# APT28, sarA Is watching you !

Robin Kwiatkowski et Charles Meslay
robin.kwiatkowski@sekoia.io
charles.meslay@sekoia.io

Sekoia.io

**Résumé.** En 2025 dans le cadre de nos travaux de veille sur les menaces étatiques, nous avons analysé une nouvelle chaine d'infection du mode opératoire adverse (MOA) APT28. L'analyse des implants de cette chaine nous a fait prendre conscience que l'IA n'était pas juste une technologie d'avenir mais pouvait dès à présent être utilisée même si l'outillage associé ne nous satisfaisait pas complètement. Notamment, il apparaissait qu'il manquait une couche d'orchestration pour obtenir des résultats pertinents dans le cadre de nos analyses. Cela nous a mené à développer SARA (*Semi Autonomous Reverse Analyst*), un outil d'aide à l'analyse de logiciel malveillant à l'aide d'intelligence artificielle. Nous souhaitons donc ici présenter comment nous sommes arrivés à ce constat ainsi qu'un retour d'expérience sur la mise en oeuvre de cet outil.

## 1 Introduction

### 1.1 Rappels sur la CTI

Le renseignement sur la menace cyber, plus communément appelé CTI pour « *Cyber Threat Intelligence* », est l'étude des modes opératoire d'attaque (MOA). Le MOA se définit par la signature de l'attaquant, c'est-à-dire sa façon d'opérer pour cibler et attaquer ses victimes. Cela consiste généralement à étudier une attaque afin d'en extraire les différentes techniques, tactiques et procédures de l'attaquant (TTP). Celles-ci peuvent correspondre à :

— les techniques de compromission initiales : un courriel d'hameçonnage, l'exploitation d'une vulnérabilité, etc.
— l'infrastructure utilisée : la location de serveurs chez un hébergeur peu coopératif avec les autorités, la compromission de routeurs, l'utilisation de VPN, etc.
— les codes utilisés : certains MOA préféreront développer leur propre arsenal alors que d'autres peuvent se baser sur des outils open source.

Une erreur régulièrement commise est de confondre un mode opératoire d'attaque avec un « Acteur de la menace » (« *Threat Actor* » en anglais).

Un acteur de la menace est une entité physique et réelle. Il peut par exemple s'agir d'un individu, d'un groupe d'individus, d'une entreprise ou d'une unité militaire. Un MOA n'est pas une entité réelle, il s'agit ici d'un regroupement d'opérations présentant des caractéristiques techniques communes.

L'attribution est l'action qui consiste à relier un MOA avec un acteur de la menace. Par exemple, en 2025 la France a officiellement attribué le mode opératoire russe APT28 a un service du renseignement militaire russe (GRU) [2].

### 1.2 Une nouvelle tendance en 2025

Un fait majeur de l'année 2025 dans l'informatique et l'adoption des technologies basées sur l'intelligence artificielle pour le développement. Cette tendance n'a pas échappé aux attaquants informatiques où nous avons aussi pu observer cette évolution. Depuis début 2025, nous avons observé de plus en plus de scripts malveillants comportant de nombreux commentaires commneçant par un emoji, ce qui était typique des scripts générés par IA. Dans la deuxième moitié de 2025, nous avons ensuite observé des codes compilés dont nous suspectons fortement qu'ils aient été développés à l'aide d'IA. C'est le cas notamment de DeskRAT [5], un RAT associé à Transparent Tribe, un MOA supposément d'origine Pakistanaise. L'analyse des différents codes malveillants de DeskRAT soulevait pas mal d'indices allant dans ce sens. Notamment, certains codes contenaient ces noms de fonctions :

```
main.___implement_memory_protection();
main.___calculate_activation_delay();
main.___evasion_dummy_check_1();
main.___evasion_dummy_check_3();
main.___evasion_perform_dummy_computation();
main.___garbage_computation_2();
main.___execute_polymorphic_routine();
main.___garbage_string_operations();
main.___garbage_math_operations();
main.___linux_polymorphic_behavior();
```

Ces fonctions implémentent effectivement des opérations cohérentes avec leurs intitulés, mais sans effet observable sur le comportement du binaire. Entre autres, `___execute_polymorphic_routine` réalisait une transformation locale, dont le résultat n'était ensuite jamais utilisé. Ce type de « code plausible mais inopérant » peut être rapproché d'un mode de génération où l'on demanderait à un LLM : « *liste les techniques*

*d'évasion pour un malware Linux, puis implémente-les* », sans validation fonctionnelle de l'intégration dans la logique globale.

Enfin, certains échantillons contenaient des routines de *heartbeat* transmettant des valeurs de *healthcheck* manifestement artificielles et hardcodées (par exemple « `cpu=20%` » ou « `office version: 2018` », y compris dans un contexte Linux), ce qui renforce l'hypothèse d'un remplissage automatique visant davantage la vraisemblance que l'opérationnalité.

### 1.3 Problématique

L'utilisation de l'IA par les attaquants comme aide au développement de leur code pose deux principaux problèmes. Le premier problème, non traité ici, est une uniformisation dans le développement des codes utilisés par les attaquants. Cela induit une difficulté accrue dans la clusterisation des TTPs qui permet l'identification des MOA. Une deuxième problématique est la capacité pour les attaquants à développer de nouveaux codes très rapidement ce qui augmente d'autant la quantité de travail des analystes CTI. Pour répondre à cela, il devient primordial de trouver des moyens d'automatiser le travail de reverse engineering afin de ne pas être submergé par la quantité de codes malveillants à étudier.

## 2 APT28 et sa nouvelle chaîne d'infection : le déclic

APT28 (alias Fancy Bear, Bleu Delta, Sednit Group) est un groupe Russe rattaché à la Direction Générale des Renseignements (GRU) de l'État-Major des Forces Armées de la Fédération de Russie. Parmi les campagnes associées à ce groupe, nous pouvons citer le piratage des mails du Parti Démocrate Américain en 2015 ainsi que la publication de mails du parti En Marche ! dans l'entre deux tours de l'élection présidentielle française en 2017 ou le piratage de TV5 Monde. Durant les années 2010, APT28 disposait de deux codes signatures : XTunnel et XAgent. Depuis le début de la guerre en Ukraine, APT28 est connu pour accéder à des caméras de vidéosurveillance pour tracer le déplacement du matériel en Ukraine.

