

DynaSpyLinux User Manual

Written by: Khanh Le, Eva Shrestha, Youmin Zhou

1. Install the project

The project could be found from the attachments or cloned from this URL:

<https://github.com/lekq/DynaSpyLinux>

Please install python3 and all of its specified libraries in requirements.txt

2. Run the project

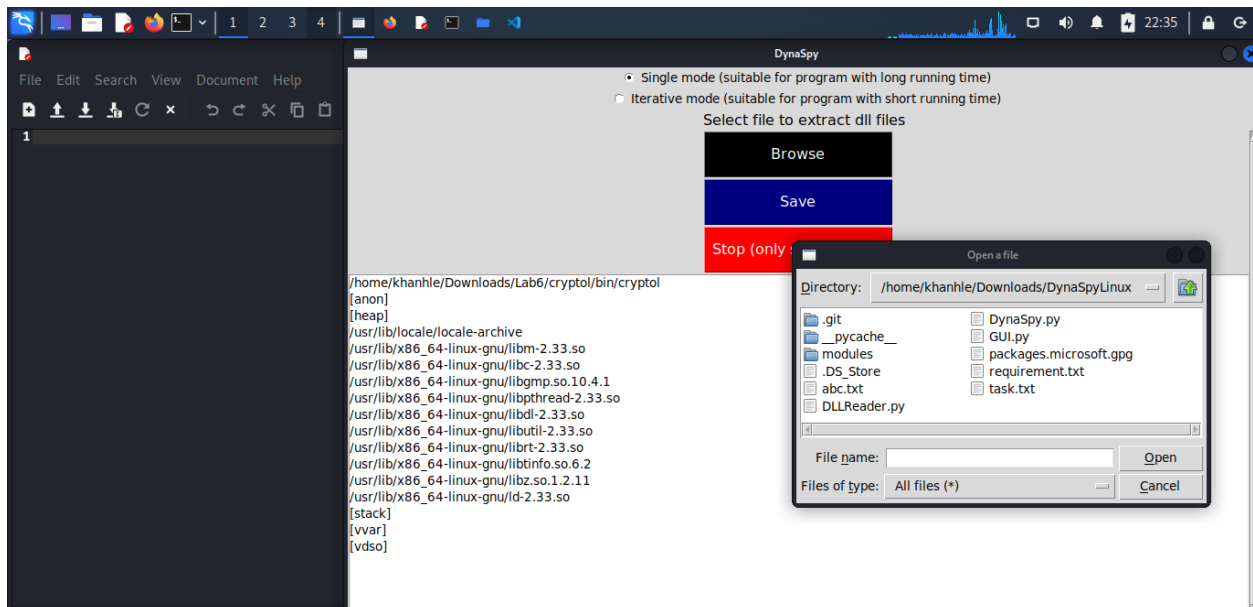
For our project, we provided two ways with which the user could interact: GUI mode and command line mode. With each mode, the user can pick either single reading mode or iterative reading mode.

NOTE: It is very important that the user can pick the most suitable reading mode for your respective program. Failing to do so will result in crash or nonstop analysis of DynaSpyLinux. The guideline is simple and included in section 3

a. GUI mode

Inside our main folder, simply run `python3 DynaSpy.py -gui`

Figure 1: The user interface of DynaSpyLinux



From the interface, the user can pick between two reading modes (single and iterative), **browse** to select the executable file, **save** the output to a txt file, or **stop** the analysis (this will be only applicable to single modes). The difference between single mode and iterative mode will be elaborated in section 3. **Reading modes.**

b. Command line mode

Inside our main folder, run `python3 DynaSpy.py [parameters]`. We have the following parameters:

-file <file_path>: the absolute path to your executable file. This is a **compulsory** parameter.

-output <file_path>: the absolute path to your output. This is an **optional** parameter. If specified, the program will write the result to the specified path. If not specified, it will print the result in the terminal.

