Automatisation de l'analyse de binaires : de la collecte source ouvert à la Threat Intelligence

Frederic Grelot[1], Sebastien Larinier[2] and Marie Salmon[1]

[1]GLIMPS
[2]Epita et Laboratoire Sécurité et Système

Abstract

De nombreuses sources ouvertes de binaires, et particulièrement de malware ont émergé dans le paysage ces dernières années. Et leur qualité n'a rien à envier aux sources commerciales comme le soulignait Thibaut Binetruy (HMiser, CERT Société Generale, 2020), "Integrating operational threat intel in your defense mechanisms doesn't mean buying Threat Intel. You can start by using the [mass] of open source indicators available for free.". Certaines sont mises à disposition par des sources officielles (Abuse.ch, alimenté entre autre par le CERT national Suisse), d'autres de manières plus obscures, voire anonymement (VirusShare, Vx-underground, etc.). Le panorama que nous en avons dressé souligne la grande disparité qualitative et quantitative de ces sources. Il nous a fallu prendre en compte cette diversité dans le cadre de nos travaux de recherche, en concevant une plateforme dédiée nous permettant d'alimenter nos produits d'analyses de binaires, et ainsi rendre possible l'analyse quotidienne des corrélations inter- et intra-familles de malwares à grande échelle. Ces travaux permettent une application sur des cas concrets tels que Babuk, Ryuk et Conti. Nous avons ainsi pu mettre en évidence les liens sur ces familles grâce à l'identification immédiate de corrélations, complétée par une analyse manuelle, qui a ainsi permis de confirmer précisément la généalogie des échantillons.

Keywords

Cybersecurité, Machine Learning, Threat Intelligence, CTI, Malware, Détection, Classification.

1. Introduction

Lorsque l'attaque survient, il est urgent de répondre à la question : qui est l'attaquant et de comment il fonctionne ? Mais ce n'est pas le moment d'y passer du temps. Il est alors nécessaire de se tourner vers l'analyste de Threat Intelligence dont c'est le métier. Lui travaille sur le temps long et sa veille permanente lui permet de maîtriser jusqu'aux dernières évolutions des familles d'attaquants. Mais lui-même a besoin d'être outillé pour détecter les signaux faibles que constituent les nouveautés de ces groupes. Comment le traitement des données publiques (fichiers, bases de données de virus, etc.) peut nous aider à mieux connaître les TTP (Technics, Tactics and Procedures) des attaquants ? Comment l'outillage de l'analyste de Threat Intelligence peut lui permettre de gagner de la connaissance utile à sa réponse à incident ?

À notre connaissance, il n'existe pas, dans le secteur académique de prise en compte du sujet de l'automatisation à grande échelle de l'analyse de malware de bout en bout. La problématique rencontrée dans nos travaux nous a pourtant amené à la conclusion qu'il est nécessaire de

C&ESAR 2021: Automation in Cybersecurity, 16 - 17 Novembre 2021, Rennes, FRANCE
frederic.grelot@glimps.re (F. Grelot)
© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

prendre en compte l'ensemble de la chaîne (de la collecte à l'analyse) si nous souhaitons pouvoir obtenir un résultat final de qualité. Nous faisons ici état de l'infrastructure et des techniques que nous avons mis en place pour répondre à cette problématique : comment peut-on, dans le temps, de manière fiable et totalement automatisée, alimenter la Cyber Threat Intelligence d'une connaissance précise sur les familles de malware et leurs relations ? Nous utiliserons pour cela une méthode de corrélation basée sur les concept-codes car elle est adaptée à la corrélation de nombreux binaires, ce qui n'est pas exclusif car d'autres méthodes peuvent permettre d'obtenir des résultats de ce type [1, 2](SSDEEP, pourcentage de basic-block de code en commun, etc.).

2. Sources de données

Cette section fait l'inventaire détaillé des sources ouvertes de binaires que nous avons recensé. Cette liste se veut non exhaustive et traite majoritairement de données accessibles publiquement et sans abonnement ni enregistrement associé à un processus de validation ou de peering (à l'exception de VirusTotal, citée ici car faisant figure de référence).

Table 1
Tableau de résumé des sources de données utilisées

Nom de source	Goodware	Malware	Codes Sources	Historique	Nombre de fichiers	Facilité d'accès[1]	Tags associés	Symboles de debug
Dépots Linux	X		X	+10 ans	+10M	++		X
Dépots Opensource (Github, …)	X		X[2]	+10 ans	+10M	+		X
Binaires propriétaires	X			1-3 ans	+10M	0		
Conan.io	X		X	1-3 ans	100k-1M	+		X
Chocolatey	X			1-3 ans	100k-1M	+		
Malware-Bazar		X		2 ans	10k-100k	++	+[3]	
VirusShare		X		9 ans	+10M	-	-[4]	
VirusTotal	X	X		+10 ans	+10G	- -[5]		
VX-Undergroud		X	(leaks)	2-3 ans	1k-10k	+	++[6]	

[1] Niveaux de facilité d'accès de très facile (++) à très difficile (- -). 0 représente un niveau neutre.
[2] La compilation nécessite souvent un travail manuel.
[3] Tags manuels attribués par les chercheurs qui l'alimentent.
[4] Qualité des tags assez faible et contenant souvent uniquement le nombre et les noms des détections par les antivirus.
[5] Payant.
[6] Tags attribués directement par l'équipe de VX-Underground.

3. Collecte et traitement

3.1. Contraintes

Nos recherches nous ont amené à développer une capacité de traitement en masse des sources ouvertes, allant de leur collecte automatique jusqu'à la génération quotidienne des images dockers qui constituent nos produits. Nous présenterons dans cette partie nos méthodes et l'infrastructure de collecte, qui sont construites autour de trois contraintes majeures :

- le respect des sources : il est important de favoriser au maximum les capacités de cache local, éviter de solliciter les serveurs de multiple fois, et respecter un rythme raisonnable de requêtes. Sans cela les risques sont multiples, notamment être banni d'un serveur, mais surtout contribuer défavorablement à l'écosystème et mettre en danger le modèle du partage de données en source ouverte qui contribue efficacement à l'écosystème cyber.
- la maîtrise des volumes de données : notre objectif est de collecter et de stocker les données pertinentes sans pour autant faire gonfler nos capacités de stockage. Dans ce cadre, il est important de prendre en compte le besoin et la disponibilité des sources dans le futur : que doit-on absolument conserver ? Que doit-on dupliquer en cas de perte de données ? Que peut-on se permettre de collecter de nouveau si besoin ?
- la maîtrise des temps de calculs et des délais : l'objectif est d'être capable d'ingérer des données en permanence. Mais en cas de mise à jour de la méthode de calcul employée, il faut toujours avoir la possibilité (disponibilité des données) et la capacité (volume/temps de calcul) de soumettre à nouveau l'ensemble des fichiers pour réanalyse. Sans cela, nous nous exposons à deux risques : devoir réduire la fréquence de mise à jour ou être contraint de délaisser volontairement des données lorsqu'elles sont trop anciennes, ce qui peut nuire aux capacités de détection et d'analyse dans la durée.

