Zed-Files : Aux frontières du réel

Camille Mougey
prenom.nom@ssi.gouv.fr

ANSSI

Résumé. La suite logicielle de l'éditeur Prim'X, comprenant les archives `Zed!`, le chiffrement de fichiers (locaux et en réseau) `ZoneCentral` et le chiffrement de volume `Cryhod`, est un des moyens – le seul accessible à tous – pour les entreprises et le public souhaitant envoyer et stocker de l'information dîte « Diffusion Restreinte » (DR). Elle est donc critique dans l'écosystème français, car déployée dans les entreprises sensibles (toutes celles ayant à traiter de l'information DR) tout en représentant une surface d'attaque externe (au moins pour les archives `Zed!` reçues de l'extérieur). Une étude de ces logiciels, en particulier `Zed!` et `ZoneCentral`, a donc été menée, sans connaissances préalables de la solution. Cet article aborde la méthode et les outils utilisés pour en comprendre le fonctionnement, ainsi que les vulnérabilités qui y ont été trouvées et leurs corrections.

1 Introduction

Afin de comprendre l'intérêt et les impacts de l'étude des logiciels `Zed!` et `ZoneCentral` (produits par Prim'X), des notions associées au traitement d'information « Diffusion Restreinte » et au délivrement d'agrément par l'ANSSI sont nécessaires. Si certains professionnels français traitent quotidiennement avec ces notions, elles peuvent être assez étrangères au plus grand nombre. Cette première partie vise ainsi à en rappeler les points clés.[1]

1.1 La mention « Diffusion Restreinte »

La mention « Diffusion Restreinte » (DR) n'est pas un niveau de classification et ne relève donc pas du code pénal, contrairement à son équivalent dans d'autres pays. Elle a pour but d'inciter les utilisateurs de ces informations à être discrets dans leur gestion. Elle indique aussi que le cumul de ces informations peut mener à des informations qui seraient elles couvertes par le secret de la défense nationale.

[1] Le lecteur intéressé pourra approfondir le sujet en partant de la page dédiée sur le site de l'ANSSI : `https://cyber.gouv.fr/sinformer-sur-la-reglementation`

Les informations portant la mention DR sont réglementées par l'« Instruction interministérielle relative à la protection des systèmes d'informations sensibles » [14] (II 901 pour les intimes).

En particulier, l'II 901 décrit, dans son article 14 « Traitement des informations portant la mention Diffusion Restreinte », comment ces informations DR doivent transiter et être stockées :

Les informations portant la mention Diffusion Restreinte sont chiffrées à l'aide de moyens agréés à ce niveau par l'ANSSI dès lors qu'elles transitent ou sont stockées en dehors d'une zone physiquement protégée dans les conditions prévues à l' article 15.

Ainsi, une information DR peut-être envoyée par e-mail, dès lors qu'elle est chiffrée par un moyen agréé adapté.

1.2 L'agrément de solution par l'ANSSI

L'ANSSI propose 3 modèles :

— la « Certification » : elle est obtenue pour une cible de sécurité donnée, suite à une évaluation de type « Critères Communs » (CC, internationale) ou une analyse de plusieurs jours par un centre d'évaluation (délivrant alors une « CSPN », spécificité française). Le choix du centre d'évaluation et son paiement sont faits par le client, mais la liste est restreinte à des centres reconnus pour un ensemble de compétence donné (car évalués par des prestataires eux-même reconnus par l'ANSSI), appelés CESTI ;

— la « Qualification » : l'objectif est de recommander le produit pour un usage donné. Il est évalué avec pour objectif le respect d'un certain nombre de fonction de sécurité, validé par l'ANSSI, pour un certain niveau considéré d'attaquant (« élémentaire » < « standard » < « renforcé »). Elle a une visée réglementaire et inclut des engagements de son fournisseur sur le long terme ;

— l' « Agrément » : permet à une solution d'être utilisée dans certains cadres réglementaires. Elle est donnée suite à une décision discrétionnaire de l'ANSSI.

Ce système est prévu pour faire bénéficier à tous des travaux effectués sur une solution (dans le cas d'une réussite ; si un produit ne passe pas une évaluation, il n'en n'est pas fait publicité). Il permet aussi une visibilité sur les incidents de sécurité qui y sont liés. Néanmoins, il peut créer des situations de quasi-monopole – si un seul produit est qualifié pour un usage, il doit réglementairement être utilisé pour cet usage.

Visa de sécurité ANSSI Les solutions, services et centres d'évaluation certifiés, qualifiés ou agréés peuvent depuis 2018 utiliser la marque « Visa de sécurité ANSSI » [8].

Si cette marque est pensée pour faciliter la valorisation et le choix parmis les solutions concurrentes du marché, le lecteur attentif notera :

— **qu'elle n'est en aucun cas une garantie de sécurité absolue**. Elle indique que des moyens (analyse par un laboratoire, revue des spécifications, etc.) ont été mis en œuvre pour en jauger le niveau de sécurité, lié à un niveau d'attaquant considéré (suivant la certification ou le niveau de la qualification) ;
— dans le cadre d'une certification, que ces moyens ont été mis en œuvre pour une cible donnée, et non l'ensemble des cas possibles.

Ainsi [2] :

— le visa de sécurité de l'agent de l'EDR HarfangLab [3] indique qu'il est considéré comme robuste contre une utilisation à des fins d'élévation de privilèges locale (LPE) sur le poste où il est déployé. Cependant, il ne fournit aucune information sur sa capacité à détecter et bloquer une attaque (c'est-à-dire sa fonction d'EDR) ;
— la certification associée au visa de sécurité de la passerelle d'échange CrossinG [4] prend pour hypothèse un réseau d'administration dédié et sûr. Le travail réalisé par le CESTI ne considère donc pas cette surface d'attaque, le visa ne donne donc pas d'information sur la robustesse de l'interface d'administration (Web) de cette passerelle. Il appartient à l'utilisateur de la solution de considérer ou non cette hypothèse comme réaliste au sein de son réseau, et de prendre des mesures en conséquence ;
— le visa de sécurité du produit TixeoServer [5] ne permet pas de l'utiliser pour faire transporter de l'information DR au sein d'un réseau de moindre confiance. Il peut être utilisé au sein d'un réseau DR, comme le peuvent l'être n'importe quel produit qui ne ferait pas transiter d'information hors de ce réseau.

Agrément au niveau DR Pour obtenir un agrément au niveau DR, une solution doit :

— répondre à un besoin des bénéficiaires de l'ANSSI, à savoir la possibilité de transporter de la donnée DR ;

[2] Ces exemples sont arbitraires et ont seulement été choisis pour leur aspects illustratifs
[3] `https://cyber.gouv.fr/produits-certifies/agent-edr-hurukai-v201`
[4] Référence du certificat : ANSSI-CSPN-2022/10
[5] `https://cyber.gouv.fr/produits-certifies/tixeoserver-version-15500`

— obtenir une qualification « standard » sur une cible validée par l'ANSSI. Dans ce cas, une CSPN n'est donc pas possible. Une certification de niveau minimum CC EAL3+ est donc nécessaire ;
— subir d'autres travaux spécifiques à cette fonctionnalitée.

L'agrément peut aussi être étendu à l'agrément « Restreint UE » si au moins deux analyses faites par des pays européens distincts sont disponibles.[6]

Il peut aussi être étendu à l'agrément « Restreint OTAN », délivré par le NIAPC.[7]

1.3 Importance de l'objet de l'étude

Au moment de l'écriture de cet article, seuls quelques logiciels sont agréés au niveau DR :

— `Zed!` (de Prim'X) : permet la création d'archive chiffrée ;
— `ZoneCentral` (de Prim'X) : permet de chiffrer des dossiers (locaux ou partagés) ;
— `Cryhod` (de Prim'X) : permet de chiffrer des volumes ;
— `Acid Cryptofiler` (du Ministère des armées) : permet aussi la création d'archive chiffrée.

Néanmoins, le logiciel `Acid Cryptofiler` n'est pas disponible pour le grand public. Il est restreint aux entreprises disposant de contrat avec le Ministère des armées.

Ainsi, `Zed!` est le seul logiciel accessible à toutes les entreprises et particuliers pour l'envoi d'information DR. Il est aussi agréé pour du « DR OTAN » et « UE Restricted ».

Il est donc :

— déployé chez la grande majorité, voire la totalité, des entreprises traitant d'information sensible au sens de la mention DR ;
— « exposé » aux informations provenant de canaux non sûrs tels que les e-mails, clés USB, etc. utilisés pour l'échange de ces informations.

Ces éléments en font une cible de choix pour un attaquant s'intéressant aux intérêts français.

La cible de sécurité associée à sa certification [7], une des bases sur laquelle son agrément a été donné, est principalement orientée sur la confidentialité des données qu'il chiffre. Si cet aspect a donc été regardé

[6] Agrément « Restreint UE » pour `Zed!` : `https://www.eumonitor.nl/9353000/1/j9vvik7m1c3gyxp/vkbudvsvn7zh`
[7] Agrément « Restreint OTAN » pour `Zed!` : `https://www.ia.nato.int/niapc/Product/Prim-X-Zed--6.1_470`

en particulier, une analyse de la sécurité du logiciel en tant que tel est aussi nécessaire.

