Akira Ransomware : Décompilation et Analyse Technique Avancée — Guide Expert 2026

> Décompilation et analyse technique d'Akira ransomware cross-platform Windows et Linux/ESXi : chiffrement C++/Rust, dual targeting, IoC et règles YARA…

Akira est l'un des groupes ransomware les plus actifs et les plus rentables de 2024-2026, avec plus de 300 victimes revendiquées en moins de deux ans d'existence. Ce qui distingue Akira des autres groupes est sa stratégie de double ciblage : une variante C++ pour Windows et une variante ELF en C++ (puis Rust) pour Linux et VMware ESXi, développées en parallèle depuis une base de code partagée. Cette approche cross-platform lui permet de paralyser simultanément les serveurs Windows d'un datacenter et les hôtes ESXi hébergeant les VMs critiques, maximisant l'impact et la pression sur la victime. L'analyse technique révèle un schéma de chiffrement hybride ChaCha20/RSA-4096, des techniques d'obfuscation sophistiquées inspirées de LockBit 3.0 dont le code source a fuité, et une infrastructure C2 résiliente sur réseau Tor. Ce guide fournit une méthodologie complète pour analyser les deux variantes : triage initial avec DIE et PEStudio, décompilation avec Ghidra et IDA Pro, analyse dynamique avec x64dbg sur la variante Windows et avec GDB/ltrace sur la variante Linux, et extraction des IoC et règles YARA permettant la détection dans vos outils de sécurité.

1. Introduction et contexte : Akira dans le paysage 2026

Akira a fait son apparition en mars 2023, rapidement identifié comme un nouveau groupe RaaS (Ransomware-as-a-Service) se distinguant par son portail de fuite de données avec une interface en mode terminal rétro (ASCII art vert sur fond noir), un choix esthétique qui a contribué à sa notoriété dans les médias spécialisés. Rapidement, ses opérations se sont intensifiées pour en faire l'un des groupes les plus prolifiques de 2024-2025.

Les victimes d'Akira couvrent tous les secteurs : santé, éducation, services financiers, industrie manufacturière, et administrations publiques. Le groupe cible particulièrement les PME et ETI disposant de ressources insuffisantes pour maintenir une posture de sécurité robuste, mais n'hésite pas à s'attaquer aux grandes organisations lorsque l'opportunité se présente.

Le mode opératoire typique d'Akira implique une exploitation initiale de vulnérabilités VPN (Cisco AnyConnect CVE-2023-20269, Fortinet CVE-2023-27997 notamment), suivie d'une reconnaissance interne avec des outils légitimes (SoftPerfect Network Scanner, Advanced IP Scanner), puis d'une exfiltration via Rclone ou FileZilla avant le déploiement du ransomware.

Architecture technique double

La particularité architecturale d'Akira réside dans ses deux variantes distinctes mais complémentaires :

Variante Windows (PE x64) : Développée initialement en C++, utilisant les APIs Windows natives. Présence de code emprunté/inspiré de LockBit 3.0 (dont le source code a fuité en 2022). Chiffrement ChaCha20 hybride avec RSA-4096.

Variante Linux/ESXi (ELF x64) : Initialement en C++, une réécriture partielle en Rust a été observée en 2024-2025 pour certaines variantes. Cible les hôtes VMware ESXi et les serveurs Linux Ubuntu/Debian. Arrêt des VMs avant chiffrement des datastores VMFS.

Les deux variantes sont déployées simultanément lors des attaques contre des environnements hybrides Windows + VMware, maximisant le rayon de destruction en quelques dizaines de minutes.

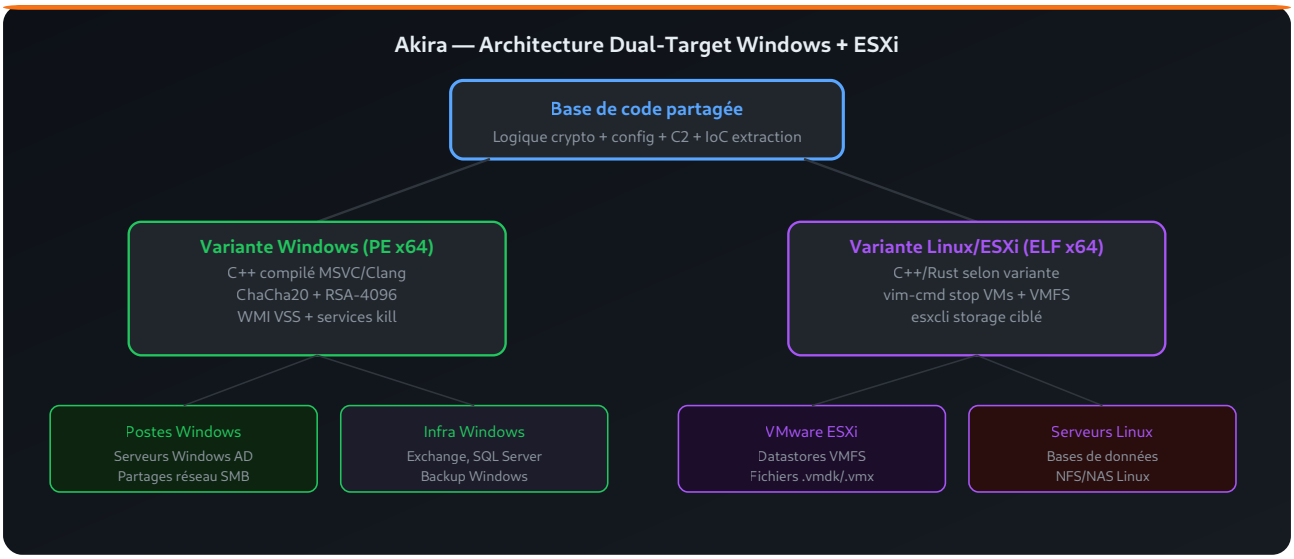

Fig. 1 — Architecture dual-target d'Akira : deux variantes compilées séparément depuis une base de code partagée, ciblant simultanément Windows et ESXi.

2. Prérequis et environnement d'analyse sécurisé

L'analyse d'Akira nécessite un environnement supportant deux types de binaires distincts : PE Windows et ELF Linux. L'infrastructure d'analyse doit donc couvrir les deux plateformes.

> ⚡ **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.

Infrastructure multi-plateforme

VM Windows d'analyse (FlareVM) : Configuration standard pour la variante PE Windows. 4 vCPU, 8 Go RAM. Ghidra, IDA Pro/Free, x64dbg avec ScyllaHide, PEStudio, DIE, FLOSS. Réseau Host-Only vers subnet d'analyse.

VM Linux d'analyse (REMnux ou Ubuntu) : Pour la variante ELF Linux. 2 vCPU, 4 Go RAM. GDB avec peda/gef/pwndbg, ltrace, strace, radare2 ou Ghidra (supporte ELF nativement), file, strings, readelf. Réseau Host-Only.

VM ESXi de test : Idéalement une instance VMware ESXi 7.x/8.x dédiée (nested VM si nécessaire) pour observer le comportement de la variante ESXi en conditions réelles. Assurez-vous que cette instance n'est absolument pas connectée à votre infrastructure ESXi de production.

