMac**. En utilisant ces trois logiciels, j'ai pu exécuter différents systèmes d'exploitation, notamment des anciennes versions de Mac OS. Cela m'a permis de comprendre le fonctionnement de ces vieux systèmes, de travailler dessus, d'y installer des applications et d'y récupérer des fichiers de configuration.
- **Charset (jeu de caractères) :** un charset est un schéma de codage utilisé pour représenter des caractères. Les charset récents reposent sur Unicode, une norme qui attribue un point de code unique (code point) à chaque caractère. Par exemple, le caractère « é » a pour code Unicode U+00E9 et peut être encodé en UTF-8 par les octets `C3 A9`.

2.3 Outils existants et réutilisés

Comme expliqué précédemment, ce stage s'inscrit dans un projet plus vaste. À notre arrivée, certains travaux étaient déjà en cours, tandis que d'autres étaient aboutis. Je vais ici présenter brièvement les différentes ressources que nous avons pu récupérer.

Tout d'abord, nous avons eu accès au rapport de Hamid Ben Omar, qui avait effectué son stage dans le projet IANEC en 2024. Ce rapport portait principalement sur l'émulation des systèmes d'exploitation Mac OS 6, 7, 8 et 9 grâce aux logiciels SheepShaver et Mini vMac. Il explorait également les possibilités de transfert de fichiers entre la machine hôte et la machine virtuelle, permettant ainsi l'installation d'applications et la récupération de fichiers générés par le système. Une partie de son rapport abordait l'idée de créer un plugin Autopsy pour trier les images selon différents critères. Nous ne l'avons pas utilisé directement, mais sa lecture nous a inspiré dans la conception de notre propre plugin pour répondre à notre problématique.

Une des grandes difficultés du projet résidait dans la quantité et l'hétérogénéité des données et des formats de stockage. Pour nous aider, mes tuteurs, Emanuel Giguet et Tanguy Gernot, nous

ont fourni différents scripts qu'ils avaient développés afin de faciliter l'accès aux données et de gérer le montage des différents types de disques.

Lors de mes développements, j'ai également exploré l'identification des types de fichiers à travers plusieurs recherches et tests. Je suis tombé sur un article rédigé par l'équipe SAFE du GREYC, notamment par *Adrien Dubettier, Tanguy Gernot, Emanuel Giguet et Christophe Rosenberger*. Bien que ce document n'ait pas été publié dans le cadre du projet, il m'a été particulièrement utile.

Enfin, il me paraît important de mentionner la plateforme GDIP, développée par le GREYC pour l'investigation numérique. Cette plateforme a été évoquée à plusieurs reprises au cours du stage. Il était parfois question d'y intégrer certaines des fonctionnalités que nous avons développées, ce qui justifie la traduction de notre outil initialement développé en Java vers Python. Cette question sera détaillée dans la partie consacrée à notre développement (partie 4). Cependant, en raison de la durée limitée du stage et du temps nécessaire pour ces implémentations, nous n'avons pas pu approfondir ce point.

3 Méthodologie et organisation du travail

Cette troisième partie présente l'organisation de mon travail au sein du projet IANEC, ainsi que les principaux choix techniques que nous avons faits. Je parlerai également des interactions que j'ai pu avoir avec les autres stagiaires sur certaines thématiques.

3.1 Choix techniques et orientation du travail

Pour répondre aux besoins du projet, plusieurs choix techniques ont dû être faits :

- Utilisation de **PostgreSQL** pour la gestion de la base de données, permettant de favoriser le travail collaboratif. Le fait de créer un serveur de base de données allait donc nous permettre d'incrémenter la base a plusieurs et en même temps, mais aussi de toujours travailler sur la même version de la base. Ce choix nous a paru le plus judicieux dans notre développement mais aussi dans le but d'une utilisation a l'IMEC. En effet les archivistes pourraient alors eux aussi travailler à plusieurs sur la base de données, sans avoir besoin d'en avoir une copie chacun, sachant que celle-ci peut faire plusieurs dizaines de gigaoctets.

- Pour l'incrémentation de la base de données, nous avons choisi d'utiliser **Python** avec le module **psycopg2**, afin de développer une interface facilitant l'ajout de données correctement formatées dans les bonnes tables de la base.

- Développement initial d'un plugin Autopsy en **Java** pour l'exploitation de notre base de données. Ce langage était recommandé par la documentation d'Autopsy, le logiciel étant lui-même développé en Java. J'ai suivi la structure proposée afin de créer un *File Ingest Module*, permettant d'analyser chacun des fichiers d'une source ajoutée dans l'interface d'Autopsy, de les annoter et de les catégoriser. La documentation proposait également des instructions pour un développement en *Python*, mais cela imposait la contrainte de la version 2.7 du langage, rendant le développement particulièrement périlleux et délicat, tout en enlevant certaines fonctionnalités notamment pour la communication avec les serveurs de base de données.

- Le développement en **Java** nous a cependant limité dans l'ajout de certaines fonctionnalités. Après quelques discussions, l'idée est apparue de créer une version « traduite » du plugin, totalement indépendante d'Autopsy et développée en **Python**. Cela permettrait alors de se libérer des contraintes de l'application, de faciliter le développement d'autres fonctionnalités et de préparer une éventuelle intégration dans la plateforme GDIP.

3.2 Mon rôle dans le projet

Mon rôle a consisté à virtualiser plusieurs anciennes versions de Mac OS afin de récupérer les fichiers génériques des systèmes et des applications les plus courantes, dans le but de constituer une base de données d'empreintes de ces fichiers. Cette base permet ensuite de comparer les fichiers des fonds de l'IMEC et de catégoriser ceux qui y sont présents comme "génériques". Ces fichiers, qu'il s'agisse de fichiers systèmes, d'applications ou de configuration, ne présentent pas d'intérêt particulier pour l'archiviste puisqu'ils ne contiennent pas de données utilisateur. En effet, une correspondance avec la base de données indique qu'aucune donnée utilisateur n'est présente.

3.3 Interactions et collaboration

J'ai régulièrement échangé avec les autres stagiaires du projet, ce qui a été très enrichissant pour avancer dans mon travail. J'ai notamment collaboré avec Guillaume HAUTOT sur la création et la gestion de la base de données, afin de développer différents outils adaptés à ce serveur. Nous avons beaucoup travaillé avec Matthias DAVID lors de la conception de son application de centralisation de plusieurs solutions de virtualisation, pour créer une interface commune facilitant l'accès aux différents environnements.

En dehors de ces collaborations, une grande partie de nos échanges était sur des questions, des conseils techniques ou des informations sur des problèmes déjà rencontrés par d'autres. Ces interactions quotidiennes nous ont permis de profiter du travail de chacun sur le projet, d'éviter des erreurs et de bénéficier de l'expérience de chacun.

4 Création d'une base d'empreintes pour les systèmes Mac OS anciens

La première préoccupation que nous avons eue a été de créer la base de données d'empreintes, afin de faciliter le stockage des différents fichiers que nous allions récupérer sur les vieux systèmes.

