GREP FORENSICS SUR IMAGE EWF

Projet 2A

Dubois Timothei
Mahier Awen
Porté-Mitatre Cydony
Janci Fabien
De Permentier Thibault

23 mars 2026

Encadrants : Emmanuel Giguet et Tanguy Gernot

Table des matières

1 Présentation du projet **2**
1.1 Cadre du projet 2

2 Découverte du sujet **3**
2.1 Description du format EWF (Expert Witness Format) 3
2.2 Découverte du langage GO 4
2.3 Bibliothèques préexistantes 4

3 Choix technologiques **4**
3.1 Création d'archive EWF 4
3.1.1 Comment créer une archive EWF 5
3.2 Architecture micro-modules 5
3.3 Implémentation naïve sans bibliothèques 5
3.4 Implémentation avec les bibliothèques GO-EXT4 et GO-EWF 6
3.5 Recherche de mots-clés et format de fichier 7
3.5.1 PDF 7
3.5.2 Zip 8
3.5.3 ODT et similaires 8

4 Résultat final **8**
4.1 CLI 8
4.2 Architecture micro-modules 9
4.3 Compatibilité inter-OS 10
4.4 Connexion inter-projet 11

5 Formats supportés **11**

6 Conclusion **11**

7 Annexes **13**

1 Présentation du projet

L'objectif de notre projet de 2A est de réussir à lire le contenu d'une archive EWF sans avoir besoin de monter l'image. Le format EWF [1] est un format d'image permettant d'assurer l'intégrité d'une image disque. Ce format est notamment prisé dans le cas d'enquêtes judiciaires afin de conserver une image non altérée d'un appareil informatique à un moment précis.

Bien que très avantageux afin de préserver l'intégrité d'une archive, une image disque EWF peut être très longue à monter et très compliquée à naviguer si le disque est très rempli. De plus, le fait que différents enquêteurs puissent avoir besoin de la même archive rend tout cela très vital, notamment pour des enquêtes où le temps est compté. C'est dans ce contexte précis que notre projet intervient : parcourir le fichier EWF sans le monter permet de gagner du temps. Nous avons aussi pour mission de fournir un outil de recherche de mots permettant de gagner encore plus de temps et notamment de savoir où chercher une fois le fichier EWF monté.

Du point de vue technique, il nous est demandé de fournir un code écrit en Go, langage très utilisé par la gendarmerie pour ses programmes, et de faire une architecture en micro-modules. Chaque module ne remplissant qu'une tâche simple et unique et étant indépendant des autres modules autant que possible.

1.1 Cadre du projet

Nous en avons aussi profité pour délimiter le cadre de notre projet en plus des contraintes techniques et contextuelles déjà imposées par notre client. Le sujet étant très large et complet et notre temps limité, nous avons ainsi défini le cadre suivant :

- Concentration sur un unique type de partition, EXT4.
- Recherche dans une archive EWF non chiffrée et non compressée.
- Recherche dans un unique fichier d'archive (EO1 uniquement).
- Recherche dans le contenu d'un fichier non chiffré.
- Recherche dans un nombre fixe de types de fichiers, basé sur leurs extensions.

Cela nous permet ainsi d'avoir un environnement de travail fixe dans lequel évoluer tout en répondant aux demandes de notre client.

1. Expert Witness Disk Image Format

2 Découverte du sujet

2.1 Description du format EWF (Expert Witness Format)

Le format EWF (Expert Witness Format) est un format d'image disque largement utilisé en investigation numérique dans le cadre de procédures judiciaires et d'analyses forensiques. Il a été conçu pour permettre la capture, le stockage et la conservation de copies fidèles de supports numériques tout en garantissant l'intégrité des données.

Concrètement, le format EWF agit comme un conteneur capable de stocker une image disque complète, réalisée sous forme de copie bit-à-bit. Cette approche assure que l'ensemble des données du support original — y compris les espaces non alloués et les données supprimées — est préservé sans altération.

Une des caractéristiques majeures du format EWF est sa structure segmentée. L'image disque est divisée en plusieurs fichiers de taille contrôlée (généralement avec des extensions telles que .E01, .E02, etc.), ce qui facilite son stockage, son transfert et sa manipulation, notamment sur des systèmes ayant des limitations de taille de fichiers.

