MalChela Documentation

None

None

None

Table of contents

1. Welcome to MalChela

This site hosts the MalChela user guide, tool documentation, and integration instructions.

- Use the tabs above to navigate. On mobile or iPad, tap the ≡ icon (top-left) to open the navigation menu.
- The full PDF version of the user guide is available here.

2. About

MalChela is a modular toolkit for digital forensic analysts, malware researchers, and threat intelligence teams. It provides both a Command Line Interface (CLI) and a Graphical User Interface (GUI) for running analysis tools in a unified environment.

mal — malware

chela — “crab hand”

A chela on a crab is the scientific term for a claw or pincer. It’s a specialized appendage, typically found on the first pair of legs, used for grasping, defense, and manipulating things; just like these programs.

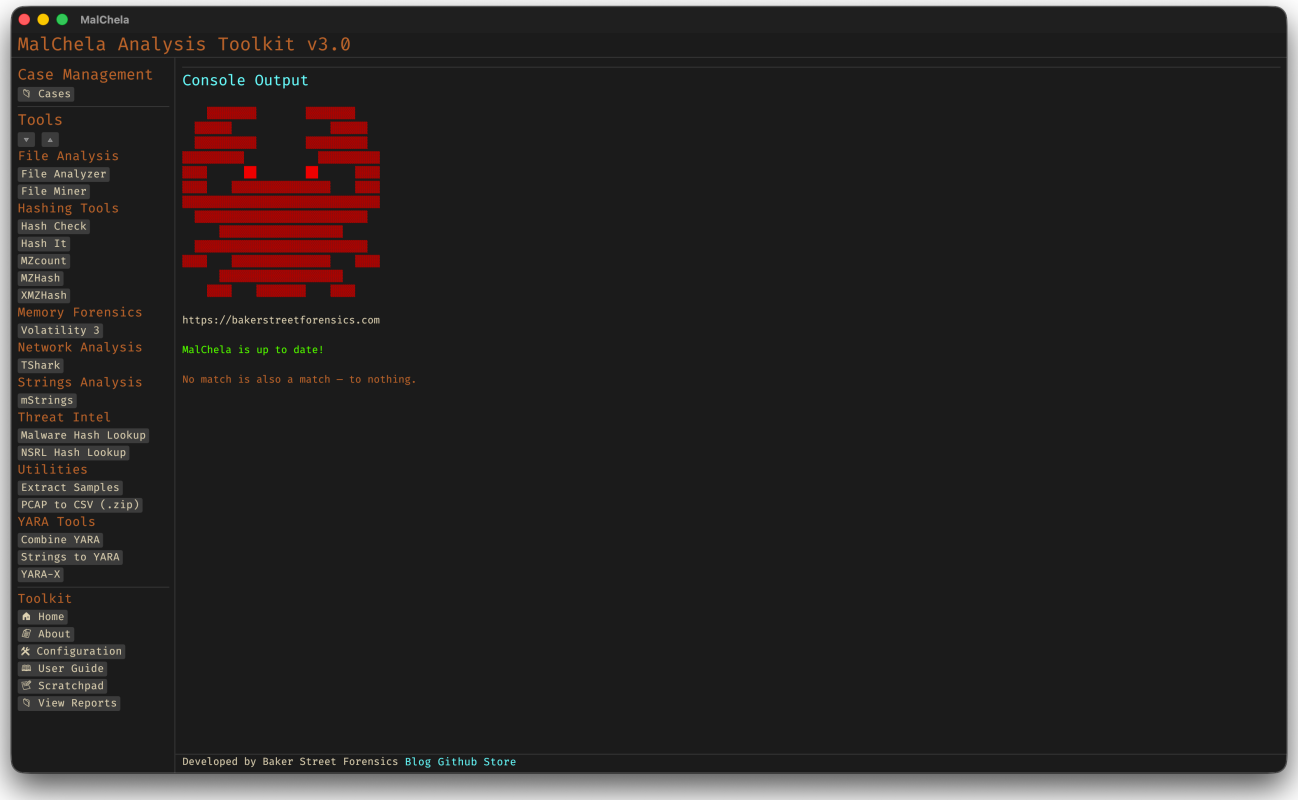

Figure 1.1: MalChela GUI

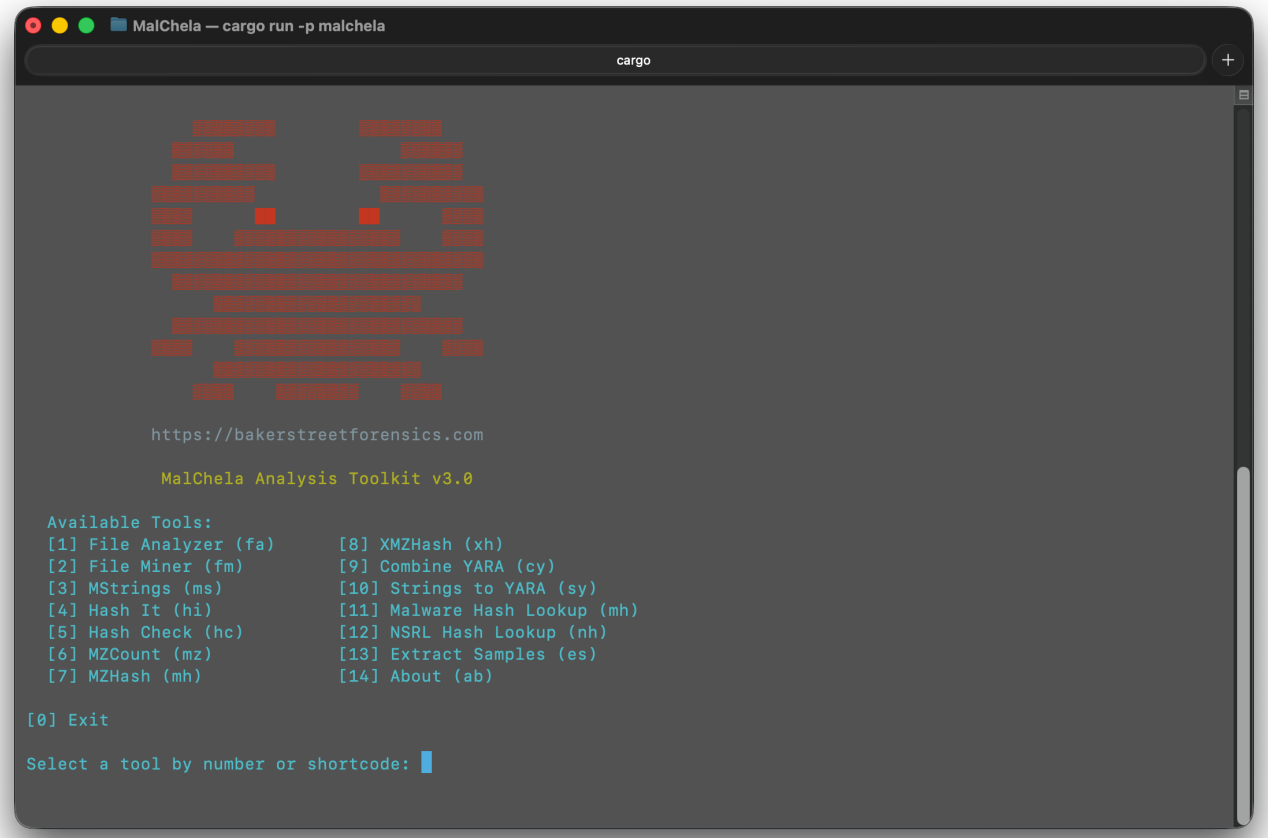

Figure 1.2: MalChela CLI

3. Installation

3.0.1 Prerequisites

- Rust and Cargo
- Git
- Unix-like environment (Linux, macOS, or Windows with WSL)

3.0.2 System Dependencies (Recommended)

To ensure all tools build and run correctly, install the following packages (especially for Linux/REMnux):

```
sudo apt install openssl libssl-dev clang yara libyara-dev pkg-config build-essential libglib2.0-dev libgtk-3-dev ssdeep
```

These are required for: - YARA and YARA-X support - Building Rust crates that link to native libraries (e.g., GUI dependencies) - TShark integration (via GTK/glib) - `ssdeep` is used for fuzzy hashing in tools like `fileanalyzer`. If not installed, fuzzy hash results may be unavailable.

3.0.3 Clone the Repository

```
git clone https://github.com/dwmetz/MalChela.git

cd MalChela
```

3.0.4 Build Tools

```
cargo build --release                  # Build all tools in release mode
cargo build -p fileanalyzer --release # Build an individual tool in release mode
```

3.0.5 One-Click Release Build (Recommended)

To build **all tools** in release mode in one step, use the script in the workspace root:

```
chmod +x release.sh
./release.sh
```

This will compile every core tool and generate optimized release binaries under `target/release/`. This is especially useful before first use of the GUI or case features, which rely on prebuilt binaries.

⚠️ Using `--release` is highly recommended to ensure optimal performance and avoid unexpected behavior when launching tools from the GUI.

3.0.6 Windows Notes

- Best experience via WSL2
- GUI is not supported natively on Windows

4. Configuration

4.1 tools.yaml

MalChela uses a central `tools.yaml` file to define which tools appear in the GUI, along with their launch method, input types, categories, and optional arguments. This YAML-driven approach allows full control without editing source code.

Key Fields in Each Tool Entry

Field	Purpose
name	Internal and display name of the tool
description	Shown in GUI for clarity
command	How the tool is launched (binary path or interpreter)
exec_type	One of `cargo`, `binary`, or `script`
input_type	One of `file`, `folder`, or `hash`
file_position	Controls argument ordering
optional_args	Additional CLI arguments passed to the tool
category	Grouping used in the GUI left panel

⚠️ All fields except `optional_args` are required.

4.1.1 Swapping Configs: REMnux Mode and Beyond

MalChela supports easy switching between tool configurations via the GUI.

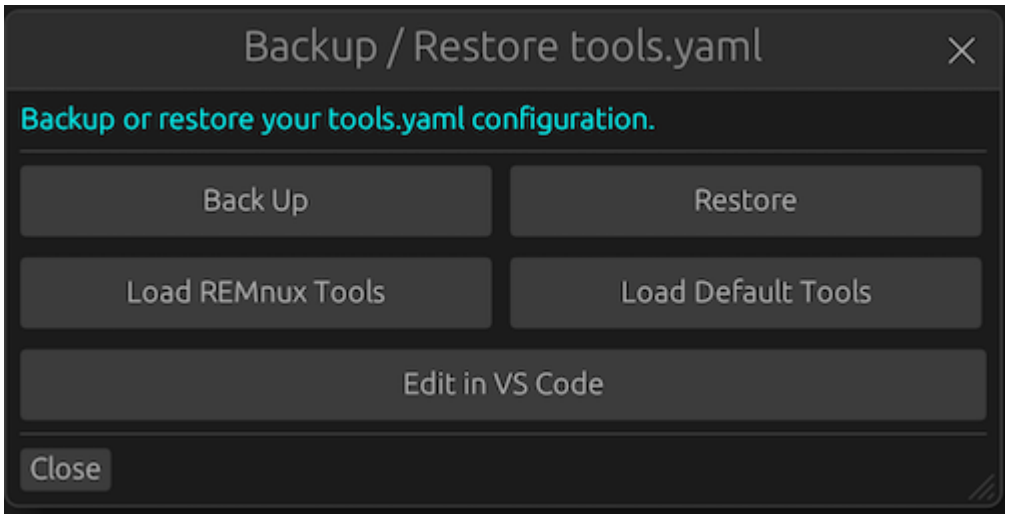

Figure 2.1: YAML Config Tool – Tool entry shown in table and form

To switch:

- Open the **Configuration Panel**
- Use **"Select tools.yaml"** to point to a different config
- Restart the GUI or reload tools

This allows forensic VMs like REMnux to use a tailored toolset while keeping your default config untouched.

A bundled `tools_remnux.yaml` is included in the repo for convenience.

KEY TIPS

- Always use `file_position: "last"` unless the tool expects input before the script
- For scripts requiring Python, keep the script path in `optional_args[0]`
- For tools installed via `pipx`, reference the binary path directly in `command`

4.1.2 Backing Up and Restoring tool.yaml

The MalChela GUI provides built-in functionality to back up and restore your `tools.yaml` configuration file.

Backup

To create a backup of your current `tools.yaml`:

- Open the **Configuration Panel**
- Click the **"Back Up Config"** button
- A timestamped copy of `tools.yaml` will be saved to the default location

You'll see a confirmation message when the operation completes successfully.

Restore

To restore from a previous backup:

- Click the **"Restore Config"** button in the Configuration Panel
- Select a previously saved backup file
- The selected file will overwrite the current configuration

This feature makes it easy to experiment with custom tool setups while retaining a safety net for recovery.

4.2 API Configuration

Some tools within MalChela rely on external services. In order to use these integrations, you must configure your API credentials.

4.2.1 Tools That Use API Keys

Tool	Service	Purpose
`malhash`	VirusTotal	Hash lookup and enrichment
`malhash`	MalwareBazaar	Hash lookup and sample classification
`fileanalyzer`	VirusTotal	Hash lookup

4.2.2 Where to Configure

MalChela uses two plain text files to store API keys for its third-party integrations:

```
vt-api.txt
mb-api.txt
```

These files should be placed in the **root of your MalChela workspace**, alongside `tools.yaml`. Each file should contain a single line with your API key.

These keys will be read at runtime by tools such as `malhash` to enable external lookups.

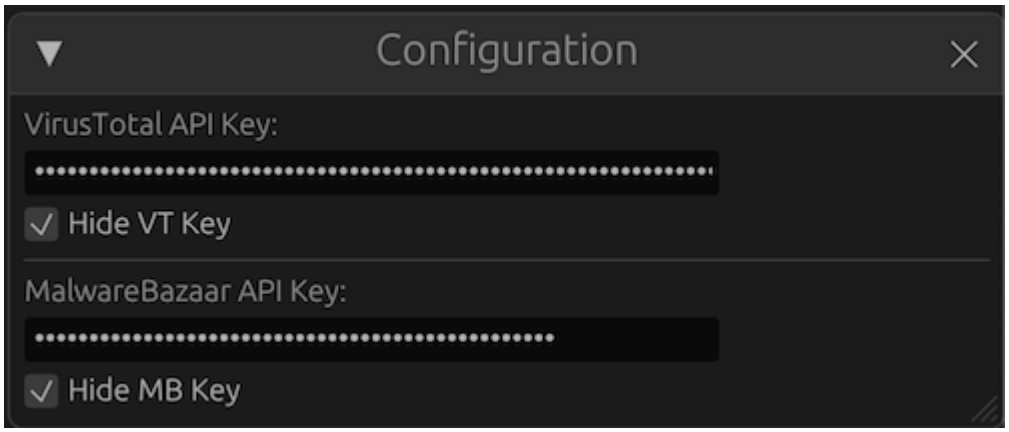

Figure 2.2: API Configuration Utility

4.2.3 Managing Your Keys with the Configuration Utility

The MalChela GUI includes a built-in Configuration Panel that lets you easily **Create or update API key files** without opening a text editor.

Look for the **API Key Management** section in the Configuration Panel. Changes take effect immediately and persist across sessions.

4.2.4 Best Practices

- **Keep these files private.** Do not commit them to Git or share them publicly.

If a tool requires an API key but none is found, it will log a warning and skip external requests.

