

Contenus Suspects

**Matéo Delerue-Houard
Sylia Bouimedj**

Objectifs :

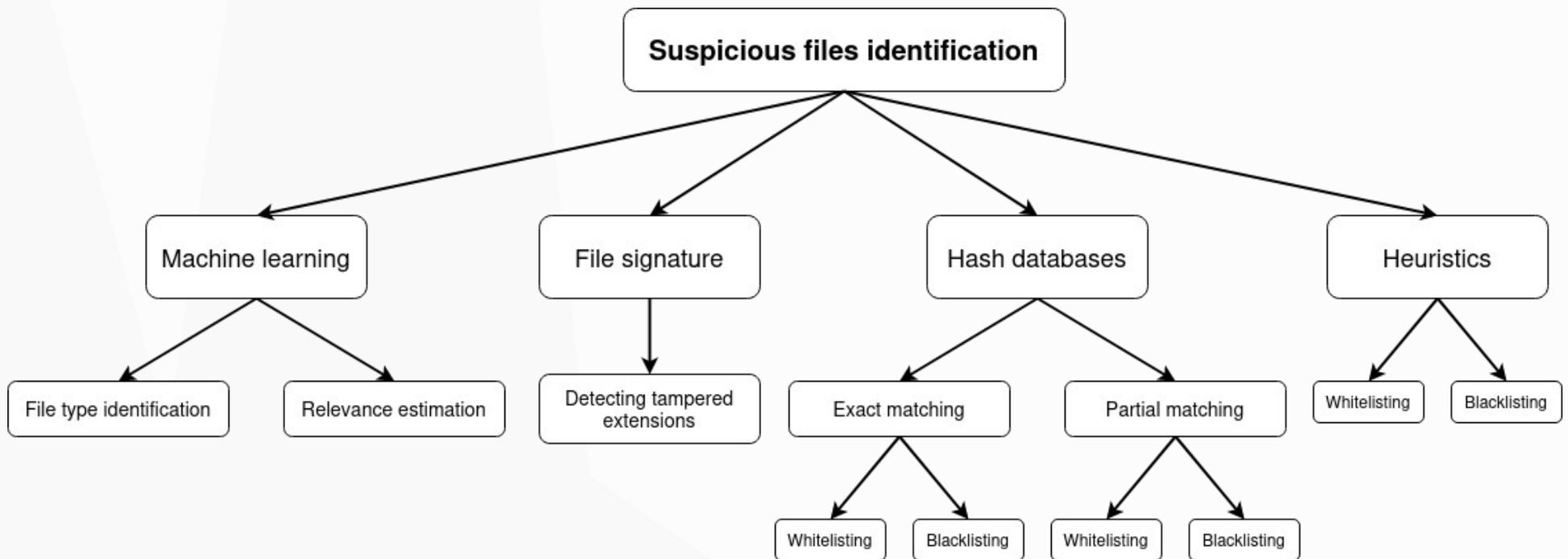
- **Trouver des fichiers cachés par des utilisateurs sans connaissance techniques :**

- **Modification des extensions de fichiers**
- **Déplacement de fichiers dans des répertoires légitimes**
- **Fichiers marqués comme cachés**

Ne concerne pas :

- **Chiffrement**
 - **Stéganographie**
 - **Anti-forensic**
- **Filtrage des fichiers légitimes**
 - **Outil rapide**

État de l'art



Implémentation

- **Programme écrit en C++**
- **Identification du type de fichier avec fidentify¹**
- **Base de donnée de hash (schéma RDS)**
- **Recherche par mot clé**

[1] : <https://www.cgsecurity.org/wiki/PhotoRec>

Détection de fichiers cachés

- **Fichiers cachés situés dans les répertoires utilisateurs (Vidéos, Images, Téléchargements...) considérés comme suspects**
- **Support de Windows & plateformes conforme au standard XDG (Linux, BSD...)**

Nouvelles heuristiques

- **Filtrage par taille : élimination des trop petit fichiers**
- **Recherche par intervalle de dates**

Optimisation

- **Profilage du code avec callgrind**

The screenshot displays the Callgrind GUI with the following components:

- &Top Cost Call Stack:** A table showing the top cost call stack. The entry for `SHA1FileChunk` is highlighted in blue.
- Flat &Profile:** A table showing the flat profile of the program. The entry for `SHA1FileChunk` is highlighted in blue.
- SHA1FileChunk Callers:** A table showing the callers of the selected function. The caller `SHA1File (libmd.so.0.1.0)` is highlighted in blue.
- Call Graph:** A call graph showing the flow of control. The path from `HashDB::analyze_file(std::filesystem::__cxx11::path const&)` to `SHA1File` to `SHA1FileChunk` to `SHA1Update` is highlighted in blue.

