

# Malware Windows Évasifs : Impact sur les Antivirus et Possible Contre-mesure

Cédric Herzog<sup>1</sup>, Valérie Viet Triem Tong<sup>1</sup>, Pierre Wilke<sup>1</sup>, and Jean-Louis Lanet<sup>1</sup>[0000–0002–4751–3941]

Inria, CentraleSupélec, Univ Rennes, CNRS, IRISA, Rennes, France  
`{firstname.lastname}@inria.fr`

**Résumé** L’opposition permanente entre antivirus et malware ne cesse d’évoluer. D’un côté, les antivirus mettent en place des solutions de plus en plus sophistiquées qui viennent s’ajouter à la classique analyse par signature. Cette sophistication conduit les antivirus à laisser de plus en plus de traces de leur présence sur la machine qu’ils protègent. Pour éviter la détection le plus longtemps possible, les malware peuvent éviter de s’exécuter sur de tels environnements en recherchant les modifications laissées par les antivirus. Cet article vise à déterminer les possibilités pour un malware de détecter les antivirus puis à évaluer l’efficacité de ces techniques sur un ensemble d’antivirus parmi les plus populaires de cette année. Nous proposons par la suite une contre-mesure visant à stopper ce genre de malware en simulant les modifications faites par un AV dans le système d’exploitation. Ces leurres sont créés par l’instrumentation de l’API Windows à l’aide de Microsoft Detours. Nous évaluerons cette contre-mesure sur quelques exemples de malware évasifs récupérés dans la nature. Une partie des résultats a été présentée dans la conférence SECURITY 2020[7].

**Keywords:** Antivirus · Evasion · Windows Malware · Windows API.

## 1 INTRODUCTION

Quand un nouveau malware est détecté, les anti-virus (AVs) conduisent des analyses plus poussées afin de produire une signature et de garder leur base à jour rapidement. Les auteurs de malware souhaitant créer des malware qui restent efficaces à bas coût, peuvent utiliser des méthodes simples pour éviter la détection et une analyse plus poussée par ces AVs. Pour cela, un malware peut chercher la présence de traces ou artéfacts laissés par un AV, puis décider ou non de s’exécuter. Dans cet article, nous appelons ce genre de malware des *malware évasifs*.

Dans un premier temps, nous évaluerons la facilité pour un auteur de malware, d’écrire ce genre de malware évasif. Nous détaillons ensuite la façon dont les AVs se comportent avec de nouveaux malware utilisant des techniques d’évasion bien connues. Pour cela, nous développons et utilisons Nuky, un ransomware ciblant Windows et implémentant plusieurs techniques d’évasion.

Dans un deuxième temps, nous concevons et évaluons une contre-mesure initialement imaginée par Chen et al.[4], qui consiste à leurrer le malware évasif en simulant la présence d'artéfacts dans certaines fonctions de l'API Windows afin de le forcer à s'évader. Le but de cette contre-mesure est de limiter la propagation de l'infection et non de remplacer la détection des malware. Nous étudions l'impact de cette approche sur des malware et des logiciels légitimes avant de conclure sur ses limitations. Pour évaluer ces expérimentations, nous créons un petit dataset et discutons le problème de la collecte de malware évasifs.

Nous donnons un aperçu des capacités des antivirus et des techniques d'évasion dans la Section 3 et la Section 4. Nous évaluons les capacités des antivirus à détecter de nouveaux malware dans la Section 5. Puis nous présentons la contre-mesure et la construction du dataset que nous utilisons pour son évaluation dans la Section 6 et Section 7. Pour terminer, nous discutons les possibilités d'évaluation de l'impact de notre contre-mesure dans la Section 8. Ce papier fait suite à la publication présentée à SECURITY 2020[7]<sup>1</sup> et entre plus en détail dans l'implémentation des leurres et leurs impacts.

## 2 ÉTAT DE L'ART

Il existe plusieurs définitions de malware évasif dans la littérature. Nous utilisons une combinaison des définitions proposées par Naval et al.[14] et Tan et al.[15] :

Un malware contenant une ou plusieurs charges servant à rester invisible à tout système de protection et persister pour une longue période. Les charges servant à s'évader déterminent l'environnement d'exécution. Si l'environnement est un environnement d'analyse ou contenant des outils d'analyse, le malware peut décider de stopper son exécution ou mimer un comportement bénin. A contrario, la charge malveillante est délivrée si l'environnement d'exécution semble sans dangers pour le malware.<sup>2</sup>

Une possibilité pour les malware de détecter un environnement d'analyse est de chercher des artéfacts spécifiques présents seulement sur ces environnements. Par exemple, Blackthorne et al.[2] détaillent une expérimentation qui extrait des empreintes d'émulateurs présents dans les AVs. Une fois la collecte d'artéfacts complète, les malware peuvent décider s'ils sont en présence d'un AV ou non.

Les malware peuvent alors utiliser ces artéfacts dans des tests d'évasion comme décrit par Bulazel et al.[3], Afanian et al.[1] . Il est également possible de créer artificiellement des artéfacts en utilisant ce Tanabe et al. appellent des implants[16].