5. 4.16 Case Management

The Cases feature in MalChela v3.0 introduces a structured way to manage analysis sessions, organize tool results, and preserve analyst notes for long-term reference. Each case acts as a container for a specific file or folder input, and captures relevant metadata, tool output, and custom annotations.

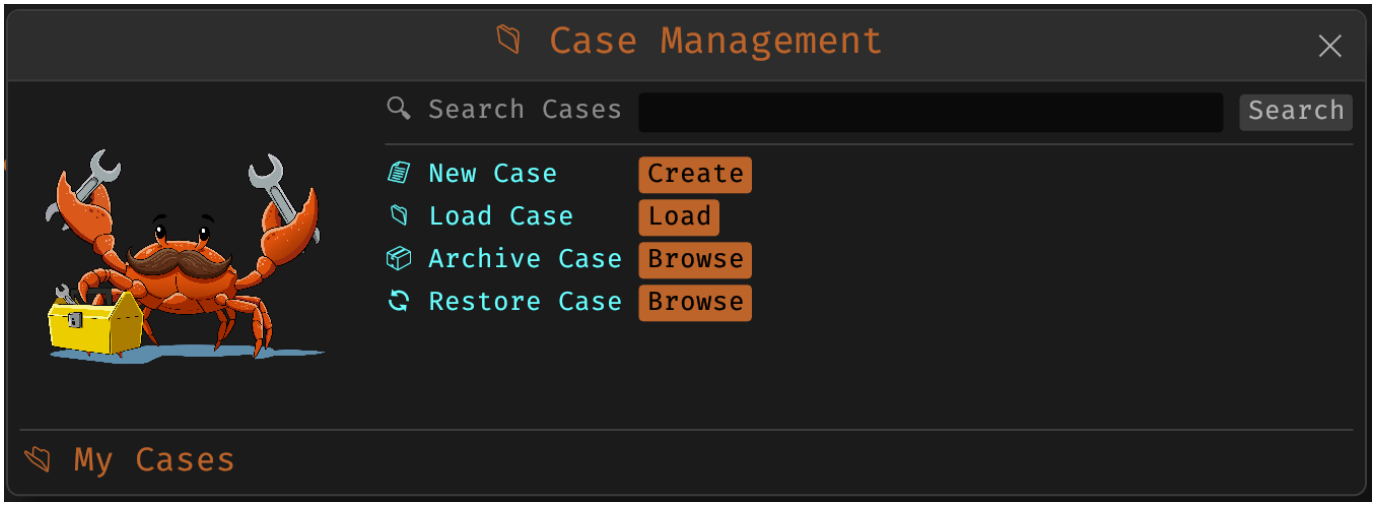

Figure 3.1: Case Management

5.1 Creating a Case

A case can be created by providing:

- A file (e.g. suspicious binary)
- A folder (directory of unknown files e.g. Tshark export http objects)

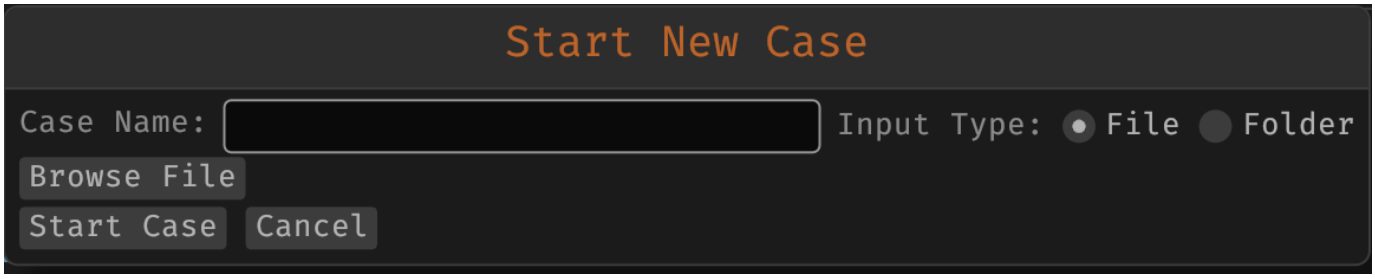

Figure 3.2: New Case

After choosing the input, assign a descriptive case name. The GUI will automatically create a new folder under:

```
saved_output/cases/<case_name>/
```

5.2 Case Contents

Each case folder includes:

- `case.yaml` : combines metadata, case tracking, and user notes in a single structured file
- `fileminer/` , `mstrings/` , `malhash/` , etc.: tool-specific output directories
- `tracking/` : flags for completed tool runs

5.3 Notes & Tagging

The scratchpad allows users to record notes, label items with tags, and track investigative context across sessions.

- Notes can include plain text, YAML fragments, or markdown.

5.3.1 Tagging:

- Any word prefixed with `#` (e.g., `#malware`, `#tshark`) becomes a tag.
- These tags will appear in the Workspace panel under the current case.
- Tags help organize and filter case context.

5.3.2 Search:

- From the Case Modal, you can search across:
- All saved tool outputs
- Notes contents
- Tags (by prefixing with `#`)

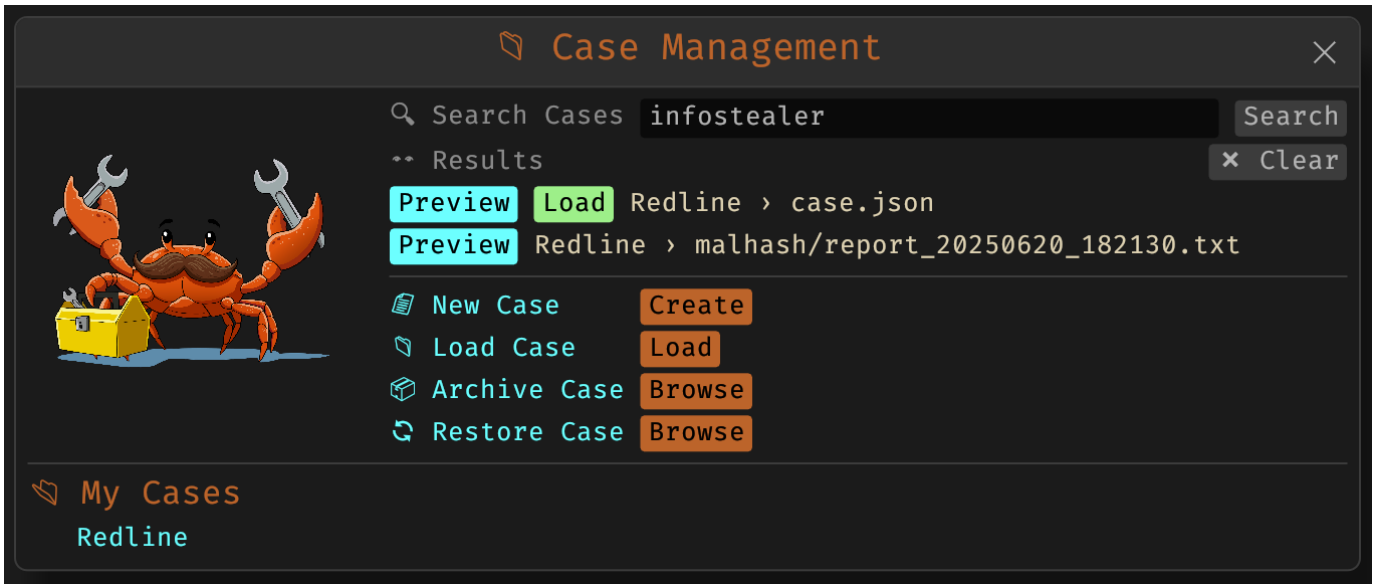

Figure 3.3: Searching Cases

5.4 Archiving & Restoring Cases

Cases can be archived into a `.zip` file using the GUI's **Archive Case** feature. This creates a portable snapshot that includes all metadata, notes, and tool outputs.

To restore a case:

- Open the GUI
- Use **Restore Case**
- Select a `.zip` archive containing a valid `case.json`

The case will be rehydrated into the `saved_output/cases/` directory and appear in the workspace.

5.5 Case-Aware Tool Tracking

Tools launched from the workspace automatically write output into the active case's subfolder and register a tracking file to avoid redundant execution.

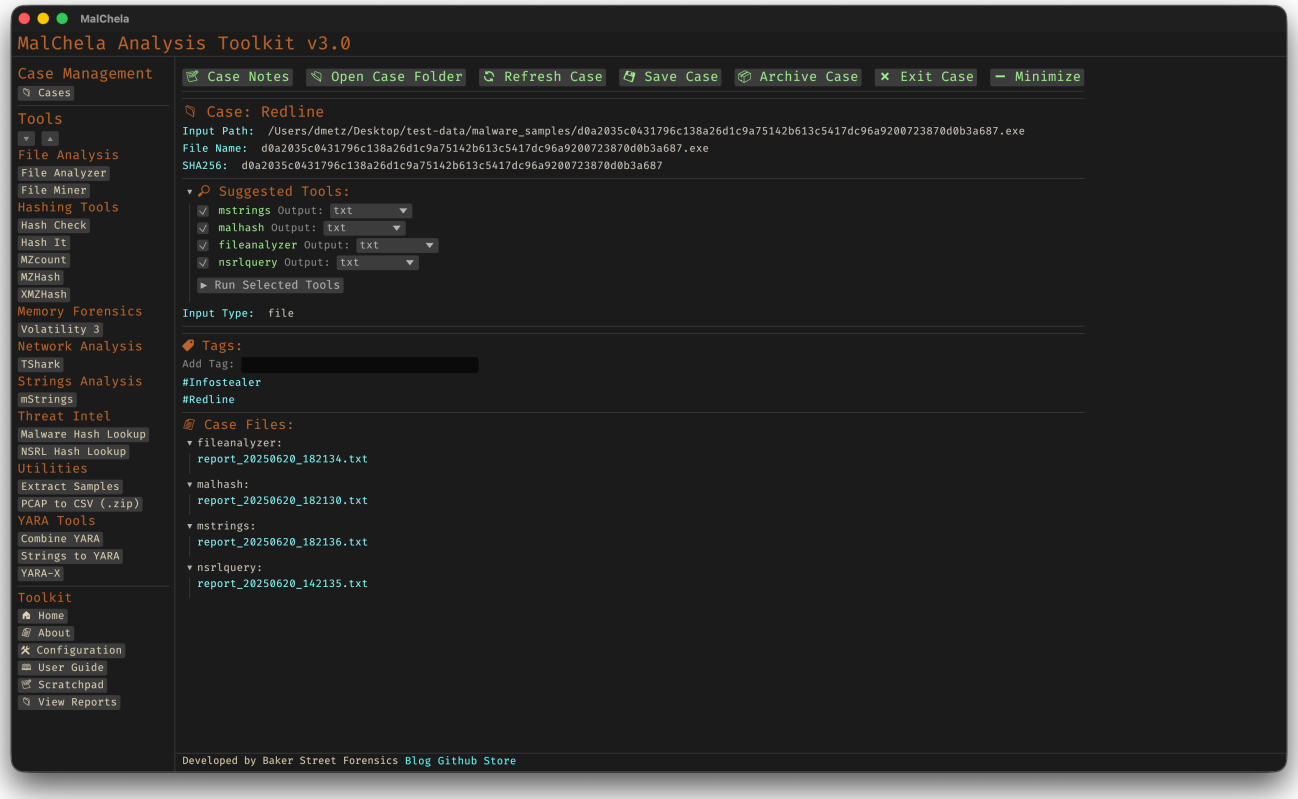

Figure 3.4: Example Case

5.6 Summary

Cases enable consistent, organized forensic workflows across sessions and machines. Whether you're triaging suspicious files, running multi-tool pipelines, or collaborating across systems, the Cases feature ensures nothing is lost.

6. Core Tools

6.1 Overview

Figure 4.1: MalChela GUI

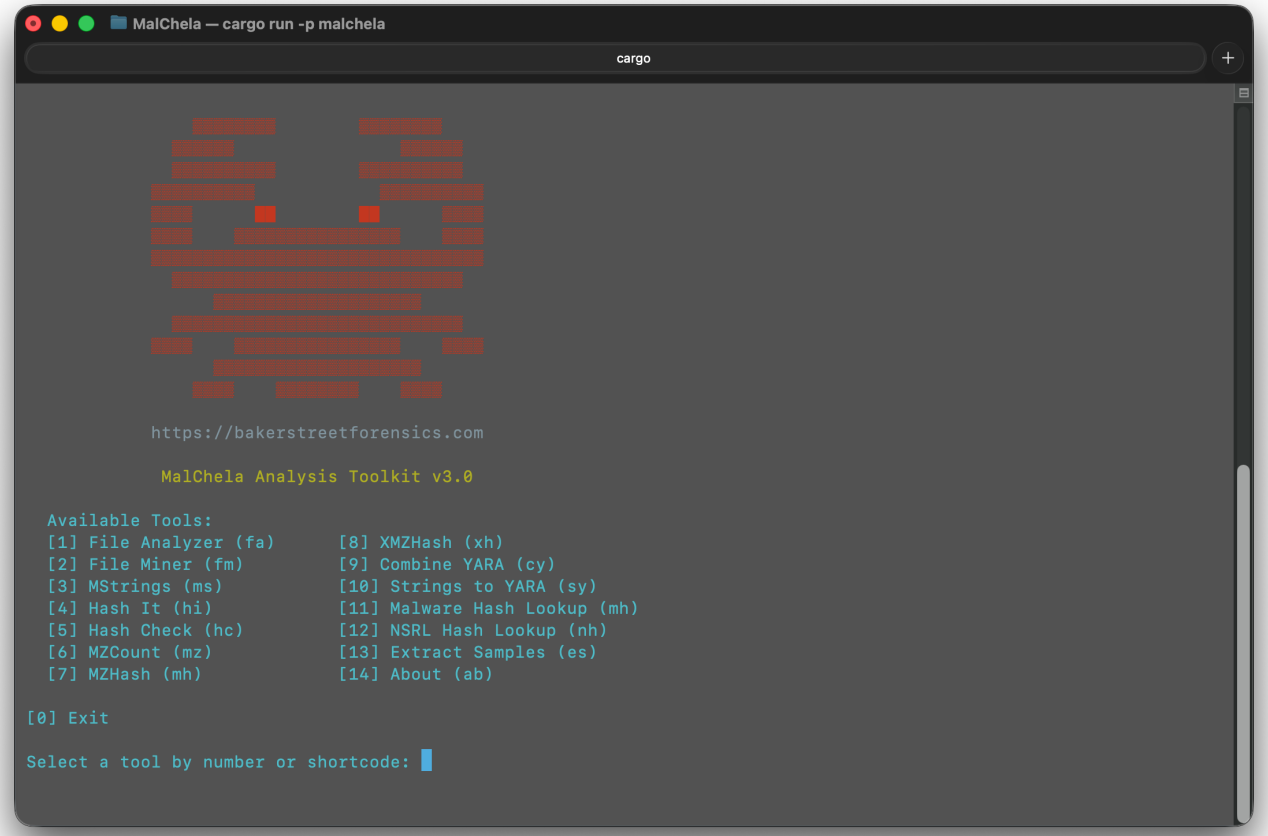

Figure 4.2: MalChela CLI

6.1.1 MalChela Core Tools

These built-in programs provide fast, flexible functionality for forensics and malware triage.

