Ecole Publique d'ingénieures et d'ingénieurs en 3 ans

Rapport

IMPLEMENTATION D'UN "FIND FORENSIC" EN GO POUR IMAGES EWF

le 25 mars 2026,
version 1.2

Lino LANDRY,
Chloë BARTOLONE,
Florestan TRILLOT,
Jean BOUDINSKI,
Isaac ABOYO,
Rania MEDIANE

Tuteurs : Emmanuel GIGUET,
Tanguy GERNOT

www.ensicaen.fr

TABLE DES MATIERES

INTRODUCTION

1. Contexte du projet

Aujourd'hui, l'informatique est présente dans la majorité des enquêtes criminelles. A titre d'exemple, un rapport du Sénat français souligne que les preuves numériques et les données de connexion apparaissent désormais dans 85 % des dossiers judiciaires.

Lorsqu'un ordinateur ou un disque dur est saisi, les enquêteurs s'abstiennent de l'allumer directement car cela risquerait d'altérer les données et donc de détruire des preuves.

Ainsi, pour figer et protéger la scène de crime numérique, les experts réalisent une copie exacte du disque dur. Cette copie est souvent enregistrée dans un format standardisé et sécurisé appelé EWF (Expert Witness Format). Une fois cette archive créée, les enquêteurs ont besoin de logiciels spécifiques pour en explorer le contenu et trouver des indices, sans jamais altérer les données d'origine.

Cependant, une contrainte de temps se pose sur le terrain. Les supports saisis peuvent contenir des quantités importantes de données. Extraire l'intégralité d'une archive EWF pour pouvoir l'examiner est donc une opération qui peut se révéler chronophage. Ce contexte demande alors de pouvoir inspecter le contenu de ces images disques directement, sans passer par une étape d'extraction.

2. Présentation du projet

Ce projet a été réalisé sous la supervision de M. Tanguy Gernot et M. Emmanuel Giguet, chercheurs au GREYC dans l'équipe cybersécurité SAFE, spécialisée dans la recherche en sécurité informatique.

Ce projet consiste à développer un nouvel outil de recherche de fichiers spécialement conçu pour l'investigation numérique. Il s'inspire d'une commande emblématique du système Linux : find.

L'objectif de notre travail est de recréer cette fonctionnalité de recherche, capable de lire directement à l'intérieur des archives EWF sans en extraire le

contenu. L'utilisation du langage de programmation Go est une contrainte technique imposée. Ce choix permet de créer un outil rapide capable de fonctionner aussi bien sous Windows que sous Linux. C'est, par ailleurs, un langage très répandu dans le domaine de l'investigation numérique.

3. Objectifs du projet

Notre mission principale est de réaliser un outil d'exploration fiable et complet. Pour cela, le logiciel doit remplir les objectifs suivants :

- Explorer le contenu : Parcourir automatiquement l'ensemble des dossiers et sous-dossiers contenus dans la copie du disque dur.

- Filtrer : Permettre à l'utilisateur de chercher des fichiers selon des critères (par le nom, par la taille, ou par les dates de création et de modification).

4. Enjeux pratiques

Cet outil est pensé pour répondre aux besoins des forces de l'ordre. Il permettra de faire gagner un temps précieux aux enquêteurs. Concrètement, le logiciel devra par exemple servir à :

- Retrouver rapidement des documents cachés au milieu de milliers d'autres fichiers.
    - Les enquêteurs pourront cibler des éléments portant des noms évocateurs, comme "mots_de_passe.txt" ...
- Isoler tous les fichiers qui ont été modifiés juste avant la saisie du matériel.
    - Pour détecter une tentative de suppression de preuves récentes, comme le nettoyage d'un dossier de téléchargements.
- Repérer les dossiers qui ont été volontairement masqués ou dissimulés par un utilisateur.
    - Mettre en évidence des répertoires rendus invisibles par défaut (dossiers commençant par un point sur Linux) ou des dossiers suspects cachés au fond de l'arborescence du disque.

