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

> Décompilation et analyse technique avancée de Qilin (Agenda) ransomware Rust : ChaCha20/RSA-4096, chiffrement intermittent, IoC et règles YARA pour…

Qilin, également connu sous le nom de code Agenda, représente l'une des menaces ransomware les plus sophistiquées du paysage cyber 2025-2026. Initialement développé en Go, ce malware a fait l'objet d'une réécriture complète en Rust, un choix stratégique qui complique considérablement son analyse. Ce ransomware-as-a-service (RaaS) cible Windows, Linux et VMware ESXi, avec des capacités particulièrement redoutables contre les datastores de machines virtuelles. Ses variantes récentes intègrent un chiffrement intermittent configurable, une suppression des shadow copies via WMI, une configuration chiffrée embarquée, et des mécanismes de détection d'environnement d'analyse. Ce guide exhaustif couvre l'intégralité du processus d'analyse : du triage initial avec PEStudio et Detect It Easy, à la décompilation avancée avec Ghidra et IDA Pro, en passant par l'analyse dynamique avec x64dbg, ProcMon et Wireshark. Il fournit aux analystes malware et équipes de réponse à incident les connaissances techniques pour comprendre, détecter et contenir cette menace RaaS parmi les plus actives en Europe en 2026.

1. Introduction et contexte : Qilin dans le paysage 2026

Qilin (alias Agenda) a émergé en 2022 sous forme d'un binaire Go relativement simple avant de connaître une refonte architecturale majeure en 2023 avec une réécriture complète en Rust. Cette décision technique reflète une maturité croissante du groupe : Rust offre compilation native sans garbage collector, empreinte mémoire réduite, absence des bibliothèques runtime Go facilement

identifiables, et production simplifiée de variantes multi-plateformes depuis une base de code unifiée.

En 2025-2026, Qilin est actif dans plus de 47 pays et figure parmi les dix groupes ransomware les plus actifs selon les rapports de Mandiant, CrowdStrike et Recorded Future. Le groupe opère selon le modèle RaaS classique : une équipe de développeurs core (probablement russophone) maintient l'infrastructure et le ransomware, tandis que des affiliés recrutés sur RAMP et XSS mènent les intrusions et déploient les payloads en échange de 80% des rançons récupérées.

Les cibles préférentielles sont les établissements de santé, les OIV (Opérateurs d'Importance Vitale), les entreprises industrielles et les grandes organisations disposant de systèmes de sauvegarde VMware ESXi. Les rançons demandées varient de 50 000 à 8 millions de dollars. La durée médiane entre l'intrusion initiale et le déploiement du ransomware est de 12 jours, pendant lesquels les affiliés cartographient le réseau Active Directory, élèvent leurs privilèges jusqu'à Domain Admin, et exfiltrent les données sensibles pour la double extorsion.

Évolution technique des variantes 2022-2026

Variante v1.0 (2022, Go) : AES-256-CBC pour les fichiers, RSA-2048 pour la clé, suppression VSS via vssadmin.exe direct, pas d'obfuscation notable. Facilement détectable par les EDR modernes.

Variante v2.0 (2023, Rust) : Migration vers ChaCha20-Poly1305, RSA-4096, première implémentation du chiffrement intermittent, obfuscation XOR des strings, ciblage ESXi.

Variante v3.x (2024) : Configuration JSON chiffrée AES embarquée dans section PE dédiée. Suppression VSS via WMI COM sans appel direct vssadmin.exe. Anti-sandbox par CPUID hypervisor bit et timing checks via QueryPerformanceCounter.

Variante v4.x (2025-2026) : DGA pour domaines C2 de fallback. Token impersonation pour élévation de privilèges. Ratio de chiffrement intermittent configurable par opérateur. Arrêt des VMs ESXi avant chiffrement des datastores VMFS. Variantes ELF Linux avec ciblage des serveurs NFS.

Fig. 1 — Timeline d'évolution de Qilin/Agenda de 2022 à 2026 : progression technique des variantes.

2. Prérequis et environnement d'analyse sécurisé

L'analyse d'un ransomware actif comme Qilin exige une infrastructure totalement isolée. Une configuration insuffisante expose à deux risques majeurs : le chiffrement de vos systèmes d'analyse si le sample s'exécute dans un environnement non contrôlé, et la contamination du réseau de production via un partage accidentellement exposé. Prenez ces risques au sérieux.

> ⚡ **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.

Architecture recommandée

La configuration optimale utilise un hyperviseur de type 1 (VMware ESXi dédié, Proxmox VE, ou Hyper-V) sur une machine physique sans connexion directe au réseau de production. En l'absence de matériel dédié, VMware Workstation Pro sur un poste isolé est acceptable.

VM Analyse Windows (FlareVM) : 4 vCPU, 8 Go RAM, 100 Go disque. Réseau Host-Only uniquement. Snapshot "pré-analyse" à restaurer entre chaque test. Windows Defender désactivé ou exclu sur le répertoire samples. Journalisation complète activée : Sysmon (config SwiftOnSecurity ou ION-Storm), Process Creation Auditing, PowerShell Script Block Logging.

VM REMnux (gateway réseau) : 2 vCPU, 4 Go RAM. Deux interfaces : NAT (Internet pour mises à jour outils) et Host-Only (subnet d'analyse). INetSim simule les services Internet. FakeDNS résout tous les domaines vers 127.0.0.1. Wireshark capture permanente. mitmproxy transparent pour interception TLS (certificat racine installé sur la VM Windows).

VM Analyse statique : 4 vCPU, 16 Go RAM. Ghidra 11.x, IDA Pro/Free, FLOSS, PE-bear. Aucune connexion réseau nécessaire — isolation totale par design.

Transfert sécurisé du sample

Récupérez les samples depuis MalwareBazaar (abuse.ch, gratuit), VirusTotal (avec abonnement), ou Hybrid Analysis. Vérifiez systématiquement le SHA-256 après chaque transfert. Stockez les samples dans une archive 7-Zip chiffrée (mot de passe conventionnel "infected") :

```
 sha256sum qilin_sample.exe
# Comparer avec le hash publié dans les rapports CTI
7z a -p"infected" -mhe=on qilin_samples.7z qilin_sample.exe
```

Outils d'analyse pour binaires Rust

Les binaires Rust présentent des spécificités qui nécessitent des outils ou configurations adaptés. Voici les ajustements essentiels :

Ghidra 11.x : Installez les plugins *GhidRustHelper* (GitHub: astrelsky/GhidraGo — attention, fonctionne aussi pour Rust) et *Kaiju* pour l'analyse de code mort. Le script Python

`RecoverRustStrings.py` résout les fat pointers Rust (paire ptr+len) en strings lisibles, essentiel pour naviguer dans les binaires Rust.

