Ransomware Analyse & Simulation

Von der Bedrohungslage zur technischen Implementierung in Rust

Demo Präsentation

21. Dezember 2025

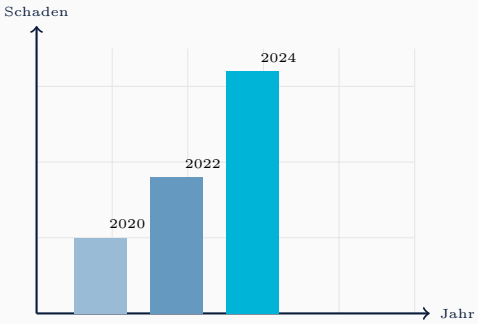
Die Bedrohungslage

Ransomware: Status Quo

Was ist Ransomware? Schadsoftware, die Daten verschlüsselt und Lösegeld (Krypto-Währung) für die Entschlüsselung fordert.

Aktuelle Trends:

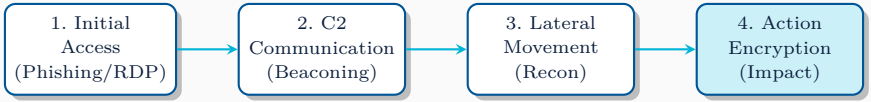
- RaaS: Ransomware-as-a-Service (Miet-Malware).
- Double Extortion: Verschlüsselung + Drohung mit Datenveröffentlichung.
- Zielgerichtete Angriffe: Fokus auf Kritische Infrastruktur (KRITIS) & Industrie.



Exponentieller Anstieg der Schäden

Die ”Cyber Kill Chain” einer Attacke

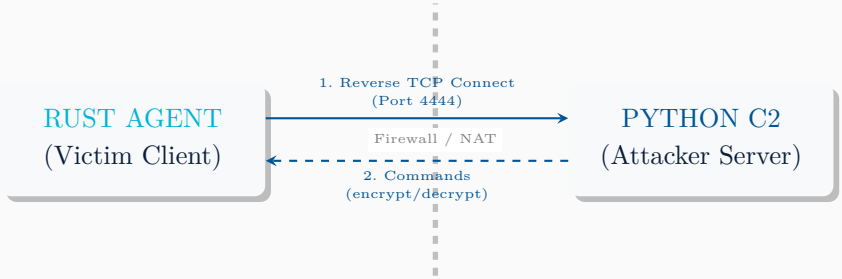
Der typische Ablauf eines modernen Ransomware-Angriffs:



Hinweis: In unserer Simulation fokussieren wir uns auf Schritt 2 (C2) und Schritt 4 (Encryption).

Analyse der Simulation

Architektur der Simulation



- Reverse Shell Prinzip: Der Agent baut die Verbindung auf, nicht der Server. Das umgeht eingehende Firewall-Regeln.
- Technologie: Rust (Client) für Performance und Sicherheit, Python (Server) für einfache Handhabung.

Der Agent: Warum Rust?

Rust etabliert sich zunehmend in der Malware-Entwicklung (”Rustyware”).

Technische Vorteile

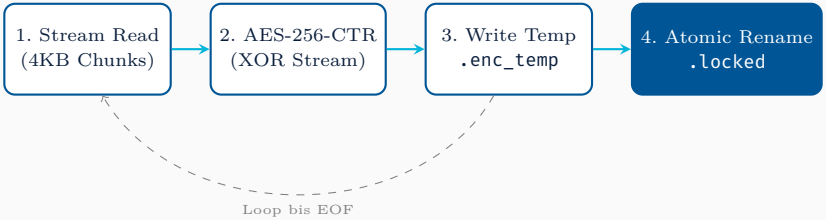
- Memory Safety: Keine Buffer Overflows (weniger Abstürze beim Opfer).
- Zero Runtime: Keine Installation von Python/Java nötig.
- Cross-Platform: Ein Code für Linux, Windows & macOS.

Evasion (Erkennung)

- Komplexer Maschinencode erschwert Reverse Engineering.
- Geringere Erkennungsrate bei klassischen AV-Signaturen im Vergleich zu C/C++.

Kryptographie: Der ”Atomic Lock”

Ein kritischer Fehler bei Malware ist Datenverlust durch Abstürze. Unser Agent nutzt ein atomares Verfahren:



AES-256-CTR: Der Counter-Mode ermöglicht schnelles Verschlüsseln ohne ”Padding” (Dateigröße bleibt identisch).

Persistenz: Das Überleben des Neustarts

Malware muss sich im System verankern. Der Rust-Code unterscheidet zur Kompilierzeit das OS:

Windows (Registry)

```
Command::new("reg")
    .args([
        "add",
        "HKCU\\...\\Run",
        "/v", "Updater",
        "/d", exe_path
    ])
```

Linux (Systemd)

```
[Unit]
Description=SystemSvc

[Service]
ExecStart=/path/to/agent
Restart=always
```

Live Demo Workflow

Ablauf der Demonstration

Wie die Komponenten im Test zusammenspielen:

1. Setup: Starten des Python C2 Servers (Listener auf Port 4444).
2. Infektion: Ausführen der `rust-mw` Binary auf dem Zielsystem.
3. Handshake: Client meldet sich beim Server (”New Session”).
4. Angriff:
   - Server sendet Befehl: `encrypt`
   - Client generiert Key → Verschlüsselt Dateien → Zeigt Lösegeldforderung.
5. Lösung: Server sendet Befehl `decrypt` (simulierte Zahlung).

Fazit

Simpel: Sockets + Standard-Krypto reichen für maximalen Schaden.

Effizient: Rust ermöglicht extrem schnelle, schwer erkennbare Malware.

Relevant: Backup-Strategien sind der einzige wirkliche Schutz.