Depuis 2022, APT28 conduit des campagnes de phishings ciblant particulièrement l'Ukraine mais aussi ces alliés. Un fait notable est qu'APT28 utilise des équipements de bordure comme infrastructure opérationnelle pour mener ces attaques, ces équipements ayant pour avantage d'être souvent peu supervisés et non mis à jour. Ce MOA a alors privilégié l'utilisation d'implants que l'on pourrait qualifié d'être « à usage unique » : ils

sont souvent développés dans des langages interprétés tels que JavaScript, PowerShell ou de simples scripts batch (.bat), créés spécifiquement pour chaque campagne. Ces implants ne sont pas sophistiqués et se limitent le plus souvent à une tâche unique : établir un tunnel, récupérer et exfiltrer des mots de passe, etc.

Depuis fin 2024, APT28 utilise une nouvelle chaîne d'infection composée d'implants plus évolués que nous avons pu analyser en 2025. Cette chaine d'infection (schématisée dans la figure 1) a fait l'objet de plusieurs publication, que ce soit, le CERT-UA, ESET ou nous même [1, 3, 4].

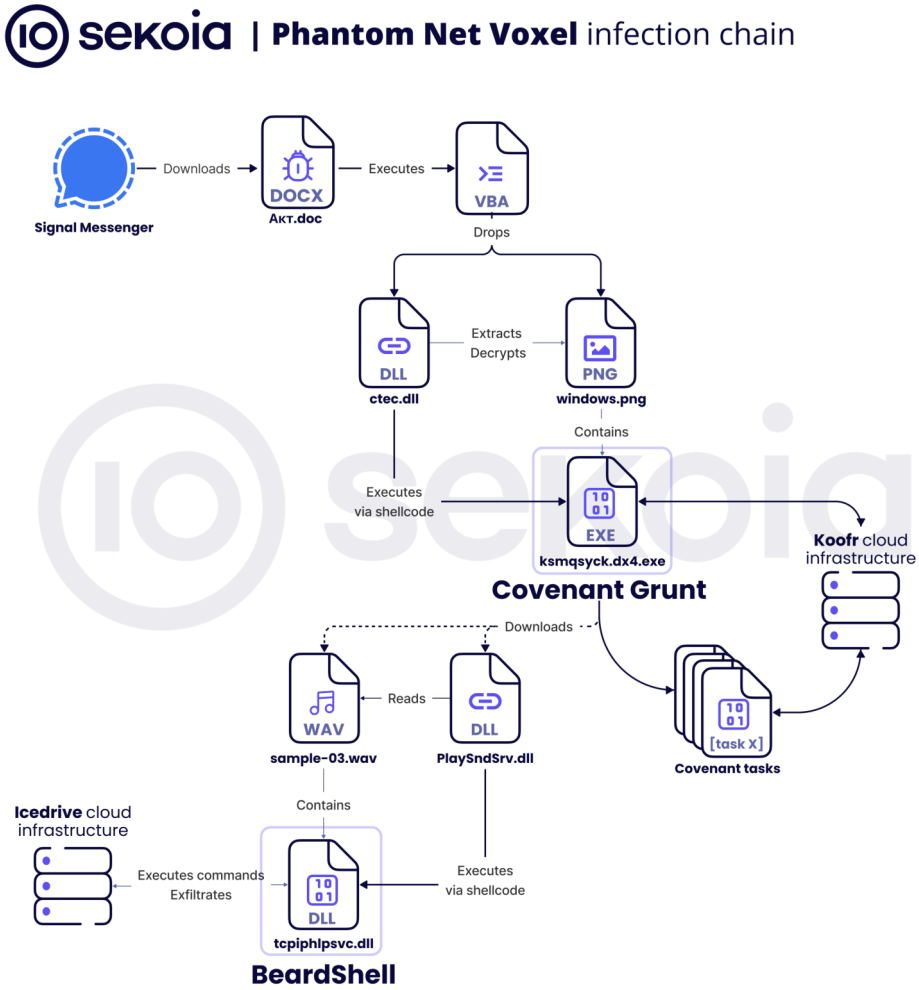


**Fig. 1.** Schéma de la nouvelle chaine d'infection d'APT28

L'accès initial est un document Word envoyé via Signal. Ce document est volontairement flouté afin de forcer l'utilisateur à activer les macros afin de rendre ce document lisible. En arrière plan, ces macros déposent une DLL, nommée Koala Loader, et une image.

**Koala Loader**

Koala Loader est un Loader développé en C/C++. Ce code persiste en utilisant une technique de *COM Hijacking* pour être exécuté à chaque ouverture de session. Après avoir mis en place sa persistance, ce loader a pour particularité d'extraire d'une image (comme celles présentées dans la figure 2), un shellcode chiffré.

**Fig. 2.** Exemples d'images contenant un shellcode chiffré

**Covenant**

Le *Shellcode* extrait de l'image initiale initialise le *Common Language Runtime* afin d'exécuter un module .NET. Ce module .NET est un implant issu du *framework* Covenant. Ce « grunt » (nom donné aux implants Covenant) dispose de quelques méthodes d'obfuscation : la *randomisation* des noms de fonctions et variables ainsi que le chiffrement des chaînes de caractères (composée de base64 et d'un XOR).

Ce *grunt* communique avec le C2 via un service légitime de partage de fichiers. Deux services ont été observé en 2025, Koofr puis Filen. Le rôle de ce *grunt* est de récupérer des modules additionnels via le C2, de les exécuter et de renvoyer les résultats.

**Beardshell**

Un autre code important dans cette chaîne d'infection est BeardShell. Bien que nous n'ayons pas pu étudier son déploiement, l'une des méthodes utilisées pour le déployer est via le Covenant. BeardShell est un code développé en C++ dont le rôle est d'exécuter des commandes ou scripts

PowerShell. Comme dans le cas de Covenant, ceux-ci sont reçu via un service de partage légitime de fichier, Icedrive.

**Slimagent**

Le dernier associé à cette chaîne d'infection est Slimagent. Ce code développé en C/C++ ne communique pas avec un C2. Son rôle est uniquement de réaliser à intervalles réguliers des captures des frappes claviers ainsi qu'une capture d'écran de la fenêtre active. Le résutlat de ces opérations est stocké dans un fichier HTML chiffré.

## 3 Du ctrl+c/ctrl+v à l'automatisation de l'analyse

Ces dernières années, de nombreux analystes ont constaté que les grands modèles de langage (LLM) étaient capables d'expliquer efficacement du pseudocode décompilé, voire de l'assembleur. Dans la pratique, cela a parfois réduit le travail d'analyse à une simple médiation entre un désassembleur/décompilateur (IDA, Ghidra) et un chatbot. Les résultats obtenus via l'assistance par LLM se sont révélés particulièrement convaincants, notamment en termes de gain de temps et de précision.