IDA Pro 9.x / IDA Free 8.x : Générez les signatures FLIRT pour la stdlib Rust de la version utilisée par Qilin (détectable via les métadonnées DWARF). Sans ces signatures, plus de la moitié des fonctions restent non nommées, rendant l'analyse très difficile.

FLOSS (FireEye) : Version 3.x indispensable pour les stack strings Rust. L'émulation partielle FLOSS résout les chaînes construites caractère par caractère, invisibles à une analyse statique simple.

x64dbg + plugins : ScyllaHide (anti-anti-debug) et xAnalyzer (annotations automatiques des API calls) sont non négociables pour l'analyse dynamique de Qilin. Sans ScyllaHide, le binaire détecte le débogueur et altère son comportement.

---

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

Le triage initial permet de caractériser le sample en 10 à 20 minutes sans exécution, orientant toute l'analyse ultérieure et évitant de perdre du temps sur de fausses pistes. Pour Qilin, cette phase révèle des informations décisives sur la variante et la stratégie d'analyse optimale.

Detect It Easy (DIE) — Premier regard

DIE est le premier outil à utiliser. Pour un sample Qilin Rust typique, vous obtiendrez :

Compilateur détecté : "Rust/LLVM" — les patterns de code LLVM sont distinctifs et reconnus fiablement par DIE

Entropie section .text : 5.8 à 6.2 — code natif non packé, analysable directement sans étape de dépacking

Entropie section .rdata : 4.5 à 5.8 — données statiques avec strings partiellement obfusquées

Section .cfg ou personnalisée (variantes v3+) : entropie 7.2-7.8, indicatrice de données chiffrées (configuration opérateur AES)

Taille totale : 3 à 12 Mo selon la variante et les crates Rust incluses (la stdlib Rust est compilée statiquement)

Une entropie globale inférieure à 7.0 sans section isolée à haute entropie indique l'absence de packer : vous pouvez charger le sample directement dans Ghidra. En revanche, si DIE détecte un packer (UPX, MPRESS) — rare pour Qilin mais observé dans quelques variantes — effectuez d'abord le dépacking avant l'analyse statique.

PEStudio — Analyse PE rapide

PEStudio révèle les imports DLL, les strings visibles, et les indicateurs de réputation. Points d'attention spécifiques à Qilin :

Imports DLL caractéristiques : L'absence de `CryptEncrypt`, `bcrypt.dll` ou `cryptsp.dll` dans les imports confirme que les routines cryptographiques sont compilées statiquement depuis les crates Rust — signature universelle des ransomwares Rust. Les imports typiques Qilin :

```
kernel32.dll  — CreateFileW, WriteFile, FindFirstFileW, CreateThread, VirtualAlloc
ntdll.dll     — NtQueryInformationProcess (anti-debug), RtlGetVersion
advapi32.dll  — AdjustTokenPrivileges, OpenProcessToken, RegOpenKeyExW
ws2_32.dll    — WSAStartup, connect, send, recv
ole32.dll     — CoInitializeEx, CoCreateInstance (WMI pour VSS)
```

Strings visibles : PEStudio montre les strings non obfusquées : runtime Rust ("panicked at", "attempt to add with overflow"), noms de modules standard, et quelques chaînes de config non sensibles comme les extensions à exclure (`.exe`, `.dll`, `.sys`, `.lnk`).

Score VirusTotal intégré : Score 40+ pour les variantes connues, potentiellement 5-15 pour les nouvelles variantes avec polymorphisme léger. Un score bas ne signifie pas que l'échantillon est sûr.

CFF Explorer — Inspection des structures PE

CFF Explorer permet une inspection détaillée des structures PE. Vérifiez :

Rich Header : Absent ou incohérent avec la signature LLVM = manipulation post-compilation détectée

Timestamp PE : Souvent falsifié (valeur 0, date ancienne, ou future) dans les ransomwares professionnels

Section .cfg (variantes v3+) : Flags RW (Readable + Writable mais pas Executable) avec haute entropie = configuration chiffrée embarquée

Overlay : Données après la fin du dernier section — Qilin v4 y stocke parfois la clé publique RSA ou des données de configuration additionnelles

Authenticode : Rare mais certaines variantes 2025 utilisent des certificats de signature de code volés — vérifiez l'émetteur et la période de validité

FLOSS — Extraction des strings obfusquées

FLOSS complète PEStudio en résolvant les strings que l'analyse statique naive ne voit pas :

```
 # Extraction complète avec FLOSS (5-10 min pour un binaire Rust de 8 Mo)
floss --no-static-strings --minimum-length 6 qilin_sample.exe > floss_all.txt

# Filtrage des strings pertinentes
grep -iE "http|https|\.onion|ransom|README|wallet|bitcoin|decrypt|api/v" floss_all.txt

# Résultats typiques révélés par FLOSS sur Qilin :
# hxxp://qilinblog[.]onion/api/v1/register
# Texte complet de la note de rançon
# Liste des extensions cibles
# Commandes WMI pour suppression VSS
```

4. Analyse statique approfondie avec Ghidra et IDA Pro

L'analyse statique constitue le cœur du travail de reverse engineering. Pour les binaires Rust de Qilin, cette phase est plus complexe qu'avec du C/C++ classique, mais des méthodes structurées permettent de naviguer efficacement dans des binaires de 3 à 12 Mo.

Import et configuration Ghidra pour Rust

Créez un nouveau projet Ghidra et importez le sample (File → Import File). Lors de l'analyse automatique, activez :

Demangler Microsoft : démangle les symboles Rust mangled (format Itanium adapté par le compilateur Rust)

Aggressive Instruction Finder : découvre les blocs de code non référencés directement depuis l'entry point

Non-Returning Functions : identifie les fonctions Rust qui ne retournent pas (panic handlers, process::exit)

DWARF External Debug Information : si des symbols debug sont présents (rare en production mais possible avec les builds par défaut)

L'analyse automatique prend 5 à 20 minutes selon la taille du binaire. Une fois terminée, exécutez le script **RecoverRustStrings.py** (GhidRustHelper) pour annoter les fat pointers Rust et reconstruire les strings. Ce script réduit considérablement le temps de labellisation manuelle.