Le format intègre également des mécanismes de compression permettant de réduire l'espace de stockage nécessaire, tout en maintenant l'exactitude des données. Par ailleurs, il inclut des fonctions de vérification d'intégrité basées sur des empreintes numériques (hash), telles que MD5 ou SHA1. Ces empreintes sont calculées lors de l'acquisition et peuvent être vérifiées ultérieurement afin de garantir que l'image n'a subi aucune modification. En complément, le format EWF permet l'enregistrement de métadonnées

détaillées relatives au processus d'acquisition. Ces informations peuvent inclure la date et l'heure de la capture, l'outil utilisé, ainsi que l'identité de l'enquêteur, contribuant ainsi à la traçabilité et à la valeur probante des éléments collectés.

Enfin, le format offre des options de chiffrement afin de sécuriser les données sensibles, notamment lors de leur stockage ou de leur transfert.

Grâce à l'ensemble de ces functionnalités, le format EWF constitue un standard en informatique légale pour la conservation fiable et sécurisée des preuves numériques.

2.2 Découverte du langage GO

Certains ont appris le langage Go grâce au site go.dev et à son tutoriel, pendant que d'autres ont directement décidé d'expérimenter le langage avec une ébauche de code explorant l'archive EWF au niveau binaire.

Cela nous a permis à la fois d'avoir une partie du groupe qui comprend en profondeur le langage Go et d'avoir une autre partie du groupe qui, elle, a déjà une idée des solutions et méthodes possibles à utiliser pour parvenir à nos fins.

2.3 Bibliothèques préexistantes

Nos recherches nous ont conduites à différentes bibliothèques écrites en Go et qui, correctement utilisées, nous ont permis de simplifier et d'accélérer notre projet. Ces bibliothèques sont :

- go-ewf [2]
- go-ext4 [3]

La première bibliothèque permet d'accéder au contenu d'une archive EWF plus simplement que via la lecture binaire directement. Elle nous permet notamment de faire abstraction de l'enveloppe EWF et de travailler directement sur la partition du disque.

La seconde bibliothèque, elle, permet de travailler directement sur le fichier disque si ce dernier est au format EXT4. Cela nous permet notamment d'accéder plus simplement aux fichiers et à leur contenu.

3 Choix technologiques

3.1 Création d'archive EWF

Afin de pouvoir utiliser et tester notre programme, nous avons besoin d'archives EWF non compressées, non chiffrées, en un unique fichier et utilisant le système de partition EXT4. La solution la plus simple que nous avons trouvée est de créer nous-mêmes des archives EWF contenant divers fichiers avec une multitude d'extensions différentes. Cela nous permet ainsi d'être sûrs de ce que nous cherchons et de vérifier l'exactitude de nos fonctions de recherche.

2. `https://github.com/sydp/goewf`
3. `https://github.com/dsoprea/go-ext4`

3.1.1 Comment créer une archive EWF

La création d'une image disque au format EWF sous Linux s'effectue à l'aide de l'outil `ewfacquire`, issu de la suite `ewf-tools`. Cette opération permet de réaliser une copie bit-à-bit d'un support de stockage tout en encapsulant les données dans un conteneur EWF.

La commande suivante permet de lancer l'acquisition :

```
sudo ewfacquire /dev/sdX
```

où `/dev/sdX` correspond au périphérique source à imager.

Il est également possible d'automatiser l'acquisition en spécifiant directement les paramètres principaux :

```
sudo ewfacquire -u -C "Nom_du_cas" -E "Examinateur" \
-d "Description" -t image_evidence /dev/sdX
```

Cette commande génère une image EWF segmentée (fichiers `.E01`, `.E02`, etc.), compressée et accompagnée de métadonnées et d'empreintes de vérification.

Le résultat obtenu constitue une archive EWF exploitable aussi bien dans un cadre légale ou juridique que dans le cadre de notre projet

3.2 Architecture micro-modules

Comme demandé par notre client, notre programme respecte une architecture en micro-modules. Pour ce faire, nous avons décidé de créer un module permettant de lire le fichier EWF et un module de recherche de mots-clés afin de nous répartir le travail. Ces modules seront agrémentés au fur et à mesure de notre projet par d'autres afin de réaliser toutes nos tâches.