ANALYSE ET ETAT DE L'ART

5. Spécificités techniques du format EWF

Puisque l'objectif est d'interroger directement l'archive sans en extraire le contenu, il est nécessaire de comprendre comment celle-ci est construite. Le format EWF découpe la copie du disque en multiples blocs compressés et intègre des algorithmes de hachage pour garantir que la preuve n'a pas été corrompue. C'est cette architecture complexe qui entrave l'utilisation de commandes de recherche standards et impose un développement spécifique.

5.1. Défi technique de l'organisation des données

Notre outil doit être capable de s'orienter sur le disque et de comprendre comment les informations y sont rangées. Cela implique de pouvoir gérer deux niveaux d'organisation :

- Les différentes tables de partition (MBR, GPT) : Un disque physique est souvent découpé en plusieurs partitions. L'outil doit savoir lire correctement les différents formats de découpage (MBR pour les anciens, GPT pour les récents) afin de localiser précisément le point de départ de chaque partition.

- Les différents formats de fichiers (NTFS, Ext4) : A l'intérieur de chaque partition, les fichiers sont référencés selon des règles strictes. Windows utilise par exemple le format NTFS, tandis que Linux utilise Ext4. L'outil doit pouvoir gérer les différents formats pour en extraire l'arborescence, retrouver les chemins de fichiers et lire leurs métadonnées (dates, tailles).

6. Limites des outils actuels

Il existe déjà des solutions utilisées en investigation numérique, mais elles ne répondent pas parfaitement au besoin d'une recherche rapide en ligne de commande ciblée :

- Extraction de l'image : Comme expliqué précédemment, le fait d'extraire l'image pour utiliser une commande classique « find » est une étape

coûteuse en temps et qui nécessite un espace de stockage trop important.

- Les logiciels d'investigation (ex : Autopsy) : Il existe des outils professionnels avec des interfaces graphiques très complètes. Toutefois, ils sont significativement lourds. Ils imposent généralement une phase de traitement de l'image avant de pouvoir lancer la moindre recherche.

7. Utilisation de bibliothèques externes

Face à la complexité du format EWF et de la diversité des systèmes de fichier, redévelopper entièrement les algorithmes de lecture serait trop complexe. Nous avons donc fait le choix de nous appuyer sur des bibliothèques externes :

- EWF : Cette bibliothèque gère l'ouverture d'archives EWF et leur lecture. Elle fournit ensuite les fonctions permettant de se déplacer dans l'image disque contenue dans l'archive et d'en lire le contenu.

- TSK (The Sleuth Kit) : C'est un ensemble de fonctions d'analyse de systèmes de fichiers. Son implémentation permet de déléguer la lecture des tables de partition comme MBR ou GPT et des partitions comme NTFS ou Ext4, assurant un parcours de l'arborescence sans avoir besoin de monter l'image sur le système d'exploitation de l'enquêteur.

ARCHITECTURE ET CONCEPTION

8. Architecture globale

8.1. Interface TSK – EWF

Le défi majeur et structurant a été l'interfaçage entre les bibliothèques TSK et EWF. En effet, la librairie EWF permet d'obtenir des pointeurs et fournit des fonctions de lecture, tandis que la librairie EWF permet une ouverture à partir d'un fichier. Une nouvelle structure étendue d'image disque TSK a été définie en C pour contenir également les données EWF.

Néanmoins, la librairie TSK permet une ouverture externe en fournissant en argument les fonctions read et seek. C'est ce point d'entrée que nous avons utilisé en ouvrant d'abord l'archive EWF puis en utilisant des pointeurs vers des fonctions que nous avons définies avec les bonnes signatures pour redéfinir les méthodes de lecture. La librairie TSK passe alors par la fonction de lecture fournie par la librairie EWF au lieu d'utiliser les méthodes classiques fournies par le système d'exploitation sur les fichiers.

8.2. Extension du module Go-TSK existant

Aucune méthode d'ouverture à partir d'une source autre qu'un fichier par son chemin n'ayant été fournie dans le projet Go-TSK, notre travail a nécessité une extension du module en Go afin de permettre la réécriture des fonctions read et seek. Nous avons exploité la compilation de code C en langage Go.

Cette extension nous permet d'accéder à toutes les méthodes et structures existant dans la librairie EWF en C qui ne sont pas exposées dans l'état actuel du module Go TSK.