4.1 Création du serveur et choix du schéma de base

Comme expliqué plusieurs fois dans le rapport, nous avons décidé d'utiliser un serveur de base de données PostgreSQL, afin de permettre le travail à plusieurs sur les bases. Cependant, nous n'avions alors pas encore d'idée sur la manière dont nous allions formaliser le stockage des fichiers. Après quelques recherches, nous avons trouvé la base de données RDS du NIST. Pour rappel, le NIST est une agence du département du Commerce des États-Unis (Wikipedia). Il est notamment en charge de la création de certaines normes informatiques, ce qui nous a orientés vers l'utilisation de ce schéma de base de données, mais aussi vers la récupération de cette base pour tenter différentes expérimentations avec nos fonds. Ainsi, nous avons récupéré une version de la

base RDS (la version legacy-minimal : empreintes d'applications antérieures à 2014), qui est une base de données SQLite contenant environ 160 millions d'empreintes de fichiers.

Nous avons donc rencontré ici l'un de nos premiers problèmes : le format de cette base n'était pas directement compatible avec notre serveur. Il nous a donc fallu chercher une solution efficace pour transformer cette base SQLite en une base PostgreSQL. Après quelques recherches et tentatives, nous avons finalement trouvé l'outil **pgloader**, un utilitaire permettant de transformer différentes bases de données, et qui, dans notre cas, permet de migrer de **SQLite** vers **PostgreSQL**. Nous avons donc pu intégrer cette base dans notre serveur de base de données et en créer une autre pour stocker les empreintes que nous allions calculer nous-mêmes. Pour décrire rapidement le schéma de ces deux bases, elles se composent de cinq tables. L'une d'entre elles est la table *VERSION*, qui n'est pas liée aux autres et ne sert qu'à indiquer la version de la table, donc en résumé son nom, la date de parution et une brève description. Pour les quatre autres tables, elles sont toutes liées. La table *MFG* stocke les fabricants, *OS* les systèmes d'exploitation, *PKG* les applications et *FILE* les fichiers. Nous aurons donc, pour chaque fichier, une entrée dans la table FILE, qui sera liée à une application dans la table PKG, elle-même liée à un système d'exploitation dans la table OS et à un fabricant dans la table MFG. Pour une meilleure compréhension de ce schéma, une modélisation est disponible ci-dessous.

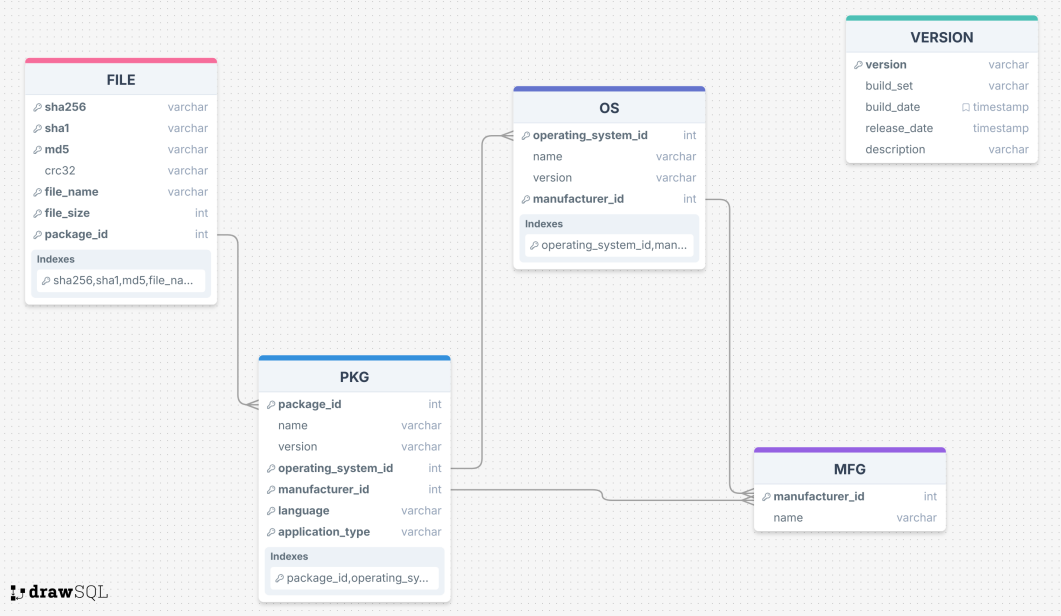


FIGURE 3 – Schéma de la base de données au format RDS du NIST

Nous avons alors noté que ce schéma fournit, pour chaque fichier, quatre empreintes issues d'algorithmes différents : SHA256, SHA1, MD5 et CRC32. Cependant, dans notre cas, nous n'avions besoin que du calcul d'une seule empreinte, afin d'être efficaces lors du calcul et de ne pas gaspiller de mémoire. Il nous fallait donc un compromis : pouvoir calculer rapidement nos empreintes, que celles-ci ne soient pas trop volumineuses pour le stockage, tout en conservant la possibilité de réaliser les comparaisons futures. Mais il ne fallait pas non plus négliger le risque de collision.

C'est pourquoi nous avons éliminé le CRC-32, qui est plutôt utilisé pour détecter les modifications accidentelles des données. Il nous restait alors MD5, SHA1 et SHA256. Malgré la faible sécurité de MD5, aujourd'hui considéré comme cassé, la probabilité de collisions accidentelles reste

quasi nulle. Si l'on considère un milliard de fichiers, la probabilité d'une collision est d'environ 10^{-21}. De plus, dans notre cas, personne n'a l'intention de générer volontairement une collision. Nous avons donc choisi de n'utiliser que des empreintes calculées avec l'algorithme MD5.

4.2 Développement d'une application d'incrémentation

Une fois notre base de données créée, nous avons eu besoin de créer différents outils/fonctions pour pouvoir parcourir les répertoires que nous voulions hacher et calculer les empreintes tout en les ajoutant à la base.

4.2.1 Fonctions de parcours et de hachage de disque

Après de multiples tentatives de création de scripts pour parcourir et hacher les fichiers d'un répertoire donné, il s'est avéré qu'un outil existait déjà. Il s'agit de l'utilitaire open-source **rHash**, disponible sur GitHub. Nous avons intégré son utilisation via un appel système dans un script Python auquel nous fournissons un répertoire. Cet appel nous renvoie ensuite la liste des fichiers et des empreintes que nous pouvons alors écrire dans notre base.

