Fingerprint Documentation

Release 0.3

Luca Clementi and Phil Papadopoulos

March 06, 2015

Contents

FingerPrint is a software tool which can analyze arbitrary lists of binaries and save all their dependencies information in a file (called Swirl) along with other information.

A Swirl can then be used to understand if the given application can run on another system or if some of the dependencies got modified since the Swirl creation. Swirl can also be used to deploy the traced application on a Rocks cluster.

Contents:

CHAPTER 1

Userguide

If you want to use Fingerprint you should start from this userguide.

1.1 Requirements

FingerPrint will work only on a Linux system, it does not have any major requirement other than Python from version 2.4 up to 2.7. FingerPrint is currently tested on RHEL (5.x and 6.x) and (Debian 5.x and 6.x) systems.

It also requires a minimal set of core utilities (bash, sed, grep, ldd, and objdump) but all these tools are generally present on most of the systems.

If found on the system (they are not required), fingerprint uses:

- prelink (to remove pre-linking information from libraries and get their hash)
- dpkg or rpm (to record package version and info regarding dependencies)

FingerPrint comes with a stack tracing facility that can be used to determine which shared library opens a file. The stack tracing module is not required for the proper functioning. To compile the module you will need libunwind shared libraries (version 0.99 comes with libunwind-ptrace compiled statically so it does not work :-(). The stack tracing facility is written in C, so it requires gcc.

1.2 Installation

The simplest way to use FingerPrint is to checkout the source code

```
# git clone https://github.com/rocksclusters/FingerPrint.git
```

and then add to your `PATH` the `./bin` directory of the source code

```
# cd FingerPrint
# export PATH=$PATH:$PWD/bin
```

After this steps you can start to use fingerprint. The following steps are only required for advanced users. To invoke unit-tests run:

```
# python setup.py test
```

Unit-tests generate a lot of outputs and errors but if they all succeed at the end you will see the following lines:

```
Ran 4 tests in 38.870s

OK
```

If you want to install FingerPrint on your system python path you can follow the standard distutils procedure. If you want the stack tracing functionality copy the file `setup.cfg.template` into `setup.cfg` and insert the paths to your libunwind before proceeding. To build and install FingerPrint type:

```
# python setup.py build
# python setup.py install
```

This installs FingerPrint in your Python environment. You might need writing privilege on system directories for such installation.

The installation will deploy:

- a bunch of python source files inside the `FingerPrint` python module
- a command line python script called fingerprint, inside one of your `PATH` directories

1.3 Use

To get some help on the command line you can type:

```
# fingerprint -h
```

Basically there are four main actions fingerprint can do (-c create, -d display, -q query, and -y verify):

1. Create a swirl from a set of input file (flag -c) or with dynamic tracing. In this mode fingerprint will scan the list of files passed on the command line or it will (-x) trace the execution of the command specified to output a swirl file containing the dependencies fingerprint of the given input. This mode can also create a "swirl archive" (-r) which is nothing else than a tar.gz containing the swirl and all the file referenced by it. Using the create flag it is also possible to create a Rocks Cluster roll (flag -m), which will install the software described in the given "swirl archive" on all the nodes of a rocks cluster.
2. Display the content of a swirl file (flag -d). In this mode fingerprint will print to stdout a detailed description of the input swirl. The input swirl can be specified with -f, or it will be the default output.swirl.
3. Query the content of a swirl file (flag -q). In this mode fingerprint will run a query against the specified swirl file and return 0 upon success or 1 when failing. If the query is run with the verbose flag (-v) it will also print to stdout more information regarding the query.
4. Verify a swirl (flag -y). In this mode fingerprint scan the current system for the dependencies listed in the input swirl and return 0 if they were all found or 1 if some of then are unavailable. If verbose flag is given it will print also a list of unmet dependencies. Above the verify it is also possible to perform an integrity check. In this mode fingerprint scans the system where invoked and checks if any of the dependencies listed in the input swirl have been modified since its creation (to this purpose it uses the checksums stored in the swirl). It return 0 upon success or 1 in case of failure, with the verbose flag it prints also a list of modified files.

1.4 Examples

Create a fingerprint of your ls command:

```
clem@sirius:~/projects/FingerPrint/temp$ fingerprint -c /bin/ls
File output.swirl saved
```

By default it uses output.swirl as input or output Siwrl file name but you can choose your own file name with "-f"

```
clem@sirius:~/projects/FingerPrint$ ls -lh output.swirl
-rw-rw-r-- 1 clem clem 2.4K Feb 20 15:51 output.swirl
```