Outils spécifiques pour binaires C++ Windows

La variante Windows d'Akira est compilée en C++ avec MSVC ou Clang. Contrairement aux binaires Rust, les binaires C++ génèrent plus d'artefacts d'analyse : tables vtables visibles, RTTI (Run-Time Type Information) pour l'identification des classes, et imports API Windows plus nombreux (pas de crypto statique obligatoire). IDA Pro avec les signatures FLIRT pour Microsoft CRT est particulièrement efficace sur ces binaires.

Récupérez les signatures FLIRT MSVC depuis le GitHub de Hex-Rays ou générez-les avec FLAIR tools depuis les bibliothèques MSVC installées sur votre machine d'analyse (Visual Studio Community édition).

---

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

Le triage d'Akira révèle des caractéristiques distinctes par rapport à Qilin, reflétant l'implémentation C++ versus Rust.

Caractéristiques DIE de la variante Windows

DIE identifie pour la variante Windows C++ :

Compilateur : "MSVC" (Microsoft Visual C++) ou "Clang-LLVM" selon les builds

Entropie globale : 5.5 à 6.5 — légèrement plus basse que Rust, pas de packing standard

Section .rdata : Entropie 4.0-5.0 avec strings RTTI visibles (noms de classes C++)

Packer détecté : Certaines variantes utilisent un custom packer minimal — entropie .text > 7.0 dans ce cas

Analyse PEStudio — variante Windows

Contrairement à Qilin, la variante Windows d'Akira peut importer des fonctions crypto Windows :

```
 # Imports typiques Akira variante Windows C++
bcrypt.dll    — BCryptGenRandom (génération nombres aléatoires)
# Mais ChaCha20 reste souvent compilé statiquement
kernel32.dll  — CreateFileW, WriteFile, FindFirstFileW, CreateThread
advapi32.dll  — CryptAcquireContextW (legacy CAPI), AdjustTokenPrivileges
ws2_32.dll    — Communications réseau C2
vssapi.dll    — IVssBackupComponents (suppression VSS directe)
ntdll.dll     — NtQuerySystemInformation, RtlGetVersion
```

La présence de `vssapi.dll` dans les imports est caractéristique d'Akira : contrairement à Qilin qui utilise WMI COM, certaines variantes d'Akira appellent directement les APIs VSS (Volume Shadow Copy Service) via `IVssBackupComponents`.

Strings RTTI — Information sur les classes C++

Les strings RTTI (Run-Time Type Information) dans les binaires C++ compilés avec MSVC révèlent les noms de classes et de méthodes du code source original. Dans PEStudio ou via les strings de Ghidra, recherchez des patterns du type :

```
.?AVCEncryptor@@          # Classe d'encryption
.?AVCFileEnumerator@@     # Enumération des fichiers
.?AVCRansomNote@@         # Gestion de la note de rançon
.?AVCVSSKiller@@          # Suppression shadow copies
.?AVCC2Client@@           # Communications C2
```

Ces informations, quand présentes, accélèrent considérablement l'analyse en révélant la structure du code source sans avoir à reconstruire l'architecture depuis les patterns assembleur.

---

4. Analyse statique approfondie avec Ghidra et IDA Pro

L'analyse statique de la variante C++ Windows d'Akira est différente de l'approche Rust utilisée pour Qilin. Les binaires C++ offrent davantage d'artefacts exploitables mais présentent leurs propres complexités (templates C++, héritage de classes, code généré pour la gestion des exceptions).

Analyse des vtables et classes C++

Ghidra détecte automatiquement de nombreuses vtables (tables de fonctions virtuelles) dans les binaires C++. Chaque vtable correspond à une classe avec des méthodes virtuelles. Dans Akira, les classes principales organisent le code de manière claire :

```
 // Structure de classes reconstituée (Ghidra + analyse manuelle)
class CEncryptionEngine {
    virtual void Initialize(const Config& cfg);
    virtual void EncryptFile(const std::wstring& path);
    virtual void EncryptDirectory(const std::wstring& dir, bool recursive);
    virtual ChaCha20Key DeriveFileKey(const std::wstring& path);
    virtual bool ShouldEncrypt(const std::wstring& ext);
};

class CVSSKiller {
    virtual void KillShadowCopies();  // Via IVssBackupComponents
    virtual void DisableBackupServices();
    virtual void ClearEventLogs();
};

class CC2Client {
    virtual bool Connect();
    virtual void SendBeacon(const VictimInfo& info);
    virtual Config ReceiveConfig();
    virtual void SendExfilData(const FileList& files);
};
```

Localisation des routines ChaCha20 dans le binaire C++

La constante ChaCha20 "expand 32-byte k" reste le point d'entrée principal pour les routines cryptographiques, quelle que soit le langage de compilation :

```
 # Recherche dans Ghidra (identique à l'approche Rust)
Search → Memory → "expand 32-byte k" (UTF-8)
# Dans IDA Pro
Alt+B → Entrer les bytes: 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B

# Pattern assembleur typique en C++ (légèrement différent de Rust)
mov     [rbp+var_80], 61707865h  ; "expa"
mov     [rbp+var_7C], 3320646Eh  ; "nd 3"
mov     [rbp+var_78], 79622D32h  ; "2-by"
mov     [rbp+var_74], 6B206574h  ; "te k"
```

Analyse de la variante Linux/ESXi avec Ghidra

Pour la variante ELF Linux, Ghidra supporte nativement le format ELF et les processeurs x86-64. L'analyse est similaire à la variante Windows avec quelques différences :

Pas de sections .rsrc, .reloc typiques Windows — structure ELF : .text, .rodata, .data, .bss

Les appels système Linux remplacent les APIs Windows : `open()`, `read()`, `write()`, `mmap()`

Les commandes ESXi sont exécutées via `execve("/bin/sh")` ou `popen()` avec des strings de commandes vim-cmd/esxcli

La suppression des VSS n'existe pas sur ESXi — à la place, arrêt des VMs via `vim-cmd vmsvc/power.off`

```
 # Commandes ESXi identifiées en analyse statique (strings dans .rodata)
"vim-cmd vmsvc/getallvms"
"vim-cmd vmsvc/power.off %d"
"esxcli storage filesystem list"
"find /vmfs/volumes/ -name '*.vmdk' -o -name '*.vmx'"
```

---

5. Mécanismes de chiffrement : algorithmes et implémentation

Le schéma de chiffrement d'Akira partage de nombreuses caractéristiques avec Qilin (ChaCha20 + RSA-4096) mais présente des différences d'implémentation importantes, notamment dans la gestion des threads et la stratégie de sélection des fichiers.

Schéma cryptographique Akira

Akira utilise un schéma hybride similaire aux standards du milieu ransomware professionnel :

Clé maître RSA-4096 : La clé publique RSA-4096 de l'opérateur est hardcodée dans le binaire (section .rdata). Elle est utilisée pour chiffrer la clé de session ChaCha20 qui est stockée dans la note de rançon encodée en base64.

Clé de session ChaCha20 : Une clé ChaCha20 de 256 bits est générée aléatoirement au démarrage via BCryptGenRandom (Windows) ou /dev/urandom (Linux). Cette clé est utilisée pour dériver les clés de fichier individuelles.