La suite de ce document décrit l'approche utilisée à cette fin et les vulnérabilités identifiées. Afin d'être représentative, aucune information privée (comme les rapports des laboratoires sur le produit ou de la documentation interne) n'a été utilisée. Certaines observations resteront donc à l'état d'hypothèses.

2 Premières approches

Pour débuter l'analyse du produit, plusieurs stratégies ont été utilisées. Cette section décrit les prises initiales d'informations au travers de quelques approches peu coûteuses.

2.1 Éléments disponibles en ligne

Plusieurs sources disponible sur Internet permettent d'obtenir de premières informations sur le produit `Zed!` :

- les cibles de sécurité des critères communs EAL3+ du produit incluent des informations sur la cryptographie utilisée (AES-256 bits), les utilisations possibles (« secours utilisateur », etc.) ainsi que des éléments d'architecture ;
- quelques vulnérabilités ont déjà été remontées sur ce produit, en particulier la CVE-2018-16518 (CVSS : 8.3). Le bulletin officiel indique que « l'ouverture d'une archive `Zed!` peut créer des fichiers arbitraires sur l'hôte » ;
- le guide ANSSI « Recommandations pour une utilisation sécurisée de Zed ! » donne à travers des éléments de durcissement, quelques informations. Par exemple, on y apprends que les extensions de fichier peuvent être en clair dans l'archive ou que le chiffrement peut être réalisé à l'aide de mot de passe ou de certificats ;
- une analyse du format `Zed!` [10]. Même si cette analyse concerne une version antérieure du format, elle permet d'obtenir de nombreuses pistes :
  - le format d'une archive `Zed!` est basé sur le format MS Office,
  - une clée de chiffrement globale est utilisé pour « obscurcir » certaines parties,
  - les données sont présentes sous une forme de type TLV (*Type, Length, Value*),
  - un mode de chiffrement appelé « AES-CBC-STREAM » est présenté (il sera décrit plus loin dans cet article) ;

— un format `Zed!` et un utilitaire `zed2john.py` sont présents dans l'utilitaire *John-The-Ripper*. Ils permettent de comprendre comment sont dérivés les mots de passe pour la protection de l'archive, en implémentant les résultats de l'analyse citée ci-dessus :

An intermediary user key, a user IV and an integrity checksum will be derived from the user password, using the deprecated PKCS#12 method as described at rfc7292 appendix B.

Cette méthode utilise une source dépendante de l'utilisation (authentification – pour vérifier que le mot de passe saisie est celui attendu –, IV pour le chiffrement, mot de passe pour le chiffrement) accompagnée d'un vecteur d'initialisation. Le résultat est ensuite hashé de nombreuses fois.

2.2 Observations initiales

Lors de la première création d'une archive `Zed!`, l'utilitaire demande de créer *une clé privée*, sous la forme d'un couple identifiant / mot de passe (note : un certificat peut aussi être utilisé). Cette clé permettra à son propriétaire de déchiffrer des archives s'il est présent dans la « liste d'accès » de cette archive. C'est le cas par défaut s'il en est le créateur.

Quelques tests peu coûteux peuvent déjà être effectués.

Stockage des listes d'accès En utilisant l'utilitaire `ProcMon` de la suite *Sysinternals*, il est possible d'observer les accès aux fichiers lors des différentes opérations.

En particulier :

— lors de la création de la clé privée, puis lors de l'ouverture du logiciel ensuite, un fichier `NOM_UTILISATEUR.zaf` est créé puis accédé ;

— si tous les fichiers `.zaf` sont supprimés (ils existent en plusieurs exemplaires), le programme redemande de créer une clé privée.

Ainsi, le fichier `.zaf` correspond ou contient au moins la clé privée de l'utilisateur.

Format des listes d'accès En ouvrant le fichier de liste d'accès dans l'un des utilitaires de `Zed!` appelé « *Encrypted Zone Management* », nous observons que des informations sont affichées *avant* la saisie du mot de passe de l'utilisateur (voir figure 1).

Pourtant, la recherche de chaîne de caractère au sein du fichier `.zaf` ne donne rien. Le calcul de son entropie (à l'aide de l'utilitaire `binwalk -E`) donne un score très proche de 1, caractéristique d'une donnée qui est a minima compressée ou chiffrée.

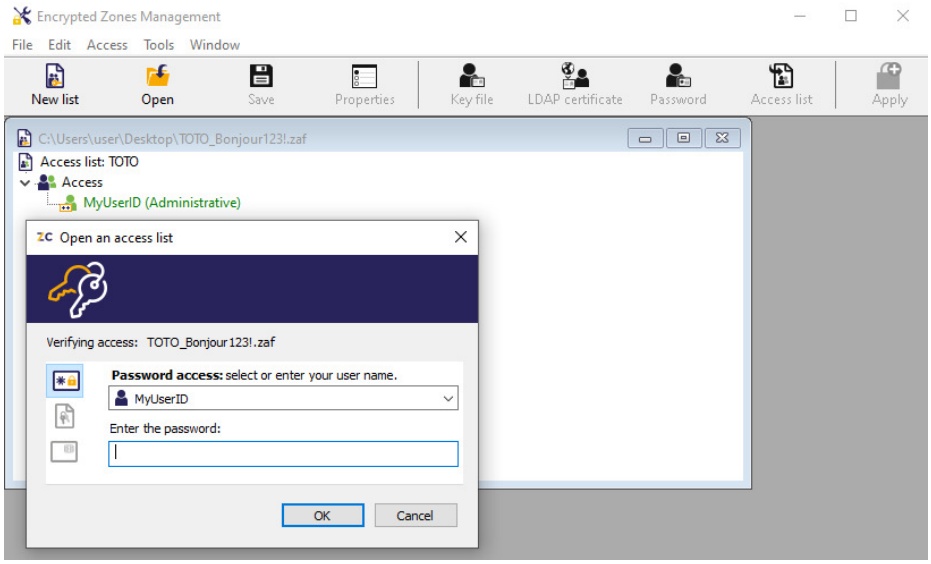


Fig. 1. Ouverture d'un fichier ZAF

Emport de la clé de déchiffrement Le test suivant est réalisé :

1. création du compte (donc d'un fichier `.zaf`) c_1 sur le poste p_1 ;
2. création d'une archive `Zed!` a_1 sur ce même poste ;
3. vérification : l'archive a_1 peut bien être déchiffrée sur p_1 avec le compte c_1 ;
4. sur un poste vierge p_2, ouverture de l'archive a_1 :
   — le compte c_1 est proposé pour le déchiffrement ;
   — en rentrant le mot de passe associée, l'archive est bien déchiffrée.

D'une manière ou d'une autre, l'archive `Zed!` emporte donc en son sein le moyen de déchiffrer l'archive à l'aide du compte du créateur, sans besoin de son fichier `.zaf`.

2.3 Analyse statique

`Zed!` est présent sur plusieurs plateformes et en plusieurs éditions (Free, Commercial, etc.).

La majorité des utilitaires sont écrits en C++ et sont assez volumineux. Même si des outils d'aide à l'analyse [9] existent, ce type d'analyse est réputé pour être notoirement fastidieux.

Néanmoins, sachant que de la cryptographie est utilisée, quelques éléments sont simples à obtenir :

— à l'aide de plugin IDA comme *FindCrypt*[8] et ses dérivés, de nombreuses fonctions de hashage sont retrouvées ;
— en cherchant les instructions d'accélération matérielle `AES-NI` permettrant de trouver une partie des fonctions implémentant AES ;
— en cherchant des instructions typiques de l'algorithme HMAC : `XOR xxx, 0x5c` et `XOR xxx, 0x36` ;
— etc.

2.4 Analyse dynamique

Afin de comprendre comment la donnée est compressée / chiffrée, une approche dynamique basée sur le flot de données a été utilisée.

À l'aide d'un debugger et de points d'arrêts *hardware*, il est possible de traquer le flot de données dans le sens croissant du temps.

Ainsi :

1. l'analyse démarre de la lecture de la donnée (API `ReadFile`). Ici, la donnée à forte entropie (qui démarre un peu après le début du fichier) est ciblée ;
2. à l'aide de points d'arrêts *hardware*, la donnée est suivie au gré des nombreuses copies (principalement dues aux créations et copies d'objets C++). Cette approche a une limite forte : le nombre de points d'arrêt simultanés est limité par l'architecture. Cette limitation peut-être contournée en utilisant une émulation type QEMU (sans accélération) ou via des hypothèses sur le trajet de la donnée (par exemple, « la donnée à un endroit *addr* de la mémoire ne sert plus après être passée comme source dans un `memcpy` ») ;
3. cette évolution est suivie en parallèle avec le résultat de l'analyse statique décrite dans la section précédente, grâce à des outils de synchronisation comme `ret-sync`.[9]

Après quelques copies, la donnée finit dans une fonction impliquant un grand nombre de calculs caractéristiques d'une fonction de décompression ou de déchiffrement. L'analyse de la pile d'appel permet d'identifier qu'il s'agit d'un sous-appel des fonctions AES identifiées lors de l'analyse statique.