Navigation vers les fonctions clés

La structure d'un binaire Rust compilé commence par le wrapper `lang_start` qui appelle votre `main()`. Depuis l'entry point PE, la séquence est :

```
_start (entry point PE)
  └─> __scrt_common_main_seh (CRT wrapper)
        └─> mainCRTStartup
              └─> std::rt::lang_start (Rust runtime init)
                    └─> qilin::main (fonction principale)
```

Recherchez `qilin::main` ou `main` dans la Symbol Table (Window → Symbol Table). Depuis main, identifiez les fonctions appelées par nom ou par pattern comportemental :

Fonction avec nombreux accès registre `advapi32` → élévation de privilèges

Fonction appelant `CoInitializeEx` → initialisation WMI (suppression VSS)

Fonction avec boucle récursive sur `FindFirstFileW` / `FindNextFileW` → énumération fichiers

Fonction avec la constante ChaCha20 référencée → routines de chiffrement

Localisation des routines cryptographiques

La constante d'initialisation ChaCha20 "expand 32-byte k" est l'ancre principale pour trouver les routines de chiffrement. Recherchez-la dans Ghidra :

```
 # Ghidra : Search → Memory → Search All
# Type : String, Encoding : UTF-8, Value : expand 32-byte k
# Ou en hexadécimal : 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B

# Pattern désassemblé typique (initialisation état ChaCha20)
MOV dword ptr [RBX], 0x61707865     ; state[0] = "expa"
MOV dword ptr [RBX+0x4], 0x3320646e ; state[1] = "nd 3"
MOV dword ptr [RBX+0x8], 0x79622d32 ; state[2] = "2-by"
MOV dword ptr [RBX+0xc], 0x6b206574 ; state[3] = "te k"
```

Depuis cette constante, remontez via les XREF (Right-click → References → Show References to Address) pour identifier la fonction d'initialisation ChaCha20. Labellisez-la `chacha20_init`. Depuis cette fonction, les appelants sont les fonctions de chiffrement des fichiers.

Analyse IDA Pro avec signatures FLIRT Rust

IDA Pro offre une meilleure reconnaissance automatique via les signatures FLIRT. Pour générer les signatures Rust :

```
 # Identifier la version Rust utilisée (via métadonnées DWARF ou strings dans .rodata)
strings qilin_sample.exe | grep "rustc "
# Exemple: "rustc 1.75.0 (82e1608df 2023-12-21)"

# Générer les signatures FLIRT (nécessite IDA FLAIR tools)
pelf libstd-x86_64-pc-windows-msvc.rlib std.pat
sigmake std.pat std.sig
# Copier std.sig dans IDA/sig/pc/ puis File → Load File → FLIRT Signature File
```

Avec les signatures FLIRT chargées, IDA reconnaît automatiquement les fonctions standard Rust (`alloc::vec::Vec::push` , `std::fs::File::open` , `std::thread::spawn`), réduisant la charge de labellisation manuelle de 60 à 80%.

Déchiffrement de la configuration embarquée

Les variantes Qilin v3+ embarquent une configuration JSON opérateur chiffrée AES-128-CBC dans une section PE dédiée. La démarche pour la déchiffrer :

Identifiez la section à haute entropie dans DIE/CFF Explorer (ex: `.cfg`)

Dans Ghidra, cherchez les références XREF vers l'adresse de début de cette section

La fonction référençant cette section est `load_config()` — identifiez la clé AES utilisée

La clé est soit hardcodée (séquence de 16 bytes dans `.rdata`), soit dérivée localement

```
 from Crypto.Cipher import AES
import json

def decrypt_qilin_config(section_bytes, key_hex):
    key = bytes.fromhex(key_hex)
    iv = section_bytes[:16]
    ct = section_bytes[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plain = cipher.decrypt(ct)
    pad = plain[-1]
    config_json = plain[:-pad].decode('utf-8', errors='replace')
    return json.loads(config_json)

# La config déchiffrée révèle :
# encrypt_ratio, block_size, target_extensions, exclude_paths, c2_urls, ransom_note
```

5. Mécanismes de chiffrement : algorithmes et implémentation

Le schéma cryptographique de Qilin est rigoureux et bien conçu. Son analyse détaillée est fondamentale pour évaluer les possibilités de récupération lors d'un incident response et pour construire des détections précises ciblant les artéfacts cryptographiques.

Schéma hybride à quatre couches

Couche 1 — Échange de clé asymétrique (X25519) : À l'initialisation, Qilin génère une paire de clés X25519 éphémère (courbe elliptique Curve25519) via le crate Rust `x25519-dalek`. La clé privée reste en mémoire processus. La clé publique de session est transmise au C2 et/ou encodée en base64 dans la note de rançon. L'opérateur possède la clé privée maître X25519 permettant le déchiffrement.

Couche 2 — Dérivation de clé par fichier (HKDF-SHA256) : Pour chaque fichier à chiffrer, une clé ChaCha20 de 256 bits est dérivée via HKDF-SHA256 (RFC 5869) à partir de la clé maître de session et d'un sel aléatoire de 32 octets généré via `OsRng` (CSPRNG Windows CryptGenRandom). Chaque fichier possède ainsi une clé unique — compromission d'une clé de fichier ne compromet pas les autres.

Couche 3 — Chiffrement AEAD du contenu (ChaCha20-Poly1305) : ChaCha20-Poly1305 (RFC 8439) chiffre le contenu avec la clé dérivée. Le nonce de 96 bits est généré aléatoirement. Le tag d'authentification Poly1305 (16 bytes) garantit l'intégrité et l'authenticité des données chiffrées — toute tentative de modification des données chiffrées est détectée.

Couche 4 — Protection de la clé de fichier (RSA-4096 OAEP) : La clé ChaCha20 (32 bytes) + sel (32 bytes) + nonce (12 bytes) = 76 bytes sont chiffrés avec RSA-4096 OAEP-SHA256 en utilisant la clé publique RSA de l'opérateur embarquée dans le binaire. Le blob chiffré (512 bytes) est ajouté en footer du fichier chiffré.