4.2.2 Outils d'incrémentation de la base

Dans le but d'enrichir notre base, il était impératif de pouvoir ajouter des entrées dans les tables OS, PKG et MFG, pour pouvoir les relier à nos fichiers. L'implémentation la plus simple était la création d'un script Python utilisable en ligne de commande. Cet utilitaire permet de se connecter à la base de données depuis un terminal mais aussi d'ajouter des entrées pour chacune des tables de façon simplifiée. Pour les fichiers, on fournit par exemple un répertoire ainsi que les informations relatives à l'application et au fabricant, mais aussi au système d'exploitation de provenance. Cet utilitaire, bien que fonctionnel, nous a paru un peu redondant et désagréable à utiliser. Tout en sachant que nous allions avoir besoin d'ajouter énormément de fichiers et de différentes entrées dans la base, et que les archivistes de l'IMEC pourraient eux aussi vouloir incrémenter cette base, nous avons décidé de développer une interface graphique. Celle-ci nous permet de simplifier encore l'ajout d'informations dans la base. Elle se compose de quatre volets, un pour chacune des tables de la base (excepté pour la table VERSION). Tout cela permet de fluidifier les ajouts, en affichant directement la liste des applications, des systèmes d'exploitation et des fabricants de la base dans des menus déroulants, pour ainsi lier facilement leurs identifiants à l'ensemble des fichiers que nous souhaitons ajouter.

FIGURE 4 – Interface graphique de l'application d'incrémentation de la base de données

4.2.3 Encodage des noms de fichiers

L'un des problèmes rencontrés lors de ce développement a été les noms de fichiers. Ces fichiers provenaient en effet de différents systèmes d'exploitation, qui ont des encodages différents. Ainsi, certains fichiers n'avaient pas le même nom dans le système virtualisé et dans la base de données. La base, gérant des noms en UTF-8, supposait les entrées dans cet encodage alors que parfois elles étaient encodées dans d'autres charset. J'ai alors pu en discuter avec mes maîtres de stage, Tanguy GERNOT et Emanuel GIGUET, qui m'ont rappelé la présence d'un script permettant d'encoder des noms de fichiers afin de faciliter leur stockage. Ce script utilise le principe d'encodage URL (*URL ENCODE et URL DECODE*), en remplaçant les caractères spéciaux par des séquences codées en UTF-8. J'ai donc testé l'efficacité de cette fonction d'encodage pour voir si elle pouvait s'appliquer à mon cas, en transposant le script initialement écrit en **PHP** vers du **Python**.

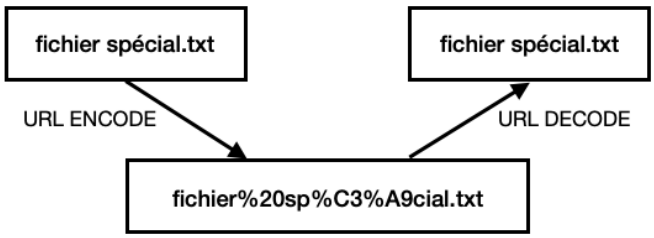


FIGURE 5 – Utilisation des fonctions URL ENCODE et URL DECODE

Cependant, lors de l'encodage en URL, ma version du script supposait que la chaîne était en UTF-8. Dans mon cas, c'était un peu plus complexe : elle pouvait être encodée en MacRoman, Macintosh encoding, UTF-8 ou autre. J'avais donc des caractères invalides lors du décodage de chaînes MacRoman que j'avais préalablement encodées en URL, car les charset n'attribuaient pas les mêmes caractères au même code point.

Exemple :

- Nom du fichier en MacRoman : TravailNoté.txt
- URL encode : TravailNot%8E.txt
- URL decode (UTF-8) : TravailNot ?.txt

Cela provoquait une perte d'information. Pour pallier ce problème, j'ai décidé de convertir tous les noms de fichiers vers l'UTF-8 avant d'effectuer l'encodage URL. Il m'a ensuite fallu tester l'efficacité de cette méthode. Après la création de différents fichiers avec des noms spéciaux (émoticônes, caractères spéciaux, accents), j'ai tenté d'encoder et de décoder avec ma méthode pour vérifier si les noms restaient inchangés. Ces tests se sont révélés concluants. Je pouvais alors stocker les noms dans la base, puis les récupérer et, en les décodant, retrouver le nom initial du fichier sans aucune perte. Cela m'a également permis de mieux comprendre les subtilités liées à l'encodage des chaînes de caractères et aux différences entre les charset.

4.3 Travail sur la virtualisation des systèmes Mac OS

Une fois notre base de données et nos outils d'incrémentation prêts et fonctionnels, nous avions besoin de données à y inscrire et avons donc commencé nos travaux sur la virtualisation.

4.3.1 Mac OS 7, 8 et 9

Pour les systèmes Mac OS 7 à 9, le rapport d'Hamid Ben Omar était très détaillé et nous fournissait une marche à suivre claire et guidée. C'est grâce à cela que nous avons pu rapidement commencer à travailler sur les machines virtuelles, en comprenant aisément le fonctionnement de l'émulateur **SheepShaver** et la manière d'y installer ces versions de Mac OS.

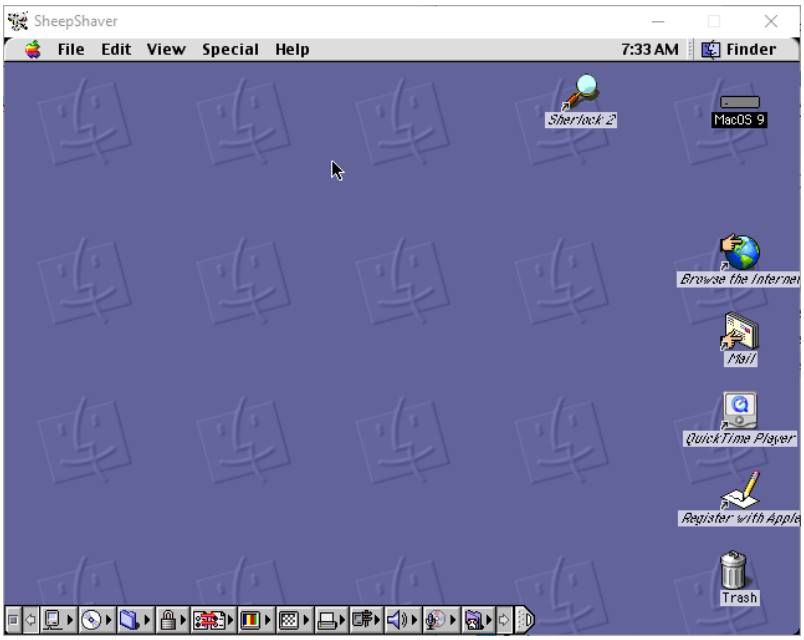


FIGURE 6 – Émulation par SheepShaver de MacOS 9

Une fois les systèmes d'exploitation installés, nous avons récupéré des listes d'applications présentes sur les fonds d'archives. Une partie de ces informations nous a été directement transmise par l'IMEC, qui avait recensé ces applications, mais j'ai également créé une petite application permettant d'analyser les fichiers systèmes et de récupérer une liste d'applications potentiellement présentes sur un disque. Cette application sera décrite dans la partie 6.