To see the list of libraries your /bin/ls depends on along with the local package name (this is what is stored in a swirl). You can always use the verbose flag (-v) to create more output.

```
clem@hermes:~/projects/FingerPrint$ fingerprint -dv
File name:  output.swirl
Swirl 2013-08-23 17:27
 ls.so.conf path list:
  /lib/i386-linux-gnu
  /usr/lib/i386-linux-gnu
  /usr/local/lib
  /lib/x86_64-linux-gnu
  /usr/lib/x86_64-linux-gnu
  /usr/lib/x86_64-linux-gnu/mesa
  /lib32
  /usr/lib32
 -- File List --
  /bin/ls  - coreutils 8.13-3ubuntu3.2 amd64
    Deps: librt.so.1, ld-linux-x86-64.so.2, libselinux.so.1, libacl.so.1, libc.so.6
    Provs:
    /lib/x86_64-linux-gnu/ld-2.15.so  - libc6 2.15-0ubuntu10.4 amd64
    -> /lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      Deps:
      Provs: ld-linux-x86-64.so.2
    /lib/x86_64-linux-gnu/libacl.so.1.1.0  - libacl1 2.2.51-5ubuntu1 amd64
    -> /lib/x86_64-linux-gnu/libacl.so.1
      Deps: libattr.so.1, libc.so.6
      Provs: libacl.so.1
    /lib/x86_64-linux-gnu/libc-2.15.so  - libc6 2.15-0ubuntu10.4 amd64
    -> /lib/x86_64-linux-gnu/libc.so.6
      Deps: ld-linux-x86-64.so.2
      Provs: libc.so.6
    /lib/x86_64-linux-gnu/librt-2.15.so  - libc6 2.15-0ubuntu10.4 amd64
    -> /lib/x86_64-linux-gnu/librt.so.1
      Deps: libpthread.so.0, libc.so.6
      Provs: librt.so.1
    /lib/x86_64-linux-gnu/libselinux.so.1  - libselinux1 2.1.0-4.1ubuntu1 amd64
      Deps: ld-linux-x86-64.so.2, libc.so.6, libdl.so.2
      Provs: libselinux.so.1
    /lib/x86_64-linux-gnu/libattr.so.1.1.0  - libattr1 1:2.4.46-5ubuntu1 amd64
    -> /lib/x86_64-linux-gnu/libattr.so.1
      Deps: libc.so.6
      Provs: libattr.so.1
    /lib/x86_64-linux-gnu/libpthread-2.15.so  - libc6 2.15-0ubuntu10.4 amd64
    -> /lib/x86_64-linux-gnu/libpthread.so.0
      Deps: ld-linux-x86-64.so.2, libc.so.6
      Provs: libpthread.so.0
    /lib/x86_64-linux-gnu/libdl-2.15.so  - libc6 2.15-0ubuntu10.4 amd64
    -> /lib/x86_64-linux-gnu/libdl.so.2
      Deps: ld-linux-x86-64.so.2, libc.so.6
      Provs: libdl.so.2
```

Scan the current system to verify compatibility with given swirl i.e. all dependencies listed in the Swirl can be found:

```
clem@sirius:~/projects/FingerPrint$ fingerprint -y
```

Verify that none of the dependencies have been modified (it uses md5sum to check for changes).

```
clem@sirius:~/projects/FingerPrint$ fingerprint -yi
```

You can query the swirl:

```
clem@sirius:~/projects/FingerPrint$ fingerprint -q -S
/lib/x86_64-linux-gnu/librt.so.1 && echo librt is used
librt is used

clem@sirius:~/projects/FingerPrint$ fingerprint -q -v -S
/lib/x86_64-linux-gnu/libcrypt.so.1 || echo libcrypt is not used
libcrypt is not used
```

1.5 Dynamic tracing

FingerPrint can dynamically trace a running process to properly detect dynamic dependencies and opened files. To this extent it uses the POSIX ptrace system call and it can trace spawned processes as well.

Dynamic tracing can trace dynamically loaded shared libraries and opened files. If FingerPrint is compiled with stacktracer support (see Requirements for more info) it can also detect which shared library initiated the open syscall. To dynamically trace a program run FingperPrint with the '-c -x' flags:

```
clem@hermes:~/projects/FingerPrint$ fingerprint -c -x "xeyes"
Tracing terminated successfully
File output.swirl saved
```

When displaying a Swirl created with the dynamic tracing it includes information regarding open files and dynamically loaded libraries.

```
clem@hermes:~/projects/FingerPrint$ fingerprint -d
File name:  output.swirl
Swirl 2013-08-23 17:43
 -- File List --
  /usr/bin/xeyes
    /lib/x86_64-linux-gnu/ld-2.15.so
    /lib/x86_64-linux-gnu/libc-2.15.so
      Opened files:
        /proc/meminfo
        /usr/lib/locale/locale-archive
    /lib/x86_64-linux-gnu/libm-2.15.so
    /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0
      Opened files:
        /usr/share/X11/locale/C/XLC_LOCALE
        /usr/share/X11/locale/locale.dir
        /usr/share/X11/locale/locale.alias
        /usr/share/X11/locale/en_US.UTF-8/XLC_LOCALE
    /usr/lib/x86_64-linux-gnu/libXext.so.6.4.0
    /usr/lib/x86_64-linux-gnu/libXmu.so.6.2.0
    /usr/lib/x86_64-linux-gnu/libXrender.so.1.3.0
    /usr/lib/x86_64-linux-gnu/libXt.so.6.0.0
    /lib/x86_64-linux-gnu/libdl-2.15.so
    /usr/lib/x86_64-linux-gnu/libxcb.so.1.1.0
    /usr/lib/x86_64-linux-gnu/libICE.so.6.3.0
    /usr/lib/x86_64-linux-gnu/libSM.so.6.0.1
```

```
/usr/lib/x86_64-linux-gnu/libXau.so.6.0.0
  Opened files:
    /home/clem/.Xauthority
/usr/lib/x86_64-linux-gnu/libXdmcp.so.6.0.0
/lib/x86_64-linux-gnu/libuuid.so.1.3.0
/usr/lib/x86_64-linux-gnu/libXcursor.so.1.0.2 --(Dyn)--
/usr/lib/x86_64-linux-gnu/libXfixes.so.3.1.0 --(Dyn)--
```

It the example above, thanks to the stack tracing facility, it is possible to see that the file `/home/clem/.Xauthority` was opened by the `/usr/lib/x86_64-linux-gnu/libXau.so.6.0.0` shared library.

CHAPTER 2

Hacking

This is a hacking guide intended for developer and not for final users.

2.1 Built system

Fingerprint uses distutils to built so the standard task can be used to package, built, and install the software:

```
./setup.py install
```

To install it on the local machine (although Fingerprint can be installed simply by setting the PATH, see the user guide for this):

```
./setup.py sdist
```

To create a source package (which is the one used in the Fingerprint Roll):

```
./setup.py upload
```

To upload Fingerprint on the PIP. TODO add some info on how to do this.