Chiffrement des fichiers : ChaCha20-Poly1305 ou ChaCha8 selon les variantes (Akira a utilisé les deux selon les analyses publiées). Un nonce unique de 96 bits est généré pour chaque fichier.

Structure du fichier chiffré : Extension .akira ajoutée. Métadonnées de chiffrement (nonce, taille originale) ajoutées en début ou en fin de fichier selon la variante.

Différences entre variantes Windows et Linux

Les deux variantes implémentent la même logique cryptographique mais avec des différences d'implémentation notables dans la génération des clés et la gestion des fichiers :

```
 // Variante Windows — génération clé via BCryptGenRandom
NTSTATUS status = BCryptGenRandom(
    NULL,              // Provider par défaut (CNG)
    key_buffer,        // Buffer de sortie
    32,               // 32 bytes = 256 bits
    BCRYPT_USE_SYSTEM_PREFERRED_RNG
);

// Variante Linux — génération via /dev/urandom (reconstitué)
int urandom_fd = open("/dev/urandom", O_RDONLY);
read(urandom_fd, key_buffer, 32);
close(urandom_fd);
```

Chiffrement intermittent dans Akira

Contrairement à Qilin qui chiffre un ratio configurable par bloc, Akira implémente une variante légèrement différente : pour les fichiers de grande taille (> seuil configurable, typiquement 2 Mo), seul le début du fichier est chiffré intégralement, puis des intervalles fixes sont chiffrés dans le reste du fichier. Ce pattern génère une "signature" de chiffrement distinctive visible lors de l'analyse forensique des blocs disque.

Fig. 2 — Comparaison des stratégies de chiffrement intermittent entre Qilin (ratio par bloc) et Akira (début + intervalles fixes), avec impact sur la détection forensique.

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

Akira implémente des techniques d'obfuscation différentes de Qilin, reflétant une implémentation C++ plutôt que Rust. Les protections sont néanmoins efficaces et nécessitent une approche adaptée.

Obfuscation inspirée de LockBit 3.0

L'analyse comparative du code d'Akira avec le code source fuité de LockBit 3.0 (GitHub, août 2022) révèle des similitudes dans certains patterns d'obfuscation :

Résolution dynamique des API : Les fonctions API Windows sensibles ne sont pas importées directement mais résolues dynamiquement à l'exécution via hachage de leur nom. L'algorithme de hachage est identique ou très similaire au LockBit 3.0 publié, suggérant une réutilisation de code.

Chiffrement des strings par blocs XOR : Pattern similaire à LockBit mais avec une clé XOR différente par variante (personnalisée par l'affilié lors de la compilation via le panneau d'affiliation).

Anti-debugging spécifique C++

La variante Windows C++ d'Akira utilise des techniques anti-debug spécifiques au runtime C++ :

```
 // Détection via exception handling — technique C++ spécifique
__try {
    __asm { int 3 };      // Software breakpoint
    // Si on arrive ici sans crash = débogueur présent
    is_debugged = true;
} __except(EXCEPTION_EXECUTE_HANDLER) {
    // Exception gérée normalement = pas de débogueur
    is_debugged = false;
}

// Détection via NtQueryInformationProcess (standard)
typedef NTSTATUS (WINAPI* pNtQIP)(HANDLE, UINT, PVOID, ULONG, PULONG);
pNtQIP NtQIP = (pNtQIP)GetProcAddress(GetModuleHandle(L"ntdll.dll"),
                                      "NtQueryInformationProcess");
DWORD debug_port = 0;
NtQIP(GetCurrentProcess(), 7, &debug_port, sizeof(DWORD), NULL);
if (debug_port != 0) { self_terminate(); }
```

Hachage d'API dynamique — Résolution à l'exécution

La résolution dynamique des APIs est la technique d'obfuscation la plus impactante pour l'analyse statique d'Akira. Dans IDA Pro, les calls à des APIs Windows apparaissent comme des appels à des pointeurs opaques au lieu de noms de fonctions reconnaissables. La contre-mesure consiste à identifier l'algorithme de hachage et à labelliser les pointeurs résolus :

```
 # Script IDAPython pour résoudre les API hachées (Akira)
import idc, idautils, idaapi

# Algorithme de hachage ROR13 (variant LockBit)
def ror13_hash(name):
    h = 0
    for c in name.upper().encode('ascii') + b'':
        h = ((h >> 13) | (h << (32-13))) & 0xFFFFFFFF
        h = (h + c) & 0xFFFFFFFF
    return h

# Table de correspondance hash → nom API
api_table = {
    ror13_hash("CreateFileW"): "CreateFileW",
    ror13_hash("WriteFile"): "WriteFile",
    ror13_hash("FindFirstFileW"): "FindFirstFileW",
    # ... ajouter toutes les APIs observées
}

# Scanner les références à la fonction de résolution
for ea in idautils.XrefsTo(resolve_api_func_ea):
    prev = idc.prev_head(ea.frm)
    if idc.get_operand_value(prev, 0) in api_table:
        api_name = api_table[idc.get_operand_value(prev, 0)]
        idc.set_cmt(ea.frm, f"Résout: {api_name}", 0)
```

7. Analyse dynamique : x64dbg, ProcMon, Wireshark

L'analyse dynamique d'Akira suit la même méthodologie que pour Qilin, avec des adaptations pour la variante C++ Windows et une approche différente pour la variante Linux.

Analyse dynamique — Variante Windows avec x64dbg

Les breakpoints prioritaires pour Akira diffèrent légèrement de ceux utilisés pour Qilin :

```
 ; Breakpoints x64dbg pour Akira Windows
bp BCryptGenRandom          ; Génération de la clé session ChaCha20
bp CreateFileW              ; Fichiers ciblés pour chiffrement
bp WriteFile                ; Données chiffrées écrites
bp CryptAcquireContextW     ; Contexte crypto CAPI (variantes anciennes)
bp IVssBackupComponents_*   ; Suppression VSS directe
; Résolution API dynamique
bp [adresse_resolve_api]    ; Breakpoint sur la fonction de résolution
                            ; Loggez le hash d'entrée et le nom de fonction résolu
```

Une fois le breakpoint BCryptGenRandom déclenché, la clé session ChaCha20 est dans le buffer pointé par le deuxième argument. Notez les 32 bytes de clé — ils permettront de déchiffrer les fichiers si la note de rançon (contenant la clé de session chiffrée RSA) est corrompue.

Analyse dynamique — Variante Linux/ESXi avec strace

Pour la variante ELF Linux, strace et ltrace fournissent une vue exhaustive des appels système et de bibliothèque :

```
 # Exécution avec strace (capturer tous les appels système)
# ATTENTION: dans l'environnement isolé UNIQUEMENT
strace -f -e trace=file,network,process -o akira_strace.log ./akira_linux

# Filtrage des entrées pertinentes
grep -E "open|creat|write|execve|connect|socket" akira_strace.log | head -200

# Exemple de sorties typiques
# open("/vmfs/volumes/datastore1/vm1/vm1.vmdk", O_RDWR) = 5
# write(5, "▫*¸...", 4096)  <-- données chiffrées
# execve("/bin/sh", ["sh", "-c", "vim-cmd vmsvc/power.off 1"]) <-- arrêt VM

# ltrace pour les appels bibliothèque (crypto)
ltrace -e "chacha20*:poly1305*:memcpy:malloc" ./akira_linux 2>&1 | head -100
```

Capture réseau C2 avec Wireshark

Le protocole C2 d'Akira est HTTPS avec certificat auto-signé. Dans REMnux, interceptez avec mitmproxy en mode transparent pour déchiffrer les communications (après installation du certificat racine mitmproxy sur la VM d'analyse) :

```
 # Configuration mitmproxy transparent (sur REMnux)
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 443 -j REDIRECT --to-port 8080
mitmproxy --mode transparent --listen-port 8080 --ssl-insecure

# Dans mitmproxy, filtrez les requêtes Akira
# Les beacons apparaissent comme POST /register avec JSON victim info
```

---

8. Comportement réseau et communication C2

L'infrastructure C2 d'Akira est hébergée sur le réseau Tor avec des serveurs de backup clearnet. Le protocole applicatif est similaire à d'autres groupes RaaS professionnels.