Un travail de collecte d'applications a donc été réalisé sur différents sites, mais la source la plus fiable et la plus complète reste Macintosh Repository. Nous avons ainsi pu tenter d'installer ces applications et de les utiliser pour, par exemple, ré-ouvrir des fichiers de fonds et obtenir la mise en forme correcte. Suite à cela, nous avons pu constater une réelle différence entre la visualisation des fichiers dans l'éditeur d'origine et dans des applications récentes comme LibreOffice, ce qui nous a encouragés à lister ces applications ainsi que la procédure d'installation afin de permettre aux archivistes de se replacer dans le contexte de la bonne mise en forme.

Après avoir installé toutes ces applications, nous avons pu commencer la collecte des fichiers systèmes et des applications. La méthodologie adoptée était la suivante :

- Pour chaque système, stocker d'abord les fichiers du système vierge.
- Pour chaque application installée sur ce système, ajouter uniquement :
  - Les fichiers nouvellement créés par l'installation.
  - Les fichiers dont le contenu a été modifié suite à l'installation.
- L'unicité des fichiers était assurée grâce à un critère basé sur l'identifiant du système d'exploitation dans la base, afin d'éviter de stocker plusieurs fois des fichiers déjà présents dans la version vierge.

4.3.2 Mac OS X 10.0 à 10.4

La virtualisation de Mac OS X n'est pas supportée par SheepShaver. Pour cela, nous avons alors recherché d'autres émulateurs, en commençant par VirtualBox, qui fait de la virtualisation et non de l'émulation. Les expérimentations n'ont donc pas été concluantes, nous pouvions seulement virtualiser des versions de Mac OS X plus récentes et sur une architecture identique à celle de l'hôte, donc x86-64.

Ces versions de Mac OS étant trop récentes et pour le moment peu intéressantes pour nos fonds, nous avons cherché un nouvel émulateur. C'est alors que nous avons trouvé QEMU, l'un des émulateurs les plus connus et offrant le plus de choix possible. Cependant, l'émulation de Mac OS X reste assez périlleuse. C'est donc après de multiples tentatives que nous avons réussi à trouver des versions stables des disques d'installation des systèmes pour les versions majeures : 10.0, 10.1, 10.2, 10.3 et 10.4. Il nous fallait désormais réussir a installer des applications sur ces machines et à récupérer les fichiers pour les ajouter a notre base de données.

Le système de dossier partagé fait partie des fonctionnalités manquantes dans cette émulation, et nous n'avions donc pas de moyen d'installer des applications sur les systèmes émulés. Nous avons alors exploré trois pistes. La première consistait simplement à tenter de configurer une connexion par pont entre l'émulateur et l'hôte pour avoir accès à Internet sur l'émulateur et y télécharger directement les disques d'installation d'applications. Même si elle paraissait intéressante, cette piste a vite été bloquée par de multiples difficultés lors de nos tentatives, pour la plupart sûrement dues aux limitations de l'émulateur.

La deuxième, qui paraissait être la plus simple, consistait à monter le disque du système émulé sur notre système hôte. Une fois cela fait, nous n'aurions plus qu'à y copier les disques téléchargés dans un dossier, puis à relancer l'émulateur pour y accéder. C'était sans compter sur une limitation logicielle due au fait que les systèmes Mac OS sont au format HFS+, format qui peut être monté sur une machine Linux, mais seulement en lecture (read-only). Aucune modification n'était donc possible, refermant ainsi cette possibilité.

Nous avons finalement trouvé une solution simple grâce à une fonctionnalité native de Mac OS X. Ce système d'exploitation met en effet à disposition la possibilité de créer facilement, depuis l'interface graphique, un serveur FTP. Ce type de serveur permet de transférer des fichiers entre des machines sur un réseau.

La mise en place était relativement simple :

- Dans Mac OS X, il suffisait d'activer le partage de fichiers via FTP dans les préférences système.
- Il fallait ensuite configurer la VM pour qu'elle utilise une interface réseau de l'hôte, afin que les deux machines se trouvent sur le même réseau local.
- Une fois cela en place, il suffisait de se connecter à l'adresse IP de la VM depuis l'hôte, via l'explorateur de fichiers, en utilisant les identifiants configurés dans Mac OS X.

Grâce à cette configuration, nous pouvions échanger des fichiers entre l'hôte et la VM et inversement. Cependant, les connexions n'étaient pas très stables : elles suffisaient pour déposer sur la VM des fichiers depuis l'hôte (jusqu'à 1 Go), mais lorsque nous voulions récupérer tous les fichiers du système pour les transférer vers l'hôte ($\approx 10GO$), la connexion ne tenait pas.

Nous avons donc conservé le serveur FTP uniquement pour les transferts de l'hôte vers la VM. Pour ajouter les fichiers à notre base, nous montions le disque du système émulé sur l'hôte, ce qui permettait de copier directement l'ensemble des fichiers. Après quelques modifications des droits d'accès, nous pouvions alors incrémenter notre base de données avec tous ces fichiers.

5 Exploitation de la base d'empreintes

Après tout ce travail sur l'émulation de système Mac OS et l'incrémentation de notre base de données, nous avions une grande quantité d'empreintes, et pouvions alors commencer notre travail sur l'exploitation de cette base. Ces travaux ont pour but de permettre, grâce à la base créée, de différencier les fichiers systèmes (ou d'applications) des fichiers utilisateur.

5.1 Plugin Autopsy

Pour ce faire, nous avons rapidement compris que l'IMEC utilisait, comme beaucoup de personnes faisant de l'analyse forensique, Autopsy. Il s'agit d'une plateforme (interface) basée sur The Sleuth Kit, une bibliothèque logicielle fournissant des outils pour faciliter l'analyse de différentes sources.

Son interface est simple à prendre en main, elle se compose d'un volet à gauche qui liste l'arborescence de la source ajoutée, et de deux volets à droite. Le volet supérieur affiche les détails sur chacun des fichiers du répertoire dans lequel nous nous trouvons dans l'arborescence. Le volet inférieur droit permet d'afficher des informations plus précises sur le fichier sélectionné. Voir figure ci-dessous.