Program	Function
Combine YARA	Point it at a directory of YARA files and it will output one combined rule
Extract Samples	Point it at a directory of password protected malware files to extract all
File Analyzer	Get the hash, entropy, packing, PE info, YARA and VT match status for a file
File Miner	Scans a folder for file type mismatches and metadata, and provides suggested tools
Hash Check	Check a hash against a .txt or .tsv lookup table
Hash It	Point it to a file and get the MD5, SHA1 and SHA256 hash
mStrings	Analyzes files with Sigma rules (YAML), extracts strings, matches ReGex
MZHash	Recurse a directory, for files with MZ header, create hash list and lookup table
MZcount	Recurse a directory, uses YARA to count MZ, Zip, PDF, other
NSRL Query	Query a MD5 or SHA1 hash against NSRL
Strings to YARA	Prompts for metadata and strings (text file) to create a YARA rule
Malware Hash Lookup	Query a hash value against VirusTotal & Malware Bazaar*
XMZHash	Recurse a directory, for files without MZ, Zip or PDF header, create hash list and lookup table

***The Malware Hash Lookup requires an API key for VirusTotal and Malware Bazaar.** If unidentified, MalChela will prompt you to create them the first time you run the malware lookup function.

6.2 Usage

6.2.1 Getting Started

Before runnning MalChela for the first time, you need to build the release binaries. There is a script provided in the workspace root.

Building the Releases

```
chmod +x release.sh
./release.sh
```

6.2.2 Execution

MalChela supports three main workflows:

Direct Tool Execution (CLI)

```
./target/release/toolname [input] [flags]
```

MalChela CLI Launcher Menu

```
./target/release/malchela
```

MalChela GUI Launcher

```
./target/release/MalChelaGUI
```

6.2.3 CLI Usage Notes

- Tools that accept paths (files or folders) should be run from the `target/release` directory after building with `release.sh`: `bash ./target/release/fileanalyzer /path/to/file -o`

Most tools now support a `--case <name>` argument to redirect saved output to a specific case folder under `saved_output/cases/`. Cases must be initiated with either a file or folder as the input. Hash-only workflows can be added to an existing case but cannot start one.

Note: Some tools (e.g., `mstrings`, `fileanalyzer`, `malhash`) require the `-o` flag to trigger output saving—even when `--case` is specified. Others (like `strings_to_yara` or `mzcount`) save automatically when a case is provided. Refer to the Tool Behavior Reference below for details.

Output Formats

All tools that support saving reports use the following scheme: `saved_output/<tool>/report_<timestamp>.<ext>`

To save output, use:

```
-o -t   # text
-o -j   # json
-o -m   # markdown
```

- `-o` enables saving (CLI output is not saved by default)

If a `--case` argument is supplied, the report will be saved to: `saved_output/cases/<case_name>/<tool>/report_<timestamp>.<ext>`

Example:

```
cargo run -p mstrings — path/to/file — -o -j
```

- If `-o` is used without a format (`-t`, `-j`, or `-m`), an error will be shown

6.2.4 GUI Usage Notes

GUI Features Summary

- Categorized tool list with input type detection (file, folder, hash)
- Arguments textbox and dynamic path browser
- Console output with ANSI coloring
- Save Report checkbox toggles `-o` flag
- Status bar displays CLI-equivalent command
- Alphabetical sorting of tools within categories
- Tool descriptions are now shown alongside tool names
- Saved reports are cleaned of internal formatting tags like [green], [reset], etc.
- Cases must be created from a file or folder. Hashes can be used later but do not initiate new cases.

GUI Walkthrough

Layout

- Top Bar: Title and status
- Left Panel: Tool categories and selections
- Center Panel: Dynamic tool input options
- Bottom Panel: Console output

Running Tools

- Select a tool
- Fill in input fields
- Configure options (save report, format, etc.)
- Click Run

Save Report

- Formats:
- .txt Analyst-readable summary
- .json Machine-parsable, structured output
- .md Shareable in tickets, wikis, etc. .txt, .json, .md
- Location: saved_output//report_. (only one file is generated per run)

Scratchpad

- An integrated notepad for recording strings, indicators or notes
- Supports saving as text, markdown and YAML formats
- Integrated “Open in VS Code” button for saved notes
- Any line starting with `hash:` is ignored when using the Scratchpad as a source for String_to_Yara to generate YARA rules

6.2.5 Tool Behavior Reference

Tool	Input Type	Supports `-o`	Prompts if Missing	Notes
combine_yara	folder	❌	✅	Combines multiple YARA rules
extract_samples	file	❌	✅	Extracts archive contents
fileanalyzer	file	✅	✅	Uses YARA + heuristics
hashit	file	✅	✅	Generates hashes
hashcheck	hash and lookup file	❌	✅	Checks files against known hashes
malhash	hash	✅	✅	Uses vt-cli + bazaar-cli
fileminer	folder	✅	✅	Identifies mismatches
mstrings	file	✅	✅	Maps strings to MITRE
mzhash	folder	✅	✅	Hashes files with MZ header
nsrlquery	file	✅	✅	Queries CIRCL
strings_to_yara	text file and metadata	Case Only	✅	Saves to case folder if `--case` is provided
mzcount	folder	❌	✅	Will save to case folder if `--case` is provided
xmzhash	folder	✅	✅	Hashes files without known headers

6.3 CombineYARA

Combine YARA merges multiple YARA rule files into a single consolidated rule set. It recursively scans a folder for .yar or .yara files and combines them into one output file. Ideal for organizing or deploying rule collections.

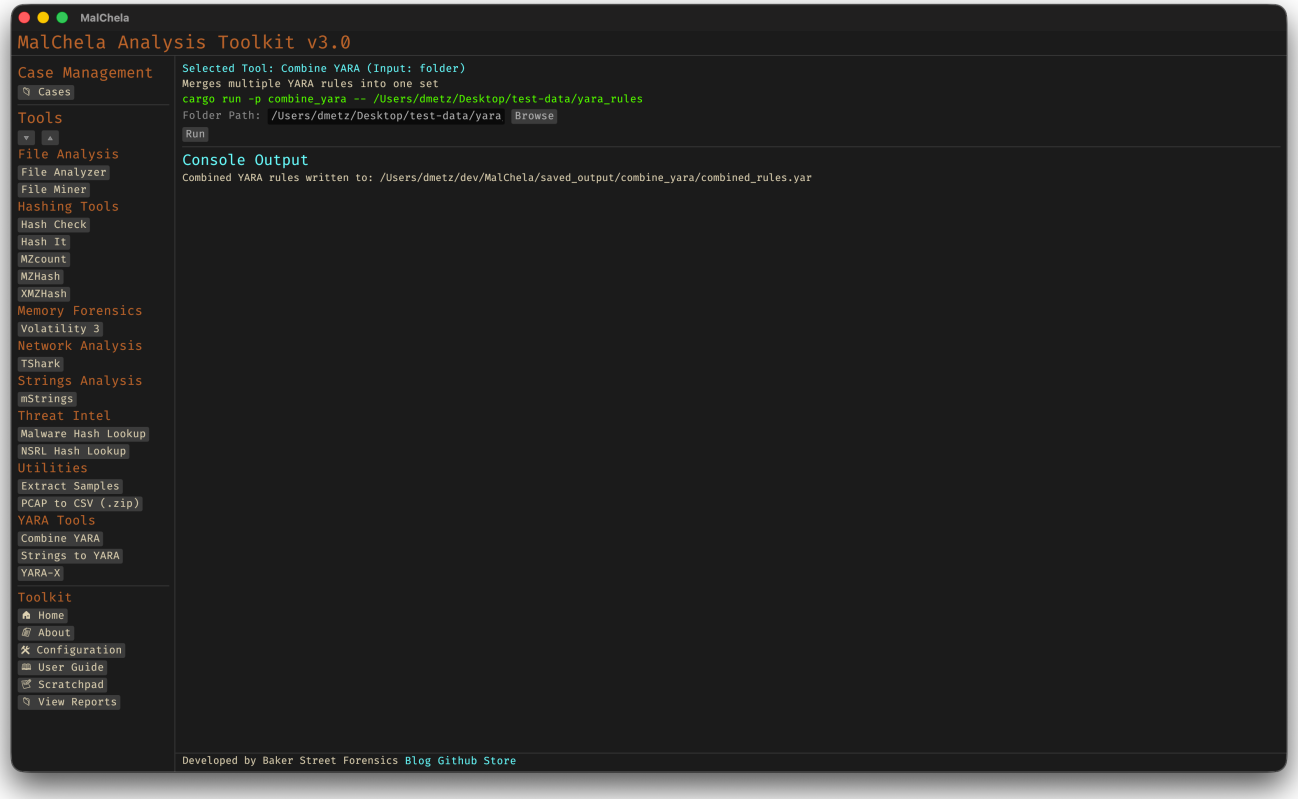

Figure 11: Combine YARA

🔧 CLI Syntax

```
cargo run -p combine_yara /path_to_yara_rules/
```

If no path is provided, the tool will prompt you to enter the directory interactively.

```
Enter the directory path to scan for YARA rules:
```

6.4 ExtractSamples

Extract Samples recursively unpacks password-protected archives commonly used in malware sharing (e.g., .zip, .rar, .7z). It uses default malware research passwords like infected and malware to extract samples in bulk for analysis.

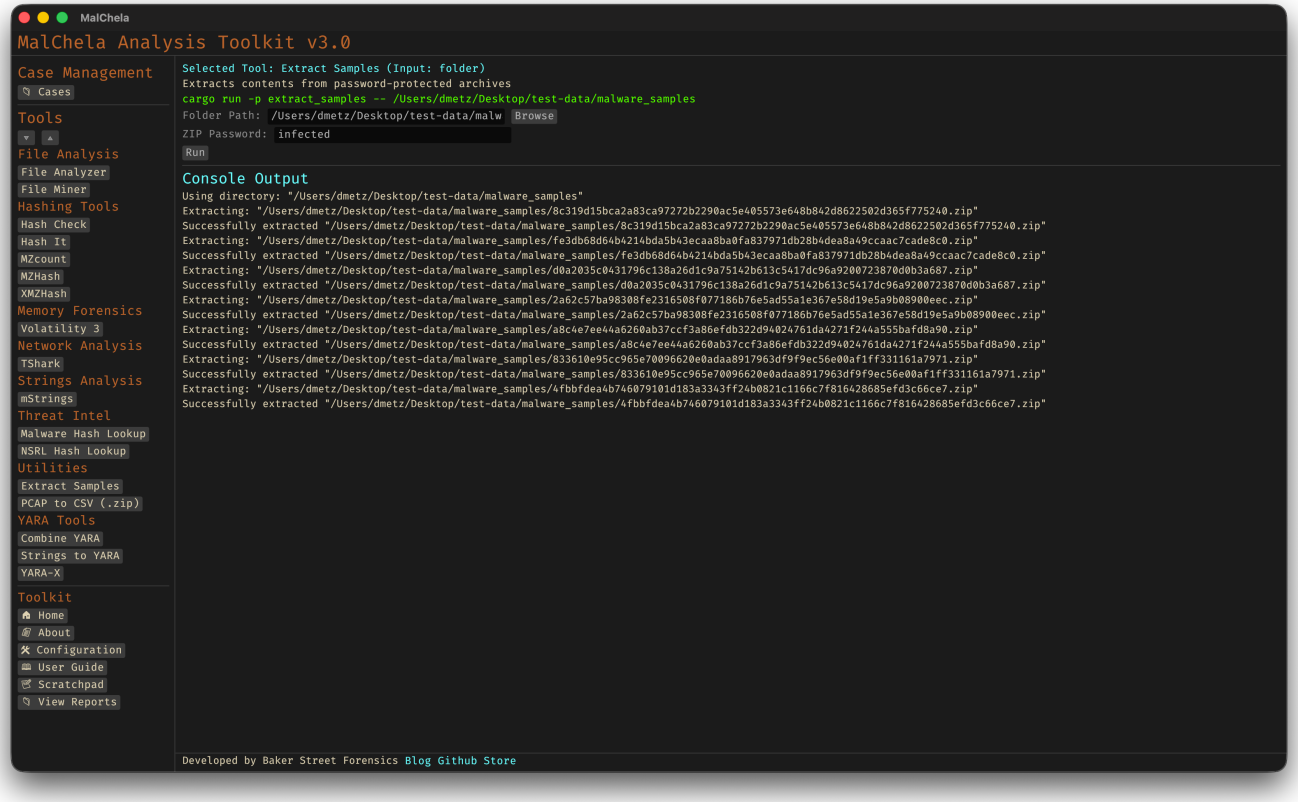

Figure 12: Extract Samples

🔧 CLI Syntax

```
# Example 1: No case name
cargo run -p extract_samples /path_to_directory/ infected
```

In this mode, extracted files will be placed in the same location as each archive found.

```
# Example 2: With case name
cargo run -p extract_samples /path_to_directory/ infected --case Case123
```

When `--case` is provided, all extracted files will be saved under:

```
/saved_output/cases/Case123/extract_samples/
```

If no path or password is provided, the tool will prompt for them interactively.

6.5 FileAnalyzer

FileAnalyzer performs deep static analysis on a single file. It extracts hashes, entropy, file type metadata, YARA rule matches, NSRL validation, and — for PE files — rich header details including import/export tables, compile timestamp, and section flags. Ideal for triaging unknown executables or confirming known file traits.

Figure 13: File Analyzer

- YARA rules for `fileanalyzer` are stored in the `yara_rules` folder in the workspace. You can modify or add rules here.

🔧 CLI Syntax

```
# Example 1: No case name
cargo run -p fileanalyzer -- /path_to_file/ -o -t

# Example 2: With case name
cargo run -p fileanalyzer -- /path_to_file/ -o -t --case Case123
```

When `--case` is provided, output will be saved under:

```
/saved_output/cases/Case123/fileanalyzer/
```

If `--case` is not specified, files will be saved in the default `saved_output/fileanalyzer/` folder.

6.6 FileMiner

Note: `FileMiner` replaces the deprecated `MismatchMiner`.

FileMiner is a command-line tool that recursively scans a directory to analyze files by magic bytes and hash, identifying mismatches between file extensions and true types. It is useful for forensic triage, anomaly detection, and preparing follow-up analysis using other tools in the MalChela suite.

