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.
- 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.

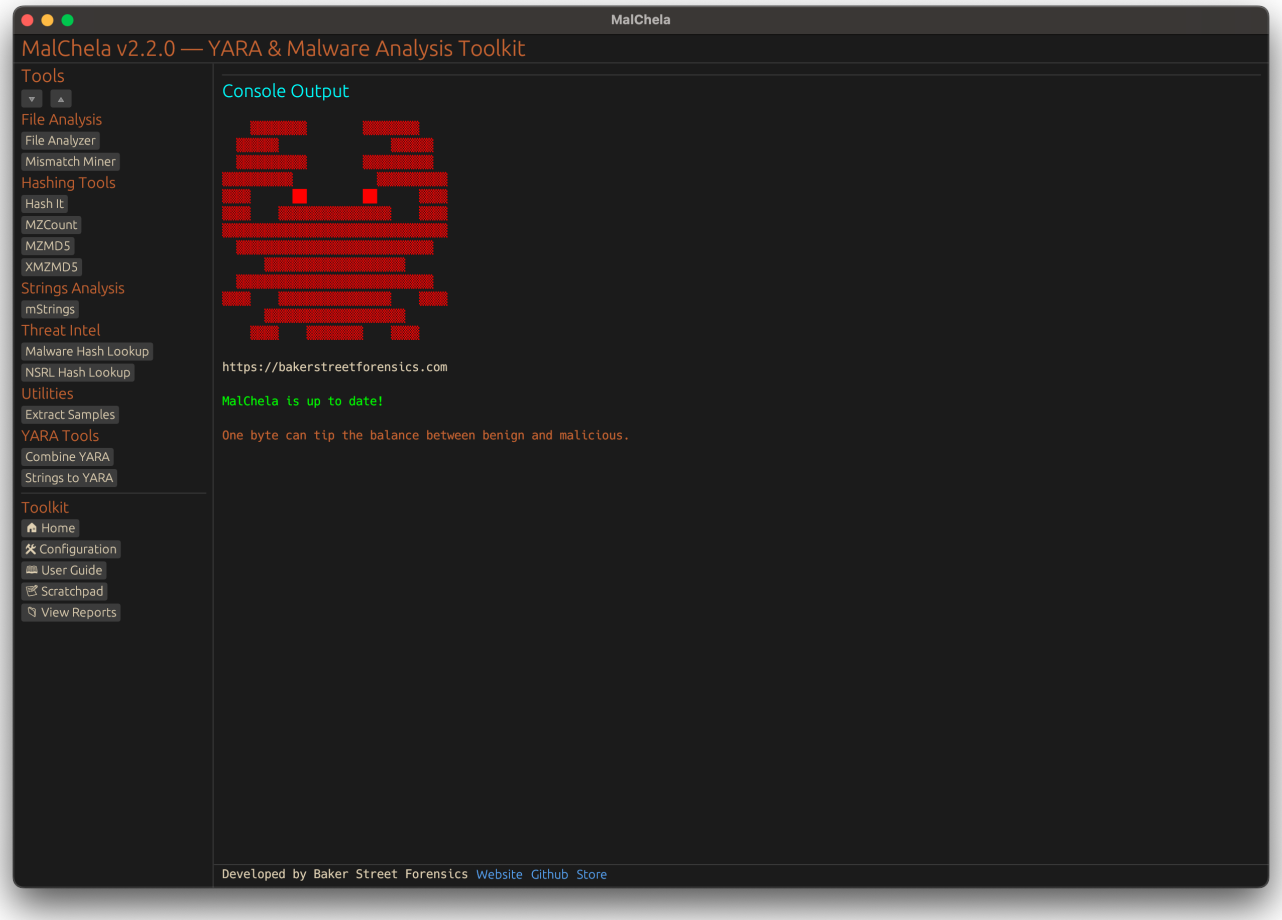

Figure 1: MalChela GUI

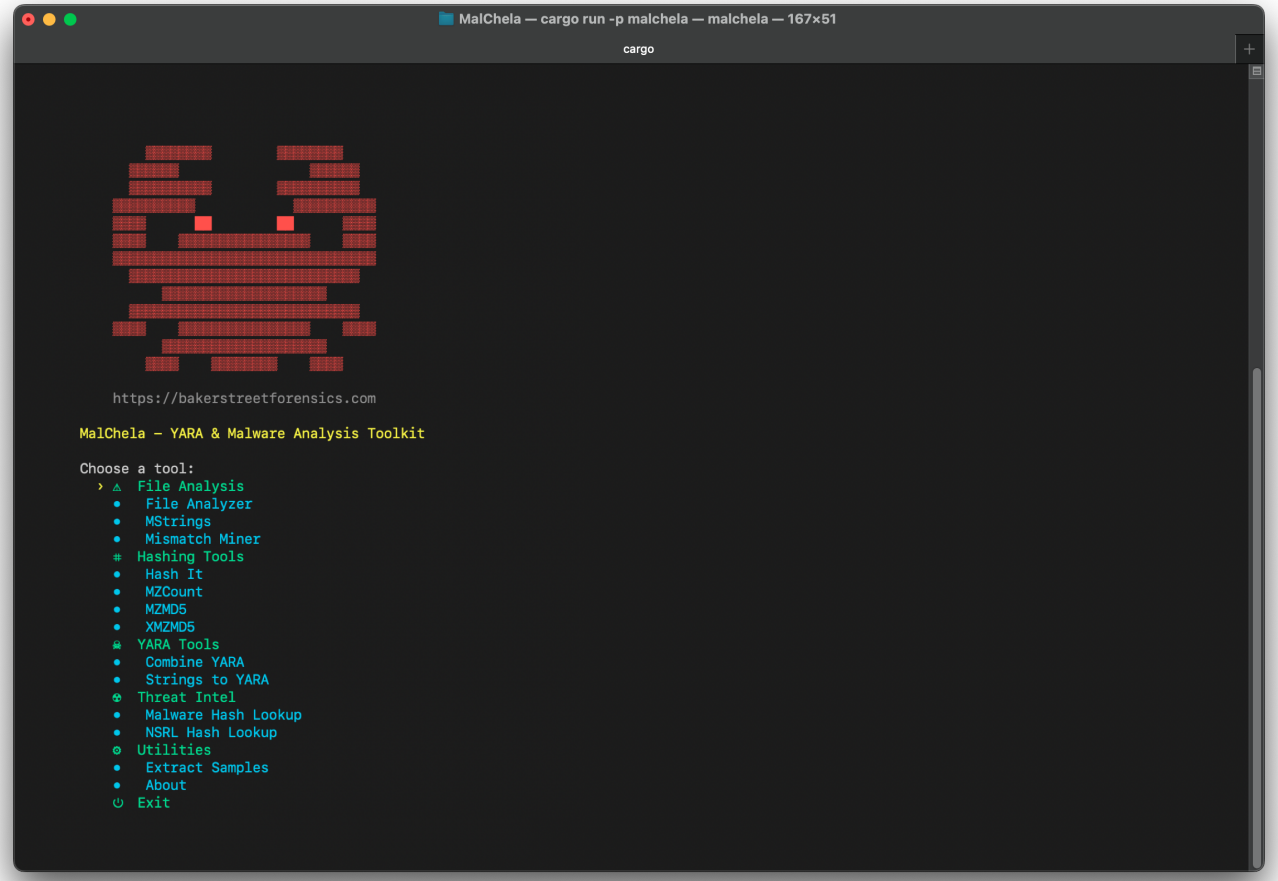

Figure 2: MalChela CLI

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                # Build all tools
cargo build -p fileanalyzer # Build individual tool
```

3.0.5 Windows Notes