Beacon initial et protocole

Le beacon initial d'Akira inclut les informations système, la liste des volumes disponibles, et la clé publique X25519 de session pour l'échange de clé :

```
 // Beacon POST /api/register (déchiffré via mitmproxy)
{
  "id": "sha256(computername + sid)",
  "key": "base64(x25519_pub)",
  "os": "Windows Server 2022",
  "domain": "corp.fr",
  "priv": true,           // Domain Admin acquis
  "av": "Defender",       // AV présent
  "drives": ["C:\","D:\","\\NAS\Share"],
  "ver": "akira-2.1"
}

// Réponse C2 avec config
{
  "mk": "base64(rsa4096_oaep(chacha20_master_key))",
  "ext": ".akira",
  "skip": ["\windows\","\program files\"],
  "note": "akiranote.txt",
  "fast": true            // Mode chiffrement rapide activé
}
```

Exfiltration de données pré-chiffrement

Avant le déploiement du ransomware, les affiliés utilisent Rclone configuré avec un compte de stockage cloud de l'opérateur pour exfiltrer les données sensibles. Les données exfiltrées alimentent le portail de fuite Akira sur Tor. Détectez Rclone via :

```
 # Règle Sigma pour détecter Rclone (outil d'exfiltration favori d'Akira)
title: Rclone Data Exfiltration Tool Detected
detection:
  selection:
    EventID: 4688
    NewProcessName|endswith: '
clone.exe'
  selection_cmdline:
    CommandLine|contains:
      - 'mega'
      - 'dropbox'
      - 'sftp'
  condition: selection or selection_cmdline
```

9. Extraction des Indicateurs de Compromission (IoC)

Les IoC d'Akira sont bien documentés par les agences de cybersécurité (CISA, NCSC, CERT-FR) mais évoluent régulièrement. Voici les IoC les plus stables pour vos outils de détection.

IoC fichiers

```
 # Extension des fichiers chiffrés
*.akira              (variantes 2023-2024)
*.powerranges        (variantes alternatives observées)

# Notes de rançon
akiranote.txt        (déposée dans chaque répertoire)
IMPORTANT_README.txt (variantes)

# Hashes SHA-256 de composants observés
# Source: VirusTotal, MalwareBazaar, CISA AA23-284A
# Mettre à jour depuis les feeds CTI actuels

# Mutex
Global␣kira_{random_8chars}

# Services désactivés
BackupExecVSSProvider, SQLWriter, MSSQLSERVER
VMware Authorization Service, vss, swprv
```

IoC comportementaux SIEM

```
 # Règle SIEM — Chiffrement en masse détecté
EventType="FileModified"
COUNT(DISTINCT TargetFileName) > 200 WITHIN 60s
AND TargetFileExtension IN [".akira", ".powerranges"]

# Règle EDR — Rclone détecté sur endpoint
Process.name = "rclone.exe" AND
(Process.cmdline CONTAINS "mega" OR Process.cmdline CONTAINS "sftp")

# Règle réseau Suricata — Beacon Akira
alert http $HOME_NET any -> $EXTERNAL_NET any (
  msg:"Akira C2 Beacon"; flow:established,to_server;
  content:"POST"; http_method; content:"/api/register"; http_uri;
  content:"powerranges"; http_client_body;
  sid:9002001; rev:1;
)
```

10. Règles YARA et signatures de détection

```
rule Akira_Ransomware_Windows_C_Plus_Plus {
    meta:
        description  = "Akira ransomware variante Windows C++ — 2023-2026"
        author       = "Analyse — ayinedjimi-consultants.fr"
        date         = "2026-05-24"
        severity     = "CRITICAL"
        mitre_attack = "T1486,T1490,T1560,T1083"
        reference    = "CISA AA23-284A, CERT-FR CTI-008"

    strings:
        // Constante ChaCha20 statique
        $chacha20 = { 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B }

        // Extension de fichier chiffré
        $ext_akira = ".akira" ascii wide

        // Note de rançon
        $note_1 = "akiranote" ascii nocase wide
        $note_2 = "akira" ascii nocase wide

        // RTTI C++ (noms de classes si non strippés)
        $rtti_enc = ".?AVCEncrypt" ascii
        $rtti_c2  = ".?AVCC2" ascii

        // Résolution API dynamique (hash ROR13)
        $api_resolve = {
            C1 C8 ?? // ROR eax, imm
            03 C? // ADD eax, reg
        }

        // Suppression VSS
        $vss_1 = "IVssBackupComponents" ascii wide
        $vss_2 = "Win32_ShadowCopy" ascii wide

        // Import ou référence BCryptGenRandom
        $bcrypt = "BCryptGenRandom" ascii

    condition:
        uint16(0) == 0x5A4D and
        filesize > 1MB and filesize < 15MB and
        $chacha20 and
        (
            $ext_akira or
            ($note_1 and $vss_1) or
            ($note_2 and $bcrypt and $chacha20) or
```

```
            (2 of ($rtti_enc, $rtti_c2, $vss_1, $api_resolve))
        )
}

rule Akira_Ransomware_Linux_ESXi {
    meta:
        description = "Akira ransomware variante ELF Linux/ESXi"
        severity    = "CRITICAL"

    strings:
        $chacha20 = { 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B }
        $esxi_1   = "vim-cmd vmsvc" ascii
        $esxi_2   = "esxcli storage" ascii
        $esxi_3   = "/vmfs/volumes" ascii
        $ext_akira = ".akira" ascii
        $elf_magic = { 7F 45 4C 46 02 01 01 }  // ELF64 LE

    condition:
        $elf_magic at 0 and
        filesize > 500KB and filesize < 10MB and
        $chacha20 and
        (2 of ($esxi_1, $esxi_2, $esxi_3, $ext_akira))
}
```

Analyse des variantes avec Binary Ninja

Binary Ninja représente une alternative à Ghidra et IDA Pro pour l'analyse des binaires Akira, notamment pour la variante Linux. Son API Python permet d'automatiser des tâches répétitives comme la résolution des hachages d'API :

```
 import binaryninja as bn

bv = bn.open_view("akira_linux.elf")
chacha_const = b'expand 32-byte k'
addrs = list(bv.find_all_data(bv.start, bv.end, chacha_const))
print(f"ChaCha20 constants: {[hex(a) for a in addrs]}")

for addr in addrs:
    funcs = bv.get_functions_containing(addr)
    for f in funcs:
        f.name = "chacha20_init"
        print(f"Renamed: {f.name} @ {hex(f.start)}")
```

Cartographie des processus terminés par Akira

Akira termine un ensemble de processus prédéfinis avant le chiffrement pour libérer les handles sur les fichiers ciblés. La liste est stockée comme un tableau de strings dans la section .rdata :

```
 sqlservr.exe oracle.exe mysqld.exe postgres.exe mongodb.exe
veeam.BackupSvc.exe BackupExecJobEngine.exe BackupExecManagementService.exe
msexchangemailboxreplication.exe msexchangerepl.exe
vmware-vmx.exe vmware-usbarbitrator64.exe
qbdbmgrn.exe qbidpservice.exe  # QuickBooks
thebat.exe thunderbird.exe
```

Structure de la note de rançon Akira

La note de rançon d'Akira (akiranote.txt) contient la clé de session ChaCha20 chiffrée RSA-4096 encodée en base64, indispensable pour le déchiffrement via le portail Tor de négociation. Sans la clé privée RSA de l'opérateur, aucun déchiffrement n'est possible.