Ir	Cost2	Calls	Function
100.00		1	main
99.99		1	Analysis::run()
99.92		115	HashDB::analyze_file(std::filesystem::__cxx11::p...
98.23		114	SHA1File
99.91		115	SHA1FileChunk
99.90		648 219	SHA1Update

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000001bb60	ld-linux-x8
100.00	0.00	1	(below main)	analyze_lit
100.00	0.00	1	(below main)	libc.so.6
100.00	0.00	1	0x00000000000025c50	libc.so.6
100.00	0.00	1	main	analyze_lit
99.99	0.00	1	Analysis::run()	analyze_lit
99.92	0.00	115	HashDB::analyze_file(std::files...	analyze_lit
99.91	0.01	115	SHA1FileChunk	libmd.so.0
99.90	0.60	652 259	SHA1Update	libmd.so.0
99.29	99.29	82 963 656	SHA1Transform	libmd.so.0
98.23	0.00	114	SHA1File	libmd.so.0
1.69	0.00	70	0x00000000000012af0	ld-linux-x8
0.06	0.00	115	ExtensionMismatchDetector::...	analyze_lit
0.06	0.00	115	file_identify	analyze_lit
0.06	0.00	893	MALLOC	analyze_lit
0.06	0.06	1 389	0x000000000000157c80	libc.so.6
0.01	0.00	1	std::_detail::_MakeUniq<Ext...	analyze_lit
0.01	0.00	1	ExtensionMismatchDetector::...	analyze_lit

Types	Callers	All Callers	Callee Map	Source Code
Ir	Ir per call	Count	Caller	
99.91	1 051 922 201	115	SHA1File (libmd.so.0.1.0)	

callgrind.out.23 [1] - Total Instruction Fetch Cost: 121 077 263 876

Optimisation

- **Benchmark OpenSSL vs libmd :**

```
Benchmark 1: ./analyze_libmd --timestamp 0 -d rds.sqlite3 -k keywords.txt root
Time (mean ± σ):      3.469 s ± 0.121 s    [User: 1.761 s, System: 1.707 s]
Range (min ... max):  3.405 s ... 3.808 s    10 runs
```

```
Benchmark 2: ./analyze_openssl --timestamp 0 -d rds.sqlite3 -k keywords.txt root
Time (mean ± σ):      1.048 s ± 0.052 s    [User: 0.669 s, System: 0.378 s]
Range (min ... max):  1.010 s ... 1.180 s    10 runs
```

Summary

```
./analyze_openssl --timestamp 0 -d rds.sqlite3 -k keywords.txt root ran
 3.31 ± 0.20 times faster than ./analyze_libmd --timestamp 0 -d rds.sqlite3 -
k keywords.txt root
```

Gains de performance : x3,3

Protocole d'évaluation

- 1. Créer un volume de stockage légitime (installation d'OS)**
- 2. Dissimulation de contenus :**
 - Placement de fichiers dans des endroits aléatoires**
 - Changement aléatoire des extensions**
 - Marquer comme cachés des fichiers dans des dossiers utilisateurs**
- 3. Tester chaque techniques individuellement et en combinaison**
- 4. Relever leur taux de détection et de faux-positifs**

Résultats

Technique	Taux de détection	Faux-positifs
Identification du type de fichier	73 %	1099
Base de donnée de hash	35 %	11
Détection de fichier caché	3 %	0
Recherche par date	100 %	0
Tous combinés	100 %	1108

Pistes d'amélioration

- **Précision :**
 - **Recherche par types de fichiers (réduction des faux-positifs)**
 - **Ré-architecturer la base de donnée en arbre de Merkle**
 - **Brise la compatibilité avec le RDS**
- **Performances :**
 - **Implémentations plus performantes (DBMS, meilleurs algorithmes...)**
 - **Mappage mémoire des fichiers volumineux (mmap)**
 - **Parallélisation**