3.2. Processus mis en place

Pour collecter les malwares, nous utilisons un pipeline (cf. Figure 1), décomposé en trois étapes majeures : l'ingestion, le tagging, et la transformation en concept-codes. Ces étapes se déroulent sous forme de flux CI sur un orchestrateur Gitlab-CI. Ce dernier permet de lancer les jobs de manière régulière et d'assurer une bonne automatisation de l'ensemble du processus.

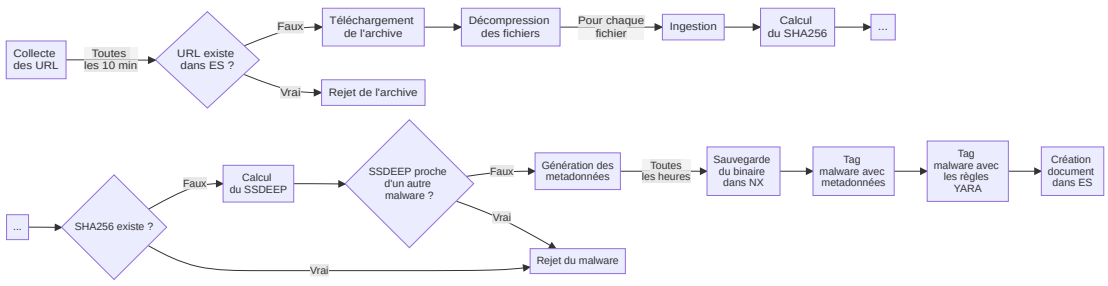


Figure 1: Étapes de collecte et de traitement des binaires.

3.2.1. Ingestion

La première phase, qui réalise l'ingestion, se découpe elle-même selon les étapes suivantes : le téléchargement, l'extraction, le filtrage et enfin le stockage des données.

Téléchargement Sur cette première phase, nous avons utilisé la collecte Web (Beautiful Soup[1]) pour rechercher les liens à télécharger dans les pages des différentes sources de données. Nous implémentons un certain nombre de mécanismes afin d'éviter de surcharger les serveurs sources : parallélisation réduite au strict minimum, reprise des téléchargements interrompus là où ils se sont arrêtés, utilisation de miroirs (voir de liens Torrent) quand c'est possible, etc. Dans 99% du temps, les liens pointent vers des archives : nous conservons un cache des URLs déjà téléchargées afin de ne jamais télécharger deux fois la même donnée. Ces différentes mesures nous permettent notamment de limiter notre propre charge, mais aussi et surtout la charge sur les services sources.

Extraction La phase d'extraction est spécifique à chaque source de données : elle nous permet d'extraire des informations supplémentaires qui dépendent de la source. Par exemple, pour les fichiers téléchargés depuis VX-Underground, nous récupérons l'information du type de Malware lorsqu'il est contenu dans le nom du fichier. L'organisation des liens sur la page de collecte, nous permet également d'extraire la famille d'attaquant (APTxx, etc.) ce qui nous servira dans la phase de tagging. Pour Virus Total (dans le cas où l'on disposerait d'un abonnement) : les archives contiennent les malwares et un fichier JSON qui résume l'analyse et peut permettre de déduire des métadonnées sur la famille.

Filtrage Pour le filtrage des données, nous nous appuyons sur 2 types de hash : le SHA256 et le SSDEEP. Le premier permet de manière très rapide de s'assurer que l'on n'est pas en train de télécharger une archive ou un malware qui serait déjà ingérés. Cela évite notamment les duplicatas, notamment entre les différentes sources de données.

Le SSDEEP nous permet quant à lui de calculer la ressemblance avec d'autres malwares en base : étant donné que nos algorithmes de corrélation sont relativement souples, nous préférons éviter de télécharger des malwares trop proches de ceux que l'on aurait déjà (là où d'autres chaînes peuvent avoir besoin de maximiser le nombre d'échantillons collectés). Le calcul de similarité SSDEEP est relativement complexe (notamment lorsque la base croit). Nous effectuons donc une recherche préliminaire partielle dans une base Elasticsearch, à l'aide d'un tokenzier N-gram de taille 7. Ce tokenizer permet de sélectionner un sous-ensemble très restreint de la base complète, sur lequel on va effectuer un calcul de comparaison complet. Avec les résultats de cette requête, si le score dépasse un seuil défini, le malware est rejeté car considéré comme trop proche d'une souche déjà ingérée.

Stockage L'infrastructure de stockage doit nous permettre de stocker des données de types variés mais aussi de les récupérer à la demande. On doit également pouvoir facilement la faire évoluer au fur et à mesure du projet. Nous avons pour cela choisi de stocker les métadonnées

[1] https://fr.wikipedia.org/wiki/Beautiful_Soup

sur un cluster Elasticsearch et les données dans un serveur Nexus (dont nous n'utilisons qu'une infime partie des fonctionnalités).

3.2.2. Tagging

Pour la phase de tagging, nous avons utilisé deux types de taggers : métadonnées et règles YARA. Le premier va utiliser les métadonnées de la source (typiquement, les tags renseignés dans MalwareBazar ou les familles VX-Underground par exemple), tandis que le deuxième va utiliser une série de règles Yara pour identifier certaines familles. Chaque malware stocké va passer par les deux taggers afin de compléter les informations du malware dans Elasticsearch.