### 3.1 Analyse de Covenant

Cette observation a favorisé l'émergence de plugins visant à supprimer la phase de copier/coller, en intégrant directement des appels aux LLM au sein des outils de *reverse engineering* (GPThidra, Gepetto, etc.). Le flux d'interaction s'est ainsi déplacé de « ctrl+c / ctrl+v » vers des actions natives du type « clic droit -> send to LLM ». Cette problématique du « copier coller » a été vécue lors de l'analyse de Covenant.

La figure 3 présente un extrait du code de cet implant.

Analyser ce code manuellement n'est en soi pas compliqué, mais prend juste un temps important :

— toutes les chaines de caractères doivent être identifiées puis déchiffrées.

— les fonctions doivent ensuite être renommées. Le .NET étant assez verbeux, cela n'est pas compliqué mais prend juste du beaucoup de temps.

La première étape peut être automatisée, par exemple via un script Python utilisant la lib pythonnet [7]. Pour ce qui est du renommage de fonctions, c'est typiquement quelque chose que les chatbots savent faire, mais ici nous manquions d'outil pour faire l'interface entre le code .NET décompilé et le chatbot.

```
// Token: 0x06000008 RID: 8 RVA: 0x0000239C File Offset: 0x0000059C
public void CCDLKA785C()
{
    try
    {
        string text = L2WSH3CNPJ.W8Y70HMX0S("SmEpdTdyJQ==").Replace(Environment.NewLine, L2WSH3CNPJ.W8Y70HMX0S("Ow=="));
        string text2 = L2WSH3CNPJ.W8Y70HMX0S("SmEpdTdyJQ==").Replace(Environment.NewLine, L2WSH3CNPJ.W8Y70HMX0S("Ow=="));
        string text3 = L2WSH3CNPJ.W8Y70HMX0S("CTRlPH8iaXRuNw==");
        string text4 = L2WSH3CNPJ.W8Y70HMX0S("dD02MgMMO242C0ccGz8mAisSFxpbOho0DQ8JCDoldDIjITlzby1uJgljYGQ=");
        string text5 = PJ8YF3UG0C.EID5TXAGCS(text4);
        PJ8YF3UG0C.B5OOGU8HM3 b5OOGU8HM = new PJ8YF3UG0C.B5OOGU8HM3(text4);
        PJ8YF3UG0C.EROYOYIZ6E eroyoyiz6E = new PJ8YF3UG0C.EROYOYIZ6E();
        ICryptoTransform cryptoTransform = b5OOGU8HM.RI89Q0UVYZ();
        HMACSHA256 hmacsha = new HMACSHA256(b5OOGU8HM.KELYRW0EEX());
        Thread.Sleep(TimeSpan.FromSeconds(2.0));
        RSACryptoServiceProvider rsacryptoServiceProvider = new RSACryptoServiceProvider(2048, new CspParameters());
        byte[] bytes = Encoding.UTF8.GetBytes(rsacryptoServiceProvider.ToXmlString(false));
        byte[] array = b5OOGU8HM.JWI0CQADMU(cryptoTransform, bytes, 0, bytes.Length);
        byte[] array2 = hmacsha.ComputeHash(array);
        string text6 = L2WSH3CNPJ.W8Y70HMX0S("HzM9Nw==");
        eroyoyiz6E.UJ94HG04AM = text3 + text5;
        eroyoyiz6E.T8C92T8KYH = L2WSH3CNPJ.W8Y70HMX0S("AQ==");
        eroyoyiz6E.BGBBMOD3VF = L2WSH3CNPJ.W8Y70HMX0S("");
        eroyoyiz6E.K8ETEVOS2Y = Convert.ToBase64String(b5OOGU8HM.AE9H66FC77());
        eroyoyiz6E.NQGM7CP4NU = Convert.ToBase64String(array);
        eroyoyiz6E.ODQOSVMYB1 = Convert.ToBase64String(array2);
```

**Fig. 3.** Extrait du code décompilé de Covenant

Durant cette analyse, la plus value de l'analyste s'est avérée assez limitée puisqu'elle a surtout consisté à être « l'assistant du LLM », ce qui a été vécu comme une perte de temps.

### 3.2 Analyse de slimagent

La suite logique consiste à inverser la perspective : plutôt que d'augmenter les outils d'analyse statique par des plugins dédiés, il devient pertinent de fournir aux LLM un accès direct à ces outils, puis de leur confier des instructions de pilotage de l'analyse. Depuis l'introduction de la spécification MCP (*Model Context Protocol* — un standard uniformisant l'exposition d'outils externes à un LLM), il est possible d'exposer des outils à un modèle via un format commun, y compris des outils de *reverse engineering*. Cela ouvre la voie à des interactions où l'utilisateur formule directement des objectifs d'analyse dans le chatbot, par exemple : « dans main, décompiler les fonctions appelées, les renommer, et décrire leur rôle en commentaires », voire même : « les fonctions `FUN_1`, `FUN_2` et `FUN_3` réalisent respectivement la désobfuscation, le déchiffrement et le chargement en mémoire du stage suivant ; produire un script Python permettant de récupérer ce stage en clair ».

Un cas d'usage marquant a été l'analyse de Slimagent, un outil attribué à APT28, dédié à la collecte d'informations (keylogging, captures d'écran). Les capacités étaient relativement simples à identifier, mais le format de sortie s'est avéré difficile à rétroconcevoir. Slimagent utilise en effet un schéma de chiffrement imbriqué : la clé de session AES est chiffrée

via RSA-2048 (PKCS#1 v1.5) et sérialisée au format SIMPLEBLOB de la CryptoAPI Windows ; les données collectées sont ensuite chiffrées en AES-256-CBC avec padding PKCS#7, et leur intégrité est vérifiée via un HMAC-SHA-256. La tâche a alors été entièrement déléguée à Claude Desktop :

« Produit un script qui génère et patche une clé RSA au bon format et à la bonne taille dans Slimeagent, ainsi qu'un script permettant de déchiffrer le fichier d'exfiltration avec cette clé. »

L'objectif a été atteint avec succès, suggérant que, dans ce contexte, le LLM pouvait surpasser notre analyse sur des aspects précis de compréhension de fonctions et d'implémentations. Nous avons ensuite tenté de confier à un LLM la conduite complète de l'analyse, du début à la fin. Les résultats se sont révélés insatisfaisants pour plusieurs raisons récurrentes :

- **Initialisation de l'analyse** : le modèle éprouve des difficultés à choisir un point d'entrée pertinent. Or, « par où commencer » constitue déjà une étape critique, y compris pour des analystes expérimentés.
- **Planification et stratégie** : le modèle peine à organiser une séquence d'étapes cohérente et, surtout, à ré-évaluer dynamiquement la pertinence de la direction prise au cours de l'investigation.
- **Définition du résultat attendu** : en fonction du type de malware (backdoor, loader, rootkit, data collection, etc.), le modèle a du mal à expliciter l'objectif final de l'analyse (extraction d'IOC, déchiffrement d'un stage, identification des capacités, etc.).
- **Gestion du contexte** : lorsque trop de fonctions sont explorées, le modèle perd le fil, produit des approximations, et devient moins apte à synthétiser le comportement global.