Il reste donc à identifier :
— le mode, et suivant le mode, l'IV associé ;

[8] `https://hex-rays.com/blog/findcrypt2/`
[9] `https://github.com/bootleg/ret-sync`

— la clé (qui ne peut être considérée comme un secret, car à ce moment, aucun secret n'a encore été saisi) ;
— la manière d'effectuer le *padding*.

En identifiant les structures lues (entrées de la fonction) et écrites sans être relues (sorties de la fonction), nous retrouvons :

— en entrée : le contenu du fichier `.zaf` ;
— en sortie : de la donnée avec une faible entropie, contenant notamment les chaînes de caractères affichées à l'écran.

2.5 AES-CBC-ZED

L'analyse dynamique a permis de trouver l'endroit de l'appel des fonctions de déchiffrement, et l'analyse statique a permit d'identifier les primitives d'AES impliquées. Ainsi, le mode peut être reconstruit.

À la connaissance de l'auteur, le mode identifié n'est pas un mode standard d'AES. Il est illustré figure 2 et appelé « AES-CBC-ZED » dans cet article.

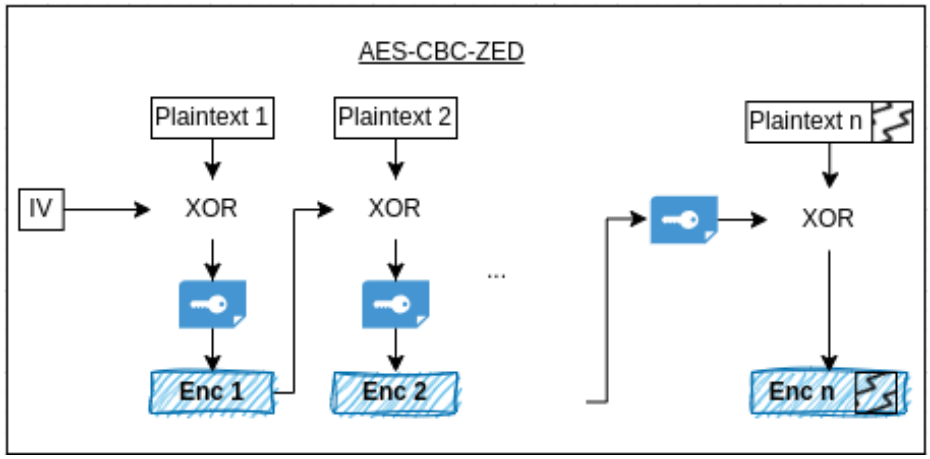


Fig. 2. Mode de chiffrement utilisé pour les fichiers ZAF

Quelques remarques sur ce mode de chiffrement :

— il ressemble à un mode CBC pour les $n-1$ premiers blocs puis un mode CFB pour le dernier bloc ;
— il permet de travailler sans *padding*, le chiffré faisant exactement la taille du clair ;

— à IV et clé identiques et si les $n-1$ premiers blocs sont aussi identiques, alors le `XOR` des blocs chiffrés n est le `XOR` des clairs correspondants ;
— en particulier, si le chiffré fait moins de 16 octets (la taille d'un bloc), le `XOR` des chiffrés est le `XOR` des clairs ;
— ce mode n'est pas authentifié.

3 Rétro-ingénierie des formats et algorithmes

Les premières approches présentées dans la section précédente permettent d'obtenir une partie des informations nécessaires à la compréhension du format et des algorithmes impliqués.

Néanmoins, les analyses statiques et dynamiques introduites plus haut montrent rapidement leurs limites pour le suivi efficace des flots de données. En particulier, il serait pertinent de pouvoir « remonter » les mutiples appels aux initialisations d'objets et à leurs copies pour trouver la source des éléments cryptographiques et la manière dont ils sont traités.

L'approche retenue ici par l'auteur est l'utilisation de *trace d'exécution*.

3.1 Analyse de trace d'exécution

Une trace d'exécution permet de s'intéresser à une seule exécution de l'application. Pour ce faire, cette exécution est instrumentée (par exemple via l'utilisation de l'API `ptrace` pour `rr` [2] sous Linux ou une exécution du programme en DBI – *Dynamic Binary Instrumentation* – pour `TTD` [3] sous Windows).

Elle peut ensuite être rejouée de nombreuses fois. Les adresses utilisées, l'ordonnancement des threads et les échanges avec le reste du système étant déterministes, il devient possible de « remonter », dans le temps d'exécution, l'écriture d'une donnée. Cette approche est donc souvent très efficace pour établir les flots de données au sein d'une application [17], même si cette dernière est obscurcie [11].

Pour cette étude, l'utilisation de TTD sous Windows a été retenue. Les traces d'exécution obtenues peuvent être analysées avec WinDBG ou instrumentées via un *binding* [15]. Plusieurs outils [6] [5] construits sur cette brique de base viennent agrémenter les analyses.

3.2 Format des fichiers ZAF

Déchiffrement En partant d'une trace TTD d'une ouverture d'un fichier `.zaf`, il devient possible d'identifier rapidement la clé utilisée en remontant

la création du contexte AES. Il s'agit d'une clé constante présente dans le binaire.

Le mode de chiffrement étant proche de CBC (voir section 2.4), il est possible de déchiffrer la majorité du contenu en utilisant un IV arbitraire. En effet, seuls les 16 premiers octets seront influencés par cet IV. Le contenu obtenu présente bien les chaînes de caractères affichées dans le logiciel (illustré figure 3).

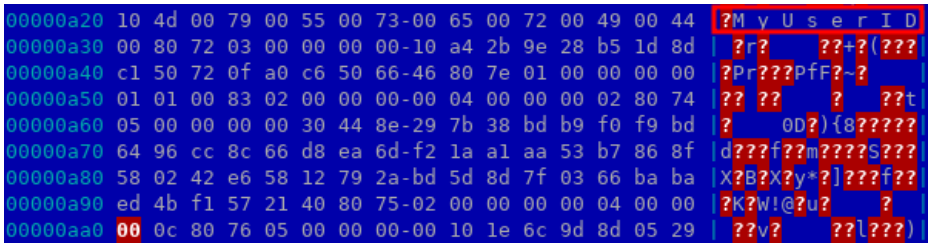


Fig. 3. Données déchiffrées des fichiers ZAF

En remontant ensuite la source de l'IV, la trace amène sur le retour de l'appel à la fonction `ReadFile`. L'IV est en effet présent au début du fichier ZAF.

En réimplémentant ces éléments, il devient possible de lire la donnée des fichiers `.zaf` (et, au besoin, de la réécrire).

Format sous-jacent Le lecteur attentif notera que les données déchiffrées sont structurées sous une forme de type TLV (*Type Length Value*). Ce format, similaire à ASN.1, est aussi récursif.

Données présentes Pour chacun des types rencontrés, une analyse a été réalisée pour en comprendre le contenu. Elles sont assistées d'une trace TTD de la création d'un fichier `.zaf`, permettant de remonter directement la source d'une donnée écrite dans le fichier de sortie.

Le format a ensuite été réimplémenté en Python (note : des *frameworks* comme Scapy ou Hachoir auraient pu être utilisés pour simplifier cette opération).

Ainsi, les informations suivantes sont directement lisibles dans le fichier, sans connaissance de secret préalable :

— le nom d'utilisateur, ses privilèges (pour la liste d'accès considérée), sa date d'ajout ;

— les emails de l'utilisateurs et leurs alias. Pour certains utilisateurs, cela inclut donc les e-mails utilisés pour l'anonymisation (« user123@domaine.fr ») ou les emails liés aux abonnements Office 365 (« ...@onmicrosoft.com ») ;
— le SID de l'utilisateur dans le domaine : `S-1-5-21-XXX-YYY-ZZZ-1234` ;
— le DN (*Distinguished Name*) de l'utilisateur : `CN=PrenomNOM,OU=Utilisateurs,DC=contoso,DC=com` ;
— si une PKI externe est utilisée, le certificat associé ;
— une clé publique RSA. *Spoiler alert* : elle sert à chiffrer les archives pour cette liste d'accès (ie. ce fichier `.zaf`).

3.3 Déchiffrement des secrets d'un fichier ZAF

Pour trouver la manière dont les secrets du fichier ZAF sont déchiffrés, une astuce est utilisée :

1. une trace d'exécution est enregistrée entre le moment où l'utilisateur entre son mot de passe et le moment où l'interface affiche que le fichier a été ouvert ;
2. en prenant pour hypothèse que la majorité des opérations réalisées sont liées au déchiffrement des secrets, il est possible d'aller au milieu de la trace (`!tt 50` sous WinDBG) et d'inspecter la pile d'appel.

Ainsi, le mot de passe entrée est dérivé suivant les algorithmes décrit dans la RFC 7292, *appendix-B : Deriving Keys and IVs from Passwords and Salt* :

— partie authentification (PBA) : 200 000 itérations d'un SHA-256. Le résultat est ensuite tronqué à 8 octets, puis comparé à une valeur présente dans le fichier ;
— si cette comparaison est bonne (« le mot de passe est celui attendu »), il est dérivé deux fois pour obtenir respectivement l'IV et la clé de déchiffrement (PBE). Ces dérivations sont faites en deux fois 100 000 itérations d'un SHA-256.