3.2.3. Transformation en concept-codes

Une fois les binaires collectés et identifiés, une dernière étape de transformation est réalisée pour les convertir dans ce que l'on appelle les concept-codes. Cette étape est purement adhérente à la méthode employée pour faire la corrélation de malwares basée sur les concept-codes, et n'est pas nécessaire si l'on souhaite utiliser d'autres méthodes. On peut par exemple la remplacer par un désassemblage qui va extraire les différents basic-blocks de chaque binaire, pour permettre un calcul de similarité basé sur les hash de basic-block. On peut également procéder à une extraction des chaînes de caractères pour détecter par la suite les binaires qui partagent les mêmes chaînes et ainsi produire un score de corrélation basé sur cette métrique. Ces différentes méthodes ne seront pas détaillées ici car elles sont très dépendantes des capacités de l'organisation qui souhaite mettre en œuvre un processus d'automatisation d'analyse de binaires (tant en terme de temps de calcul que dans son accès aux différentes technologies).

3.3. Automatisation

Pour automatiser cette chaîne, nous avons utilisé les éléments d'infrastructure suivants :

- Serveur Gitlab aux fins d'orchestration, pour déclencher, maîtriser et piloter les jobs.
- Elasticsearch pour stocker les métadonnées sur les fichiers collectés.
- Nexus pour stocker tous les binaires.

La chaîne de collecte est asynchrone, ce qui permet de faciliter les reprises, s'il était nécessaire de l'interrompre et de la relancer plus tard. Des recherches de liens pour chaque source sont effectuées toutes les 10 minutes et les tags sont mis à jour toutes les heures. Aujourd'hui, nous avons 8,262,249 malwares traités, dont 4,430,562 non rejetés et 3,282,091 tagués. Cela représente environ 3 To de malwares, ce qui a demandé un à deux mois d'ingestion. Lors des mises à jour, il faut à présent compter environ 1 heure pour une archive de 5Go.

3.4. Problématiques et voix d'améliorations

3.4.1. Taille et format des archives

Les archives peuvent avoir une taille très variable, de quelques Mo à plusieurs centaines de Go (800Go pour la plus grosse archive que nous ayons eue à ingérer). Dans ce cas là, il

devient difficile de décompresser l'archive en entier, cela nécessiterait environ deux fois l'espace disque disponible. Il est donc indispensable de prévoir le processus depuis le départ pour un fonctionnement en "flux", permettant de décompresser l'archive et de l'analyser au fur et à mesure. Au cours de nos collectes, nous avons récolté des archives sous différents formats : zip, rar, 7z et tar. Pour le format zip, tar et rar, la décompression partielle fonctionne très bien. Pour le format 7z, la décompression partielle fonctionne, mais est particulièrement lente. Nous avons donc choisi de gérer ces archives en décompression totale avant analyse, dans la mesure où nous n'avions pas d'archives 7z de taille incompatible avec ce mode.

3.4.2. Tagging

Avec les deux taggers que nous utilisons, nous arrivons à taguer une assez faible proportion des malwares (de l'ordre de quelques pourcents), principalement par l'usage des tags, et assez peu via les règles Yara. Un travail supplémentaire (que nous n'avons pas encore mis en oeuvre) est nécessaire afin de trouver de meilleures règles Yara et probablement en plus grand nombre. Cela prouve par ailleurs la limite forte de cette méthode de détection, basée finalement sur des recherches de signatures. En ce qui concerne notre usage, ce n'est pas un point bloquant (nos algorithmes sont très tolérants aux modifications, donc un faible nombre d'échantillons suffit à proposer une détection efficace) mais il s'agit d'un point important à prendre en compte si l'on souhaite faire un usage à grande échelle nécessitant un pourcentage plus important de malware correctement identifiés.

3.4.3. Évolutions envisagées

Pour le tagging, nous prévoyons d'utiliser la corrélation mise au point chez GLIMPS (basée sur les concept codes) pour rapprocher un malware d'une famille donnée. Cela est possible à partir du moment où le pipeline est parfaitement fiabilisé et évolue peu, car les volumes concernés sont incompatibles d'un passage dans cette chaîne à chaque mise à jour. Ceci sera donc effectué ultérieurement et permettra de résoudre la problématique de tagging.

4. Application des résultats : génération de matrices de cross-corrélation

Une fois les données collectées depuis les sources publiques et traitées dans notre infrastructure, nous pouvons alors calculer les liens entre malwares deux à deux, famille par famille mais également entre les familles. Les calculs de similarités de binaire à binaire ont été effectués grâce à la technologie de corrélation GLIMPS (cf. Figure 2) basée sur la conceptualisation de code. L'intérêt de la conceptualisation de code réside dans sa robustesse vis à vis des modifications, y compris des changements de chaînes ou de versions de compilateur. Cependant, comme évoqué en section 3.2.3, d'autres méthodes existent pour le calcul de similarité, c'est pourquoi nous nous concentrerons plutôt sur l'utilisation des résultats obtenus.

Par ailleurs, le calcul est particulièrement rapide puisqu'une matrice de 1000 binaires (donc un million de comparaisons) peut être obtenue en moins d'une minute, et le temps de calcul est

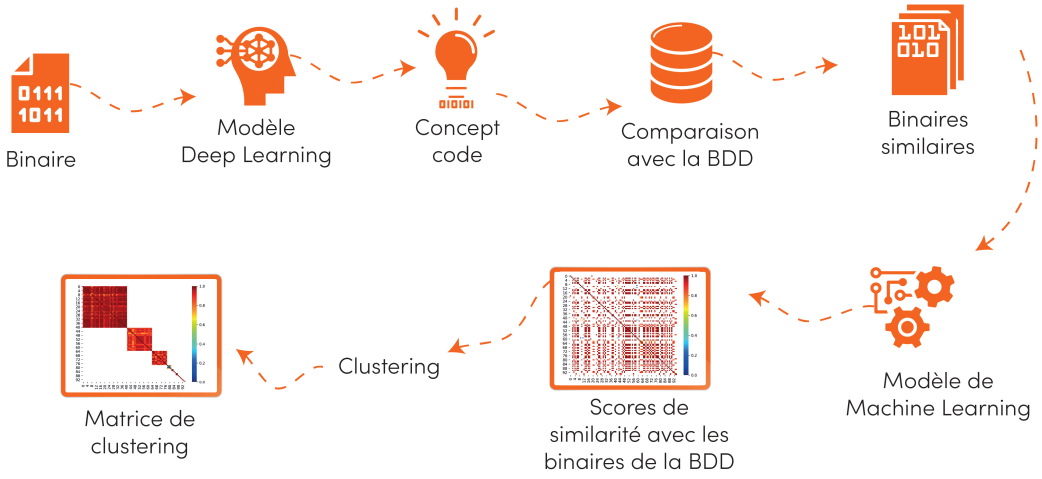


Figure 2: Workflow de traitements de la technologie de corrélation GLIMPS.

linéaire sur la taille et non quadratique comme dans la plupart des autres méthodes, permettant ainsi des calculs sur des familles de plusieurs dizaines de milliers de binaires. Enfin, cette méthode est en temps logarithmique lorsqu'il s'agit d'ajouter un binaire à une collection (l'ajout d'une ligne/colonne dépend logarithmiquement de la taille de la matrice, ce qui est très avantageux en terme de temps de calcul pour les grosses familles). En réalisant ce calcul à grande échelle nous pouvons ainsi générer automatiquement des matrices de corrélation de chaque famille et étudier les évolutions des échantillons en réordonnant ces matrices par valeur (c.à.d. en effectuant un clustering).