8.3. Création d'archives EWF

Afin de réaliser les tests de notre programme, il a fallu nous procurer des archives EWF. Celles-ci peuvent se trouver sur des sites spécialisés en investigation numérique. Une autre méthode consiste à créer nos propres archives EWF à partir d'images disques que nous avons générées nous-mêmes. Souhaitant en maîtriser le contenu, nous avons choisi cette seconde voie.

La première étape consiste à créer une image disque, ce qui peut s'apparenter à une clé USB virtuelle dont le contenu est stocké dans un fichier. Pour ce faire, nous utilisons l'utilitaire de gestion de disque inclus par défaut dans nos distributions Linux. Nous maîtrisons alors le système de partitionnement et les partitions inclues dedans. L'inconvénient de cette image disque réside dans le fait que sa taille est indépendante du contenu : elle est fixée par la taille maximale que l'on a décidée.

Il faut ensuite encapsuler cette image disque dans un fichier EWF, qui contient l'image disque et les informations de l'enquête telles que le numéro d'enquête ou le nom de l'enquêteur. L'outil `ewfacquire` en ligne de commande permet de créer cette archive en spécifiant l'image disque source et en incluant toutes les informations de l'enquête. L'outil permet également une compression, ce qui réduit très fortement la taille de l'image disque.

9. Parcours de l'arborescence de fichiers

9.1. Extraction des métadonnées

La structure de gestion des fichiers fournie par la librairie TSK en C donne accès à de nombreuses métadonnées telles que le nom, la taille, les dates de création, de modification ou d'accès des fichiers. Ces métadonnées permettent d'affiner la recherche par des filtres.

Ces métadonnées n'étant pas définies dans la structure en Go, nous avons à nouveau dû étendre le module Go-TSK en interfaçant la structure existante en C. Nous remarquons alors que certaines définitions changent d'un système à l'autre, comme les dates de création entre un système Windows et un système Unix.

9.2. Conception du moteur de filtrage

Le premier filtre à appliquer sur les fichiers s'appuie sur le nom de ceux-ci. Il est possible de sélectionner les fichiers lorsqu'ils contiennent une sous-chaîne précisée, mais la recherche peut également s'effectuer sous la forme d'expressions régulières (regex) qui permettent un filtrage efficace sur les extensions de fichiers ou des motifs de nommage par exemple.

9.3. Elimination des dossiers inutiles

Certains dossiers sont créés et contiennent des données qui ne se révèlent que rarement utiles. Nous avons alors énuméré des dossiers ou fichiers dans lesquels le

parcours de l'arborescence serait considéré superflu. Nous trouvons par exemple le dossier « Lost+Found » dans les partitions Ext4, le système de fichier le plus couramment utilisé sur Linux, qui permet la réparation du système de fichiers, ou encore les fichiers hyberfil.sys et swapfile.sys sur Windows qui correspondent à du contenu de mémoire vive (respectivement pour la mise en veille prolongée et le swap).

10. Interface utilisateur

10.1. Module Finder

Afin de faciliter les actions d'ouverture et de recherche, un unique point d'entrée a été implémenté dans un module « Finder ». Celui-ci prend en charge l'ouverture de l'archive par son chemin, puis permet par l'intermédiaire de setters de configurer les filtres de recherche, à l'image d'un patron de conception « monteur ». Il renvoie finalement la liste des fichiers correspondant aux filtres spécifiés.

10.2. Module graphique (GUI)

Dans le cadre de notre projet, une interface graphique minimale nous a été demandée. Notre choix s'est porté sur le Framework Qt, une solution que nous maîtrisions déjà grâce aux enseignements de notre cursus de deuxième année.

10.2.1. Utilisation

Pour utiliser l'interface graphique, il suffit de se placer dans le dossier /gofs/Interface_Graphique_qt et y exécuter la commande "make". Après un temps de chargement des codes distants utilisés, la fenêtre apparaîtra. Il ne reste alors plus qu'à ouvrir une image via le menu dédié et remplir les différents champs.