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

4. Third-Party Tools

4.1 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.

4.2 Configuration Reference

4.2.1 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.

4.2.2 Swapping Configs: REMnux Mode and Beyond

MalChela supports easy switching between tool configurations via the GUI.

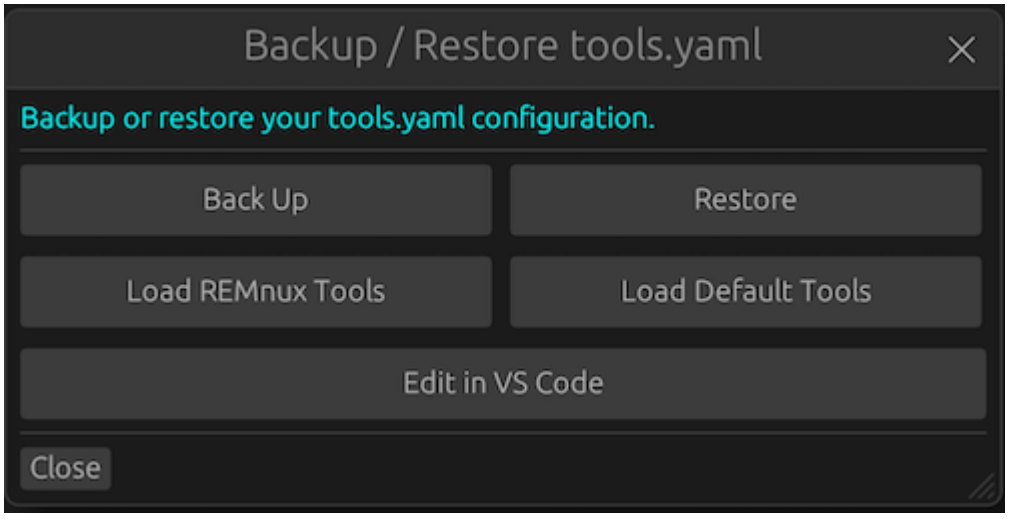

Figure 20: YAML Config Tool