I have created an extra command to run the unit test contained in the tests folder (currently only 1 file contains unit tests tests/blotter_tests.py) which can be invoked with:

```
./setup.py test
```

2.2 Stack tracing functionality

If you wanna built Fingerprint with the stack tracing functionality (which is not required for its proper functioning) you need:

- gcc to compile the c code
- libunwind (minimum required version is 1.0, only Fedora 20 and Ubuntu 14 have the proper library version), make sure to have the libunwind-devel package if you are using distro packages.

To enable the compilation of the stack tracing functionality copy the file setup.cfg.template into setup.cfg and insert the paths to your libunwind then follow the standard procedure:

```
./setup.py install
```

2.3 Batlab continuous testing

The folder *batlab* contains all the file necessary to run the unit tests on batlab at every commits. To enable that you need to request an account on batlab login to you account, checkout the source code from git (checkout the repo in read only mode) and the configure a cron job which invoke the script inside batlab/crontab.sh to run as often as you want him to check the source for new commits.

This is to run it every hour:

```
0 1-23/2 * * * ~/FingerPrint/FingerPrint/batlab/crontab.sh
```

2.4 Source code structure

The main executable is in bin/fingerprint and it takes care of simply parsing argument and calling the various component of the FingerPrint package. Below a list of the various sub modules inside Fingerprint with a short description of what is their role:

- `FingerPrint.swirl`: it contains the data model. All the object used to represent a swirl are inside this files. The main class here is `FingerPrint.swirl.Swirl` (used to represent a swirl), it holds references to a list of `FingerPrint.swirl.SwirlFile`. `FingerPrint.swirl.SwirlFile` is a class used to keep all the info relative to every single file traced inside a Swirl. `FingerPrint.swirl.Dependency` is used to represent _static_ dependencies between SwirlFiles. Swirl contains the main methods responsible for finding SwirlFile Creating new SwirlFile, finding Dependencies of SwirlFile etc.
- `FingerPrint.sergeant`: it reads an already created swirl and it can perform several checking against the current system or display the swirl content. It can also be used to create a dot file to be used with graphviz. Basically all the display (-d) query (-q) and verify (-y) options are implemented here
- `FingerPrint.blotter`: This module contains only one class which is responsible to creates a swirl file starting from, a list of binaries, a command lines that we want to execute and trace, or a PID. It uses the plugin manager (FingerPrint.plugin) to analize each file. When running a dynamic tracing it uses the (FingerPrint.syscalltracer) module to do the ptracing work.
- `FingerPrint.plugins`: it is a plugable architecture which should support different file types (at the moment only an elf is implemented `FingerPrint.plugins.elf.ElfPlugin`). Each plugin should subclass the class `FingerPrint.plugins.PluginManager` and implement two methods `FingerPrint.plugins.PluginManager.getSwirl()` given a file path create a SwirlFile and add it to the swirl and return it. If the file is already in the swirl return it (do not duplicate it). `FingerPrint.plugins.PluginManager.getPathToLibrary()` should return a path to a shared library on the system given a dependency. At the moment it tries to imitate the Linux dynamic loader.
- `FingerPrint.syscalltracer`: is in charge of ptracing a command line and if available use the strac tracing functionality
- `FingerPrint.ptrace`: a bunch of classes taken from python-ptrace used to wrap ptrace system call, they are used only by syscalltracer for dynamic tracing
- `FingerPrint.composer`: is a module which takes care of composing a roll and of creating a Swirl archive. It has two classes `FingerPrint.composer.Archiver`, which is used to create archive (-r flag), and `FingerPrint.composer.Roller` which supports composing Rolls (-m flag).
- `FingerPrint.utils`: some simple general function which are used all over. Functions to fork external program and get their output, functions to get system `LD_LIBRARY_PATH` paths etc.
- `FingerPrint.serializer`: it contains only one class `FingerPrint.serializer.PickleSerializer` which is in charge of serializing and deserializing a swirl into a file. All the other module uses this class to read and write a Swirl. To make a XML serializer it is necessary to modify only this class

- remapper: this directory contains the source code for the remapper remapper is the process which is used when porting application using the -z flag. It is in charge of remapping all the open system call using the configuration file `/etc/fp_mapping`

CHAPTER 3

API Reference

3.1 FingerPrint internal API

3.1.1 `blotter` Module

class `FingerPrint.blotter.`**`Blotter`**(*name*, *fileList*, *processIDs*, *execCmd*)

This class creates a swirl file starting from:

- a list of binaries
- command lines that we want to execute and trace
- a list of pids

Parameters

- **name** (*string*) – a internal simbolic name for this swirl
- **fileList** (*list*) – a list of string containing aboslute or relative paths to the file that should be included in this Swirl for **static** analysis
- **processIDs** (*string*) – a list of comma separated PID which should be dynamically traced by this swirl
- **execCmd** (*string*) – a command line which should be launched and dynamically traced to create a swirl.

`getSwirl`()

return the current swirl

Return type `FingerPrint.swirl.Swirl`

Returns return the current Swirl

3.1.2 `composer` Module

class `FingerPrint.composer.`**`Archiver`**(*sergeant*, *archive_filename*)

Given an already created swirl it creates a Swirl archive

Parameters

- **sergeant** (`FingerPrint.sergeant.Sergeant`) – An instance of sergenat class pointing to the swirl we want to archive

- **archive_filename** (*string*) – string containing the output file name for the archive

archive()
It triggers the creation of the archive.

Return type bool

Returns it returns false in case of failure

class FingerPrint.composer.**Roller**(*archive_filename*, *roll_name*)
this class make a roll out of an fingerprint archive

Parameters

- **archive_filename** (*string*) – a path to the Swirl Archive file
- **roll_name** (*string*) – the name of the roll that we want to create

make_roll(*fingerprint_base_path*, *use_remapping=False*)
It creates a roll from a swirl archive.