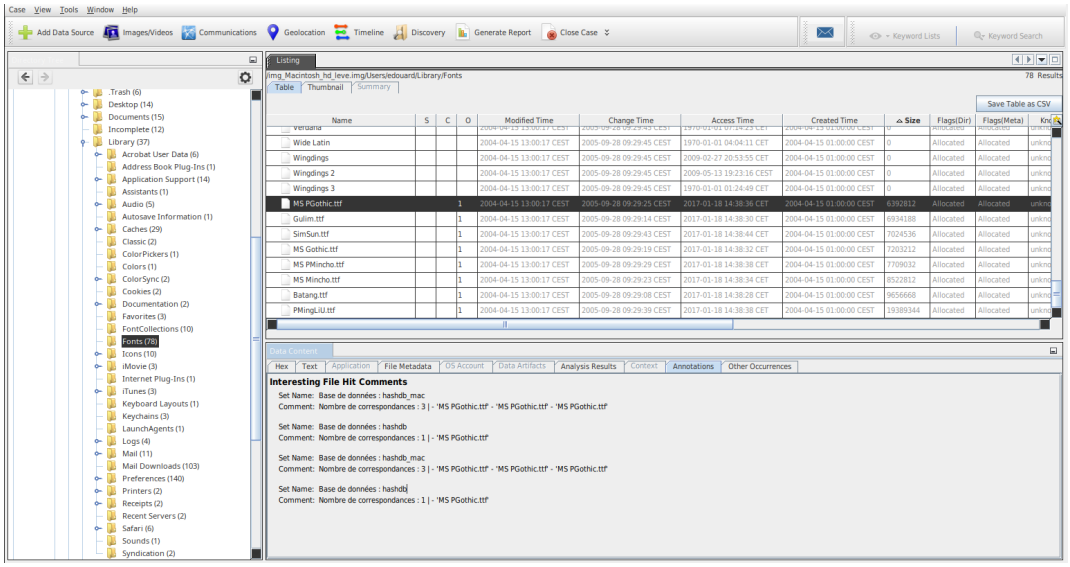


FIGURE 7 – Interface de la plateforme Autopsy

Autopsy offre la possibilité d'installer des plugins pour ajouter de nouvelles fonctionnalités. Nous disposions de la documentation officielle permettant le développement d'un plugin en Java. Bien que peu détaillée, cette documentation nous a fourni les bases nécessaires au développement.

Notre choix s'est porté sur un *File Ingest Module*, un type de plugin parcourant l'intégralité des fichiers via une fonction *process*. Durant cette fonction, nous pouvons effectuer toutes nos tâches.

Algorithm 1 Pseudo-code très simplifié de la fonction process

```
1: function PROCESS(fichier)
2:     if fichier invalide then
3:         return
4:     end if
5:     Calculer le hash du fichier
6:     Chercher les correspondances dans les bases
7:     Ajouter les informations à Autopsy
8:     return OK
9: end function
```

Ainsi, le plugin permet d'annoter les fichiers analysés avec leurs correspondances dans toutes les bases de données d'empreintes de notre serveur. Il est paramétrable directement depuis l'interface d'Autopsy, permettant à tout utilisateur de l'utiliser avec son propre serveur, à condition que le schéma de la base soit identique.

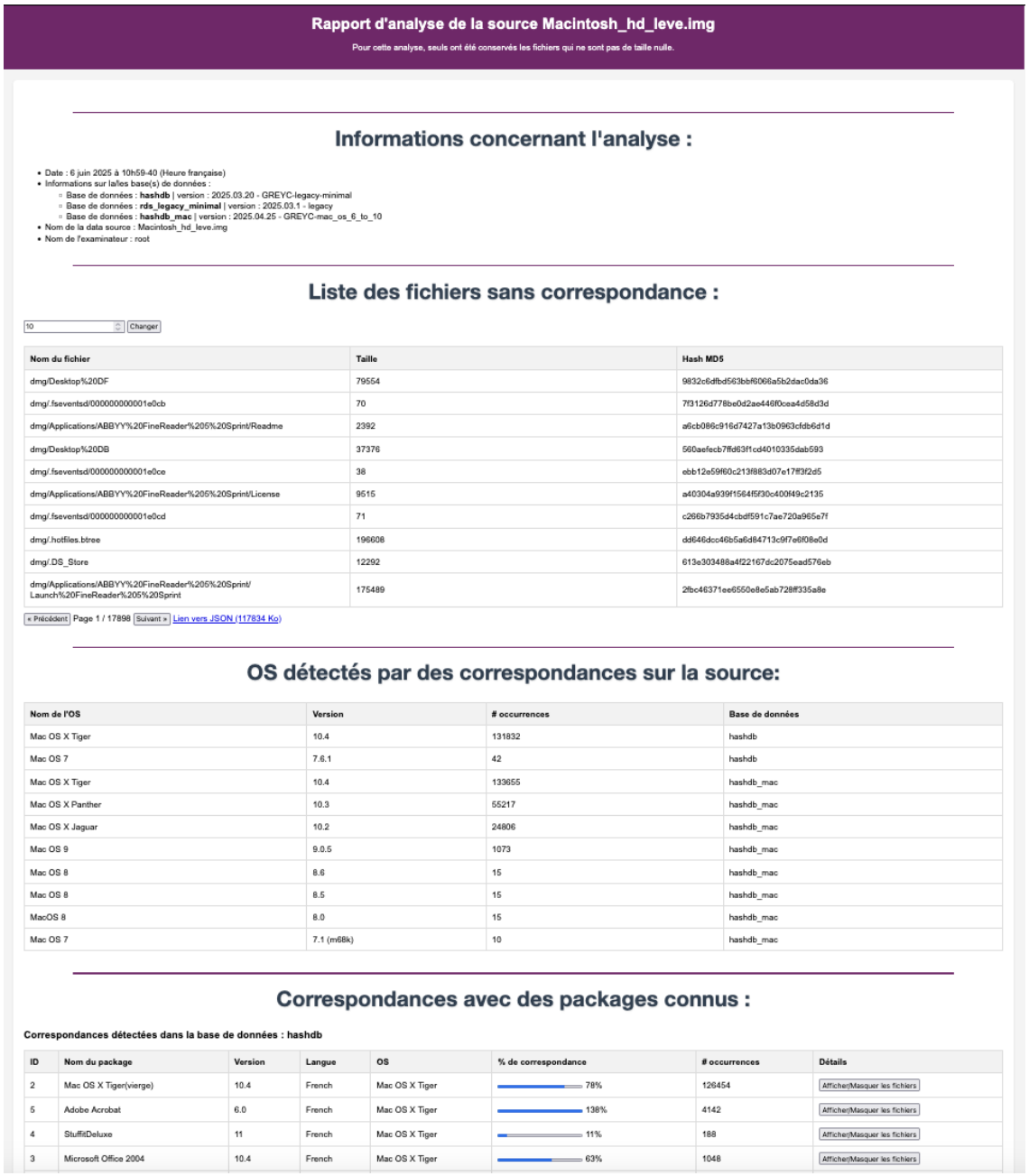
Cela étant fait, nous souhaitions permettre une sauvegarde et une transmission des résultats d'analyse en s'affranchissant d'Autopsy. J'ai donc pris l'initiative d'intégrer à ce plugin une logique de création de rapport. C'est quelque chose qui est prévu nativement lors du développement d'un plugin, mais qui reste cloisonné dans un format choisi par Autopsy. Souhaitant avoir plus de liberté dans le format de ce rapport afin qu'il s'adapte parfaitement aux besoins du projet, j'ai décidé de contourner l'option native pour gérer la création du rapport par moi-même.

Pour permettre une transmission facile et sans contrainte, nous avons choisi de créer un rapport en HTML. Cela permet à quiconque d'ouvrir le rapport dans un simple navigateur. Mais cela permet aussi de mettre en forme les résultats de manière à rendre le rapport agréable à consulter.

Durant la fonction *process*, nous pouvions récupérer énormément d'informations. Ainsi, l'ensemble était stocké directement dans un fichier JSON, qui était ensuite re-parsé afin de construire le HTML.

