

Projet de Analyse d'évidence pour terminaux mobiles

Filière : Sécurité Informatique et Cybersécurité

---

## Cellebrite UFED : défis de l'ingénierie inverse et leurs conséquences

---

Présenté le : 03/01/2025

*Préparé par :*

Mr. Achraf ESSATAB  
Mr. Ayoub SETTOU  
Mme. Fatima Zohra NOUREDDINE  
Mme. Meryem SABBAR  
Mr. Saad BOUMADYAN

*Encadré par :*

Pr. Mohammed Amine KOULALI

Année Universitaire  
2024/2025

## Table de contenu

<b>1</b>	<b>Remerciements</b>	<b>2</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Contexte . . . . .	3
2.2	Objectifs du Projet . . . . .	3
<b>3</b>	<b>Outils et Techniques Utilisés</b>	<b>3</b>
3.1	Vue d'ensemble de l'outil forensic : Cellebrite UFED . . . . .	4
3.2	Interception du Trafic avec Wireshark . . . . .	5
3.3	Analyse Statique de Code . . . . .	5
3.4	Techniques d'Anti-Débogage et de Packing . . . . .	6
3.4.1	Le Role de Themida comme un Packer . . . . .	8
<b>4</b>	<b>Méthodologie</b>	<b>12</b>
4.1	Interception de la Communication entre le Laboratoire et la Cible . . . . .	13
4.2	Analyse Statique du Code Principal . . . . .	13
4.2.1	Analyse des Mécanismes de Packing de Themida . . . . .	16
4.3	Défis d'Ingénierie Inverse dans les Applications Commerciales . . . . .	16
<b>5</b>	<b>Résultats et Analyse</b>	<b>16</b>
5.1	Fonctionnalité du Mode Basique de l'Outil . . . . .	16
5.2	Le Routine <code>IsDebuggerPresent</code> . . . . .	19
5.3	Techniques Anti-Virtualisation . . . . .	21
<b>6</b>	<b>Discussion</b>	<b>23</b>
6.1	Implications Sécuritaires des Résultats . . . . .	23
6.2	Pistes Futures pour l'Analyse . . . . .	24
<b>7</b>	<b>Conclusion</b>	<b>24</b>

## 1 Remerciements

Un grand merci à Ayoub Settou pour sa riche collaboration, ainsi qu'à Fatima Zohra, Meryem et Saad. Nous souhaitons également exprimer notre gratitude pour la suggestion du sujet de ce projet proposée par notre professeur, car nous en voyons la valeur pour toute personne entrant dans le domaine de la cybersécurité.

## 2 Introduction

Avec l'augmentation de l'utilisation des appareils mobiles au sein de la population mondiale, une partie des experts en criminalistique numérique se concentre sur l'acquisition et l'analyse des données provenant des appareils Android et iOS.

L'un des outils les plus populaires, utilisé dans de nombreux tribunaux aux États-Unis et dans d'autres pays, est Cellebrite UFED. Cet outil permet aux experts en criminalistique numérique d'acquérir, d'analyser et de rapporter leurs conclusions. Il est spécialement conçu pour les appareils Android et iOS dans différents scénarios.

Les cibles de Cellebrite UFED incluent des appareils éteints, des appareils allumés mais verrouillés, ainsi que des appareils déverrouillés.

Cellebrite UFED est un outil commercial qui utilise des techniques pour protéger la propriété intellectuelle de son fournisseur. Ces techniques peuvent également compliquer l'examen des problèmes de sécurité liés à l'outil, ce qui pourrait être exploité par des attaquants ayant réussi à le désobfusquer.

En effet, en 2021, Signal, un service de communication chiffrée open source, a découvert des vulnérabilités dans Cellebrite UFED qui pourraient permettre des attaques contre le laboratoire criminalistique et altérer les rapports générés par l'outil, rapports destinés à servir de preuves devant un tribunal.<sup>1</sup>

Ce projet vise à identifier les techniques anti-ingénierie inversée utilisées par cet outil et à expliquer son fonctionnement (de l'outil) interne sous sa forme la plus simple : l'acquisition de données forensics d'un appareil Android déverrouillé. En terminant par montrer les résultats d'une telle obfuscation sur la fiabilité de l'outil.

### 2.1 Contexte

Lorsqu'une application telle que Cellebrite UFED est fortement utilisée dans le contexte de la criminalistique numérique ou de la justice, il est essentiel d'examiner sa sécurité avec soin.

Les techniques utilisées par les entreprises développant des logiciels commerciaux partagés sont utiles dans le cadre de la protection de la propriété intellectuelle, mais elles masquent le fonctionnement interne du logiciel, même pour les chercheurs en sécurité.

Ainsi, comprendre ces mécanismes de protection constitue la première étape pour sécuriser ce type de logiciel.

### 2.2 Objectifs du Projet

Ce projet vise à :

- Mettre en lumière les obstacles qui ont interrompu l'ingénierie inversée de l'application.
- Expliquer le fonctionnement de l'application sous sa forme la plus simple, lorsque l'ingénierie inversée n'est pas nécessaire.
- En outre, nous mettons en évidence l'attaque résultant sous telles techniques d'obfuscation.

## 3 Outils et Techniques Utilisés

Pour la première tâche, à savoir la découverte des mécanismes de protection, nous avons utilisé des outils d'analyse statique:

- x32dbg pour suivre l'exécution de l'application.

---

<sup>1</sup><https://signal.org/blog/cellebrite-vulnerabilities/>

- DetectitEasy pour identifier les motifs liés à l'outil de protection utilisé.
- IDA pour analyser le code de l'exécutable après dépaquetage.

Pour intercepter les communications échangées entre le laboratoire criminalistique de Cellebrite et l'appareil cible, nous avons utilisé Wireshark.

### 3.1 Vue d'ensemble de l'outil forensic : Cellebrite UFED

Cellebrite Universal Forensic Extraction Device (UFED) est une solution matérielle et logicielle propriétaire développée par Cellebrite pour extraire et analyser les données provenant d'appareils mobiles. UFED est largement utilisé par les forces de l'ordre, les agences de renseignement et les experts en criminalistique pour récupérer des preuves numériques depuis des smartphones, tablettes et autres appareils portables. Ses fonctionnalités incluent les extractions logiques et physiques, la récupération de données supprimées et le déchiffrement des appareils verrouillés. Les preuves numériques extraites à l'aide de l'UFED sont souvent présentées devant les tribunaux dans le cadre de dossiers pénaux et civils. Pour que ces preuves soient recevables, les enquêteurs doivent respecter des normes rigoureuses afin de garantir l'intégrité et l'authenticité des données. Les mécanismes de journalisation automatisée des processus d'extraction, de vérification par hachage et de gestion de la chaîne de traçabilité de l'UFED renforcent sa crédibilité dans les procédures légales. Les tribunaux exigent généralement que les examinateurs démontrent que les extractions ont suivi des protocoles appropriés et que les données ont été manipulées de manière à préserver leur valeur judiciaire. L'architecture logicielle de Cellebrite UFED est modulaire, conçue pour prendre en charge divers protocoles d'appareils, formats de données et flux de travail forensiques. Elle comprend plusieurs composants clés :

- Couche d'Extraction de Données : Interface avec les appareils via des pilotes, exploits et protocoles propriétaires.
- Moteur de Décodage : Traduit les données binaires extraites en formats lisibles, tels que messages, contacts et journaux d'applications.
- Couche de Déchiffrement : Gère les méthodes de contournement des cryptages, comme le brute-force ou l'utilisation de vulnérabilités dans le système d'exploitation de l'appareil.
- Interface Utilisateur (UI) : Offre aux enquêteurs des flux de travail intuitifs pour configurer les extractions, consulter les journaux et générer des rapports.

Le logiciel est compatible avec les unités matérielles de Cellebrite et fonctionne sur des stations de travail forensiques basées sur Windows. Le logiciel UFED de Cellebrite est généralement distribué sous forme d'un installateur exécutable pour Windows, souvent compacté à l'aide de packers commerciaux pour empêcher tout accès ou altération non autorisés. Ces packers peuvent également obscurcir la structure interne du logiciel, rendant l'ingénierie inverse ou l'analyse non autorisée plus difficile. Cependant, une fois déployé, le logiciel se décompacte en plusieurs modules et bibliothèques dynamiques (DLL) pour son fonctionnement. Le logiciel UFED utilise de nombreuses DLLs pour :

- Communication avec les Appareils : Bibliothèques dédiées à l'interaction avec des types spécifiques d'appareils (par ex. iOS, Android ou téléphones classiques).
- Fonctions de Déchiffrement : Modules propriétaires pour déverrouiller les données chiffrées.
- Analyse et Décodage : DLLs pour interpréter les données au niveau des applications (par ex. WhatsApp, Facebook ou email).