## 4 SARA : Semi Autonomous Reverse Analyst

Malgré ces limites, il apparaît qu'un LLM puisse réussir la plupart des étapes d'une analyse à condition que l'orchestration et les instructions soient structurées. Cela nous a conduits à concevoir une couche d'orchestration basée sur la bibliothèque LangGraph. Le schéma global de la solution est présenté dans la figure 4.

### 4.1 Orchestration globale

L'orchestration est réalisée par LangGraph qui permet la définition des nœuds et de leurs enchaînements. La Figure 1 présente la définition

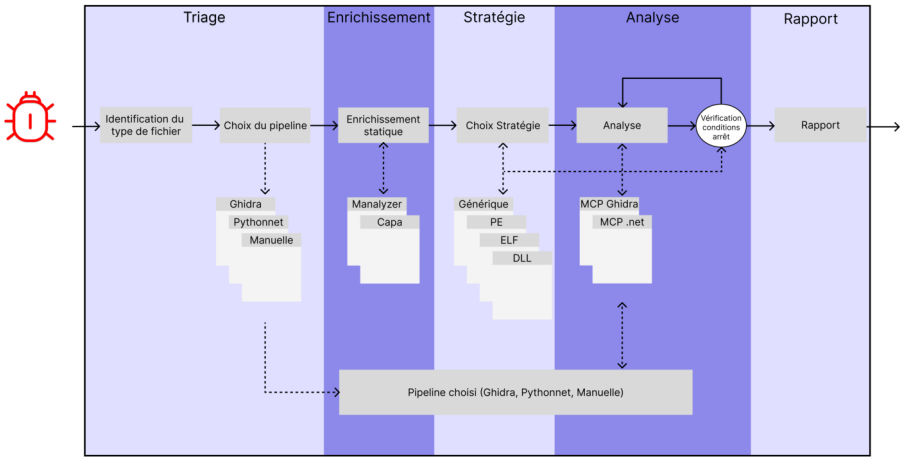


**Fig. 4.** Schéma initial du projet SARA

et les enchaînements des différents nœuds : *triage*, *enrichment*, *strategy*, *analysis*, *report*.

Listing 1: Extrait de code définissant le workflow d'un analyste

```
# Create graph
workflow = StateGraph(AnalysisState)

# Add nodes
workflow.add_node("triage", triage_node)
workflow.add_node("enrichment", enrichment_node)
workflow.add_node("strategy", strategy_node)
workflow.add_node("analysis", analysis_node)
workflow.add_node("report", report_node)

# Define edges (linear workflow for now)
workflow.add_edge("triage", "enrichment")
workflow.add_edge("enrichment", "strategy")
workflow.add_edge("strategy", "analysis")
workflow.add_edge("analysis", "report")
workflow.add_edge("report", END)

# Set entry point
workflow.set_entry_point("triage")
```

Les nœuds se répartissent en trois catégories :

— « **Nœuds déterministes** » : Enrichissement, pas de LLM, juste une tâche implémentée.

— **« Nœuds requête/réponse simple »** : *Triage*, *strategy* et *report*, avec un prompt spécifique contenant des instructions envoyées au LLM qui renvoie une réponse formatée.
— **« Nœuds ReAct »** : *Analysis*, une boucle où le LLM choisit un outil, l'exécute, analyse le résultat et recommence.

**Triage** Ce premier nœud de triage identifie le type de fichier. En fonction du résultat, un LLM est utilisé pour sélectionner le pipeline à utiliser. Le pipeline correspond à l'outil de base qui sera utilisé pour l'analyse. Actuellement trois pipelines peuvent être utilisés :

— **« Ghidra »** : utilisé pour les binaires nécessitant une décompilation
— **« Pythonnet »** : outil développé en interne utilisant la bibliothèque pythonnet pour analyser des codes développés en .NET.
— **« Manuel »** : ici, aucun outil n'est utilisé.

D'autres pipelines peuvent être facilement développés suivant les besoins. Pour cela, il faut au minimum les éléments suivants :

— Un conteneur Docker avec un serveur MCP (FastMCP) exposant les outils d'analyse
— Un enregistrement dans SARA (client HTTP et entrée pipeline dans le registre qui list les services disponible)
— Une stratégie (fichier texte) guidant le LLM pendant l'analyse

Pour le service Ghidra, bien que de nombreux serveurs MCP existent en open source, ils n'étaient pas compatibles avec la version headless de Ghidra. Un serveur MCP pour Ghidra headless a donc dû être redéveloppé. L'implémentation actuelle propose 29 outils, mais en pratique, le service pourrait fonctionner avec seulement deux outils :

— `ghidra_load_binary` : pour charger le binaire dans Ghidra
— `ghidra_decompile_function` : pour récupérer le pseudocode décompilé d'une fonction

Certains outils ont tendance à être sur-utilisés par le LLM lorsqu'ils sont disponibles. C'est notamment le cas des outils de recherche de chaînes de caractères : lorsque le LLM ne sait plus comment progresser dans l'analyse, il a tendance à rechercher des chaînes génériques susceptibles d'apparaître dans un malware, comme `cmd.exe`, `https`, etc. Pour limiter ce comportement, il est important de préciser les règles d'usage directement dans le champ `description` de l'outil dans la spécification MCP. Ce champ constitue le vecteur principal par lequel le LLM reçoit des instructions sur la façon dont un outil doit ou ne doit pas être utilisé. Les stratégies d'analyse constituent un second vecteur possible, mais en pratique les

LLM semblent mieux respecter les contraintes définies dans le champ `description` de l'outil.

Concernant l'analyse de codes .NET l'outillage existant est assez limité et repose principalement sur des outils comme ILspy ou dnSpyEx. Ces outils permettent d'avoir accès au code intermédiaire ou décompilé mais ne sont pas très pratique dès qu'il s'agit de modifier le nom des fonctions ou rajouter des commentaires. De plus, à notre connaissance ils ne sont pas aussi extensibles que des outils comme IDA Pro ou Ghidra. N'ayant pas besoin d'interface graphique et ayant déjà une base de code utilisé pour le déchiffrement des chaines de caratères de Covenant, nous avons donc choisi de développer notre propre service, RePythonNET-MCP [6]. Ce service est développé en Python, utilise dnlib pour accéder au code intermédiaire et aux bibliothèques de dnSpyEx / ILspy pour récupérer le code décompilé. Ce service expose une trentaine d'outils via MCP permettant de lister les classes, méthodes, décompiler ou récupérer le code intermédaire, etc.