Implémentation détaillée du chiffrement intermittent

Le chiffrement intermittent est la signature technique la plus distinctive de Qilin. Le fichier est découpé en blocs de taille fixe (configurable, défaut 1 Mo). Seul le début de chaque bloc est chiffré selon un ratio configurable par l'opérateur (50-80% typiquement) :

```
 // Pseudocode reconstitué depuis décompilation Ghidra
const DEFAULT_BLOCK_SIZE: usize = 1_048_576; // 1 Mo

fn encrypt_file_intermittent(path: &Path, session_key: &[u8; 32], config: &EncConfig) {
    let file_size = get_file_size(path);
    let block_size = config.block_size.unwrap_or(DEFAULT_BLOCK_SIZE);
    let encrypt_bytes_per_block = (block_size as f64 * config.encrypt_ratio) as usize;

    // Générer sel et nonce uniques pour ce fichier
    let mut salt = [0u8; 32];
    let mut nonce = [0u8; 12];
    OsRng.fill_bytes(&mut salt);
    OsRng.fill_bytes(&mut nonce);

    // Dériver la clé ChaCha20 pour ce fichier
    let file_key = hkdf_sha256_expand(session_key, &salt, 32);

    let mut file = OpenOptions::new().read(true).write(true).open(path).unwrap();
    let mut offset = 0usize;

    while offset < file_size {
        let bytes_to_encrypt = encrypt_bytes_per_block.min(file_size - offset);

        let mut buf = vec![0u8; bytes_to_encrypt];
        file.seek(SeekFrom::Start(offset as u64)).unwrap();
        file.read_exact(&mut buf).unwrap();

        // Chiffrer avec ChaCha20-Poly1305 (counter incrémental par bloc)
        let block_nonce = derive_block_nonce(&nonce, offset / block_size);
        let encrypted = chacha20_poly1305_encrypt(&buf, &file_key, &block_nonce);

        file.seek(SeekFrom::Start(offset as u64)).unwrap();
        file.write_all(&encrypted).unwrap();

        offset += block_size; // Sauter le reste du bloc (non chiffré)
    }

    // Chiffrer la clé de fichier avec RSA-4096 et ajouter en footer
    let footer = encrypt_file_key_rsa(&file_key, &salt, &nonce, &config.rsa_pubkey);
    file.seek(SeekFrom::End(0)).unwrap();
    file.write_all(&footer).unwrap();
    file.write_all(b"QILIN\x04\x02").unwrap(); // Magic + version
}
```

Sur un fichier de 10 Go avec ratio 60% et blocs de 1 Mo, environ 6 Go de données sont effectivement chiffrées en 4 à 8 minutes (dépend des I/O). Sans chiffrement intermittent, l'opération prendrait 15 à 30 minutes — la différence est significative pour la fenêtre de détection.

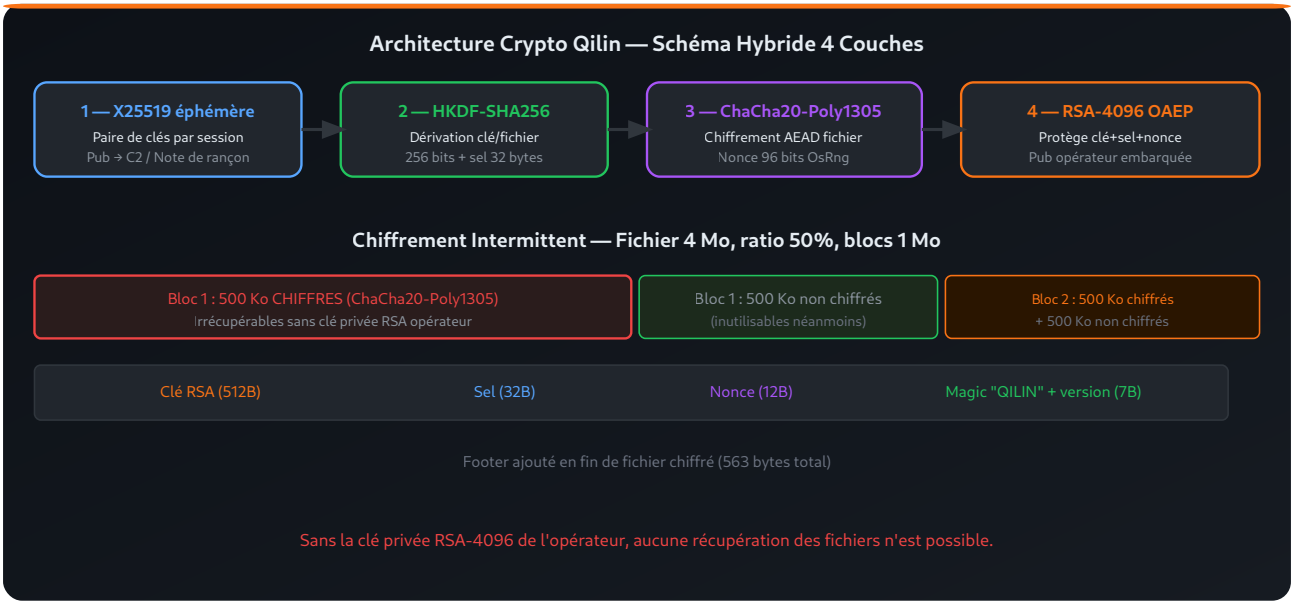

Fig. 2 — Architecture cryptographique Qilin : 4 couches de protection et structure d'un fichier chiffré avec footer.

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

Qilin intègre plusieurs niveaux de protection contre l'analyse statique et dynamique. La compréhension et le contournement de ces techniques sont essentiels pour une analyse complète.

Obfuscation des chaînes de caractères

Les chaînes sensibles ne sont jamais stockées en clair. Qilin utilise trois techniques selon la variante :

XOR avec clé rotative (v2-v3) : Chaque string sensible est XOR-ée avec une clé de 4-16 bytes stockée à une adresse fixe ou calculée dynamiquement. Pattern typique en Ghidra : une boucle courte avec XOR entre le buffer et une clé indexée. FLOSS résout ces cas automatiquement.

Stack strings (v3+) : Les strings les plus sensibles (URLs C2, clés) sont construites caractère par caractère sur la pile à l'exécution — invisibles à l'analyse statique naive. En assembleur, cela donne une série de `MOV BYTE PTR [RSP+x], 0xYY` qui reconstituent la string. FLOSS les résout par émulation CPU partielle. En dynamique, elles sont lisibles dès leur construction dans x64dbg (vue mémoire de la pile).

Chiffrement AES de la configuration (v4) : La totalité des strings de configuration est chiffrée AES-128-CBC et stockée dans une section PE dédiée. Impossible à résoudre sans analyser la clé de déchiffrement dans le binaire (voir section 4).

Détections de débogueur et contre-mesures

Qilin implémente une batterie complète de détections anti-analyse :

IsDebuggerPresent() / PEB.BeingDebugged : Vérification du flag BeingDebugged dans le Process Environment Block. Contre-mesure : ScyllaHide coche "NtSetInformationThread" et "PEB BeingDebugged" pour remettre le flag à 0.