Parameters

- **fingerprint_base_path** (*string*) – a string pointing to the base path of the fingerprint source code. Used to find the remapper source code
- **use_remapping** (*bool*) – if True it will use the remapper technology when creating the roll

Return type bool

Returns it returns false in case of failure

FingerPrint.composer.**is_special_file**(*path*)

Parameters **path** (*string*) – a path to a file

Return type bool

Returns it returns true if the path points to a file which contains personal data

FingerPrint.composer.**make_mapping_file**(*sw_files*, *output_file*, *base_path*)
this function makes a mapping file for the remapper process

3.1.3 sergeant Module

class FingerPrint.sergeant.**Sergeant**(*swirl*, *extraPath=[]*)

Given an already existent Swirl:

- it detects if it can run on this system (check())
- it detects what has been changed (checkHash())
- print this swirl on the screen (print_swirl())
- print this swirl as a dot file for Graphviz (getDotFile())

Parameters

- **swirl** (FingerPrint.swirl.Swirl) – The Swirl that we want to test
- **extraPath** (*list*) – a list of string containing system path which should be included in the search of dependencies

check()
It performs the check on the system and verifies that all the dependencies of this Swirl can be satisfied.

Return type bool

Returns True if the check passes False otherwise. The list of missing dependencies can be retrieved with `getError()`

checkDependencyPath(*fileName*)
it returns a list of SwirlFiles which requires the given fileName, if the given file is not required in this swirl it returns an empty list []

Parameters **fileName** (*string*) – a path to a file

Return type list

Returns a list of `FingerPrint.swirl.SwirlFile` required by the fileName

checkHash(*verbose=False*)
It checks if any dependency was modified since the swirl file creation (using checksumming)

Parameters **verbose** (*bool*) – if True it will generate more verbose error message

Return type bool

Returns True if the check passes False otherwise. The list of modified dependencies can be retrieved with :meth:getError()

getDotFile()
return a dot representation of this swirl

Return type string

Returns a string with the dot representation of this swirl

getError()
After running check or checkHash it return a list of the problems found

Return type list

Returns a lit of strings with all the problems encountered

getSwirl()
return the current swirl

Return type `FingerPrint.swirl.Swirl`

Returns the current swirl

print_swirl(*verbosity*)
return a string with the representation of this swirl

Parameters **verbosity** (*int*) – various verbosity level see `FingerPrint.swirl.Swirl.printVerbose()`

Return type string

Returns a human readable representation of this Swirl

searchModules()
It searches for missing dependencies using the 'module' command line. `check()` should be called before this

Return type string

Returns with a human readable list of module which can satisfy missing dependencies

setExtraPath (*path*)
These paths will be added to the search list when looking for dependency, they overwrite the extraPath passed at the constructor

Parameters **path** (*string*) – a string containing a list of path separated by :

setFilename (*filename*)
TODO remove this function

FingerPrint.sergeant.**getHash** (*fileName*, *fileType*)
It return a md5 checksum of the given file name. If we are running on a system which prelink binaries (aka RedHat based) the command prelink must be on the PATH

Parameters

- **fileName** (*string*) – a path to the file which we want to checksum
- **fileType** (*string*) – the file type (the only recognized value is EFL for triggering the prelink on RHEL base system)

Return type string

Returns an hexdadeciaml representation of the md5sum checksum

FingerPrint.sergeant.**getShortPath** (*path*)
Given a full path it shorten it leaving only /bin/../filename

Parameters **path** (*string*) – a long absolute path to the file

Return type string

Returns the shortened path

FingerPrint.sergeant.**is_special_folder** (*path*)
return true if path is to be considered special, which means it should be skipped from archivingi, checksumming, etc.

Parameters **path** (*string*) – an absolute path to the file

Return type bool

Returns True if the given path is special

FingerPrint.sergeant.**readFromPickle** (*fileName*)
helper function to get a swirl from a filename

Parameters **fileName** (*string*) – a relative or absolute path to the file to read

Return type FingerPrint.swirl.Swirl

Returns the Swirl read from the file

3.1.4 serializer Module

class FingerPrint.serializer.**PickleSerializer** (*fd*)
this class serialize a swirl into a pickle file format

Parameters **fd** (*file*) – the file descritor to be used for serialization or deserialization

load()
Return the Swirl read from the given file descriptor

Return type FingerPrint.swirl.Swirl

Returns the Swirl read from fd

save(*swirl*)
Saves the given swirl to the file descriptor

Parameters **swirl** (FingerPrint.swirl.Swirl) – the Swirl to be serialized

class FingerPrint.serializer.**XmlSerializer**(*fd*)
this serilizes the swirl into xml we can have multiple classes for serializing in other format. TODO it doesnot work. Unused at the moment.

read()
this should implement the read from xml

save(*swirl*)

save_depset(*dependencySet*)

3.1.5 swirl Module

class FingerPrint.swirl.**Arch**
old style classes for backward compability

is32bits()

is64bits()

set32bits()
set 32 bit architecture

set64bits()
set 64 bit architecture

class FingerPrint.swirl.**Dependency**(*major*, *minor=None*, *hwcap=None*)
Bases: FingerPrint.swirl.Arch

this class reperesent a dependency declarations, it can be used to represent either a dependency or a provides in a swirlFile. It is an abstract representation of a shared library as used inside the POSIX loader.

Parameters

- **major** (*string*) – it is the 'soname' of this dependency (e.g. libc.so.6, libacl.so.1, ...)
- **minor** (*string*) – it is an entry in the version symbol table (e.g. GLIBC_2.11, GLIBC_2.12, etc.)
- **hwcap** (*string*) – it stores special hardware capabilities (like sse3 or avx) this is a feature of the linux linker to support different instruction set

classmethod fromString(*string*)
Create a dependency from a string returned by find-require find-provide

Parameters **string** (*string*) – a line of output from the FingerPrint/plugin/find-requires or FingerPrint/plugin/find-provides

Return type FingerPrint.swirl.Dependency

Returns a new instance of Dependency which represent the given input string

getMajor()

Return type string

Returns the major of this dependency

getMinor()

Return type string

Returns the minor of this dependency

getName()
return a string representation of this dependency which is the same format used by the find-require find-provides (e.g. soname(minor_version)(arch)

Return type string

Returns a representation of this Dependency

isLoader()
return true if this is the loader

Return type bool

Returns true if this dependency is a loader

class FingerPrint.swirl.**Swirl**(*name*, *creationDate*)
Bases: object

Swirl hold in memory the representation of a swirl. It is made of a list of SwirlFiles aka files tracked by this swirl. There is one instance of this class for each fingerprint process.

