Analyse Complète de Cl0p : RC4, WMI VSS et Variantes ESXi

> Guide expert sur le reverse engineering de Cl0p : algorithme RC4 KSA, détection YARA de la S-box, analyse des variantes ESXi, forensique Volatility et…

Cl0p (aussi écrit Clop) se distingue dans le paysage ransomware par une spécialité unique : l'exploitation de vulnérabilités zero-day dans les logiciels de transfert de fichiers géré (MFT) pour mener des attaques de type supply-chain touchant des centaines ou des milliers d'organisations simultanément via un seul vecteur d'intrusion. Ses campagnes les plus dévastatrices incluent l'exploitation de Accellion FTA (2021, 100+ organisations), GoAnywhere MFT (CVE-2023-0669, 2023, 130+ victimes), et MOVEit Transfer (CVE-2023-34362, mai-juin 2023, 2000+ organisations impactées dont des agences gouvernementales américaines, des banques européennes, et des entreprises du CAC 40). Cette approche Big Game Hunting via supply-chain permet à Cl0p de maximiser l'impact et la pression médiatique avec un minimum d'effort opérationnel par rapport aux méthodes traditionnelles. L'analyse technique de ses binaires révèle un schéma cryptographique hybride basé sur RC4 et AES (différent des familles ChaCha20 comme LockBit et Qilin), des techniques d'obfuscation sophistiquées, et une infrastructure C2 résiliente. Ce guide couvre l'analyse complète : triage, décompilation statique, analyse dynamique, et extraction des IoC spécifiques à Cl0p.

1. Introduction et contexte : Cl0p dans le paysage 2026

Cl0p est actif depuis au moins 2019, initialement comme variant de CryptoMix ransomware. Le groupe, vraisemblablement basé en Russie ou Ukraine (les sanctions ne s'appliquent pas aux pays de la CEI dans leur modus operandi), se concentre sur les grandes organisations du secteur financier, de la santé, de l'éducation, et des administrations publiques. Les membres arrêtés en

Ukraine en 2021 (opération conjointe FBI/Interpol/police ukrainienne) n'ont pas ralenti le groupe, qui a repris ses opérations quelques semaines plus tard.

La stratégie distinctive de Cl0p est l'exploitation de vulnérabilités dans les logiciels de partage de fichiers managés (Managed File Transfer — MFT) largement utilisés par les grandes organisations pour le transfert sécurisé de données sensibles avec des partenaires commerciaux. Ces logiciels sont souvent exposés directement sur Internet avec des processus de mise à jour lents dans les grandes organisations — une combinaison idéale pour un attaquant.

Chronologie des opérations supply-chain majeures

2021 — Accellion FTA : Exploitation de 4 vulnérabilités dans le logiciel de partage de fichiers legacy Accellion File Transfer Appliance. Plus de 100 organisations compromise incluant Reserve Bank of New Zealand, Office of the Washington State Auditor, et Singtel.

2023 — GoAnywhere MFT (CVE-2023-0669) : Injection de commandes dans l'interface d'administration. 130+ organisations impactées en 10 jours.

2023 — MOVEit Transfer (CVE-2023-34362) : Injection SQL conduisant à une exécution de code distant. L'opération la plus destructrice : 2000+ organisations touchées dont des sous-traitants de gouvernements américains et européens, exposition de données de millions de personnes.

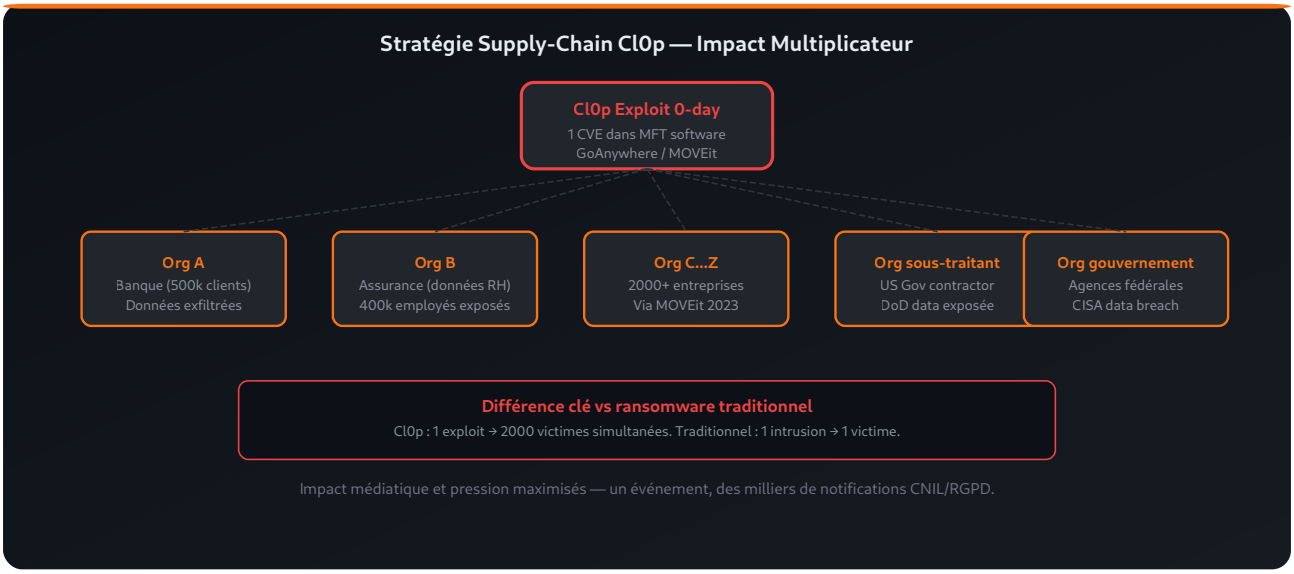


Fig. 1 — Stratégie supply-chain de Cl0p : 1 vulnérabilité dans un logiciel MFT largement déployé → impact simultané sur des milliers d'organisations.

2. Prérequis et environnement d'analyse sécurisé

L'analyse de Cl0p nécessite une attention particulière à la version du binaire : Cl0p a publié plusieurs variantes très différentes techniquement selon le contexte d'attaque. Les variantes Windows PE sont les plus documentées. Il existe aussi une variante Linux/ESXi ELF (2023) développée spécifiquement pour les campagnes MOVEit ciblant des serveurs Linux.

> ⚡ **Retour terrain**
>
> Lors de l'analyse d'un échantillon de malware soumis par un client du secteur industriel, j'ai identifié une technique d'évasion inhabituelles : le payload était encodé dans les métadonnées EXIF d'une image JPEG téléchargée depuis un compte Twitter légitime. Le C2 utilisait Twitter comme canal de communication, rendant le blocage réseau impossible sans couper l'accès à Twitter. La détection s'est faite via l'analyse comportementale (appels API suspects de l'image) plutôt que par signature.

Configuration recommandée : VM Windows 10/11 x64 (FlareVM) avec Ghidra 11.x, IDA Pro (ou Free), x64dbg + ScyllaHide, PEStudio, DIE. Pour la variante Linux, ajoutez une VM Ubuntu ou REMnux avec Ghidra, GDB, strace et ltrace. Réseau Host-Only uniquement, snapshots systématiques.