- Génération de Rapports : DLLs formatant les données extraites en formats recevables par les tribunaux, comme PDF ou HTML.

Chaque DLL remplit une fonction spécifique, permettant des mises à jour modulaires et le support de nouveaux appareils et formats. *Pour comprendre le fonctionnement de cette application, nous devons reconstituer ses fonctionnalités à partir du code machine. Dans le cas de Cellebrite, l'utilisation du packer rend ce processus difficile et loin d'être simple.*

### 3.2 Interception du Trafic avec Wireshark

### 3.3 Analyse Statique de Code

La première chose que nous avons commencée est de découvrir sur quelle architecture informatique l'application est censée fonctionner. Cellebrite UFED est conçue pour fonctionner sur une architecture informatique 32 bits exécutant un système d'exploitation Windows. Cela guide le choix de l'outil d'analyse statique approprié : x32dbg. Le dossier d'installation de Cellebrite UFED contient, entre autres : une application d'interface utilisateur, des fichiers DLL et des fichiers ZIP cryptés liés aux fonctionnalités principales de l'application. Après une investigations des ressources sur internet<sup>2</sup>, les

th	12/22/2024 6:19 AM	File folder	
ThirdPartyNotices	12/22/2024 6:40 AM	File folder	
tr	12/22/2024 6:19 AM	File folder	
Utils	12/22/2024 6:27 AM	File folder	
vi	12/22/2024 6:19 AM	File folder	
x64	12/22/2024 6:19 AM	File folder	
x86	12/22/2024 6:19 AM	File folder	
zh-CHS	12/22/2024 6:19 AM	File folder	
zh-CHT	12/22/2024 6:19 AM	File folder	
python38._pth	6/13/2023 6:03 PM	_PTH File	1 KB
CefSharp.BrowserSubprocess	1/16/2023 2:33 AM	Application	7 KB
Cellebrite.ResetDefaultValues	12/7/2023 7:22 PM	Application	14 KB
Cellebrite.UFEDPermissionManager	12/7/2023 7:22 PM	Application	455 KB
DemofyExodus	12/7/2023 7:22 PM	Application	22 KB
Exodus.CellebriteTouch	12/7/2023 10:07 AM	Application	19,494 KB
ExportLog	12/7/2023 7:23 PM	Application	436 KB
hasp_rt	4/20/2023 12:28 AM	Application	3,261 KB
HookExe64	12/5/2023 2:11 PM	Application	20 KB
InnerLoader	12/7/2023 10:07 AM	Application	2,706 KB
Loader	12/3/2023 2:27 PM	Application	10,031 KB

Figure 1: Le dossier d'installation

Zip files et dll sont chiffrés et contient des généralement des exploitation soit pour des vulnérabilités

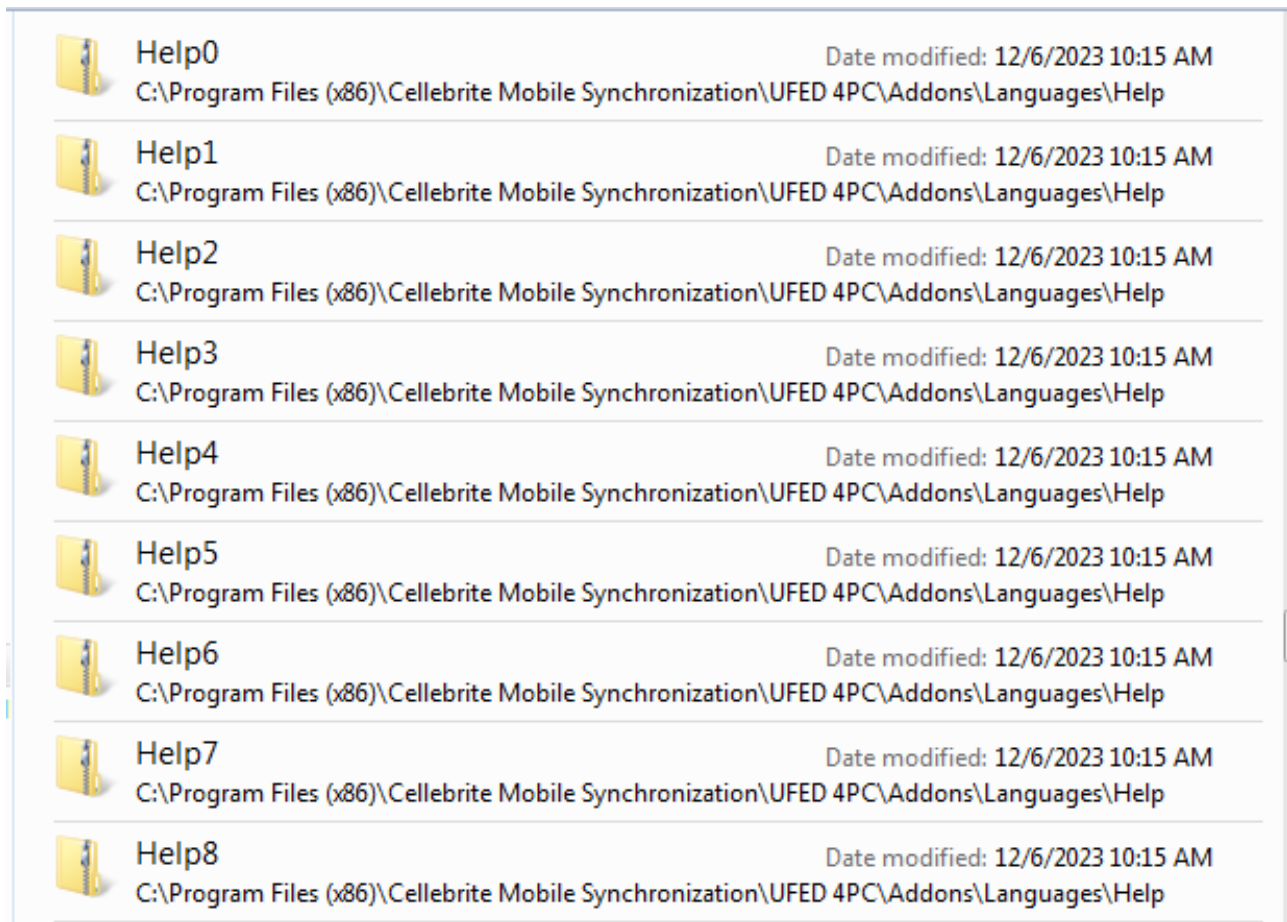


Figure 2: ZIP files

de l'implémentation des méthodes cryptographique, le design du boot-sequence ou bien du kernel. Avant d'examiner le code, pour confirmer l'utilisation de chiffrement, nous avons utilisé un outil appelé PEId, accompagné d'une série de plugins. En particulier, le plugin KryptoANALyzer a été utilisé pour vérifier s'il existe des motifs indiquant l'utilisation de méthodes de cryptage dans un échantillon d'exécutables et de DLL.

La figure 4 présente le résultat de l'un des tests. En effet, des bibliothèques cryptographiques sont utilisées, ainsi que des nombres premiers et d'autres preuves solides. À ce stade, l'utilisation d'un débogueur était impossible – Ollydbg affichait un message d'erreur lorsque nous avons essayé de l'utiliser. Nous avons donc tenté de récupérer une forme approximative du code principal exécutable en utilisant le décompilateur IDA.

Le pseudocode dans la figure 5 est suffisamment concis pour indiquer l'utilisation de techniques d'obfuscation. Ce n'est pas le code original de l'application.