Pour détecter des malware évasifs, il est possible de comparer leurs comportements sur un environnement d'analyse et un environnement bare-metal sans AVs. Pour comparer ces comportements, Kirat et al.[8] cherchent des différences dans la trace d'appels système. Kirat et al.[9] extraient des données brutes des disques durs des deux environnements puis comparent les opérations effectuées

1. DOI : 10.5220/0009816703020309

2. Ces définitions ont été traduites, les articles originaux étant en anglais.

sur les fichiers et les registres Windows par exemple, afin de repérer une déviation de comportement. Enfin, Lindorfer et al.[11] comparent le comportement du malware sur plusieurs sandboxes d'analyse et découvrent de multiples techniques d'évasion.

Il est également possible de créer un environnement d'analyse indistinguable d'un environnement de base afin que les malware évasifs ne puissent pas repérer qu'ils sont analysés et exécutent leur payload. Ce type d'environnement est dit transparent comme présenté par Dinaburg et al.[5]. Toutefois, Garfinkel et al.[6] exprime des réserves quant aux possibilités de créer un environnement entièrement transparent, « le matériel virtualisé et natif resteront probablement différents »<sup>3</sup>.

Au contraire, nous décidons ici d'utiliser cette faiblesse des environnements d'analyse comme un atout défensif. Nous créons de faux artéfacts dans un environnement classique afin de faire croire aux malware évasifs qu'ils s'exécutent dans un environnement d'analyse. À notre connaissance, cette idée a déjà été discutée par Chen et al.[4] mais jamais testée sur de véritables malware.

### 3 CAPACITÉS DES ANTIVIRUS

Les AVs sont des outils visant à détecter et stopper l'exécution de nouveaux malware. Ils sont un agrégat de plusieurs éléments conçus pour détecter la présence de malware par différents moyens comme décrit par Koret et al.[10]. Ces différents éléments induisent des overheads qui, si trop importants, peuvent limiter l'usage de programmes légitimes. Afin de rester compétitifs, les éditeurs d'AVs doivent limiter cet overhead. Il leur est difficile pour cette raison d'utiliser des techniques demandant un long temps d'exécution.

La plupart des AVs utilisent des analyses par signature en comparant tout ou une partie du fichier à leur base. Cette base de signature a besoin d'être mise à jour régulièrement pour détecter les derniers malware.<sup>4</sup>

D'autres AVs utilisent des techniques produisant des overheads importants comme une exécution des malware dans des émulateurs.<sup>5</sup>

Pendant leur installation, les AVs ajoutent des fichiers et modifient le système d'exploitation. Par exemple, un AV peut charger ses modules en allouant des pages mémoires ou en utilisant la fonction de l'API Windows *LoadLibrary*, pour charger un fichier *DLL* comme décrit par Koret et al.[10]. Comme nous le détaillons dans la section suivante, les malware évasifs peuvent détecter ces modifications et ensuite adapter leur comportement.

Après avoir détecté un malware, les AVs l'envoient à leurs serveurs pour des analyses plus poussées. Pour effectuer une analyse plus en profondeur, un analyste peut utiliser plusieurs outils comme un débogueur pour s'arrêter à une

3. virtual and native hardware are likely to remain dissimilar

4. <https://www.kaspersky.co.uk/blog/the-wonders-of-hashing/3629/>

5. <https://eugene.kaspersky.com/2012/03/07/emulation-a-headache-to-develop-but-oh-so-worth-it/>

instruction particulière ou des Machines Virtuelles (VMs) et des émulateurs pour l'analyser dans un environnement fermé.

Un malware souhaitant durer longtemps peut alors essayer de compliquer le travail de l'analyste en utilisant des techniques d'anti-débogueur ou anti-VM afin de ralentir son analyse.

## 4 ÉVASION DES ANTIVIRUS

Nous détaillons maintenant les possibilités pour un malware de détecter les traces d'AV. Il est très aisé pour un auteur de malware de rajouter une étape d'évasion. Plusieurs techniques d'évasion sont détaillées par Koret et al.[3], Catalin-Valeriu et al.[12]. De plus, plusieurs projets traitant de ces techniques de détection sont disponibles sur Github<sup>6</sup>.

Nous classons ces techniques en 3 catégories : détection de débogueurs, détection d'artéfacts d'installation et d'exécution des AVs, et détection de VMs. Dans cette section nous détaillons ces catégories et listons les artéfacts recherchés par Nuky pour chacun d'eux dans la Table 1.

*Détection de débogueurs* : Afin d'observer un malware en détail, un AV peut utiliser un débogueur afin d'ajouter un point d'arrêt à une instruction spécifique. L'utilisation de débogueur implique qu'un processus prenne le contrôle du processus débogué. Cette prise de contrôle laisse des traces visibles depuis le processus débogué. Ce processus peut être un malware évasif, qui peut alors savoir s'il est entrain d'être débogué et modifier son comportement en conséquence. Ces traces peuvent par exemple être l'initialisation de registres spécifiques, le lancement de processus de débogage en mémoire, ou encore le comportement changeant de certaines fonctions de l'API Windows.

*Détection des artéfacts d'installation et d'exécution des AVs* : L'installation d'antivirus crée des changements dans le système d'exploitation. Pour Windows cela commence par la création d'un dossier souvent dans le dossier *Program Files*. Ce dossier contient l'exécutable de l'antivirus ainsi que les ressources dont il a besoin. Pendant l'installation d'un AV, des clefs de registre peuvent être ajoutées ou modifiées. Des instrumentations de certaines fonctions de l'API Windows peuvent être installées dans l'espace utilisateur ou noyau.