3.3 Implémentation naïve sans bibliothèques

Par méconnaissance de l'existence de différentes bibliothèques utilisables en Go, nous avons exploré les différentes possibilités que pouvait nous offrir Go natif.

Nous avons donc écris un programme qui parcourt simplement le binaire de notre archive EWF et qui nous en renvoie les données. Bien que

fonctionnel, cela est très fastidieux car nous avons eu besoin de nous baser sur différents schémas disponibles en ligne [Lau] afin de parcourir correctement notre fichier.

Ce parcours simple nous a permis de nous rendre compte de l'organisation complète du format EWF et d'en lire l'intégralité. Nous avons réussi à découvrir où se trouve la partie intéressante de notre image disque dans ce format contenant énormément d'informations, que ce soit dans la section "header" ou dans les nombreux autres types de sections présentes. Cette découverte du format EWF a été possible grâce à la lecture de la documentation disponible sur le dépôt Git de EWF-GO, sans nous rendre compte du potentiel de la librairie, afin de parcourir les données de l'entête EWF et des différentes parties enregistrées.

Nous avons donc traité la section "sector" contenant le EXT4 de notre disque, soit l'intégralité des fichiers. Pour ce faire, de la même façon que pour le format EWF, nous nous sommes énormément documentés sur le EXT4.

Bien que disposant du format précis de stockage d'une partition EXT4 grâce à ce schéma 1 [Ora], il fût compliqué de la parcourir.

En effet, le principe de ce format de fichier est qu'après une série d'espaces réservés respectivement pour la section de boot, le superbloc et un espace libre quelconque, on arrive sur une liste de Group Descriptors, qui pointent vers d'autres adresses mémoire contenant différentes bitmaps d'Inodes.

Le problème de ce format en Inode est qu'il n'y a pas de détail sur la manière de le naviguer proprement. Ici devient nécessaire l'utilisation des bibliothèques GO-EXT4 et GO-EWF.

3.4 Implémentation avec les bibliothèques GO-EXT4 et GO-EWF

Nous nous sommes attelés à réécrire notre code en utilisant la bibliothèque GO-EWF. Cela est assez simple, la bibliothèque étant très documentée et avec beaucoup d'informations génériques sur EWF, nous avons assez facilement pu créer un programme qui, pour un fichier EWF passé en entrée, nous renvoie un pointeur sur la partition disque de l'archive.

L'accès à la partition n'étant plus un problème, il reste à lire la par-

tition EXT4. La bibliothèque EXT4 étant bien moins fournie que la bibliothèque EWF, nous avons eu un peu plus de mal à comprendre comment fonctionne le système d'Inodes de EXT4.

Après avoir surmonté ces problèmes de compréhension, nous avons réussi à parcourir notre partition via les Inodes. Ayant accès à l'arborescence du disque, nous passons à l'étape de recherche.

3.5 Recherche de mots-clés et format de fichier

Ayant accès à l'intégralité des fichiers et dossiers de notre partition, nous nous attelons donc à la recherche de mots-clés dans notre disque.

Nous avons cherché des mots-clés dans le nom des fichiers et dossiers ainsi que dans des fichiers simples. Nous pouvons notamment citer les .txt, .html ou .xml qui sont des fichiers où le texte est disponible en clair, ce qui permet de ne pas avoir besoin de bibliothèques extérieures pour en lire le contenu.

Pour cela, nous utilisons un ReadSeeker, structure de données permettant de parcourir le binaire et Avec cela, nous avons pu récupérer le contenu du fichier. Après avoir normalisé le contenu du fichier (retrait des majuscules et des accents), nous avons pu enfin compter le nombre d'occurrences de nos mots dans nos fichiers. En plus du nombre d'occurrences, nous récupérons aussi la ligne de l'occurrence. Cette ligne correspond soit à un format utilisé par une bibliothèque (comme pour le PDF) ou bien simplement à son emplacement grâce au nombre de retours à la ligne.

Une fois que nous sommes en capacité de chercher un mot dans un texte non compressé, il reste encore à extraire des autres fichiers compressés les informations afin de pouvoir les traiter.