### 3.4 Techniques d'Anti-Débogage et de Packing

Le packing consiste à transformer l'exécutable d'un programme en un format différent qui reste fonctionnel mais plus difficile à analyser. Cette transformation inclut généralement :

<sup>2</sup>[https://blog.korelogic.com/blog/2020/06/29/cellebrite\\_good\\_times\\_come\\_on](https://blog.korelogic.com/blog/2020/06/29/cellebrite_good_times_come_on)





















 Dump_FileSystem.dll	12/7/2023 10:06 AM	Application extens...	1,507 KB
 Dump_FileSystem_MTK.dll	12/7/2023 7:22 PM	Application extens...	78 KB
 Dump_FileSystemSrc.dll	12/7/2023 7:22 PM	Application extens...	62 KB
 Dump_Huawei.dll	12/7/2023 10:06 AM	Application extens...	2,948 KB
 Dump_Huawei_Vendor.dll	12/7/2023 10:06 AM	Application extens...	3,172 KB
 Dump_iDEN.dll	12/7/2023 7:22 PM	Application extens...	100 KB
 Dump_iPhone.dll	12/7/2023 7:22 PM	Application extens...	212 KB
 Dump_iPhone_MNM.dll	12/7/2023 10:06 AM	Application extens...	2,872 KB
 Dump_Lg_Android_LAF.dll	12/7/2023 10:06 AM	Application extens...	2,534 KB
 Dump_LgCDMA.dll	12/7/2023 7:22 PM	Application extens...	67 KB
 Dump_MotCxx.dll	12/7/2023 7:22 PM	Application extens...	57 KB
 Dump_MotGSM.dll	12/7/2023 10:06 AM	Application extens...	2,830 KB
 Dump_MotP2K.dll	12/7/2023 7:22 PM	Application extens...	62 KB
 Dump_MTK_KO.dll	12/7/2023 10:06 AM	Application extens...	2,784 KB
 Dump_MTP.dll	12/7/2023 10:06 AM	Application extens...	2,328 KB
 Dump_Nokia_WP.dll	12/7/2023 10:06 AM	Application extens...	2,392 KB
 Dump_NokiaAsha.dll	12/7/2023 10:06 AM	Application extens...	2,276 KB
 Dump_NokiaTA.dll	12/7/2023 7:22 PM	Application extens...	310 KB
 Dump_Oasis.dll	12/7/2023 10:06 AM	Application extens...	3,657 KB
 Dump_Palm.dll	12/7/2023 7:22 PM	Application extens...	65 KB

Figure 3: Les modules externes

- La compression de l'exécutable pour réduire sa taille.
- Le chiffrement ou l'obfuscation de l'exécutable pour compliquer son analyse.
- L'ajout d'un décompresseur ou déchiffreur à l'exécution, qui restaure le code en mémoire lors de l'exécution.

Son objectif principal est la protection de la propriété intellectuelle en empêchant tout accès non autorisé au code source ou aux algorithmes propriétaires. Cela vise également à rendre plus difficile pour les attaquants ou les chercheurs de comprendre la logique du programme, ce que l'on appelle l'anti-ingénierie inversée.

Comme effets secondaires, les packers permettent de compresser l'exécutable afin de minimiser l'espace de stockage et l'utilisation de la bande passante. Dans le cas de logiciels malveillants, le packing est souvent utilisé pour échapper à la détection basée sur les signatures par les outils antivirus.

Pour packer un programme, plusieurs techniques sont utilisées :

- On peut compresser le binaire du programme en une forme non compréhensible et l'envelopper avec une routine de décompression. Le code original est compressé et stocké dans un nouvel exécutable. Ainsi, lorsque le programme est exécuté, la routine de décompression s'exécute en premier, restaurant le code original en mémoire.
- On peut chiffrer le binaire du programme pour en obscurcir le contenu. Le binaire devient dépendant d'une clé qui est transmise à une routine de déchiffrement. Cette routine est intégrée dans la version packée du logiciel.



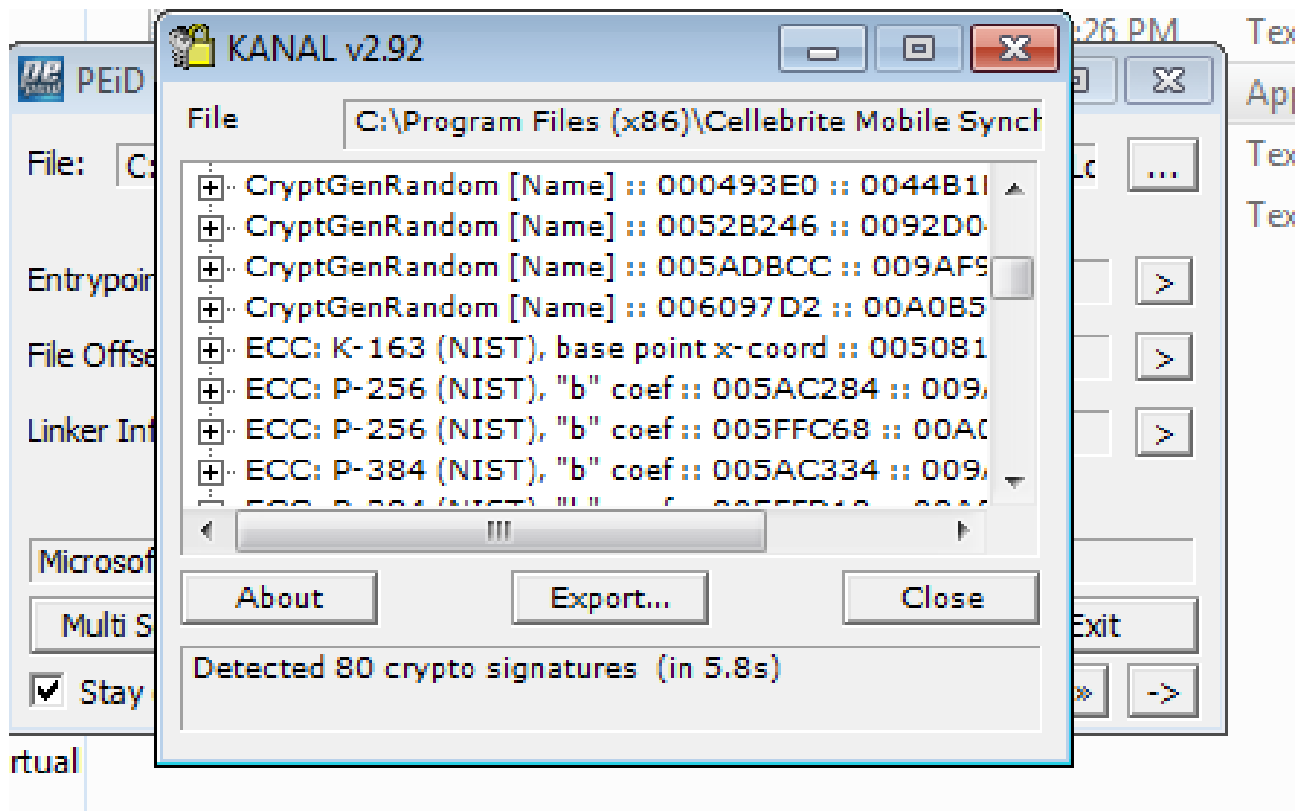


Figure 4: Evidence de chiffrement

- On peut également réorganiser ou cacher le code à l'intérieur du binaire pour le rendre plus difficile à comprendre. Cela se fait généralement en insérant des instructions inutiles qui conservent la même fonctionnalité du programme, en renommant les fonctions ou les variables avec des chaînes sans signification, ou encore en réorganisant les blocs de code dans un ordre différent de celui d'origine.
- Certains outils dédiés entravent activement le processus d'analyse. Ils peuvent, par exemple, invoquer des fonctions qui vérifient si le programme est en cours de débogage ou modifier leur propre comportement pendant l'exécution.
- Dans les cas extrêmes, les packers peuvent modifier le jeu d'instructions du programme et intégrer un interpréteur spécialisé.

Comme indiqué dans l'introduction de ce document, les packers sont utiles pour protéger la propriété intellectuelle, mais ils peuvent retarder le processus de détection des vulnérabilités dans des applications considérées comme fiables.

### 3.4.1 Le Role de Themida comme un Packer

Lorsqu'une application est créée, le compilateur compile le code source de l'application en plusieurs fichiers objet constitués de code en langage machine. Ensuite, les fichiers objet sont liés ensemble pour créer l'exécutable final.<sup>3</sup>

<sup>3</sup><https://www.oreans.com/Themida.php>

```

/* WARNING: Restarted to delay deadcode elimination for space: stack */

int entry(byte *param_1,undefined4 param_2,byte *param_3)
{
    byte bVar1;
    byte bVar2;
    uint uVar3;
    int iVar4;
    byte bVar5;
    int iVar6;
    uint unaff_EBP;
    byte *pbVar7;
    byte *pbVar8;
    bool bVar9;
    bool bVar10;
    bool bVar11;
    bool bVar12;
    bool bVar13;

    FUN_0082e1a8();
    bVar5 = 0x80;
    pbVar8 = param_3;
    do {
        bVar1 = *param_1;
        param_1 = param_1 + 1;
        *pbVar8 = bVar1;
        pbVar8 = pbVar8 + 1;
        iVar6 = 2;
LAB_0082e075:
        bVar9 = CARRY1(bVar5,bVar5);
        bVar5 = bVar5 * '\x02';
        bVar10 = bVar9;
        if (bVar5 == 0) {
            bVar5 = *param_1;
            param_1 = param_1 + 1;
            bVar10 = CARRY1(bVar5,bVar5) || CARRY1(bVar5 * '\x02',bVar9);
        }
    } while (bVar10);
}

```

Figure 5: Pseudocode du program principal

De la même manière que le code source d’une application est converti en code machine au moment de la compilation, il existe des outils capables de convertir une application compilée en langage d’assemblage ou en un langage de programmation plus avancé. Ces outils sont appelés désassembleurs et décompilateurs.