Cet IV et cette clé permettent alors de déchiffrer une clé globale au fichier `.zaf`. Cette clé globale, ainsi qu'un IV associé, permettent finalement de déchiffrer la clé privée RSA associée au fichier `.zaf`.

Cette manière de procéder étant très proche du format `zed` déjà implémenté dans *John-The-Ripper*, un format similaire a été implémenté pour les fichiers `.zaf`, comme illustré figure 4.

```
user@user-laptop:~/etudes/binaries/ZoneCentral$ python3.8 read_zaf.py --quiet --john TOTO_Bonjour123\!.zaf | tee /tmp/t
o_crack
MyUserID:$zed$1$22$200000$71fc1704150d6bdcbd94b3d3c7592c87$df0ff55ba8084781:::TOTO_Bonjour123!.zaf
user@user-laptop:~/etudes/binaries/ZoneCentral$ ~/tools/john/run/john /tmp/to_crack -w=/tmp/wordlist --skip-self-tests
Using default input encoding: UTF-8
Loaded 1 password hash (zed, Prim'X Zed! encrypted archives [PKCS#12 PBE (SHA1/SHA256) 256/256 AVX2 8x])
Cost 1 (iteration count) is 200000 for all loaded hashes
Cost 2 (hash-func [21:SHA1 22:SHA256]) is 22 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
Warning: Only 1 candidate buffered, minimum 32 needed for performance.
Bonjour123!      (MyUserID)
1g 0:00:00:00 DONE (2023-08-11 11:08) 7.692g/s 7.692p/s 7.692c/s 7.692C/s Bonjour123!
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Fig. 4. Cassage d'un mot de passe de fichier ZAF

Le nombre d'essais par seconde, grâce au nombre d'itérations requises, reste faible. Sur un CPU 4 cœurs i5-7200U, environ 200 mots de passe sont essayés par seconde.

Si le mot de passe recherché ne figure pas dans un dictionnaire, ou n'est pas obtenable par dérivation simple de ce dictionnaire, il est donc très peu probable qu'il soit récupérable.

Conséquences Le choix est laissé à l'utilisateur de changer son mot de passe. Au sein de certaines entités, cette règle est même forcée, de deux manières distinctes :

— via l'application d'une politique imposant une rotation des mots de passe tous les n mois. Cette politique peut provenir d'une interprétation pour les fichiers `.zaf` de la recommandation R10 du guide ANSSI « Recommandations pour une utilisation sécurisée de `Zed!` » prévue pour les archives `Zed!` :

> *[R10] Unicité et péremption des mots de passe :*
> *Utiliser un mot de passe différent pour chaque conversation. Changer le mot de passe régulièrement, idéalement tous les 3 mois.*

— via le processus de création des fichiers `.zaf` :

1. le fichier est créé par l'administrateur, avec un mot de passe par défaut connu de tous ;
2. dès sa récupération, l'utilisateur modifie son mot de passe.

L'analyse ci-dessus montre que le changement de mot de passe d'un utilisateur ne modifie pas le secret porté par le fichier `.zaf`, à savoir la clé privée RSA.

Ainsi, si un attaquant a un accès à plusieurs versions du même fichier `.zaf`, il lui suffira de trouver le mot de passe du plus faible d'entre eux pour compromettre les précédentes et futures utilisations de ce `.zaf`.

Applications Le fichier `.zaf` tiens donc lieu de clé publique (contenant une liste d'accès) et de clé privée (chiffrée avec le mot de passe de l'utilisateur). Chez certaines entités, un annuaire des utilisateurs – un partage réseau contenant l'ensemble des fichiers `.zaf` des utilisateurs – est disponible et accessible pour tous.

Lors d'un test d'intrusion, cet accès à l'annuaire des utilisateurs a permis d'utiliser une attaque par dictionnaire pour retrouver plusieurs mot de passe `.zaf`. Certains utilisateurs ayant tendance à réutiliser leurs mot de passe du domaine Active Directory (AD), cette méthode a permis d'obtenir plusieurs comptes valables au sein de cet AD, avec un très faible bruit.

Compte de secours À la création d'un fichier `.zaf`, un second utilisateur est ajouté automatiquement : `**SOS**`. Son mot de passe est généré aléatoirement et chiffré avec la clé globale du fichier `.zaf`, pour être récupérable par les autres utilisateurs présents dans le `.zaf`.

Il n'est pas affiché par l'interface graphique, même si celle-ci empêche l'ajout d'un utilisateur à ce même nom (illustré figure 5).

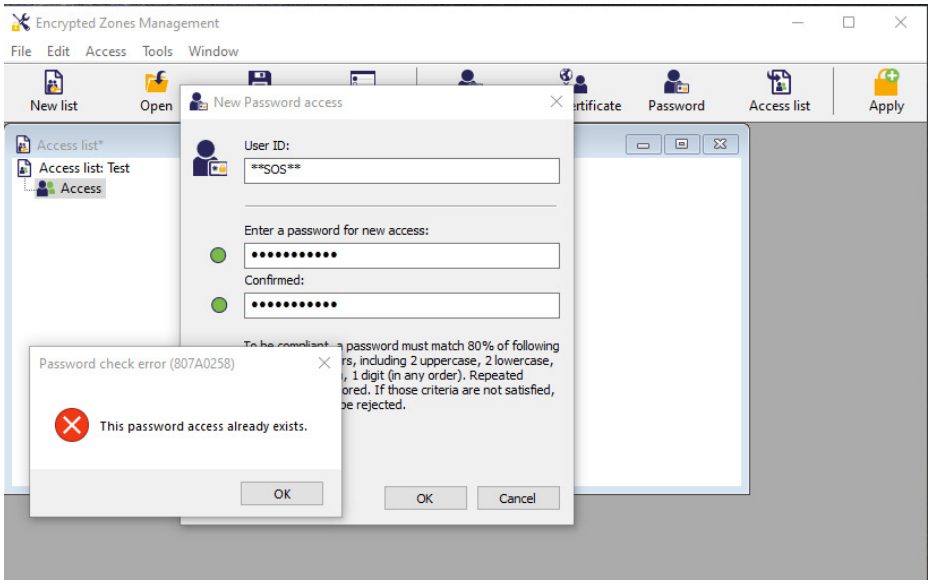


Fig. 5. Tentative d'ajout de l'utilisateur SOS

De la compréhension de l'auteur, ce compte est présent pour permettre à un administrateur qui serait par exemple présent dans l'ensemble

des fichiers `.zaf` de l'entreprise, d'obtenir un mot de passe de secours transmissible à l'utilisateur d'un `.zaf` ayant oublié le sien.

La figure 10 en annexe A résume le format des fichiers `.zaf`.

3.4 Format des fichiers Zed

Composition Comme les versions précédentes (voir section 2.1), les archives `Zed!` sont basées sur le format MS Office. Ce format est composé de *stream*, qui peuvent être hiérarchisés.

L'analyse des *streams* montre la présence :

— de méta-données (`_ctlfile`) liées aux fichiers de l'archive ;
— du contenu des fichiers, chiffrés ;
— suivant les versions, d'un *stream* `PGI-STREAM` ;
— un bloc a priori chiffré, nommé `_catalog` ;
— potentiellement, un fichier image en clair (pour l'image de fond liée à une archive, affichée dans l'interface graphique de `Zed!`) ;
— le fichier `.zaf` du propriétaire, expliquant le comportement observé section 2.2.

Vulnérabilité 1 *Une archive `Zed!` embarque le fichier de clé de son créateur.*

Conséquences *Envoyer une archive `Zed!`, c'est donc envoyer sa clé privée (protégée par son mot de passe) ainsi que ses informations de domaine AD, emails, etc.*

Un attaquant récupérant une archive `Zed!` d'un utilisateur peut essayer de retrouver le mot de passe de cet utilisateur. Si plusieurs archives sont disponibles, il aura à retrouver le mot de passe le plus faible parmis ceux utilisés.

S'il parvient à trouver le mot de passe, il peut :

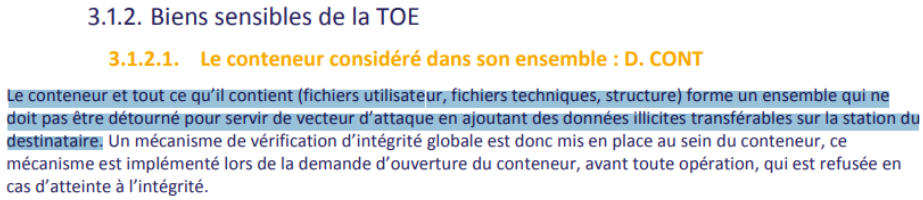
— *déchiffrer l'archive en question ;*
— *déchiffrer l'ensemble des archives passées et futures de l'utilisateur, auquelles il a accès.*

Autres informations Le *stream* `_catalog` est chiffré de la même manière que les fichiers `.zaf`. Il est donc possible de le lire sans connaissance de secret préalable. Il contient :

— le nombre, la taille et potentiellement l'extension de chacun des fichiers ;
— les clés publiques des destinataires.

Avec ces éléments, il est possible de créer une archive similaire, ouvrable par les destinataires d'origine, indistinguable en l'absence du secret de l'archive initiale, mais dont le contenu est contrôlé par l'attaquant.

Cette possibilité est à mettre en regard avec la cible de sécurité, notamment la section 3.1.2.1, reprise en figure 6.



3.1.2. Biens sensibles de la TOE

3.1.2.1. Le conteneur considéré dans son ensemble : D. CONT

Le conteneur et tout ce qu'il contient (fichiers utilisateur, fichiers techniques, structure) forme un ensemble qui ne doit pas être détourné pour servir de vecteur d'attaque en ajoutant des données illicites transférables sur la station du destinataire. Un mécanisme de vérification d'intégrité globale est donc mis en place au sein du conteneur, ce mécanisme est implémenté lors de la demande d'ouverture du conteneur, avant toute opération, qui est refusée en cas d'atteinte à l'intégrité.

Fig. 6. Extrait de la cible de sécurité

Remarque 1 *Une archive* `Zed!` *assure l'intégrité des données s'y trouvant mais pas leur authenticité.*

Le *stream* `_catalog` contient aussi :
— un extrait du numéro de license ;
— la version du logiciel utilisé, qui peut permettre de savoir que le créateur est vulnérable à CVE-2018-16518 ;
— la date de création et de dernière modification de l'archive ;
— le chemin complet de l'archive initiale.

Vulnérabilité 2 *Le chemin complet d'origine (*`X:\PartagesSecret\IncidentSSTIC\archive.zed`*) est présent dans l'archive* `Zed!`*, en clair.*

Conséquences *Le créateur peut révéler de l'information, potentiellement considérée secrète, dans le chemin de création de l'archive : nom de projet, date d'un incident, entité concernée, etc.*

Vérifications sur de vraies archives Pour vérifier les constats ci-dessus, des archives `Zed!` légitimes sont nécessaires. Afin de conserver une analyse faisable par une entité externe, le choix s'est porté sur VirusTotal.

En effet, sur ces dernières années, environ 500 archives `Zed!` ont été posées sur ce service. Pour rappel, certains abonnements à ce service permettent de récupérer l'ensemble des éléments mis en ligne. Normalement, la présence de ces archives en ligne, même s'il ne s'agit pas d'une bonne

pratique, ne devrait pas révéler d'information sensibles. Les archives `Zed!` sont utilisées pour cet objectif.

Quelques exemples des données disponibles sont repris ci-dessous (volontairement censurés).

Listing 1: Chemins d'archive