Ce rapport d'analyse se constitue de la manière suivante :

- **Informations concernant l'analyse :** listant la date de l'analyse, les bases de données utilisées, le nom de la source et le nom de l'examinateur (l'utilisateur ayant lancé le plugin).
- **Liste des fichiers sans correspondance :** cette partie constitue une liste de tous les fichiers n'ayant eu aucune correspondance avec les bases de données. Ainsi, l'archiviste peut rapidement voir les fichiers qui peuvent être intéressants à prioriser dans son analyse du fonds. Cette liste est disponible sous forme de pagination pour éviter de surcharger le rapport, et un lien vers le JSON est fourni. C'est en effet plutôt avec le JSON que l'archiviste devrait travailler pour exploiter cette liste.
- **OS détectés par des correspondances sur la source :** le rapport détaille ici les systèmes d'exploitation qu'il estime avoir trouvés sur la source. Cette partie doit cependant être bien analysée, afin de ne garder en tête que les OS avec un nombre d'occurrences élevé. La présence de seulement 20-30 occurrences n'est en fait pas représentative, mais elle est tout de même inscrite dans le rapport.
- **Correspondances avec des packages connus :** cette section reprend le même principe que pour les OS mais cette fois pour les applications. On affiche ainsi la liste des applications avec au moins une correspondance, en indiquant les détails de chaque application ainsi qu'un pourcentage de correspondance. C'est ce pourcentage qui permet à l'archiviste de savoir si les probabilités de présence de l'application sur la source sont élevées ou non. Pour finir, nous avons décidé d'ajouter un bouton permettant d'afficher la liste des noms des fichiers du fonds analysé ayant eu un match.



Rapport d'analyse de la source Macintosh_hd_leve.img

Pour cette analyse, seuls ont été conservés les fichiers qui ne sont pas de taille nulle.

Informations concernant l'analyse :

- Date : 6 juin 2025 à 10h59-40 (Heure française)
- Informations sur la/les base(s) de données :
  - Base de données : **hashdb** | version : 2025.03.20 - GREYC-legacy-minimal
  - Base de données : **rds_legacy_minimal** | version : 2025.03.1 - legacy
  - Base de données : **hashdb_mac** | version : 2025.04.25 - GREYC-mac_os_6_to_10
- Nom de la data source : Macintosh_hd_leve.img
- Nom de l'examinateur : root

Liste des fichiers sans correspondance :

10 Changer

Nom du fichier	Taille	Hash MD5
dmg/Desktop%20DF	79554	9832c6dfbd563bbf6066a5b2dac0da36
dmg/.fseventsd/000000000001e0cb	70	7f3126d778be0d2ae446f0cea4d58d3d
dmg/Applications/ABBYY%20FineReader%205%20Sprint/Readme	2392	a6cb086c916d7427a13b0963cfdb6d1d
dmg/Desktop%20DB	37376	560aefecb7ffd63f1cd4010335dab593
dmg/.fseventsd/000000000001e0ce	38	ebb12e59f60c213f883d07e17ff3f2d5
dmg/Applications/ABBYY%20FineReader%205%20Sprint/License	9515	a40304a939f1564f5f30c400f49c2135
dmg/.fseventsd/000000000001e0cd	71	c266b7935d4cbdf591c7ae720a965e7f
dmg/.hotfiles.btree	196608	dd646dcc46b5a6d84713c9f7e6f08e0d
dmg/.DS_Store	12292	613e303488a4f22167dc2075ead576eb
dmg/Applications/ABBYY%20FineReader%205%20Sprint/Launch%20FineReader%205%20Sprint	175489	2fbc46371ee6550e8e5ab728ff335a8e

« Précédent Page 1 / 17898 Suivant » Lien vers JSON (117834 Ko)

OS détectés par des correspondances sur la source:

Nom de l'OS	Version	# occurrences	Base de données
Mac OS X Tiger	10.4	131832	hashdb
Mac OS 7	7.6.1	42	hashdb
Mac OS X Tiger	10.4	133655	hashdb_mac
Mac OS X Panther	10.3	55217	hashdb_mac
Mac OS X Jaguar	10.2	24806	hashdb_mac
Mac OS 9	9.0.5	1073	hashdb_mac
Mac OS 8	8.6	15	hashdb_mac
Mac OS 8	8.5	15	hashdb_mac
MacOS 8	8.0	15	hashdb_mac
Mac OS 7	7.1 (m68k)	10	hashdb_mac

Correspondances avec des packages connus :

Correspondances détectées dans la base de données : hashdb

ID	Nom du package	Version	Langue	OS	% de correspondance	# occurrences	Détails
2	Mac OS X Tiger(vierge)	10.4	French	Mac OS X Tiger	78%	126454	Afficher/Masquer les fichiers
5	Adobe Acrobat	6.0	French	Mac OS X Tiger	138%	4142	Afficher/Masquer les fichiers
4	StuffitDeluxe	11	French	Mac OS X Tiger	11%	188	Afficher/Masquer les fichiers
3	Microsoft Office 2004	10.4	French	Mac OS X Tiger	63%	1048	Afficher/Masquer les fichiers

FIGURE 8 – Rapport HTML généré par le plugin

Après quelques tests de génération effectués, nous avons rencontré un problème majeur : le temps. Le plugin mettait en effet plusieurs dizaines de minutes à parcourir un seul pourcent de notre fond. Cela nous a évidemment éveillés sur le fait que nous devions avoir un point de ralentissement majeur, car les comparaisons MD5 ne sont logiquement pas très longues. En réalité, nous utilisions la base RDS, qui contient environ 160 millions d'entrées de fichiers. Et comme nous ne l'avions pas bien indexée, la recherche se faisait en temps linéaire. Suite à quelques recherches complémentaires, nous avons réussi à indexer nos différentes bases de données pour accélérer la recherche via MD5 et ainsi diminuer drastiquement le temps d'exécution de notre plugin. Pour ce faire, nous avons ajouté un index de type `btree` sur les colonnes contenant les empreintes MD5, permettant ainsi au serveur de réaliser des recherches beaucoup plus rapides. Cette optimisation a transformé des

requêtes qui prenaient auparavant plusieurs secondes en opérations quasi instantanées, rendant enfin ce plugin viable sur des corpus volumineux.

5.2 Application Python

Durant l'avancée du développement de ce plugin, j'ai à plusieurs reprises rencontré des problèmes techniques, qu'ils soient liés au développement en Java ou à l'utilisation d'Autopsy. En ajoutant à cela les différents questionnements autour d'une implémentation dans la plateforme GDIP, il est rapidement devenu nécessaire de traduire ce plugin. Cette migration vers Python permettrait ainsi de simplifier le développement, d'ajouter de nouvelles fonctionnalités, mais aussi de faciliter la transition en dehors d'Autopsy, vers une application propre au GREYC.