De la même manière, l'exécution du processus principal de l'AV et de ses plug-ins laisse des artéfacts au sein du système d'exploitation qui sont visibles au niveau utilisateur. Une simple technique permettant de détecter la présence d'un antivirus est alors de chercher en mémoire les processus correspondants à des d'AV connus. Un malware évasif pourra alors chercher ces artéfacts pour savoir s'il est en présence d'un AV et adapter son exécution.

---

6. <https://github.com/LordNoteworthy/al-khaser>

*Détection de VMs, émulateurs, sandboxes* : Pour finir, un AV peut s'exécuter dans une sandbox, *i.e.*, un environnement de test contrôlé et contenu dans lequel le comportement du malware peut être soigneusement analysé. Ces sandboxes tentent de reproduire l'environnement ciblé le plus fidèlement possible, mais de légères différences permettent de le distinguer d'un environnement réel. Cela peut être la présence d'une interface réseau virtuelle par exemple. Un malware évasif peut alors savoir s'il s'exécute sur une VM ou une sandbox en cherchant ces différences et changer son exécution.

TABLE 1: Artéfacts recherchés par Nuky dans chaque catégorie.

Artéfacts	Débogueur	AV	VM
Process Names	×	×	×
GUI Windows Names	×		
débogueur registers values	×		
Imported functions	×		×
Registries Names & Values	×		
Folder Names		×	×
.DLL Names			×
Usernames			×
MAC addresses			×

## 5 EXPÉRIMENTATIONS

### 5.1 Nuky, un malware configurable

Nous souhaitons maintenant évaluer l'efficacité des techniques d'évasion présentées précédemment. Pour cela, nous développons Nuky, un ransomware évasif et configurable. Il est composé d'une partie évasion implémentant plusieurs techniques de détection d'outils d'analyse et d'une partie payload comprenant différentes méthodes de modification de fichiers que nous voulons tester contre des AVs.

- La partie évasion est composée de trois blocs implémentant des techniques recherchant les artéfacts listés dans la Table 1. Ces blocs peuvent être lancés ensemble ou indépendamment.
- La partie payload possède 4 payloads permettant de modifier les fichiers contenus dans *Mes Documents* comme détaillé dans la Table 2. Ces blocs sont activables indépendamment en fonction de la configuration voulue.

7. <https://github.com/SergeyBel/AES/>

8. <https://github.com/nayuki/Reference-Huffman-coding>

9. [https://www.eicar.org/?page\\_id=3950](https://www.eicar.org/?page_id=3950)

TABLE 2: Payloads de Nuky.

Type	Méthode
Chiffrement	AES ECB mode <sup>7</sup>
Compression	Huffman <sup>8</sup>
Stash	Passe l'attribut caché à vrai
Test	Dépose un fichier Eicar sur le Bureau <sup>9</sup>

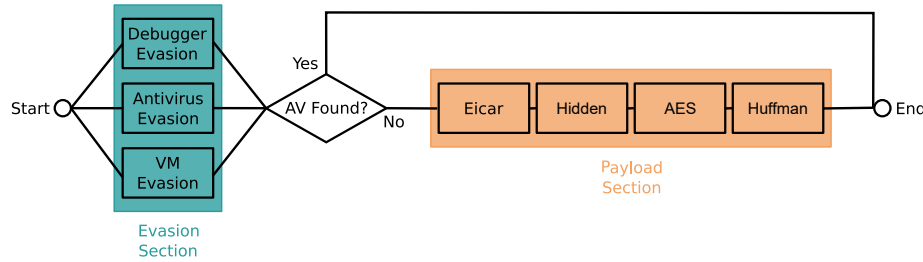


FIGURE 1: Fonctionnement de Nuky.

Dans la suite, nous configurons Nuky de manière à tester les capacités des malware à s'évader. Pour cela, nous utilisons les techniques théoriques détaillées dans la Section 4.

*Mise en place* : Nous préparons une image Windows par Antivirus choisis parmi les plus populaires du moment selon AVTest.<sup>10</sup> Ces images sont toutes dérivées d'une même image Windows 10 remplie de fichiers pris du corpus Govdocs1.<sup>11</sup> Chaque AV est paramétré pour déclencher le plus d'alertes possible. Si un AV propose un moyen spécifique anti-ransomware, nous créons une nouvelle image avec cette fonctionnalité activée pour surveiller *Mes Documents*. Nous utilisons cette configuration pour donner toutes les chances aux AVs de détecter Nuky.

## 5.2 Capacités de détection des Antivirus

Cette première expérimentation vise à définir quel AV est capable de détecter Nucky en fonction de la payload activée et de s'assurer qu'ils repèrent tous le fichier Eicar. Nuky n'utilise aucune méthode d'évasion et exécute chacune de ses 4 payloads. Nuky est un nouveau malware et par conséquent non présent dans les bases de signature des AVs. Néanmoins, ses 3 premières payloads (Compression, Encryption et Stash) sont caractéristiques des ransomware communs. Un AV devrait donc être capable de repérer ces actions dynamiquement.