**Enrichissement** Ce nœud réalise la collecte d'indices permettant d'orienter l'analyse. Actuellement deux outils sont utilisés : Capa et Manalyze. Ce nœud est entièrement statique et ne fait pas appel à un LLM.

**Stratégie et objectifs** Cette étape est une première analyse des résultats obtenus afin de choisir la bonne stratégie d'analyse. En effet, l'objectif et la méthodologie d'analyse d'un code diffère suivant qu'il s'agisse d'un exécutable, un PE, un Loader ou un shellcode par exemple. Pour cette étape, le choix de la stratégie est réalisé par un LLM et peut être influencé par l'analyste lors de la soumission du fichier. Plusieurs stratégies ont préalablement été définies :

- `generic_binary` : stratégie générique pour les binaires dans Ghidra. Son rôle est de préparer le travail d'un analyste en décompilant les fonctions et renommant les fonctions et variables dont le rôle est clairement identifié
- `elf_analysis` : stratégie pour les fichiers ELF
- `pe_analysis` : stratégie pour les fichiers PE
- `dll_analysis` : stratégie spécifique pour les DLL

**Analyse** Le nœud « *analyze* » correspond à l'analyse en tant que telle du code malveillant. Celle-ci est pilotée par un LLM et guidée par la stratégie sélectionnée dans le noeud précédent. Ce nœud a accès aux serveurs MCP mis à disposition. Deux serveurs MCP sont actuellement utilisés :

— Ghidra Headless MCP : Un serveur MCP Ghidra en mode headless afin de pouvoir être hébergé sur une infrastructure interne à l'équipe.
— Pythonnet MCP : Un serveur MCP utilisant pythonnet dont le but est d'aider l'analyse des codes .NET. Les fonctionnalités actuelles sont : le listing, la décompilation et le renommage des classes et méthodes.

La stratégie utilisée peut définir les conditions d'arrêt de l'analyse. Deux cas peuvent généralement provoquer l'arrêt de l'analyse :

— lorsque le nombre maximum d'itérations (échanges entre le LLM et le MCP) est atteint
— lorsque les conditions de succès sont réunies.

**Rapport** Le dernier nœud, « rapport », est chargé de produire un rapport structuré à destination de l'analyste. Ce nœud fait appel à un LLM, guidé par un prompt spécifique imposant un format de sortie précis. En particulier, pour limiter les hallucinations, le LLM est contraint de fournir une preuve pour chaque capacité identifiée. Par exemple : « *FUN_xyz* implémente la capacité X via l'appel à *CreateMutexA* ».

### 4.2 Contenu d'une stratégie

Les stratégies sont comparables aux *system prompts* spécialisés, *skills* ou aux *prompt templates* que l'on trouve dans d'autres *frameworks* d'agents : elles fournissent au LLM le savoir-faire métier nécessaire pour conduire l'analyse de manière structurée. Le terme *stratégie* a été retenu pour refléter leur rôle de guide méthodologique.

1. Un en-tête définissant les cas où cette stratégie doit être utilisée.

```
# Strategy: Windows DLL Analysis

**Best for:** Windows Dynamic Link Libraries (.dll)
**File type:** Portable Executable (PE) - DLL
**Entry points:** DllMain + Exported Functions
```

2. La philosophie de l'analyse

```
## Analysis Philosophy
DLLs have multiple entry points: DllMain for initialization
↪  and exported functions as APIs.
Focus on DllMain and exports - these are where the real
↪  logic lives.
```

3. Une méthode permettant de suivre l'évolution de l'analyse. Par exemple, dans le cas de l'analyse de DLL, trois données sont suivies : les fonctions exportées, les capacités identifiées par Capa et les fonctions décompilées.

```
### Tracked Data:
1. **Exported Functions** - From ghidra_get_entry_points() → ALL must be analyzed
2. **CAPA Capabilities** - From enrichment → ALL must be mapped to functions
3. **Function Analysis** - Every decompile tracked → Progress displayed in real-time
```

4. La partie suivante définit les phases d'analyse attendues. Par exemple, dans le cas d'une DLL, il est attendu que :
   (a) chaque fonction exportée ait été analysée
   (b) chaque capacité identifiée par Capa doit avoir été associée à une fonction
   (c) chaque fonction détectée comme suspecte par Manalyze doit avoir été analysée.

5. Enfin, des conditions de succès sont définies.

```
## Success Criteria (Verified Deterministically)

The analysis is complete when ALL of these are TRUE:

- [SUCCESS] **Entry point analyzed** (DllMain or entry)
- [SUCCESS] **ALL exported functions decompiled** (exact count: N/N)
- [SUCCESS] **ALL capa capabilities mapped** (exact count: N/N)
- [SUCCESS] **All suspicious imports analyzed** (if any were detected)

**How verification works:**
- NOT based on LLM guessing
- Uses exact counts from trackers
- Shows specific missing items: "DllMain, ProcessData not analyzed"
- No false positives possible

```

De façon optionnelle, il est aussi possible de d'aider le LLM en lui proposant des outils MCP spécifiques à utiliser à certaines étapes, les

erreurs courantes à ne pas commettre ou comment interpréter certaines lignes de codes.

### 4.3 Mise en oeuvre de SARA

L'objectif de SARA n'est pas (encore) de remplacer un analyste mais d'accélérer le travail de celui-ci. SARA est donc déployé depuis le début d'année sur une infrastructure interne de l'équipe Threat Detection & Research (TDR) de Sekoia.io afin que chaque analyste de l'équipe puisse y accéder.

Ce service est utilisé par :

— des non-experts en analyse de code malveillants : ces analystes avaient besoin de l'aide d'un *reverser* pour comprendre les codes, même les codes simples. L'utilisation de SARA permet à ces analystes d'être autonome dans ce cas.

— des analystes expérimentés en analyse de codes malveillants : SARA est utilisé afin d'avoir une première vision du binaire. En fonction des résultats, il peut demander à SARA d'approfondir l'analyse ou aller vérifier les résultats et se concentrer sur les parties les plus intéressantes du binaire et celles qu'une analyse automatique n'est pas encore capable d'analyser correctement.