-single / -iterative: the flag to specify with reading mode you want to use. Again, the difference between single mode and iterative mode will be elaborated in section **3. Reading modes**.

-threshold_to_break <integer>: the number of maximum iteration where the dll/so map can remain the same. That means if your analyzed executable file does not load any new library after this number of analyzing iterations, DynaSpyLinux will stop analyzing. If not specified, the default value is infinite. This parameter is only applicable to single mode. The reason why this parameter is extremely important will be elaborated in section **3. Reading modes**.

Below are some examples of acceptable commands:

```
python3 DynaSpy.py -file /home/khanhle/Downloads/temp
```

```
python3 DynaSpy.py -file /home/khanhle/Downloads/temp -output  
/home/khanhle/Downloads/result.txt
```

```
python3 DynaSpy.py -file /home/khanhle/Downloads/temp -output  
/home/khanhle/Downloads/result.txt -iterative
```

```
python3 DynaSpy.py -file /home/khanhle/Downloads/temp -output  
/home/khanhle/Downloads/result.txt -single -threshold_to_break 1000
```

3. Reading modes

a. Quick guideline

For program which has finite and short running time such as **a HelloWorld program or a script that performs basic calculations**, **iterative mode** is strongly suggested. Single mode can also be used but proves to be less consistent and accurate than iterative mode.

For a program which has long running time, infinite running time, or required interactions such as Linux video games or other applications, **single mode** must be used. When command line is used to run DynaSpyLinux, `threshold_to_break` must be set. The recommended number is in range of [1000..10000]

b. Single reading mode explanation

Our technique to read the loaded library of a process is very simple: with the assistance of `psutil` and `subprocess` library, we can ping a process, read the information from `.memory_maps`, and

know which library had been loaded at that time. The only issue is that we have no mechanism to control when that ping happens: sometimes, the ping happens too early such as right at the beginning of the analyzed program when not every library is loaded yet. That means DynaSpyLinux might miss some important information. Sometimes, the ping might happen at the end of the analyzed program when the process is no longer there and we have nothing else to read.

Therefore, our key target is that no matter how long the running time of a program is, we have to break it into as many checkpoints as possible to get an accurate result. Here is the pseudo code of how we define a single reading mode.

```
my_process = create_process (executable_file_you_want_to_test)
my_process.suspend ()
while my_process is still alive:
    # this is what we call a checkpoint
    my_process.resume ()
    new_memory_map = my_process.memory_maps
    update_memory_map (current_memory_map, new_memory_map)
    my_process.suspend ()
```

In other words, we force the analyzed program to execute only small step to get a more accurate result. This reading mode works very well for programs with from average to infinite running time. It is called single reading mode because it only runs the target file once.

c. Iterative reading mode explanation

The single mode did not work very well with program whose running time is extremely little such as a HelloWorld program. In fact, the analyzed results were different in each run. To increase the accuracy, we simply repeat the single reading mode 10 times and pick the best result among 10. This approach is called iterative reading mode. Because the tedious single reading mode has to be repeated 10 times, we should not use this for long and complicated programs.