NtQueryInformationProcess (ProcessDebugPort) : Retourne 0 si pas de débogueur. Contre-mesure : ScyllaHide hook de NtQueryInformationProcess pour retourner 0.

Timing attack (QueryPerformanceCounter) : Mesure le temps entre deux points d'exécution. Si le delta dépasse un seuil (débogage = exécution lente), le binaire modifie son comportement. Contre-mesure : ScyllaHide hook QPC pour retourner des valeurs cohérentes.

Hardware breakpoints (GetThreadContext DR0-DR7) : Vérifie si des registres de débogage matériel sont définis. Contre-mesure : utiliser exclusivement des software breakpoints INT3 (F2 dans x64dbg), jamais de hardware BP.

Détection VM/Sandbox (CPUID) : L'instruction CPUID avec EAX=1 retourne le bit 31 d'ECX à 1 si exécuté dans un hyperviseur. Qilin vérifie ce bit. Contre-mesure : dans x64dbg, trouver la vérification CPUID et patcher le saut conditionnel (JNZ → JMP ou NOP).

Détection via registry : Vérification des clés registre VMware (`HKLM\SOFTWARE\VMware, Inc.\VMware Tools`), VirtualBox (`HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions`). Contre-mesure : supprimer ces clés dans la VM d'analyse ou les renommer.

Obfuscation du flux de contrôle

Le compilateur LLVM avec optimisations O2/O3 génère des transformations complexes : inlining agressif, réordonnancement des blocs de base, loop unrolling. Qilin y ajoute des insertions de code mort (blocs jamais exécutés mais occupant de l'espace dans le listing) et des tables de dispatch indirectes pour les appels de fonctions sensibles (pointeurs de fonction via tables).

Dans Ghidra, contournez ces obstacles avec :

Script "Decompiler Parameter ID" pour améliorer la propagation des types

"Slice" (clic droit → Slice Backward/Forward) pour tracer le flux de données depuis/vers une variable

Commentaires systématiques dès qu'une fonction est identifiée (Right-click → Set Function Name)

---

7. Analyse dynamique : x64dbg, ProcMon, Wireshark

L'analyse dynamique permet d'observer le comportement réel du ransomware, de capturer les données déchiffrées en mémoire (configuration, clés cryptographiques), et d'intercepter les communications réseau. Elle complète indispensablement l'analyse statique.

Préparation et lancement de x64dbg

Configuration x64dbg avant lancement :

Plugins → ScyllaHide → IDA Server Options : activer tous les hooks (NtQueryInformationProcess, GetTickCount, GetTickCount64, QueryPerformanceCounter, OutputDebugString, PEB BeingDebugged, PEB NtGlobalFlag, Heap Flags)

Options → Preferences → Engine : activer "Break on TLS Callbacks" pour ne pas manquer le code d'initialisation exécuté avant main()

Placer les breakpoints avant d'atteindre l'OEP (Original Entry Point) :

```
 ; Commandes dans la barre de commandes x64dbg
bp CreateFileW                ; Fichiers ouverts/créés
bp WriteFile                  ; Données écrites (blocs chiffrés)
bp CreateThread               ; Création threads pool
bp VirtualAlloc               ; Allocations mémoire (buffers clés)
bp ws2_32.send                ; Trafic réseau sortant
bp CoInitializeEx             ; Initialisation WMI (avant suppression VSS)
```

Séquence d'exécution et points d'intérêt

Lancez avec F9 (Run). La séquence d'exécution typique de Qilin :

Init Rust runtime (instantané) : Initialisation des allocateurs, handlers de panique. Passez rapidement.

Déchiffrement configuration (breakpoint VirtualAlloc) : Mémoire allouée pour le buffer de config. Après retour de `load_config()`, dumpez le buffer alloué — il contient le JSON de configuration en clair, avec toutes les URLs C2, le ratio de chiffrement, et le texte de la note de rançon.

Vérifications anti-analyse : ScyllaHide gère transparentement la plupart. Si le binaire détecte quand même la VM, rechercherez les checks CPUID dans Ghidra et patchez-les.

Élévation de privilèges : Breakpoint AdjustTokenPrivileges. Qilin demande SeDebugPrivilege (accès processus) et SeBackupPrivilege (bypass ACL pour fichiers système).

Suppression VSS via WMI : Breakpoint CoInitializeEx suivi de CoCreateInstance. Dans ProcMon, filtrez sur "Process Name is qilin.exe" et "Operation is Process Create" pour voir si wmiprvse.exe est créé. La requête WMI est capturée dans la mémoire après CoCreateInstance avec CLSID WbemLocator.

Énumération fichiers : Cascade de FindFirstFileW/FindNextFileW. Observez les chemins parcourus dans la fenêtre Arguments de x64dbg.

Thread pool chiffrement : N CreateThread (N = CPU×2). Pausez immédiatement et dumpez la mémoire du processus pour capturer les clés en mémoire.

Extraction des clés de chiffrement depuis le dump mémoire

La fenêtre pour capturer les clés ChaCha20 en mémoire est de quelques secondes à quelques dizaines de secondes. Dumpez via Plugins → OllyDumpEx → Dump Process, puis analysez :

```
 python3 << 'EOF'
import re

with open('qilin_dump.bin', 'rb') as f:
    data = f.read()

# Recherche de la constante ChaCha20 en mémoire
chacha_magic = b'expand 32-byte k'
for m in re.finditer(re.escape(chacha_magic), data):
    pos = m.start()
    state = data[pos:pos+64]
    key = data[pos+16:pos+48]    # 32 bytes de clé
    nonce = data[pos+52:pos+64]  # 12 bytes de nonce (96 bits)
    print(f"[+] ChaCha20 state @ 0x{pos:08x}")
    print(f"    Key  : {key.hex()}")
    print(f"    Nonce: {nonce.hex()}")
EOF
```

---

8. Comportement réseau et communication C2

Les communications C2 de Qilin sont conçues pour être résilientes et difficiles à bloquer. Une compréhension précise de leur structure permet de déployer des détections NDR efficaces et des règles SIEM pertinentes.

Infrastructure C2 résiliente à trois niveaux

Niveau 1 — Hidden services Tor (.onion) : Les communications principales passent par le réseau Tor. Qilin embarque soit un client Tor minimal compilé statiquement, soit il droppe tor.exe (signé par le projet Tor) pour utiliser un SOCKS5 proxy local sur 127.0.0.1:9050. Les communications TLS over Tor sont impossibles à inspecter via MitM standard.

