# Zircolite documentation

## Usage



## Advanced use



## Internals



### Requirements and Installation

You can install dependencies with : `pip3 install -r requirements.txt`

The use of evtx_dump is **optionnal but required by default (because it is for now much faster)**, I you do not want to use it you have to use the '–noexternal' option. The tool is provided if you clone the Zircolite repository (the official repository is here).

**Known issues** Sometimes `evtx_dump` hangs under MS Windows, this is not related to Zircolite. If it happens to you, usually the use of `--noexternal` solves the problem.

If you can share the EVTX files on whose the blocking happened, feel free to post an issue in the evtx_dump repository.

---

### Basic usage

Help is available with `zircolite.py -h`.

**For EVTX files :** If your evtx files have the extension ".evtx" :

```
python3 zircolite.py --evtx <EVTX_FOLDER> --ruleset <Converted Sigma rules>
python3 zircolite.py --evtx ../Logs --ruleset rules/rules_windows_sysmon.json
```

It also works directly on an unique EVTX file.

By default :

- Results are written in the `detected_events.json` in the same directory as Zircolite
- There is a `zircolite.log`file that will be created in the current working directory

**JSONL/NDJSON** It is possible to use Zircolite directly on JSONL/NDJSON files (NXLog files) with the `--jsononly` or `-j` arguments :

```
python3 zircolite.py --evtx <EVTX_FOLDER> --ruleset <CONVERTED_SIGMA_RULES> --jsononly
```

A simple use case is when you have already run Zircolite and use the `--keeptmp` option. Since it keeps all the converted EVTX in a temp directory, if you need to re-execute Zircolite, you can do it directly using this directory as the EVTX source (with `--evtx <EVTX_IN_JSON_DIRECTORY>` and `--jsononly`) and avoid to convert the EVTX again.

/!\ If you you can change the file extension with `--fileext`.

**SQLite database files** Since everything in Zircolite is stored in a in-memory SQlite database, you can choose to save the database on disk for later use. It is possible with the option `--dbfile <db_filename>`.

```
python3 zircolite.py --evtx <EVTX_FOLDER> --ruleset <CONVERTED_SIGMA_RULES> --dbfile output.db
```

If you need to re-execute Zircolite, you can do it directly using the SQLite database as the EVTX source (with `--evtx <SAVED_SQLITE_DB_PATH>` and `--dbonly`) and avoid to convert the EVTX, post-process the EVTX and insert data to database. **Using this technique can save a lot of time...**

**Sysmon for Linux XML log files** Sysmon for linux has been released in October 2021. It outputs XML in text format with one event per-line. As of version 2.6.0, **Zircolite** has an *initial* support of Sysmon for Linux log files. To test it, just add `-S` to you command line :

```
python3 zircolite.py --evtx <EVTX_FOLDER> --ruleset <CONVERTED_SIGMA_RULES> -S
```

/!\ Since the logs come from Linux, the default file extension when using `-S` case is `.log`

---

**Generate your own rulesets**

Default rulesets are already provided in the `rules` directory. These rulesets only are the conversion of the rules located in rules/windows directory of the Sigma repository. These rulesets are provided to use Zircolite out-of-the-box but you should generate your own rulesets.

**With sigmatools** Zircolite use the SIGMA rules in JSON format. To generate your ruleset you need the official sigmatools (version 0.20 minimum) :

```
git clone https://github.com/SigmaHQ/sigma.git
cd sigma
```

**You must have the sigma dependencies installed, check here :**

[DEPRECATED] The pip version of sigmatools works but the backend has been updated since

**Sysmon rulesets (when investigated endpoints have Sysmon logs)**

```
tools/sigmac \
    -t sqlite \
    -c tools/config/generic/sysmon.yml \
    -c tools/config/generic/powershell.yml \
    -c tools/config/zircolite.yml \
    -d rules/windows/ \
   --output-fields title,id,description,author,tags,level,falsepositives,filename,status \
   --output-format json \
   -r \
   -o rules_sysmon.json \
   --backend-option table=logs
```

Where :