Une interface graphique permet de téléverser un fichier ainsi que de configurer son analyse en définissant le LLM à utiliser, le nombre d'itérations et le contexte d'utilisation du fichier pour orienter l'analyse du fichier (figure 5).

L'utilisateur a ensuite la possibilité d'observer la progression de l'analyse et de dialoguer avec le LLM à la fin de l'analyse. Cela peut permettre à l'utilisateur de demander de reprendre l'analyse, approfondir certains aspects de celle-ci, etc (figure 6).

### 4.4 Modèles et tokens

Un aspect important du développement de SARA était de trouver un équilibre entre coût et précision d'une analyse. Dans un contexte où les capacités des modèles évoluent rapidement et où les prix des tokens fluctuent considérablement, tant entre fournisseurs (OpenAI, Anthropic, Gemini) qu'entre les différents modèles d'un même fournisseur (claude-haiku, claude-sonnet, etc.) nous avons choisi d'utiliser LiteLLM comme couche d'abstraction entre la partie analytique de SARA et les APIs des LLM. Cette approche permet de changer de modèle ou de fournisseur sans

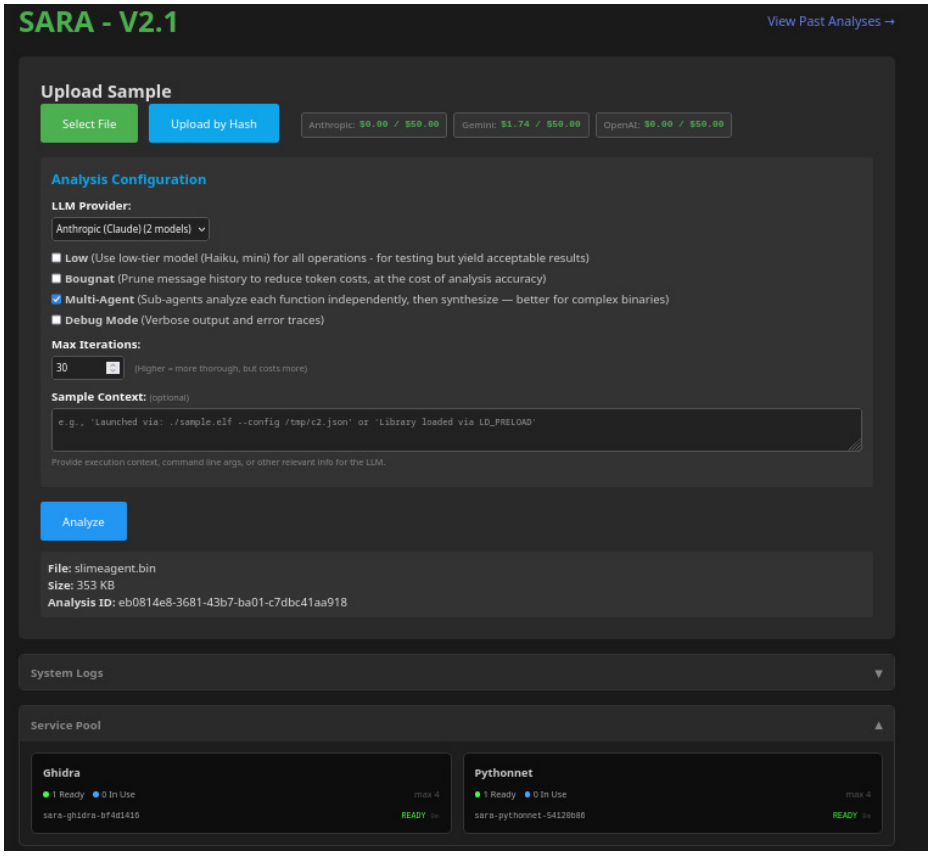


**Fig. 5.** Interface web de SARA permettant de téléverser un code et configurer son analyse

modifier le code, ce qui est essentiel dans un domaine où il est difficile d'anticiper les prochaines évolutions.

Une première remarque concerne l'écart considérable entre le coût des tokens dans le cadre d'un abonnement comme Claude Pro et celui de l'API. Un abonnement permet, via un chatbot connecté à Ghidra par un serveur MCP, de réaliser une analyse d'une centaine de millions de tokens sans coût additionnel. Tenter de reproduire un fonctionnement similaire via l'API génèrerait des coûts importants, potentiellement plusieurs dizaines d'euros par sample. En cause : le fonctionnement même des LLM. Contrairement à ce que suggère l'interface conversationnelle, un LLM n'a pas de mémoire entre les échanges. À chaque appel, l'intégralité de l'historique est retransmise au modèle pour reconstituer le contexte. Ainsi, à chaque itération d'analyse, le volume de tokens traités croît de

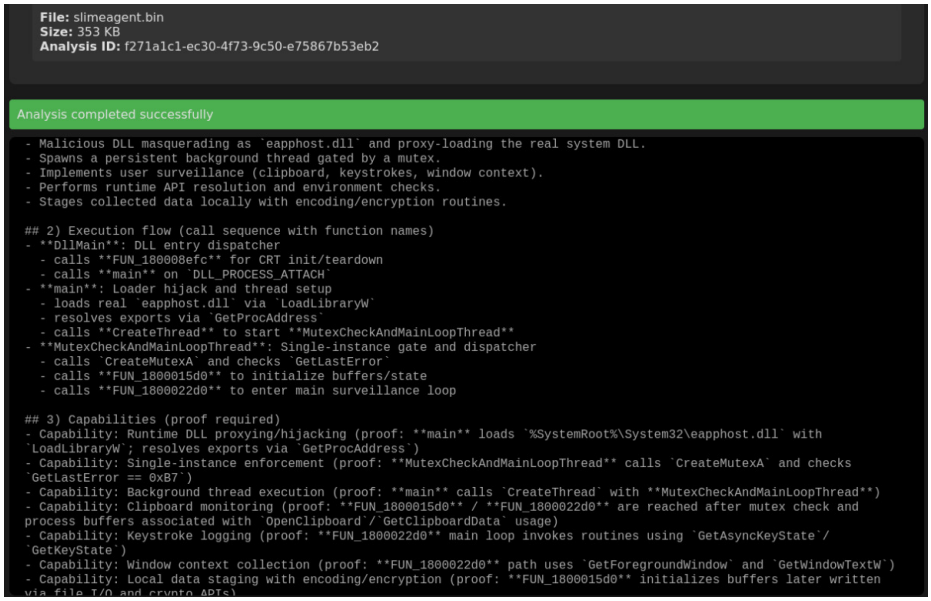


**Fig. 6.** Interface web de SARA présentant le rapport

manière cumulative : la centième itération consomme cent fois plus de tokens de contexte que la première.