Un acteur malveillant peut utiliser un désassembleur ou un décompilateur pour étudier le fonctionnement d’une application spécifique et ce que fait une routine particulière. Lorsqu’un attaquant possède une bonne connaissance de l’application cible, il peut modifier l’application compilée pour en altérer le comportement. Par exemple, l’attaquant pourrait contourner la routine qui vérifie la période d’essai d’une application pour la faire fonctionner indéfiniment, ou pire encore, faire en sorte que



Figure 6: Compilation du code source

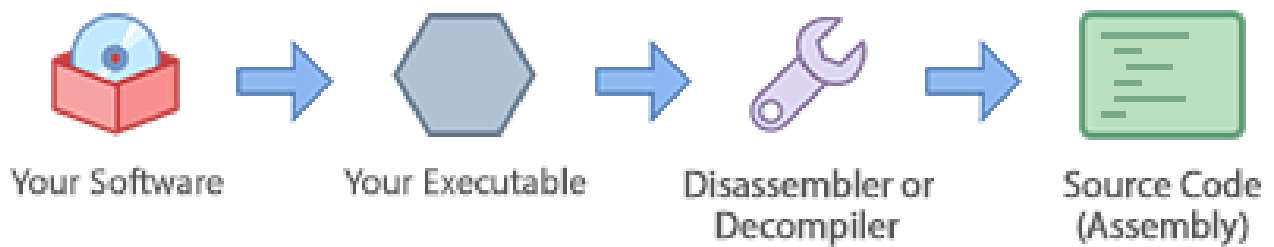


Figure 7: Décompilation de l'exécutable

l'application se comporte comme si elle était enregistrée.

C'est la raison qui a motivé l'invention des protecteurs logiciels. Les protecteurs logiciels ont été créés pour empêcher un attaquant d'inspecter directement ou de modifier une application compilée. Un protecteur logiciel agit comme un bouclier qui garde une application cryptée et protégée contre d'éventuelles attaques. Lorsqu'une application protégée doit être exécutée par le système d'exploitation, le protecteur logiciel prend d'abord le contrôle du processeur (CPU) et vérifie la présence éventuelle d'outils de cracking (désassembleurs ou décompilateurs) en cours d'exécution sur le système. Si tout est sécurisé, le protecteur logiciel procède au décryptage de l'application protégée et lui transfère le contrôle du processeur pour une exécution normale.

Les avantages d'utiliser un protecteur logiciel sont :

- Protéger une application contre le piratage.
- Empêcher les attaquants d'étudier la manière dont une application est implémentée.
- Ne pas permettre aux attaquants de modifier une application pour en changer le comportement.

Depuis la naissance des protecteurs logiciels, de nombreux attaquants ont concentré la plupart de leurs efforts sur l'attaque des protecteurs logiciels eux-mêmes plutôt que des applications. De nombreux outils ont été développés pour aider à attaquer les protecteurs logiciels. Ces attaques aboutissent souvent à ce que l'attaquant obtienne l'application originale, décryptée et dépourvue de son enveloppe de protection.

Le principal problème des protecteurs logiciels est qu'ils utilisent des techniques de protection bien connues des crackers, ce qui les rend facilement contournables avec des outils de cracking traditionnels. Un autre problème important est que les protecteurs logiciels s'exécutent avec des privilèges normaux d'application, ce qui signifie que des attaquants peuvent utiliser des outils de cracking fonctionnant au même niveau de priorité que le système d'exploitation, leur permettant ainsi de superviser entièrement les actions d'un protecteur logiciel et de l'attaquer à des endroits spécifiques.



Figure 8: Le processus

Comme OREANS (la société derrière Themida) l'affirme dans la documentation de leur produit Themida, leur principal objectif est de contrer les attaques contre les protecteurs/packers. Grâce à une technologie appelée SecureEngine, ils peuvent stopper les attaques contre les packers car cette technologie fonctionne à un niveau de priorité supérieur.

La protection est mise en œuvre au niveau des blocs de code comme illustré ci-dessous.



Figure 9: Blocs protégés

OREANS affirme que SecureEngine défie tous les outils de cracking actuels pouvant être utilisés contre des applications protégées et s'assure que vos applications protégées ne s'exécutent que dans des environnements sécurisés.



Figure 10: SecureEngine

Une liste complète ci-dessous présente toutes les principales fonctionnalités de Themida :

- Techniques anti-débogueur qui détectent/trompent tout type de débogueur.
- Algorithmes et clés de cryptage différents dans chaque application protégée.
- Techniques anti-scanners API qui empêchent la reconstruction de la table d'importation originale.
- Techniques de décompilation automatique et de brouillage dans l'application cible.
- Émulation de machine virtuelle dans des blocs de code spécifiques.
- Moteur de mutation avancé.
- Communication SDK avec la couche de protection.
- Techniques anti-désassemblage pour tout désassembleur statique et interactif.

- Multiples couches polymorphiques avec plus de 50 000 permutations.
- Techniques avancées d’encapsulation API.
- Techniques anti-moniteurs contre les moniteurs de fichiers et de registres.
- Insertion de code inutile entre les instructions réelles.
- Threads de protection spécialisés.
- Communication avancée en réseau des threads.
- Techniques anti-patch mémoire et CRC dans l’application cible.
- Moteur métamorphique pour brouiller les instructions originales.
- Protection avancée des points d’entrée.
- Cryptage dynamique dans l’application cible.
- Insertion de code anti-trace entre les instructions réelles.
- Gestionnaire avancé anti-point de rupture.
- Protection en temps réel dans l’application cible.
- Compression de l’application cible, des ressources et du code de protection.
- Techniques anti-“debugger hidere”.
- Mutation complète dans le code de protection pour éviter la reconnaissance de modèles.
- Simulation en temps réel dans l’application cible.
- Insertion intelligente de code de protection à l’intérieur de l’application cible.
- Relocalisation aléatoire des données internes.
- Possibilité de personnaliser les boîtes de dialogue dans l’application protégée.
- Support de la ligne de commande.

## 4 Méthodologie

- À l’aide d’un intercepteur de communication, Wireshark, nous avons visualisé comment l’outil effectue l’acquisition de données dans le scénario suivant :
  - Le dispositif cible est un ASUS ZenFone 5Q fonctionnant sous Android 7.1.
  - L’appareil est allumé.
  - L’appareil est déverrouillé et le mode débogage est activé.

La raison de ce choix de conditions est qu’elles ne nécessitent pas d’efforts d’ingénierie inverse pour analyser le fonctionnement interne de l’application dans ces circonstances.

- Lorsqu’une ingénierie inverse est nécessaire, nous avons mis en lumière uniquement les aspects qui compliquent ce processus en présence de techniques de packing.

### 4.1 Interception de la Communication entre le Laboratoire et la Cible

Après avoir connecté l'appareil cible avec un câble USB relié à la station de travail où Cellebrite UFED est en cours d'exécution, nous avons démarré une session sur Wireshark pour capturer les paquets de données transitant par le même port USB (figure 11). Sous Cellebrite UFED, nous avons choisi l'option