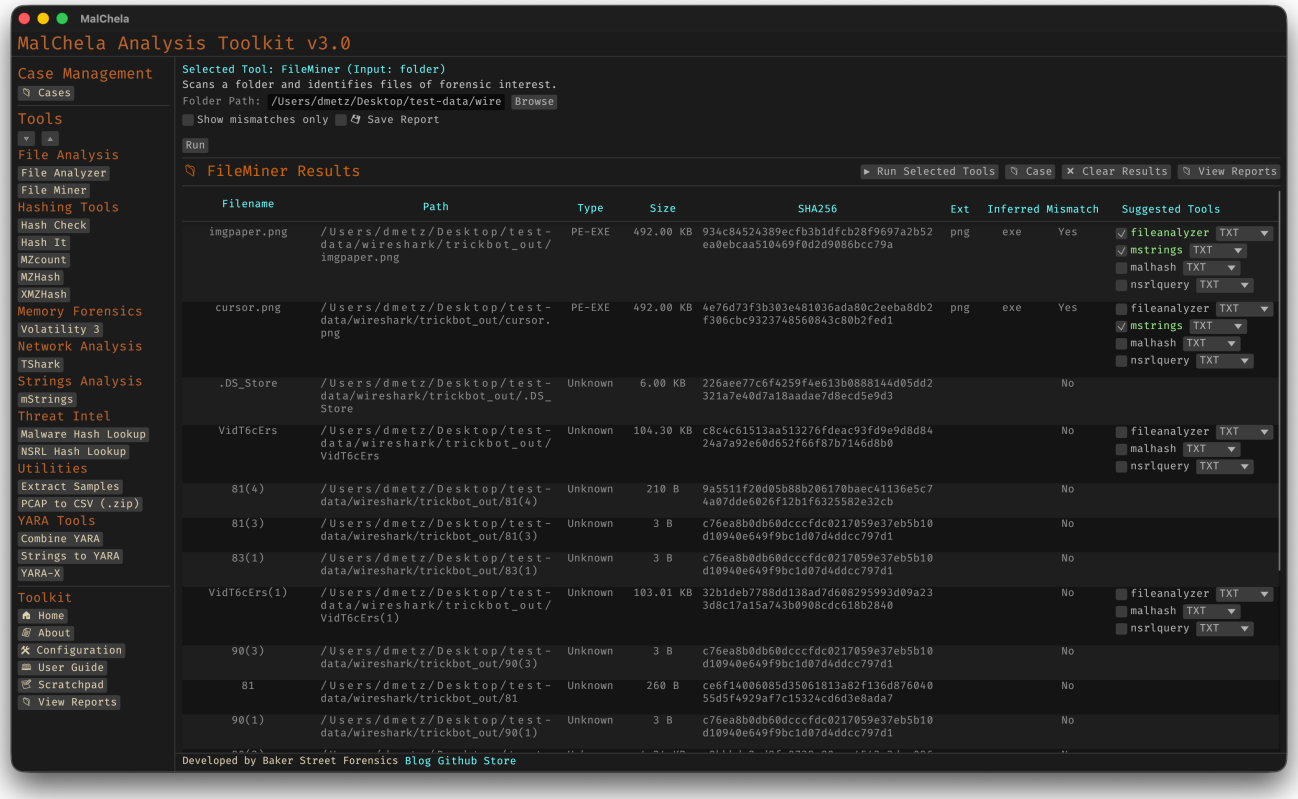

Figure 14: File Miner

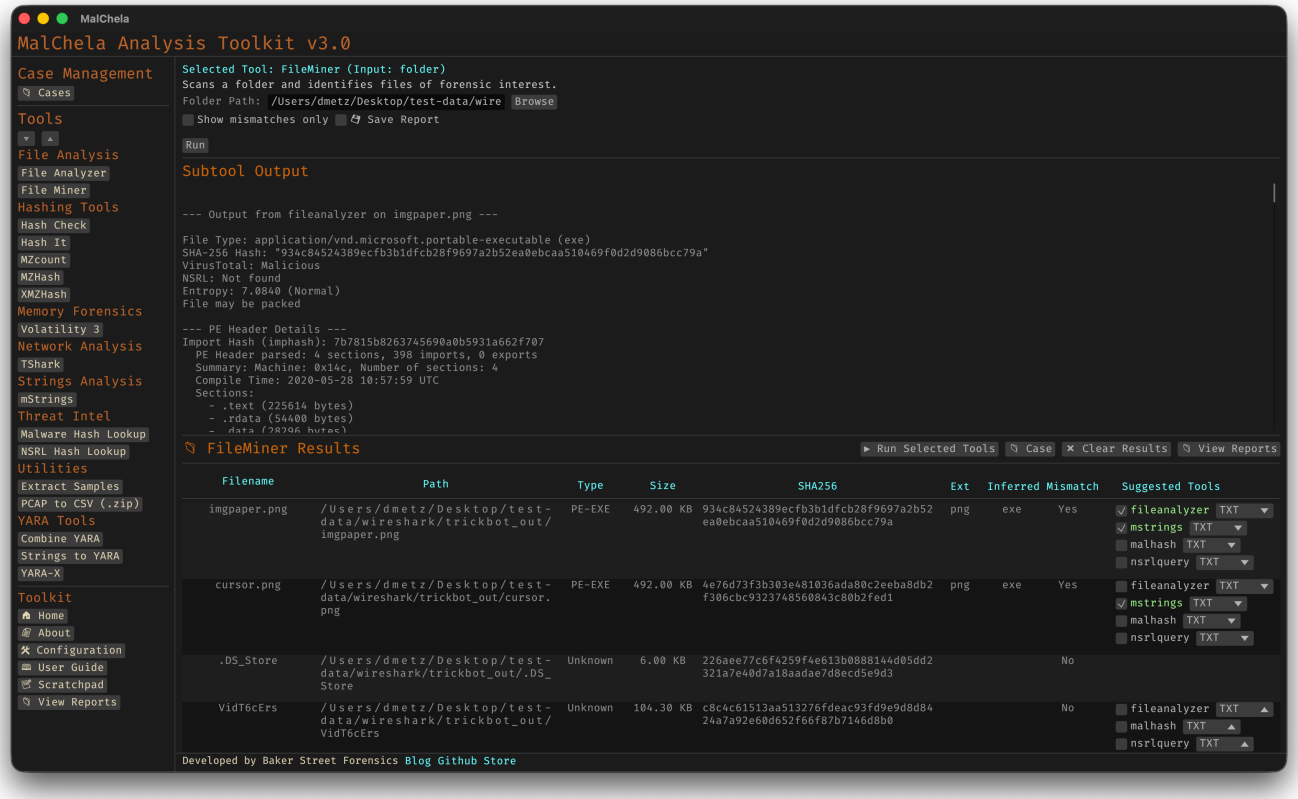

Figure 15: File Miner with Subtool Output

6.6.1 Function Overview

- Identifies file types using magic byte detection (`infer`)
- Computes SHA-256 hashes for all files
- Detects extension mismatches
- Suggests relevant analysis tools (e.g., FileAnalyzer, mStrings, malhash)
- Outputs results in a styled table or optional JSON format
- Integrates with case management via the `--case` flag
- Automatically launches in GUI when a folder-based case is created or restored
- Results populate an interactive table in the GUI
- Users can launch suggested tools on a per-file basis directly from the GUI

6.6.2 CLI Usage

```
cargo run -p fileminer -- [OPTIONS] [DIR]
```

Options

Option	Description
`DIR`	Directory to analyze. Optional — will prompt if not supplied.
`--json`	Save results to JSON. Defaults to `fileminer_output.json` unless `--output` is used.
`--output <filename>`	Overrides the default output file name. Used internally by the GUI.
`--case <case-name>`	Saves output under `saved_output/<case-name>/fileminer/`. Also passes case name to downstream tools.
`-m`, `--mismatches-only`	Only display entries with extension mismatches.

Examples

```
# Analyze interactively
cargo run -p fileminer --

# Analyze directory and save JSON
cargo run -p fileminer -- /path/to/files --json

# Save to specific case folder
cargo run -p fileminer -- /path/to/files --case case123

# Filter mismatches only
cargo run -p fileminer -- /path/to/files -m

# Combine all
cargo run -p fileminer -- /path/to/files --case suspicious_usb -m
```

6.6.3 GUI Usage Notes

- When a new case is created or restored using a folder, FileMiner runs automatically in the GUI.
- Results are saved under `saved_output/cases/<case-name>/fileminer/`.
- FileMiner displays an interactive table of results with suggested tools per file.
- Suggested tools can be launched directly from within the GUI results panel.

6.7 HashCheck

HashCheck lets you quickly verify whether a given set of files (or hash values) match any entry in one or more known-good or known-bad hash lists. It's designed to help analysts triage large collections of files by comparing against reference datasets — for example, malware repositories, NSRL exports, or your own curated lists.

Hash lists should be in `.tsv` format (tab-separated values) for best compatibility, though `.txt` files are also accepted.

Figure 16: Hash Check

You can generate .tsv lookup files using MZHash or XMZHash.

HashCheck supports **MD5**, **SHA1**, and **SHA256** formats.

🔧 CLI Syntax

```
# Example 1: Basic usage
cargo run -p hashcheck ./hashes.tsv 44d88612fea8a8f36de82e1278abb02f

# Example 2: Save output as .txt
cargo run -p hashcheck ./hashes.tsv 44d88612fea8a8f36de82e1278abb02f -- -o -t

# Example 3: Save output to case folder
cargo run -p hashcheck ./hashes.tsv 44d88612fea8a8f36de82e1278abb02f -- -o -t --case CaseName
```

HashCheck accepts a hash and a lookup file (TSV or TXT). If not provided, you'll be prompted interactively.

When `--case` is used, output will be saved under:

```
saved_output/cases/CaseName/hashcheck/
```

Without `--case`, reports are saved to the default:

```
saved_output/hashcheck/
```

6.8 HashIt

`hashit` is a flexible hashing utility that supports calculating multiple hash types for one or more files. It can be used to verify file integrity, generate forensic reports, or build hash sets for analysis. Output can be saved in text, JSON, or Markdown format, and optionally redirected into a case folder.

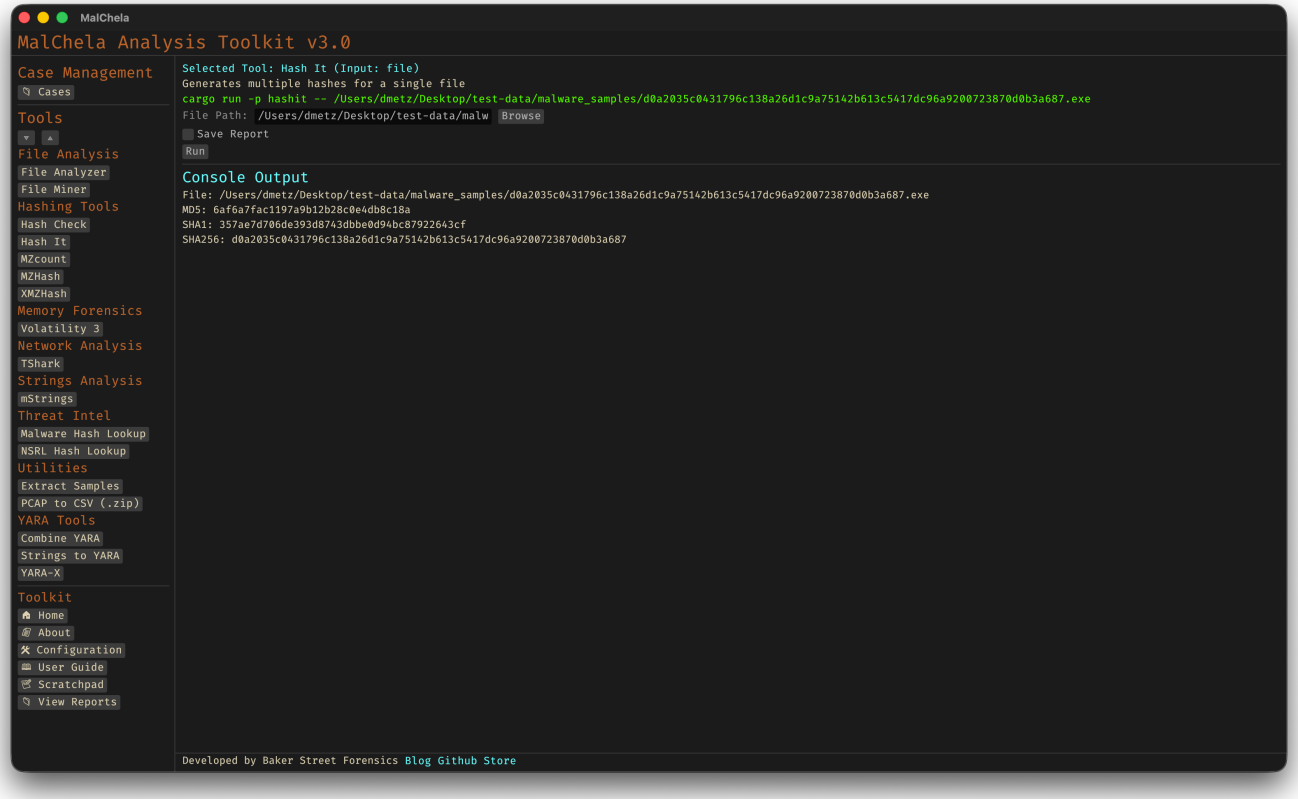

Figure 17: Hash It

The tool supports batch operations on directories and automatically recurses through subfolders. It's especially useful during triage to generate hash inventories or validate file integrity against known datasets.

🔧 CLI Syntax

```
# Example 1: Show hash values only
cargo run -p hashit -- /path_to_file/

# Example 2: Save as .txt
cargo run -p hashit -- /path_to_file/ -o -t

# Example 3: Save to case folder
cargo run -p hashit -- /path_to_file/ -o -t --case CaseName
```

Use `-o` to save output and include one of the following format flags: - `-t` → Save as `.txt` - `-j` → Save as `.json` - `-m` → Save as `.md`

If no file is provided, the tool will prompt you to enter the path interactively.

When `--case` is used, output will be saved under:

```
saved_output/cases/CaseName/hashit/
```

Otherwise, reports are saved to:

```
saved_output/hashit/
```

6.9 MalHash

MalHash queries malware intelligence sources using a provided hash. It checks VirusTotal and MalwareBazaar for file metadata, threat labels, antivirus detections, and known associations. A quick way to enrich an unknown sample or confirm if a hash is already known and classified in the wild.

Figure 18: Malware Hash Lookup

The first time you run MalHash, you'll be prompted to configure API keys for VirusTotal and MalwareBazaar if they're not already set.

🔧 CLI Syntax

```
# Example 1: Lookup only
cargo run -p malhash -- d41d8cd98f00b204e9800998ecf8427e

# Example 2: Save output as .txt
cargo run -p malhash -- d41d8cd98f00b204e9800998ecf8427e -o -t

# Example 3: Save output to a case folder
cargo run -p malhash -- d41d8cd98f00b204e9800998ecf8427e -o -t --case Case123
```

Use `-o` to save output and include one of the following format flags: - `-t` → Save as `.txt` - `-j` → Save as `.json` - `-m` → Save as `.md`

If no hash is provided, the tool will prompt you to enter it interactively.

When `--case` is used, output is saved to:

```
saved_output/cases/Case123/malhash/
```

Otherwise, reports are saved to:

```
saved_output/malhash/
```

6.10 MStrings

mStrings extracts strings from files and classifies them using regular expressions, YARA rules, and MITRE ATT&CK mappings. It highlights potential indicators of compromise and suspicious behavior, grouping matches by tactic and technique. Ideal for quickly surfacing malicious capabilities in binaries, scripts, and documents.