Parameters

- **name** (*string*) – a internal simbolic name for this swirl
- **creationDate** (datetime.datetime) – the creation time of this Swirl

createSwirlFile(*fileName*)
given a fileName it return the associated swirlFile if present otherwise it creates a new one with all the symlinks resolved

Parameters **fileName** (*string*) – the path of the file to add to this swirl

Return type FingerPrint.swirl.SwirlFile

Returns a SwirlFile for the given fileName

getDateString()
return the creation time in a readable format

Return type string

Returns a string with the representation of the creation time of this swirl

getDependencies()
return a list with all the dependencies in this swirl

Return type list

Returns a list of FingerPrint.swirl.Dependency which are needed inside by all the binaries inside this Swirl

getListSwirlFileProvide(*dependencies*, *excludeSwirlFile*=[])
return a list of FingerPrint.swirl.SwirlFile from the current Swirl which can satisfy the given list of dependencies

This function does not find recursive dependencies like getListSwirlFilesDependentStatic and getListSwirlFilesDependentStaticAndDynamic

Parameters

- **dependencies** (*list*) – a list of FingerPrint.swirl.Dependency

- **excludeSwirlFile** (*list*) – a list of `FingerPrint.swirl.SwirlFile` which should be excluded from the returned list

Return type list

Returns a list of `FingerPrint.swirl.SwirlFile` which can satisfy the list of dependencies

getListSwirlFilesDependentStatic(*swirlFile*)

Given a swirlFile it return a list of all the recursively required dependent swirlFiles (only static).

It _recursively_ find all the required swirlFile invoking getListSwirlFile until all dependencies and dependencies of dependencies are resolved (when the loader start program 'a' which depend on lib 'b' which in its turn depends on lib 'c', the loader will load a, b, and c at the same time).

Parameters **swirlFile** (`FingerPrint.swirl.SwirlFile`) – a swirlFile which is part of this Swirl

Return type list

Returns a list of `FingerPrint.swirl.SwirlFile` which are all the static dependencies of the input swirlFile

getListSwirlFilesDependentStaticAndDynamic(*swirlFile*)

Given a swirlFile it returns a list of all its required swirlfiles. It includes both static recursive and dynamic dependencies

Parameters **swirlFile** (`FingerPrint.swirl.SwirlFile`) – a swirlFile which is part of this Swirl

Return type list

Returns a list of `FingerPrint.swirl.SwirlFile` which are all the dependencies of the input swirlFile

getLoader(*swirlFile*)

return a swirlfile which is the loader of the given swirlFile

Parameters **swirlFile** (`FingerPrint.swirl.SwirlFile`) – a swirlFile which is part of this Swirl

Return type `FingerPrint.swirl.SwirlFile`

Returns a SwirlFile which is the loader of the input swirlFile or None in case the input swirlFile is static

getSwirlFileByProv(*dependency*)

find the swirl file which provides the given dependency

Parameters **dependency** (`FingerPrint.swirl.Dependency`) – the dependency which should be satisfied

Return type `FingerPrint.swirl.SwirlFile`

Returns a SwirlFile which provides the given dependency None if it could not be found

isFileTracked(*fileName*)

return true if fileName is already tracked by this swirl

Parameters **fileName** (*string*) – the path of the file to look up

Return type bool

Returns true if fileName is tracked by this swirl

printVerbose(*verbosity=1*)
return a string representation of this swirl. This method is called by the -d flags

Parameters **verbosity** (*int*) – the level of verbosity 0 minimum 2 maximum

Return type string

Returns a string with a representation of this Swirl

class FingerPrint.swirl.**SwirlFile**(*path*, *links*)
Bases: FingerPrint.swirl.Arch

Encapsulate all the info we need to track for each file. At the moment only ELF aka binary file are really supported everything else is considered 'data'.

There is only 1 swirlFile instance for each file in a given swirl for example if libabc is used by /bin/ls and /bin/ps they will both point to the same instance of libabc

Parameters

- **path** (*string*) – The aboslute path of this SwirlFile this is the identificative key for this SwirlFile
- **links** (*list*) – a list of string with all the discovered simbolic links pointing to this SwirlFile

addDependency(*dependency*)
if dependency is not already in the static dependency of this swirl file it gets added

Parameters **dependency** (FingerPrint.swirl.Dependency) – an instance of Dependency to be added

addProvide(*dependency*)
if dependency is not already in the provides of this SwirlFile it gets added

Parameters **dependency** (FingerPrint.swirl.Dependency) – an instance of Dependency to be added

getDependenciesDict(*provides=False*)
Return a dictionary containing the dependencies or the provides of this SwirlFile

Parameters **provides** (*bool*) – if provides is equal to True this function returns what this SwirlFile provides instead of what it requires

Return type dict

Returns a dict where the keys are sonames of the values are lists of library versions (e.g. {'libc.so.6' : ['GLIBC_2.10', 'GLIBC_2.11', 'GLIBC_2.12']})

getPaths()
return a list of path used by this SwirlFile (it includes all the symbolic links)

Return type list

Returns return a list of strings

getProvidesDict()

Return type dict

Returns a dict which represent all the Dependecy provided by this class see getDependenciesDict for the format of the dictionary

isELFExecutable()

Return type bool

Returns true if this SwirlFile is executable

isLoader()

Return type bool

Returns return True if this SwirlFile is a loader

isYourPath(*path*)
check if this path is part of this swirlFile looking into the links as well