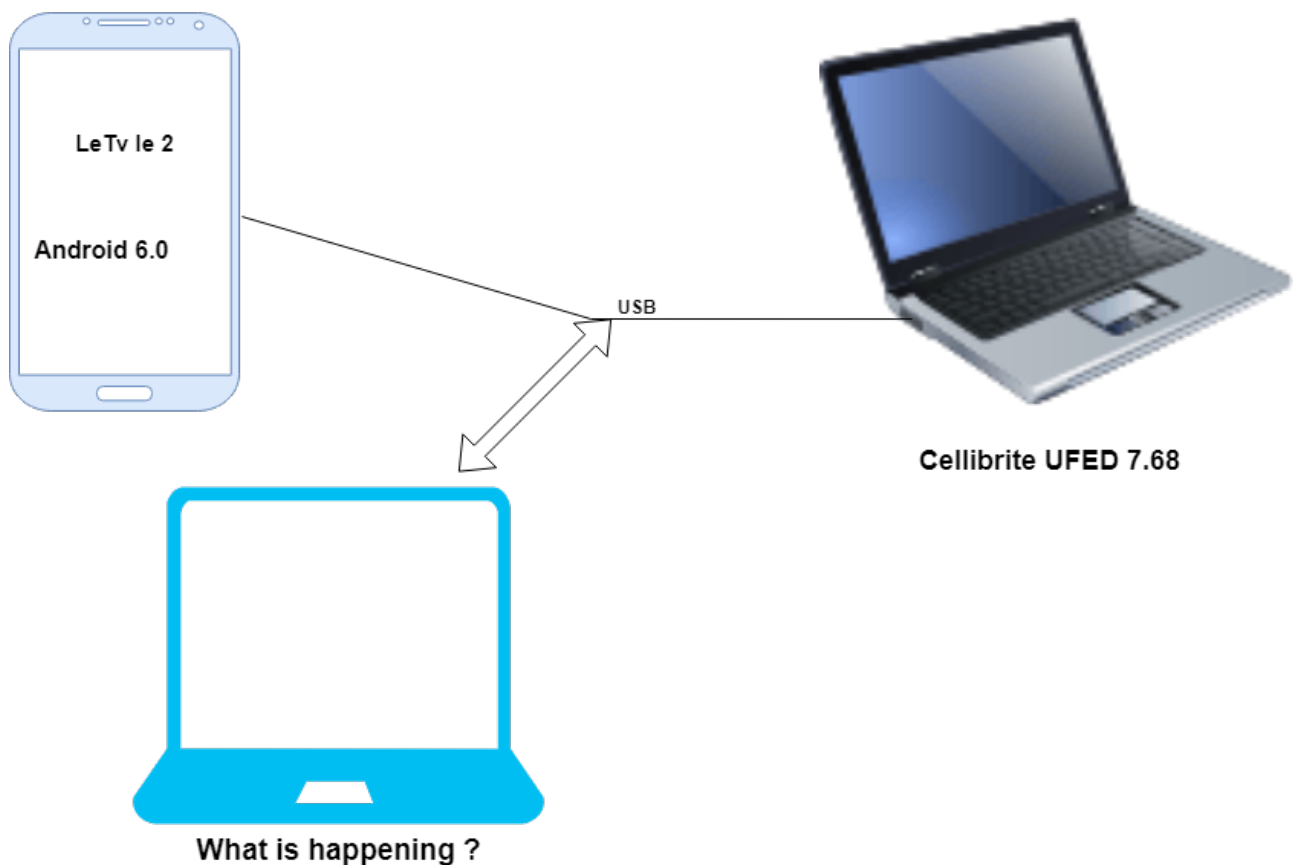


Figure 11: Lab setup

d'acquisition logique avancée pour le modèle spécifique de l'appareil, comme illustré sur la figure 12.

### 4.2 Analyse Statique du Code Principal

Dans le contexte de ce document, par analyse statique de code, nous entendons l'identification de motifs qui permettent de détecter statiquement l'utilisation de techniques d'anti-ingénierie inverse. Un outil appelé Detect-It-Easy est couramment utilisé pour cette tâche. Il fournit une analyse approfondie d'un exécutable ou d'une bibliothèque dynamique (DLL) cible afin de trouver des traces indiquant l'utilisation d'un packer. De plus, il peut spécifier quel packer a été utilisé pour produire le binaire. Ensuite, nous avons utilisé le décompilateur IDA et le désassembleur x32dbg pour examiner certaines des techniques employées par le packer.

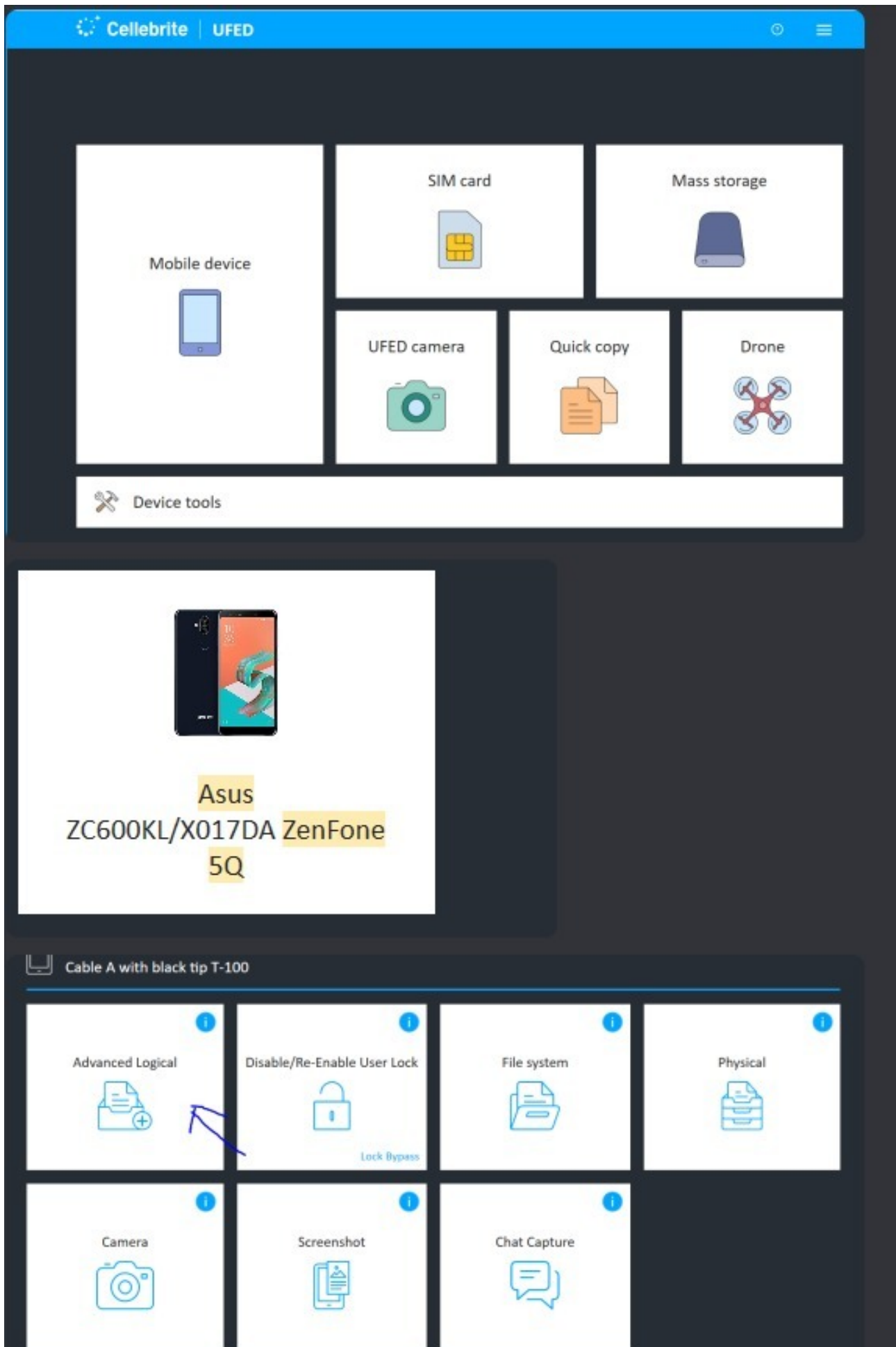


Figure 12: L'interface de cellebrite UFED

value
section[7]
.themida
n/a
n/a
n/a
0x00029200
0x00029200
0x00000000 (0 bytes)
0x00046000
0x003E8000 (4096000 bytes)

Figure 13: La section .themida



#### 4.2.1 Analyse des Mécanismes de Packing de Themida

L'utilisation du packer Themida a également été confirmée par la présence d'une section appelée `.themida` dans le fichier PE de l'exécutable principal (voir figure 13). Lorsque nous étions suffisamment certains de l'utilisation du packer Themida, nous avons tenté de repérer où leurs techniques de packing étaient implémentées en effectuant une analyse statique du binaire principal. Cela a nécessité de contourner l'une des techniques de packing : `IsDebuggerPresent`. La méthode de contournement s'est appuyée sur un plugin de x32dbg appelé `ScyllaHide`.

### 4.3 Défis d'Ingénierie Inverse dans les Applications Commerciales

Les défis qui rendent difficile la progression de ce projet incluent :

- Le temps nécessaire pour se familiariser avec les routines spécifiques à l'environnement Windows et le style de programmation associé.
- Le temps et les efforts requis pour suivre la chaîne que l'application utilise pour déchiffrer le contenu des fichiers de grande valeur (exploits).
- Ce dernier défi résulte de la couche d'obfuscation ajoutée par Themida au niveau des blocs de code.

## 5 Résultats et Analyse

À la suite de la méthodologie décrite, nous avons pu mettre en évidence les messages envoyés par Cellebrite UFED à l'appareil cible. Nous avons constaté que, dans les conditions spécifiées, les messages sont des commandes ADB typiques et sont envoyés en texte clair.