4.1. Matrice de corrélation et clustering

Afin de mettre en avant les liens intra- et inter-familles de malware nous définissons de manière automatique des clusters de famille regroupant des binaires similaires (cf. algorithmes 1 à 3). Ceux-ci sont construits à partir d'une matrice de corrélation générée en analysant les binaires deux à deux pour calculer leur scores de proximité.

Une matrice représente en lignes et en colonnes les binaires considérés que l'on identifie par les cinq premiers caractères de leur hash. Dans les Figures 3.b et 4, on préfixe également chaque identifiant du cluster auquel il appartient. L'échelle de valeur va de 0 (bleu) à 1 (rouge) et représente le score de proximité obtenu entre chaque binaire. 1 étant la valeur maximale et représentant une similarité parfaite entre deux binaires (typiquement le même fichier ou une copie). Une absence de couleur indique qu'aucune similarité n'a été détecté entre deux binaires. Un exemple de matrice de corrélation (avant clustering) est représenté en Figure 3.a.

Une fois la matrice construite, nous l'utilisons comme point d'entrée à deux algorithmes de clustering que nous utilisons de manière combinée : DBSCAN[3] et un algorithme de clustering hiérarchique (nommé AgglomerativeClustering dans scikit-learn[4]). La première méthode trouve des échantillons centraux de haute densité et développe des clusters à partir de ceux-ci,

alors que la deuxième place tous les échantillons dans son propre cluster et les fusionnent ensuite successivement selon un critère de sélection donné. Pour les matrices supérieures à 8.000 x 8.000 nous sélectionnons un sous ensemble de binaires de manière aléatoire sur lesquels nous effectuons un clustering, puis nous affectons un cluster aux binaires restant par un vote majoritaire modifié réalisé sur les 5 plus proches voisins de chaque binaire (cf. algorithmes 2, 3).

Algorithm 1: best_clustering

input : Correlation matrix M

$c_1 \leftarrow$ optimal dbscan clustering on M;
$c_2 \leftarrow$ optimal agglomerative clustering on M;
if *silhouette_score*(c_1) $\geq$ *silhouette_score*(c_2) **then**
    return c_1;
else
    return c_2;
end

Les clusterings optimaux sont calculés en testant un ensemble de valeurs pour les deux méthodes.

- Agglomerative Clustering : nombre de clusters à créer $\in [2, |M|/2]$.
- DBSCAN : paramètre epsilon (distance maximale entre deux échantillons pour que l'un soit considéré comme étant dans le voisinage de l'autre) $\in [0.02, 0.55]$ (pas de 0.02).

Algorithm 2: improved_majority_vote

input : binary x, correlation Matrix M, number of neighboors K
output : Associated cluster

$x_{nn}, l_{nn} \leftarrow$ find binaries and clusters of the K-NN of x in M;
$m_{nn} \leftarrow$ mean correlation value on x_{nn} by cluster label from l_{nn};
$v_{max} \leftarrow max(m_{nn})$;
if $\exists!\ v \in m_{nn} \mid v = v_{max}$ **then**
    return cluster with mean correlation value $= v_{max}$;
else
    return cluster with mean correlation value $= v_{max}$ that is the most represented in l_{nn};
end

Le calcul des clusters est effectué de manière itérative (cf. algorithme 3): on détermine des clusters initiaux (avec l'une ou l'autre des méthodes selon la qualité du clustering produit), puis on redécoupe ceux-ci tant que faire ainsi améliore le clustering global. On évalue la qualité du clustering (degré de séparation et densité) en calculant la moyenne du *silhouette coefficient*[5] de chaque binaire (qu'on nomme silhouette score dans la suite de l'article). Il est borné entre -1 pour un clustering incorrect et +1 pour un clustering très dense et bien séparé. Les scores autour de zéro indiquent de nombreux chevauchements entre clusters.

4.2. Exemple de clustering

La Figure 3.b montre un calcul effectué sur une famille relativement petite (13 échantillons). Dans ce cas là, le clustering est relativement rapide (de l'ordre de la seconde) et aboutit à deux clusters bien distincts. Pour obtenir cette figure, on ordonne les échantillons pour les regrouper du plus important au plus petit cluster. Ici un échantillon se retrouve seul dans un cluster (3dda3)

Algorithm 3: Iterative clustering

input : a set of binaries B, a list of family labels by binary L, the associated correlation matrix M, the maximal size for a matrix to recluster N
output : The found clustering c_t
init: $modif$ = True, $blacklist = \emptyset$