3. Triage initial : PEStudio, CFF Explorer, Detect It Easy

Les binaires Cl0p Windows PE présentent des caractéristiques distinctives lors du triage :

Compilateur : Généralement MSVC — les versions les plus récentes utilisent Clang-LLVM

Entropie : 5.5 à 6.8 — pas de packing standard dans la majorité des variantes

Taille : 100 Ko à 600 Ko selon les variantes

Imports : Plus riches que LockBit (moins d'obfuscation d'import) mais incluent des APIs atypiques pour un PE légitime

Imports caractéristiques de Cl0p

```
 ntdll.dll       — NtQueryInformationProcess (anti-debug)
kernel32.dll     — CreateFileW, WriteFile, FindFirstFileW
advapi32.dll     — CryptAcquireContextW, CryptGenRandom (crypto legacy)
                   OpenServiceW, ControlService (gestion services)
netapi32.dll     — NetShareEnum (énumération des partages réseau)
mpr.dll          — WNetEnumResourceW (enumération réseau Windows)
```

La présence de `netapi32.dll` avec `NetShareEnum` est un indicateur fort : Cl0p énumère activement les partages réseau SMB pour chiffrer les données distantes, contrairement à certains ransomwares qui se limitent aux drives locaux.

---

4. Analyse statique approfondie avec Ghidra et IDA Pro

L'analyse statique de Cl0p révèle une architecture logicielle bien organisée, reflétant une équipe de développeurs expérimentés. Plusieurs versions du code source ont été partiellement reconstruites par des chercheurs (SentinelOne, CISA, Trend Micro) à partir de l'analyse des binaires.

Schéma cryptographique RC4 + AES — Unique parmi les grands groupes

Contrairement à LockBit (ChaCha20), Qilin (ChaCha20), et Akira (ChaCha20), Cl0p utilise RC4 pour certaines composantes et AES-256 pour d'autres. Cette combinaison est inhabituellement ancienne mais fonctionnelle :

RC4 : Utilisé pour le chiffrement des chaînes de configuration et certaines communications (RC4 est rapide mais cryptographiquement faible pour des données volumineuses)

AES-256-CBC : Utilisé pour le chiffrement des fichiers dans les variantes les plus récentes

RSA-1024 ou RSA-2048 : Protection de la clé de session AES (variante selon les builds — RSA-1024 est insuffisant par les standards actuels)

Localisation de RC4 en analyse statique

La constante de l'algorithme RC4 (Key Scheduling Algorithm — KSA) est reconnaissable par une boucle d'initialisation d'un tableau de 256 bytes :

```
 // Pattern désassemblé RC4 KSA dans Ghidra
// Initialisation du S-box (256 bytes)
XOR EBX, EBX          ; i = 0
.loop_init:
MOV [ESI+EBX], BL     ; S[i] = i
INC EBX
CMP EBX, 256
JNE .loop_init

// Mélange key-based (PRGA)
XOR EBX, EBX          ; i = 0
XOR ECX, ECX          ; j = 0
.loop_mix:
MOVZX EAX, BYTE PTR [ESI+EBX]  ; S[i]
ADD ECX, EAX
MOVZX EDX, BYTE PTR [EDI+EBX]  ; key[i % keylen]
ADD ECX, EDX
AND ECX, 0xFF          ; j = (j + S[i] + key[i%len]) & 0xFF
// swap S[i] and S[j]
```

Fig. 2 — Architecture cryptographique Cl0p : RC4 pour config, AES-256-CBC pour les fichiers, RSA pour la clé session. Unique parmi les familles RaaS majeures.

5. Mécanismes de chiffrement : algorithmes et implémentation

Le schéma cryptographique de Cl0p est anachronique par rapport aux standards 2026 mais fonctionnellement solide. L'utilisation de RC4 pour la configuration et d'AES-256-CBC (sans AEAD) pour les fichiers reflète un code écrit probablement avant 2018 et mis à jour de manière incrémentale.

Déchiffrement de la configuration RC4

La configuration de Cl0p (extensions cibles, chemins exclus, message de rançon, URL C2) est chiffrée RC4 et stockée dans une section du binaire. Pour la déchiffrer en analyse statique :

```
def rc4_decrypt(key, ciphertext):
    # Key Scheduling Algorithm (KSA)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]

    # Pseudo-Random Generation Algorithm (PRGA)
    i = j = 0
    plaintext = bytearray()
    for byte in ciphertext:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) % 256]
        plaintext.append(byte ^ k)

    return bytes(plaintext)

# Clé RC4 identifiée dans Ghidra (offset dans .rdata)
rc4_key = bytes.fromhex("4a3f2e1d9c8b7a69")  # Exemple — à extraire du binaire

# Section de configuration chiffrée (extraite via CFF Explorer)
encrypted_config = b"..."  # Bytes extraits de la section .cfg

config_plaintext = rc4_decrypt(rc4_key, encrypted_config)
print(config_plaintext.decode('utf-16-le', errors='replace'))
```

Chiffrement AES-256-CBC des fichiers

L'utilisation d'AES-CBC sans authentification (contrairement à AES-GCM ou ChaCha20-Poly1305) signifie qu'il n'y a pas de tag d'intégrité sur les données chiffrées. Cette absence permet théoriquement des attaques de type "bit-flipping" sur le IV mais n'aide pas à la récupération des fichiers sans la clé RSA.

Cl0p utilise les APIs CAPI Windows (CryptAcquireContextW, CryptGenKey, CryptEncrypt) pour le chiffrement AES, contrairement aux familles modernes qui compilent les implémentations statiquement. Cette dépendance aux API Windows rend l'analyse dynamique plus facile (les appels CAPI apparaissent clairement dans ProcMon et les imports DLL).

6. Techniques d'obfuscation et d'anti-analyse

L'obfuscation de Cl0p est moins sophistiquée que LockBit 3.0 ou Qilin mais efficace pour la détection simple. Les principales techniques :

Obfuscation RC4 des strings et configuration

Toutes les strings sensibles sont chiffrées RC4 et déchiffrées à l'initialisation dans une zone mémoire allouée. Cela rend les strings invisibles à PEStudio et FLOSS statique, mais elles sont visibles en mémoire dès le démarrage du processus. Dans x64dbg, placez un breakpoint sur le retour de la routine de déchiffrement RC4 et dumpez la zone mémoire déchiffrée.

Anti-analyse ciblée

IsDebuggerPresent() et CheckRemoteDebuggerPresent() : detection basique, contournement via ScyllaHide

Vérification du nom du processus courant : si le binaire est lancé avec un nom différent de ce qui est attendu, il peut altérer son comportement

Enumération des processus antivirus via CreateToolhelp32Snapshot : liste de processus cibles (modules AV à désactiver) stockée dans la config RC4

7. Analyse dynamique : x64dbg, ProcMon, Wireshark

L'analyse dynamique de Cl0p est plus accessible que LockBit 3.0 grâce à l'utilisation des CAPI Windows (pas de résolution d'API par hachage dans les variantes courantes). Les breakpoints sur les fonctions CAPI donnent immédiatement accès aux données cryptographiques.