En ce qui concerne les routines utilisées pour entraver le processus d'analyse, l'application vérifie la présence d'un débogueur en invoquant une routine Windows appelée `IsDebuggerPresent`. En tentant d'exécuter l'application dans un environnement virtualisé, nous avons découvert que l'application utilise des techniques anti-virtualisation. Nous n'avons pas pu mapper les techniques associées dans les exécutables, mais nous avons présenté un compte rendu de cette technique qui pourrait justifier un tel comportement.

### 5.1 Fonctionnalité du Mode Basique de l'Outil

L'acquisition logique dans le cadre de la mobile forensics est une méthode utilisée pour extraire et analyser les données stockées sur un appareil mobile. Elle consiste à collecter les données de manière non intrusive et à un niveau plus superficiel que l'acquisition physique. Donc consiste à récupérer les informations directement accessibles depuis l'interface de l'appareil, telles que les contacts, les messages (SMS, MMS, chats), les photos, les vidéos, les historique de navigation, et d'autres fichiers accessibles via les interfaces de l'utilisateur. Dans notre scénario, l'application utilise des commandes ADB pour communiquer avec l'appareil Android. [colframe=black, colback=gray!10, title=Compte rendu sur ADB (Android Debug Bridge)] Android Debug Bridge (ADB) est un outil essentiel fourni avec le SDK Android qui permet de communiquer directement avec un appareil Android via un ordinateur. Fonctionnant comme un pont, ADB offre des fonctionnalités telles que l'installation ou la désinstallation d'applications, l'accès au système de fichiers, l'exécution de commandes via un terminal interactif (`adb shell`), et l'analyse des journaux système pour le débogage. Grâce à son architecture client-serveur, il transmet les commandes de l'utilisateur à un démon (`adbd`) exécuté sur l'appareil. Bien que puissant, ADB peut poser des risques si le mode débogage est activé sur un appareil non sécurisé, permettant potentiellement à un attaquant d'exploiter cet accès. Il reste néanmoins un outil incontournable pour

les développeurs et les chercheurs en sécurité, combinant contrôle total et flexibilité dans la gestion des appareils Android.

Sur Wireshark, nous avons remarqué le flux suivant de messages : Lorsque nous avons cliqué pour démarrer le processus d'acquisition logique avancée dans l'application, une requête a été envoyée à l'appareil cible, mise en évidence dans la figure 14.

```
.....! ...shell
:du -hc /data/*/
*/*/*cac he*.
```

Figure 14: La première requête

L'application tente ici d'estimer la taille des données du dossier *cache* en utilisant la commande *du*. L'appareil a répondu comme indiqué dans la figure 15. Le dossier *cache* n'existe pas.

```
.. *d_...
.....; ...du: /
data/*/* /*/*cach
e*: No s uch file
or dire ctory.0.
total.
```

Figure 15: La première réponse

Ensuite, la même requête a été envoyée depuis la station Cellebrite, mais cette fois pour des répertoires avec des chemins différents (voir figure 16).

Cette fois-ci, comme le montrent les figures 17 et 18, l'appareil a répondu avec la taille du dossier demandé par Cellebrite UFED.

Après cette phase de découverte, des requêtes ont été envoyées pour extraire chaque fichier lié au dossier contenant les photos de l'utilisateur.

```
..... shell  
:du -hc /sdcard/  
Android/ data/*/*  
cache* .
```

Figure 16: La 2<sup>ème</sup> requête

```
..... 5 ..... 1.8M.  
/sdcard/ Android/  
data/com .asus.ep  
hotoburs t/cache.
```

Figure 17: La 2<sup>ème</sup> réponse

```
..... D ... 12K · /  
sdcard/A ndroid/d  
ata/com. whatsapp  
/cache/g if/gif_c  
ache_mem _store ·
```

Figure 18: La 3<sup>ème</sup> réponse

## 5.2 Le Routine IsDebuggerPresent

Par défaut, l'exécution de l'exécutable principal dans un débogueur entraîne une erreur qui arrête le processus de débogage (voir figure 19). Les chaînes liées à cette boîte de message ont été trouvées en



Figure 19: Un débogueur est détecté

utilisant le même débogueur et sont présentées dans la figure 20.

Tout d'abord, nous avons contourné cette routine pour pouvoir analyser l'exécutable principal sous le débogueur `x32dbg`. Pour ce faire, nous avons utilisé le plugin `ScyllaHide`, comme illustré dans la figure 21.

Une fois l'accès accordé dans le débogueur, nous avons examiné la table des adresses d'importation (*Import Address Table*) du module. La fonction `IsDebuggerPresent` est effectivement invoquée pour vérifier la présence d'un débogueur. La fonction `IsDebuggerPresent` est une API Windows qui permet à une application de détecter si elle est exécutée sous un débogueur.

La fonction `IsDebuggerPresent`, définie dans la bibliothèque `Kernel32.dll`, renvoie une valeur booléenne : - `TRUE` (ou 1) si un débogueur est attaché au processus en cours. - `FALSE` (ou 0) si aucun débogueur n'est détecté.

La suivante est la signature de la fonction en C :

016FFD98	00CCBC00	innerloader.00CCBC00
016FFD9C	016FFDC4	
016FFDA0	00F78D26	return to innerloader.00F78D26 from ???
016FFDA4	2EE74AFA	
016FFDA8	018F75C0	L"A debugger is running"
016FFDAC	00000001	
016FFDB0	00000000	
016FFDB4	00C76000	innerloader.00C76000
016FFDB8	018F75C0	L"A debugger is running"
016FFDBC	00000001	

Figure 20: Chaînes de caractères

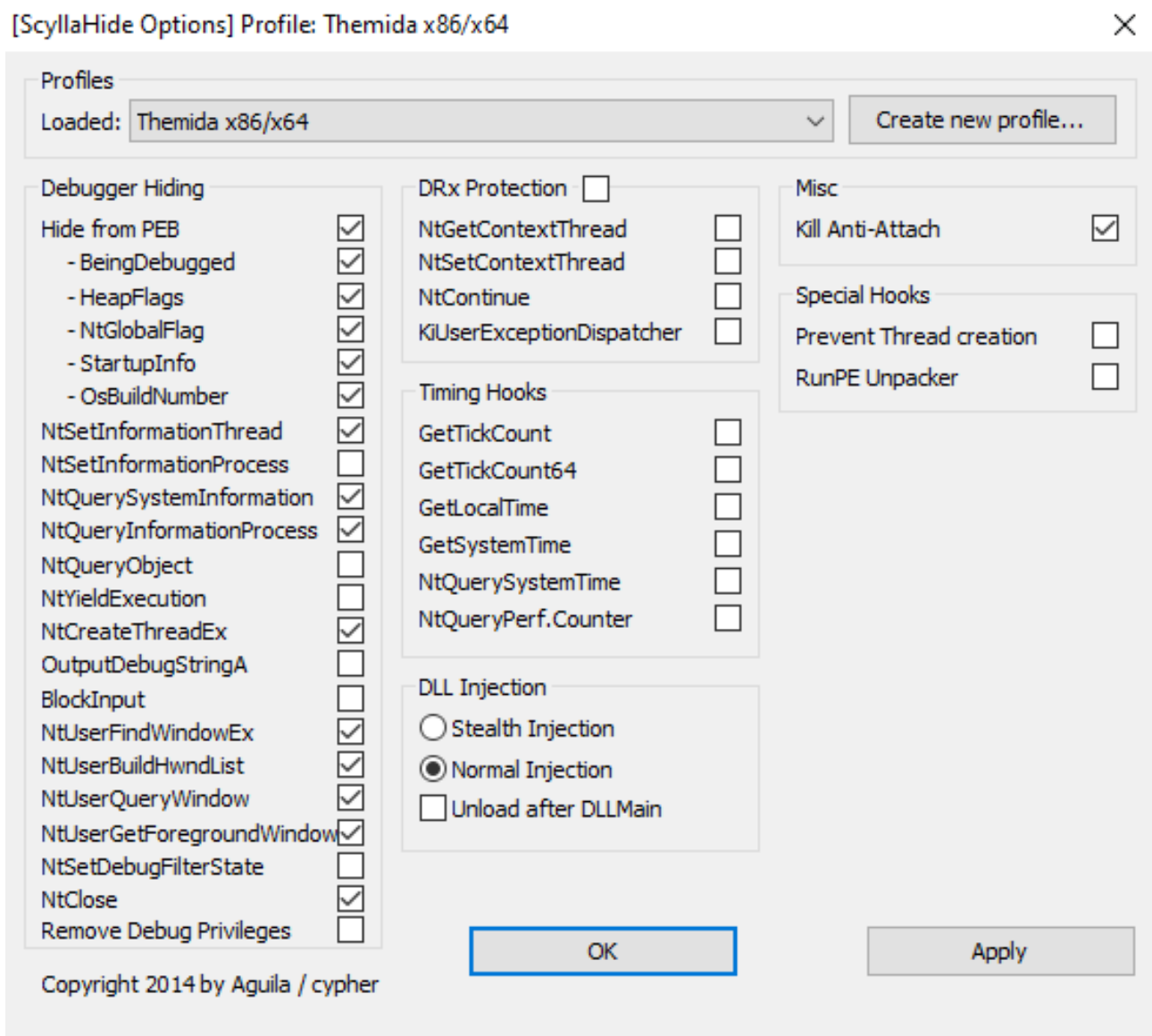


Figure 21: Configuration du plugin

BOOL IsDebuggerPresent(void); [colframe=black, colback=gray!10, title=Understanding IsDebuggerPresent]  
La fonction `IsDebuggerPresent` est une API Windows qui permet à une application de détecter si un débogueur est attaché au processus en cours d'exécution. Voici une explication narrative détaillée de son fonctionnement.