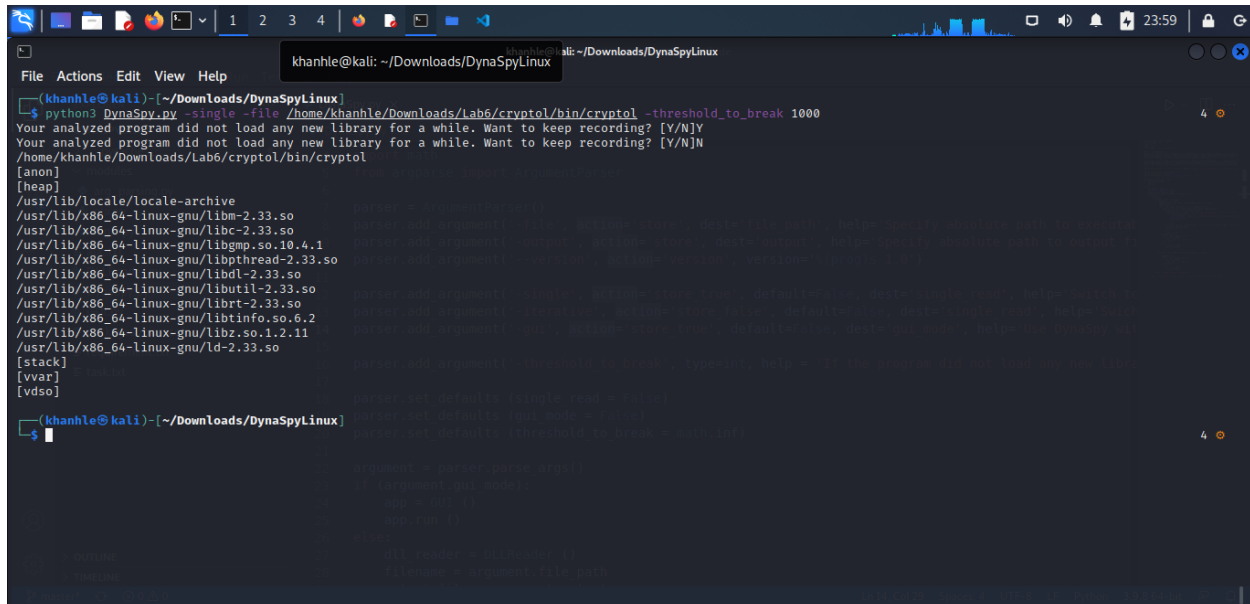
d. Threshold to break explanation

Because single mode is used for time-consuming program, we have to find a way to print out the output as long as the analyzed program is running (in contrast to iterative mode where we gather all results at the end and pick the best one).

With GUI, the output can be formatted nicely: with every checkpoint, we can clean the previous output and insert the new one. With Linux command line, there is no way to clean the output.

Because we have many checkpoints but don't want to print all results out, we set up a threshold. Let's say the threshold = 1000. That means if during the whole analysis we detect that the dll/so map does not change between the most recent 1000 checkpoints, we give the user the choice to either carry on 1000 more checkpoints or stop and receive the output.

Figure 2: Example of `threshold_to_break` usage



```
khanhle@kali: ~/Downloads/DynaSpyLinux
(khanhle@kali)~[~/Downloads/DynaSpyLinux]
$ python3 DynaSpy.py -single -file /home/khanhle/Downloads/Lab6/cryptol/bin/cryptol -threshold_to_break 1000
Your analyzed program did not load any new library for a while. Want to keep recording? [Y/N]Y
Your analyzed program did not load any new library for a while. Want to keep recording? [Y/N]N
/home/khanhle/Downloads/Lab6/cryptol/bin/cryptol
[anon]
[heap]
/usr/lib/locale/locale-archive
/usr/lib/x86_64-linux-gnu/libm-2.33.so
/usr/lib/x86_64-linux-gnu/libc-2.33.so
/usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1
/usr/lib/x86_64-linux-gnu/libpthread-2.33.so
/usr/lib/x86_64-linux-gnu/libdl-2.33.so
/usr/lib/x86_64-linux-gnu/libutil-2.33.so
/usr/lib/x86_64-linux-gnu/librt-2.33.so
/usr/lib/x86_64-linux-gnu/libtinfo.so.6.2
/usr/lib/x86_64-linux-gnu/libz.so.1.2.11
/usr/lib/x86_64-linux-gnu/ld-2.33.so
[stack]
[vvar]
[vdso]
(khanhle@kali)~[~/Downloads/DynaSpyLinux]
$
```