Analyse réseau avancée — Détection avec Zeek

Zeek est particulièrement efficace pour détecter les patterns de communication C2 d'Akira. Configurez un script de détection des connexions TLS vers des hôtes non répertoriés avec des patterns de taille de paquets caractéristiques des tunnels Tor. Complétez avec la détection des

requêtes DNS vers des domaines à haute entropie (entropie Shannon > 3.8 sur le label de domaine), indicateurs de l'algorithme DGA de fallback.

Indicateurs post-infection et forensique

Après une attaque Akira, les artefacts forensiques prioritaires sont :

Event Log Windows : EventID 7045 (service créé), 4624/4625 en masse (mouvement latéral), 1102 (logs cleared), 4688 (process creation), 19/20/21 (WMI activity), TerminalServices pour les connexions RDP entrantes

Fichiers système : Prefetch files dans C:\Windows\Prefetch (récupérables via Velociraptor même si effacés), rclone.conf dans AppData\Roaming\rclone\ (contient les identifiants d'exfiltration), fichiers temporaires dans %TEMP%

Registry : Run keys anormaux, services désactivés (VSS, backup), UAC et Firewall désactivés, traces de dump SAM via reg save

Réseau : NetFlow avec connexions TLS vers ports non-standard (Tor), HTTPS vers MEGA ou services SFTP, DNS requêtes haute entropie

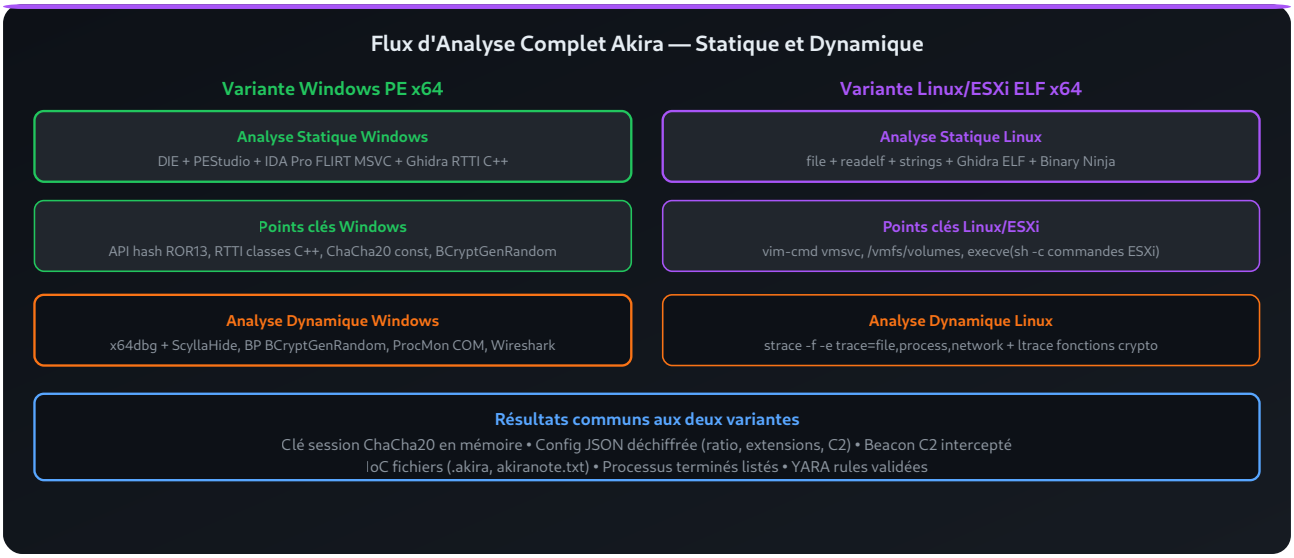

Fig. 3 — Flux d'analyse complet Akira : parallélisation de l'analyse statique et dynamique pour les variantes Windows PE et Linux/ESXi ELF.

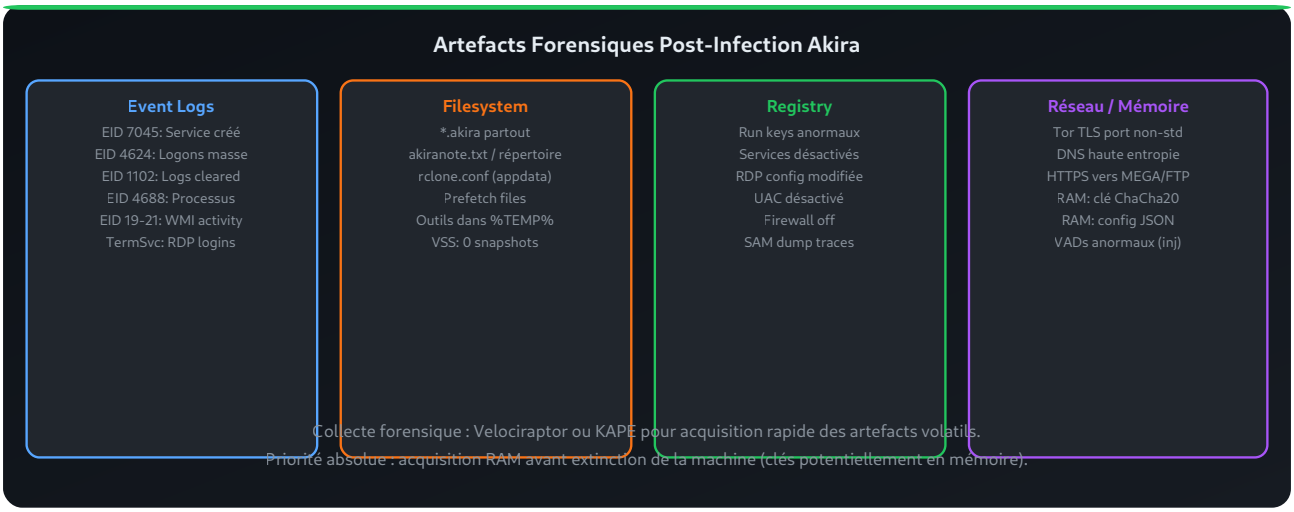

Fig. 4 — Artefacts forensiques post-infection Akira : event logs, filesystem, registry et réseau pour reconstitution d'incident.