Chaque fournisseur propose plusieurs niveaux de modèles. Dans SARA, deux niveaux sont utilisés : un modèle performant (GPT-4o, claude-sonnet, etc.) et un modèle moins coûteux (GPT-mini, claude-haiku, etc.). Malgré leur coût moindre, les modèles « légers » sont adéquats pour la sélection de pipeline. Ils se révèlent également étonnamment efficaces sur l'analyse d'une fonction unique lorsqu'on leur fournit le pseudocode décompilé. Ils sont en revanche moins performants sur des tâches complexes ou nécessitant un contexte étendu. Deux modes ont été développés pour le nœud d'analyse de SARA :

1. **Mode agent unique** : un seul contexte, avec un fonctionnement linéaire similaire à un chatbot connecté à un serveur MCP. L'analyse suit la stratégie, mais le contexte croît rapidement. Pour limiter les coûts, un nombre maximum d'itérations force l'arrêt de SARA, ce qui peut s'avérer problématique sur des samples complexes nécessitant l'analyse d'un grand nombre de fonctions. Ce mode est en revanche très efficace sur des samples comme les loaders ou les backdoors minimalistes. Avec un maximum de 30 itérations — une itération correspondant non pas à une fonction isolée mais à

un groupe de fonctions analysées ensemble, le coût se situe entre 1 et 3 euros avec claude-sonnet. Augmenter ce seuil de seulement 10 itérations fait croître le coût de manière significative, pour les raisons évoquées précédemment.

2. **Mode multi-agents** : composé de trois agents disposant chacun d'un ensemble distinct d'outils du serveur MCP :
   (a) **Coordinator** (modèle léger, boucle ReAct) : le navigateur. Il explore le binaire en appelant les outils Ghidra (`get_entry_points`, `get_called_functions`, `get_xrefs_to`) et dispatche les fonctions au *Classifier* via `classify_function`. Il ne lit jamais le code décompilé directement, il ne voit que les noms de fonctions et les verdicts retournés. Lorsqu'un verdict *interesting* est reçu, les fonctions appelées sont récupérées automatiquement de manière déterministe et ajoutées à la file d'exploration. Il maintient le graphe d'appels en mémoire.
   (b) **Classifier** (modèle léger, appel unique, contexte frais) : l'analyste par fonction. Instancié à chaque appel de `classify_function` avec un contexte vierge, il reçoit le code décompilé, le contexte de l'appelant, le code des fonctions appelées et les résultats CAPA. Il retourne un verdict JSON (*interesting* / *boring* / *standard_lib*) accompagné d'un résumé, des IOCs identifiés et des cibles d'extraction. Sans outils ni historique, il ne connaît rien des autres fonctions du binaire — c'est précisément ce qui lui permet de passer à l'échelle : 30 fonctions correspondent à 30 appels indépendants, chacun avec un contexte réduit, soit environ 3 000 tokens en moyenne par appel.
   (c) **Synthesis** (modèle performant, appel unique) : le rédacteur final. Il reçoit l'ensemble des rapports du *Classifier* et produit une analyse unifiée reliant les fonctions entre elles. C'est le seul agent disposant d'une vision globale du binaire, et le seul à utiliser un modèle performant, c'est à cette étape que se concentre le raisonnement de haut niveau.

   Le mode multi-agents impose quelques contraintes : certains outils doivent nécessairement être disponibles dans le serveur MCP de la pipeline associée. En contrepartie, les coûts sont nettement inférieurs à ceux du mode agent unique, moins de 0,50€ par analyse contre 1 à 3€, et surtout, ils restent stables quelle que soit la complexité du sample. Là où le mode agent unique voit ses

coûts croître rapidement avec le nombre de fonctions à analyser, le mode multi-agents conserve un coût quasi constant : analyser deux fois plus de fonctions ne double pas le coût, car chaque appel au Classifier opère sur un contexte indépendant et de taille fixe. Les resultats de ce mode semble cependant etre plus variable d'une analyse a l'autre. Ce mode est encore en developpement actif, et remplacera probablement le mode agent unique pour l'analyse de binaires.

Nous avons uniquement testé trois fournisseurs occidentaux : Anthropic, OpenAI et Gemini. Les ratios de prix entre tokens d'entrée et de sortie varient selon les fournisseurs, mais au moment de l'écriture de cet article, les tokens Anthropic sont environ 30% plus chers que ceux de Gemini, pour une efficacité comparable. Une prochaine étape sera de tester des modèles nettement moins coûteux, comme GLM, qui semble très prometteur. Pour ceux qui en ont la possibilité, l'hébergement local de modèles pourrait également constituer une solution financièrement viable, en fonction du volume d'analyses à effectuer. La conclusion de cette partie est que l'automatisation de l'analyse d'un sample est relativement simple à implémenter. Les modèles actuels sont excellents pour analyser des fonctions décompilées, voire directement de l'assembleur. La question du coût des tokens est centrale dans l'implémentation de solutions de reverse engineering automatisé, d'autant plus que les prix évoluent rapidement entre fournisseurs et entre générations de modèles.

### 4.5 Résultats

Dans l'ensemble, les résultats renvoyés par l'outil sont bons et nous font gagner un temps d'analyse considérables.

Par rapport aux codes de la chaine d'infection, l'outil s'en sort très bien sur Covenant, Slimagent et KoalaLoader pour un coût d'analyse inférieur à 1 euro :

— Slimagent et KoalaLoader sont codés en C/C++ et la compréhension du code ne pose pas de problèmes particuliers.

— Covenant, codé en .NET, dispose d'une couche d'obfuscation. Il se trouve que le décodage des strings (base64 et XOR) est alors géré par SARA ce qui permet d'automatiquement ressortir le C2 utilisé

Un exemple de résultat de l'analyse de Slimagent est donné dans l'annexe A.

Concernant BeardShell, les résultats sont moins probants, probablement du fait qu'il est codé en C++ et que son analyse nécessite une phase

importante de travail préparatoire pour reconstruire les classes. SARA n'est alors pas probant sur ce code. Les résultats sur ce code sont assez similaires à ceux observés sur des codes avec du « *Control Flow Flattening* » (CFF). Ces résultats consistent alors juste à identifier les fonctionnalités du code.

## 5 Conclusion

L'analyse du mode opératoire APT28 en 2025 nous a fait comprendre que l'utilisation de LLM pour le *reverse* de *malware* n'était pas qu'un simple objet marketing mais qu'il pouvait dès à présent être utile pour aider à l'automatisation des analyses travail. De plus, les attaquants utilisant eux aussi de plus en plus les LLMs pour développer de nouveaux implants, il nous est apparu nécessaire d'automatiser au maximum nos analyses.