4. Sample results

Figure 3: DynaSpyLinux (GUI + iterative) on HelloWorld program

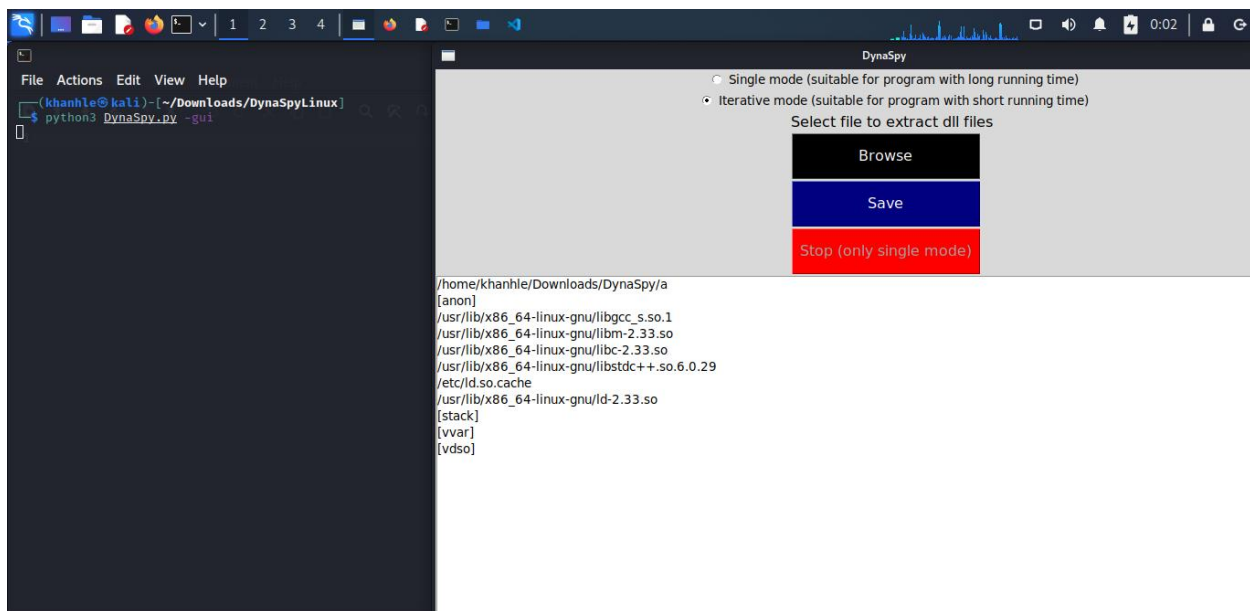


Figure 4: DynaSpyLinux (GUI + single) on gnome-calculator