Comme décrit dans la Table 3, seuls les AVs équipés d'une fonction dédiée à la détection des ransomware ont été capables de détecter les payloads de Nuky.

10. <https://www.av-test.org/en/antivirus/home-windows/>

11. <http://downloads.digitalcorpora.org/corpora/files/govdocs1/>

TABLE 3: Payloads détectées par les AVs. \*Seulement une partie des fichiers a été chiffrée.

Antivirus	AES	Huffman	Hide	Eicar
Windows Defender				×
Windows Defender+	×	×	×	×
Immunet (ClamAV)				×
Kaspersky				×
Avast				×
Avast+	×	×	×	×
AVG				×
Avira				×
K7 Computing	×	×	×	×

Nous suspectons K7 Computing et Windows Defender de générer de faux positifs, car certaines manipulations légitimes déclenchent des alertes. Toutefois, ces AVs détectent Nuky et sont par conséquent les AVs que nous souhaitons que Nuky puisse évader. Finalement, tous les AVs sont capables de détecter la payload qui dépose un fichier Eicar sur le bureau.

Le fait que la majorité des AVs ne détecte pas la compression, le chiffrement et la stash payload peut signifier qu'ils se basent encore beaucoup sur la détection par signature.

### 5.3 Capacités de Nuky à s'évader

Dans cette expérimentation, nous cherchons à vérifier le bon fonctionnement des blocs d'évasion implémentés dans Nuky. Pour cela, nous utilisons la payload Eicar qui est repérée par tous les AVs. Les 3 blocs d'évasion sont activés un à un, puis nous vérifions si l'AV lance une alerte. Si aucune alerte n'est lancée, cela signifie que le malware a repéré un AV et n'a pas déposé de fichier Eicar. Nous saurons alors que le bloc d'évasion testé a bien fonctionné. Autrement, le fichier sera déposé et une alerte sera lancée par l'AV. Le bloc testé aura alors échoué à s'évader.

Le seul bloc pour lequel nous observons une absence d'alerte est le bloc ciblant les artéfacts d'installation et d'exécution des AVs. Pour les autres, soit l'évasion est effectuée dans un environnement fermé pour lequel nous ne pouvons pas observer les résultats, soit les AVs ne génèrent pas les artéfacts recherchés par Nuky. La Table 4 montre le nombre d'artéfacts trouvés par les 2 techniques d'évasion ciblant les AVs.

Le bloc d'évasion ciblant les artéfacts d'installation est donc capable de détecter et identifier facilement un AV. Nuky peut alors décider soit de ne pas exécuter sa payload soit de lancer un exploit ciblant cet AV afin d'augmenter sa durée de vie.

TABLE 4: Nombre d’artéfacts repérés par le bloc d’évasion AV.

Antivirus	Nom de dossiers	Noms de processus
Windows Defender	3	1
Immunet (ClamAV)	1	1
Kaspersky	4	4
Avast	2	2
AVG	2	2
Avira	1	9
K7 Computing	2	9

## 6 CRÉATION D’UN DATASET

Dans la précédente section, nous avons conclu que les méthodes d’évasion ciblant les AVs sont simples à mettre en place et efficaces contre les AVs modernes. Nous détaillons maintenant la façon dont nous avons collecté des échantillons de malware évasifs afin de créer un dataset permettant de tester notre contre-mesure. Premièrement, nous décrivons notre façon de construire un filtre permettant de garder seulement des échantillons évasifs. Deuxièmement, nous détaillons comment nous avons utilisé ce filtre pour collecter les échantillons.

### 6.1 Filtrage via des règles Yara

Pour créer notre filtre, nous utilisons un programme appelé Yara<sup>12</sup>, qui nous permet de trouver des échantillons qui correspondent à un ensemble de règles. Yara peut trouver des échantillons contenant des chaînes de caractères spécifiques ou chargeant une librairie ou fonction Windows spécifique.

Nous avons créé une règle contenant cinq sous-règles chacune composée de plusieurs conditions et conçues pour trouver respectivement des malware qui essaient de *détecter un débogueur*, *détecter un AV ou une sandbox qui s’exécute*, *manipuler et itérer le contenu de dossiers*, *trouver une adresse MAC de VM* et *trouver une fenêtre d’un débogueur*.

Un tel ensemble conçu pour trouver des échantillons évasifs existe déjà<sup>13</sup>. Toutefois cet ensemble retourne trop d’échantillons avec beaucoup de malware qui ne sont pas évasifs. Nous avons donc modifié ces règles en rajoutant des conditions afin de récupérer le plus possible de malware ayant un véritable comportement évasif.

### 6.2 Collecte d’échantillons

La difficulté dans la collecte d’échantillons réside dans le fait que par définition, ces malware évitent de se faire détecter. Nous avons manuellement parcouru

12. <http://virustotal.github.io/yara/>

13. [https://github.com/YaraRules/rules/blob/master/antidebug\\_antivm/antidebug\\_antivm.yar](https://github.com/YaraRules/rules/blob/master/antidebug_antivm/antidebug_antivm.yar)



le WEB dans des datasets publics tels que hybrid-analysis<sup>14</sup> ainsi que des répertoires publics tels que theZoo<sup>15</sup>, par exemple. La collecte couvre une période de 3 mois entre le 18 novembre 2019 et le 21 janvier 2019.