Nous avons donc développés SARA, un outil d'automatisation du *reverse* dont le but était de rajouter une couche d'orchestration au dessus des appels aux serveurs MCP. Cette couche permet de diriger les analyses suivant différentes stratégies et objectifs définis en fonction du contexte.

Nous avons donc confronté notre approche à la chaîne d'infection d'APT28 qui a pour intérêt d'être diverse et d'actualité. Les résultats montrent que pour des codes en C ou .NET, notre approche permet d'identifier rapidement les TTPs du *malware* et les fonctions associées. Pour des codes plus complexes, ou comportant des méthodes d'obfuscation avancés, SARA reste encore limité.

## A Résultat de SARA sur Slimagent

Cette annexe présente le résultat d'analyse de SARA sur Slimagent (hash MD5 : 89b83d375a0fb00670af5276816080e). Le coût total de cette analyse s'est élevé à $0,30, dont $0,24 pour le modèle léger (coordinateur et classificateurs, 792,091 tokens en entrée et 27,005 en sortie) et $0,06 pour le modèle performant (synthèse, 4,960 tokens en entrée et 4,479 en sortie).

### A.1 Summary

The malware operates as a malicious DLL proxy for eapphost.dll, likely used for persistence or privilege escalation. It functions as a comprehensive information stealer, featuring keylogging, clipboard monitoring, and screen

capture capabilities. Stolen data is formatted into HTML (with Base64 encoded JPEGs), encrypted using RSA and AES-256, and staged in the It employs obfuscated dynamic API resolution and basic anti-debugging techniques (INT 3) to evade detection. Execution is restricted to a single instance using a hardcoded mutex.

### A.2 Execution flow

- init_eapphost_proxy_and_explorer_thread : Initializes the DLL proxy for eapphost.dll and checks if running under explorer.exe. calls check_mutex_and_spawn_thread for single-instance enforcement and payload initialization.
- check_mutex_and_spawn_thread : Checks for the mutex hey4kmr8oj46n45n3p.
  - calls spawn_worker_thread_and_write_file to spawn background worker threads for data collection.
- monitor_and_log_activity_loop : Acts as the primary surveillance loop.
  - calls keylogger_process_input for keystroke capture.
  - calls monitor_clipboard_and_process_content for clipboard theft.
  - calls monitor_active_window_title for activity tracking.
  - calls capture_screen_to_base64_html for screenshot exfiltration.
- write_buffer_to_file_loop : Periodically flushes collected, encrypted data to disk.
  - calls generate_temp_log_filepath to create staging files in the %TEMP% directory.

### A.3 Capabilities

- DLL Proxying (proof : init_eapphost_proxy_and_explorer_thread proxies eapphost.dll and targets explorer.exe)
- Single-instance execution (proof : check_mutex_and_spawn_thread checks for the mutex hey4kmr8oj46n45n3p)
- Keylogging (proof : keylogger_process_input polls keyboard state and maps virtual keys like [BKSP] and [TAB])
- Clipboard monitoring (proof : monitor_clipboard_and_process_content accesses clipboard data)

— Screen capture (proof : capture_screen_to_base64_html captures screen content and encodes to Base64)
— Anti-debugging (proof : init_eapphost_proxy_and_explorer_thread utilizes INT 3 instructions)
— Data exfiltration staging (proof : write_buffer_to_file_loop writes encrypted data to %TEMP%)

### A.4 Key functions

— init_eapphost_proxy_and_explorer_thread : Entry point ; handles DLL proxying and environment checks.
— check_mutex_and_spawn_thread : Enforces single-instance execution via mutex and initiates worker threads.
— spawn_worker_thread_and_write_file : Manages background thread creation and file I/O operations.
— monitor_and_log_activity_loop : Main surveillance loop orchestrating data collection.
— keylogger_process_input : Captures and maps keyboard input.
— monitor_clipboard_and_process_content : Monitors and extracts clipboard contents.
— monitor_active_window_title : Tracks the active window title for user activity monitoring.
— capture_screen_to_base64_html : Captures screenshots and formats them into HTML/Base64.
— write_buffer_to_file_loop : Handles periodic flushing of encrypted data to disk.
— generate_temp_log_filepath : Generates file paths in the %TEMP% directory for staging.

### A.5 Notes / uncertainties

— The specific RSA and AES-256 implementation details were not fully reversed, though the presence of an RSA key blob was confirmed.
— The dynamic API resolution mechanism was identified but the specific resolver function was not explicitly named in the provided corpus.
— No network communication functions were identified in the provided analysis, suggesting exfiltration might be handled by a separate module or secondary stage.

### A.6 Extraction / Decryption Scripts

### A.7 what's missing from this analyse

— The specific implementation of the RSA/AES encryption routines needs to be analyzed to confirm the encryption scheme.
— The network communication module is missing ; the sample likely relies on a secondary stage or C2 configuration not present in this DLL.

### A.8 Threat assessment

The malware is a persistent information stealer that uses DLL proxying to hide within legitimate system processes. It actively monitors user activity, including keystrokes and screen content, for exfiltration. The use of encryption and anti-debugging techniques indicates a sophisticated attempt to evade detection and analysis.

## Références

1. CERT-UA. UAC-0001 (APT28) cyberattacks against government agencies using BEARDSHELL and COVENANT. https://cert.gov.ua/article/6284080, 2025.
2. France Diplomatie. Russie - Attribution de cyberattaques contre la France au service de renseignement militaire russe (APT28). https://www.diplomatie.gouv.fr/fr/presse-et-ressources/decouvrir-et-informer/actualites/russie-attribution-de-cyberattaques-contre-la-france-au-service-de-renseignement-militaire-russe, 2025.
3. ESET Research. Sednit reloaded : Back in the trenches. https://www.welivesecurity.com/en/eset-research/sednit-reloaded-back-trenches/, 2026.
4. Sekoia.io. APT28 Operation Phantom Net Voxel. https://blog.sekoia.io/apt28-operation-phantom-net-voxel/, 2025.
5. Sekoia.io. TransparentTribe targets Indian military organisations with DeskRAT. https://blog.sekoia.io/transparenttribe-targets-indian-military-organisations-with-deskrat/, 2025.
6. Sekoia.io. RePythonNET-MCP. https://github.com/SEKOIA-IO/RePythonNET-MCP, 2026.
7. Sekoia.io. Script used to automatically decrypt strings in some Covenant samples of APT28. https://gist.github.com/cmeslay/9ff499776ca2c2d84890c30a16001c20, 2026.