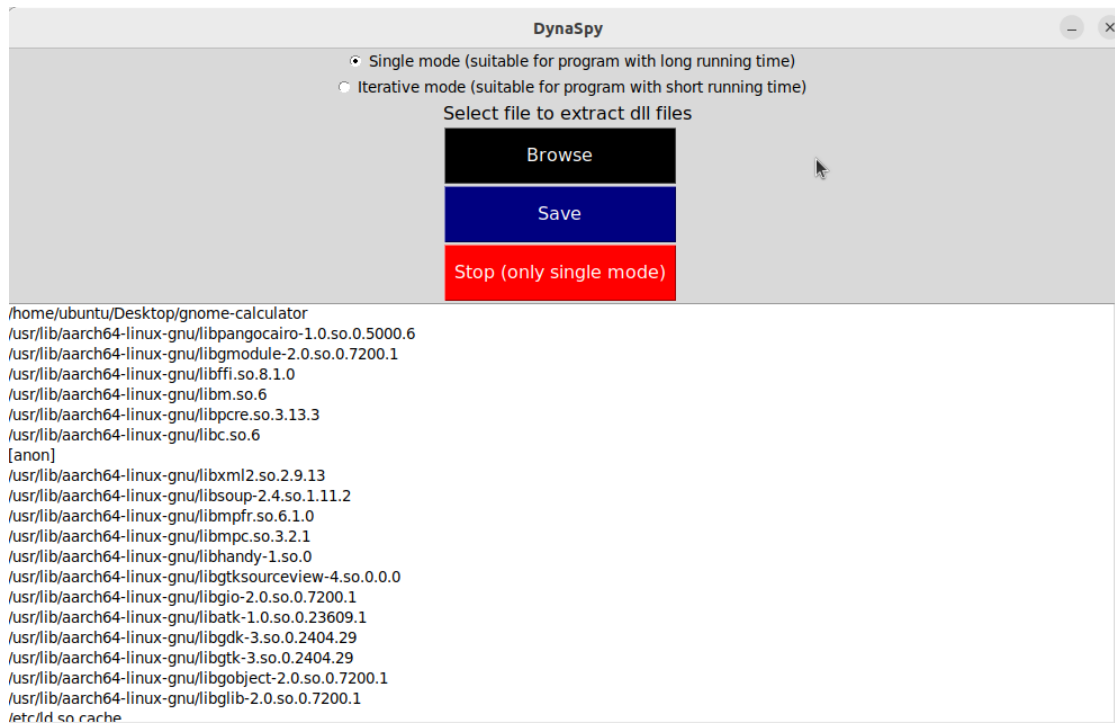


Figure 5: DynaSpyLinux (commandline + single) on gnome-calculator

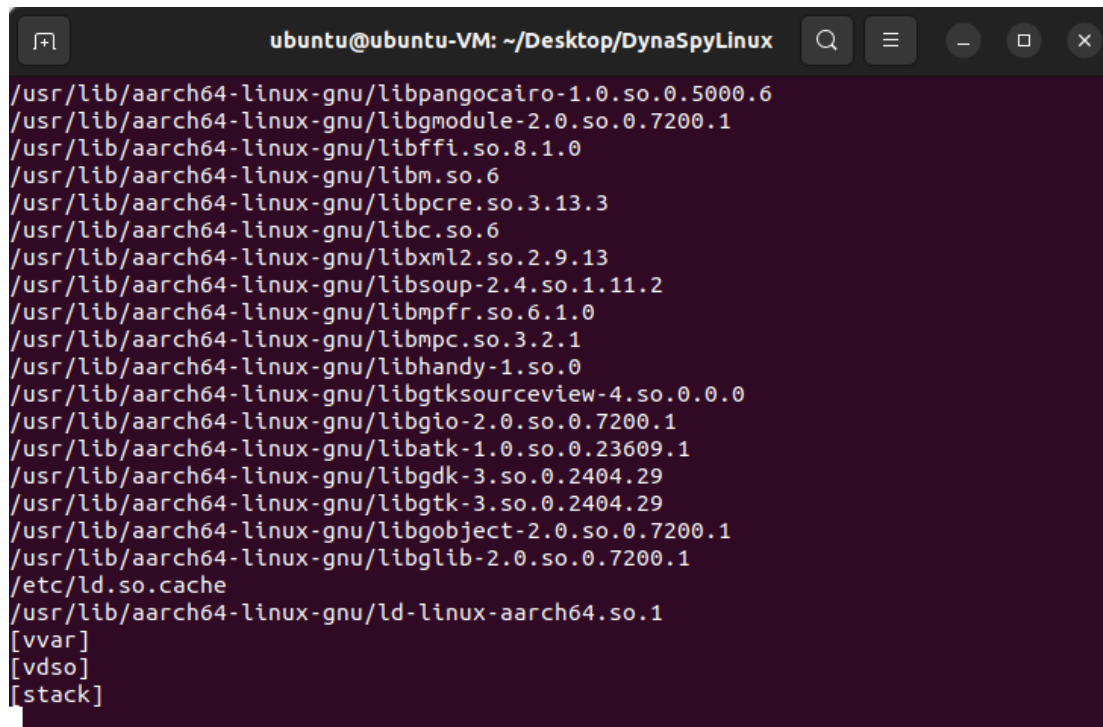


Figure 6: DynaSpyLinux (GUI + single) on dp game

