MalChela User Guide

📘 This guide covers MalChela v2.1.2 (May 2025)

📚 Table of Contents

🦀 Introduction

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.

🛠️ Installation

Prerequisites

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

🔗 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-de
```

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.

Clone the Repository

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

cd MalChela
```

Build Tools

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

Windows Notes

```
• Best experience via WSL2
• GUI is not supported natively on Windows
```

🚀 Getting Started

MalChela supports three main workflows:

1. **Direct Tool Execution (CLI):**

```
cargo run -p toolname -- [input] [flags]
```

2. **MalChela CLI Launcher Menu:**

```
cargo run -p malchela
```

3. **MalChela GUI Launcher:**

```
cargo run -p MalChelaGUI
```

🔧 CLI Usage Notes

📄 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)

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

---

💻 GUI Walkthrough

✨ 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.

Layout

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

Running Tools

```
1.  Select a tool
2.  Fill in input fields
3.  Configure options (save report, format, etc.)
4.  Click Run
> - The GUI uses `exec_type` to determine whether a tool is launched using `cargo`, a native binary, or a script
> - Input file position is handled based on the `file_position` value in `tools.yaml`.
```

Save Report

```
• Formats: .txt, .json, .md
• Location: saved_output/<tool>/report_<timestamp>.<ext> (only one file is generated per run)
```

Scratchpad

```
• Save as .txt, .md, or .yaml
• Tip: hash: lines are ignored when used for strings_to_yara
```

Configuration Panel

```
• Stores API keys in vt-api.txt and mb-api.txt
• Keys are required for malhash, fileanalyzer (for VT)
• Quick-access button to edit `tools.yaml` from the GUI
```

📝 Scratchpad Tips (strings_to_yara)

- Any line starting with `hash:` is ignored when generating YARA rules
- Supports markdown and YAML save formats

- Integrated “Open in VS Code” button for saved notes

🧪 Tool Behavior Reference

Tool	Input Type	Supports -o	Prompts if Missing	Notes					
combine_yara	folder	❌	✅	Identifies mismatches	extract_samples	file	❌	✅	Extracts archive contents
fileanalyzer	file	✅	✅	Uses YARA + heuristics					
hashit	file	✅	✅	Generates hashes	malhash	hash	✅	✅	Uses vt-cli + bazaar-cli
mismatchminer	folder	✅	✅	Identifies mismatches					
mstrings	file	✅	✅	Maps strings to MITRE					
nsrlquery	file	✅	✅	Queries CIRCL	strings_to_yara	file	❌	✅	Generates YARA rules
mzmd5	folder	❌	✅	MD5 only; no output flag					
mzcount	folder	❌	✅	file counts					
strings_to_yara	text file and metadata	❌	✅	Combined yara rule					
xmzmd5	folder	❌	✅	Extended MD5 scan					

🧮 Tool-Specific Notes

```
• fileanalyzer: YARA rules for tools like `fileanalyzer` are stored in the `yara_rules` folder in the workspace.
• mstrings: Maps strings to MITRE ATT&CK from detections.yaml
• strings_to_yara: CLI/GUI dual support; hash: lines from **scratchpad** ignored
• malhash: Needs API keys to run
```

⚙️ Tool Configuration Mode (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.*

Swapping Configs: REMnux Mode and Beyond

MalChela supports easy switching between tool configurations via the GUI.

To switch:

1. Open the **Configuration Panel**
2. Use **"Select tools.yaml"** to point to a different config
3. 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.*

🔌 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.*

⚙️ Installing and Configuring YARA-X

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

Installation

1. **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.

1. **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.

1. **Verify installation:**

Run the following command to confirm YARA-X is installed correctly:

```
yara-x --version
```

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
```

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.

Tips

- If you want to use YARA-X as a drop-in replacement for the standard YARA engine, ensure your tool configurations point to the `yara-x` binary.
- For advanced usage, consult the YARA-X documentation for command-line options and rule syntax.

🧩 FLOSS Notes

- 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.

🐍 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

🦈 TShark Field Reference Panel

If TShark is included in your `tools.yaml` (or if you're using the REMnux configuration), the GUI provides an integrated reference panel for display filter fields.

- Launchable via the "?" icon next to filter fields
- Provides examples, tooltips, and a copy-to-clipboard feature
- Helps users construct and test display filters visually

```
#### ✅ 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`
```

⚠️ Known Limitations & WSL Notes

```
• CLI works in WSL
• GUI requires macOS or Linux (may work in WSLg on Win11)
• Paths must be POSIX-style
• If `exec_type` is omitted or misconfigured in `tools.yaml`, the GUI may attempt to run the tool incorrectly.
• GUI execution behavior no longer depends on the `category` field.
• FLOSS may print a warning such as `from multiprocessing.resource_tracker import main;main(6)` due to a known bu
```

🦀 Support & Contribution

```
• GitHub: https://github.com/dwmetz/MalChela
• Issues/PRs welcome
• Extend via tools.yaml for external tools
```

For more information, visit https://bakerstreetforensics.com.