Sur cette période, nous avons trouvé 62 échantillons correspondant à notre règle Yara sur les quelques milliers d'échantillons crawlés.

Afin d'éviter les faux positifs et être sûr de ne garder seulement que les fichiers PE semblant le plus être de vrais malware, nous gardons seulement ceux qui ne sont pas signés et marqués comme malveillants par VirusTotal<sup>16</sup>.

La Table 6 liste tous les MD5 des 18 malware retenus, et la Table 5, montre le nombre de sample récupéré par chaque règle Yara que nous avons utilisé.

Le fait que certaines règles ne retournent aucun résultat peut-être dû au fait que nous avons fait une recherche à la main et donc n'avions pas suffisamment de malware à mettre en entrée du filtre Yara.

Ce filtre Yara est très restrictif et ne retient seulement quelques échantillons pendant la durée de notre récolte. Toutefois, nous avons peu de faux positifs et avons l'intention d'automatiser ce procédé pour continuer l'acquisition de malware.

TABLE 5: Nombre de malware trouvés par chaque règle.

Règles	Tous les échantillons	Echantillons retenus
Débogueur	20	16
AV et sandbox	0	0
Manipulation de dossiers	5	0
Adresses MAC	37	2
Recherche de fenêtre graphique	0	0

### 6.3 Description des échantillons récupérés

Nous regroupons les échantillons en fonction de la ressemblance de leur code assembleur via l'outil diff de Ghidra. Tous les échantillons d'un même groupe contiennent le même code dans leur section *text*. Celle-ci semble contenir le code exécutant l'évasion.

Les groupes A, B et C partagent une partie significative de leur section *text* avec quelques petits ajouts de code ou modifications de la signature de certaines méthodes. Les autres groupes sont très différents et contiennent un code qui leur est propre.

Nous calculons l'entropie de chaque échantillon et trouvons une valeur minimale de 6.58 et une valeur maximale de 7.11. En se basant sur l'expérimentation

14. <https://www.hybrid-analysis.com/>

15. <https://github.com/ytisf/theZoo>

16. <https://www.virustotal.com>

faite par [13], ces valeurs suggèrent que les échantillons sont empaquetés ou chiffrés. Nous trouvons dans chaque échantillon une section contenant les ressources différentes. Dans certains échantillons, cette section semble contenir un PE qui pourrait être dépaqueté ou déchiffré après les tests d'évasion.

Toutes ces similarités entre les échantillons pourraient être dues au partage de code dans des projets publics tels que Al-Khaser<sup>17</sup> ou des projets de plus petite envergure<sup>18</sup>. Certains packers proposent d'ajouter une partie évasion avant le dépaquetage tel que Trojka Crypter.<sup>19</sup>

TABLE 6: Échantillons sélectionnés et résultats de la contre-mesure.

Groupe	MD5	Résultats
A	de3d1414d45e762ca766d88c1384e5f6	OK
	2d57683027a49d020db648c633aa430a	OK
	d3e89fa1273f61ef4dce4c43148d5488	OK
	bd5934f9e13ac128b90e5f81916eebd8	OK
	512d425d279c1d32b88fe49013812167	OK
	2ccc21611d5afc0ab67ccea2cf3bb38b	OK
	6b43ec6a58836fd41d40e0413eae9b4d	OK
B	ee12bbee4c76237f8303c209cc92749d	OK
	5253a537b2558719a4d7b9a1fd7a58cf	OK
	8fdab468bc10dc98e5dc91fde12080e9	OK
	e5b2189b8450f5c8fe8af72f9b8b0ad9	OK
	ee88a6abb2548063f1b551a06105f425	OK
	4d5ac38437f8eb4c017bc648bb547fac	OK
C	f4d68add66607647bf9cf68cd17ea06a	OK
D	862c2a3f48f981470dcb77f8295a4fcc	CRASH
E	e51fc4cdd3a950444f9491e15edc5e22	NOK
F	812d2536c250cb1e8a972bdac3dbb123	NOK
G	5c8c9d0c144a35d2e56281d00bb738a4	CRASH

## 7 CONTRE-MESURE

Il est possible de contrecarrer ces évasions en instrumentant une partie de l'API Windows afin de créer de faux artéfacts d'outils d'analyse et donc inciter le malware à s'évader.

Pour effectuer ces instrumentations, nous utilisons Microsoft Detours qui produit une DLL que nous chargeons en utilisant la clef de registre *AppInit*.<sup>20</sup>.

17. <https://github.com/LordNoteworthy/al-khaser>

18. <https://github.com/maikel233/X-HOOK-For-CSGO>

19. MD5 : c74b6ad8ca7b1dd810e9704c34d3e217

20. <https://attack.mitre.org/techniques/T1103/>

Nous décrivons maintenant les fonctions de l'API Windows que nous modifions et discutons de leurs effets sur de vrais malware avant de mesurer l'overhead induit.

## 7.1 Instrumentation de l'API Windows

Nous implémentons 3 types d'instrumentations sur 7 fonctions. Le premier type de modification change la fonction pour retourner une valeur constante. Le second type de modification décide en fonction des valeurs passées en paramètre de retourner ou non une fausse valeur que nous contruisons de sorte à provoquer l'évasion. Le dernier type de modification crée des changements persistants qui contrairement aux 2 autres types de modifications précédent seront toujours visible après l'appel à la fonction.