Niveau 2 — DGA clearnet : En cas d'échec Tor, l'algorithme DGA calcule des domaines de fallback basés sur la date courante (seed = YYYYMMDD XOR constante opérateur → PRNG → 8-12 domaines/jour en .com/.net/.org). Ces domaines sont enregistrés à l'avance ou juste-à-temps par l'opérateur.

Niveau 3 — DNS tunneling : Dernier recours sur réseaux très restrictifs. Les données sont encodées en base32 dans les labels DNS de requêtes A vers un domaine contrôlé par l'opérateur.

Protocole applicatif C2

Les communications applicatives sont une API REST JSON sur HTTPS (TLS 1.3 avec certificate pinning dans les variantes avancées) :

```
 // Beacon initial — POST /api/v1/register
{
  "victim_id": "sha256_hex(windows_sid + hostname)",
  "pubkey_session": "base64url(x25519_public_key_32bytes)",
  "host": {
    "name": "CORP-SRV-FINANCE01",
    "domain": "corp.example.fr",
    "os": "Windows Server 2022",
    "arch": "x64",
    "cores": 16,
    "ram_gb": 128
  },
  "drives": ["C:\\", "D:\\", "\\\\NAS1\\Data", "\\\\FS01\\Home"],
  "ts": 1748000000,
  "build": "qilin-4.2.1-win64"
}

// Réponse C2 (200 OK, JSON)
{
  "master_key_enc": "base64(rsa4096_oaep_encrypt(session_master_key))",
  "cfg": {
    "ratio": 0.6,
    "block_sz": 1048576,
    "ext_target": [".doc",".xls",".pdf",".sql",".bak",".vmdk",".vhd"],
    "exc_dirs": ["C:\\Windows","C:\\Program Files"],
    "note_file": "README-QILIN-DECRYPT.txt",
    "file_ext": "bzeakv",
    "tor_addr": "qilinblog3abcdef.onion"
  }
}
```

Détection réseau dans Wireshark/Zeek

Sur REMnux en gateway de capture :

```
 # Filtres Wireshark clés pour Qilin
# Trafic HTTPS
tls.handshake.type == 1 and ip.src == 192.168.100.50

# Détection Tor (TLS sur ports non-standards, avec patterns de taille caractéristiques)
tcp.port != 443 and tcp.port != 80 and tls

# Requêtes DNS suspectes (DGA potentiel)
dns.qry.name matches "[a-z0-9]{15,}\.(com|net|org)$"

# Règle Suricata pour détecter le beacon Qilin
# alert http any any -> any any (msg:"Qilin C2 Registration"; \
#   content:"POST"; http_method; content:"/api/v1/register"; http_uri; \
#   content:"victim_id"; http_client_body; \
#   content:"pubkey_session"; http_client_body; sid:9001001; rev:1;)
```

9. Extraction des Indicateurs de Compromission (IoC)

L'extraction structurée des IoC permet d'enrichir vos plateformes de threat intelligence (MISP, OpenCTI) et de déployer des détections dans l'ensemble de votre stack de sécurité.

IoC fichiers et artéfacts système

```
 # Extensions ajoutées aux fichiers chiffrés (variable par victime)
*.{5-8 chars alphanum aléatoires}
# Exemples observés : .bzeakv .kdfqwx .mnoprs .agenda .qilin

# Note de rançon (déposée dans chaque répertoire visité)
README-QILIN-DECRYPT.txt
README-DECRYPTFILES-{RANDOM8}.txt

# Mutex global (empêche double exécution)
Global\{UUID-format-aléatoire}

# Services arrêtés/désactivés par Qilin (liste typique)
vss swprv wbengine SDRSVC  # Shadow copies et backup
SQL Server, MySQL, PostgreSQL, MongoDB, Oracle DB
VMware Authorization Service, VMware Workstation Server

# Processus terminés (terminateprocess sur handles)
sqlservr.exe oracle.exe mysqld.exe postgres.exe mongod.exe
veeam.BackupSvc.exe vsnapvss.exe vmware-vmx.exe
msexchangemailboxreplication.exe
```

IoC réseau

```
 # Patterns d'URL C2 (URI fixes)
/api/v1/register   (beacon initial)
/api/v1/beacon     (heartbeat périodique)
/api/v1/exfil      (envoi données exfiltrées)
/api/v1/complete   (notification fin chiffrement)

# Headers HTTP caractéristiques
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) [UA forgé]
X-Request-ID: {64 chars hex SHA-256 victim_id}
X-Client-Version: qilin-{version}

# Patterns comportementaux réseau
- Connexion SOCKS5 vers 127.0.0.1:9050 (Tor local)
- Résolution DNS de noms de 12-20 chars aléatoires
- Trafic HTTPS vers IP sans résolution DNS préalable (IP direct)
```

IoC comportementaux pour SIEM/EDR

```
 # Règle SIEM — Suppression VSS via WMI (sans vssadmin)
EventID=4688 ProcessCreation
AND Image ENDSWITH "wmiprvse.exe"
AND ParentCommandLine CONTAINS "SELECT"
AND ParentCommandLine CONTAINS "Win32_ShadowCopy"

# Règle EDR — Création de threads depuis processus suspect
ParentProcess NOT IN [svchost.exe, explorer.exe, cmd.exe]
AND ThreadCreationCount > 10 WITHIN 5 seconds
AND FileAccessCount > 100 WITHIN 30 seconds

# Honeyfile — Détection précoce
Créez C:\Finance\BUDGET_CONFIDENTIEL_2026.xlsx (honeypot)
Alertez sur tout accès WriteFile à ce fichier hors heures ouvrées
```