Parameters **path** (*string*) – a file path

Return type bool

Returns true if the given path is part of this SwirlFile

printOpenedFiles(*execFile*, *tabs=''*)
return a string of opened file by the given executable path execFile

Parameters

- **execFile** (*string*) – used to get the list of opened file by a specific executable, shared libs can open different file when loaded under different executable
- **tabs** (*string*) – used to indent the output, it will be placed at the beginning of each line

Return type string

Returns a string with all the opened file of this SwirlFile (used by the -d flags)

printVerbose(*separator=''*, *dynamic=''*, *verbosity=1*)
returns a string represeting this SwrilFile

Parameters

- **seprator** (*string*) – used to indent the output, it will be placed at the beginning of each line
- **dynamic** (*string*) – used to add a string to the first output line. Currently it is used to put the –dyn– if this SwirlFile was a dynamic loaded file
- **verbosity** (*int*) – verbosity level. 0 for the lower level 1 or 2 to get more info

Return type string

Returns a detailed representation of this SwirlFile (used by the -d flags)

setLinks(*links*)
update the list of symbolic links pointing to this swirl file

Parameters **links** (*list*) – a list of string with file path names

setPluginName(*name*)
Set the plugin type of this file (at the moment we have only elf plugin)

Parameters **name** (*string*) – the plugin name as in FingerPrint/plugins

3.1.6 syscalltracer Module

class FingerPrint.syscalltracer.**ObjectFile**(*filename*)
This class wraps an elf object file and its assembler code used by the stack tracing facility. This class depend on objdump to disassemble binaries. This class need several optimization (uses a lot of memory and CPU time).

Parameters **filename** (*string*) – the path to the binary will be disassembled

getInstruction(*vma*)
it decodes the instruction at the given virtual memory address

Parameters **vma** (*string*) – the virtual memory address in an hexadecimal format

Return type tuple

Returns a tuple of tree strings where the first string is the opcode at the given address vma, the second is the address referred by the instruction and the third is the symbolic name referred by the address

getPrevInstruction(*vma*)
it decodes the previous instruction at the given virtual memory address

Parameters **vma** (*string*) – the virtual memory address in an hexadecimal format

Return type tuple

Returns a tuple of tree strings where the first string is the opcode at the given address vma, the second is the address referred by the instruction and the third is the symbolic name referred by the address

isDynamic()

Return type bool

Returns true if this is a dynamic object aka a shared library

class FingerPrint.syscalltracer.**SyscallTracer**
this class can spawn a process and trace its' execution to record what are its dynamic dependency requirements

Usage:

```
tracer = SyscallTracer()
execcmd = shlex.split(execcmd)
tracer.main(execcmd)
# output will in the TracerControlBlock static variables
TracerControlBlock.[files|dependencies|env|cmdline]
```

main(*command*)
start the trace with the given command

Parameters **command** (*string*) – command line to trace passed through shlex.split

Return type bool

Returns false if something went wrong

readCString(*address*, *pid*)

test()

class FingerPrint.syscalltracer.**TracerControlBlock**(*pid*)
This class hold data needed for tracing a processes. Inspired by strace code (struct tcb).

PS: I don't really like this solution of static variable but for the moment ti does its job

Parameters **pid** (*int*) – the PID of the process that we are tracing

cmdline = {}
dictionary that keeps track of the executed command line. Keys are the full path to the executable and values are a list of strings containing all the token of the command line

dependencies = {}
Dictionary of shared libraries used by the various processes. E.g.: `'{ 'binarypath': [list of file it depends to], '/bin/bash': ['/lib/x86_64-linux-gnu/libnss_files-2.15.so', '/lib/x86_64-linux-gnu/libnss_nis-2.15.so']}'`

env = {}
Dictionary that keeps track of process environment variables. Keys are the full path to the executable of the process and values are a list of strings containing all the variables

files = {}
Dictionary of dictionary of opened files by the various processes. E.g. files[libraryA][executableB] and files[libraryA][executableC] return respectively the list of opened file by the libraryA when run under executableB and when run under executableC.

getFileOpener()
if Fingerprint is compiled with the stack tracer module it will find the file object who contains the code which initiated this open system call if not it will return the path to the current process. This function is called after each open system call.

Return type string

Returns the path of the library who triggered the current open system call

getProcessCWD()

Return type string

Returns return the current working directory of this process

getProcessName()

Return type string

Returns the process name (this is used in all the static attribute of this class as a key)

classmethod **get_env_variable**(*process_name*, *variable_name*)
returns the value of the variable_name if found int he process_name environment

Parameters

- **process_name** (*string*) – the full path to the executable representing this process
- **variable_name** (*string*) – the name of the variable

Return type string

Returns a environment variable value

classmethod **set_trace_function**()
This class method load the function needed to set up the stack tracer which require the external shared library. Called only once.

updateProcessInfo()
This method updates the process information into the global static variables `TracerControlBlock.cmdline`, `TracerControlBlock.env` of this class. This method is called only once when this instance is created (aka when the process is created).

updateSharedLibraries()
This method scans the procfs to find the shared libraries loaded by this process and it updates the static `TracerControlBlock.dependencies` variable accordingly. This function is called every time the process invoke the mmap system call.

3.1.7 `utils` Module

FingerPrint.utils.**all**(*iterable*)

FingerPrint.utils.**any**(*iterable*)

FingerPrint.utils.**getLDLibraryPath**(*env*)
given a list of environment variables it return a list of absolute path defined in LD_LIBRARY_PATH (if a path is relative it will be transformed in an absolute with PWD)

FingerPrint.utils.**getOutputAsList**(*binary*, *inputString=None*)
run popen pipe inputString and return a touple of (the stdout as a list of string, return value of the command)

FingerPrint.utils.**which**(*program*, *extra_paths=None*)
extra path is a string containing a list of path separated by : which

3.1.8 Subpackages

plugins Package

plugins Package

class FingerPrint.plugins.**PluginManager**
Bases: object

Super class of the various plugins. All plugins should inherit from this class.