10.2.2. Conception

Nous avons utilisé un projet GitHub proposant déjà l'interfaçage nécessaire à l'utilisation de Qt avec le langage Go. Cela nous a permis un gain de temps et d'efficacité en utilisant simplement les fonctions et objets Qt légèrement différemment. Par exemple, plusieurs fonctions au même nom mais numérotés "une_fonctionN" permettent de reproduire la surcharge de Qt.

10.3. Module ligne de commande (CLI)

Une Interface CLI a été réalisée pour ce projet, c'est-à-dire une interface accessible depuis le terminal qui permet d'effectuer une recherche dans une archive EWF.

Il s'agit d'un exécutable Go s'appuyant sur notre implémentation du module Finder.

10.3.1. Utilisation

Son utilisation repose sur l'appel de l'exécutable accompagné de plusieurs arguments, dont le détail est accessible via la commande --help.

Dans sa configuration standard, le premier paramètre correspond toujours à une image EWF et le second à une chaine de caractère (à chercher dans l'image). Puis, conformément à l'ordre choisi, une taille maximale en Ko puis la date au format JJ/MM/AAAA.

```
 $ ./cli_tool --help

Utilisation :     [image EWF] [nom de fichier]

Utilisation : -d  [image EWF] [nom de fichier] [dateMin] [dateMax]

Utilisation : -s  [image EWF] [nom de fichier] [sizeMin] [sizeMax]

Utilisation : -ds [image EWF] [nom de fichier] [dateMin] [dateMax]
[sizeMin] [sizeMax]

Utilisation : -sd [image EWF] [nom de fichier] [sizeMin] [sizeMax]
[dateMin] [dateMax]

Format [dateMin] [dateMax] : jj/mm/aaaa

Unité [sizeMin] [sizeMax] Ko
```

10.3.2. Exemple

```
 $ ./cli_tool -d ../../images/text_files.E01 "." 28/01/2026 29/01/2026


lost+found

my_folder

my_folder/text1.txt

my_folder/text2.jpg

my_folder/text3
```

10.3.3. Conception

L'interface CLI se veut volontairement minimaliste :

- La sortie n'est pas stylisée pour que les résultats soient facilement exploitables
- Le code est simple et court
- Le Finder dispose d'un nombre restreint de paramètres, ce qui limite la taille de l'outil

DEFIS RENCONTRES ET SOLUTIONS

11. Librairies locales

Utilisant les librairies EWF et TSK, notre programme devient dépendant des programmes déjà installés sur l'ordinateur hôte. Afin de réduire ces dépendances, nous avons cherché à les fournir localement dans le dépôt pour que l'utilisateur n'ait pas besoin d'installer les librairies requises. Nous gagnons ainsi en autonomie, cependant cela rend le projet plus lourd puisqu'il est nécessaire de fournir les librairies pour les deux principaux systèmes d'exploitation que sont Windows et Linux.

12. Cross compilation

La compilation sur différents systèmes d'exploitation a représenté un réel défi en raison des bibliothèques externes utilisées. Nous avons implémenté une solution pour compiler la librairie EWF en local à partir de notre unique dépôt Git sur les systèmes Linux Git et Windows, sans dépendance externe sur le système hôte. Toutefois, ce mécanisme n'a pas encore été étendu à la bibliothèque TSK.

CONCLUSION

Notre projet permet une exploitation des archives EWF en langage Go et s'adapte à l'ensemble des systèmes trouvables aujourd'hui. La définition des tâches à réaliser a demandé un travail significatif de recherche et documentation individuel afin d'appréhender et comprendre le Go, les archives EWF, les images disques et les systèmes de partitionnement.

Notre travail peut se placer en amont d'une recherche plus avancée basée sur le contenu des fichiers. Leur exploitation peut mener à vérifier la cohérence entre les extensions des fichiers et leur vraie nature, dans l'éventualité où l'extension ait par exemple été volontairement modifiée. D'autres modules permettraient de pousser la recherche dans des fichiers plus complexes tels que les documents PDF, Word, les images ou encore les archives ZIP.

Nous remercions M. Emmanuel GIGUET et M. Tanguy GERNOT pour le suivi continu de notre projet, nous rencontrant fréquemment pour réaliser un point sur notre avancée et nous encourageant à explorer certaines pistes lorsqu'elles étaient prometteuses.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'ingénieures et d'ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04