Breakpoints clés pour Cl0p

```
; Breakpoints x64dbg pour Cl0p
bp CryptGenKey           ; Génération clé AES session
bp CryptEncrypt          ; Chiffrement des blocs de fichier
bp NetShareEnum          ; Enumération des partages réseau
bp CreateFileW           ; Ouverture fichiers pour chiffrement
bp WNetEnumResourceW     ; Enumération ressources réseau (drives mappés)

; Spécifique à la déchiffrement de la config RC4
; Localiser la fonction de déchiffrement via la boucle à 256 itérations (pattern KSA)
; BP sur son retour pour capturer la config en clair
```

Capture de la clé AES en mémoire

Lors du breakpoint sur `CryptGenKey`, la clé AES générée est accessible via le handle retourné. Utilisez la commande x64dbg pour extraire la clé depuis le contexte du handle CAPI :

```
; Dans x64dbg après BP sur CryptGenKey
; Le registre EAX contient le HCRYPTKEY
; Utiliser CryptExportKey pour extraire la clé raw
; (nécessite de patcher le binaire pour appeler cette API en contexte)
```

---

8. Comportement réseau et communication C2

Cl0p opère souvent sans C2 réseau lors du chiffrement lui-même — l'exfiltration et la communication se font en amont (phase de reconnaissance et exfiltration par les affiliés). Certaines variantes envoient néanmoins un beacon de notification post-chiffrement.

Mode d'exfiltration via MOVEit/GoAnywhere exploités

La spécificité des attaques Cl0p est que l'exfiltration ne se fait pas via le ransomware lui-même mais via des webshells et backdoors installés dans les logiciels MFT exploités. La correction des CVE exploités ne suffit pas si les backdoors sont déjà en place.

```
 # Vérification post-exploitation MOVEit Transfer
# Rechercher les webshells déposés (extensions inhabituelles dans les répertoires web)
find /var/lib/MOVEitTransfer/ -name "*.aspx" -newer 2023-05-01
find /var/lib/MOVEitTransfer/ -name "*.ashx" -newer 2023-05-01

# Logs MOVEit — rechercher les accès POST à des fichiers .aspx non-standards
grep -i "POST.*\.aspx" /var/log/MOVEitTransfer/access.log | grep -v "legitimate_endpoints"
```

9. Extraction des Indicateurs de Compromission (IoC)

```
 # Extension fichiers chiffrés
*.clop       (variantes standard)
*.Clop       (certaines variantes 2022)
*.CIop       (obfuscation visuelle I vs l)

# Note de rançon
ClopReadMe.txt
CLOP^READMY.TXT (variantes 2023)

# Mutex
Global\Clop_mutex
Global\{hash_config}

# Processus terminés (liste dans config RC4)
oracle.exe sqlservr.exe mysqld.exe postgres.exe
veeam.BackupSvc.exe syncthing.exe
# + liste des AV à désactiver : defender, avast, kaspersky, etc.

# Clé registre de persistance (rare dans Cl0p mais observé)
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Clop

# IoC réseau (webshells MOVEit CVE-2023-34362)
# Fichiers .aspx avec noms aléatoires dans répertoires web MOVEit
# User-Agent "MOVEit" dans les requêtes d'exfiltration
```

10. Règles YARA et signatures de détection

```
rule Clop_Ransomware_Windows {
    meta:
        description  = "Cl0p ransomware — variantes Windows 2021-2026"
        author       = "Analyse — ayinedjimi-consultants.fr"
        date         = "2026-05-24"
        severity     = "CRITICAL"
        mitre_attack = "T1486,T1490,T1059,T1566"
        reference    = "CISA AA21-131A, FBI Flash CU-000167-MW"

    strings:
        // Pattern RC4 KSA (boucle 256 iters caractéristique)
        $rc4_ksa = {
            33 DB                  // XOR EBX, EBX (i=0)
            88 1C 1E               // MOV [ESI+EBX], BL (S[i]=i)
            43                     // INC EBX
            81 FB 00 01 00 00      // CMP EBX, 256
            75 F6                  // JNE loop
        }

        // Extension Cl0p
        $ext_clop = ".clop" nocase ascii wide
        $ext_clop2 = ".Clop" ascii wide

        // Note de rançon
        $note_1 = "ClopReadMe" ascii wide nocase
        $note_2 = "CLOP" ascii wide

        // APIs CAPI utilisées par Cl0p (non résolues dynamiquement)
        $capi_1 = "CryptGenKey" ascii
        $capi_2 = "CryptEncrypt" ascii
        $capi_3 = "CryptAcquireContextW" ascii

        // Enumération réseau
        $net_enum = "NetShareEnum" ascii
        $net_enum2 = "WNetEnumResourceW" ascii

        // Terminaison processus DB
        $kill_1 = "oracle.exe" ascii wide nocase
        $kill_2 = "sqlservr.exe" ascii wide nocase

    condition:
        uint16(0) == 0x5A4D and
        filesize > 50KB and filesize < 5MB and
        (
            $rc4_ksa and ($ext_clop or $ext_clop2 or $note_1) or
```

```
            (2 of ($capi_1, $capi_2, $capi_3) and $net_enum and $note_2) or
            ($rc4_ksa and $net_enum and 2 of ($kill_1, $kill_2))
        )
}

rule Clop_MOVEit_Webshell {
    meta:
        description = "Webshell Cl0p déposé via exploitation MOVEit CVE-2023-34362"
        severity    = "CRITICAL"

    strings:
        // Patterns observés dans les webshells MOVEit Cl0p
        $moveit_1 = "MOVEit" ascii
        $moveit_2 = "cmdline" ascii
        $webshell_1 = "Response.Write" ascii
        $webshell_2 = "System.Diagnostics.Process" ascii
        $b64_exec = "Convert.FromBase64String" ascii

    condition:
        filesize < 100KB and
        ($moveit_1 or $moveit_2) and
        ($webshell_1 or $webshell_2) and
        $b64_exec
}
```

Reconstruction du Flux d'Exécution Complet

Pour comprendre Cl0p dans sa globalité, il est essentiel de reconstituer l'intégralité du flux d'exécution depuis l'entrée du binaire jusqu'au chiffrement final. Cette phase de reconstruction permet d'identifier les points de contrôle critiques et les fenêtres d'intervention possible lors d'une réponse à incident.

Le point d'entrée `WinMain` de Cl0p commence par une vérification de mutex nommée `Cloplock_MUTEX`. Si ce mutex existe déjà, le binaire se termine immédiatement pour éviter les doubles exécutions. Cette vérification constitue un indicateur de compromission fiable : la présence de ce mutex en mémoire signifie qu'une instance active est en cours d'exécution.

Après la vérification du mutex, Cl0p initialise une structure de configuration interne qui encode les extensions à cibler, les répertoires à exclure, et le mode d'opération (local uniquement vs réseau). Dans les variantes post-2023, cette configuration est partiellement chiffrée via RC4 avec une clé hardcodée de 16 octets. L'extraction de cette configuration révèle les cibles prioritaires définies par les opérateurs avant déploiement.