To implement a new Plugin you should subclass this class and provide the following attributes/methods:

- pluginName: this must be a unique string representing the plugin name
- **getPathToLibrary(): a class method which return a file name pointing to the** file which can provide the given dependency
- **getSwirl(): a class method that given a path to a file it return None if the** file can not be handled by the given plugin or a SwirlFile with the dependency set if the plugin can handle the file

classmethod addSystemPaths(*paths*)
add an additional paths to the search for dependency

Parameters **paths** (*list*) – a list of string with the extra path to be added

classmethod getPathToLibrary(*dependency*, *useCache=True*, *rpath=[]*)
Given a dependency it find the path of the library which provides that dependency

Parameters

- **dependency** (FingerPrint.swirl.Dependency) – the Dependency that we need to satisfy with the returned library
- **useCache** (*bool*) – if true it will use a cache that will speed up a lot searching for libraries
- **rpath** (*list*) – a list of string which contains extra paths that we want to add during the search for the dependency Generally used to add RPATH to the search path.

Return type string

Returns the path to the library which satisfy the given dependency

classmethod getSwirl(*fileName*, *swirl*, *env=None*)
helper function given a filename it return a SwirlFile. This should be re-implemented by the various plugins. If none of the plugins return a SwirlFile this method will return a 'data' SwirlFile.

ATT: only one plugin should return a SwirlFile for a given file

ATT2: this is a class method

Parameters

- **fileName** (*string*) – a path to the new file we want to add
- **swirl** (FingerPrint.swirl.Swirl) – the current Swirl object. Static dependencies of the new SwirlFile are resolved first inside the Swirl and if not found then they are resolved recursively invoking this function and recursively added to the Swirl
- **env** (*list*) – a list of string with all the environment variable available to this file when it was executing. This field is used only when doing dynamic tracing.

Return type FingerPrint.swirl.SwirlFile

Returns a SwirlFile representing the given fileName. The SwirlFile should have all the static dependencies resolved (if they could be find on the system)

plugins = {'ELF': <class 'FingerPrint.plugins.elf.ElfPlugin'>}

systemPath = []
list of string containing the paths we should look for dependencies

class FingerPrint.plugins.**PluginMount**(*name*, *bases*, *attrs*)
Bases: type

this is a singleton object which can return a list of all available plugins. All plugin available inside the FingerPrint.plugins are loaded inside the PluginMount when this module is loaded.

Insipired by (or totaly copied from) Marty Alchin: http://martyalchin.com/2008/jan/10/simple-plugin-framework/

get_plugins()
return the list of currently registered plugins

Return type list

Returns a list of PluginManager registered

elf Module

class FingerPrint.plugins.elf.**ElfPlugin**
Bases: FingerPrint.plugins.PluginManager

This plugin manages all ELF file format. This class requires the find-provides and find-requires script present in this folder which require: objdump, awk, sed, grep.

For nicer documentation on this functions see FingerPrint.plugins.PluginManager

classmethod getPathToLibrary(*dependency*, *useCache=True*, *rpath=[]*)
given a dependency it find the path of the library which provides that dependency

classmethod getSwirl(*fileName*, *swirl*, *env=None*)
helper function given a filename it return a SwirlFile if the given plugin does not support the given fileName should just return None

ATT: only one plugin should return a SwirlFile for a given file

pluginName = 'ELF'

ptrace Package

cpu_info Module

Constants about the CPU:

- CPU_BIGENDIAN (bool)
- CPU_64BITS (bool)
- CPU_WORD_SIZE (int)
- CPU_MAX_UINT (int)
- CPU_PPC32 (bool)
- CPU_PPC64 (bool)
- CPU_I386 (bool)
- CPU_X86_64 (bool)
- CPU_INTEL (bool)
- CPU_POWERPC (bool)

ctypes_errno Module

Function get_errno(): get the current errno value.

Try different implementations:

- ctypes_support.get_errno() function
- __errno_location_sym symbol from the C library
- PyErr_SetFromErrno() from the C Python API

FingerPrint.ptrace.ctypes_errno.**get_errno**()
Read errno using Python C API: raise an exception with PyErr_SetFromErrno and then read error code 'errno'.

This function may raise an RuntimeError.

ctypes_libc Module

Load the system C library. Variables:

- LIBC_FILENAME: the C library filename
- libc: the loaded library

ctypes_tools Module

FingerPrint.ptrace.ctypes_tools.**bytes2array**(*bytes*, *basetype*, *size*)
Cast a bytes string to an array of objets of the specified type and size.

FingerPrint.ptrace.ctypes_tools.**bytes2type**(*bytes*, *type*)
Cast a bytes string to an objet of the specified type.

FingerPrint.ptrace.ctypes_tools.**bytes2word**(*bytes*)
Convert a bytes string to an unsigned integer (a CPU word).

FingerPrint.ptrace.ctypes_tools.**formatAddress**(*address*)
Format an address to hexadecimal. Return "NULL" for zero.

FingerPrint.ptrace.ctypes_tools.**formatAddressRange**(*start*, *end*)
Format an address range, eg. "0x80004000-0x8000ffff".

FingerPrint.ptrace.ctypes_tools.**formatUintHex16**(*value*)
Format an 16 bits unsigned integer.

FingerPrint.ptrace.ctypes_tools.**formatUintHex32**(*value*)
Format an 32 bits unsigned integer.

FingerPrint.ptrace.ctypes_tools.**formatUintHex64**(*value*)
Format an 64 bits unsigned integer.

FingerPrint.ptrace.ctypes_tools.**formatWordHex**(*value*)
Format an 64 bits unsigned integer.

FingerPrint.ptrace.ctypes_tools.**int2uint**(*value*)
Convert a signed 32 bits integer into an unsigned 32 bits integer.