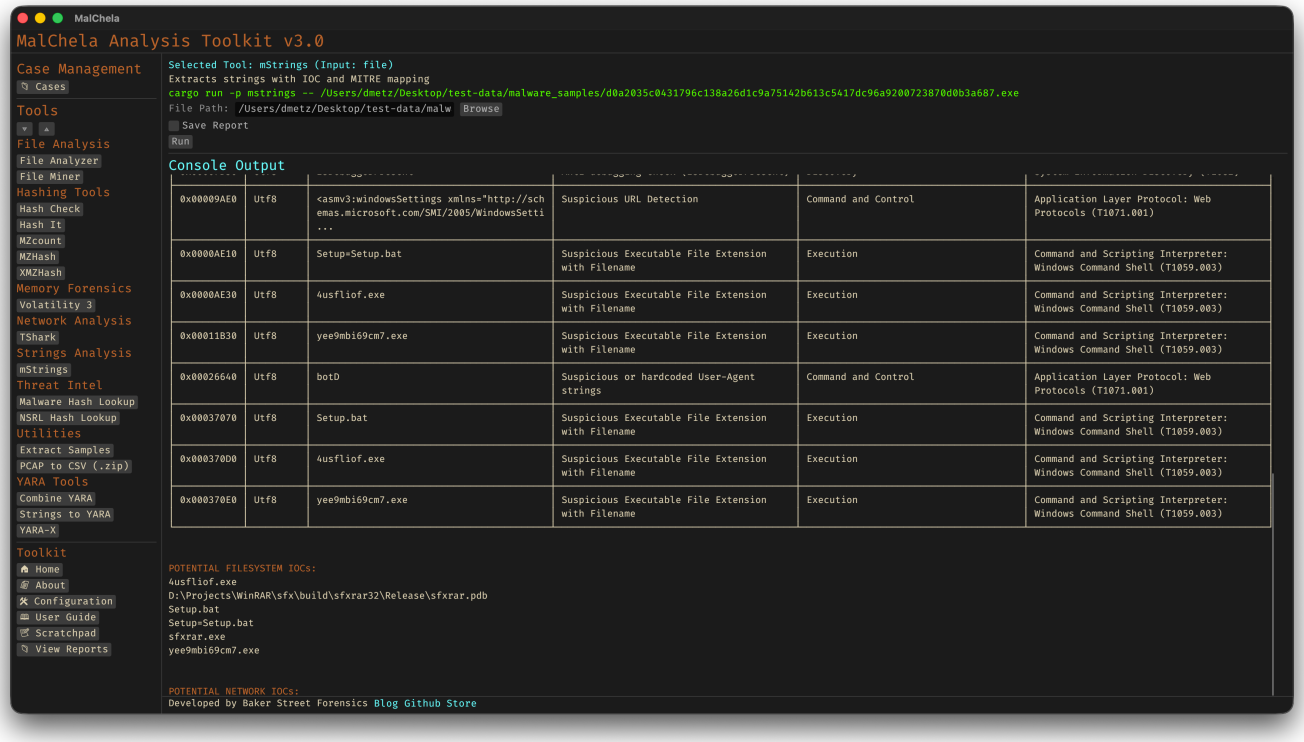

Figure 19: MStrings

🔧 CLI Syntax

```
# Example 1: Scan a file
cargo run -p mstrings -- /path_to_file/

# Example 2: Save output as .txt
cargo run -p mstrings -- /path_to_file/ -o -t

# Example 3: Save output to a case folder
cargo run -p mstrings -- /path_to_file/ -o -t --case CaseXYZ
```

Use `-o` to save output and include one of the following format flags: - `-t` → Save as `.txt` - `-j` → Save as `.json` - `-m` → Save as `.md`

If no file is provided, the tool will prompt you to enter the path interactively.

When `--case` is used, output is saved to:

```
saved_output/cases/CaseXYZ/mstrings/
```

Otherwise, results are saved to:

```
saved_output/mstrings/
```

6.11 MZCount

MZcount recursively scans a directory and counts the number of files that match key signatures like MZ (Windows executables), ZIP, PDF, and others. It uses lightweight YARA rules to classify files by type, giving a quick overview of the content breakdown within a dataset. Results can be displayed in either a detailed per-file view or a clean summary table, depending on your analysis needs.

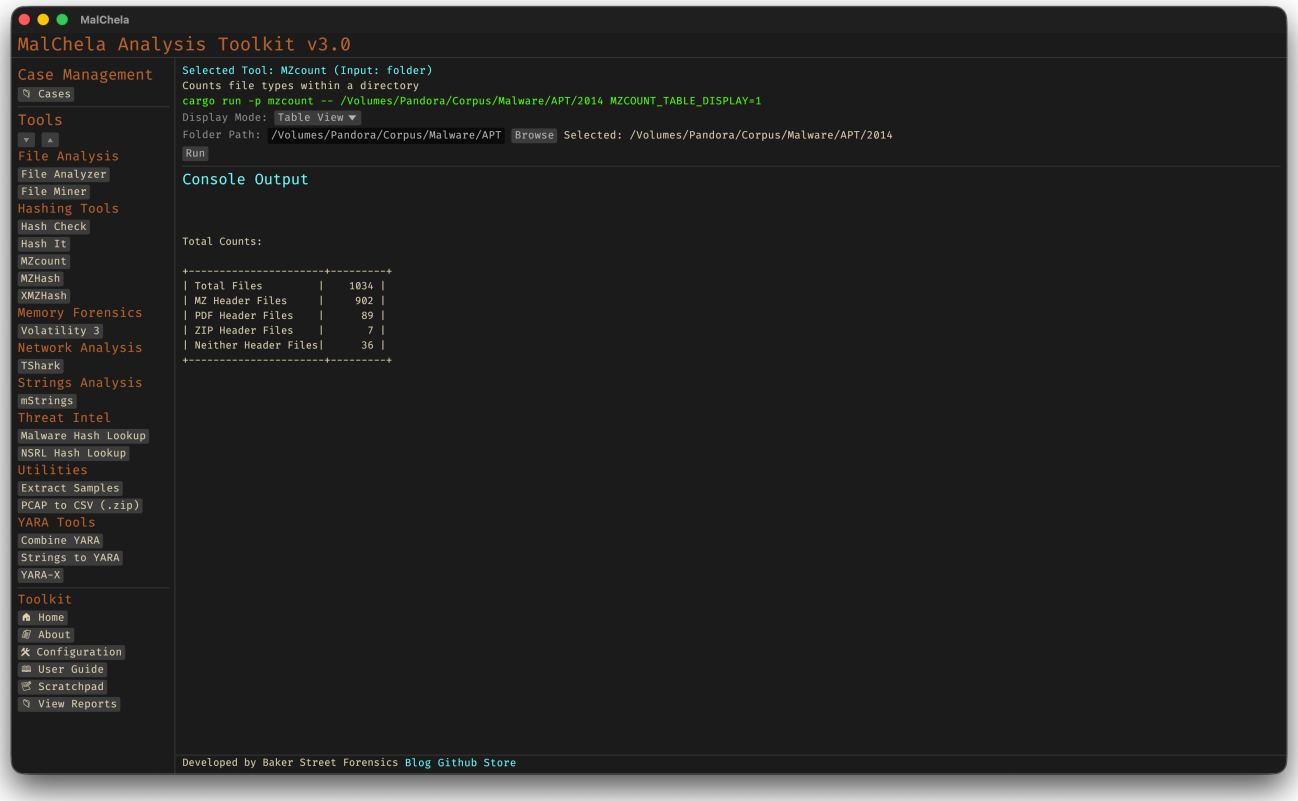

Figure 20: MZCount Table View

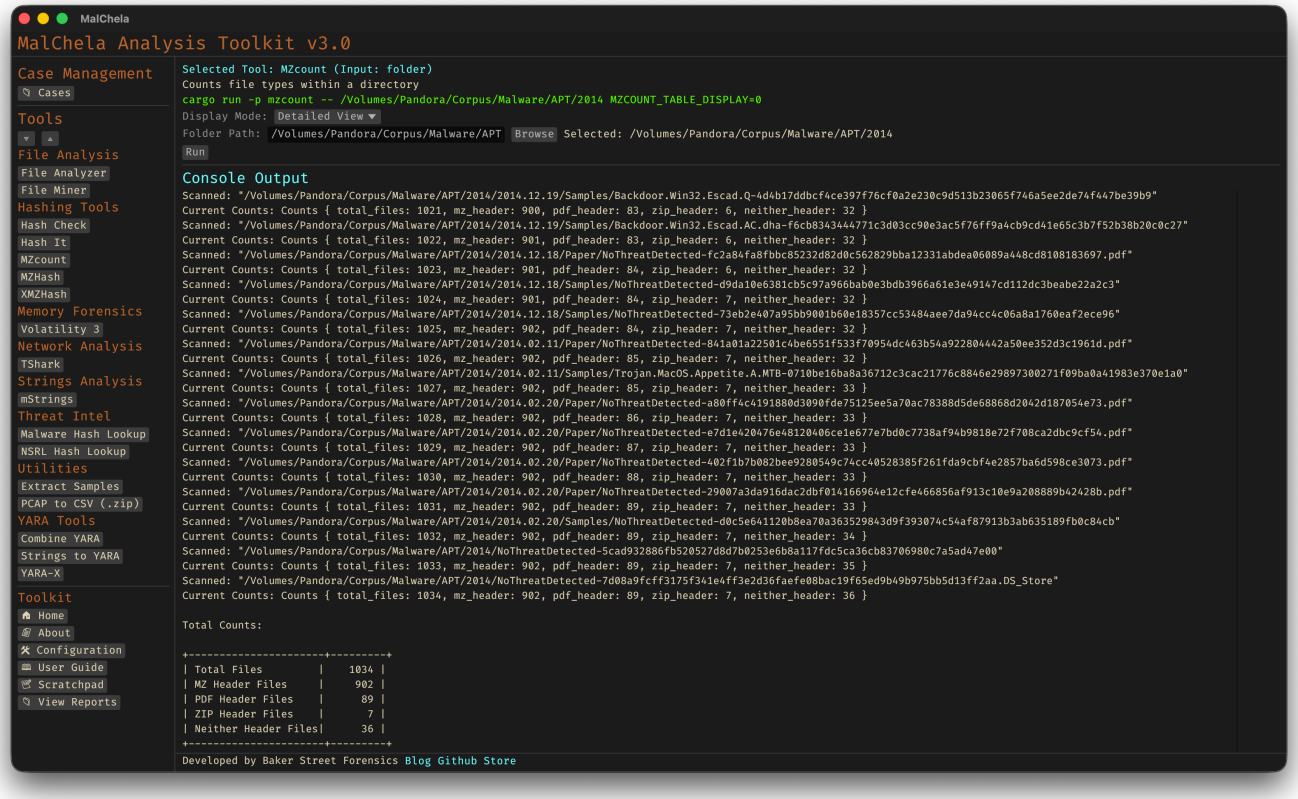

Figure 21: MZCount Detail View

🔧 CLI Syntax

```
# Example 1: Scan a directory and view results in terminal
cargo run -p mzcount -- /path_to_scan/

# Example 2: Enable table mode
MZCOUNT_TABLE_DISPLAY=1 cargo run -p mzcount -- /path_to_scan/

# Example 3: Save results as .txt
cargo run -p mzcount -- /path_to_scan/ -- -o -t

# Example 4: Save results to a case folder
cargo run -p mzcount -- /path_to_scan/ -- -o -t --case CaseName
```

If no path is provided, the tool will prompt you to enter it interactively.

Use `-o` to save output and `-t` to specify plain text format.

When `--case` is used, output is saved under:

```
saved_output/cases/CaseName/mzcount/
```

Otherwise, output is saved under:

```
saved_output/mzcount/
```

6.12 MZHash

Note: `MZHash` replaces the deprecated `MZMd5`.

`MZHash` recursively scans a folder and generates hashes for all files that begin with the MZ header — the signature of Windows PE executables. It's useful for building known-good or known-bad hash sets during triage, reverse engineering, or threat hunting.

You can select one or more hash algorithms at runtime: **MD5**, **SHA1**, or **SHA256**. If multiple algorithms are selected, a hash file and TSV lookup table will be generated for each.

The program outputs: - A text file with one hash per line. - A TSV (tab-separated values) file with full file paths and their corresponding hashes.

By default, hashes are saved to `saved_output/mzhash/`. If the file already exists, the user will be prompted before overwriting.

These hash sets (.tsv preferred) can be used with HashCheck.

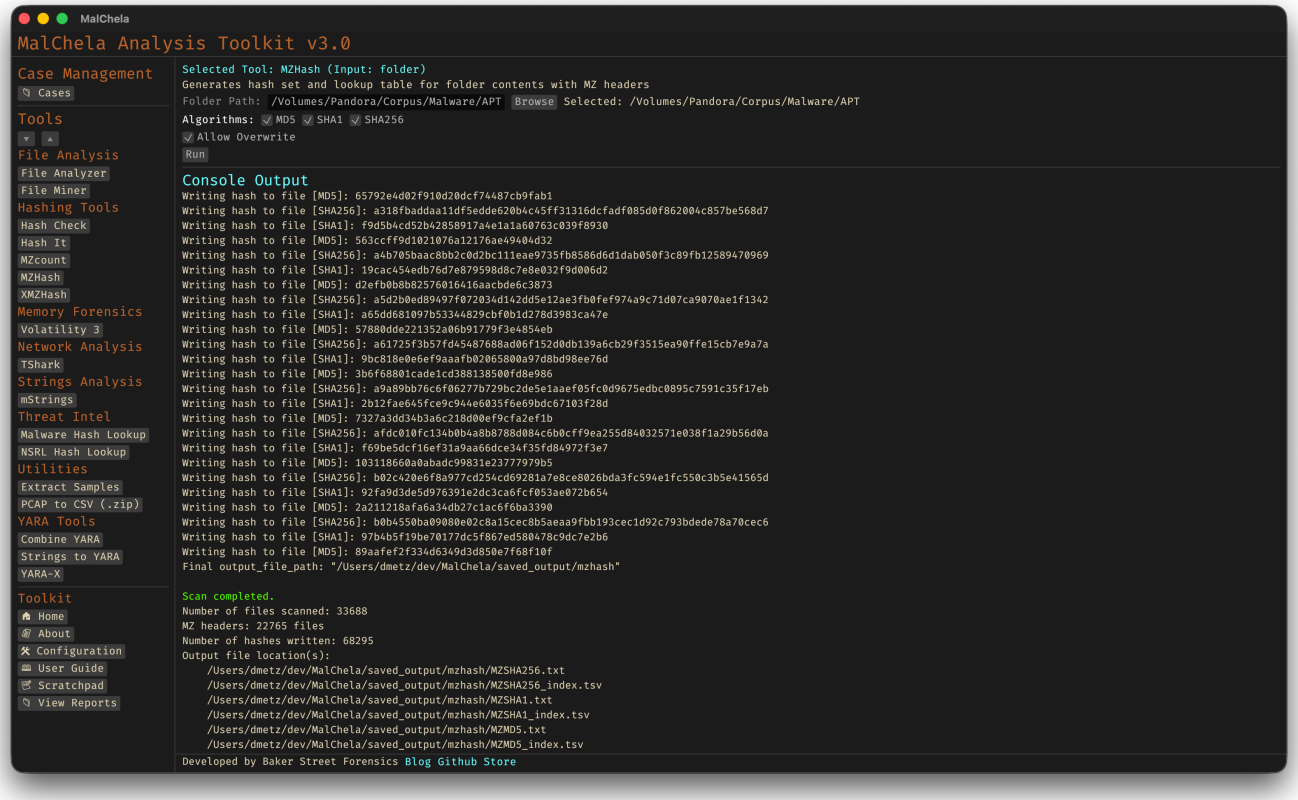

Figure 22: MZHash

🔧 CLI Syntax

```
# Example 1: Generate default SHA256 hashes
cargo run -p mzhash -- /path_to_directory/

# Example 2: Generate all three hash types (MD5, SHA1, SHA256)
cargo run -p mzhash -- /path_to_directory/ -a MD5 -a SHA1 -a SHA256

# Example 3: Generate SHA1 and SHA256 only
cargo run -p mzhash -- /path_to_directory/ -a SHA1 -a SHA256

# Example 4: Save output to a case folder
cargo run -p mzhash -- /path_to_directory/ -a SHA256 --case CaseName
```

If `--case` is specified, output will be saved to:

```
saved_output/cases/CaseName/mzhash/
```

Otherwise, hashes are saved to:

```
saved_output/mzhash/
```

You can combine multiple `-a` flags in any order.

6.13 NSRLQuery

NSRL Query checks a file hash against the National Software Reference Library (NSRL) by querying the CIRCL hash lookup service. It helps identify known, trusted software — allowing analysts to filter out benign files and focus on unknown or suspicious ones during forensic triage.