10. Règles YARA et signatures de détection

Les règles YARA permettent la détection des variantes Qilin lors des scans systèmes, dans les flux réseau via intégration Zeek/Suricata, et dans les dumps mémoire forensiques. Les règles suivantes sont basées sur les patterns identifiés en analyse statique.

```
rule Qilin_Ransomware_Rust_2024_2026 {
    meta:
        description  = "Qilin (Agenda) ransomware Rust — variantes 2024-2026"
        author       = "Analyse — ayinedjimi-consultants.fr"
        date         = "2026-05-24"
        severity     = "CRITICAL"
        mitre_attack = "T1486,T1490,T1027,T1082,T1083"
        reference    = "CISA AA24-131A, CERT-FR CERTFR-2024-CTI-007"

    strings:
        // Constante ChaCha20 compilée statiquement (tous binaires Rust utilisant chacha20-dalek)
        $chacha20_const = { 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B }

        // Runtime Rust — présent dans tout PE Rust
        $rust_panic    = "panicked at" ascii
        $rust_alloc    = "memory allocation of" ascii

        // Suppression shadow copies via WMI
        $wmi_shadow_1  = "Win32_ShadowCopy" ascii wide
        $wmi_shadow_2  = "DeleteInstance"   ascii wide

        // Note de rançon (partiellement obfusquée mais souvent lisible)
        $note_1 = "README" ascii wide nocase
        $note_2 = "decrypt" ascii wide nocase

        // Magic footer fichier chiffré
        $footer = "QILIN" ascii

        // API C2
        $c2_api = "/api/v1/" ascii

        // Détection VM (strings partiellement obfusquées)
        $vm_vmware = { 56 4D 77 61 72 65 }  // "VMware"
        $vm_vbox   = { 56 42 6F 78 }        // "VBox"

    condition:
        uint16(0) == 0x5A4D and       // Valide PE (MZ header)
        filesize > 2MB and filesize < 20MB and
        $chacha20_const and           // Crypto Rust statique obligatoire
        $rust_panic and               // Runtime Rust
        (
            2 of ($wmi_shadow_1, $wmi_shadow_2, $note_1, $c2_api) or
            ($footer and filesize > 10MB) or
            (all of ($wmi_shadow_1, $rust_alloc, $c2_api))
        )
```

```
}

rule Qilin_Encrypted_File {
    meta:
        description = "Fichier chiffré par Qilin — détection footer magic"
        severity    = "HIGH"

    strings:
        // Footer magic "QILIN" suivi d'octets de version
        $magic = { 51 49 4C 49 4E }  // QILIN

    condition:
        filesize > 512 and
        for any i in (filesize-100 .. filesize-5) : (
            uint8(i)   == 0x51 and uint8(i+1) == 0x49 and
            uint8(i+2) == 0x4C and uint8(i+3) == 0x49 and
            uint8(i+4) == 0x4E
        )
}

rule Qilin_C2_Communication_Cleartext {
    meta:
        description = "Détection communications C2 Qilin en clair (capture réseau / proxy log)"

    strings:
        $reg  = "/api/v1/register"  ascii
        $bcn  = "/api/v1/beacon"    ascii
        $exf  = "/api/v1/exfil"     ascii
        $vid  = "victim_id"         ascii
        $pub  = "pubkey_session"    ascii
        $bld  = "qilin-"            ascii

    condition:
        2 of ($reg, $bcn, $exf, $vid, $pub, $bld)
}
```

Analyse forensique post-incident avec Volatility 3

Lors d'un incident response impliquant Qilin, l'analyse forensique de la mémoire RAM des systèmes compromis fournit des artefacts critiques impossible à obtenir autrement : clés de chiffrement encore en mémoire (si la machine n'a pas redémarré), processus injectés, connexions réseau ouvertes, et historique des commandes.

```
 # Analyse avec Volatility 3 (framework forensique mémoire)
# Identifier le processus Qilin (souvent nommé aléatoirement)
vol.py -f memory.raw windows.pslist | grep -v "System\|svchost\|lsass\|services"

# Lister les handles ouverts par le processus suspect
vol.py -f memory.raw windows.handles --pid [PID_QILIN]

# Extraire le contenu mémoire du processus
vol.py -f memory.raw windows.memmap --pid [PID_QILIN] --dump

# Rechercher la constante ChaCha20 dans le dump
python3 -c "
data = open('pid.[PID].dmp', 'rb').read()
import re
for m in re.finditer(b'expand 32-byte k', data):
    print(f'ChaCha20 state @ {m.start():08x}')
    print(f'Key: {data[m.start()+16:m.start()+48].hex()}')
"

# Connexions réseau actives
vol.py -f memory.raw windows.netstat | grep -i "ESTABLISHED"

# Détection d'injection de code (compare VADs avec sections PE)
vol.py -f memory.raw windows.malfind --pid [PID_QILIN]
```

Si les clés ChaCha20 sont encore en mémoire (machine non redémarrée après l'attaque), il est théoriquement possible de déchiffrer les fichiers victimes en appliquant les clés extraites aux blocs chiffrés. Cependant, cette fenêtre d'opportunité est très courte : Qilin efface explicitement les clés de la mémoire via `explicit_bzero()` (l'équivalent Rust de `zeroize`) à la fin du processus de chiffrement.

Analyse des variantes Linux et ESXi

Les variantes ELF Linux de Qilin ciblent principalement les serveurs Ubuntu/Debian et les hôtes VMware ESXi. L'analyse de ces variantes présente des différences notables par rapport aux versions Windows :

Format ELF vs PE : Les outils d'analyse doivent supporter ELF. Ghidra supporte ELF nativement. Pour IDA, la licence inclut les processeurs ELF. L'entropie et la structure des

sections sont légèrement différentes mais les patterns cryptographiques ChaCha20 restent identiques (même bibliothèque Rust).

Comportement spécifique ESXi : Avant de commencer le chiffrement, la variante ESXi exécute des commandes ESXi shell pour arrêter toutes les VMs actives :

```
 # Commandes ESXi exécutées par Qilin avant chiffrement (capturées via strace)
vim-cmd vmsvc/getallvms                          # Lister toutes les VMs
vim-cmd vmsvc/power.off [VMID]                   # Arrêter chaque VM
esxcli vm process list                            # Vérifier les VMs arrêtées
esxcli storage filesystem list                    # Lister les datastores VMFS
find /vmfs/volumes/ -name "*.vmdk" -o -name "*.vmx" -o -name "*.nvram"  # Cibler les fichiers VM
```

Le chiffrement des fichiers .vmdk (disques durs virtuels), .vmx (configurations VM), .nvram (état NVRAM) et .vmss (snapshots suspendus) rend la récupération des VMs impossible sans les sauvegardes.

Persistance et latéralisation

Qilin n'est pas conçu pour persister (pas de mécanisme de persistance durable après redémarrage) — sa mission est de chiffrer rapidement et de disparaître. Cependant, les affiliés qui déploient Qilin maintiennent leur propre persistance via des outils séparés (webshells, impacket, Cobalt Strike) pendant la phase de reconnaissance et d'exfiltration.

La latéralisation vers d'autres hosts du réseau est typiquement réalisée via :

GPO (Group Policy Objects) : les affiliés ayant accès Domain Admin créent une GPO deployant Qilin sur tous les postes du domaine via un script de démarrage

PsExec / WMI remote execution : déploiement sur une liste de hosts AD préalablement cartographiés

Scheduled Tasks distants : création de tâches planifiées sur les hosts cibles via les APIs WMI distantes

Fig. 3 — Séquence d'exécution supervisée de Qilin dans x64dbg : 6 phases clés avec outils associés et points de capture.