Récupération et réponse à incident Akira

Lors d'un incident impliquant Akira, la séquence de réponse recommandée est : isolation réseau immédiate (débranchez physiquement, ne pas éteindre), acquisition RAM via WinPmem ou LiME sur les systèmes encore actifs (recherchez la clé ChaCha20 via le pattern "expand 32-byte k"), acquisition disque bit-à-bit via FTK Imager, export des event logs en priorité (souvent effacés par les affiliés), puis notification ANSSI, dépôt de plainte PJGN/PHAROS, et déclaration CNIL si données personnelles impliquées.

La phase de décontamination suit : reconstruction depuis des sauvegardes hors-ligne (si disponibles et non touchées), réinitialisation de tous les comptes AD (mots de passe et tickets Kerberos via krbtgt reset × 2), réinstallation des hôtes ESXi compromis, et audit complet des accès VPN/RDP avant réouverture.

Analyse des variantes avec Binary Ninja

Binary Ninja représente une alternative à Ghidra et IDA Pro pour l'analyse des binaires Akira, notamment pour la variante Linux. Son API Python permet d'automatiser des tâches répétitives comme la résolution des hachages d'API :

```
 import binaryninja as bn

bv = bn.open_view("akira_linux.elf")
chacha_const = b'expand 32-byte k'
addrs = list(bv.find_all_data(bv.start, bv.end, chacha_const))
print(f"ChaCha20 constants: {[hex(a) for a in addrs]}")

for addr in addrs:
    funcs = bv.get_functions_containing(addr)
    for f in funcs:
        f.name = "chacha20_init"
        print(f"Renamed: {f.name} @ {hex(f.start)}")
```

Cartographie des processus terminés par Akira

Akira termine un ensemble de processus prédéfinis avant le chiffrement pour libérer les handles sur les fichiers ciblés. La liste est stockée comme un tableau de strings dans la section .rdata :

```
 sqlservr.exe oracle.exe mysqld.exe postgres.exe mongodb.exe
veeam.BackupSvc.exe BackupExecJobEngine.exe BackupExecManagementService.exe
msexchangemailboxreplication.exe msexchangerepl.exe
vmware-vmx.exe vmware-usbarbitrator64.exe
qbdbmgrn.exe qbidpservice.exe  # QuickBooks
thebat.exe thunderbird.exe
```

Structure de la note de rançon Akira

La note de rançon d'Akira (akiranote.txt) contient la clé de session ChaCha20 chiffrée RSA-4096 encodée en base64, indispensable pour le déchiffrement via le portail Tor de négociation. Sans la clé privée RSA de l'opérateur, aucun déchiffrement n'est possible.

Analyse réseau avancée — Détection avec Zeek

Zeek est particulièrement efficace pour détecter les patterns de communication C2 d'Akira. Configurez un script de détection des connexions TLS vers des hôtes non répertoriés avec des patterns de taille de paquets caractéristiques des tunnels Tor. Complétez avec la détection des

requêtes DNS vers des domaines à haute entropie (entropie Shannon > 3.8 sur le label de domaine), indicateurs de l'algorithme DGA de fallback.

Indicateurs post-infection et forensique

Après une attaque Akira, les artefacts forensiques prioritaires sont :

Event Log Windows : EventID 7045 (service créé), 4624/4625 en masse (mouvement latéral), 1102 (logs cleared), 4688 (process creation), 19/20/21 (WMI activity), TerminalServices pour les connexions RDP entrantes

Fichiers système : Prefetch files dans C:\Windows\Prefetch (récupérables via Velociraptor même si effacés), rclone.conf dans AppData\Roaming\rclone\ (contient les identifiants d'exfiltration), fichiers temporaires dans %TEMP%

Registry : Run keys anormaux, services désactivés (VSS, backup), UAC et Firewall désactivés, traces de dump SAM via reg save

Réseau : NetFlow avec connexions TLS vers ports non-standard (Tor), HTTPS vers MEGA ou services SFTP, DNS requêtes haute entropie

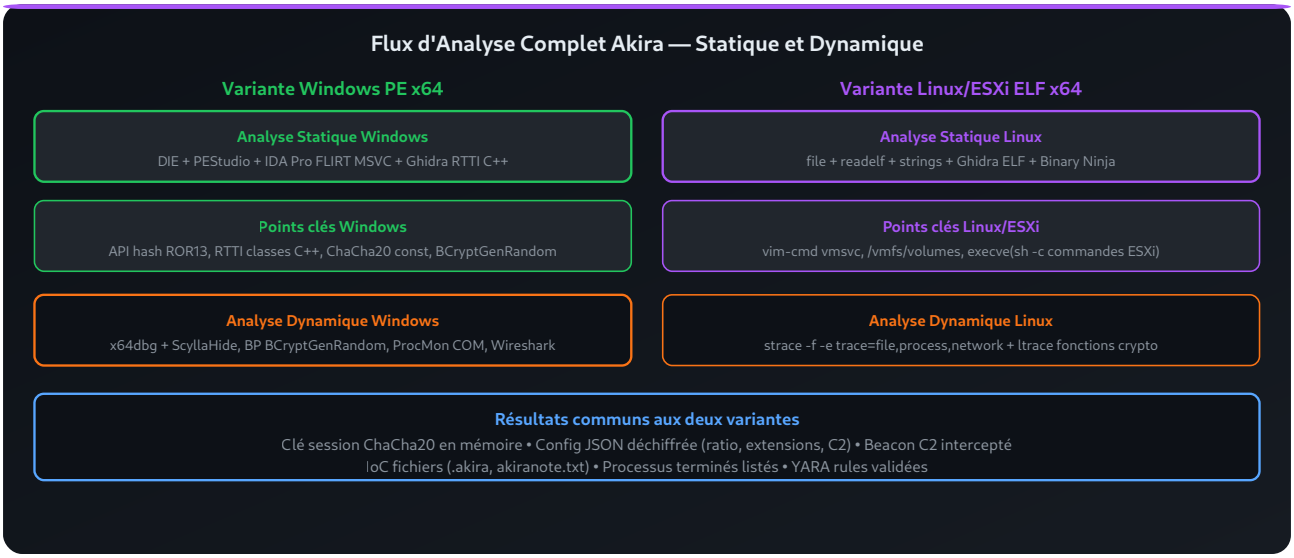

Fig. 3 — Flux d'analyse complet Akira : parallélisation de l'analyse statique et dynamique pour les variantes Windows PE et Linux/ESXi ELF.