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`

4.2.3 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.

4.3 Enhanced Integrations

Enhanced configurations have been preconfigured for several third-party tools such as TShark and Volatility, enabling streamlined integration with MalChela. 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.

4.4 FLOSS

- FLOSS extracts static, stack, tight, and decoded strings from binaries.
- The GUI supports all CLI flags (e.g., `--only`, `--format`, `-n`, etc.).
- Occasionally, FLOSS may print a multiprocessing-related error such as: `from multiprocessing.resource_tracker import main;main(6)` This is a known issue and does not affect output. It can be safely ignored.

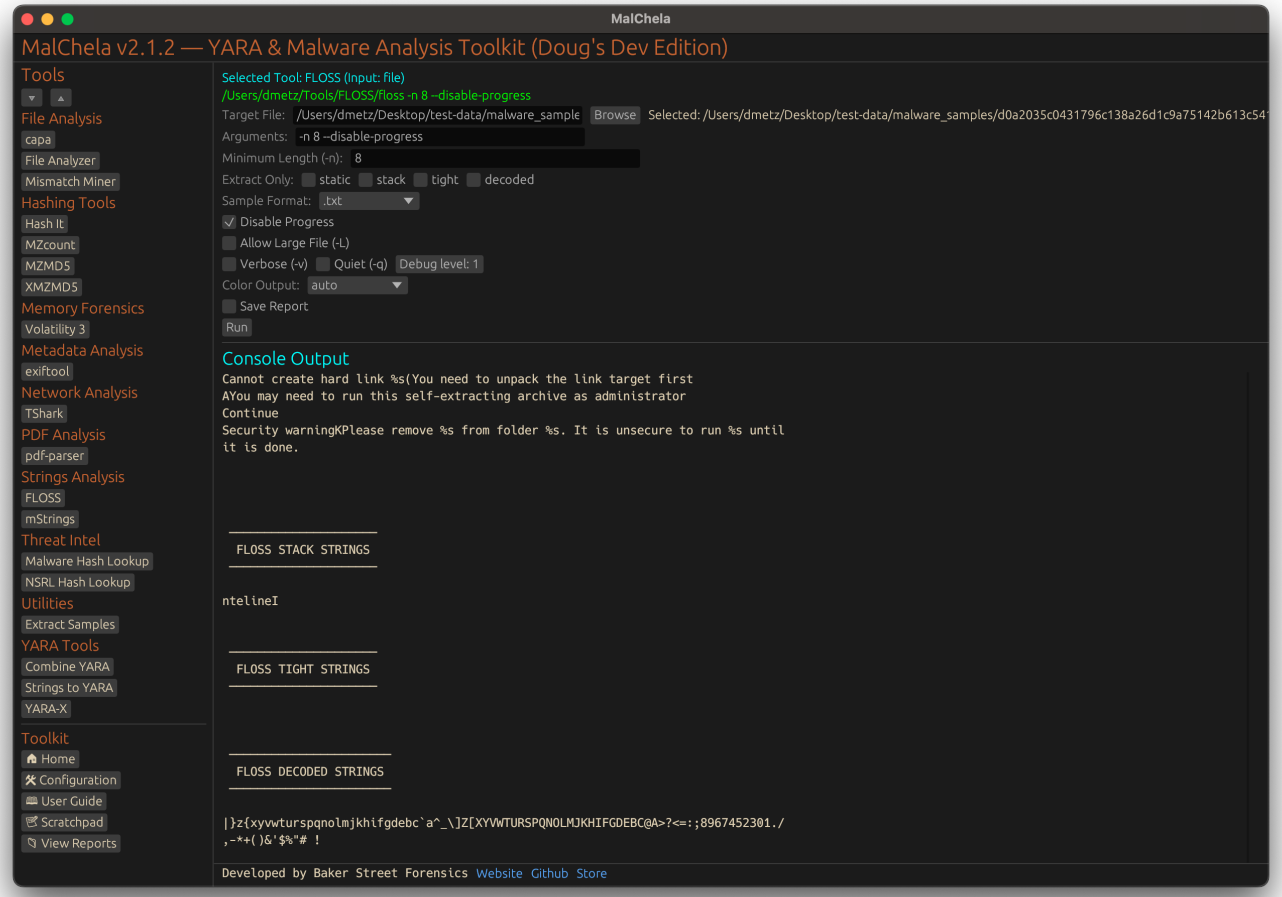

Figure 21: FLOSS

4.5 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.