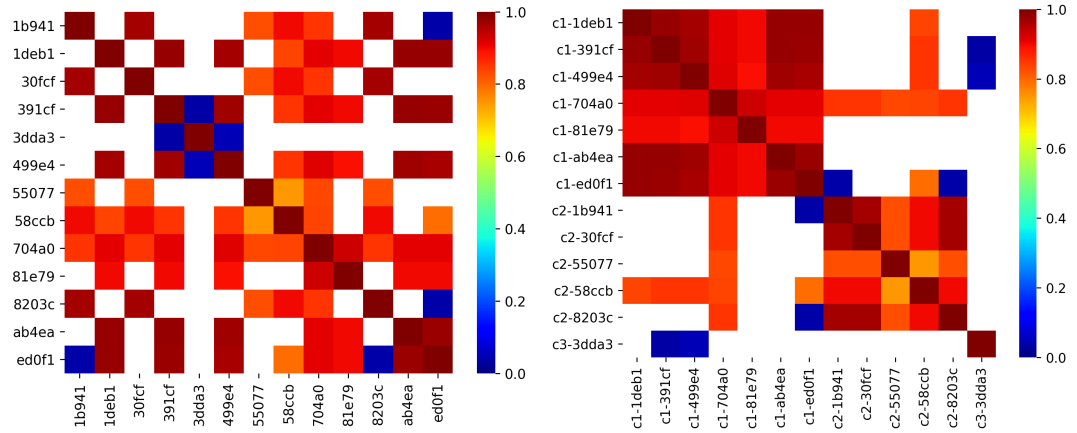
$m_{com} \leftarrow max(|family| \forall family \in \{L\})$;
$n \leftarrow max(N/|L|, 0.5 * m_{com})$;
$B_s \leftarrow \emptyset$;
for $family \in \{L\}$ **do**
    $B_f \leftarrow$ Select at most n random binaries from $family$;
    $B_s \leftarrow B_s \cup B_f$;
end
$c_s \leftarrow$ best_clustering($M[L_s]$);
$c_t \leftarrow c_s$;
for $binary \in B \setminus L_s$ **do**
    $c_t[binary] \leftarrow$ improved_majoritary_vote($binary, M[B \setminus L_s], 5$);
end
$s \leftarrow silhouette_score(c_t)$;
while $modif$ = *True* **do**
    $c_R \leftarrow c_t \setminus blacklist$;
    $modif \leftarrow$ False;
    for $cluster \in c_R$ **do**
        $c_u \leftarrow c_t \setminus cluster$;
        $c_u \leftarrow c_u \cup$ best_clustering($cluster$);
        if *silhouette_score*(c_u) $> s$ **then**
            $c \leftarrow c_u$;
            $modif \leftarrow True$;
        else
            $blacklist \leftarrow blacklist \cup \{cluster\}$;
        end
    end
end

mais il possède un léger lien avec le cluster C1. Le silhouette score est très élevé et montre une bonne séparation des clusters. On note dans la matrice deux échantillons qui vont attirer l'attention de l'analyste : 704a0 et 58ccb. Ces deux échantillons appartiennent à un cluster mais le score de corrélation avec les éléments de l'autre cluster est néanmoins élevé ce qui indique une portion de code commune.

La Figure 4 montre un exemple de reclustering effectué sur les familles Conti et Ryuk fusionnées, dans le but d'étudier le lien entre elles[2]. On obtient un silhouette score élevé également ce qui indique des clusters globalement denses et qui se chevauchent peu. On notera tout de même quelques exceptions qui serviront d'appui aux analyses de la section suivante.

A titre de comparaison, nous avons aussi calculé les distances SSDEEP[2] sur ces familles de malware et généré les matrices de corrélations associées (cf. Figure 5). Celles-ci sont organisées dans le même ordre que les clusterings que nous avons définit. On remarque que pour la famille Babuk on retrouve le cluster C1 de la Figure 3.b privé du binaire 704a0 qui faisait le lien entre

[2]Le temps de calcul sur ces 69 échantillons est de l'ordre de 2.5s



(a) (b)

Figure 3: Matrice de corrélation de la famille Babuk (a) et son clustering (b) qui se compose de 3 clusters (silhouette score = 0.786).

les deux clusters. La méthode SSDEEP n'identifie aucun autre lien entre les binaires Babuk. Pour les familles Ryuk et Conti cette méthode ne met en évidence que très peu de similarités et aucune entre famille. On retrouve cependant un fort lien entre les binaires f0278, 3cd91, c5032 et fa13f de la famille Conti qui appartienne au cluster c5 dans la Figure 4. Nous trouvons donc des similarités en commun en utilisant l'une ou l'autre des méthodes mais sommes capable d'identifier plus de liens entre les binaires y compris entre famille avec notre méthode de calcul de similarité.

Les matrices de corrélations utilisées pour générer ces clusterings (ainsi que d'autres exemples) sont disponibles sur notre dépôt github[3]. Nous fournissons à chaque fois les données d'entrée (c.à.d. la matrice de corrélation entre échantillons) et les images des clusterings obtenus avec notre méthode.

4.3. Applications

Les clusterings générés permettent de faciliter le travail de l'analyste en Threat Intel en lui spécifiant dans quel regroupement se situe un malware ainsi que les liens éventuels avec d'autres groupes et binaires. Cela réduit son temps d'analyse puisqu'il peut cibler les binaires à comparer. Les sorties de cette section serviront de base à une analyse orientée connaissance de la menace dans la section 5 pour illustrer ce fait et valider l'approche proposée.

D'autres applications existent : on peut par exemple utiliser les clusterings générés pour retaguer automatiquement les binaires (en fonction des clusters trouvés) et ainsi obtenir une meilleure vérité terrain pour des apprentissages (détection de malwares, classification de type de malwares, etc.). Ceci permettra, *in fine*, d'améliorer les résultats des algorithmes entraînés.

Il est important de noter que l'on peut facilement faire évoluer un clustering en y injectant automatiquement un nouveau binaire. Pour cela, il suffit de calculer ses scores de similarités

[3] https://github.com/glimps-re/automate_malware_CTI

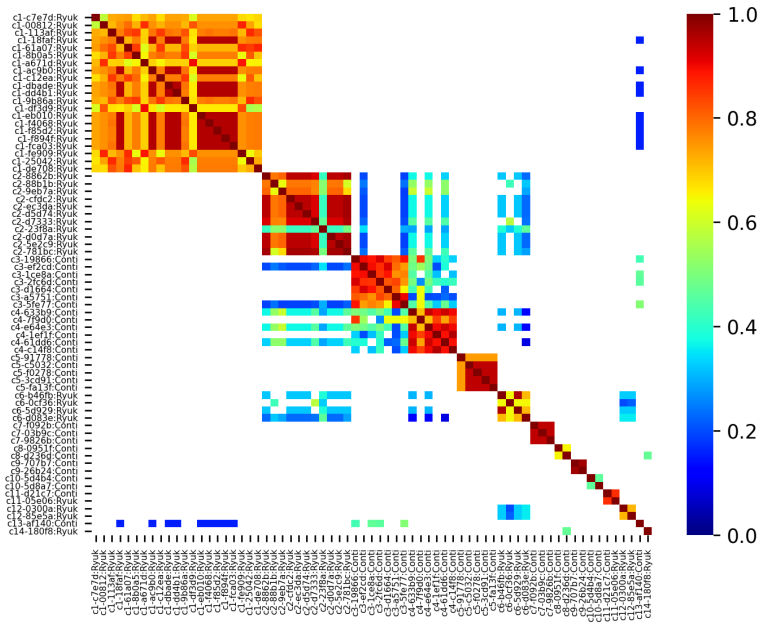


Figure 4: Clustering des familles Conti et Ryuk (14 clusters, silhouette score = 0.725).

avec les binaires déjà clusterisés puis de lui affecter un cluster par vote majoritaire (par exemple) sur les clusters de ses N plus proches voisins.