3.5.1 PDF

Pour parcourir les fichiers PDF, nous avons utilisé la bibliothèque pdf [4] que nous avons trouvée sur le site de Go. Cette bibliothèque permet de simplifier énormément le parcours : le PDF est découpé en pages, elles-mêmes découpées en lignes similaires à la représentation humaine du fichier. Nous n'avons plus qu'à faire une simple comparaison pour trouver nos différents mots voulus.

4. `https://pkg.go.dev/github.com/dslipak/pdf`

3.5.2 Zip

Le cas des fichiers Zip "normaux" est un peu plus complexe dans le sens où nous parcourons notre arborescence de partition via la bibliothèque EXT4 et donc via les inodes. Un fichier ZIP ouvert via la bibliothèque standard de Go se parcourt lui via un simple reader, ce qui nous a obligés à nous adapter. Mais une fois notre répartiteur de fichier créé pour le zip, son contenu est traité comme le reste des fichiers et dossiers.

3.5.3 ODT et similaires

Dans le cas des fichiers ODT et autres formats similaires, notre fichier est en réalité un simple fichier ZIP contenant différentes données et métadonnées. Nous devons donc simplement ouvrir le fichier ZIP et accéder directement au fichier contenant le texte de notre fichier. Ce contenu est ensuite analysé via notre outil de traitement habituel.

4 Résultat final

4.1 CLI

Afin de permettre à des utilisateurs divers d'utiliser notre programme, nous avons décidé de créer une CLI (Command Line Interface, que nous avons nommée ici "searchEWF") avec diverses commandes afin de pouvoir utiliser les différentes fonctions de notre outil.

- search <chemin img ewf> <-w 'liste de mots' | -t '.extension'> [-v]
- infoEWF <chemin img ewf>
- les commandes par défaut (help)

La première permet d'effectuer une recherche dans une archive EWF précise en donnant soit des mots-clés séparés par un espace ("apple orange raspberry"), soit des extensions de fichier (".pdf .txt"), soit les deux.

La recherche de mots-clés est menée à la fois dans le nom et le contenu des fichiers. Par défaut seront affichés les fichiers trouvés et le nombre d'occurrences par mots recherchés à l'intérieur.

Avec -v (pour verbose) sont affichés l'emplacement dans chaque fichier des occurrences des mots-clés (ligne et nombre d'occurrences sur cette dernière).

Dans tous les cas, les résultats de la recherche sont enregistrés dans un fichier "output.txt" qui se réinitialise à chaque nouvelle recherche : il est donc important de renommer ou déplacer manuellement ce fichier si l'utilisateur souhaite en conserver le contenu.

Concernant la seconde commande, il s'agit d'afficher des informations générales sur l'archive (type, nombre de sections, etc.).

Pour chacune des commandes, des erreurs spécifiques ont été créées pour donner des indications personnalisées à l'utilisateur et ainsi le guider au mieux dans la manière d'utiliser notre solution.

Grâce à ces différentes commandes, un utilisateur peut donc utiliser notre outil et chercher le mot de son choix dans une archive EWF sans avoir besoin de la monter.

4.2 Architecture micro-modules

Il nous est demandé dans notre sujet de projet de réaliser une architecture en micro-modules. L'avantage de ce genre d'architecture est qu'un projet ayant besoin d'une unique partie du nôtre pourrai n'importer que le module de lecture de partition EXT4 et ainsi ne pas avoir besoin d'importer le projet entier.

Voici la liste de nos modules :

- ReadPDF
- ReadName
- ReadFile
- ReadEWF
- ReadCompressedFile
- ReadUncompressedFile

Bien que nous chaque fonctionnalité soit dans d'unique module, certains restent tout de même liés entre eux. Le module ReadFile correspond en réalité à notre routeur. C'est lui qui lie l'extension d'un fichier à la fonction à appeler pour le traiter. Ainsi, il est de ce fait lié aux autres modules de lecture de fichier. Chaque fichier contenant des fonctions de lecture de fichier vient se connecter au routeur et l'informe des extensions qu'il est capable de traiter. Cela veut donc dire que chaque module de lecture de fichier doit

connaître le routeur et que le fichier principal doit connaître tous les modules de lecture de fichier.