### Retourner une valeur unique

- `IsDebuggerPresent` :
  - Description : Cette fonction retourne *True* si un débogueur est attaché au processus qui l'appelle.
  - Usage évasif : Un malware vérifiera la valeur retournée par *IsDebuggerPresent*. Si *True* est retourné, alors un débogueur est présent, et le malware stoppe son exécution.
  - Modifications : Nous modifions cette fonction pour toujours retourner *True*.

### Modification du comportement en fonction des arguments

- `GetModuleHandle` :
  - Description : Cette fonction retourne un handle au module dont le nom est passé en paramètre.
  - Usage évasif : Les malware vérifient la présence de DLL spécifiques appartenant à un AV ou un outil d'analyse en demandant l'accès à un handle. Si le malware obtient le handle, alors la librairie est présente et chargée. Le malware évasif stoppe donc son exécution.
  - Modifications : Nous créons une liste de noms de librairies susceptibles d'être recherchées par les malware évasifs. Nous modifions le comportement de la fonction pour faire en sorte que si un des noms de la liste est présent en argument nous retournions un handle à *User32.dll*. Si la librairie n'est pas dans notre liste, nous appelons la méthode *GetModuleHandle* originale.
- `RegOpenKeyEx` :
  - Description : La fonction *RegOpenKeyEx* retourne un handle à une clef de registre ouverte.
  - Usage évasif : Des clefs de registre peuvent être ajoutées ou modifiées par les VMs et les outils d'analyse. Un malware évasif peut chercher la présence de ces clefs de registre afin de vérifier la présence de ces outils.

- Modifications : Nous créons une liste de noms de clefs de registre susceptibles d’être recherchées par les malware évasifs. La fonction *RegOpenKey* est modifiée pour renvoyer *ERROR\_SUCCESS* si la clef recherchée fait partie de cette liste. Sinon le comportement normal est utilisé et nous retournons le résultat de *RegOpenKeyEx* original.
- RegQueryValueEx :
  - Description : Cette fonction retourne le contenu du registre dont le nom est passé en paramètre.
  - Usage évasif : En plus de vérifier la présence d’une clef de registre, un malware évasif peut décider de lire son contenu.
  - Modifications : Nous dressons une liste de noms de clefs de registre associée à des valeurs potentiellement recherchées par des malware évasifs. Si un malware demande le contenu d’une clef présente dans notre liste, nous retournons la valeur conçue pour le faire s’évader.
- CreateFile :
  - Description : La fonction *CreateFile* est utilisée pour créer ou ouvrir un fichier ou un périphérique I/O.
  - Usage évasif : Les AVs ou outils d’analyses peuvent ajouter des fichiers ou des périphériques I/O. Les malware évasifs peuvent vérifier leur présence en essayant de les ouvrir.
  - Modifications : Nous avons créé une liste de noms de fichier et périphérique I/O susceptibles d’être recherchés par les malware évasifs. Cette fonction modifiée retourne *ERROR\_SUCCESS* si le fichier ou périphérique demandé fait partie de la liste.
- GetFileAttributes :
  - Description : Cette fonction retourne les métadonnées associées à un fichier passé en paramètre.
  - Usage évasif : Une autre façon de vérifier la présence de fichiers est de demander l’accès à ses métadonnées. Si le malware évasif arrive à accéder à ces métadonnées, le fichier est présent donc un AV est potentiellement installé.
  - Modifications : Si le fichier est présent dans notre liste, nous retournons *FILE\_ATTRIBUTE\_NORMAL*.  
De la même manière, nous retournons *FILE\_ATTRIBUTE\_DIRECTORY* pour un répertoire.

### Leurres persistants après l’exécution de la fonction

- CreateToolhelp32Snapshot :
  - Description : *CreateToolhelp32Snapshot* crée un snapshot de tous les processus et threads dans le système.
  - Usage évasif : Un malware peut chercher en mémoire la présence de processus appartenant aux AVs.
  - Modifications : Au premier appel à *CreateToolhelp32Snapshot*, nous créons plusieurs processus de boucles infinies nommés avec des noms d’AV connus. Aux prochains appels à *CreateToolhelp32Snapshot*, la fonction

instrumentée vérifiera que les processus ne soient pas déjà présents avant d'en recréer.

## 7.2 Chargement des fonctions instrumentées dans les malware

Nous utilisons le framework Microsoft Detours pour injecter nos fonctions instrumentées une fois que le processus a démarré. Detours génère une DLL que nous devons charger dans le malware. Pour être certains de ne rater aucun malware, nous injectons nos DLL 32-bit et 64-bit dans tous les processus. Pour cela, nous utilisons les 2 registres *AppInit* afin de charger nos DLL dans tous les processus utilisant *User32.dll*. Comme cette librairie est très utilisée, nos modifications seront injectées dans la plupart des programmes.

## 7.3 Efficacité de la contre-mesure

Nous testons la contre-mesure sur 18 malware pour lesquels nous effectuons 2 exécutions. Premièrement sur un Windows non modifié afin d'avoir un comportement normal de l'application. Deuxièmement sur une machine similaire avec un Windows équipé de la contre-mesure. Nous comparons ensuite le comportement par rapport à la première exécution.