5. Cas d'usages : les familles Babuk, Ryuk et Conti et leurs liens

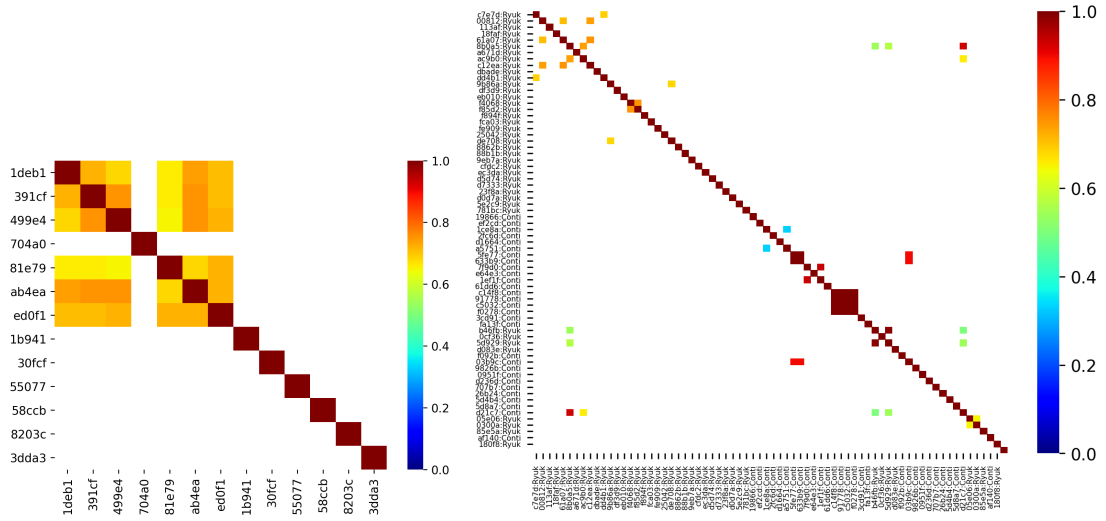
Afin de valider l'approche présentée ci-dessus, il est important de la mettre en regard d'une analyse manuelle des échantillons. Nous avons donc croisés ces résultats avec de l'analyse en rétro-ingénierie menée avec des outils classiques tels qu'IDA[4] qui permet de naviguer dans le désassemblé d'un binaire et d'en tirer une compréhension fine, au prix parfois de nombreuses heures de travail.

5.1. Babuk

Babuk est un virus informatique de la famille des rançongiciels. Il neutralise la machine de la victime en chiffrant l'ensemble des fichiers de l'utilisateur, voire du système si le virus est lancé avec des droits suffisants. Ce virus est apparu pour la première fois sur Virustotal le 22-12-2020 sous le nom de Vasa Locker et le groupe l'opérant s'appelle lui-même Babuk. Ce groupe a fait son apparition sur Raid Forum le 02 Janvier 2021[6]. Comme beaucoup d'acteurs, le groupe met en place un site de "*leak and shaming*" sur Tor[5] afin de prouver la compromission des victimes et les faire chanter dans le but de négocier une rançon la plus élevée possible. La première

[4] https://hex-rays.com/ida-pro/

[5] hxxp://gtmx56k4hutn3ikv[.]onion/



(a) (b)

Figure 5: Matrices de corrélation des familles Babuk (a), Ryuk et Conti (b) calculées en utilisant les distances SSDEEP entre chaque binaire.

analyse publique de Babuk a été faite par le chercheur Chuong Dong[6], étudiant à l'université de Georgia Tech et stagiaire dans l'équipe Flare de FireEye. Les fonctions génériques d'un rançongiciel sont toujours plus ou moins les mêmes, et Babuk n'échappe pas à cette règle :

- Une fonction de parcours du système de fichiers pour chiffrer les données est toujours présente.
- Un algorithme à clé publique pour l'échange de clé de chiffrement des données : cette clé publique sera envoyée par la cible à l'attaquant lors du paiement de la rançon pour que la clé secrète soit utilisée par le logiciel de déchiffrement. A noter que dans la plupart des cas, le logiciel de déchiffrement est fonctionnel : le but est en effet que la victime paie la rançon, et une mauvaise image n'est jamais bon pour les affaires. Par ailleurs, si la victime peut récupérer ses données, le travail n'est pas terminé car elle se doit alors de s'assurer que les attaquants ne sont plus sur son parc et refaire une construction complète de son système d'information. Sans cela, elle se garantit un abonnement de chiffrement de son parc : selon plusieurs études, plus de 20% des entreprises ne parviennent pas à récupérer leurs données après paiement de la rançon et près de 10% des victimes d'une attaque par ransomware en subissent une 2e par la suite. [7, 8].
- Un algorithme à clé secrète pour chiffrer les données. Ce type d'algorithme est préféré pour sa rapidité que les algorithmes à clés publiques.
- La création d'un fichier comportant les indications à suivre pour contacter l'opérateur de l'attaque et des modalités de paiements de la rançon.

Ce sont ces fonctionnalités au cœur du ransomware qui vont subir plusieurs modifications au fil des évolutions, correspondant très exactement à la matrice de corrélation présentée dans la partie précédente (cf. Figure 3.b).