- **Structure interrogée :** La fonction vérifie une structure interne de Windows appelée `Process Environment Block` (PEB). Cette structure contient des informations sur le processus, y compris un indicateur nommé `BeingDebugged`.
- **Accès au PEB :** Le PEB est accessible via la structure `Thread Environment Block` (TEB), qui est spécifique à chaque thread. Dans les systèmes x86, l'adresse du PEB est stockée dans le registre FS.
- **Vérification du drapeau `BeingDebugged` :** Une fois l'adresse du PEB obtenue, `IsDebuggerPresent` lit le drapeau `BeingDebugged`, qui est un simple indicateur booléen situé à un offset fixe (0x2). Si ce drapeau est défini sur 1, cela signifie qu'un débogueur est attaché ; sinon, il est défini sur 0.

Lorsqu'un débogueur comme `x32dbg` ou `OllyDbg` attache un processus, le noyau de Windows met automatiquement à jour le drapeau `BeingDebugged` dans le PEB pour signaler qu'un débogueur est présent. `IsDebuggerPresent` exploite cette mise à jour automatique pour effectuer sa détection. Bien que cette méthode soit simple, elle est également vulnérable à plusieurs techniques de contournement :

- **Modification directe du PEB :** Un attaquant peut modifier directement la valeur du drapeau `BeingDebugged` dans le PEB à l'aide d'un éditeur de mémoire ou d'un outil comme `ScyllaHide`, ce qui rend `IsDebuggerPresent` inefficace.
- **Détournement (*hooking*) de la fonction :** Il est possible de détourner la fonction `IsDebuggerPresent` pour la forcer à toujours retourner `FALSE`, quel que soit l'état réel.
- **Utilisation de débogueurs furtifs :** Certains outils, comme les plugins de dissimulation (`ScyllaHide`), masquent la présence du débogueur en manipulant les structures internes de Windows.

### 5.3 Techniques Anti-Virtualisation

L'anti-virtualisation est une technique utilisée par certains logiciels malveillants pour détecter qu'ils s'exécutent dans un environnement virtuel, tel qu'une machine virtuelle (VM) ou une sandbox, plutôt que sur une machine physique réelle. Cette pratique est largement adoptée par les créateurs de malwares afin d'échapper aux analyses et aux tests effectués par des chercheurs ou des systèmes de cybersécurité. En effet, les environnements virtualisés sont souvent utilisés pour isoler et étudier les comportements malveillants en toute sécurité. Lors de l'installation de l'outil Cellebrite UFED dans une machine virtuelle de VirtualBox qui exécute Windows 7, nous avons été confrontés à une erreur indiquant que le logiciel ne peut pas être exécuté dans un environnement virtualisé. Ce message explicite, 'Please do not run this application under a virtual machine', révèle que Cellebrite intègre des mécanismes d'anti-virtualisation pour empêcher son utilisation dans une machine virtuelle. Cela indique que le logiciel est conçu pour détecter automatiquement les environnements non physiques et refuse de s'exécuter dans de tels cas. Cette stratégie vise à renforcer la sécurité et la protection du logiciel en empêchant son contournement ou son analyse dans un environnement contrôlé, comme une VM, qui est souvent utilisé pour le reverse engineering ou l'exploitation non autorisée. Par conséquent, ce mécanisme d'anti-virtualisation démontre l'efficacité des protections intégrées dans Cellebrite pour garantir un usage exclusif sur des machines physiques. Pour détecter l'environnement virtualisé, nous supposons que l'application pourrait effectuer l'une des vérifications suivantes :

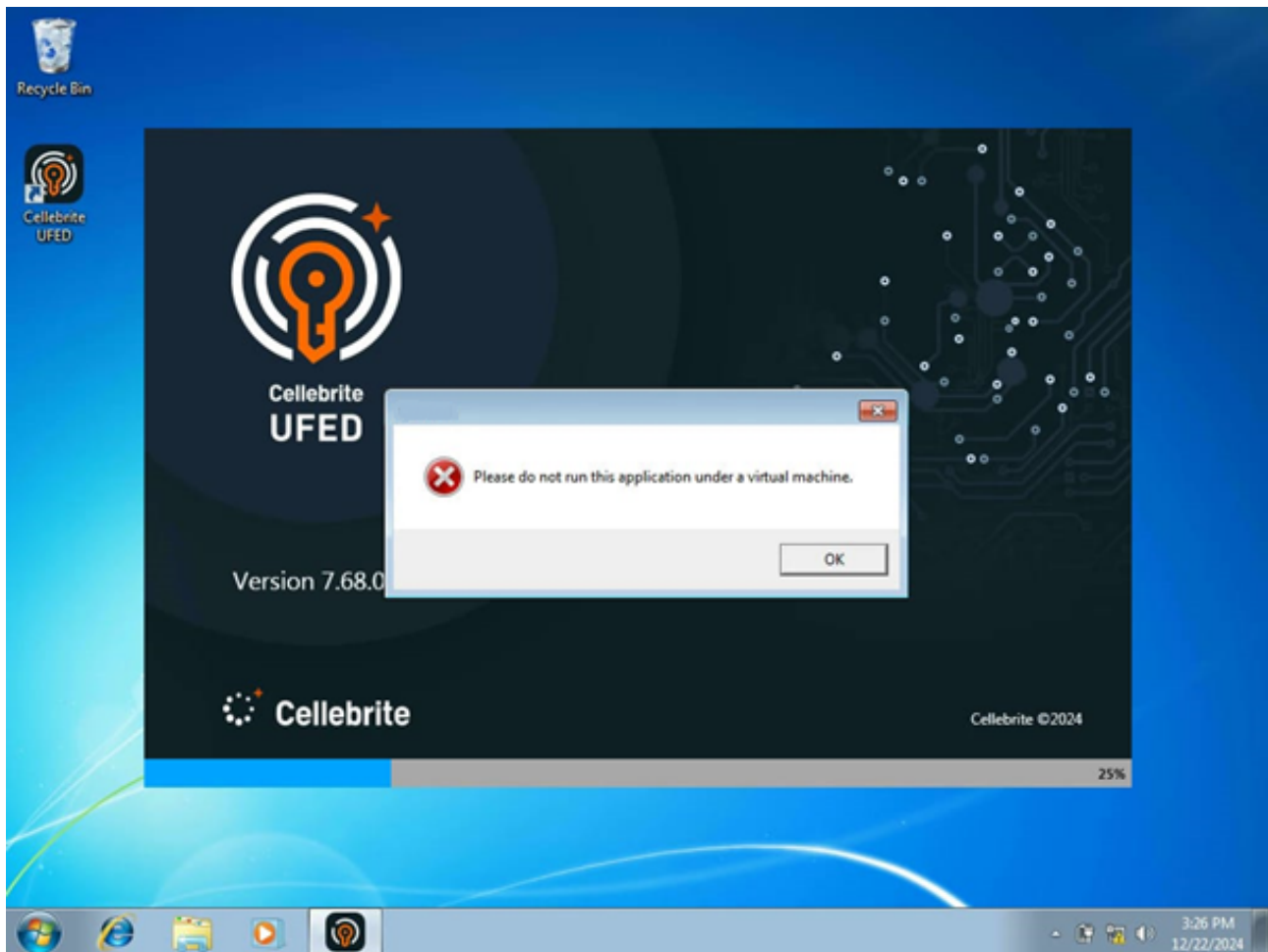


Figure 22: Detection d'une machine virtuelle résulte un crash

- Recherche de processus ou de services spécifiques associés aux environnements virtualisés (par exemple, `VBoxService.exe` ou `vmtoolsd.exe`).
- Vérification de la présence de pilotes ou d'adaptateurs réseau caractéristiques des machines virtuelles, comme `vmxnet3` ou `vboxnet`.
- Inspection des registres Windows pour identifier des clés spécifiques indiquant l'exécution dans un environnement virtualisé, telles que celles associées à VirtualBox ou VMware.
- Analyse des informations matérielles (comme les identifiants du processeur ou du BIOS) pour détecter des signatures d'émulation.
- Mesure des latences ou des anomalies dans le comportement matériel qui sont caractéristiques d'une virtualisation.
- **Pilotes de périphériques** Cellebrite peut inspecter les pilotes système pour identifier des signatures caractéristiques d'environnements virtuels, comme ceux utilisés par VirtualBox. Ces pilotes sont souvent présents uniquement dans des machines virtuelles et permettent au logiciel de confirmer qu'il ne s'agit pas d'un environnement physique. Les VM n'utilisent généralement pas de périphériques physiques tels que des souris, claviers ou disques externes connectés directement. Cette absence est un indicateur fort pour un logiciel cherchant à déterminer si l'environnement est virtualisé.