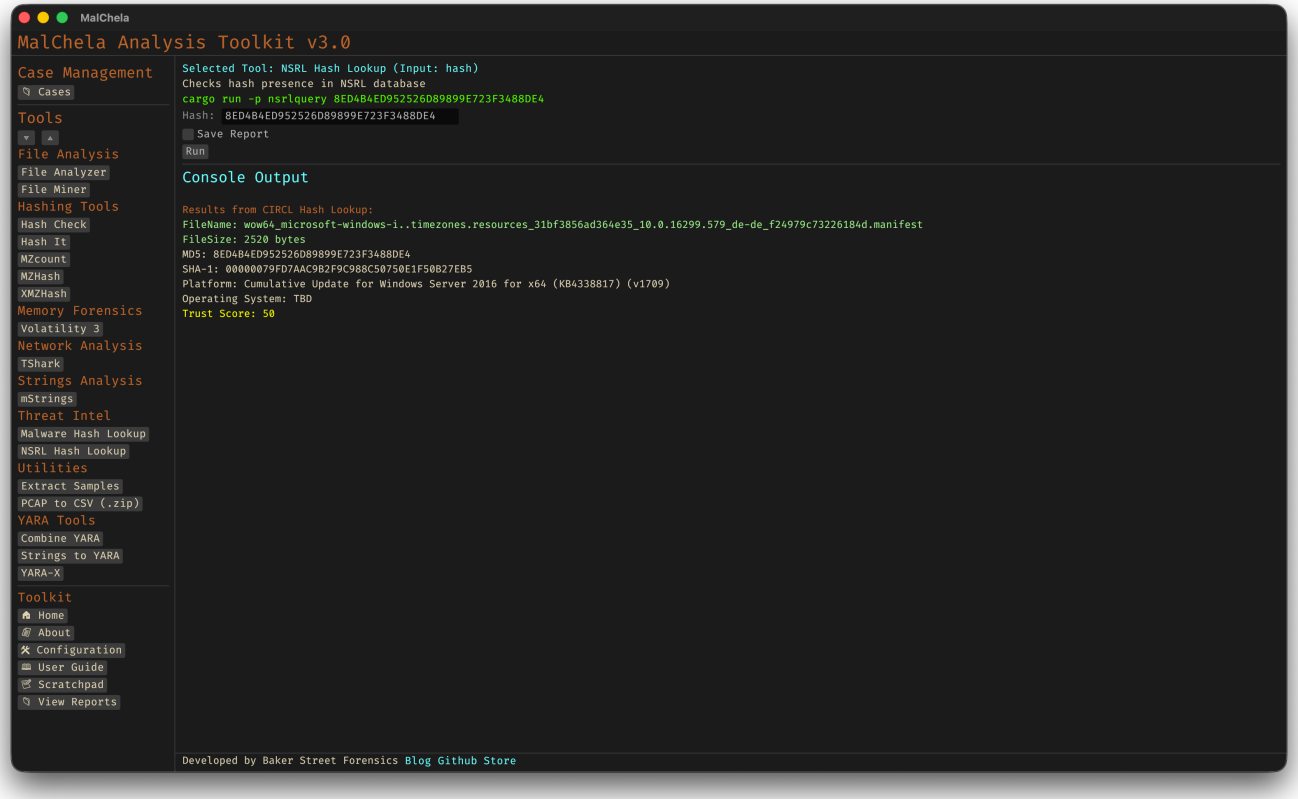

Figure 23: NSRL Hash Lookup

🔧 CLI Syntax

```
cargo run -p nsrlquery -- -o -t d41d8cd98f00b204e9800998ecf8427e
```

Performs a lookup using the CIRCL hashlookup API and saves the result as a `.txt` file.

Use `-o` to save output and include one of the following format flags: - `-t` → Save as `.txt` - `-j` → Save as `.json` - `-m` → Save as `.md`

If no hash is provided, the tool will prompt you to enter one interactively:

```
Enter the hash value:
```

Only MD5 and SHA1 hashes are supported. If an unsupported hash length is entered:

```
Error: Unsupported hash length. Please enter a valid MD5 (32 chars) or SHA1 (40 chars) hash.
```

To associate output with a case folder:

```
cargo run -p nsrlquery -- -o -j --case APT2025 d41d8cd98f00b204e9800998ecf8427e
```

This saves the `.json` output in `saved_output/cases/APT2025/nsrlquery/`.

6.14 StringsToYARA

Strings to YARA helps you rapidly build custom YARA rules by prompting for a rule name, optional metadata, and a list of string indicators. It integrates with the MalChela scratchpad, allowing you to paste or collect candidate strings interactively.

Lines beginning with hash: are deliberately ignored during rule generation — this lets you use the scratchpad to track hashes alongside strings without polluting your YARA rule content.

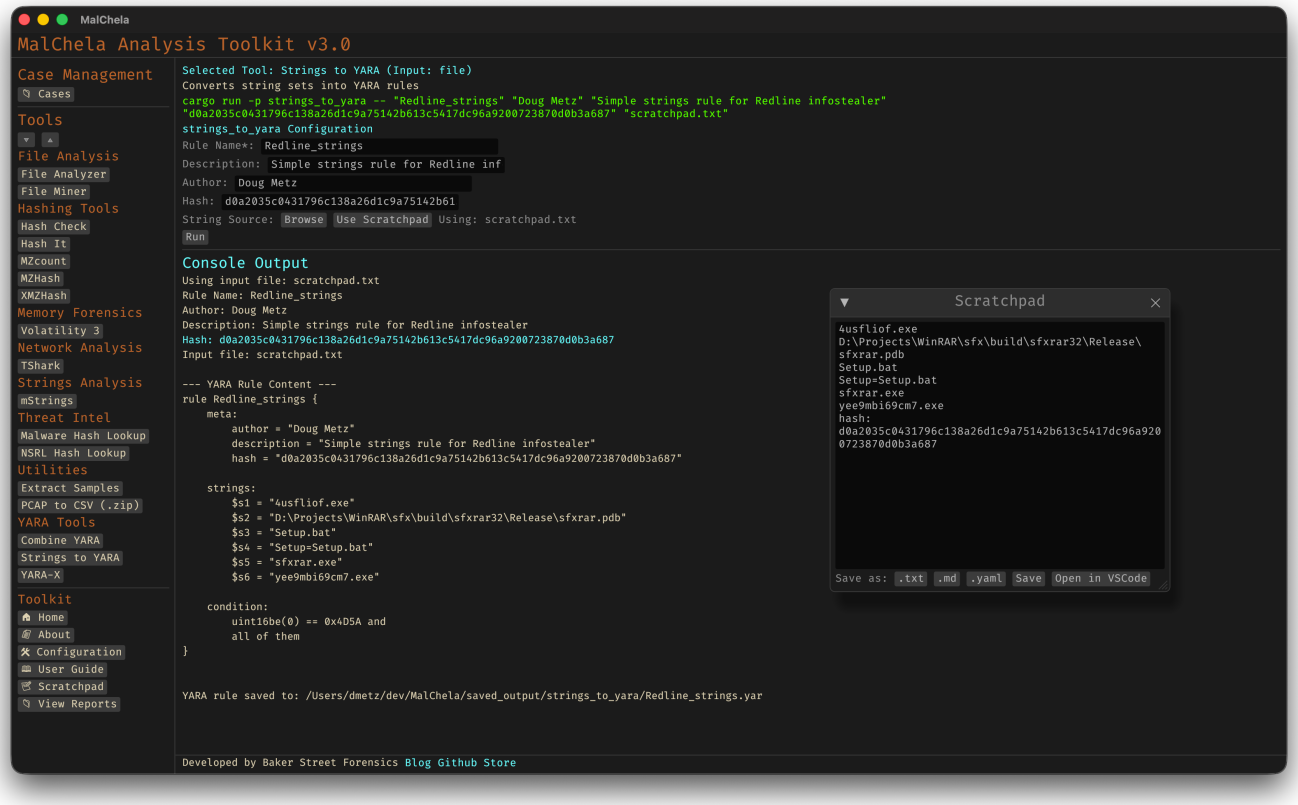

Figure 24: Strings to YARA

🔧 CLI Syntax

```
cargo run -p strings_to_yara -- RuleName Author Description Hash /path/to/strings.txt --case CaseName
```

You can supply up to five positional arguments. If any are omitted, the tool will prompt you interactively.

```
Enter rule name:
Enter author:
Enter description:
Enter hash (optional):
Enter path to string list file:
```

If the `--case` flag is supplied, the resulting YARA rule will be saved in the corresponding `saved_output/cases/CaseName/` directory.

Lines in the string file that begin with `hash:` are ignored and will not be included in the generated rule.

6.15 XMZHash

Note: `XMZHash` replaces the deprecated `XMZMd5`.

`XMZHash` recursively scans a directory and generates hashes for all files that **do not** match common binary or archive signatures such as MZ, ZIP, or PDF. It's ideal for uncovering unusual or misclassified files that may require deeper inspection or reverse engineering. Use this on a malware corpus to help surface non-Windows malware samples.

You can select one or more hash algorithms at runtime: **MD5**, **SHA1**, or **SHA256**. If multiple algorithms are selected, a hash file and TSV lookup table will be generated for each.

The program outputs: - A text file with one hash per line. - A TSV (tab-separated values) file with full file paths and their corresponding hashes.

By default, hashes are saved to `saved_output/mzhash/`. If the file already exists, the user will be prompted before overwriting.

These hash sets (.tsv preferred) can be used with HashCheck.

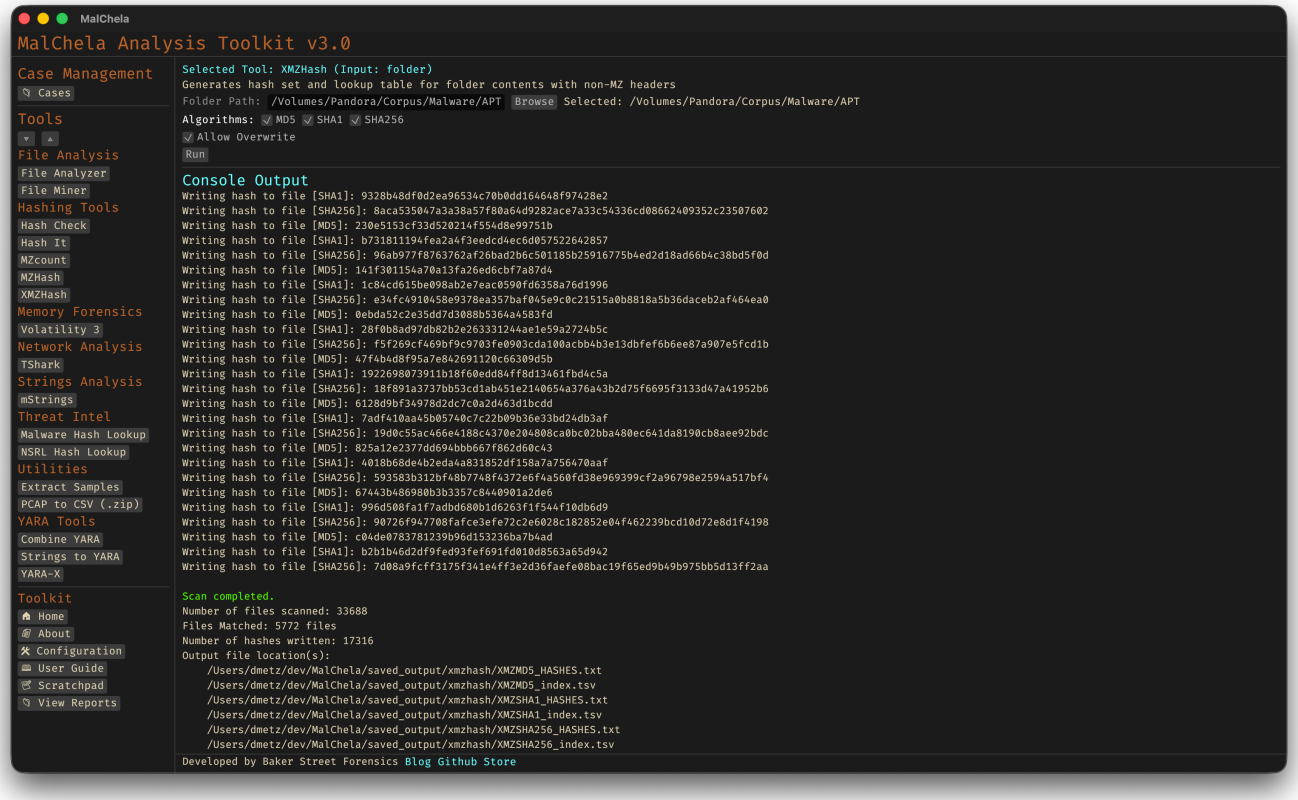

Figure 25: XMZHash

🔧 CLI Syntax

```
cargo run -p xmzhash -- /path_to_directory/
```

Generates SHA256 hashes (default).

```
cargo run -p xmzhash -- -a MD5 -a SHA1 -a SHA256 /path_to_directory/
```

Generates all three hash types.

```
cargo run -p xmzhash -- -a SHA1 -a SHA256 /path_to_directory/
```

Generates SHA1 and SHA256 only.

```
cargo run -p xmzhash -- -a SHA256 /path_to_directory/ --case MyCase
```

Saves results to the specified case folder.

You can combine multiple `-a` flags in any order. If no directory is passed, you will be prompted.

7. Third-Party Tools

7.1 Integrating Third-Party Tools

MalChela supports the integration of external tools such as Python-based utilities (`oletools` , `oledump`) and high-performance YARA engines (`yara-x`). These tools expand MalChela's capabilities beyond its native Rust-based toolset.

Tools now require `exec_type` (e.g., `cargo` , `binary` , `script`) to define how they are launched, and `file_position` to clarify argument order when needed.

To integrate a new tool into the GUI, ensure the tool: - Accepts CLI arguments in the form `toolname [args] [input]` - Outputs results to stdout - Is installed and available in `$PATH`

```
- name: toolname
  description: "Short summary of tool purpose"
  command: ["toolname"]
  input_type: file  # or folder or hash
  category: "File Analysis"  # or other GUI category
  optional_args: []
  exec_type: binary  # or cargo / script
  file_position: last  # or first, if required
```

You can switch to a prebuilt `tools.yaml` for REMnux mode via the GUI configuration panel — useful for quick setup in forensic VMs.

7.2 Configuration Reference

7.2.1 Tool Configuration

MalChela uses a central `tools.yaml` file to define which tools appear in the GUI, along with their launch method, input types, categories, and optional arguments. This YAML-driven approach allows full control without editing source code.

Key Fields in Each Tool Entry

Field	Purpose
name	Internal and display name of the tool
description	Shown in GUI for clarity
command	How the tool is launched (binary path or interpreter)
exec_type	One of `cargo`, `binary`, or `script`
input_type	One of `file`, `folder`, or `hash`
file_position	Controls argument ordering
optional_args	Additional CLI arguments passed to the tool
category	Grouping used in the GUI left panel

⚠️ All fields except `optional_args` are required.

7.2.2 Swapping Configs: REMnux Mode and Beyond

MalChela supports easy switching between tool configurations via the GUI.