```
M:\str-hfds-planification\04 - Planification\VIGIPIRATE\POSTURE
↪ ETE - AUTOMNE 2023\...
... Strasbourg - Audit PASSI LPM ... 2023 - Procès Verbal de
↪ réception ...
C:\Users\...\Documents\GMR\_UKRAINE\GM60 ...
X:\...\Pole de compétence SYSTEMES DE COMBAT\...\3 - PROGRAMMES
↪ ...
H:\ANALYSES DE LA MENACE ET AUTRES DOCS ENVOYES AUX
↪ OPERATEURS\CAMPAGNE DE SIGNALEMENT ... 2021\...
```

Listing 2: Données utilisateurs et clés privées protégées

```
<SID: S-1-5-21-X-Y-Z-RID
  CN=...,OU=nom-a,OU=Centrale,OU=Personnes,OU=ADC,DC=sfer,DC=in,
    DC=adc,DC=education,DC=fr
  ...@sfer.in.ads.education.fr
>
<...
  Subject: O=Thales, CN=..., emailAddress=...@fr.thalesgroup.com,
  ↪ UID=...
>
```

Listing 3: Données contextuelles

```
CN = Requisition Judiciaire
Entreprise : ...
```

Chiffrement des fichiers dans l'archive En analysant les binaires `zed.exe`, deux modes de chiffrement sont identifiés, suivant la version utilisée :

— AES-CBC-CTS ;
— « AES-CBC-ZED ».

Le mode est indiqué dans les méta-données et, pour raison de rétro-compatibilité, les deux modes sont acceptés par les versions les plus récentes.

Comme décrit plus haut, le mode « AES-CBC-ZED » permet de modifier les octets du clair en modifiant les octets du chiffré, en particulier

pour les fichiers de moins de 16 octets. Des tests ont donc été réalisés pour tester cette attaque et ses potentielles contremesures.

Intégrité des archives Dans les archives récentes, le *stream* `PGI-STREAM` est présent pour aider aux tests d'intégrité. Ainsi, une modification d'un contenu d'un fichier de l'archive est détectée à l'ouverture, après entrée du secret.

Néanmoins, cette entrée étant apparue dans les versions plus récentes (aux alentours de 2021), les clients supportent son absence, sans afficher d'avertissement.

La suppression ou l'absence de ce *stream* permet une ouvreture d'une archive modifiée. Néanmoins, une erreur est affichée au moment du déchiffrement du fichier de l'archive.

Pour trouver le code responsable de cette partie, l'approche suivante a été utilisée :

1. deux traces TTD sont réalisées : l'une avec un fichier non modifié, s'ouvrant correctement. Le second, avec un fichier chiffré modifié, affichant une erreur à l'ouverture ;
2. en utilisant [15], les arbres des appels des fonctions et de leurs valeurs de retour sont comparés entre les deux traces.

Le raisonnement derrière cette approche est l'hypothèse que le *code path* changera au moment où la vérification d'intégrité échouera. Des ajustements doivent cependant être réalisés. En particulier, les allocations mémoires ou certaines recherches dans des structures (au sein d'une liste ou d'un arbre AVL) peuvent, pour la même « intention », générer une suite d'appels différents.

Un mécanisme de resynchronisation en sortie de ces fonctions particulières est utilisé. De plus, les appels réalisés dans certaines bibliothèques, comme celles du système, sont filtrés pour ne retenir que les logiques propres aux binaires de `Zed!`.

Cette approche a permis de montrer l'utilisation de HMAC pour l'intégrité des fichiers chiffrés, basé sur une clé liée aux secrets de l'archive et donc non usurpable.

Néanmoins, dans une approche similaire à *Drop-The-MIC*,[10] que se passe-t-il si le champ contenant le HMAC est lui-même supprimé ?

[10] Vulnérabilité CVE-2019-1040, où le MIC assurant l'intégrité des échanges NTLM est simplement supprimé

Remarque 2 *La suppression du HMAC de la section* `_seals` *rend inopérante la protection en intégrité des fichiers chiffrés ou de leur nom. Cependant, la protection en intégrité des sections elles-mêmes entraîne un message d'avertissement à l'ouverture de l'archive, reproduit figure 7.*

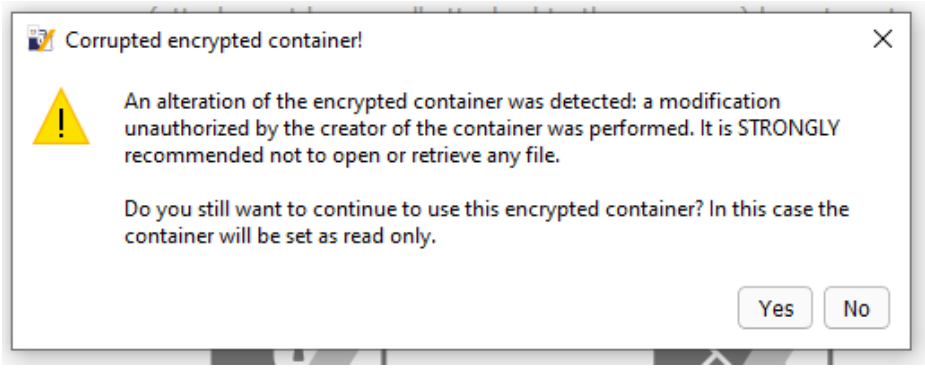


Fig. 7. Avertissement sur l'intégrité de l'archive à l'ouverture

Ce message d'avertissement n'est normalement jamais vu par les utilisateurs. De plus, il est par défaut autorisé à être accepté. Ainsi, un scénario de type *phishing* est envisageable. Suite à l'utilisation du mode « AES-CBC-ZED », si le clair du fichier chiffré fait moins de 16 octets (en réalité, si sa version compressée avec ZLIB fait moins de 16 octets), le contenu du fichier final est contrôlable ainsi que l'extension.

La figure 11 en annexe A résume le format des fichiers `.zed`.

3.5 Zed : champs inexpliqués

Les binaires de `Zed!` n'ont pas été analysés exhaustivement. Cependant, deux champs présents dans l'archive, par utilisateur, sautent aux yeux :

Listing 4: Contenu des blocs

```
- Unknown1 :
0000 0c 5f af d3 e2 d6 be 8a 73 32 88 61 c6 b9 79 2a
[...]
0170 53 c8 f9 f9 a3 10 d4 7b 8d ba 42 88 3b 1d b4 11