- **Caractéristiques du matériel** Les machines virtuelles ont souvent des configurations spécifiques et standardisées, comme des noms de processeur ("VMware CPU"), des tailles de mémoire RAM fixes, ou des résolutions d'écran par défaut. Cellebrite peut exploiter ces informations pour détecter un environnement non physique.
- **Inspection des processus en cours** Étant donné que les noms des processus utilisés par les applications de virtualisation sont connus, l'application vérifie la présence de ces processus pour déterminer si elle a été lancée par l'un d'entre eux.
- **Time-based** L'application peut également vérifier les délais temporels qui pourraient indiquer un environnement virtualisé.

À ce stade, nous n'avons pas été en mesure d'identifier précisément la vérification utilisée.

## 6 Discussion

### 6.1 Implications Sécuritaires des Résultats

Signal a affirmé que le problème réside dans les fonctions d'analyse utilisées par Cellebrite, en particulier les DLL qui traitent une variété de formats utilisés par de nombreuses applications. Autrement dit, les données que le logiciel de Cellebrite doit extraire et afficher sont générées et contrôlées par les applications sur l'appareil, et non par une source "fiable". Par conséquent, Cellebrite ne peut pas faire d'hypothèses sur la "validité" des données formatées qu'il reçoit. C'est dans cet espace que pratiquement toutes les vulnérabilités de sécurité prennent naissance.

"Puisque presque tout le code de Cellebrite existe pour analyser des entrées non fiables qui pourraient être formatées de manière inattendue pour exploiter des corruptions de mémoire ou d'autres vulnérabilités dans le logiciel d'analyse, on pourrait s'attendre à ce que Cellebrite ait été extrêmement prudent. En examinant à la fois UFED et Physical Analyzer," a affirmé Signal, "nous avons été surpris de constater qu'une très faible attention semble avoir été accordée à la sécurité du logiciel de Cellebrite. Les défenses d'atténuation des exploits, conformes aux normes de l'industrie, sont absentes, et de nombreuses opportunités d'exploitation sont présentes."

FFmpeg est un projet de logiciel libre et open source comprenant une suite de bibliothèques et de programmes pour la gestion des fichiers multimédia tels que la vidéo, l'audio et d'autres flux multimédia. Cellebrite utilise les DLL de FFmpeg qui ont été construites en 2012 et n'ont pas été mises à jour depuis. Plus d'une centaine de mises à jour de sécurité ont été publiées depuis lors, aucune n'ayant été appliquée.<sup>4</sup> L'équipe de Signal affirme avoir découvert qu'il est possible d'exécuter du code arbitraire sur une machine Cellebrite simplement en incluant un fichier spécialement formaté mais autrement inoffensif dans n'importe quelle application sur un appareil qui est ensuite branché à Cellebrite et scanné. Il n'y a pratiquement aucune limite au code qui peut être exécuté.

Avec cette attaque, **il est possible d'exécuter un code qui modifie non seulement le rapport Cellebrite en cours de création lors de ce scan, mais aussi tous les rapports précédemment et ultérieurement générés par Cellebrite pour tous les appareils déjà scannés et à venir, de manière arbitraire (en insérant ou supprimant du texte, des emails, des photos, des contacts, des fichiers ou toute autre donnée), sans modification détectable des horodatages ou des sommes de contrôle. Cela pourrait même être fait de manière aléatoire et remettrait sérieusement en question l'intégrité des données des rapports de Cellebrite.**

<sup>4</sup><https://www.cvedetails.com/product/6315/?q=ffmpeg>



### Vulnerabilities By Year

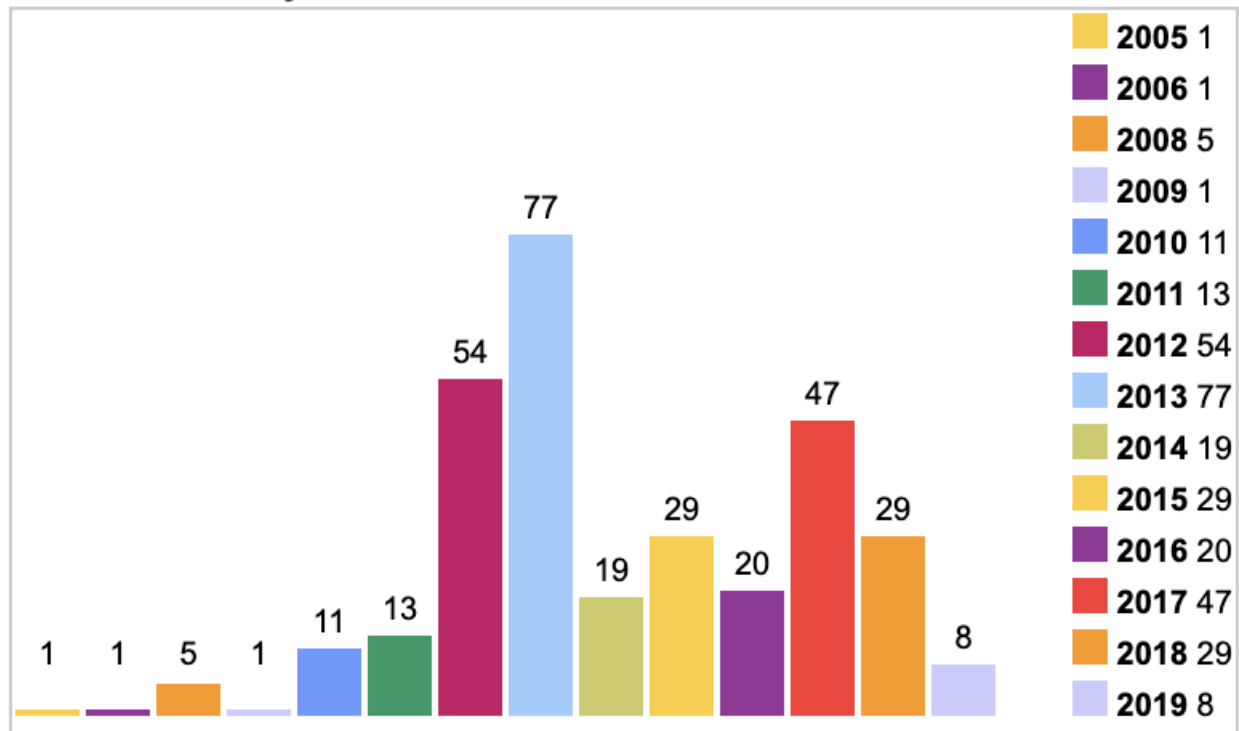


Figure 23: Vulnérabilités dans FFmpeg

Ils ont annoncé qu'ils mettraient à jour leurs utilisateurs avec des fichiers difficiles à prédire qui seraient analysés par l'outil forensic, et immédiatement après la mise à jour suivante de Cellebrite UFED, la fonctionnalité d'extraction de données liées à l'application Signal a été supprimée.<sup>5</sup>

## 6.2 Pistes Futures pour l'Analyse

Il est nécessaire d'être équipé des compétences et des connaissances nécessaires, qu'elles soient générales ou spécifiques au packer. Cela inclut des méthodes pour contourner les mécanismes de l'unpacker et étudier le fonctionnement interne de l'application. Ensuite, une analyse complète peut être menée.

## 7 Conclusion

Le packing des logiciels présente des avantages et des inconvénients. Vous verrouillez l'application pour empêcher quiconque de la comprendre, mais vous vous éloignez également de potentielles opportunités. L'opportunité de permettre à des chercheurs d'étudier la sécurité de votre logiciel et d'offrir à vos utilisateurs un produit de confiance.

<sup>5</sup>[https://www.reddit.com/r/signal/comments/mvjm82/exploiting\\_vulnerabilities\\_in\\_cellebrite\\_ufed\\_and/](https://www.reddit.com/r/signal/comments/mvjm82/exploiting_vulnerabilities_in_cellebrite_ufed_and/)