- `-t` is the backend type (SQlite)
- `-c` options are the backend configurations from the official repository
- `-r` option is used to convert an entire directory (don't forget to remove if it is a single rule conversion)
- `-o` option is used to provide the output filename
- `--backend-option` is used to specify the SQLite table name (leave as is)

**Generic rulesets (when investigated endpoints *don't* have Sysmon logs)**

```
tools/sigmac \
    -t sqlite \
    -c tools/config/generic/windows-audit.yml \
    -c tools/config/generic/powershell.yml \
    -c tools/config/zircolite.yml \
    -d rules/windows/ \
   --output-fields title,id,description,author,tags,level,falsepositives,filename,status \
   --output-format json \
   -r \
   -o rules_generic.json \
   --backend-option table=logs
```

**On the fly rules conversion** Since Zircolite 2.2.0, if you have sigmatools >= 0.20, Zircolite is able to convert the rules on-the-fly if you provide a SIGMA config file and the `sigmac` path. It is very convenient for testing but you should avoid it since this is slower :

```
python3 zircolite.py --evtx ../Samples/EVTX-ATTACK-SAMPLES/ \
                     --ruleset <DIRECTORY>/sigma/rules/windows/ \
                     --sigma <DIRECTORY>/sysmon.yml \
                     --sigmac <DIRECTORY>/sigmac
```

In this case, as some rules are not supported by the SIGMA SQL/SQLite backends, it is possible to show which rule was not converted with the `--sigmaerrors` option.

**Why you should build your own rulesets** The default rulesets provided are the conversion of the rules located in `rules/windows` directory of the Sigma repository. You should take into account that :

- **Some rules are very noisy or produce a lot of false positives** depending on your environment or the config file you used with genRules
- **Some rules can be very slow** depending on your logs

For example :

- "Suspicious Eventlog Clear or Configuration Using Wevtutil" : **very noisy** on fresh environment (labs etc.), commonly generate a lot of useless detections
- Notepad Making Network Connection : **can slow very significantly** the execution of Zircolite

---

### Generate embedded versions

If you deploy (manually or via GPO/SCCM) Zircolite directly on an endpoint you may want to have a binary that contains everything (rules, templates, tools, config etc.). As of 2.0, it is possible to generate your own embedded version of Zircolite with the **genEmbed** tool available in the repository tools directory

**Using genEmbed** Please check help in the **genEmbed** repository : tools/genEmbed.

---

### Docker

Zircolite is also packaged as a Docker image (cf. wagga40/zircolite on Docker Hub), which embeds all dependencies (e.g. `evtx_dump`) and provides a platform-independant way of using the tool.

### Build and run your own image

```
docker build . -t <Image name>
docker container run --tty --volume <EVTX folder>:/case <Image name> \
    --ruleset rules/rules_windows_sysmon.json \
    --evtx /case \
    --outfile /case/detected_events.json
```

This will recursively find EVTX files in the `/case` directory of the container (which is bound to the `/path/to/evtx` of the host filesystem) and write the detection events to the `/case/detected_events.json` (which finally corresponds to `/path/to/evtx/detected_events.json`).

Event if Zircolite does not alter the original EVTX files, sometimes you want to make sure that nothing will write to the original files. For these cases, you can use a read-only bind mount with the following command:

```
docker run --rm --tty -v <EVTX folder>:/case/input:ro -v <Results folder>:/case/output \
    <Zircolite Image name>
    --ruleset rules/rules_windows_sysmon.json \
    --evtx /case/input -o /case/output/detected_events.json
```

**Docker Hub** You can use the Docker image available on Docker Hub. Please note that in this case, the configuration files and rules are the default ones.

```
docker container run --tty --volume <EVTX folder>:/case docker.io/wagga40/zircolite:1.4.0 \
    --ruleset rules/rules_windows_sysmon.json \
    --evtx /case --outfile /case/detected_events.json
```

### Working with large datasets

Zircolite tries to be as fast as possible so a lot of data is stored in memory. So :

- **Zircolite memory use oscillate between 2 or 3 times the size of the logs**
- It is not a good idea to use it on very big EVTX files or a large number of EVTX **as is**

The tool has been created to be used on very big datasets and there are a lot of ways to speed up Zircolite :

- Using as much CPU core as possible : see below "Using GNU Parallel"
- Using Filtering

**Using GNU Parallel** Except when `evtx_dump` is used, Zircolite only use one core. So if you have a lot of EVTX files and their total size is big, it is recommanded that you use a script to launch multiple Zircolite instances. On Linux or MacOS The easiest way is to use **GNU Parallel**.