En mai 2021, le groupe a changé de nom se faisant appeler Payloadbin et arrêtant l'activité de rançongiciel[9]. Durant cette période, le code du virus a évolué, et nous l'avons analysé afin de confirmer le clustering proposé automatiquement. La matrice de corrélation présentée montre les trois grosses évolutions de Babuk que l'on résume dans le tableau 2.

hash	date compilation	fonctions	cluster
81e7942a1f	2021-03-15 21:55:04	queue recursive h-128 curve25519	c1
ed0f154481	2021-04-12 15:40:26	queue recursive h-128 curve25519	c1
ab4eae618b	2021-02-23 11:12:08	queue recursive h-128 curve25519	c1
1deb1efad2	2021-03-15 21:50:31	queue recursive h-128 curve25519	c1
391cfcd153	2021-02-19 16:06:32	queue recursive h-128 curve25519	c1
499e43933c	2021-02-19 21:19:12	queue recursive h-128 curve25519	c1
704a0fa7de	2021-01-13 13:12:33	queue recursive salsa20 curve25519	c1
1b9412ca5e	2021-01-02 16:50:38	pas de queue salsa20 ecdh	c2
8203c2f00e	2020-12-30 11:03:14	pas de queue salsa20 ecdh	c2
30fcff7add	2021-01-04 11:20:28	pas de queue salsa20 ecdh	c2
550771bbf8	2021-01-11 17:15:09	pas de queue salsa20 ecdh	c2
58ccba4fb2	2021-01-27 20:41:07	queue recursive salsa20 rsa	c2
e8ccba4fb2	2021-01-28 20:41:07	queue recursive salsa20 curve25519	c3

Table 2
Fonctions de la famille Babuk

L'analyse manuelle par rétro-ingénierie nous apprend que le premier cluster (que nous appellerons C1) utilise :

- un système de queue avec un parcours récursif du système de fichier,
- échange de secret partagé utilisant l'algorithme à clé publique curve25519[6],
- majoritairement l'algorithme hc-128 pour chiffrer les données, sauf un échantillon qui utilise Salsa20.

Le second cluster, C2, est antérieur en terme de date de compilation (le classement dans la matrice se fait en fonction des tailles des clusters et n'a aucun lien avec leurs dates d'apparition) et montre l'utilisation d'algorithmes moins aboutis (notamment pour le parcours/chiffrement) :

- un parcours du système de fichier parallélisé avec des threads uniquement basé sur les disques qui sont présents. Cela a pour conséquence d'avoir un processus de chiffrement très lent. Le parcours des fichiers s'arrête au 16ème niveau d'arborescence. Si des fichiers sont donc plus bas dans l'arborescence, ils ne seront pas impactés.
- L'algorithme d'échange de clés est ECDH en exploitant directement une bibliothèque opensource [7] ou rsa.
- L'algorithme à clés secrètes est Salsa20.

Les échantillons 704a0 et 58ccb nous fournissent cependant une information cruciale : ils font le lien entre ces deux clusters et permettent d'établir un lien entre les deux séries de souches. Ce

[6] https://github.com/agl/curve25519-donna/blob/master/curve25519-donna.c
[7] https://github.com/kokke/tiny-ECDH-c/blob/master/ecdh.c

lien est encore plus visible lorsqu'on le met en regard de la chronologie de compilation (cf. Figure 6). Pour finir, on note la présence d'un outlier, l'échantillon 3dda3 : il possède des fonctionnalités en commun avec le cluster 1 mais celles-ci ont été altérées à la compilation. Il apparaît donc comme isolé du cluster, avec néanmoins un lien faible établi avec deux échantillons.

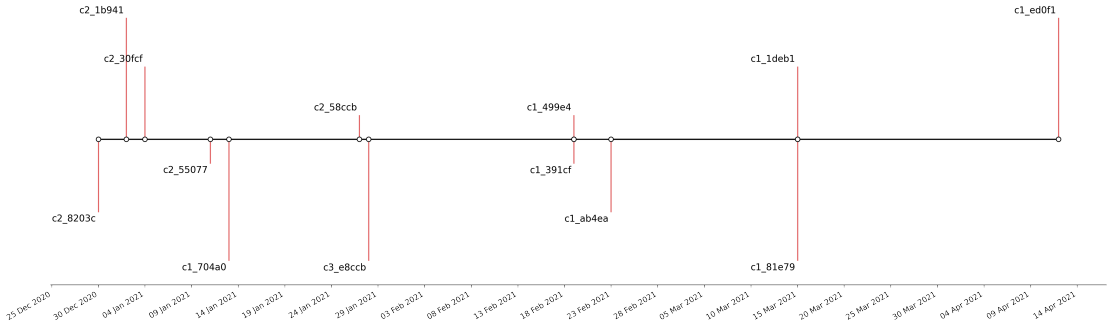


Figure 6: Timeline de la famille Babuk.

5.2. Ryuk et Conti

Cette première analyse sur un groupe avec un nombre d'échantillons réduit a permis de valider l'approche de création de matrice de cross-corrélation et l'exploitation des informations ainsi obtenues. Nous avons alors pu appliquer cette méthode à un nombre d'échantillons plus important de deux groupes de virus (Ryuk et Conti), pour lesquels des liens semblent exister selon la littérature.

Ryuk est, comme Babuk, issu de la famille des rançongiciels. À la différence de Babuk, les développeurs de Ryuk ne sont pas directement les opérateurs de Ryuk. Ryuk est opéré par des affiliés comme l'ont démontré plusieurs articles dont celui de l'ANSSI en mars 2019[10]. Un des opérateurs est EvilCorp, nom donné à un groupe de cybercriminel : ceux-ci compromettent l'ensemble du système d'information de la victime, et ensuite déploient leurs ransomwares sur le parc avant chiffrement. élevé ce qui indique une portion de code commune.

Ryuk a compromis beaucoup de victimes en France, en particulier des hôpitaux[11].

Conti est aussi de la même famille. Ce dernier est apparu en 2020. De la même manière que Babuk, les opérateurs de Conti mettent à disposition un site en ligne listant les victimes et les données que les attaquants ont chiffrées. Comme pour Babuk, cette technique a pour but de faire pression sur la victime et ainsi garantir le paiement d'une rançon la plus élevée possible.

Contrairement à Babuk, ces deux familles utilisent des techniques pour contrer l'analyse par rétro-ingénierie : dans le cas de Conti, aucun appel n'est fait via la table d'import par exemple, mais des chaînes de caractères sont décodées à la volée pour reconstruire la table et les adresses des fonctions qui sont stockées pour être appelées ultérieurement.