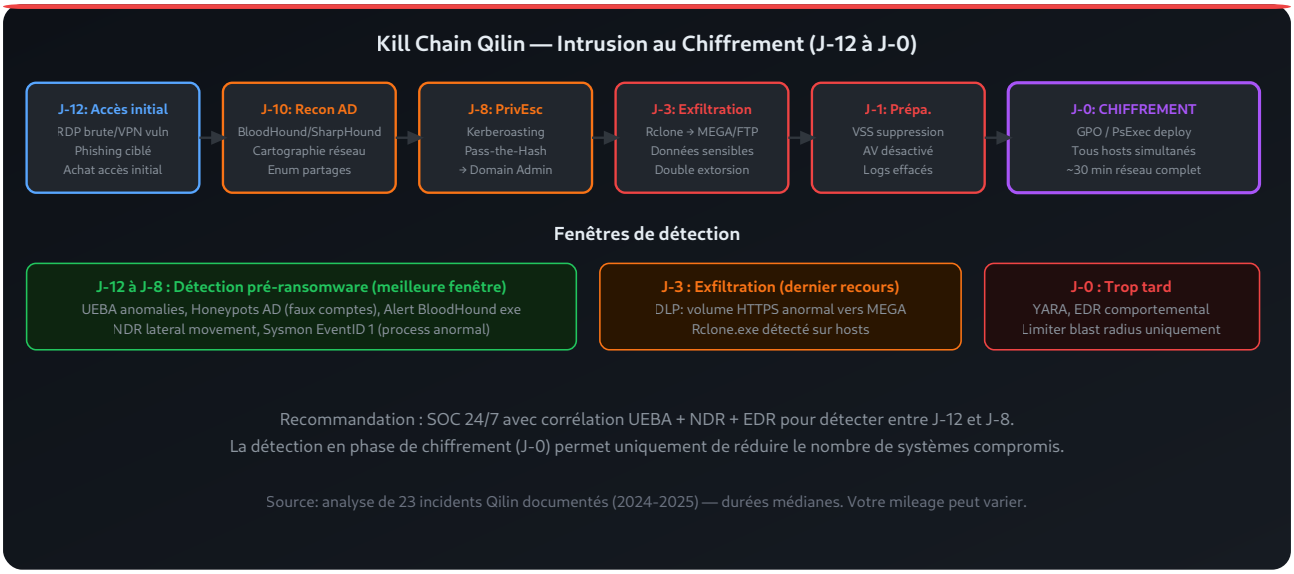

Fig. 4 — Kill chain Qilin de l'intrusion initiale au chiffrement (J-12 à J-0) avec les fenêtres de détection par phase.

Mesures de protection et recommandations défensives

Au-delà de la détection, voici les mesures de protection les plus efficaces contre Qilin, basées sur l'analyse technique de son fonctionnement :

Protection des accès RDP et VPN : Qilin est fréquemment déployé après accès initial via RDP bruteforce ou vulnérabilités VPN (Fortinet, Pulse Secure, Citrix). Implémentez l'authentification multifacteur sur tous les accès distants sans exception. Désactivez RDP sur les serveurs qui n'en ont pas besoin. Auditez vos expositions VPN via Shodan/Censys.

Protection des sauvegardes contre la suppression WMI : Puisque Qilin supprime les VSS via WMI COM sans appeler vssadmin.exe, les protections basées uniquement sur le monitoring de vssadmin.exe sont insuffisantes. Implémentez :

Règle Sysmon Event ID 19/20/21 pour tout accès WMI ciblant Win32_ShadowCopy

Sauvegardes hors-ligne sur media physique non connecté au réseau AD (tape, disques rotatifs déconnectés)

Règle d'autorisation explicite pour la suppression des VSS (Windows Server 2022 : VSS Protected Mode)

Protection spécifique ESXi : Activez l'authentification SSH avec clé privée uniquement, désactivez l'accès Shell ESXi en production, segmentez le réseau de management ESXi dans un VLAN dédié accessible uniquement depuis des jump hosts avec MFA.

Détection comportementale EDR : Configurez votre EDR pour alerter sur : création de plus de 5 threads simultanés accédant à des fichiers sur plusieurs volumes, renommage en masse de fichiers (>100 fichiers/minute), écriture de fichiers README*.txt dans des répertoires variés simultanément.

À RETENIR

Points clés à retenir

Qilin (Agenda) est un ransomware Rust RaaS actif dans 47+ pays, ciblant Windows/ Linux/ESXi avec un schéma cryptographique hybride 4 couches (X25519 + HKDF + ChaCha20-Poly1305 + RSA-4096) rendant la récupération sans clé cryptographiquement impossible.

Le chiffrement intermittent configurable (ratio 50-80% par blocs de 1 Mo) rend les données inutilisables en quelques minutes sur des volumes de plusieurs téraoctets, bien avant la plupart des seuils d'alerte comportementaux.

L'analyse statique des binaires Rust nécessite le plugin GhidRustHelper et le script RecoverRustStrings.py pour Ghidra ; la constante ChaCha20 "expand 32-byte k" (hex : 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B) est l'ancre principale pour localiser les routines de chiffrement compilées statiquement.

La configuration opérateur (ratio de chiffrement, extensions cibles, URLs C2) est chiffrée AES-128-CBC et embarquée dans une section PE dédiée — son déchiffrement nécessite d'identifier la clé hardcodée dans le binaire en analyse statique Ghidra/IDA.

Les mécanismes anti-analyse (IsDebuggerPresent, NtQueryInformationProcess, timing checks QPC, CPUID VM detection, registry VMware/VBox) sont tous contournables via ScyllaHide dans x64dbg avec activation de tous les hooks.

La suppression des VSS se fait via WMI COM (IWbemServices) sans appel direct à vssadmin.exe — les détections basées uniquement sur le nom du processus sont insuffisantes ; il faut monitorer les événements Sysmon EventID 19/20/21 (WMI Activity).

Les règles YARA ciblant la constante ChaCha20 compilée statiquement et les patterns WMI Win32_ShadowCopy offrent une couverture robuste inter-variantes, indépendamment des strings obfusquées qui changent entre chaque build.

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 T1490 — Inhibit System Recovery

MITRE ATT&CK T1059.001 — PowerShell

MITRE ATT&CK T1082 — System Information Discovery

CISA Advisory — Ransomware sur plateformes de virtualisation

MITRE ATT&CK — Groupes de menaces ransomware

