RustReaper

RustReaper is a powerful, cross-platform memory forensic analyzer designed to detect malware artifacts in memory dumps or live processes. Built with Rust for performance and safety, it supports Windows, Linux, and macOS, offering both a command-line interface (CLI) and a web-based graphical user interface (GUI) for real-time analysis. RustReaper excels in identifying hooks, shellcode, encrypted payloads, and other suspicious patterns using YARA rules, entropy analysis, and disassembly.

Features

Cross-Platform Support: Analyzes memory on Windows (VirtualQueryEx), Linux (/proc), and macOS (Mach APIs).
Flexible Analysis Modes:
quick: Fast scanning for common artifacts.
deep: Comprehensive analysis with YARA and entropy.
stealth: Low-impact scanning for live systems.

Dynamic YARA Rules: Load custom rules for tailored malware detection.
Real-Time GUI: React-based interface with dual progress bars (parsing/analysis) and artifact visualization.
SQLite Storage: Persists artifacts for consolidated reporting.
Report Generation: Outputs in JSON and HTML formats with context data.
Unconventional Tactics:
Real-time artifact streaming to GUI via WebSocket.
Placeholder for ML-based anomaly detection (entropy thresholds).
Memory snapshot diffing (planned).

Architecture

RustReaper is modular, with the following components:

main.rs: CLI entry point, orchestrates parsing, analysis, and reporting.
parser.rs: Extracts memory regions from dumps or live processes.
analyzer.rs: Detects artifacts using YARA, disassembly, and entropy.
output.rs: Generates JSON/HTML reports.
models.rs: Defines MemoryRegion and Artifact structs.
gui_server.rs: Actix-web server for the React GUI.
web/: React SPA with Tailwind CSS for visualization.
rules/: YARA rules for malware pattern matching.

Installation

Prerequisites

Rust: Version 1.70+ (install via rustup).
SQLite:
Windows: Install via winget install SQLite.SQLite.
Ubuntu: sudo apt-get install libsqlite3-dev.
macOS: brew install sqlite.

Web Browser: For GUI (Chrome, Firefox recommended).
Optional: Docker for isolated builds.

Beginner Setup
Option 1: Download Pre-Built Binary

Visit the Releases page.
Download the zip/tarball for your platform (e.g., rustreaper--windows.zip).
Extract to a directory (e.g., C:\RustReaper or ~/rustreaper).
Run the binary:
Windows: .\rustreaper.exe --help
Linux/macOS: ./rustreaper --help

Option 2: Build from Source

Clone the repository:git clone https://github.com/SunnyThakur25/rustreaper.git
cd rustreaper

Run the build script:
Windows (PowerShell):.\build.ps1

Linux/macOS:chmod +x build.sh
./build.sh

Find the binary and assets in dist/:
Run: dist/rustreaper --help (or dist\rustreaper.exe).

Advanced Setup
Custom Build

Install Rust nightly for advanced features:rustup install nightly
rustup default nightly

Clone and build with custom flags:git clone https://github.com/SunnyThakur25/rustreaper.git
cd rustreaper
cargo build --release --features "advanced-ml"

Cross-compile (e.g., for Windows on Linux):rustup target add x86_64-pc-windows-gnu
cargo build --release --target x86_64-pc-windows-gnu

Docker Setup

Build the Docker image:docker build -t rustreaper .

Run in a container:docker run -p 8080:8080 -v $(pwd)/dumps:/dumps rustreaper

Dockerfile Example:
```
FROM rust:1.70
WORKDIR /usr/src/rustreaper
COPY . .
RUN apt-get update && apt-get install -y libsqlite3-dev
RUN cargo build --release
EXPOSE 8080
CMD ["target/release/rustreaper", "serve"]
```

Usage
CLI Commands
Run rustreaper --help for details. Key commands:

Analyze a Memory Dump:
rustreaper analyze ./dump.bin --yara-rules ./rules/rules.yara --profile deep --output-dir ./reports

Output: JSON report in ./reports/memory_analysis.json.

Scan a Live Process:
rustreaper scan --live --pid 1234 --profile stealth

Output: JSON report in ./reports/live_scan_1234.json.

Generate Consolidated Report:
rustreaper report --format html --include-context

Output: HTML report in ./reports/consolidated_report.html.

Start GUI Server:
rustreaper serve --addr 0.0.0.0:8080

Access: http://localhost:8080.

List Profiles:
rustreaper profiles

GUI Usage

Start the server:rustreaper serve

Open http://localhost:8080 in a browser.
Features:
Dual progress bars for parsing and analysis.
Interactive artifact table with type filtering.
Download JSON reports via the "Download Report" button.

Sample YARA Rule (rules/rules.yara)
rule shellcode {
meta:
description = "Common shellcode patterns"
strings:
$xor_decrypt = { 80 30 ?? 40 }
$get_pc = { E8 00 00 00 00 5? }
$syscall = { 0F 05 }
condition:
any of them
}

Sample Memory Dump
Generate a sample dump for testing:

Linux/macOS

dd if=/dev/urandom of=sample_dump.bin bs=4K count=1

Windows (PowerShell)

$sampleData = [byte[]]::new(0x1000); 0..0xFFF | % { $sampleData[$_] = Get-Random -Max 256 };
[System.IO.File]::WriteAllBytes("sample_dump.bin", $sampleData)

Modifications
Code Structure

Extend YARA Rules:
Add rules to rules/rules.yara.
Test with rustreaper verify-yara ./rules/rules.yara (requires implementation).

Add New Artifact Types:
Update models.rs to include new ArtifactType variants.
Modify analyzer.rs to detect new patterns.

Enhance GUI:
Edit web/static/ProgressBar.jsx for new visualizations (e.g., entropy graphs).
Update gui_server.rs for additional WebSocket messages.

ML Integration:
Add a dependency (e.g., tch-rs for PyTorch).
Implement anomaly detection in analyzer.rs (e.g., neural network for entropy patterns).

Example: Adding a New Artifact Type

Update models.rs:#[derive(Debug, Serialize, Deserialize, PartialEq)]
pub enum ArtifactType {
// Existing types...
CustomMalware,
}

Update analyzer.rs:fn detect_custom_malware(&self, region: &MemoryRegion) -> Result<Vec,
AnalysisError> {
let mut artifacts = Vec::new();
if region.data.contains(&b"CUSTOM_PATTERN"[..]) {
artifacts.push(Artifact {
address: region.base_address,
artifact_type: ArtifactType::CustomMalware,
description: "Custom malware pattern detected".to_string(),
confidence: 0.9,
entropy: None,
context: None,

```
});
}
Ok(artifacts)
}
```

Call in analyze_internal:artifacts.extend(self.detect_custom_malware(region)?);

Building Modifications

Rebuild after changes:./build.sh # or .\build.ps1

Test modifications:cargo test

Contributing

Fork the repository.
Create a feature branch (git checkout -b feature/new-feature).
Commit changes (git commit -m "Add new feature").
Push to the branch (git push origin feature/new-feature).
Open a pull request.

Please follow the Code of Conduct and include tests for new features.
License
RustReaper is licensed under the MIT License.
Acknowledgments

Capstone for disassembly.
YARA for pattern matching.
Actix-web and React for GUI.

For issues or questions, open a ticket on GitHub.