/!\ on MacOS, please use GNU find (`brew install find` will install `gfind`)

- **"DFIR Case mode" : One directory per computer/endpoint**

  This mode is very useful when you have a case where all your evidences is stored per computer (one directory per computer containing all EVTX for this computer). It will create one result file per computer in the current directory.

  ```
  find <CASE_DIRECTORY> -maxdepth 1 -mindepth 1 -type d | \
      parallel --bar python3 zircolite.py --evtx {}
      --ruleset rules/rules_windows_sysmon.json --outfile {/.}.json
  ```

  One downside of this mode is that if you have less computer evidences than CPU Cores, they all will not be used.

- **"WEF/WEC mode" : One zircolite instance per EVTX**

  You can use this mode when you have a lot of aggregated EVTX coming from multiple computers. It is generaly the case when you use WEF/WEC and you recover the EVTX files from the collector. This mode will create one result file per EVTX.

  ```
  find <CASE_DIRECTORY> -type f -name "*.| \
      parallel -j -1 --progress python3 zircolite.py --evtx {} \
      --ruleset rules/rules_windows_sysmon.json --outfile {/.}.json
  ```

  In this example the `-j -1` is for using all cores but one. You can adjust the number of used cores with this arguments.

**Using Zircolite MP** ***deprecated***

---

**Filtering**

Zircolite has a lot of filtering options to speed up the detection process. Don't overlook these options because they can save you a lot of time.

**File filters** Some EVTX files are not used by SIGMA rules but can become quite large (a good example is `Microsoft-Windows-SystemDataArchiver%4Diagnostic.evtx`), if you use Zircolite with a directory as input argument, all EVTX files will be converted, saved and matched against the SIGMA Rules.

To speed up the detection process, you may want to use Zircolite on files matching or not matching a specific pattern. For that you can use **filters** provided by the two command line arguments :

- `-s` or `--select` : select files partly matching the provided a string (case insensitive)
- `-a` or `--avoid` : exclude files partly matching the provided a string (case insensitive)

/!\ When using te two arguments, the "select" argument is always applied first and then the "avoid" argument is applied. So, it is possible to exclude files from included files but not the opposite.

- Only use EVTX files that contains "sysmon" in their names

  ```
  python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json --select sysmon
  ```

- Exclude "Microsoft-Windows-SystemDataArchiver%4Diagnostic.evtx"

  ```
  python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json \
      --avoid systemdataarchiver
  ```

- Only use EVTX files with "operational" in their names but exclude "defender" related logs

  ```
  python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json \
  --select operational --avoid defender
  ```

For example, the **Sysmon** ruleset available in the `rules` directory only use the following channels (names have been shortened) : *Sysmon, Security, System, Powershell, Defender, AppLocker, DriverFrameworks, Application, NTLM, DNS, MSexchange, WMI-activity, TaskScheduler.*

So if you use the sysmon ruleset with the following rules, it should speed up `Zircolite`execution :

```
python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json \
    --select sysmon --select security.evtx --select system.evtx \
    --select application.evtx --select Windows-NTLM --select DNS \
    --select powershell --select defender --select applocker \
    --select driverframeworks --select "msexchange management" \
    --select TaskScheduler --select WMI-activity
```

**Time filters** Sometimes you only want to work on a specific timerange to speed up analysis. With Zircolite, it is possible to filter on a specific timerange just by using the `--after` and `--before` and their respective shorter versions `-A` and `-B`. Please note that :

- The filter will apply on the `SystemTime` field of each event
- The `--after` and `--before` arguments can be used independently
- The timestamps provided must have the following format : YYYY-MM-DD**T**HH:MM:SS (hours are in 24h format)

Examples :

- Select all events between the 2021-06-02 22:40:00 and 2021-06-02 23:00:00 :

  ```
  python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json \
      -A 2021-06-02T22:40:00 -B 2021-06-02T23:00:00
  ```

- Select all events after the 2021-06-01 12:00:00 :

  ```
  python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json \
      -A 2021-06-01T12:00:00
  ```

**Rule filters** Some rules can be noisy or slow on specific datasets (check here) so it is possible to skip them by using the `-R` or `--rulefilter` argument. This argument can be used multiple times.