J'ai ainsi pu, en plus de donner a cette application les mêmes fonctionnalités que le plugin Autopsy et les mêmes sorties, pu ajouter les fonctionnalités suivantes :

- **Reprise d'exécution** : l'application peut reprendre le traitement là où il a été interrompu, évitant de relancer l'analyse complète sur de gros corpus en cas de coupure ou d'incident.
- **Base de données SQLite** : toutes les informations collectées lors de l'analyse sont stockées dans une base `analyse.sqlite`. Cette base contient non seulement les correspondances et métadonnées habituelles, mais également des informations sur le type des fichiers analysés. Pour ce faire, j'ai utilisé `fidentify`, un utilitaire très connu, mais aussi Magika AI, un modèle qui tente de détecter le type d'un fichier en analysant une partie de celui-ci. J'ai pu prendre le temps de comprendre son fonctionnement en lisant le papier publié par l'équipe de développement, dans lequel ils affichent des résultats très concluants, surpassant les autres méthodes comme `fidentify`. Dans notre cas pratique, les résultats sont plus mitigés : le modèle, n'étant pas entraîné sur les vieux fichiers, se trompe souvent et ne résout pas complètement les lacunes des autres utilitaires. Il faudrait donc penser a un ré-entrainement sur des fichiers anciens.
- **Liste des fichiers sans correspondance** : l'application génère également un fichier texte listant tous les fichiers n'ayant eu aucune correspondance, permettant à l'archiviste de prioriser l'analyse de ces éléments.
- **Visualisation interactive** : j'ai développé une petite application StreamLit qui se connecte directement à `analyse.sqlite`. Cette interface permet de visualiser différentes volumétries et statistiques sur les fichiers analysés, avec la possibilité de trier selon différents critères. Cette interface reste cependant perfectible et pourra être améliorée pour offrir plus de filtres et de visualisations avancées.

Cette application Python permet ainsi de disposer d'une solution complète et autonome, indépendante d'Autopsy, tout en offrant des capacités supplémentaires d'analyse, de reprise d'exécution et de visualisation des données. Elle constitue ainsi un outil supplémentaire, proposant des fonctionnalités différentes d'Autopsy. La partie volumétrie permet en effet une meilleure visualisation du fonds et peut donc être très utile.

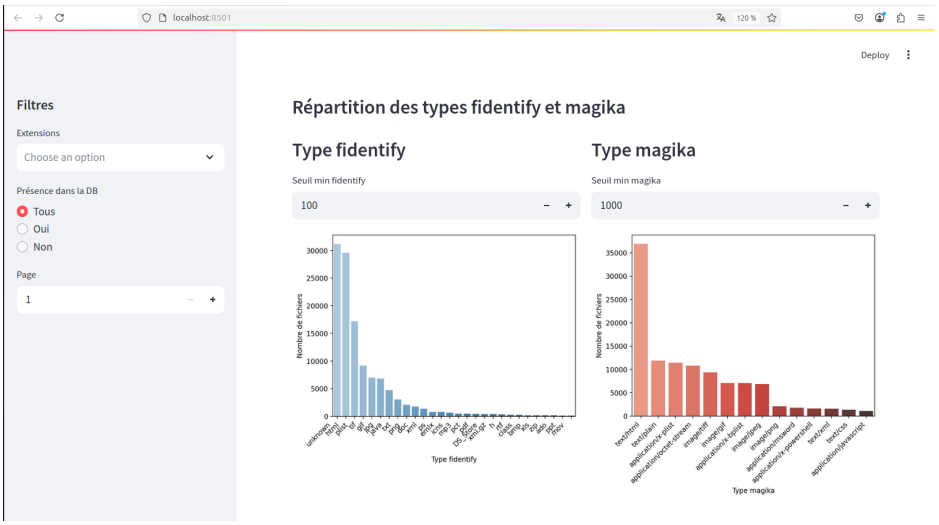


FIGURE 9 – Une partie de l'interface de visualisation interactive des résultats d'analyse

6 Autres travaux réalisés

Au cours de ce stage, j'ai eu l'opportunité de travailler sur de nombreuses thématiques liées à la problématique générale. Toutes ne peuvent pas être détaillées ici, mais je présente ci-dessous deux pistes qui m'ont paru particulièrement intéressantes, car elles illustrent bien la diversité des approches et des outils mobilisés.

6.1 QuarkXPress et reconstitution de fichiers

Au début du stage, nous avons eu accès à un dossier contenant plusieurs fichiers jugés « problématiques » par l'IMEC, car ils n'arrivaient pas à être ouverts correctement. Parmi eux figuraient des fichiers `.qxd` (documents QuarkXPress) et des fichiers `.rsrc` (ressources système Mac OS).

Les fichiers `.qxd` proviennent de QuarkXPress, un logiciel de publication assistée largement utilisé dans le domaine de l'édition pour la mise en page de livres et de documents avant impression. Le principal obstacle rencontré fut l'absence de version exploitable de QuarkXPress pour les anciens Mac OS, et le fait que les versions récentes soient payantes et sous licence restrictive. Après plusieurs recherches, j'ai pu identifier une ancienne version fonctionnelle pour Windows. L'ouverture des fichiers était possible, mais générait un message d'erreur indiquant des « polices manquantes ».

C'est à ce moment que les fichiers `.rsrc`, retrouvés dans un répertoire nommé « Polices1968 », ont pris tout leur sens. Contrairement aux documents classiques, ces fichiers ne sont pas destinés à être manipulés directement par l'utilisateur : ce sont des fichiers systèmes propres à Mac OS, habituellement invisibles. Leur structure s'est révélée particulière : les 82 premiers octets formaient une sorte d'entête (ou encapsulation), suivis du contenu brut de la police. Après avoir retiré cet entête, il a été possible de reconstruire les polices et de les intégrer dans QuarkXPress.

Grâce à cette démarche, les fichiers `.qxd` ont pu être ouverts dans leur contexte d'origine, avec les polices correctement restituées. Cette méthode a ensuite été documentée et transmise à l'IMEC, car elle pourrait constituer une piste de résolution pour d'autres fonds rencontrant des difficultés similaires.

6.2 Les fichiers FINDER.DAT

Lors de tests réalisés sur des disques de machines virtuelles exécutant Mac OS 7 à 9, j'ai observé la présence systématique de fichiers nommés *FINDER.DAT*. Ces fichiers, spécifiques à Mac OS, stockent des informations sur l'organisation des dossiers et des fichiers dans l'explorateur **Finder**, équivalent du gestionnaire de fichiers sous Windows ou Linux.

L'analyse hexadécimale de ces fichiers a montré qu'ils contenaient à la fois des noms de fichiers et des signatures. Des recherches complémentaires ont permis d'établir qu'ils utilisaient un mécanisme appelé **CREATOR/TYPE**. Ces fichiers sont constitués de blocs de 92 octets, chacun décrivant le nom d'un fichier, sa signature et les fichiers de ressources associés.

Afin d'exploiter ces informations, j'ai développé un script en Python capable de parcourir automatiquement tous les *FINDER.DAT* présents sur un disque et d'en extraire les couples CREATOR/TYPE. La liste obtenue a été confrontée à une documentation pour développeur ancienne de Mac OS, qui contient une liste (non exhaustive) des correspondances entre ces couples et les applications associées. Ainsi, une signature telle que `MSWD/W8BN` pouvait être reliée à un document édité dans Microsoft Word 8 (1998).