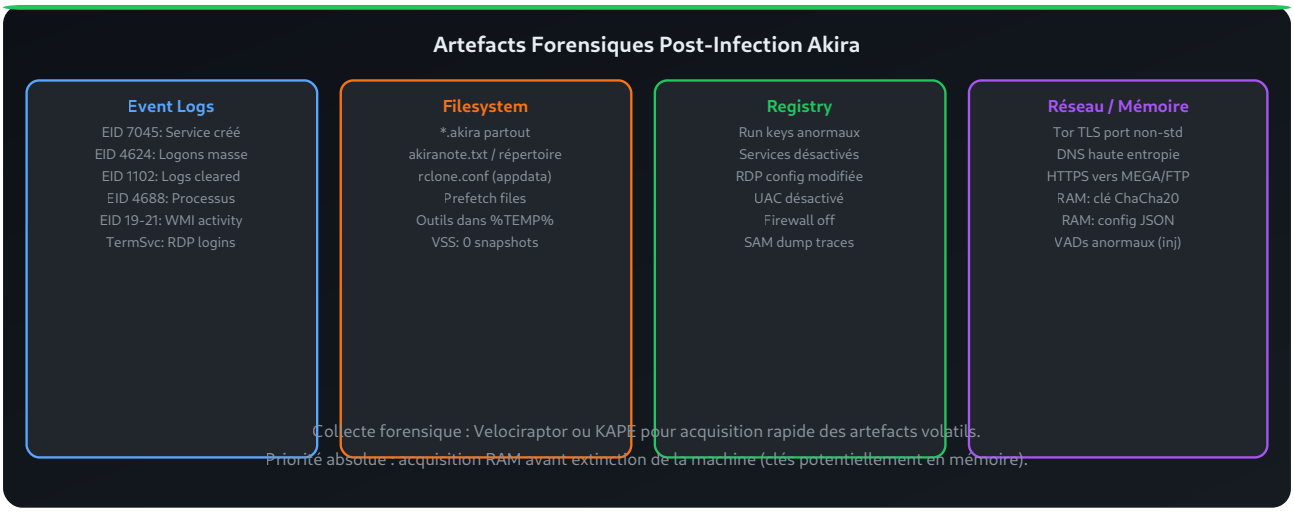

Fig. 4 — Artefacts forensiques post-infection Akira : event logs, filesystem, registry et réseau pour reconstitution d'incident.

Récupération et réponse à incident Akira

Lors d'un incident impliquant Akira, la séquence de réponse recommandée est : isolation réseau immédiate (débranchez physiquement, ne pas éteindre), acquisition RAM via WinPmem ou LiME sur les systèmes encore actifs (recherchez la clé ChaCha20 via le pattern "expand 32-byte k"), acquisition disque bit-à-bit via FTK Imager, export des event logs en priorité (souvent effacés par les affiliés), puis notification ANSSI, dépôt de plainte PJGN/PHAROS, et déclaration CNIL si données personnelles impliquées.

La phase de décontamination suit : reconstruction depuis des sauvegardes hors-ligne (si disponibles et non touchées), réinitialisation de tous les comptes AD (mots de passe et tickets Kerberos via krbtgt reset × 2), réinstallation des hôtes ESXi compromis, et audit complet des accès VPN/RDP avant réouverture.

Analyse avancée des routines de chiffrement Akira avec Ghidra

L'analyse des routines de chiffrement dans Ghidra pour Akira nécessite une approche méthodique. Une fois la constante ChaCha20 localisée, il faut reconstituer l'intégralité du flux cryptographique pour comprendre exactement comment les fichiers sont chiffrés et quelles informations sont nécessaires pour le déchiffrement.

La démarche recommandée dans Ghidra pour reconstruire le schéma cryptographique complet :

Identifiez la fonction d'initialisation ChaCha20 via la constante "expand 32-byte k". Labellisez-la `chacha20_init`. Examinez ses paramètres (clé, nonce, compteur) pour comprendre comment ils sont fournis.

Remontez les appelants de `chacha20_init` via les XREF. La fonction parente est typiquement la fonction de chiffrement d'un bloc de données — labellisez-la `chacha20_block` ou `encrypt_chunk`.

Remontez encore d'un niveau pour trouver la fonction `encrypt_file` qui gère le découpage du fichier en blocs et appelle `encrypt_chunk` pour chaque bloc.

Depuis `encrypt_file`, identifiez comment la clé ChaCha20 est fournie — elle devrait venir d'une fonction de dérivation de clé appelant BCryptGenRandom ou /dev/urandom.

Depuis la fonction de dérivation de clé, identifiez comment la clé maître de session est utilisée comme input — c'est le lien vers la couche RSA de protection.

En suivant cette démarche ascendante, vous reconstituez l'intégralité du flux cryptographique en quelques heures plutôt qu'en plusieurs jours d'analyse non structurée.

Comparaison Akira avec d'autres familles RaaS

Positionner Akira dans le paysage plus large des familles RaaS aide à contextualiser les choix techniques et à anticiper les évolutions futures :

Critère	Akira	Qilin	LockBit 3.0
Langage	C++ (Win) / C++/Rust (Linux)	Rust (tous)	C (source fuité)
Crypto symétrique	ChaCha20-Poly1305	ChaCha20-Poly1305	AES-256 + ChaCha20
Crypto asymétrique	RSA-4096	RSA-4096 / X25519	Curve25519 + AES
Chiffrement interm.	Oui (début + intervalles)	Oui (ratio par bloc)	Oui (configurable)
Ciblage ESXi	Oui (variante ELF dédiée)	Oui (Rust cross-platform)	Oui (variante Python)
Obfuscation principale	API hash ROR13 (LockBit)	Stack strings + AES config	API hash + string obfusc
Suppression VSS	IVssBackupComponents + WMI	WMI COM uniquement	Multiple méthodes

Contre-mesures spécifiques Akira recommandées

Sur la base de l'analyse technique d'Akira, les contre-mesures les plus efficaces sont :

Prévention de l'accès initial :

Patch immédiat des vulnérabilités VPN connues exploitées par Akira (Cisco AnyConnect CVE-2023-20269, Fortinet CVE-2023-27997) — le groupe les exploite activement jusqu'à 6 mois après leur divulgation

Authentification multifacteur obligatoire sur tous les accès VPN et RDP, sans exception

Désactivation du RDP sur les serveurs qui n'en ont pas besoin (`Set-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Terminal Server" -Name fDenyTSConnections -Value 1`)

Monitoring des tentatives de connexion VPN échouées avec alerte au-delà de 5 échecs/minute/IP source

Détection comportementale précoce :

Honeyfiles dans les répertoires sensibles : tout accès en écriture à ces fichiers (qui ne devraient jamais être modifiés) déclenche une alerte immédiate

Surveillance des accès en masse à des fichiers sur plusieurs volumes simultanément (threshold : > 100 fichiers/minute sur > 2 volumes)

Alertes sur les premiers accès à des paths sensibles depuis des processus non habituels (IVssBackupComponents, BCryptGenRandom depuis processus non-système)

Protection des sauvegardes :

Règle Sysmon EventID 19/20/21 pour tout accès WMI ciblant Win32_ShadowCopy ou IVssBackupComponents depuis des processus non autorisés

Sauvegardes hors-ligne sur media physique déconnecté du réseau AD (règle 3-2-1-1 : 3 copies, 2 supports différents, 1 hors-site, 1 air-gapped)

Tester la restauration mensuellement — une sauvegarde non testée n'est pas une sauvegarde

Segmentation réseau anti-propagation :

VLAN dédié pour le management ESXi, accessible uniquement depuis des jump hosts avec MFA

Micro-segmentation des workloads critiques (bases de données, systèmes de backup) avec politiques deny-by-default

Surveillance des connexions SMB latérales (source non-DC vers destination non-DC sur port 445) — indicateur fort de mouvement latéral