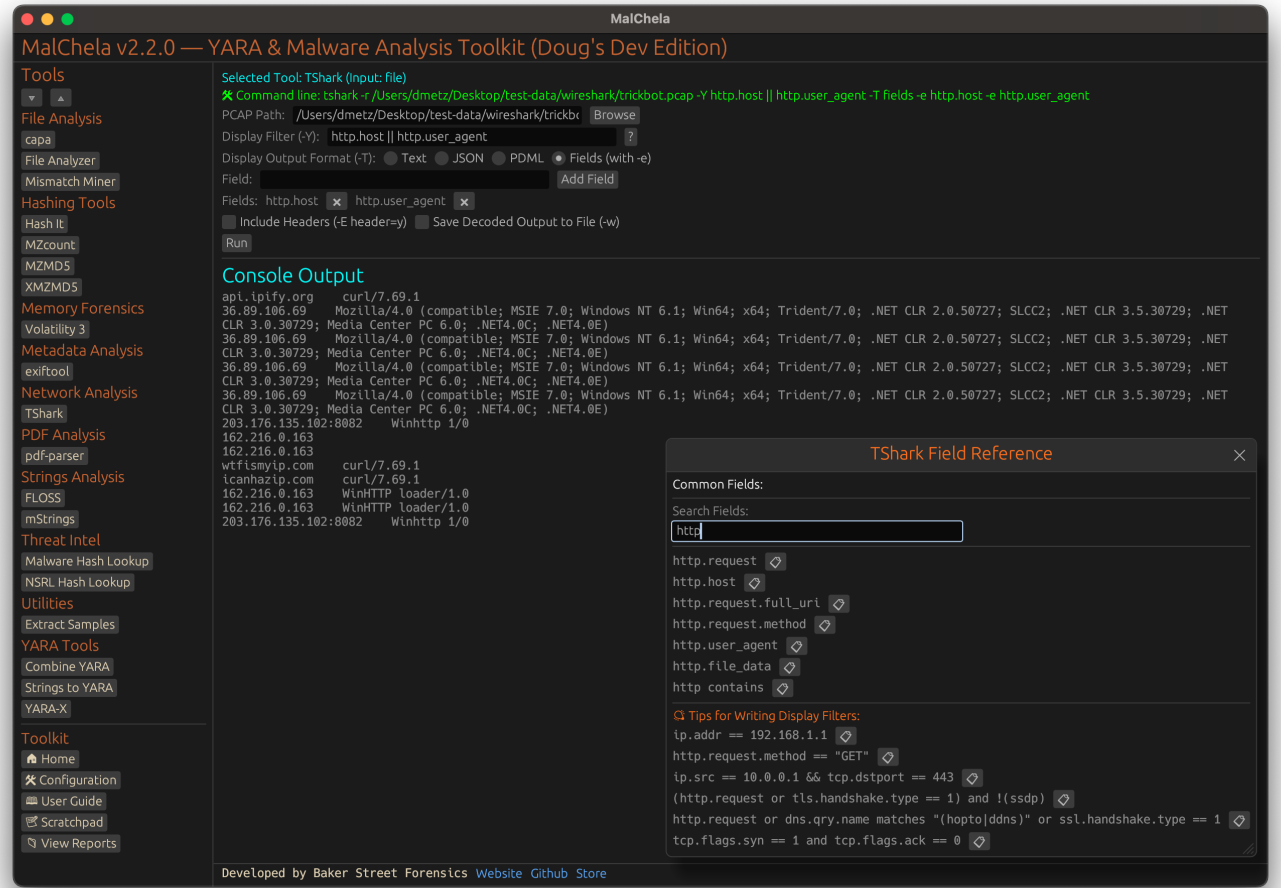

Figure 22: TShark

4.6 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.

4.6.1 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

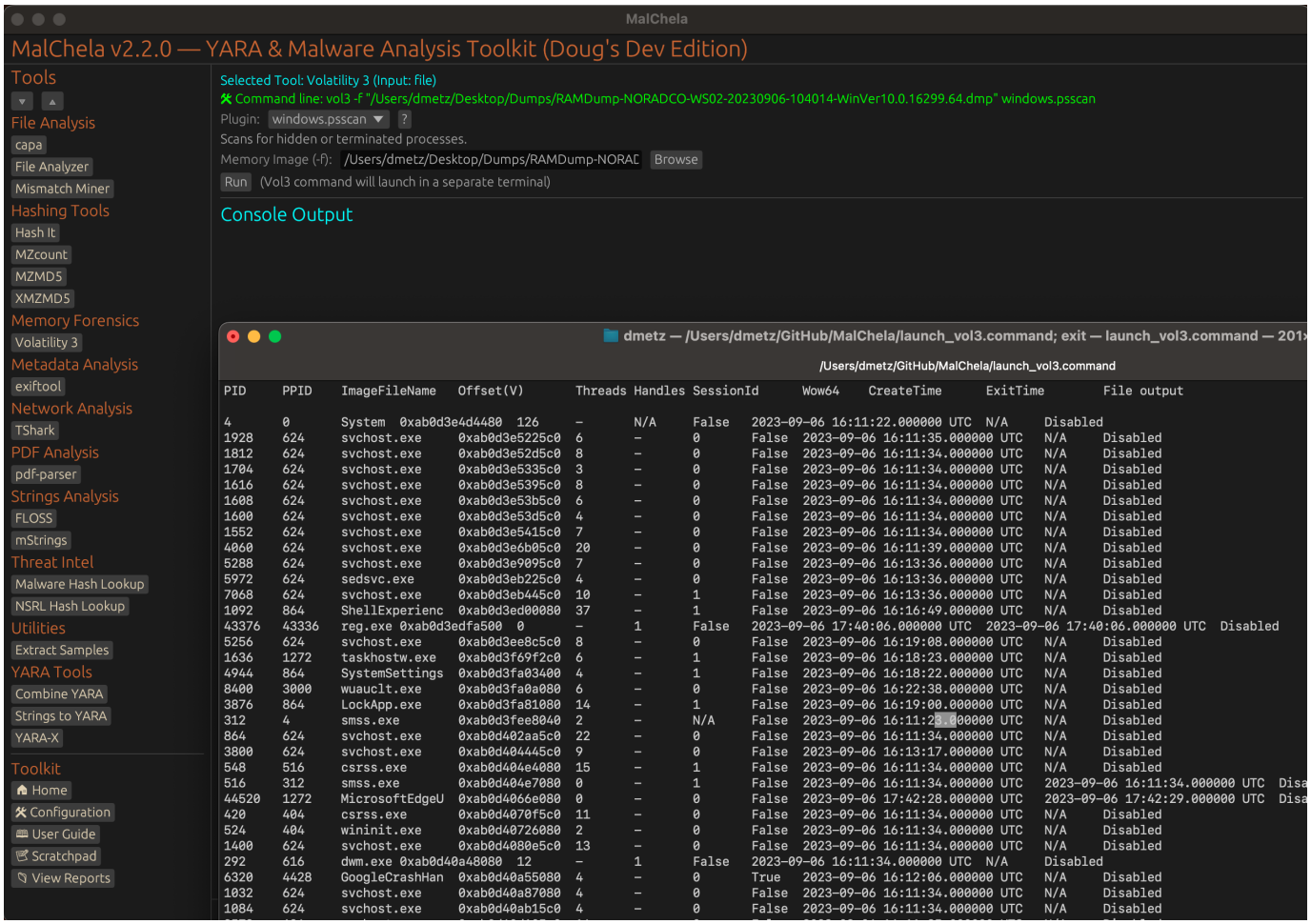

Figure 23: Volatility (launches in separate terminal)

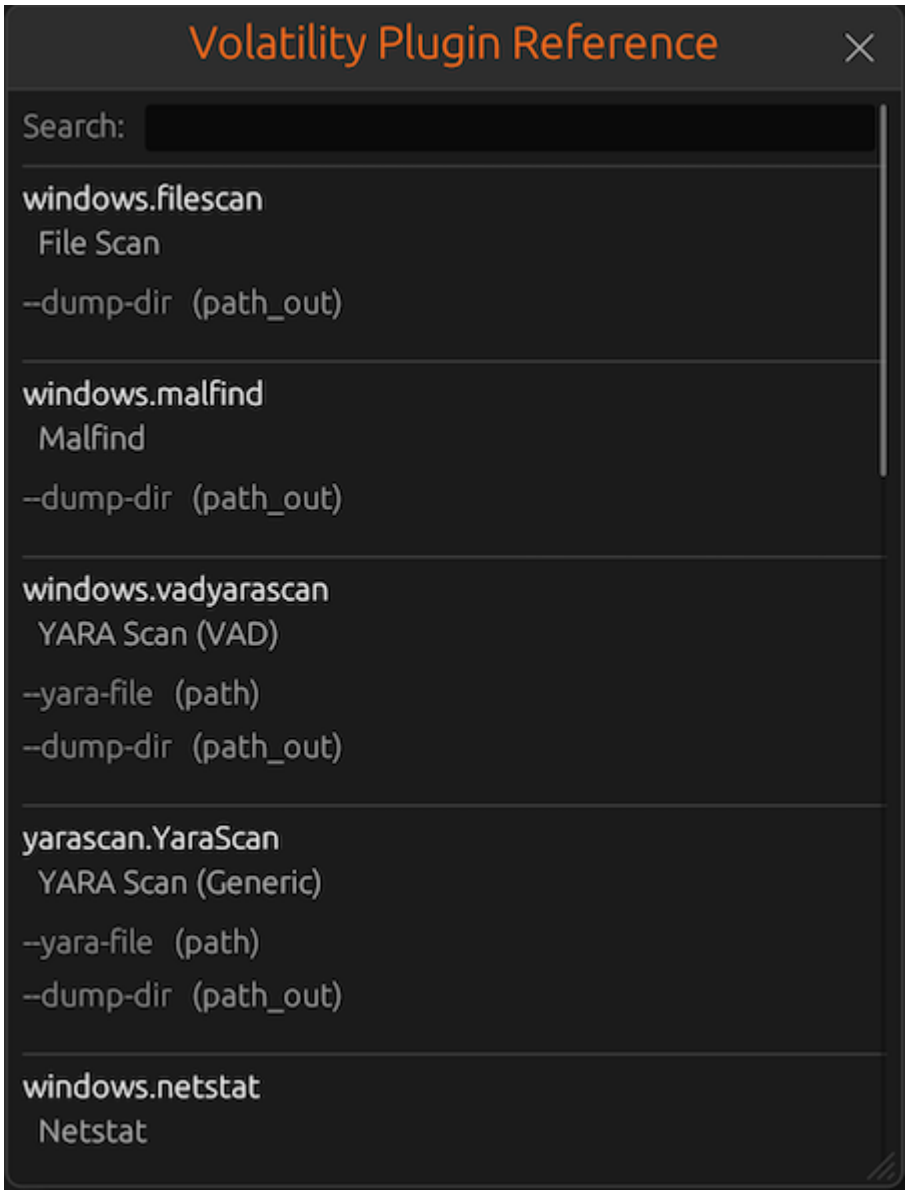

Figure 24: Volatility Plugin Reference

4.6.2 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`