- Unknown2 :
0000 0c 79 4d 37 1a 85 71 1b 75 d7 c0 ab 52 bb ba 9c
0010 0c 79 4d 37 1a 85 71 1b 75 d7 c0 ab 52 bb ba 9c
[...]
0160 0c 79 4d 37 1a 85 71 1b 75 d7 c0 ab 52 bb ba 9c
0170 0c 79 4d 37 1a 85 71 1b 75 d7 c0 ab 52 bb ba 9c
```

Il est en effet très inhabituel de voir des blocs de 0x10 octets répétés. De plus, le premier bloc a une forte entropie.

Analyse des blocs À l'aide de l'analyse de la trace d'exécution lors de la création d'une archive, la création du contenu de ces blocs est retrouvée. Elle est résumé dans la figure 8 et s'articule ainsi :

1. un bloc d'aléa de 0x10 octet est créé ;
2. il est répété jusqu'à faire la taille d'un module RSA ;
3. il est copié dans l'archive ;
4. il est chiffré, en AES-CBC, avec un IV à zéro et la clé de l'archive ;
5. ce résultat est ensuite chiffré en RSA, PKCSv1.5 ;
6. ce résultat est ensuite chiffré en AES-CBC, avec un IV à zéro et la clé de l'archive ;
7. ce résultat est ajouté à l'archive.

Ainsi, seul un connaisseur du secret de l'archive peut produire le second bloc à partir du premier et vérifier que ce bloc a bien été créé selon ce processus. La clé privée RSA n'intervient à aucun moment dans le processus, le bloc à chiffrer par la clé publique étant connu. Néanmoins, la clé publique est liée à ce processus.

La seule hypothèse formulée par l'auteur est que ce bloc pourrait être lié à des mécanismes pour vérifier la connaissance de secret lors de rechiffrement de l'archive (renouvellement de clé), mais d'autres mécanismes plus simple aurait permis d'obtenir le même résultat.

Source de l'aléa La trace permet de remonter dans le code produisant l'aléa. Assez vite, l'analyse montre que ce code manipule des données du type :

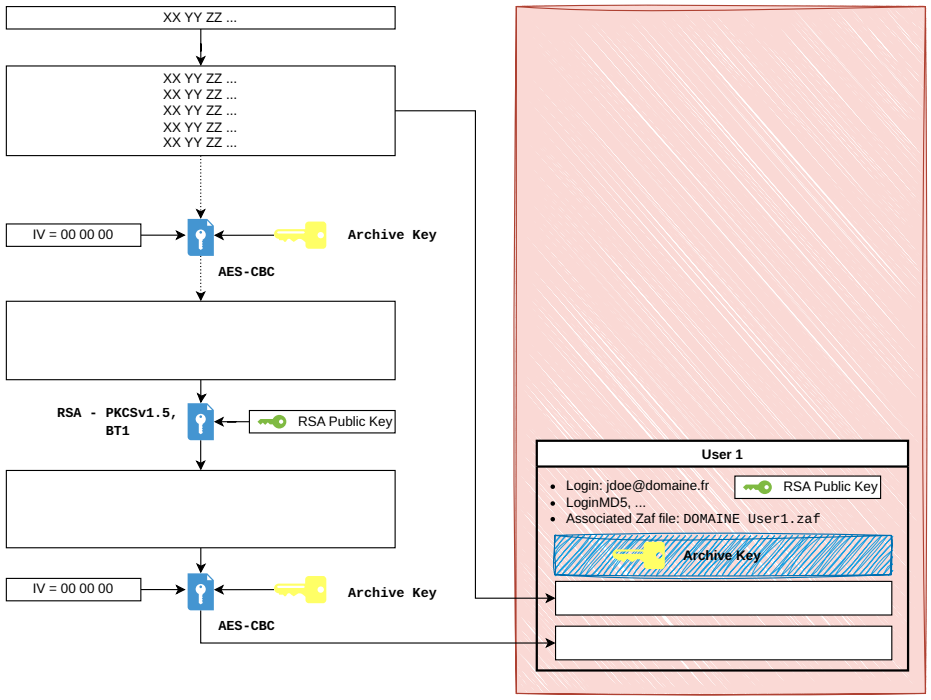


Fig. 8. Résumé de la création des champs spéciaux

Listing 5: Exemples de données manipulées

```
[01 00 00 00] [05 00 00 00] [01 00 00 00]

[05 00 00 00] [05 00 00 00] [AF 9F FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF]
```

Ces structures données font penser à des implémentations de manipulations de grands nombres (*BigNum*), comprenant la taille actuelle, la taille allouée et la donnée des nombres.

Afin de comprendre les algorithmes impliqués, il est donc nécessaire de comprendre les opérations effectuées par les fonctions concernant traitant *BigNum*.

Analyse des fonctions de BigNum L'approche utilisée est basée sur l'émulation [12] [4] des différentes fonctions.

Pour chacune des fonctions (trouvées grâces aux graphes d'appels et à leur emplacement dans le binaire) :

1. une analyse manuelle rapide du flot de donnée est faite pour trouver les entrées / sorties de ces fonctions (note : cela pourrait être automatisé en se basant sur les analyses et propagation de type déjà réalisée par IDA) ;
2. un ensemble d'entrée est généré ;
3. la fonction est exécutée pour chacune de ces entrées ;
4. les résultats sont comparés avec une liste d'opération communes : addition, soustraction, soustraction avec retenue, division, division modulaire, etc.

Cette manière de faire est similaire et se base sur les mêmes éléments que l'outil Sibyl [16].

Cette approche a ensuite été améliorée en utilisant de l'exécution concolique, avec Miasm [12] (mais elle aurait pu être réalisée avec d'autres outils [13]). La base de la méthode reste la même, mais les modifications suivantes sont apportées :

— les parties concernants les valeurs des nombres en entrées sont rendues « symbolique » ;
— à la fin de l'exécution, les valeurs de retour sont donc elles aussi symboliques, et contiennent directement l'équation du résultat suivant les entrées ;
— une seule exécution est donc nécéssaire par fonction, sauf si plusieurs chemin doivent être explorés. La connaissance des chemins à explorée peut être obtenue en analysant les décisions de branchement lors de l'exécution concolique (ie. dépendent-elles de la donnée ?).

Ci-dessous, quelques exemples de résultats d'exécution :

Listing 6: Émulation de la fonction de mise au carré

```
# INPUT :
  NUM 1 : 0x3, NUM 2 : 0x2
# OUTPUT:
  @32[NUM2.DATA] = (NUM1.zeroExtend(128) *
  ↪  NUM1.zeroExtend(128))[0:32]
  NUM 1:
    01 00 00 00 01 00 00 00 03 00 00 00
  NUM 2:
    01 00 00 00 01 00 00 00 09 00 00 00
```

Listing 7: Émulation de la fonction de décalage à gauche

```
# INPUT :
  NUM 1 : 0x3, NUM 2 : 0x2
# OUTPUT:
  @32[NUM3.DATA] = {0x0 0 2, NUM1 2 10, 0x0 10 32}
  NUM 1:
    01 00 00 00 01 00 00 00 03 00 00 00
  NUM 2:
    01 00 00 00 01 00 00 00 02 00 00 00
  NUM 3:
    01 00 00 00 01 00 00 00 0c 00 00 00
```

Reconstruction d'algorithme Une fois les fonctions de manipulations des *BigNum* identifiées, il est possible de réaliser une exécution symbolique des algorithmes qui les appellents. Durant cette exécution, les appels aux fonctions de manipulations sont remplacées par leur équivalent sémantique. Par exemple, l'appel à la fonction `big_num::square` est remplacé par `ARG1 * ARG1`. Cela peut amener à une faible perte de précision, mais permet de dégager l'algorithme général pour essayer de l'identifier.

La figure 9 est un exemple d'une telle reconstruction.