Scripts d'automatisation pour l'analyse Akira

Pour accélérer l'analyse de nouvelles variantes Akira, plusieurs scripts Python automatisent les tâches répétitives :

```
 #!/usr/bin/env python3
# Script d'extraction automatique des IoC d'un sample Akira
import pefile, hashlib, re, subprocess, json

def analyze_akira_sample(filepath):
    results = {}

    # Hash du sample
    with open(filepath, 'rb') as f:
        data = f.read()
    results['sha256'] = hashlib.sha256(data).hexdigest()
    results['md5'] = hashlib.md5(data).hexdigest()

    # Analyse PE
    pe = pefile.PE(filepath)
    results['compile_timestamp'] = pe.FILE_HEADER.TimeDateStamp
    results['sections'] = []
    for section in pe.sections:
        entropy = section.get_entropy()
        results['sections'].append({
            'name': section.Name.decode().strip('\x00'),
            'entropy': round(entropy, 2),
            'suspicious': entropy > 7.0
        })

    # Recherche constante ChaCha20
    chacha_const = b'expand 32-byte k'
    results['has_chacha20_static'] = chacha_const in data

    # Extraction strings ASCII (> 6 chars)
    strings_raw = re.findall(b'[\x20-\x7e]{6,}', data)
    strings = [s.decode() for s in strings_raw]

    # Filtrage des strings pertinentes
    results['suspicious_strings'] = [
        s for s in strings if any(kw in s.lower() for kw in
        ['akira', 'ransom', 'decrypt', 'bitcoin', '.onion', '/api/',
         'readme', 'shadow', 'vssadmin', 'rclone', 'vmfs'])
    ]

    # Extension de chiffrement
    akira_ext = [s for s in strings if s.startswith('.') and len(s) < 15
                 and not s.endswith(('.exe','.dll','.sys'))]
    results['potential_extensions'] = akira_ext[:10]
```

```
    print(json.dumps(results, indent=2))
    return results

if __name__ == "__main__":
    import sys
    analyze_akira_sample(sys.argv[1])
```

Ce script fournit en quelques secondes les informations de triage essentielles : hash, structure PE avec entropies par section, présence de la constante ChaCha20 (indicateur de crypto statique), et strings suspectes filtrées par mots-clés pertinents. Il constitue un excellent point de départ avant de passer à l'analyse approfondie dans Ghidra ou IDA Pro.

Évolution attendue d'Akira en 2026

Sur la base de la trajectoire observée depuis 2023 et des évolutions techniques des autres groupes RaaS de premier plan, plusieurs évolutions sont attendues pour Akira en 2026 :

Migration progressive vers Rust : La variante Linux est déjà partiellement en Rust. Une réécriture complète des deux variantes en Rust est probable pour unifier la base de code et améliorer les performances et la résistance à l'analyse.

Ciblage OT/ICS : Plusieurs groupes RaaS ont commencé à développer des modules de reconnaissance OT (Operational Technology) pour identifier et potentiellement cibler les systèmes SCADA/ICS dans les environnements industriels. Akira pourrait suivre cette tendance.

Intégration de techniques d'IA pour l'obfuscation : La génération de code polymorphe assistée par LLM est expérimentée par plusieurs groupes pour contourner les signatures YARA statiques.

Chiffrement partiel des sauvegardes cloud : Les affiliés Akira ciblent de plus en plus les buckets S3, les Azure Blob Storage, et les Backblaze B2 accessibles depuis les réseaux compromis pour rendre la récupération via cloud backup impossible.

Ressources CTI et veille sur Akira

Pour maintenir une connaissance à jour des variantes Akira et de ses TTPs, les ressources suivantes sont recommandées pour les équipes de threat intelligence :

CISA Advisory AA23-284A : Advisory conjoint FBI/CISA sur Akira avec IoC détaillés et recommandations de mitigation. Mis à jour régulièrement, c'est la référence officielle américaine.

CERT-FR CERTFR-2024-CTI-008 : Analyse technique française avec focus sur les victimes européennes et les spécificités des variantes ciblant les organisations francophones.

MalwareBazaar (abuse.ch) : Tag "akira" pour accéder aux samples récents avec hashes SHA-256, dates de soumission, et détections VirusTotal. Flux RSS disponible pour intégration SIEM.

Hybrid Analysis et ANY.RUN : Analyses comportementales des samples Akira en sandbox avec capture réseau, arborescence de processus et IoC extraits automatiquement.

Ransomwhere (ransomwhe.re) : Tracker des paiements de rançon Bitcoin/Monero permettant d'estimer les revenus du groupe et d'identifier les adresses de wallets associées.

Ransomlook.io : Monitoring automatisé du portail de fuite Akira sur Tor, avec alertes sur les nouvelles victimes publiées.

Intégrez ces sources dans votre plateforme de threat intelligence (MISP, OpenCTI, ou ThreatConnect) pour corréler automatiquement les IoC Akira avec vos événements de sécurité et recevoir des alertes proactives sur les nouvelles variantes ou victimes dans votre secteur d'activité.

À RETENIR

Points clés à retenir

Akira est un ransomware cross-platform avec deux variantes distinctes : C++ pour Windows (PE x64) et C++/Rust pour Linux/ESXi (ELF x64), déployées simultanément

pour maximiser le rayon de destruction en paralysant à la fois l'infrastructure Windows et les VMs hébergées sur ESXi.

La variante Windows hérite de techniques de LockBit 3.0 (résolution dynamique d'API par hachage ROR13, obfuscation des strings), suggérant une réutilisation du code source publié en 2022. Identifiez la fonction de résolution d'API et loggez les hashes pour labelliser automatiquement les appels dans IDA Pro.

La stratégie de chiffrement intermittent d'Akira chiffre intégralement le début de chaque fichier (2 Mo) puis applique des intervalles fixes — différent du ratio-par-bloc de Qilin, créant une signature distinctive sur analyse forensique des blocs disque.

La variante ESXi exécute des commandes vim-cmd et esxcli via execve pour arrêter les VMs avant de chiffrer les fichiers .vmdk, .vmx, .nvram — protégez votre interface de management ESXi dans un VLAN dédié accessible uniquement depuis des jump hosts avec MFA.

L'analyse statique de la variante C++ bénéficie des informations RTTI (Run-Time Type Information) qui révèlent les noms de classes (CEncryptionEngine, CVSSKiller, CC2Client) dans les binaires non strippés, accélérant considérablement la compréhension de l'architecture.

Akira utilise Rclone pour l'exfiltration pré-chiffrement vers des services cloud (MEGA, FTP) — déployez des règles Sigma sur EventID 4688 pour détecter rclone.exe avec des arguments pointant vers des services d'exfiltration.

Les règles YARA combinant la constante ChaCha20, l'extension .akira et les strings ESXi (vim-cmd vmsvc, /vmfs/volumes) offrent une détection robuste couvrant les deux variantes (PE Windows et ELF Linux).

Sources et références techniques

Cette analyse s'appuie sur les frameworks de threat intelligence publics et les advisories officiels.

MITRE ATT&CK T1486 — Data Encrypted for Impact

MITRE ATT&CK T1078 — Valid Accounts

MITRE ATT&CK T1059.001 — PowerShell

MITRE ATT&CK T1133 — External Remote Services

CISA Advisory AA23-187A — Akira Ransomware

MITRE ATT&CK — Groupes de menaces ransomware