4.6.3 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`.

4.6.4 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`

4.6.5 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.

4.6.6 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.

4.6.7 Additional Resources

- Volatility 3 GitHub
- Official Documentation

4.7 Installing and Configuring YARA-X

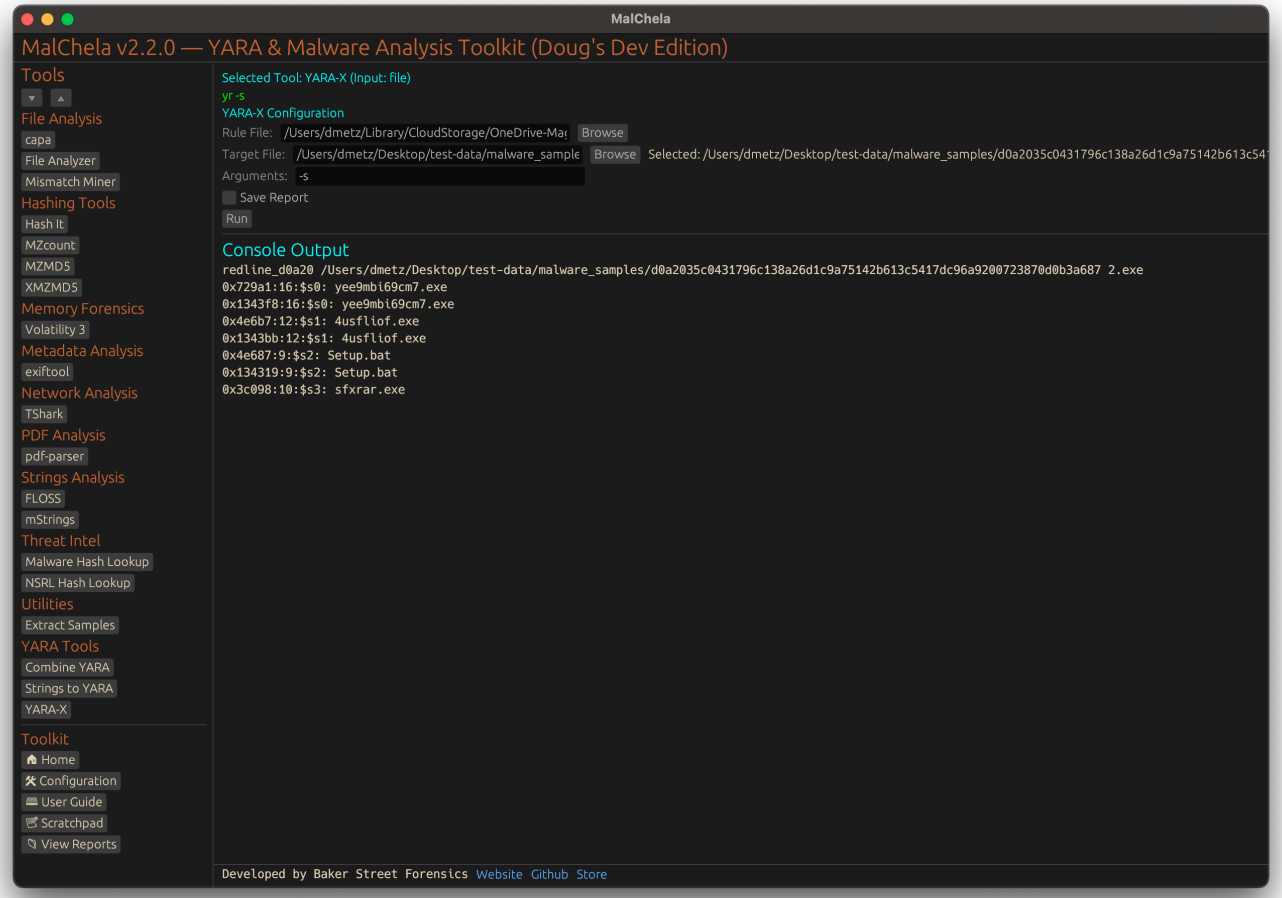

Figure 25: YARA-X

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

4.7.1 Installation

- **Download the latest release:**
  Visit the official YARA-X GitHub releases page at https://github.com/Yara-Rules/yara-x/releases and download the appropriate binary for your platform.
- **Extract and install:**
  Extract the downloaded archive and place the `yara-x` binary 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:

```
yara-x —version
```

4.7.2 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
```

4.7.3 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.

4.7.4 Tips

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

4.8 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

5. 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.

5.1 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.

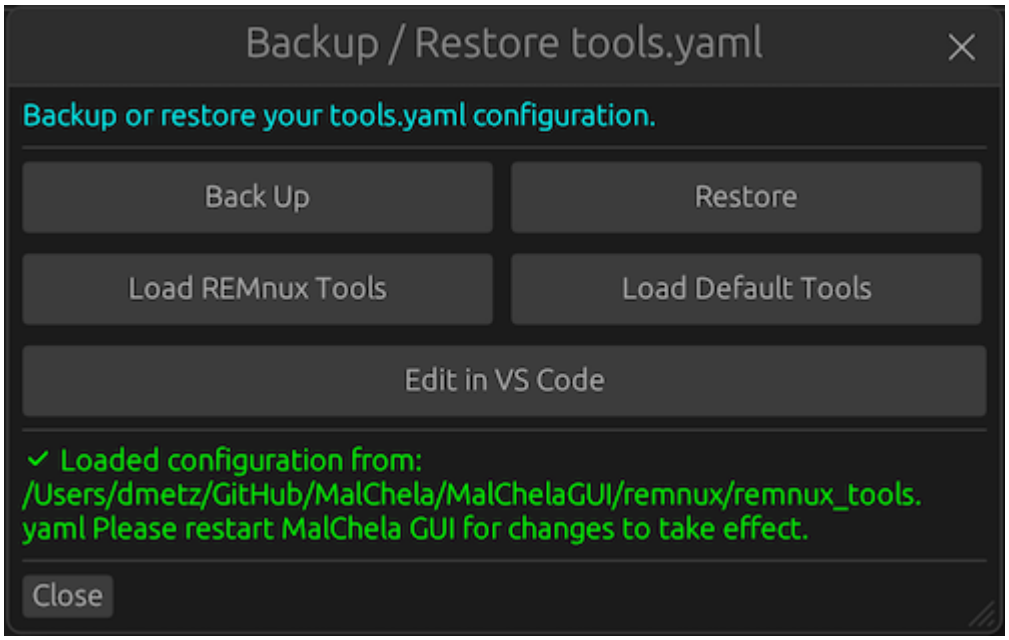

Figure 26: Enabling REMnux mode

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

5.2 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:

5.2.1 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`

5.2.2 Memory Forensics

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

5.2.3 Metadata Analysis

Tool	Description	Command
exiftool	Extract metadata from files	`exiftool`

5.2.4 Network Forensics

Tool	Description	Command
TShark	Analyze network traffic	`tshark`

5.2.5 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`

5.2.6 PDF Analysis

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

5.2.7 Utilities

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

5.3 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

5.4 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.

6. Support

6.1 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.

6.2 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.