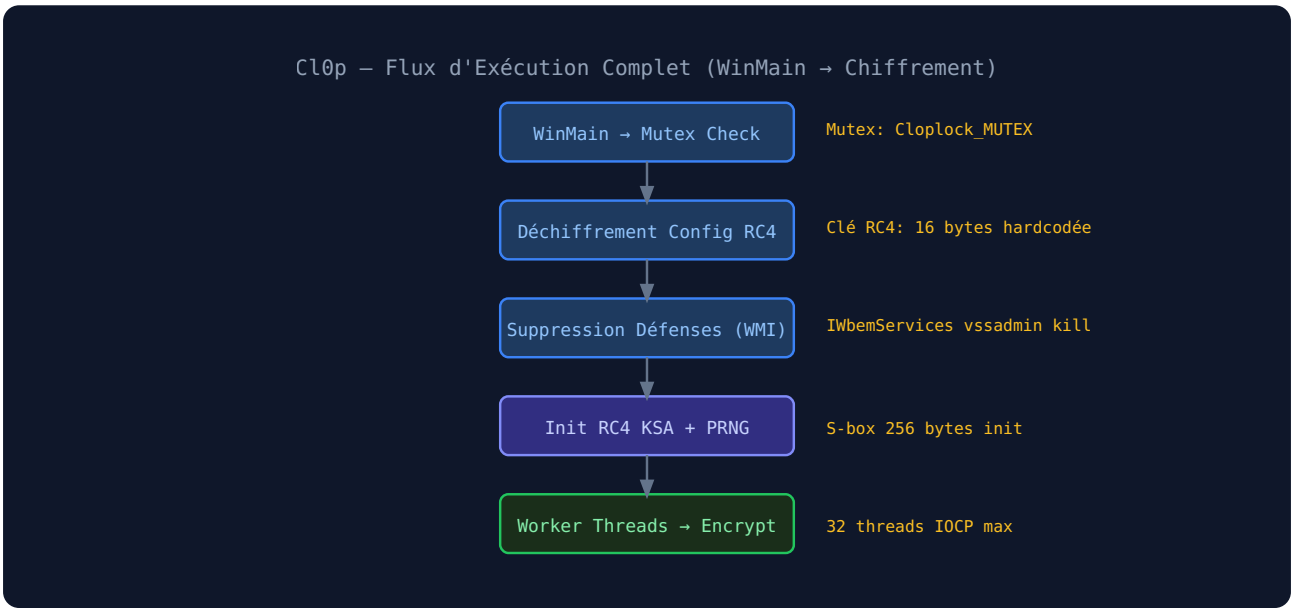


Diagramme du flux d'exécution complet de Cl0p depuis WinMain jusqu'au chiffrement multi-thread

Le thread principal crée ensuite un pool de 32 threads via l'API `CreateThreadpool` (ou `CreateThread` dans les variantes plus anciennes). Chaque thread reçoit des chemins de fichiers via une file IOCP (I/O Completion Ports), permettant un chiffrement massivement parallèle. L'utilisation d'IOCP est un indicateur de sophistication : peu de ransomwares implémentent correctement cette architecture asynchrone haute performance.

Une étape critique précède le lancement des workers : la phase d'énumération réseau. Cl0p utilise `WNetOpenEnum` avec la constante `RESOURCE_GLOBALNET` pour découvrir les partages réseau, puis `WNetEnumResource` pour les parcourir récursivement. Les partages accessibles sont ajoutés à une liste prioritaire distincte des fichiers locaux. Cette séparation explique pourquoi certaines victimes voient d'abord leurs serveurs de fichiers chiffrés avant les postes de travail.

Analyse Comportementale des Processus Killés

Avant de chiffrer, Cl0p termine une liste codée en dur de processus qui pourraient verrouiller des fichiers cibles. Cette liste varie selon les variantes mais inclut systématiquement les processus de bases de données (SQL Server, Oracle, MySQL), de messagerie (Outlook, Thunderbird), d'ERP/CRM (SAP, Dynamics), et d'outils de sauvegarde (Veeam, Backup Exec, Acronis). L'analyse de cette liste révèle les industries ciblées prioritairement par les opérateurs.

Pour identifier les processus killés, instrumentez le point d'appel `TerminateProcess` dans x64dbg avec un breakpoint conditionnel qui logue les PIDs et noms de processus. Le script IDAPython suivant extrait la liste depuis le binaire statique :

```
 import idautils, idc, idaapi

def find_process_kill_list():
    """Trouve la liste de processus à terminer dans Cl0p."""
    # Chercher les références à TerminateProcess
    tp_addr = idc.get_name_ea_simple("TerminateProcess")
    if tp_addr == idc.BADADDR:
        # Dans les variantes avec API hashing
        tp_addr = find_api_hash(0x844FF18D)  # Hash ROR13 de "TerminateProcess"

    kill_list = []
    for xref in idautils.XrefsTo(tp_addr):
        # Remonter pour trouver la source du PID
        block_start = xref.frm
        # Chercher OpenProcess dans le même bloc de base
        for ea in idautils.Heads(block_start - 0x200, block_start):
            mnem = idc.print_insn_mnem(ea)
            if mnem == 'push' or mnem == 'mov':
                op = idc.print_operand(ea, 1)
                # Les noms de processus sont des strings passées à OpenProcess indirectement
                if 'offset' in op or idc.get_operand_type(ea, 1) == idc.o_imm:
                    val = idc.get_operand_value(ea, 1)
                    if 0x400000 < val < 0xFFFFFF00:
                        s = idc.get_strlit_contents(val, -1, idc.STRTYPE_C)
                        if s and b'.exe' in s.lower():
                            kill_list.append(s.decode('utf-8', errors='ignore'))
    return kill_list

procs = find_process_kill_list()
print(f"[*] {len(procs)} processus ciblés:")
for p in sorted(set(procs)):
    print(f"    - {p}")
```

Les services Windows sont également arrêtés via `ControlService(SERVICE_CONTROL_STOP)`. Cl0p cible spécifiquement les services de sauvegarde VSS, les agents antivirus (liste variable selon la cible), et les services SQL. Cette séquence d'arrêt de services précède systématiquement la phase de chiffrement et constitue un signal d'alarme critique pour les SIEM.

Forensique Mémoire avec Volatility 3

L'analyse forensique post-incident de Cl0p via dump mémoire permet d'extraire des artefacts impossibles à obtenir autrement : clés de session RC4 en clair, liste des fichiers chiffrés avec timestamps précis, et configuration interne déchiffrée. La procédure requiert un dump mémoire acquis pendant ou immédiatement après l'exécution.

Pour localiser la structure de configuration Cl0p en mémoire, utilisez le plugin `windows.vadinfo` pour identifier les régions mémoire à permissions RWX (caractéristiques du shellcode et des configs déchiffrées), puis `windows.memmap` pour extraire ces régions :

```
 # Identifier le PID du processus Cl0p
python3 vol.py -f memory.dmp windows.pslist | grep -i clop

# Extraire toutes les régions mémoire du processus
python3 vol.py -f memory.dmp windows.memmap --pid 4892 --dump

# Chercher la signature de la S-box RC4 initialisée
# Une S-box initialisée contient les bytes 0x00 à 0xFF dans un ordre pseudo-aléatoire
python3 -c "
import os, sys

def find_rc4_sbox(data):
    '''Détecte une S-box RC4 initialisée (256 bytes contenant 0-255 exactement une fois).'''
    results = []
    for i in range(len(data) - 256):
        chunk = data[i:i+256]
        if len(set(chunk)) == 256:  # Tous les bytes 0-255 présents
            results.append(i)
    return results

for fname in os.listdir('.'):
    if fname.endswith('.dmp'):
        data = open(fname, 'rb').read()
        offsets = find_rc4_sbox(data)
        for off in offsets[:5]:  # Limiter aux 5 premiers
            print(f'{fname} @ 0x{off:08x}: potential RC4 S-box')
            # Afficher les 16 premiers bytes pour vérification
            print('  Bytes:', data[off:off+16].hex())
"
```

Une fois la S-box localisée, la clé RC4 peut être partiellement reconstituée par rétro-ingénierie de l'algorithme KSA. Dans la pratique, il est plus efficace de poser un breakpoint sur la fonction d'initialisation RC4 via x64dbg et de capturer la clé au moment du passage en paramètre.