The filter will apply on the rule title. Since there is a CRC32 in the rule title it is easier to use it. For example, to skip execution of the rule "Suspicious Eventlog Clear or Configuration Using Wevtutil - BFFA7F72" :

```
python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json -R BFFA7F72
```

You can also specify a string, to avoid unexpected side-effect **comparison is case-sensitive**. For example, if you do not want to use all MSHTA related rules and skip the execution of the rule "Suspicious Eventlog Clear or Configuration Using Wevtutil - BFFA7F72":

```
python3 zircolite.py --evtx logs/ --ruleset rules/rules_windows_sysmon.json -R BFFA7F72 -R MSHTA
```

/**!**\ As of version 2.2.0 of Zircolite, since the rulesets are directly generated from the official `sigmac` tool there is no more CRC32 in the rule title. Rule filtering is still available but you have to rely on other criteria.

**Limit the number of detected events** Sometimes, SIGMA rules can be very noisy (and generate a lot of false positives) but you still want to keep them in your rulesets. It is possible to filter rules that returns too mich detected events with the option `--limit <MAX_NUMBER>`. Please note that when using this option, the rules are not skipped the results are just ignored. But this is useful when forwarding events to Splunk.

---

### Forwarding detected events

Zircolite provide 2 ways to forward events to a collector :

- the HTTP forwarder : this is a very simple forwarder and pretty much a "toy" example and should be used when you have nothing else. An **example** server called is available in the tools directory
- the Splunk HEC Forwarder : it allows to forward all detected events to a Splunk instance using **HTTP Event Collector**.

For now, the forwarders are not asynchronous so it can slow Zircolite execution. There are two modes to forward the events :

- By default all events are forwarded after the detection process
- The argument `--stream` allow to forward events during the detection process

If you forward your events to a central collector you can disable local logging with the Zircolite `--nolog` argument.

**Forward to a HTTP server** If you have multiple endpoints to scan, it is usefull to send the detected events to a central collector. As of v1.2, Zircolite can forward detected events to an HTTP server :

```
python3 zircolite.py --evtx sample.evtx  --ruleset rules/rules_windows_sysmon.json \
    --remote "http://address:port/uri"
```

An **example** server called is available in the tools directory.

**Forward to a Splunk instance via HEC** As of v1.3.5, Zircolite can forward detections to a Splunk instance with Splunk **HTTP Event Collector**.