Nous considérons que la contre-mesure a forcé avec succès le malware à s'évader si le comportement est différent entre les 2 images. Tous les 18 malware sont testés à la main en utilisant Process Monitor de la librairie *SysInternals*.<sup>21</sup>

Pour 14 malware, nous observons un changement de comportement sur la machine équipée de la contre-mesure. Un subprocess n'est pas créé si la contre-mesure est activée. Avec la contre-mesure activée, le process parent s'arrête immédiatement ce qui pourrait signifier qu'ils s'évadent.

Les résultats détaillés dans la Table 6 montrent que sur 18 malware, 2 malware crashent dans les 2 environnements et 2 autres ne changent pas de comportement.

## 7.4 Overhead de la contre-mesure

Nous évaluons l'overhead produit par la contre-mesure sur 4 logiciels Windows connus que sont VLC, Notepad, Firefox, et Microsoft Paint. Nous ajoutons également Al-Khaser, un programme exécutant plusieurs techniques d'évasion et générant un rapport sur leurs résultats. Nous avons retiré d'Al-Khaser toutes les techniques reposant sur l'observation du temps.

Chaque logiciel est lancé 100 fois, est manipulé pendant un temps en suivant un comportement scripté. Nous calculons ensuite la moyenne des temps d'exécution avec et sans la contre-mesure comme détaillé dans la Table 7.

Le seul overhead significatif que nous observons est avec Al-Khaser qui ajoute 14.62% de temps d'exécution. Cela peut être dû au fait qu'il appelle beaucoup les fonctions que nous avons instrumentées, ce que nous considérons comme le pire scénario.

21. <https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>

TABLE 7: Temps d'exécution (en secondes) avec et sans la contre-mesure activée.

Logiciel	Moyenne avec contre-mesure	Moyenne sans contre-mesure	Ecart-type avec contre-mesure	Ecart-type sans contre-mesure
Al-Khaser	15.24	13.29	1.38	1.36
VLC	0.8772	0.8593	0.0609	0.0677
Notepad	0.0615	0.0607	0.0033	0.0028
Firefox	0.8005	0.8026	0.0270	0.0259
MS-Paint	0.0581	0.0583	0.0096	0.0106

## 8 DISCUSSIONS ET TRAVAUX FUTURS

Cette contre-mesure fonctionne sur certains malware mais nous ne connaissons pas encore bien son impact sur des logiciels légitimes. Pour répondre à cela, nous préparons des expérimentations ayant pour but de trouver et quantifier les différences entre le comportement d'un logiciel en présence de la contre-mesure et le comportement du même logiciel sans la présence de la contre-mesure. Pour cela, nous comptons utiliser *drldtrace*<sup>22</sup> afin de créer une trace listant toutes les fonctions appelées par un programme lors d'une exécution.

Avec cet outil, nous souhaitons enregistrer plusieurs traces dans un environnement sans contre-mesure puis dans un environnement avec contre-mesure pour ensuite les comparer entre elles. Il se pourrait que la présence ou non de la contre-mesure impacte le nombre d'appels de fonctions présents dans les traces. Il est également possible que ces différences de nombre d'appels ne soient valables que pour quelques fonctions précises. Nous pensons rechercher les différences entre les traces à l'aide d'outils statistiques tels que le test de Student<sup>23</sup> par exemple.

Nous pouvons aussi créer pour chaque exécution un vecteur contenant des données déduites des traces puis appliquer un algorithme de clustering tel que le K-moyennes<sup>24</sup>. Si les deux environnements génèrent des exécutions différentes, le clustering devrait être capable de classer ces exécutions en deux groupes distincts. À l'inverse, ne pas pouvoir séparer les exécutions en deux groupes pourrait signifier que l'impact de la contre-mesure serait moindre, car il générerait peu de différences.

Quelques tests de clustering ont été effectués et semblent indiquer qu'il est possible de distinguer les exécutions avec la contre-mesure des exécutions sans la contre-mesure. Cependant nous ne sommes pas encore capables de définir si ces différences sont dues au code généré par Microsoft Detours et à la façon d'injecter la DLL dans les programmes, ou aux modifications que nous avons apportées à l'API Windows. Si cela est principalement dû à l'utilisation de Microsoft Detours,

22. <https://github.com/mxmssh/drltrace>

23. [https://fr.wikipedia.org/wiki/Test\\_de\\_Student](https://fr.wikipedia.org/wiki/Test_de_Student)

24. <https://fr.wikipedia.org/wiki/K-moyennes>

nous pourrions utiliser une méthode de modification de l'API Windows différente et obtenir une contre-mesure plus discrète.

## 9 CONCLUSION

*Nuky* : Lors des tests de résistance des AVs contre les techniques d'évasion de Nuky, nous utilisons un petit nombre de techniques basiques d'évasion bien connues des attaquants et facilement reproductibles. Ces techniques ne sont pas représentatives de toutes les manières possibles de s'évader. Pour l'instant Nuky représente un malware que n'importe quel attaquant soit capable d'écrire en suivant des tutoriels et non un malware avancé créé par de grands groupes ou états. Nous n'avons pas réussi à tester les techniques d'évasion ciblant les débogueurs et VM's avec notre méthode, mais pensons qu'il est possible les tester avec des tests en boîte noire plus élaborés.