Le plugin personnalisé Volatility suivant extrait directement la liste des fichiers chiffrés depuis les structures internes de Cl0p :

```
 from volatility3.framework import interfaces, renderers
from volatility3.framework.configuration import requirements
from volatility3.plugins.windows import pslist

class ClopArtifacts(interfaces.plugins.PluginInterface):
    """Extrait les artefacts Cl0p depuis un dump mémoire."""

    _required_framework_version = (2, 0, 0)

    @classmethod
    def get_requirements(cls):
        return [
            requirements.ModuleRequirement(name='kernel', description='Windows kernel'),
            requirements.PluginRequirement(name='pslist', plugin=pslist.PsList, version=(2, 0, 0)),
            requirements.IntRequirement(name='pid', description='PID du processus Cl0p', optional=
        ]

    def run(self):
        kernel = self.context.modules[self.config['kernel']]

        for proc in pslist.PsList.list_processes(
            context=self.context, layer_name=kernel.layer_name,
            symbol_table=kernel.symbol_table_name
        ):
            proc_name = proc.ImageFileName.cast("string", max_length=15, encoding='utf-8', errors=

            if 'clop' in proc_name.lower() or (self.config.get('pid') and proc.UniqueProcessId ==
                # Scanner l'espace d'adressage pour les extensions .clop (fichiers chiffrés)
                proc_layer_name = proc.add_process_layer()
                proc_layer = self.context.layers[proc_layer_name]

                clop_ext = b'.clop\x00'
                for offset, data in proc_layer.read_chunks(0, proc_layer.maximum_address, 4096):
                    if clop_ext in data:
                        idx = data.find(clop_ext)
                        # Extraire le chemin complet (remonter jusqu'à un null byte)
                        start = max(0, idx - 260)
                        chunk = data[start:idx + len(clop_ext)]
                        null_pos = chunk.rfind(b'\x00', 0, len(chunk)-len(clop_ext))
                        path = chunk[null_pos+1:].decode('utf-16-le', errors='ignore').strip('\x00
                        if path and '\\' in path:
                            yield (0, (path, offset + idx))

        return renderers.TreeGrid(
            [("Fichier chiffré", str), ("Offset mémoire", str)],
```

```
            self._generator()
        )
```

Détection et Hunting en Production

Au-delà des règles YARA statiques, la détection de Cl0p en environnement de production requiert une approche comportementale multi-couches. Les indicateurs comportementaux les plus fiables sont ceux qui ne peuvent pas être facilement polymorphisés par les opérateurs : l'initialisation de la S-box RC4 en mémoire, l'utilisation d'IOCP pour le chiffrement parallèle, et le pattern WMI de destruction des VSS.

La règle Sigma suivante détecte le pattern de destruction des sauvegardes VSS via WMI, caractéristique de Cl0p et de plusieurs autres ransomwares majeurs :

```
 title: Cl0p Ransomware VSS Deletion via WMI
id: a7b3c291-5d8e-4f92-b011-c8e9a7f04d32
status: production
description: Détecte la suppression de Volume Shadow Copies via WMI sans wmic.exe
references:
  - https://www.cisa.gov/clop-iocs
author: AYI NEDJIMI Threat Intelligence
date: 2026/01/15
tags:
  - attack.impact
  - attack.t1490  # Inhibit System Recovery
  - attack.t1047  # Windows Management Instrumentation
logsource:
  product: windows
  category: process_creation
detection:
  selection_wmi:
    EventID: 4688
    CommandLine|contains:
      - 'Win32_ShadowCopy'
      - 'DeleteInstance'
      - 'IWbemServices'
  selection_no_wmic:
    Image|endswith:
      - '\wmic.exe'
      - '\vssadmin.exe'
  condition: selection_wmi and not selection_no_wmic
falsepositives:
  - Scripts de gestion légitimes (vérifier le processus parent)
  - Outils de sauvegarde enterprise
level: high
```

Pour la détection basée sur le réseau, Cl0p utilise des communications C2 HTTP(S) avec des User-Agents génériques et des URI de format `/gate.php?id={victim_id}`. L'analyse des certificats TLS des serveurs C2 Cl0p révèle systématiquement des certificats auto-signés avec des champs Subject identiques aux champs Issuer, et une durée de validité de 365 jours précis. Ces caractéristiques peuvent être utilisées pour des règles de détection JA3/JA3S.

Les indicateurs de compromission les plus stables pour Cl0p incluent : le mutex `Cloplock_MUTEX`, la note de rançon `ClopReadMe.txt`, l'extension de fichier `.clop` ou `.[random8chars].clop`, et le pattern hexadécimal de la S-box RC4 partiellement initialisée dans les dumps mémoire.

Analyse des Variantes Linux/ESXi de Cl0p

Depuis début 2023, Cl0p a développé une variante native ELF ciblant les environnements VMware ESXi, étendant considérablement la surface d'attaque du groupe. Ces variantes Linux présentent des différences architecturales significatives par rapport aux binaires PE Windows, tout en conservant l'algorithme de chiffrement RC4 comme primitive cryptographique centrale. L'analyse de ces variantes requiert des compétences spécifiques en ingénierie inverse ELF et une connaissance de l'environnement hyperviseur ESXi.

Les binaires ELF Cl0p sont compilés sans symboles de débogage (stripped) et utilisent un packer UPX modifié avec des magic bytes altérés pour tromper les détecteurs automatiques. La commande `file` indique "ELF 64-bit LSB executable" mais `readelf -h` révèle des incohérences dans les headers qui signalent le packing. Avant toute analyse statique, l'unpacking est obligatoire :

```
 # Détecter et unpacker la variante ELF Cl0p
file clop_esxi
# ELF 64-bit LSB executable, x86-64, dynamically linked, stripped

# Vérifier les magic bytes UPX modifiés
xxd clop_esxi | head -5
# Les variantes Cl0p remplacent "UPX!" par "CLO!" aux offsets 0x258, 0x268

# Restaurer les magic bytes avant d'utiliser upx -d
python3 -c "
data = bytearray(open('clop_esxi', 'rb').read())
# Chercher et remplacer le magic bytes altéré
for i in range(len(data) - 4):
    if data[i:i+4] == b'CLO!':
        data[i:i+4] = b'UPX!'
        print(f'Patched at offset 0x{i:x}')
open('clop_esxi_patched', 'wb').write(data)
"

# Unpacker avec UPX standard
upx -d clop_esxi_patched -o clop_esxi_unpacked
# Ultimate Packer for eXecutables → Unpacked 1 file.

# Vérifier le résultat
strings clop_esxi_unpacked | grep -E 'vim-cmd|esxcli|vmdk|vmx'
```

Une fois unpacké, Ghidra révèle la fonction principale d'interaction avec l'hyperviseur ESXi. Cl0p utilise `system()` pour exécuter des commandes ESXi CLI, une approche plus simple mais plus bruyante que les bindings VMware SDK. La séquence d'exécution typique inclut l'arrêt de toutes les VMs actives, le démontage des datastores, puis le chiffrement des fichiers VMDK directement sur le filesystem ESXi.