1. Configure HEC on you Splunk instance : check here
2. Get your token and you are ready to go :

```
python3 zircolite.py --evtx /sample.evtx  --ruleset rules/rules_windows_sysmon.json \
    --remote "https://x.x.x.x:8088" --token "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

:warning: On Windows do not forget to put quotes

**No local logs** When you forward detected events to an server, sometimes you don't want any log file left on the system you have run Zircolite on. It is possible with the `--nolog` option.

---

### Templating and Formatting

Zircolite provides a templating system based on Jinja 2. It allows you to change the output format to suits your needs (Splunk or ELK integration, Grep-able output…). There are some templates available in the Templates directory of the repository : CSV, Splunk, Mini-GUI. To use the template system, use these arguments :

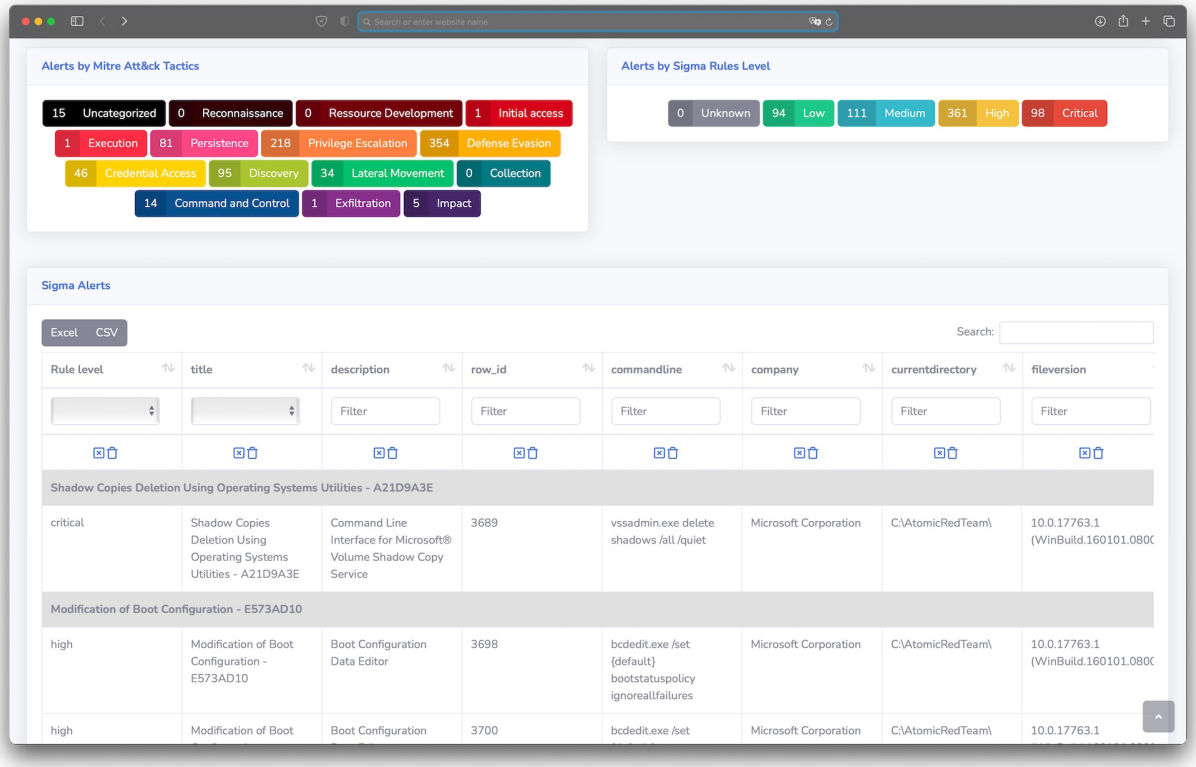
- `--template <template_filename>`
- `--templateOutput <output_filename>`

```
python3 zircolite.py --evtx sample.evtx  --ruleset rules/rules_windows_sysmon.json \
--template templates/exportCSV.tmpl --templateOutput test.csv
```

It is possible to use multiple templates if you provide for each `--template` argument there is a `--templateOutput` argument associated.

---

### Mini-GUI




The Mini-GUI can be used totaly offline, it allows the user to display and search results. It uses datatables and the SB Admin 2 theme.

**Automatic generation** As of Zircolite 2.1.0, with the non-embedded versions, the easier way to use the Mini-GUI is to generate a package with the `--package` option. A zip file containing all the necessary data will be generated at the root of the repository.

**Manual generation** You need to generate a `data.js` file with the `exportForZircoGui.tmpl` template, decompress the zircogui.zip file in the gui directory and replace the `data.js` file in it with yours :

```
python3 zircolite.py --evtx sample.evtx
    --ruleset rules/rules_windows_sysmon.json \
    --template templates/exportForZircoGui.tmpl --templateOutput data.js
7z x gui/zircogui.zip
mv data.js zircogui/
```

Then you just have to open `index.html` in your favorite browser and click on a Mitre Att&ck category or an alert level.

:warning: **The mini-GUI was not built to handle big datasets**.

---

### Packaging Zircolite

### PyInstaller

- Install Python 3.8 on the same OS as the one you want to use Zircolite on
- Install all dependencies : `pip3 install -r requirements.txt`

- After Python 3.8 install, you will need PyInstaller : `pip3 install pyinstaller`
- In the root folder of Zircolite type : `pyinstaller -c --onefile zircolite.py`
- The `dist` folder will contain the packaged app

**Nuitka**

- Install Python 3.8 on the same OS as the one you want to use Zircolite on
- Install all dependencies : `pip3 install -r requirements.txt`
- After Python 3.8 install, you will need Nuitka : `pip3 install nuitka`
- In the root folder of Zircolite type : `python3 -m nuitka --onefile zircolite.py`

:warning: When packaging with PyInstaller some AV may not like your package.

---

**Using With DFIR Orc**

**DFIR Orc** is a Forensics artefact collection tool for systems running Microsoft Windows (pretty much like Kape or artifactcollector). For more detailed explaination, please check here : https://dfir-orc.github.io.

ZikyHD has done a pretty good job at integrating **Zircolite** with **DFIR Orc** in this repository : https://github.com/Zircocorp/dfir-orc-config.

Basically, if you want to integrate Zircolite with **DFIR Orc** :

- Clone the DFIR Orc Config repository : `git clone https://github.com/Zircocorp/dfir-orc-config.git`
- Create a `DFIR-ORC_config.xml` (or add to an existing one) in the `config` directory containing :