```
list_num::square(point->num2, tmp[2]);          // t2 = a2_2 ** 2
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[1]);// t1 = t2 % mod
bignum::mul(tmp[1], mod->g_mult_to_use, tmp[2]);// t2 = t1 * g_mult
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[3]);// t3 = t2 % mod
list_num::square(point->num1, tmp[2]);          // t2 = a2_1 ** 2
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[4]);// t4 = t2 % mod
bignum::mul(tmp[1], point->num2, tmp[2]);       // t2 = t1 * a2_2
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[5]);// t5 = t2 % mod
bignum::mul(tmp[5], mod->g_second_mult_to_use, tmp[2]);// t2 = t5 * g_second_mult
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[1]);// t1 = t2 % mod
bignum::sub_mod(tmp[4], tmp[3], mod->g_modulus, tmp[5]);// t5 = t4 - t3 % mod
list_num::square(tmp[5], tmp[2]);               // t2 = t5 ** 2
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[5]);// t5 = t2 % mod
bignum::mul(tmp[1], point->num1, tmp[6]);       // t6 = t1 * a2_1
list_num::shl(tmp[6], 3u, tmp[2]);              // t2 = t6 << 3
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[6]);// t6 = t2 % mod
bignum::add(tmp[4], tmp[3], tmp[4]);            // t4 = t4 + t3
bignum::mul(tmp[4], point->num1, tmp[2]);       // t2 = t4 * a2_1
bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tmp[4]);// t4 = t2 % mod
bignum::sub_mod(tmp[5], tmp[6], mod->g_modulus, tuple_output->num1);// OUTPUT_1 = t5 - t6 % mod
bignum::add(tmp[4], tmp[1], tmp[2]);            // t2 = t4 + t1
bignum::mul(tmp[2], point->num2, tmp[5]);       // t5 = t2 * a2_2
list_num::shl(tmp[5], 2u, tmp[2]);              // t2 = t5 << 2
return bignum::mod___(tmp[2], mod->g_modulus, (BYTE *)*tmp, tuple_output->num2);// OUTPUT_2 = t2 % mod
```

Fig. 9. Reconstruction d'un algorithme

En appliquant succcessivement cette méthode, des fonctions de plus haut niveau sont identifiées, notamment une implémentation d'échelle de Montgoméry.

Wrap-up De la compréhension de l'auteur, cet aléa est finalement produit par un Dual EC DRBG (NIST SP 800-90A), sur une courbe propriétaire avec des points propriétaires.

Dans la version qualifiée de `Zed!`, ce générateur d'aléa n'est pas utilisé pour générer les secrets des archives.[11]

Le lecteur intéressé pourra consulter la base de donnée HyperElliptic [1] recensant la plupart des algorithmes de calcul sur courbe elliptique connues. Il est ainsi possible de trier les algorithmes en se basant sur le nombre d'opération de base : multiplication, soustraction, etc.

L'implémentation dans `Zed!` est visiblement une implementation *x-only*, ne calculant pas la valeur de la coordonnée `y`. En effet, cette coordonnée n'est pas nécessaire pour l'algorithme Dual EC DRBG.

3.6 ZoneCentral

`ZoneCentral` permet de faire du chiffrement de dossiers, appelés « zones ».

La configuration des zones est stockée à leurs racines, dans un fichier `.zcctl`. Un *mini-filter driver* est chargé de chiffrer et déchiffrer les fichiers à la volée, rendant l'opération transparente pour l'utilisateur.

Format des fichiers de zone Fort des analyses précédentes, l'analyse du format des fichiers de zone est simplifié : il est basé sur les mêmes principes de TLV et de chiffrement du contenu.

Ainsi, les fichiers de zone contiennent :

- un IV global ;
- le chemin absolu de la zone ;
- le nom de la zone ;
- le mode de chiffrement ;
- la liste des fichiers exclus du chiffrement, comme des fichiers liés à `DirectX`, les exécutables, les liens, les fichiers `.zaf`, etc. ;
- la liste d'accès à utiliser (similaire aux archives `Zed!`, basée donc sur l'utilisation de `.zaf`) ;
- un HMAC de l'ensemble.

De même que précedemment, le HMAC n'a été ajouté que dans les dernières versions.

[11] Il est *deprecated* et peut être réactivé à l'aide d'une clé de registre spécifique, non recommandé par l'éditeur

Vulnérabilité 3 *La section contenant le HMAC d'un fichier de zone peut être supprimée. Le client* `ZoneCentral` *ne lève aucune alerte et accepte le fichier résultant.*

Conséquences *Un attaquant ayant les droits de modification d'un fichier de zone, ou pouvant créer un fichier de zone à la racine d'un dossier, peut modifier les propriétés de cette zone. En particulier, il peut ajouter des exceptions pour des fichiers qui ne seront donc pas chiffrés lors de leur sauvegarde sur disque.*

De plus, lorsque le fichier de zone est modifié, il est directement pris en compte par le mini-filter driver sans besoin d'interaction utilisateur.

Chiffrement des fichiers Les fichiers sont chiffrés en « AES-CBC-ZED » (ou AES-CBC-CTS pour les versions plus récentes). L'IV utilisé dépends uniquement de la Zone et du numéro de bloc (par bloc de 512 octets).

Remarque 3 *Cette manière de procéder a pour conséquences :*

- *le XOR de deux petits fichiers (moins de 16 octets) au sein d'une zone est le XOR de leur clair ;*
- *certaines zones peuvent contenir des milliers de fichier. En effectuant des XOR sur l'ensemble des petits fichiers, il peut être possible d'apprendre le masque utilisé et donc de déchiffrer l'ensemble des petits fichiers de la zone (de moins de 16 octets) ;*
- *un attaquant connaissant le clair de la version $n-1$ d'un fichier* `password.txt` *(de moins de 16 octets) peut obtenir le clair du fichier chiffré* `password.txt` *dans sa version n en appliquant un XOR des deux contenus ;*
- *il n'y a pas d'intégrité. Un fichier* `command.bat`*, si suffisamment petit, peut-être modifié par un attaquant connaissant son clair.*

Chemin des fichiers de clés Lors d'une ouverture de zone, l'analyse des I/O systèmes montre que le nom du fichier `.zaf` présent dans les informations de la zone est recherché à plusieurs endroits sur le disque. S'il existe, le fichier est ouvert (dans le contexte de l'utilisateur) pour en calculer son hash.

Vulnérabilité 4 *Si le chemin du fichier* `.zaf` *est un chemin UNC, un accès authentifié sera réalisé vers la cible du chemin.*