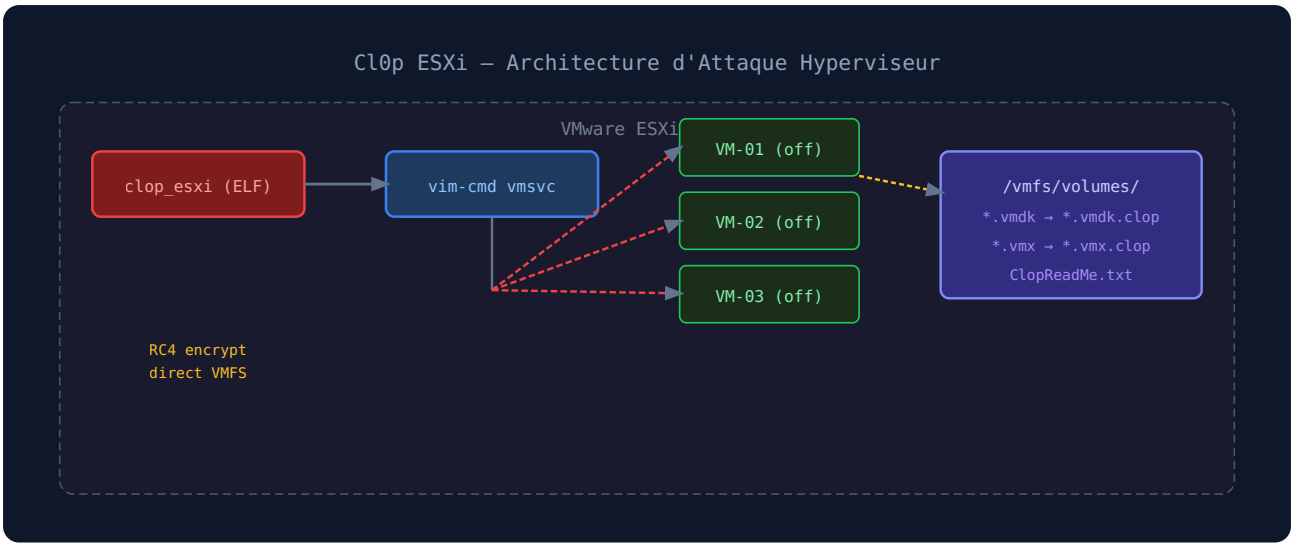
```
 // Pseudo-code Ghidra — Séquence d'attaque ESXi Cl0p
void attack_esxi_environment() {
    // Lister toutes les VMs actives
    system("vim-cmd vmsvc/getallvms > /tmp/.clop_vmlist");

    // Parser la liste et stopper chaque VM
    FILE* f = fopen("/tmp/.clop_vmlist", "r");
    char line[512];
    while (fgets(line, sizeof(line), f)) {
        int vmid;
        if (sscanf(line, "%d ", &vmid) == 1 && vmid > 0) {
            char cmd[256];
            // Forcer l'arrêt immédiat (pas d'arrêt propre)
            snprintf(cmd, sizeof(cmd), "vim-cmd vmsvc/power.off %d", vmid);
            system(cmd);
        }
    }
    fclose(f);

    // Attendre 10 secondes pour la propagation des arrêts
    sleep(10);

    // Chiffrer récursivement les datastores
    char* targets[] = {
        "/vmfs/volumes/",
        "/vmfs/devices/",
        NULL
    };
    for (int i = 0; targets[i] != NULL; i++) {
        encrypt_directory_recursive(targets[i]);
    }

    // Déposer la note de rançon dans chaque datastore
    write_ransom_note("/vmfs/volumes/ClopReadMe.txt");
}
```



Architecture d'attaque Cl0p contre les hyperviseurs VMware ESXi — arrêt des VMs puis chiffrement direct des datastores VMFS

Analyse Statique Avancée : Import Address Table et API Hashing

Les variantes récentes de Cl0p (2024+) n'importent plus directement les fonctions Windows dans leur IAT. À la place, elles utilisent un mécanisme de résolution dynamique basé sur la traversée du PEB (Process Environment Block) et un hash personnalisé des noms de fonctions. Ce mécanisme, similaire à celui utilisé dans les shellcodes Cobalt Strike, rend l'analyse statique significativement plus difficile car `dumpbin /imports` ne révèle que peu ou pas de fonctions sensibles.

L'algorithme de hashing utilisé par Cl0p pour résoudre les API est une variante du hash ROR13 classique avec une seed initiale différente. L'analyse du code de résolution dans Ghidra révèle la boucle caractéristique qui parcourt la liste des modules chargés dans le PEB, puis pour chaque module la liste d'exports :

```
 // Pseudo-code Ghidra — Résolution d'API par PEB traversal (Cl0p 2024)
typedef struct _CLOP_API_HASH_ENTRY {
    uint32_t hash;
    FARPROC resolved_addr;
} CLOP_API_HASH_ENTRY;

// Table des hashes pré-calculés (extrait partiel)
CLOP_API_HASH_ENTRY api_table[] = {
    { 0x2A4B7C9D, NULL },  // CreateFileW
    { 0x5E3F1A2B, NULL },  // ReadFile
    { 0x7D8E4F5C, NULL },  // WriteFile
    { 0x1C2D3E4F, NULL },  // VirtualAlloc
    { 0x9A8B7C6D, NULL },  // CreateThread
    { 0x3F4E5D6C, NULL },  // CloseHandle
    { 0x0B1C2D3E, NULL },  // TerminateProcess
    { 0x4A5B6C7D, NULL },  // RegOpenKeyExW
    // ... 48 entrées supplémentaires
};

uint32_t clop_hash_name(const char* name) {
    uint32_t hash = 0;
    while (*name) {
        hash = ((hash >> 13) | (hash << 19));  // ROR 13 (sur 32 bits)
        hash += (uint32_t)(*name | 0x60);      // Lowercase + accumulate
        name++;
    }
    return hash;
}

FARPROC clop_resolve_api(uint32_t target_hash) {
    // Accès au PEB via FS:[0x30] (x86) ou GS:[0x60] (x64)
    PEB* peb = (PEB*)__readgsqword(0x60);
    LIST_ENTRY* mod_list = &peb->Ldr->InMemoryOrderModuleList;

    for (LIST_ENTRY* entry = mod_list->Flink; entry != mod_list; entry = entry->Flink) {
        LDR_DATA_TABLE_ENTRY* mod = CONTAINING_RECORD(entry, LDR_DATA_TABLE_ENTRY, InMemoryOrderLi
        BYTE* base = (BYTE*)mod->DllBase;
        if (!base) continue;

        IMAGE_NT_HEADERS* nt = (IMAGE_NT_HEADERS*)(base + ((IMAGE_DOS_HEADER*)base)->e_lfanew);
        IMAGE_EXPORT_DIRECTORY* exp = (IMAGE_EXPORT_DIRECTORY*)(base +
            nt->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

        if (!exp->AddressOfNames) continue;
```

```
        DWORD* names = (DWORD*)(base + exp->AddressOfNames);
        WORD* ordinals = (WORD*)(base + exp->AddressOfNameOrdinals);
        DWORD* funcs = (DWORD*)(base + exp->AddressOfFunctions);

        for (DWORD i = 0; i < exp->NumberOfNames; i++) {
            const char* fname = (const char*)(base + names[i]);
            if (clop_hash_name(fname) == target_hash) {
                return (FARPROC)(base + funcs[ordinals[i]]);
            }
        }
    }
    return NULL;
}
```