```
<?xml version="1.0" encoding="utf-8"?>
<wolf childdebug="no" command_timeout="1200">
    <log disposition="truncate">DFIR-ORC_{SystemType}_{FullComputerName}_{TimeStamp}.log</log>
    <outline disposition="truncate">DFIR-ORC_{SystemType}_{FullComputerName}_{TimeStamp}.json
    </outline>

    <!-- BEGIN ZIRCOLITE SPECIFIC CONFIGURATION-->
    <!-- This part creates a specific archive for Zircolite -->
    <archive name="DFIR-ORC_{SystemType}_{FullComputerName}_Zircolite.7z"
             keyword="Zircolite" concurrency="1"
             repeat="Once"
             compression="fast"
             archive_timeout="120" >
        <restrictions ElapsedTimeLimit="480" />
        <command keyword="GetZircoliteSysmon" winver="6.2+">
            <execute name="zircolite_win10_nuitka.exe"
                     run="7z:#Tools|zircolite_win10_nuitka.exe"/>
            <input  name='rules_windows_generic.json'
                    source='res:#rules_windows_generic.json'
                    argument='-r {FileName}' />
            <input  name='fieldMappings.json'
                    source='res:#fieldMappings.json'
                    argument='-c {FileName}' />
            <argument> --cores 1 --noexternal -e C:\windows\System32\winevt\Logs</argument>
            <output  name="detected_events.json" source="File" argument="-o {FileName}" />
            <output  name="zircolite.log" source="File" argument="-l {FileName}" />
        </command>
    </archive>
    <!-- /END ZIRCOLITE SPECIFIC CONFIGURATION-->
```

```
</wolf>
```

/!\ Please note that if you add this configuration to an existing one, you only need to keep the part between `<!-- BEGIN ... -->` and `<!-- /END ... -->` blocks.

- Put your custom or default mapping file `zircolite_win10_nuitka.exe` (the default one is in the Zircolite repository `config` directory) `rules_windows_generic.json` (the default one is in the Zircolite repository `rules` directory) in the the `config` directory.
- Put **Zircolite** binary (in this example `zircolite_win10_nuitka.exe`) and **DFIR Orc** binaries (x86 and x64) in the the `tools` directory.
- Create a `DFIR-ORC_Embed.xml` (or add to an existing one) in the `config` directory containing :

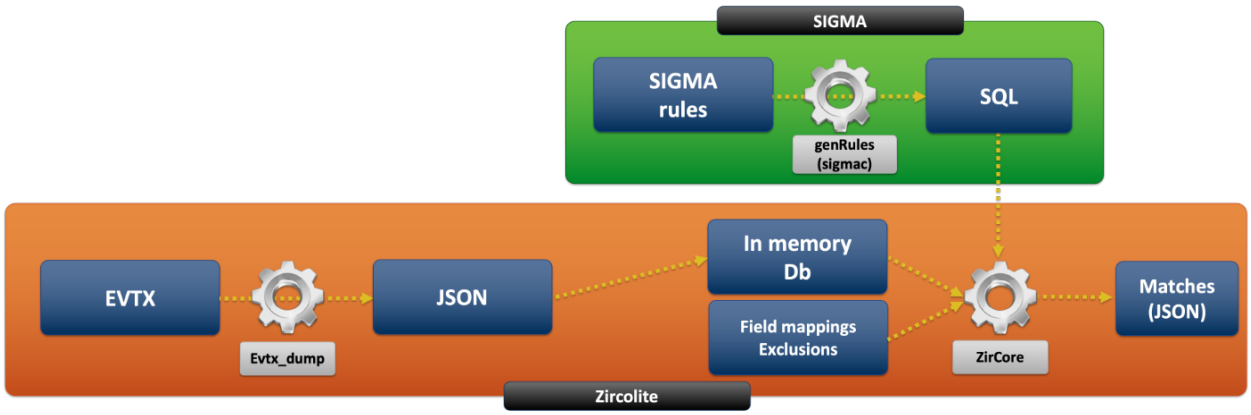
```
<?xml version="1.0" encoding="utf-8"?>
<toolembed>
    <input>.\tools\DFIR-Orc_x86.exe</input>
    <output>.\output\%ORC_OUTPUT%</output>