Nous avons aussi le module ReadCompressedFile qui a besoin de ReadUncompressedFile. Cela nous permet notamment de normaliser nos sorties afin de garantir que tous les fichiers ont une sortie similaire.

Voici un exemple de la manière dont nos modules interagissent entre eux : Le module ReadEWF va lire un fichier EWF et exploiter le secteur EXT4 dessus afin de récupérer la liste des fichiers et dossiers. Il va ensuite appeler le module ReadFile pour traiter les fichiers et le module ReadName pour lire les noms de dossiers. Ce dernier va rediriger les fichiers vers leur module de traitement (ReadUncompressedFile, ReadCompressedFile ou encore ReadPDF).

Dans l'ensemble, nous sommes très contents de notre architecture en micro-modules. Nous avons découplé au maximum nos modules les uns des autres afin qu'ils soient réutilisables individuellement. Pour ceux qui ne le sont pas, c'est soit de par leur nature (le module de routage, qui a besoin de connaître toutes les autres fonctions), soit par souci de simplicité et de non-redondance de code (le module de traitement des fichiers compressés pourrait contenir le code normalisé pour la sortie, mais il est bien plus logique de simplement importer la fonction afin de ne pas faire de duplicata de code).

De plus, nous avons aussi développé nos modules dans un esprit de réutilisabilité de façon à ce qu'ils puissent vraiment être utilisés dans un autre cadre. Ainsi, nos fonctions de recherche prennent en paramètre la liste de mots à chercher, le fichier entier sous forme de tableau de bits et si la fonction doit être bavarde ou non.

4.3 Compatibilité inter-OS

Une des contraintes que nous avions lors de la conception de notre projet était de faire en sorte que notre application soit utilisable sous Linux et sous Windows. En effet, dans le cas d'une utilisation par des forces de l'ordre, le logiciel pourrait avoir besoin de fonctionner sur différents systèmes. Malheureusement, nous n'avons pas pu tester notre projet sur une machine Windows, les étapes pour installer les bibliothèques étant compliquées. Néanmoins, le projet étant développé entièrement en Go, la compatibilité inter-OS est censée être assurée.

4.4 Connexion inter-projet

Pendant que nous travaillons sur un outil de recherche de mots-clés dans une archive EWF, un second groupe travaille sur un projet bien différent mais sur une base commune à la nôtre : une commande find pour une archive EWF. Le but de leur projet étant de convertir une fonction simple dans de nombreux environnements EWF (différents modes de compression, de type de partition, etc.).

Il nous a été proposé d'essayer de lier nos deux projets ensemble. Comme nos fonctions de recherche et de routage sont indépendantes du format de disque, nous sommes en théorie prêts afin de connecter les deux projets ensemble. Cependant, suite à un problème de temps, cela ne s'est pas fait, mais la connexion reste toutefois possible modulo quelques petites erreurs d'intégration.

5 Formats supportés

Voici la liste des différentes extensions de fichiers que notre programme est capable de parcourir dans son état actuel :

- .txt
- .xml
- .odt
- .ots
- .pdf
- .docx
- .html (Pas d'interprétations des entités)
- .zip

Nous avons fait le choix de nous limiter à ces extensions car ce sont pour nous les extensions les plus présentes dans différents systèmes. Dans le cas où l'on souhaiterait ajouter d'autres extensions, il n'y aurait qu'à modifier le module ReadFile afin de traiter cette nouvelle extension de fichier.

6 Conclusion

En conclusion, nous sommes très heureux d'avoir pu travailler sur ce projet ! Que ce soit le sujet en lui-même, le fait de devoir développer dans

un nouveau langage, le Go, ou bien le fait d'appliquer une architecture en micro-modules, nous avons beaucoup appris en réalisant ce projet.

Nous sommes aussi contents du travail que nous avons fourni : notre projet répond en grande partie aux attentes de notre client ainsi qu'aux nôtres, et nous espérons que d'autres personnes en prendront la suite dans le futur.

Dans cette optique, nous aimerions ainsi proposer une liste de pistes d'améliorations auxquelles nous avons pensé mais que nous n'avons pas pu réaliser faute de temps :

- Une interface graphique complète
- Possibilité de chercher dans d'autres types de partition
- Assurer la compatibilité avec Windows
- Ajouter de nouveaux formats d'extension

7 Annexes

Table des figures

1 Format d'une partition EXT4 14

Références

[Dso] Dsoprea. Page github de go-ext4. https://github.com/dsoprea/go-ext4.