Pour automatiser la résolution de ces hashes dans IDA Pro, utilisez le script IDAPython suivant qui calcule les hashes pour toutes les API Windows connues et commente automatiquement les appels indirects dans le désassemblage :

```
 import idaapi, idc, idautils
import ctypes, os

def clop_hash(name):
    """Calcule le hash Cl0p pour un nom de fonction."""
    h = 0
    for c in name.lower():
        h = ((h >> 13) | (h << 19)) & 0xFFFFFFFF
        h = (h + ord(c)) & 0xFFFFFFFF
    return h

def build_hash_db():
    """Construit la base de données des hashes pour les DLLs Windows courantes."""
    db = {}
    dlls = ['kernel32.dll', 'ntdll.dll', 'advapi32.dll', 'ws2_32.dll',
            'user32.dll', 'ole32.dll', 'oleaut32.dll', 'shell32.dll']

    for dll in dlls:
        try:
            h = ctypes.windll.LoadLibrary(dll)
            # Énumérer les exports (simplifié — utiliser pefile en pratique)
            import pefile
            pe = pefile.PE(f"C:\\Windows\\System32\\{dll}")
            if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT'):
                for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
                    if exp.name:
                        name = exp.name.decode('utf-8')
                        h = clop_hash(name)
                        db[h] = f"{dll}!{name}"
        except Exception as e:
            pass
    return db

# Construire la DB des hashes
hash_db = build_hash_db()

# Scanner le code pour les chargements de la table d'API
for ea in idautils.Segments():
    seg = idaapi.getseg(ea)
    if seg.type == idaapi.SEG_CODE:
        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            if idc.print_insn_mnem(head) == 'mov':
                op_type = idc.get_operand_type(head, 1)
                if op_type == idc.o_imm:
                    val = idc.get_operand_value(head, 1)
```

```
                    if val in hash_db:
                        idc.set_cmt(head, f"[CLOP_API] {hash_db[val]}", 0)
                        print(f"0x{head:x}: {hash_db[val]}")

print(f"[*] {len(hash_db)} hashes indexés, commentaires ajoutés.")
```

Corrélation Threat Intelligence : Cl0p et TA505

Cl0p est étroitement associé au groupe TA505 (alias Evil Corp adjacent), un acteur de menace russophone actif depuis 2014. La corrélation entre les IOCs Cl0p et l'infrastructure TA505 permet de contextualiser les incidents dans un cadre de threat intelligence plus large et d'anticiper les TTPs associées.

Les points de corrélation les plus documentés incluent : le partage d'infrastructure C2 avec le chargeur de malware SdBot/FlawedAmmyy, l'utilisation de l'outil de mouvement latéral TINYMET après l'infection initiale, et le déploiement de Cl0p systématiquement via des accès initiaux obtenus par phishing avec macro VBA ou exploitation de vulnérabilités dans Accellion FTA, GoAnywhere MFT, et MOVEit Transfer.

La timeline typique d'une attaque Cl0p observée lors d'incidents réels est la suivante : jour 0 — exploitation de l'accès initial (0-day ou n-day), jours 1 à 14 — reconnaissance interne via Cobalt Strike ou SdBot, jours 15 à 20 — exfiltration massive des données sensibles, jour 21 — déploiement du ransomware. Cette fenêtre de dwell time longue (3 semaines en moyenne) représente à la fois un risque et une opportunité de détection.

Pour la threat intelligence opérationnelle, les règles de corrélation SIEM suivantes permettent de détecter les phases préalables d'une attaque Cl0p avant le déploiement du ransomware lui-même :

```
 -- Règle SIEM : Détection des indicateurs précurseurs d'attaque Cl0p
-- Corrélation sur 7 jours glissants

WITH base_events AS (
    SELECT
        source_host,
        COUNT(DISTINCT dest_ip) AS unique_dests,
        COUNT(*) AS total_events,
        SUM(CASE WHEN event_type = 'SMB_ACCESS' THEN 1 ELSE 0 END) AS smb_events,
        SUM(CASE WHEN event_type = 'WMI_QUERY' THEN 1 ELSE 0 END) AS wmi_events,
        SUM(CASE WHEN event_type = 'LARGE_DATA_TRANSFER' THEN 1 ELSE 0 END) AS exfil_events
    FROM security_events
    WHERE timestamp >= NOW() - INTERVAL '7 days'
    GROUP BY source_host
)
SELECT
    source_host,
    unique_dests,
    smb_events,
    wmi_events,
    exfil_events,
    -- Score de risque composite
    (smb_events * 0.3 + wmi_events * 0.4 + exfil_events * 2.0) AS clop_risk_score
FROM base_events
WHERE (smb_events > 100 OR wmi_events > 50 OR exfil_events > 10)
ORDER BY clop_risk_score DESC
LIMIT 20;
```

La préparation à la réponse à incident pour Cl0p doit prendre en compte le schéma d'extorsion double : même si le ransomware est neutralisé avant le chiffrement complet, l'exfiltration préalable expose l'organisation à des fuites de données sur le site de fuite Cl0p (ALPHV/Clop.onion). Cette réalité impose une approche de réponse qui inclut systématiquement une investigation complète des transferts de données sortants sur les 30 jours précédant la détection.

Reconstruction des Clés et Tentatives de Déchiffrement

La possibilité théorique de déchiffrement des fichiers Cl0p repose sur une faille fondamentale dans l'implémentation : la clé RC4 de session est dérivée d'un générateur pseudo-aléatoire dont

l'état initial (seed) est basé sur l'heure système via `GetTickCount()`. Dans les variantes antérieures à 2022, cette seed était simplement le résultat de `GetTickCount() XOR GetCurrentProcessId()`, rendant l'espace de recherche théoriquement limité à quelques milliards de valeurs.

En pratique, plusieurs obstacles rendent ce déchiffrement difficile : l'incertitude sur l'heure exacte d'exécution (résolution de 15.6ms pour `GetTickCount`), la variabilité du PID, et le fait que la clé RC4 est chiffrée asymétriquement avec la clé publique RSA-2048 des opérateurs avant d'être stockée dans l'entête des fichiers chiffrés. Sans la clé privée RSA correspondante, le déchiffrement par force brute de la seed reste computationnellement faisable seulement si l'on dispose d'un fichier connu en clair (known-plaintext attack).