Ce travail permet, à partir d'un disque d'un fond d'archive, d'obtenir une liste (non exhaustive) des applications potentiellement installées ou utilisées. Cette approche permet de fournir rapidement des informations plus détaillées sur un fond à l'archiviste afin qu'il puisse, par exemple, comprendre l'environnement dans lequel travaillait l'auteur.

7 Résultats et validation

7.1 Volumétrie de la base

Après avoir rempli notre base d'empreintes avec les fichiers récupérés et calculés depuis les anciens systèmes Mac OS, nous avons pu analyser sa volumétrie. La base finale que nous avons créer contient environ 3 millions de fichiers, plus de 7 systèmes d'exploitation et une soixantaine d'applications différentes.

Chaque entrée de la base contient le hash MD5 du fichier, sa taille, son nom, l'application à laquelle il appartient, le système d'exploitation et le fabricant associé. Grâce à l'utilisation d'index `btree` sur les colonnes MD5, les recherches et comparaisons sont devenues extrêmement rapides, permettant de traiter efficacement des corpus volumineux.

7.2 Tests effectués sur les disques des fonds de l'IMEC

Pour valider l'utilité réelle de nos applications, nous avons décidé d'effectuer des tests sur les fonds de l'IMEC. Pour cela, nous avons choisi un disque d'environ 60Go, appartenant à un auteur contemporain. Ce disque contient, comme nous le savons, un système Mac OS 10.4 et environ 300 000 fichiers.

Nous l'avons donc testé à la fois dans Autopsy et avec notre application Python, afin de comparer les résultats.

Temps d'exécution	autopsy_hash_match	py_hash_match
Classique	≈ 22 min	≈ 50 min
Avec ajout de Tags	≈ 30 heures	
Sans 'fidentify'		≈ 25 min

FIGURE 10 – Temps d'exécution des deux méthodes

	autopsy_hash_match	py_hash_match
Nombre de fichiers	318 928 *	321 106 *
Nombre de fichiers **sans** correspondances	178 140	178 975
Nombre de fichiers **avec** correspondance	140 788	142 131
Pourcentage de fichiers « éliminés »	**≈ 44 %**	**≈ 44 %**

Différences liés a la non-résolution des **liens symboliques dans Autopsy*

FIGURE 11 – Nombre de fichiers traités en fonction des deux méthodes

Ces tableaux récapitulatifs de nos différentes expérimentations montrent que les deux méthodes sont similaires sur de nombreux points. Le temps d'exécution peut être comparable en désactivant certaines options. Quant au nombre de fichiers traités, malgré de petites différences liées à une particularité d'Autopsy dans la gestion des liens symboliques, les chiffres restent dans le même ordre de grandeur. Nous avons donc réussi à fournir deux solutions utilisant des technologies différentes et produisant certaines fonctionnalités non communes, tout en conservant des résultats bruts cohérents.

Nous pouvons également observer un point très important concernant l'efficacité de nos outils pour les archivistes. Sur un disque de plus de 300 000 fichiers, nous parvenons à en "éliminer" plus de 44%. Cela représente un gain de temps considérable pour l'archiviste, qui peut se concentrer en priorité sur un nombre réduit de fichiers, et ainsi rendre les fonds disponibles plus rapidement aux historiens ou autres chercheurs.

8 Conclusion et perspectives d'améliorations

Pour conclure ce rapport, ce stage m'a permis d'apprendre énormément et de me perfectionner dans plusieurs domaines importants pour la cybersécurité et la forensique : gérer et exploiter une base de données, virtualiser et émuler de vieux systèmes mais aussi développer des outils pour automatiser certaines analyses. J'ai aussi eu une vraie immersion dans le monde de la recherche, ce qui m'a permis de voir comment fonctionne un laboratoire et comment se déroulent les projets. Au-delà de ça, toutes les petites recherches et expérimentations m'ont donné une meilleure vue d'ensemble de l'informatique que je n'avais pas jusqu'alors.

Parmi ce que j'ai pu réaliser concrètement :

- La création et la mise en place d'une base de données d'empreintes pour regrouper des fichiers systèmes/génériques, afin de permettre un tri dans des fonds d'archive.
- Le développement d'outils pour incrémenter cette base de données efficacement.
- Le déploiement de deux moyens d'analyse à travers le plugin autopsy et sa traduction en python afin de donner toujours plus de moyen d'accélérer une analyse de fond.
- La mise en place de machines virtuelles et d'émulateurs pour tester différents Mac OS et récupérer les applications et fichiers systèmes.

Ce stage a été aussi un vrai apprentissage humain et professionnel. J'ai eu la chance de travailler sur un projet avec plusieurs stagiaires et des maîtres de stage très impliqués, toujours en quête de nouvelles recherches. Cela favorise grandement le travail en équipe et l'entraide, obligeant chacun à apprendre à se faire comprendre sur des sujets parfois techniques. Lors des différentes recherches, chaque réponse trouvée créait de nouvelles questions, ce qui attisait un peu plus notre curiosité collective. Cet environnement d'échanges permanents a été une réelle motivation tout au long du stage.

Perspectives d'amélioration

Plusieurs idées pourraient permettre d'aller plus loin :

- Créer une interface graphique pour l'application Python, afin de regrouper la gestion de la base d'empreintes avec le montage et l'analyse des images disques, un peu comme un Autopsy simplifié, tout en permettant d'y inclure la visualisation des volumétries.
- Étendre la virtualisation à d'autres systèmes d'exploitation anciens, qu'il s'agisse de systèmes Mac OS avant la version 6 ou de versions de Mac OS X plus récentes. Nous pourrions également imaginer ajouter des versions de Windows ou d'autres OS.
- Compléter l'article sur les différentes méthodes d'identification de types de fichiers publié par *Adrien Dubettier, Tanguy Gernot, Emanuel Giguet et Christophe Rosenberger*, en rejouant les tests sur les jeux de données, mais en y intégrant l'utilisation de MagikaAI.

En résumé, ce stage m'a permis d'apprendre autant sur le plan technique que sur le plan professionnel, tout en contribuant à un projet concret visant à aider à la préservation et à l'analyse des fonds de l'IMEC.

Bibliographie

- RDS (NIST) : `https://s3.amazonaws.com/rds.nsrl.nist.gov/RDS/RDSv3_Docs/RDSv3.pdf`
- rHash (GitHub) : `https://github.com/rhash/RHash`
- Documentation Autopsy : `https://sleuthkit.org/autopsy/docs/user-docs_fr/4.19.0/`
- Papier sur l'identification de fichiers : `https://hal.science/hal-04128864v1/document`
- Papier Magika AI : `https://arxiv.org/pdf/2409.13768`
- Ancienne documentation Apple pour développeurs : `https://vintageapple.org/macprogramming/pdf/The_Programmers_Apple_Mac_Sourcebook_1989.pdf`