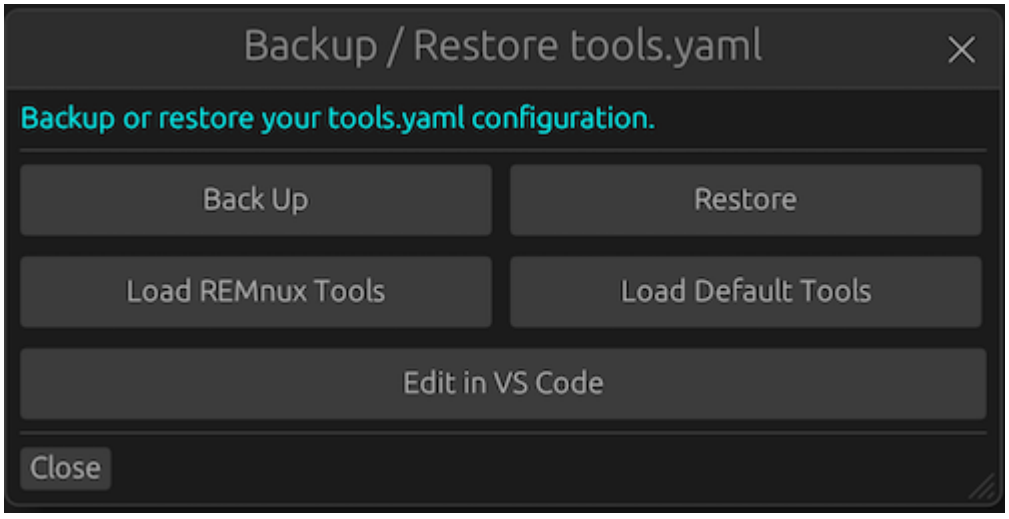

Figure 26: YAML Config Tool

To switch:

- Open the **Configuration Panel**
- Use **"Select tools.yaml"** to point to a different config
- Restart the GUI or reload tools

This allows forensic VMs like REMnux to use a tailored toolset while keeping your default config untouched.

A bundled `tools_remnux.yaml` is included in the repo for convenience.

KEY TIPS

- Always use `file_position: "last"` unless the tool expects input before the script
- For scripts requiring Python, keep the script path in `optional_args[0]`
- For tools installed via `pipx`, reference the binary path directly in `command`

7.2.3 Backing Up and Restoring tool.yaml

The MalChela GUI provides built-in functionality to back up and restore your `tools.yaml` configuration file.

Backup

To create a backup of your current `tools.yaml`:

- Open the **Configuration Panel**
- Click the **"Back Up Config"** button
- A timestamped copy of `tools.yaml` will be saved to the default location

You'll see a confirmation message when the operation completes successfully.

Restore

To restore from a previous backup:

- Click the **"Restore Config"** button in the Configuration Panel
- Select a previously saved backup file
- The selected file will overwrite the current configuration

This feature makes it easy to experiment with custom tool setups while retaining a safety net for recovery.

7.3 Enhanced Integrations

Enhanced configurations have been preconfigured for several third-party tools such as TShark and Volatility, enabling streamlined integration with MalChela. These tools now support saving output directly into the active Case folder when launched from the GUI, preserving structured reports and maintaining context across sessions.

Additionally, dedicated setup instructions are provided for Python-based tools like oledump and olevba, as well as installation guidance for utilities like YARA-X to ensure consistent and reliable operation across environments.

7.4 TShark

TShark Field Reference Panel

If TShark is included in your `tools.yaml` (or if you're using the REMnux configuration), the GUI offers a powerful set of tools to assist with display filter creation and usage. This includes both an integrated filter builder and a TShark Field Reference panel.

- The filter builder allows users to construct and modify complex TShark display filters directly within the GUI, with real-time syntax support and validation.
- The "?" icon next to filter fields launches the Field Reference panel, which provides searchable field definitions, examples, tooltips, and a copy-to-clipboard feature.
- Together, these tools help analysts visually explore and test filter syntax without needing to memorize protocol-specific field names.
- Output can be saved to the default location or to a specific case directory.

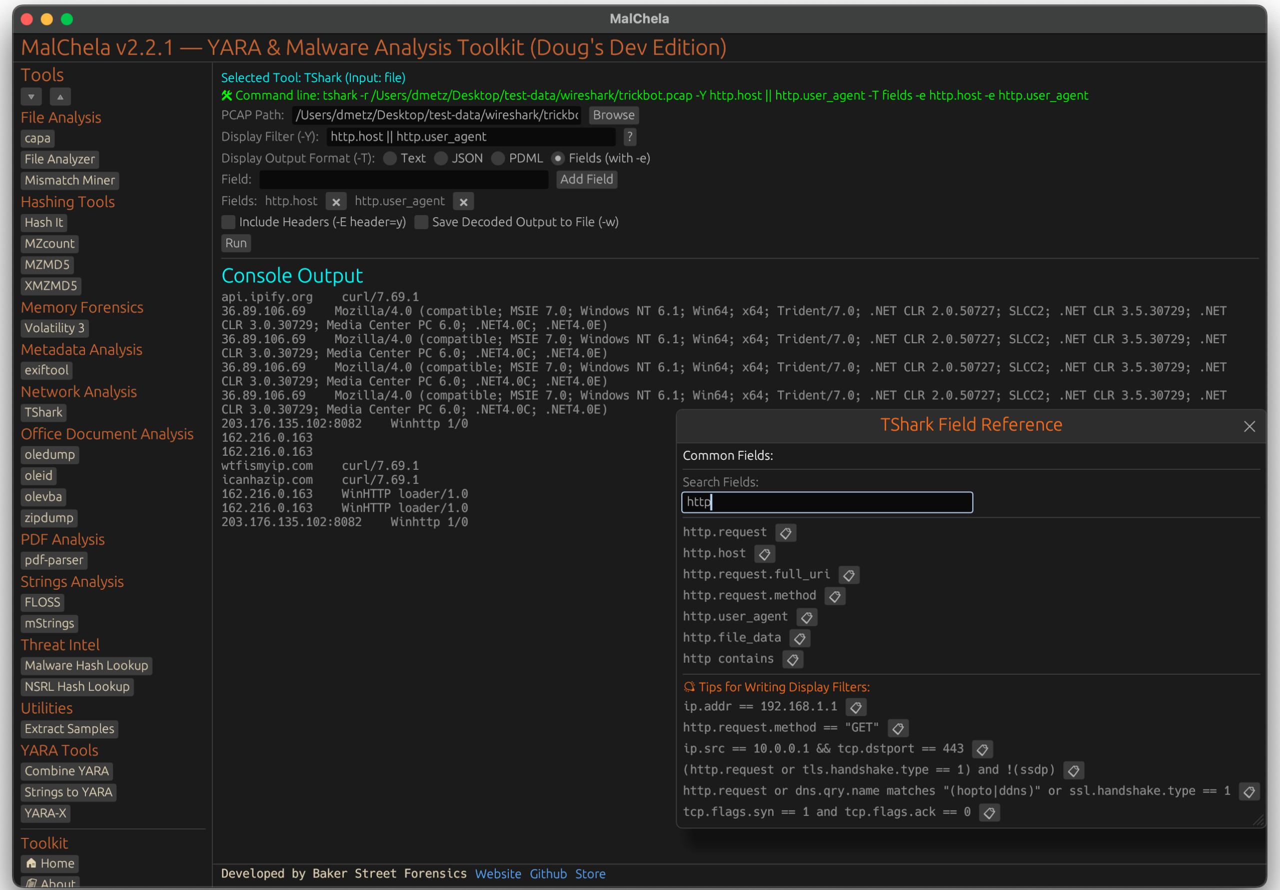

Figure 27: TShark

7.5 PCAP to CSV Conversion

The `tshark_to_csv.py` script is a helper utility that converts `.pcap` or `.pcapng` files into structured `.csv` files using `tshark`. This allows packet captures to be processed and analyzed using familiar spreadsheet or database tools.

Figure 28: PCAP to CSV

The script uses `tshark` to extract key fields from each packet and writes them to a CSV file with headers. Users can specify custom fields, or rely on a default set commonly useful for forensic review.

Timestamps in the CSV output are converted into a human-readable ISO 8601 format with microsecond precision (`YYYY-MM-DDTHH:MM:SS.ssssssZ`), ensuring compatibility with downstream tools such as the Magnet Custom Artifact Generator.

This script is included in the MalChela repository under `Utilities/`, and is referenced in `tools.yaml` but commented out by default. Uncomment the entry in `tools.yaml` to enable it in the GUI.

Requirements:

- Python 3
- Wireshark/tshark (must be installed and available in PATH)

7.6 Volatility 3

MalChela integrates support for **Volatility 3**, a powerful memory forensics framework. This tool enables analysts to examine memory dumps for signs of compromise, persistence mechanisms, and malicious activity.

7.6.1 Integration Overview

Volatility 3 is available in MalChela as an enhanced third-party tool. The GUI provides a dedicated interface for selecting plugins, supplying arguments, and reviewing results — all within a structured panel that mimics the CLI workflow but adds quality-of-life improvements like:

- Live search and categorized plugin reference
- Plugin-specific argument helpers
- Color-coded output for easier review
- Output saving and file dump options
- Output can be saved to the default location or to a specific case directory.

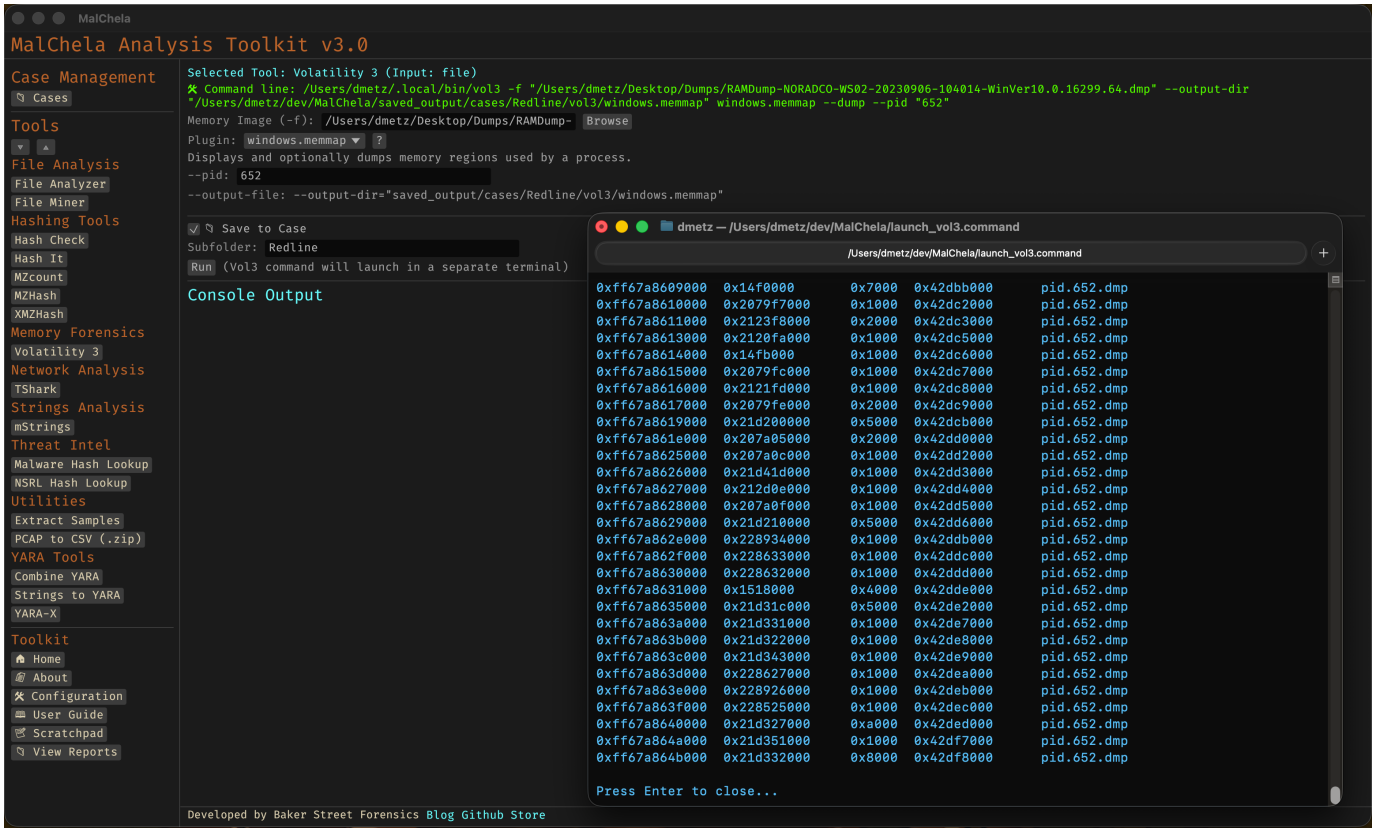

Figure 29: Volatility (launches in separate terminal)

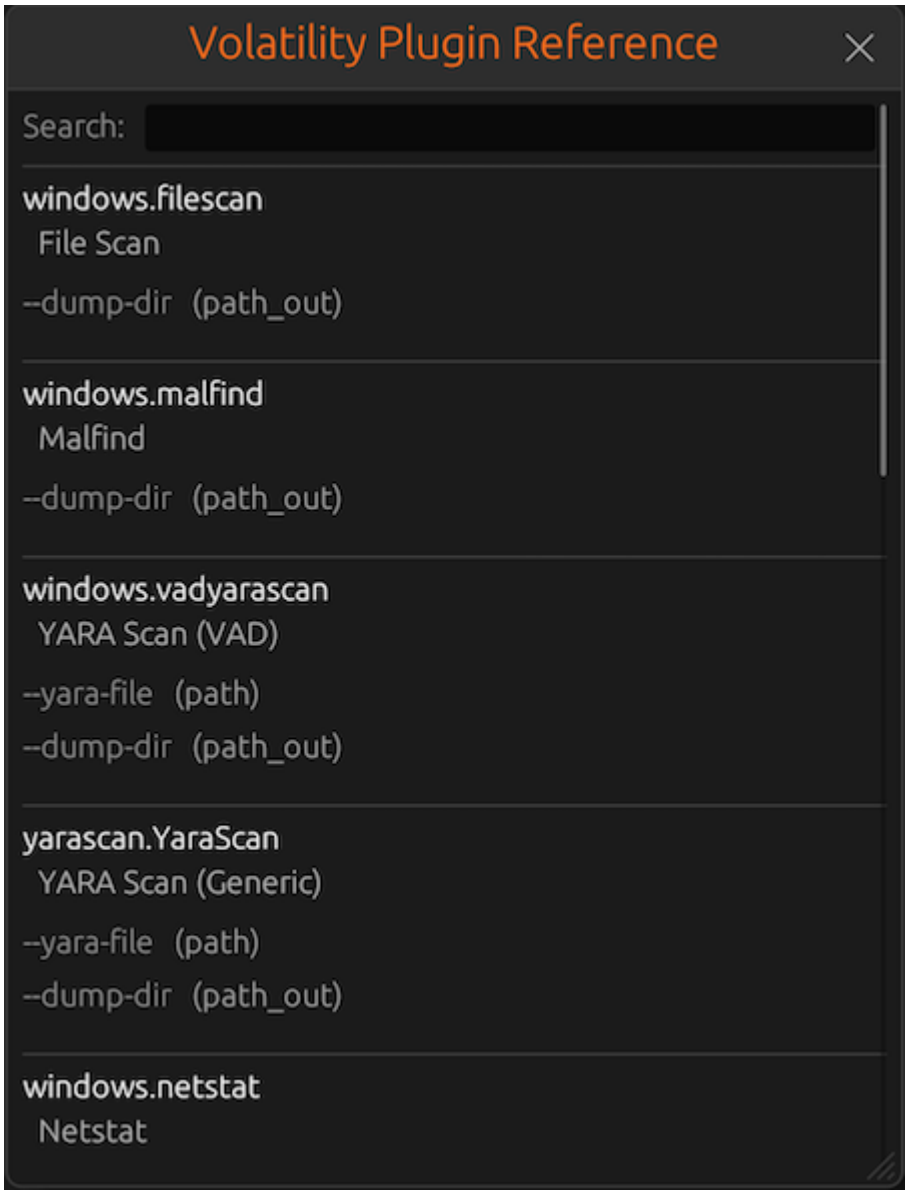

Figure 30: Volatility Plugin Reference

7.6.2 Requirements

To use Volatility 3 within MalChela:

- You must have `vol3` (Volatility 3 CLI) installed and accessible in your system `$PATH`.
- On REMnux, `vol3` is preinstalled and configured automatically.
- On macOS or Linux, you can install it via pip: `pip install volatility3`

7.6.3 tools.yaml Configuration

To use Volatility 3 with the GUI launcher, ensure the correct `command` value is defined in your `tools.yaml` configuration. Depending on your environment, the binary may be installed under different names or locations.