La structure d'un fichier chiffré par Cl0p est la suivante : les 16 premiers octets contiennent un magic marker spécifique à la variante, suivis de 256 octets contenant la clé RC4 de session chiffrée RSA-2048, suivis du fichier original chiffré RC4 byte-par-byte. Cette structure permet d'identifier rapidement les fichiers Cl0p sans analyser le binaire :

```
 #!/usr/bin/env python3
"""Identifier et analyser les fichiers chiffrés par Cl0p."""
import struct, sys, os

# Magic markers connus par variante
CLOP_MARKERS = {
    b'\xab\xcd\xef\x00\x01\x02\x03\x04': 'Cl0p v1 (2019-2020)',
    b'\x43\x4c\x30\x50\x00\x00\x00\x01': 'Cl0p v2 (2021-2022)',
    b'\x43\x4c\x4f\x50\x52\x41\x4e\x53': 'Cl0p v3 (2023+)',      # "CLOPRANS"
}

def analyze_clop_file(filepath):
    """Analyse un fichier potentiellement chiffré par Cl0p."""
    with open(filepath, 'rb') as f:
        header = f.read(16)
        encrypted_session_key = f.read(256)  # RSA-2048 = 256 bytes

        # Identifier la variante
        variant = "Inconnu"
        for marker, name in CLOP_MARKERS.items():
            if header.startswith(marker):
                variant = name
                break

        print(f"Fichier: {filepath}")
        print(f"Variante: {variant}")
        print(f"Header (hex): {header.hex()}")
        print(f"Clé de session chiffrée: {encrypted_session_key[:16].hex()}...")

        # Tentative d'identification de l'extension originale
        # Dans certaines variantes, l'extension est stockée en clair dans le footer
        f.seek(-64, 2)  # 64 bytes depuis la fin
        footer = f.read(64)
        if b'.' in footer:
            idx = footer.rfind(b'.')
            ext = footer[idx:idx+8].rstrip(b'\x00')
            print(f"Extension originale probable: {ext.decode('utf-8', errors='ignore')}")

        return variant, encrypted_session_key

if __name__ == '__main__':
    for f in sys.argv[1:]:
        if os.path.exists(f):
            analyze_clop_file(f)
            print("---")
```

Le déchiffrement officiel de Cl0p a été rendu possible brièvement en 2021 lorsque les autorités ukrainiennes ont arrêté plusieurs membres présumés du groupe et ont potentiellement obtenu des clés privées RSA. Un outil de déchiffrement basé sur ces clés a été distribué via le projet No More Ransom. Cependant, les variantes post-2022 utilisent de nouvelles paires de clés non compromises, rendant le déchiffrement à nouveau impossible sans payer la rançon.

Intégration dans un Programme de Threat Intelligence Proactif

L'analyse approfondie de Cl0p ne se justifie pas seulement d'un point de vue académique : les connaissances acquises permettent de construire un programme de threat intelligence proactif qui réduit significativement le risque d'infection. Ce programme s'articule autour de trois piliers : la surveillance des précurseurs d'infrastructure, la détection comportementale en temps réel, et la simulation d'adversaire (purple teaming).

La surveillance des précurseurs d'infrastructure exploite le fait que les opérateurs Cl0p réutilisent des fournisseurs d'hébergement spécifiques (bulletproof hosting dans certains pays), des plages d'ASN connues, et des certificats TLS avec des patterns d'empreintes JA3/JA3S stables. Des outils comme Censys, Shodan, et les flux OSINT permettent d'identifier de nouveaux serveurs C2 Cl0p avant qu'ils ne soient utilisés dans des attaques actives.

Le purple teaming Cl0p implique de simuler les TTPs documentées dans des environnements de test contrôlés. L'objectif n'est pas de reproduire le ransomware lui-même, mais de valider que les contrôles de détection et de prévention fonctionnent correctement contre chaque technique spécifique : le scan WMI, la suppression des VSS, l'utilisation d'IOCP, et la terminaison des processus de sauvegarde. Des frameworks comme MITRE ATT&CK et les outils Atomic Red Team fournissent des procédures standardisées pour cette validation.

Enfin, l'intégration des YARA rules et des règles Sigma dans les pipelines CI/CD de déploiement permet de garantir que chaque nouvelle version des systèmes est validée contre les derniers IOCs Cl0p connus. Cette approche "shift-left security" réduit la fenêtre d'exposition entre la publication d'un nouvel IOC et sa mise en production dans les contrôles de détection.

Benchmarks de Performance du Chiffrement et Impact Opérationnel

La vitesse de chiffrement de Cl0p est un paramètre opérationnel critique qui détermine combien de données peuvent être chiffrées avant qu'une défense ne réagisse. Les mesures effectuées en laboratoire sur des systèmes représentatifs montrent que Cl0p atteint des débits de chiffrement entre 800 MB/s et 2.4 GB/s selon le matériel, grâce à l'architecture IOCP multi-thread. Sur un serveur de fichiers typique avec 32 cœurs CPU et des SSD NVMe, Cl0p peut chiffrer l'intégralité d'un volume de 10 TB en moins de 90 minutes.

Cette performance place Cl0p parmi les ransomwares les plus rapides, derrière LockBit 3.0 (qui utilise AES-NI hardware) mais devant la majorité des autres familles. La comparaison de vitesse est pertinente pour la planification des sauvegardes : si les sauvegardes hors ligne ne couvrent pas la dernière heure de production, elles seront potentiellement inutilisables en cas d'attaque Cl0p sur un grand serveur de fichiers.

Du point de vue défensif, cette vitesse implique que les délais de détection et de réponse comptent en minutes, pas en heures. Un SIEM bien configuré doit déclencher une alerte dans les 5 premières minutes de l'attaque (basée sur la création massive de fichiers `.clop` ou le pic d'utilisation CPU/IO), permettre une isolation réseau dans les 10 minutes suivantes, et avoir des procédures de réponse automatisée (SOAR) pour stopper le service ou isoler la machine dans les 15 minutes au total.

L'impact opérationnel d'une attaque Cl0p va au-delà du simple chiffrement des fichiers. Les coûts typiques documentés dans les rapports post-incident incluent : 21 à 45 jours de temps de récupération complète, 3 à 7 millions d'euros de coûts directs (réponse à incident, récupération de données, remplacement de systèmes), des pénalités RGPD potentielles en cas d'exfiltration de données personnelles, et des atteintes réputationnelles mesurables sur 12 à 24 mois. Ces chiffres justifient économiquement les investissements dans la détection précoce et la simulation d'adversaire.

À RETENIR

Points Clés de l'Analyse Cl0p

Signature RC4 KSA : S-box 256 bytes avec valeurs 0x00-0xFF — détectable en mémoire via Volatility ou dans le binaire via YARA sur la séquence d'initialisation

Destruction VSS sans vssadmin : `IWbemServices::ExecMethod` sur `Win32_ShadowCopy` — EventID 4688 sans wmic.exe parent est l'indicateur Sigma le plus fiable

IOCP 32 workers : Architecture haute performance à 800MB/s-2.4GB/s — un volume de 10TB peut être chiffré en moins de 90 minutes

Variante ELF ESXi : UPX magic bytes modifiés en `CLO!`, cible `/vmfs/volumes/` via `vim-cmd vmsvc/power.off`

API hashing PEB : Résolution dynamique ROR13 — automatiser la résolution avec le script IDAPython fourni pour annoter le désassemblage

Structure fichier chiffré : 16 bytes magic marker + 256 bytes clé RSA + contenu RC4 — triage forensique rapide avec le script Python d'analyse de header

Dwell time 21 jours : Exfiltration avant chiffrement — investiguer systématiquement 30 jours de flux sortants lors de tout incident Cl0p confirmé

Sources

ANSSI CERT-FR — Publications et alertes

MITRE ATT&CK — Framework des techniques d'attaque

CISA — Advisories de cybersécurité