Cependant, Nuky a été capable de détecter et identifier des AVs pour ensuite adapter son comportement.

*Contre-mesure* Tout comme une base de signature, notre contre-mesure a besoin d'être mise à jour de façon continue pour être capable de stopper de nouvelles techniques. Naturellement, avec cette méthode nous pouvons seulement bloquer les malware évasifs utilisant l'API Windows et pas ceux passant par d'autres moyens. Nous continuons l'étude de l'impact de la contre-mesure sur les malware, les logiciels légitimes et le système d'exploitation à l'aide d'outils statistiques.

*Dataset* Concernant le dataset, il n'y a pas de répertoires publics fournissant des malware proprement détaillés avec les techniques d'évasion qu'ils utilisent. Pour cette raison nous utilisons un petit dataset que nous avons créé et analysé à la main, mais sur lequel nous ne pouvons généraliser notre conclusion pour le moment.

Après avoir testé les techniques d'évasion sur des AVs, nous avons élaboré une contre-mesure en instrumentant l'API Windows. Les faux artéfacts mis en place ont permis de stopper 14 malware sur 18 qui ont montré un changement de comportement en leurs présences. L'overhead sur le temps d'exécution le plus important est mesuré sur Al-Khaser avec un ajout de 14.62%. Le code de la partie évasive de Nuky et la règle Yara utilisée pour le filtrage sont accessibles sur demande.

## Références

1. Afanian, A., Niksefat, S., Sadeghiyan, B., Baptiste, D. : Malware dynamic analysis evasion techniques : A survey. CSUR Computing Surveys - ACM **52**(6) (jan 2020)
2. Blackthorne, J., Bulazel, A., Fasano, A., Biernat, P., Yener, B. : Avleak : Fingerprinting antivirus emulators through black-box testing. In : WOOT Workshop on Offensive Technologies. pp. 91–105. No. 10, USENIX Association, Austin, TX, USA (aug 2016)

3. Bulazel, A., Yener, B. : A survey on automated dynamic malware analysis evasion and counter-evasion : Pc, mobile, and web. In : ROOTS Reversing and Offensive-Oriented Trends Symposium. pp. 1–21. No. 1, ACM, Vienna, Austria (nov 2017)
4. Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J. : Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In : DSN Dependable Systems and Networks. pp. 177–186. No. 38, IEEE Computer Society, Anchorage, Alaska, USA (jun 2008)
5. Dinaburg, A., Royal, P., Sharif, M.I., Lee, W. : Ether : malware analysis via hardware virtualization extensions. In : CCS Conference on Computer and Communications Security. pp. 51–62. No. 15, ACM, Alexandria, Virginia, USA (oct 2008)
6. Garfinkel, T., Adams, K., Warfield, A., Franklin, J. : Compatibility is not transparency : VMM detection myths and realities. In : HotOS Hot Topics in Operating Systems. pp. 30–36. No. 11, USENIX Association, San Diego, California, USA (may 2007)
7. Herzog, C., Viet Triem Tong, V., Wilke, P., Van Straaten, A., Lanet, J. : Evasive windows malware : Impact on antiviruses and possible countermeasures (17), 302–309 (jul 2020)
8. Kirat, D., Vigna, G. : Malgene : Automatic extraction of malware analysis evasion signature. In : SIGSAC Conference on Computer and Communications Security. pp. 769–780. No. 22, ACM, Denver, Colorado, USA (oct 2015)
9. Kirat, D., Vigna, G., Kruegel, C. : Barecloud : Bare-metal analysis-based evasive malware detection. In : USENIX Security Symposium. pp. 287–301. No. 23, USENIX Association, San Diego, California, USA (aug 2014)
10. Koret, J., Bachaalany, E. : The Antivirus Hacker’s Handbook. No. 1, Wiley Publishing (oct 2015)
11. Lindorfer, M., Kolbitsch, C., Comparetti, P.M. : Detecting environment-sensitive malware. In : RAID Recent Advances in Intrusion Detection. pp. 338–257. No. 14, Springer, Menlo Park, California, USA (sep 2011)
12. Lita, C., Cosovan, D., Gavrilut, D. : Anti-emulation trends in modern packers : a survey on the evolution of anti-emulation techniques in UPA packers. *Computer Virology and Hacking Techniques* **12**(2) (fev 2018)
13. Lyda, R., Hamrock, J. : Using entropy analysis to find encrypted and packed malware. *SP Security & Privacy - IEEE* **5**(2) (fev 2007)
14. Naval, S., Laxmi, V., Gaur, M.S., Raja, S., Rajarajan, M., Conti, M. : Environment-reactive malware behavior : Detection and categorization. In : DPM/QASA/SETOP Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance. pp. 167–182. No. 3, Springer, Wroclaw, Poland (mar 2015)
15. Tan, J.W.J., Yap, R.H.C. : Detecting malware through anti-analysis signals - A preliminary study. In : CANS Cryptology and Network Security. pp. 542–551. No. 15, Springer, Milan, Italy (nov 2016)
16. Tanabe, R., Ueno, W., Ishii, K., Yoshioka, K., Matsumoto, T., Kasama, T., Inoue, D., Rossow, C. : Evasive malware via identifier implanting. In : DIMVA Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 162–184. No. 15, Springer, Saclay, France (jun 2018)