Two common examples:

```
- name: Volatility 3
  description: "Memory analysis using Volatility 3"
  command: ["/Users/dmetz/.local/bin/vol3"]
  input_type: "file"
  file_position: "first"
  category: "Memory Forensics"
  gui_mode_args: []
  exec_type: binary

- name: Volatility 3
  description: "Memory analysis using Volatility 3"
  command: ["vol3"]
  input_type: "file"
  file_position: "first"
  category: "Memory Forensics"
  gui_mode_args: []
  exec_type: script
```

Make sure that the specified binary path or command is accessible in your system's `$PATH`.

7.6.4 Example Use Cases

- Enumerate processes: `windows.pslist`
- Dump suspicious files from memory: `windows.dumpfiles --dump-dir /output/path`
- Detect injected code: `windows.malfind`
- YARA scanning on memory: `windows.vadyarascan --yara-file rules.yar`

7.6.5 Output and Reports

All plugin results are streamed to the GUI console with formatting preserved. Where supported, plugins that produce dumped files will output to a user-specified folder.

7.6.6 Known Limitations

- Some plugins require symbol files (`.pdb`) to function correctly. Volatility will display a warning if missing.
- Ensure sufficient system memory when analyzing large memory dumps.

7.6.7 Additional Resources

- Volatility 3 GitHub
- Official Documentation

7.7 Installing and Configuring YARA-X

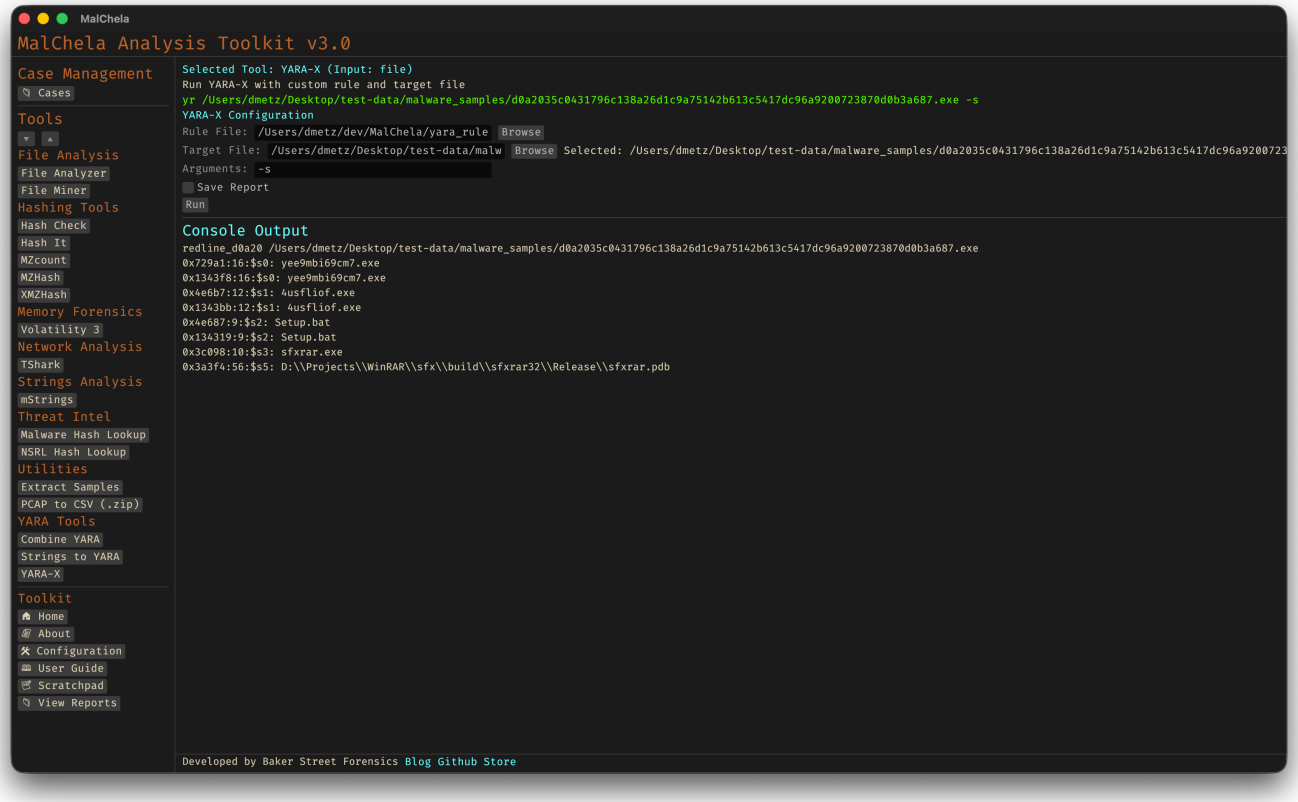

Figure 31: YARA-X

YARA-X is an extended version of YARA with enhanced performance and features. To integrate YARA-X with MalChela, follow these steps:

7.7.1 Installation

- **Download the latest release:**
  Visit the official YARA-X GitHub releases page at https://github.com/VirusTotal/yara-x/releases and download the appropriate binary for your platform.
- **Extract and install:**
  Extract the downloaded archive and place the `yara-x` binary (`yr`) in a directory included in your system's `$PATH`, or note its absolute path for configuration.
- **Verify installation:**
  Run the following command to confirm YARA-X is installed correctly:

```
yr –version
```

7.7.2 Configuration in MalChela

To use YARA-X within MalChela tools, update your `tools.yaml` with the following example entry:

```
- name: yara-x
  description: "High-performance YARA-X engine"
  command: ["yara-x"]
  input_type: "file"
  file_position: "last"
  category: "File Analysis"
```

```
  optional_args: []
  exec_type: binary
```

7.7.3 Using YARA-X Rules

- Place your YARA rules in the `yara_rules` folder within the workspace.
- YARA-X supports recursive includes and extended features; ensure your rules are compatible.
- The MalChela GUI and CLI will invoke YARA-X when configured as above, providing faster scans and improved detection.

7.7.4 Tips

- For advanced usage, consult the YARA-X documentation for command-line options and rule syntax.

7.8 Python Integrations

Configuring Python-Based Tools (oletools & oledump)

MalChela supports Python-based tools as long as they are properly declared in `tools.yaml`. Below are detailed examples and installation instructions for two commonly used utilities:

🔧 `olevba` (FROM `oletools`)

Install via `pipx`:

```
pipx install oletools
```

This installs `olevba` as a standalone CLI tool accessible in your user path.

`tools.yaml` configuration example:

```
- name: olevba
  description: "OLE document macro utility"
  command: ["/Users/youruser/.local/bin/olevba"]
  input_type: "file"
  file_position: "last"
  category: "Office Document Analysis"
  optional_args: []
  exec_type: script
```

Notes:

- `olevba` is run directly (thanks to pipx)
- No need to specify a Python interpreter in `command`
- Ensure the path to `olevba` is correct and executable

—

`oledump` (STANDALONE SCRIPT)

Manual installation:

```
mkdir -p ~/Tools/oledump
cd ~/Tools/oledump
curl -O https://raw.githubusercontent.com/DidierStevens/DidierStevensSuite/master/oledump.py
chmod +x oledump.py
```

Make sure the script path in `optional_args` is absolute, and that the file is executable if it's run directly (not through a Python interpreter in `command`).

Dependencies:

```
python3 -m pip install olefile
```

Alternatively, create a virtual environment to isolate dependencies:

```
python3 -m venv ~/venvs/oledump-env
source ~/venvs/oledump-env/bin/activate
pip install olefile
```

`tools.yaml` configuration example:

```
- name: oledump
  description: "OLE Document Dump Utility"
  command: ["/usr/local/bin/python3"]
  input_type: "file"
  file_position: "last"
  category: "Office Document Analysis"
  optional_args: ["/Users/youruser/Tools/oledump/oledump.py"]
  exec_type: script
```

Notes:

- The GUI ensures correct argument order: `python oledump.py <input_file>`
- `command` points to the Python interpreter
- `optional_args` contains the path to the script

8. REMnux Mode

MalChela includes built-in support for running in **REMnux Mode**, a configuration designed specifically for seamless operation within the REMnux malware analysis distribution.

8.1 How REMnux Mode Works

REMnux Mode is a configuration profile that aligns MalChela's behavior with the REMnux malware analysis distribution. It adjusts paths, tool definitions, and GUI presentation to match the REMnux environment.

This mode is manually enabled by selecting **"Load REMnux"** from the tools.yaml Configuration Panel in the GUI. Once selected, a REMnux-specific `tools.yaml` file is loaded and remains active until you replace it with another configuration.

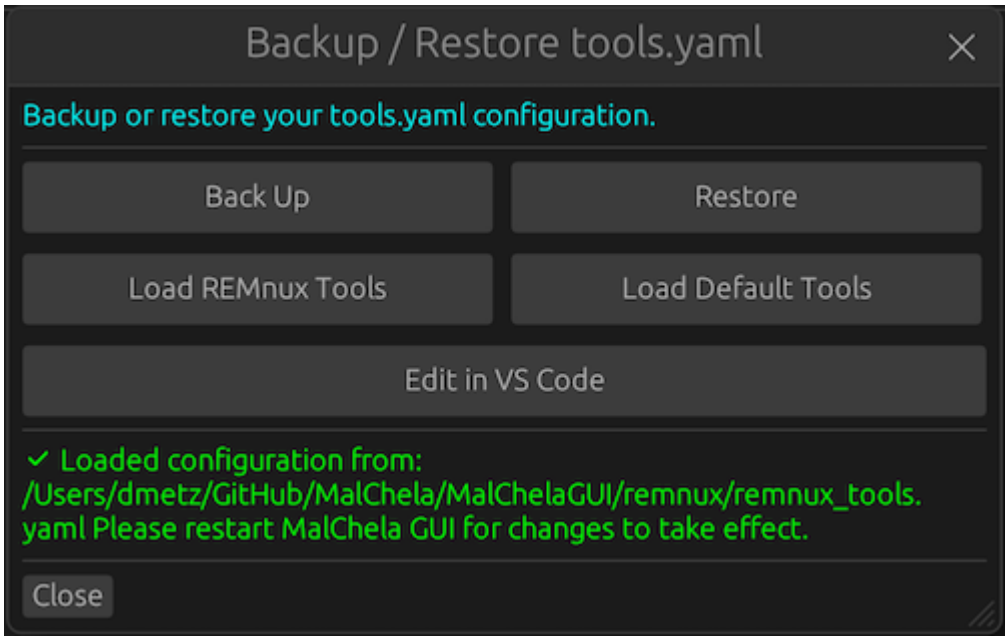

Figure 32: Enabling REMnux mode

Note: Whenever you change the tools.yaml, you need to restart the GUI for it to take effect.

Figure 33: MalChela in REMnux Mode

8.2 Preconfigured Tools in REMnux Mode

The following tools will appear in the GUI when REMnux Mode is enabled. Each is preconfigured with known-good paths for the REMnux environment:

8.2.1 File Analysis

Tool	Description	Command
binwalk	Scan binary files for embedded files	`binwalk`
capa	Detects capabilities in binaries via rules	`capa`
FLOSS	Extract obfuscated strings from binaries	`floss`
radare2	Scan binary files	`/usr/bin/r2 -i`

8.2.2 Memory Forensics

Tool	Description	Command
Volatility 3	Memory analysis using Volatility 3	`vol3`

8.2.3 Metadata Analysis

Tool	Description	Command
exiftool	Extract metadata from files	`exiftool`

8.2.4 Network Forensics

Tool	Description	Command
TShark	Analyze network traffic	`tshark`

8.2.5 Office Document Analysis

Tool	Description	Command
mraptor	Detect auto-executing macros in Office docs	`mraptor`
oledump	Dump streams from OLE files	`oledump.py`
oleid	Analyze OLE files for suspicious indicators	`oleid`
olevba	Extract VBA macros from OLE files	`olevba`
rtfobj	Extract embedded objects from RTF files	`rtfobj`
zipdump	Parses and analyzes suspicious PDF structures	`zipdump.py`

8.2.6 PDF Analysis

Tool	Description	Command
pdf-parser	Parse structure and objects of a PDF file	`python3 /usr/local/bin/pdf-parser.py`

8.2.7 Utilities

Tool	Description	Command
clamscan	Antivirus scan using ClamAV	`clamscan`
strings	Extracts printable strings from binary files	`strings`

8.3 Benefits

- Tools like **Volatility 3**, **FLOSS**, **oledump**, and **olevba** are preconfigured and ready to go
- `tools.yaml` is auto-tailored to the REMnux environment
- You can still customize your tool entries, but defaults are optimized for REMnux paths and permissions
- Useful for education, triage labs, and portable analysis setups

8.4 Customizing the REMnux Experience

Although REMnux Mode provides sane defaults, you can still:

- Override tool entries in `tools.yaml`
- Add new third-party tools via the GUI or YAML
- Use the Configuration Panel to backup/restore your configuration

For more information about REMnux, visit REMnux.org.

9. Support

9.1 Support & Contribution

The MalChela project is open source and actively maintained. Contributions, feedback, and bug reports are always welcome. You can find the project on GitHub, where issues and pull requests are encouraged.

9.2 Known Limitations & Platform Notes

MalChela is designed to be cross-platform but has some current limitations:

- The **CLI** runs well on macOS, Linux, and WSL environments.
- The **GUI** is supported on macOS and Linux. It may also work under WSLg on Windows 11, but this is not officially tested.
- File paths must use **POSIX-style formatting** (e.g., `/home/user/file.txt`). Windows-style paths are not supported.
- If the `exec_type` field is missing or misconfigured in `tools.yaml`, GUI execution may fail or behave incorrectly.
- The `category` field in `tools.yaml` no longer impacts GUI execution behavior—it is only used for grouping in the interface.

9.3 iOS 26 Development Panic

If you're running iOS 26 beta and encounter a runtime panic like the following:

```
thread 'main' panicked at /Users/you/.cargo/registry/src/index.crates.io-*/icrate-0.0.4/.../NSEnumerator.rs:6:1: invalid message send ...
```

This is due to stricter signature checks in the Apple SDK when running in debug mode. The panic does **not** occur in release mode builds.

To avoid the crash, build and run in release mode:

```
cargo build --release
./target/release/MalChelaGUI
```

This issue does not affect normal usage or distribution of the application. Debug mode remains usable for building, but the GUI should be run from a release build on affected platforms.