La matrice de corrélation met en évidence l'évolution de ces techniques et comment elles ont été implémentées dans les deux familles. Les techniques en elles-mêmes ne sont pas spécifiques, et d'autres virus les ont déjà employées. En revanche, ce sont ici les implémentations de ces techniques qui ont été mises en relation et ont permis d'établir un lien entre les familles Conti

et Ryuk. Par ailleurs, nous avons pu confirmer par une étude bibliographique que des liens ont été démontrés selon lesquels les auteurs de logiciels seraient les mêmes personnes[12, 13]. La matrice de cross-corrélation montre ici un persona de développement identique sur certains échantillons Ryuk (23f8a, 5e2c9, 781bc, 8862b, 88b1b, 9eb7a, cfdc2, d0d7a, d5d74, d7333, ec3da) et Conti (61dd6, e64e3, 633b9). l'auteur ou les auteurs utilisent les mêmes bibliothèques avec le même séquencement dans la programmation. Si l'on regarde la chronologie de compilation (cf. Figure 7), on voit bien l'entrelacement entre Conti et Ryuk. Cette chronologie permet d'ajouter un élément de contexte en plus de la similarité trouvée ci-dessus.

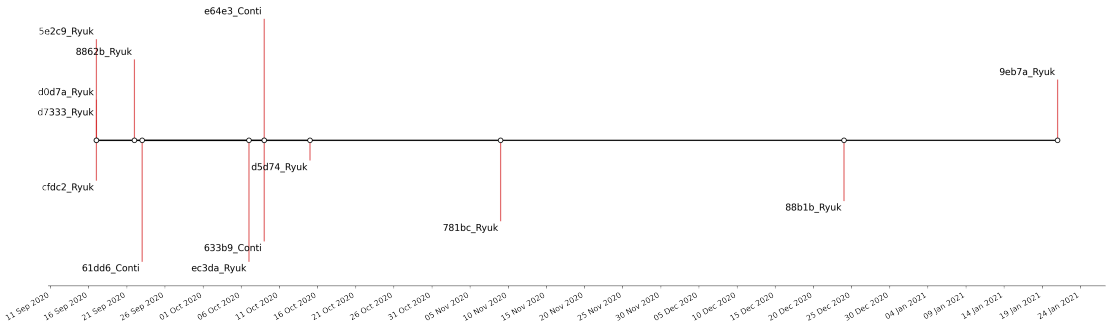


Figure 7: Timeline des familles Conti et Ryuk.

6. Conclusion

Une approche traditionnelle de la Threat Intel permet de fournir une information globale sur la menace liée à tel ou tel groupe. Or comme nous l'avons vu, derrière un nom (Ryuk, Conti, Babuk, etc.) peuvent se cacher de nombreuses variantes d'une même menace. Lorsqu'un incident survient et qu'une entreprise ou une administration a besoin de répondre efficacement, il est essentiel de savoir à quel échantillon la menace est reliée. La collecte continue de malwares et binaires en source ouverte ainsi que leur clusterisation rendue possible par une technologie performante de corrélation permettent de maintenir en permanence à jour une connaissance exhaustive des groupes d'attaquants et de leurs différents virus. Ainsi l'analyste de Threat Intel, lorsqu'il intervient en appui des équipes de réponse à incident, pourra orienter différemment la réaction selon que l'on a affaire à un binaire de chiffrement, de mouvement latéral ou de fuite d'information.

De plus, une réponse à incident efficace implique le plus souvent de mêler les décideurs au processus afin de les tenir au courant des travaux, des avancées à la réponse, et de leur permettre la gestion de crise. L'approche présentée ici permet d'imaginer de proposer automatiquement, en plus des indicateurs classiques de la CTI qui sont des marquants purement techniques (règles Yara, hash, etc.), une visualisation concrète et explicite de positionnement de tel ou tel échantillon trouvé sur le SI dans la « galaxie » des malwares utilisés par l'attaquant. Cet outil de communication permet de rendre tangible la menace et contribue à sa bonne compréhension par les cadres de l'entreprise, et donc une meilleure prise en compte dans les processus de décision.

7. Remerciements

Nous remercions chaleureusement tous les acteurs qui contribuent au partage de données et d'informations sur les menaces cyber; la défense face à ces menaces fait face aujourd'hui à deux enjeux : la capacité à disposer de données pertinentes, et à être capable de traiter cette donnée dans un temps compatible avec la réponse à apporter. Notre métier est l'automatisation, mais il repose sur la donnée. Nous remercions également ceux qui ont soutenu GLIMPS depuis notre création et ont permis ces travaux, notamment le Ministère des Armées, la DGA, la Cyberdéfense Factory, le Poool, Rennes Métropoles, BDI et tous nos partenaires.

References

[1] I. U. Haq, J. Caballero, A survey of binary code similarity, arXiv preprint arXiv:1909.11424 (2019).

[2] J. Kornblum, Identifying almost identical files using context triggered piecewise hashing, Digital investigation 3 (2006) 91–97.

[3] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise., in: Kdd, volume 96, 1996, pp. 226–231.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[5] P. J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of computational and applied mathematics 20 (1987) 53–65.

[6] C. Dong, Babuk ransomware overview, 2021. URL: https://chuongdong.com/reverse%20engineering/2021/01/03/BabukRansomware/.

[7] Sophos, The State of Ransomware 2021, Technical Report, 2021. URL: https://secure2.sophos.com/en-us/medialibrary/pdfs/whitepaper/sophos-state-of-ransomware-2021-wp.pdf.

[8] D. Cecile, Ransomwares : faut-il payer la rançon ?, 2020. URL: https://business.lesechos.fr/directions-financieres/comptabilite-et-gestion/gestion-des-risques/0602742669879-ransomwares-faut-il-payer-la-rancon-334974.php.

[9] Dissent, Babuk re-organizes as payload bin, offers its first leak, 2021. URL: https://www.databreaches.net/babuk-re-organizes-as-payload-bin-offers-its-first-leak/.

[10] Informations concernant les rançongiciels lockergoga et ryuk, Technical Report CERTFR-2019-ACT-005, Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), 2019.

[11] L. Monde, Après celui de dax, l'hôpital de villefranche paralysé par un rançongiciel, 2021. URL: https://www.lemonde.fr/pixels/article/2021/02/15/rancongiciel_6070049_4408996.html.

[12] S. Kalollu, The ryuk-conti connection: A ransomware blog, 2020. URL: http://blog.escanav.com/2020/07/the-ryuk-conti-connection-a-ransomware-blog/.

[13] Le rançonlogiciel Ryuk, Technical Report CERTFR-2020-CTI-011, Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), 2021.