[Lau] Clévy Laurent. Page d'informations sur le format ewf. https://connect.ed-diamond.com/misc/misc-117/description-du-format-de-stockage-forensique-encase-ewf.

[Lib] Libyal. Page github de lib-ewf. https://github.com/libyal/libewf.

[Ora] Oracle. Page source du schéma du ext-4. https://blogs.oracle.com/linux/understanding-ext4-disk-layout-part-1.

[Syd] Sydp. Page github de goewf. https://github.com/sydp/goewf.

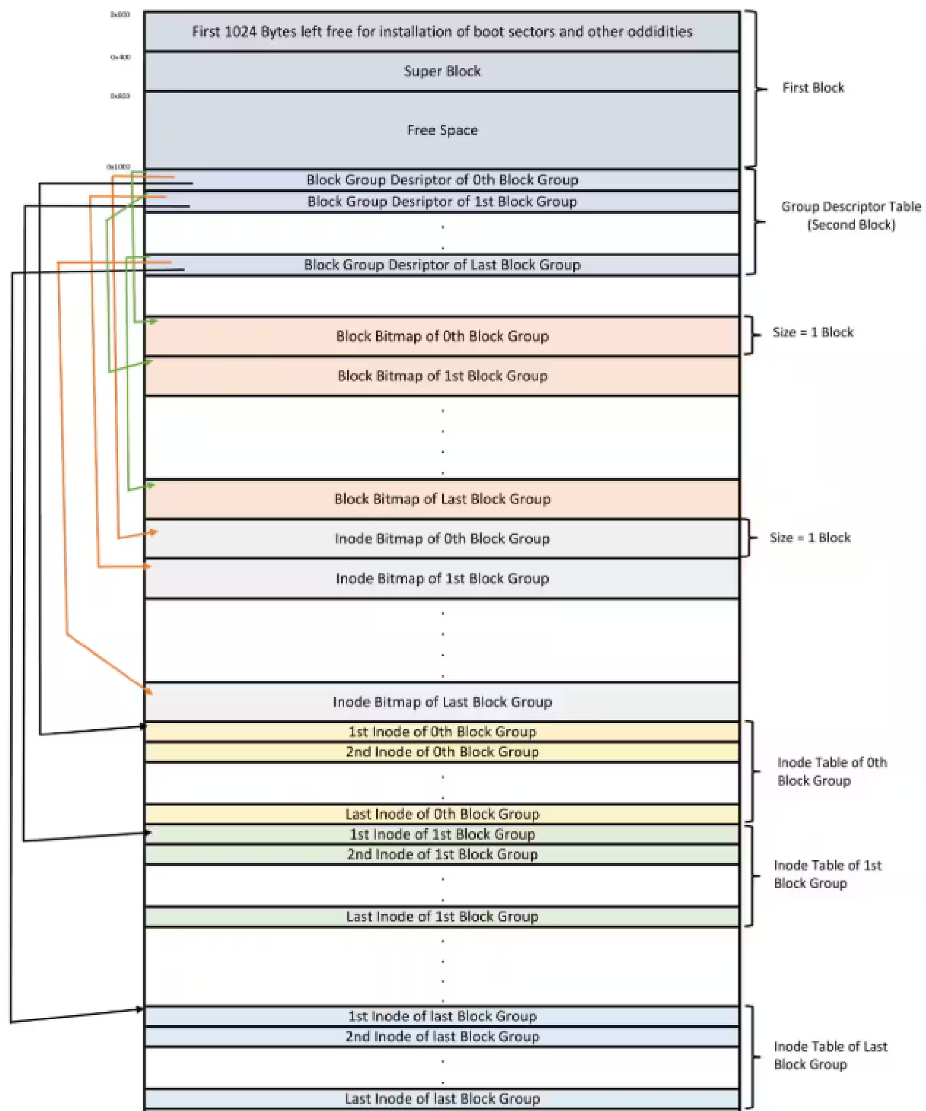

Figure 1 – Format d'une partition EXT4