Conséquences *En créant ou modifiant (via la suppression du HMAC) un fichier de zone contenant un chemin UNC* `\\cible\dir`*, dès qu'un*

utilisateur va réaliser une action (accès disque en lecture ou écriture) dans un des sous-dossier de la zone, une authentification implicite vers `cible` *sera réalisée.*

Cette tentative est invisible pour l'utilisateur, qu'elle réussisse, timeout ou échoue.

En précisant une IP pour `cible`*, il est ainsi possible d'obtenir une authentification relayable de l'utilisateur (usuellement appelée coerced authentication*`https://www.thehacker.recipes/a-d/movement/mitm-and-coerced-authentications`*), menant in-fine à la prise de contrôle de son compte ou à l'usurpation de ses droits sur le système d'information.*

Les partages réseaux très utilisés (distribution de logiciel, échanges métiers, etc.) et les périphériques USB sont des cibles particulièrement efficaces pour mener ces attaques.

Recherche de variants Les archives `Zed!` permettent aussi l'utilisation de clés `.zaf` externes. Le code responsable de l'ouverture de ces clés dans étant le même que celui de `ZoneCentral`, la vulnérabilité 4 est aussi présente dans ce produit.

Conséquences *Les clés* `.zaf` *permettant le déchiffrement de l'archive, elles sont donc récupérées avant l'entrée d'un secret utilisateur (comme son mot de passe). Il est donc possible, dès la pré-ouverture d'une archive* `Zed!`*, de provoquer une authentification maîtrisée (à des fins de coerced authentication) de l'utilisateur.*

L'extension `ZedMail` *permet de chiffrer des e-mails. Elle est implémentée sous la forme de l'envoi d'une archive* `msg.zed` *par e-mail. Ainsi, un attaquant envoyant un e-mail contenant une archive fabriquée peut potentiellement prendre le contrôle de l'utilisateur ouvrant cet e-mail.*

4 Corrections des vulnérabilités

4.1 Chronologie

- Août 2023 : analyse du produit ;
- mi-septembre 2023 : contact de l'éditeur ;
- 17 octobre 2023 : partage des détails techniques avec l'éditeur ;
- octobre → début décembre 2023 : discussion des impacts, aide à la reproduction par l'éditeur ;
- 13/12/2023 : diffusion coordonnée :
  - par l'éditeur, du patch des produits concernés,

  - par l'éditeur, de bulletin de sécurité sur son site Web,
  - par l'agence, d'un bulletin d'alerte,
  - par l'éditeur auprès de ses clients, de détails sur les vulnérabilités,
  - par l'agence auprès de ses bénéficiaires, de détails sur l'analyse d'impact et les corrections ;
- 15/12/2023 : les agréments « DR » des solutions passent en « Critique » (ie. ne doivent pas être utilisés) et de nouveaux agréments sont émis pour les versions patchées ;
- 13/12/2023 → 19/12/2023 : discussion avec l'éditeur et proposition de l'agence de distribuer un outil permettant d'évaluer les impacts ;
- 19/12/2023 : distribution par l'éditeur à ses clients d'un outil remplissant ce rôle.

4.2 Observations

Analyse d'impact Les archives créés après mise à jour ne contiennent plus les chemins initiaux de création. Néanmoins, il reste nécessaire d'analyser les informations qui ont été émises, en particulier à l'extérieur des SI des entités concernées. Ces informations doivent être considérées comme compromise dans l'analyse d'impact qui en est fait.

De même, les fichiers `.zaf` ayant pu être embarqués dans ces archives, une évaluation de la robustesse des mots de passe utilisés doit être réalisée. Dans la mesure du possible, ces fichiers de clés doivent être renouvelés, ce qui implique de re-chiffrer des éléments (archives et zones).

Agrément et unicité de la solution Si un agrément a pu rapidement être ré-émis, une situation a failli se produire : l'absence temporaire de solution agréé pour les entités pour l'échange de données DR. De l'avis de l'auteur, l'existence d'une seconde solution agréée disponible pour le grand public permettrait de limiter ce risque.

4.3 Résumé des vulnérabilités et remarques

Vulnérabilité ou remarque	Correction	Remarque (éditeur ou auteur)
Vulnérabilité 1	CVE-2023-5044 (8.7)	
Remarque 1		`Zed!` n'a pas vocation à vérifier la provenance
Vulnérabilité 2	CVE-2023-50439 (5.3)	
Remarque 2	CVE-2023-50442 (4.1)	L'analyse par l'éditeur a mené à une autre vulnérabilité
Vulnérabilité 3	CVE-2023-50442 (4.1)	
Remarque 3		Cet algorithme de stockage est utilisé pour avoir un chiffré de même taille que le clair, et faciliter le déplacement de fichier
Vulnérabilité 4	CVE-2023-50441 (4.8), CVE-2023-50443 (4.0), CVE-2023-50440 (7.5)	L'éditeur a identifié un variant dans le produit `Cryhod`

Références

1. HyperElliptic. `https://hyperelliptic.org/`.
2. Projet RR. `https://rr-project.org/`.
3. Time Travel Debugging. `https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/time-travel-debugging-overview`.
4. Unicorn-engine. `https://www.unicorn-engine.org/`.
5. CERT Airbus. TTD-DBG. `https://github.com/airbus-cert/ttddbg`.
6. CERT Airbus. Yara TTD. `https://github.com/airbus-cert/yara-ttd`.
7. ANSSI. Cible de certification pour Zed. `https://cyber.gouv.fr/sites/default/files/document_type/ANSSI-Cible-2022_40fr.pdf`.
8. ANSSI. Visa de sécurité de l'ANSSI. `https://cyber.gouv.fr/le-visa-de-securite`.
9. Royal Midget Baptiste Verstraeten. Symless, un outil de documentation automatique de base IDA. *SSTIC*, 2023.
10. Sylvain Beucler. Zed. `https://www.beuc.net/zed/`.

11. Francis Gabriel Camille Mougey. Désobfuscation de DRM par attaques auxiliaires. *SSTIC*, 2014.
12. CEA. Miasm. `https://github.com/cea-sec/miasm`.
13. Jonathan Salwan Florent Saudel. Triton : Framework d'exécution concolique. *SSTIC*, 2015.
14. Légifrance. II 901. `https://www.legifrance.gouv.fr/circulaire/id/39217`.
15. Camille Mougey. TTD Bindings. `https://github.com/commial/ttd-bindings`.
16. Camille Mougey. Sibyl : function divination. *SSTIC*, 2017.
17. Valentino Ricotta. Bug hunting in Steam : a journey into the Remote Play protocol. *SSTIC*, 2023.

A Résumé des formats

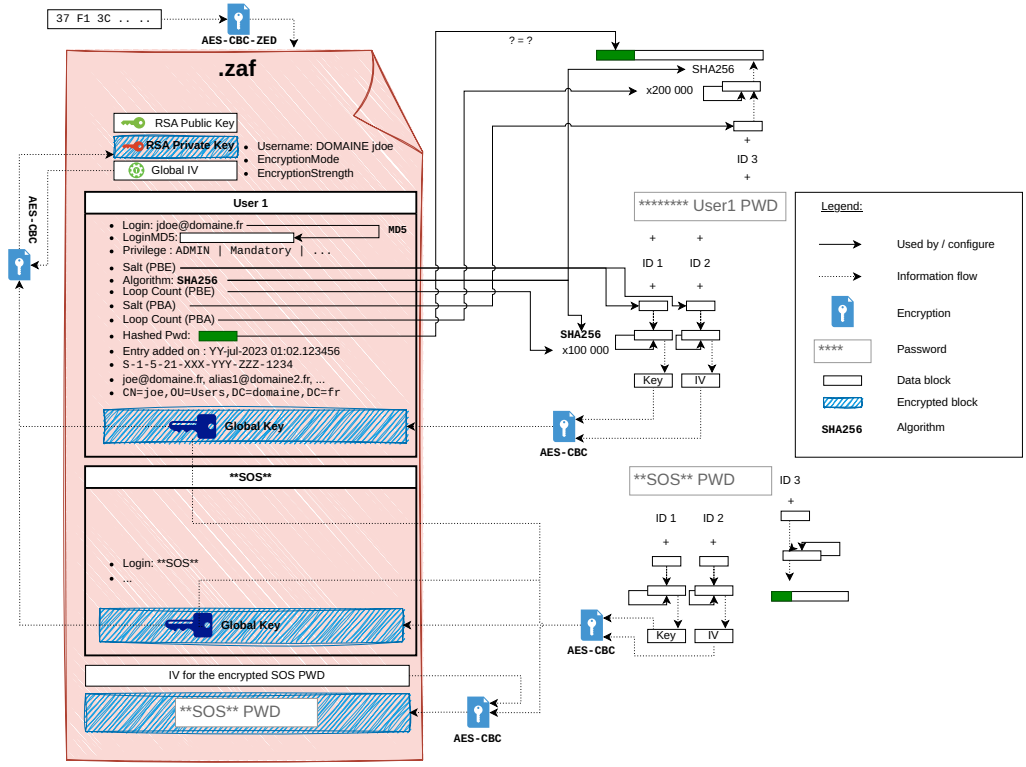


Fig. 10. Résumé d'un fichier ZAF

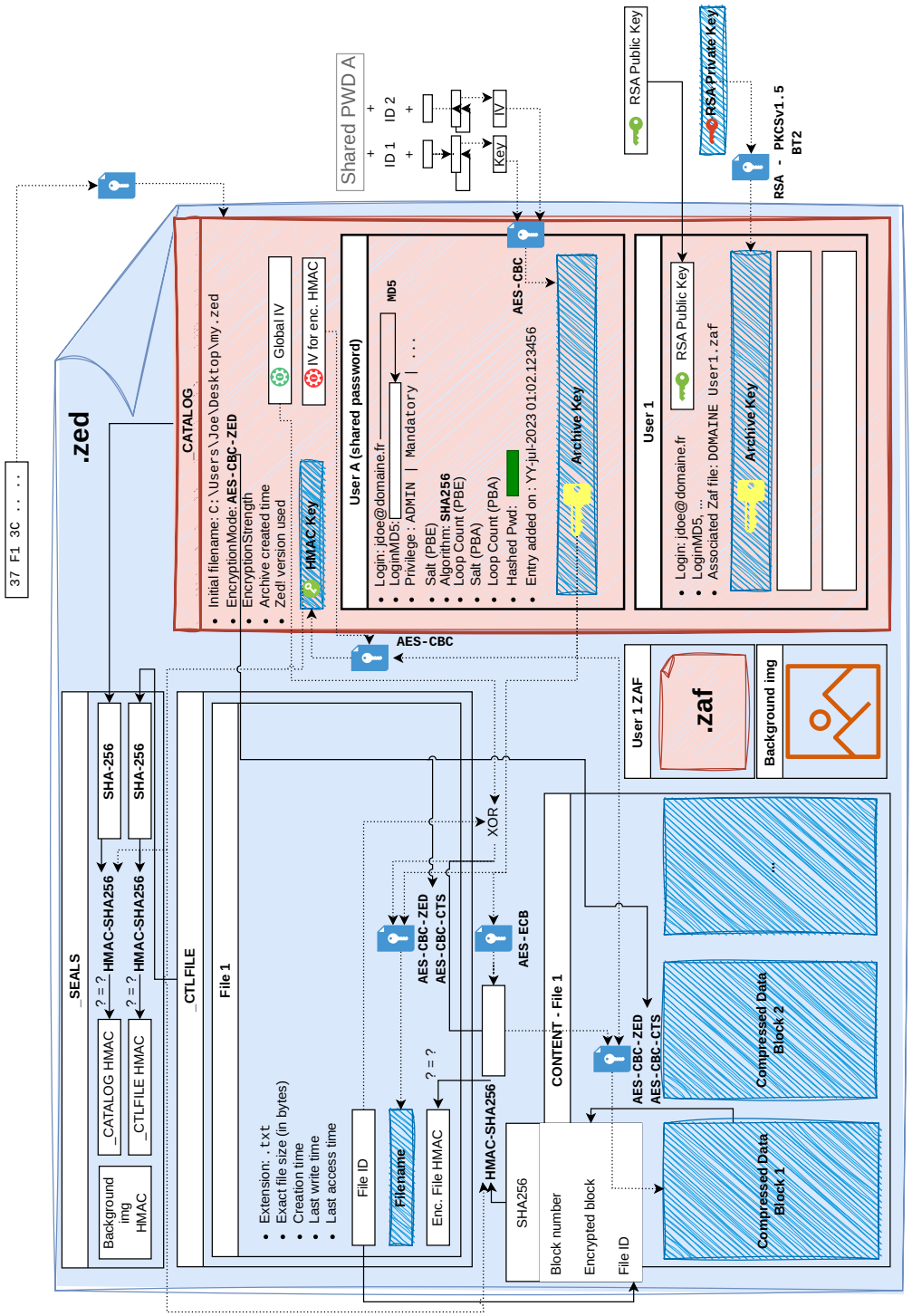


Fig. 11. Résumé d'un fichier ZED