    <run64 args="WolfLauncher" >7z:#Tools|DFIR-Orc_x64.exe</run64>
    <run32 args="WolfLauncher" >self:#</run32>

    <file name="WOLFLAUNCHER_CONFIG"
          path=".\%ORC_CONFIG_FOLDER%\DFIR-ORC_config.xml"/>

    <!-- BEGIN ZIRCOLITE SPECIFIC CONFIGURATION-->
    <file name="rules_windows_generic.json"
          path=".\%ORC_CONFIG_FOLDER%\rules_windows_generic.json" />
    <file name="fieldMappings.json"
          path=".\%ORC_CONFIG_FOLDER%\fieldMappings.json" />
    <!-- /END ZIRCOLITE SPECIFIC CONFIGURATION-->

    <archive name="Tools" format="7z" compression="Ultra">
        <file name="DFIR-Orc_x64.exe"
              path=".\tools\DFIR-Orc_x64.exe"/>

        <!-- BEGIN ZIRCOLITE SPECIFIC CONFIGURATION-->
        <file name="zircolite_win10_nuitka.exe"
              path=".\tools\zircolite_win10_nuitka.exe"/>
        <!-- /END ZIRCOLITE SPECIFIC CONFIGURATION-->

    </archive>
</toolembed>
```

/!\ Please note that if you add this configuration to an existing one, you only need to keep the part between `<!-- BEGIN ... -->` and `<!-- /END ... -->` blocks.

- Now you need to generate the **DFIR Orc** binary by executing `.\configure.ps1` at the root of the repository
- The final output will be in the `output` directory

### Zircolite architecture

**Zircolite is more a workflow than a real detection engine**. To put it simply, it leverages the ability of the sigma converter to output rules in SQLite format. Zircolite simply applies SQLite-converted rules to EVTX stored in an in-memory SQLite DB.




---

### Project structure

```
Makefile                # Very basic Makefile
Readme.md               # Do I need to explain ?
bin                     # Directory containing all external binaries (evtx_dump)
config                  # Directory containing the config files
docs                    # Directory containing the documentation
pics                    # Pictures directory - not really relevant
rules                   # Sample rules you can use
templates               # Jinja2 templates
tools                   # Directory containing all tools (genRules, zircolite_server)
zircolite.py            # Zircolite !
```

---

### Benchmarks (Updated 22nd May 2021)

On an Intel Core-i9 8c/16t - 64 GB RAM - with **765 sigma rules** :

| | Monocore | Multicore |
|---|---|---|
| EVTX : 34 GB - 16 files | - | 9 Min |
| EVTX : 7.8 GB - 4 files | - | 162 sec |
| EVTX : 1.7 GB - 1 file | 99 sec | - |
| EVTX : 40 MB - 263 files | 3 sec | 1 sec |
| MORDOR Datasets - APT29 Day 1 (196 081 events) | 62 sec | - |
| MORDOR Datasets - APT29 Day 2 (587 286 events) | 4 min | - |
| MORDOR Datasets - APT3 Scenario 1 (101 904 events) | 70 sec | - |
| MORDOR Datasets - APT3 Scenario 2 (121 659 events) | 27 sec | - |

/!\ These results can be largely improved with fine-tuned rulesets and filtering.