FingerPrint.ptrace.ctypes_tools.**int2uint32**(*value*)
Convert a signed 32 bits integer into an unsigned 32 bits integer.

FingerPrint.ptrace.ctypes_tools.**int2uint64**(*value*)
Convert a signed 64 bits integer into an unsigned 64 bits integer.

FingerPrint.ptrace.ctypes_tools.**long2ulong**(*value*)
Convert a signed 64 bits integer into an unsigned 64 bits integer.

FingerPrint.ptrace.ctypes_tools.**ntoh_uint**(*value*)
Convert an unsigned integer from network endiant to host endian.

FingerPrint.ptrace.ctypes_tools.**ntoh_ushort**(*value*)
Convert an unsigned short integer from network endiant to host endian.

FingerPrint.ptrace.ctypes_tools.**truncateWord**(*value*)
Truncate an unsigned integer to 64 bits.

FingerPrint.ptrace.ctypes_tools.**truncateWord32**(*value*)
Truncate an unsigned integer to 32 bits.

FingerPrint.ptrace.ctypes_tools.**truncateWord64**(*value*)
Truncate an unsigned integer to 64 bits.

FingerPrint.ptrace.ctypes_tools.**uint2int**(*value*)
Convert an unsigned 32 bits integer into a signed 32 bits integer.

FingerPrint.ptrace.ctypes_tools.**uint2int32**(*value*)
Convert an unsigned 32 bits integer into a signed 32 bits integer.

FingerPrint.ptrace.ctypes_tools.**uint2int64**(*value*)
Convert an unsigned 64 bits integer into a signed 64 bits integer.

FingerPrint.ptrace.ctypes_tools.**ulong2long**(*value*)
Convert an unsigned 64 bits integer into a signed 64 bits integer.

FingerPrint.ptrace.ctypes_tools.**word2bytes**(*word*)
Convert an unsigned integer (a CPU word) to a bytes string.

error Module

exception FingerPrint.ptrace.error.**PtraceError**(*message*, *errno=None*, *pid=None*)
Bases: exceptions.Exception

Ptrace error: have the optional attributes errno and pid.

func Module

FingerPrint.ptrace.func.**WPTRACEEVENT**(*status*)

FingerPrint.ptrace.func.**ptrace**(*command*, *pid=0*, *arg1=0*, *arg2=0*, *check_errno=False*)

FingerPrint.ptrace.func.**ptrace_attach**(*pid*)

FingerPrint.ptrace.func.**ptrace_cont**(*pid*, *signum=0*)

FingerPrint.ptrace.func.**ptrace_detach**(*pid*, *signal=0*)

FingerPrint.ptrace.func.**ptrace_geteventmsg**(*pid*)

FingerPrint.ptrace.func.**ptrace_getfpregs**(*pid*)

FingerPrint.ptrace.func.**ptrace_getregs**(*pid*)

FingerPrint.ptrace.func.**ptrace_getsiginfo**(*pid*)

FingerPrint.ptrace.func.**ptrace_kill**(*pid*)

FingerPrint.ptrace.func.**ptrace_peekdata**(*pid*, *address*)

FingerPrint.ptrace.func.**ptrace_peektext**(*pid*, *address*)

FingerPrint.ptrace.func.**ptrace_peekuser**(*pid*, *address*)

FingerPrint.ptrace.func.**ptrace_pokedata**(*pid*, *address*, *word*)

FingerPrint.ptrace.func.**ptrace_poketext**(*pid*, *address*, *word*)

FingerPrint.ptrace.func.**ptrace_pokeuser**(*pid*, *address*, *word*)

FingerPrint.ptrace.func.**ptrace_setfpregs**(*pid*, *fpregs*)

FingerPrint.ptrace.func.**ptrace_setoptions**(*pid*, *options*)

FingerPrint.ptrace.func.**ptrace_setregs**(*pid*, *regs*)

FingerPrint.ptrace.func.**ptrace_setsiginfo**(*pid*, *info*)

FingerPrint.ptrace.func.**ptrace_singlestep**(*pid*)

FingerPrint.ptrace.func.**ptrace_syscall**(*pid*, *signum=0*)

FingerPrint.ptrace.func.**ptrace_traceme**()

linux_struct Module

class FingerPrint.ptrace.linux_struct.**siginfo**

Bases: _ctypes.Structure

pad
Structure/Union member

si_code
Structure/Union member

si_errno
Structure/Union member

si_signo
Structure/Union member

class FingerPrint.ptrace.linux_struct.**user_fpregs_struct**
Bases: _ctypes.Structure

cwd
Structure/Union member

fop
Structure/Union member

ftw
Structure/Union member

mxcr_mask
Structure/Union member

mxcsr
Structure/Union member

padding
Structure/Union member

rdp
Structure/Union member

rip
Structure/Union member

st_space
Structure/Union member

swd
Structure/Union member

xmm_space
Structure/Union member

class FingerPrint.ptrace.linux_struct.**user_regs_struct**
Bases: _ctypes.Structure

cs
Structure/Union member

ds
Structure/Union member

eflags
Structure/Union member

es
Structure/Union member

fs
Structure/Union member

fs_base
Structure/Union member

gs
Structure/Union member

gs_base
Structure/Union member

orig_rax
Structure/Union member

r10
Structure/Union member

r11
Structure/Union member

r12
Structure/Union member

r13
Structure/Union member

r14
Structure/Union member

r15
Structure/Union member

r8
Structure/Union member

r9
Structure/Union member

rax
Structure/Union member

rbp
Structure/Union member

rbx
Structure/Union member

rcx
Structure/Union member

rdi
Structure/Union member

rdx
Structure/Union member

rip
Structure/Union member

rsi
Structure/Union member

rsp
Structure/Union member

ss
Structure/Union member

os_tools Module

Constants about the operating system:

- RUNNING_PYPY (bool)

- RUNNING_WINDOWS (bool)
- RUNNING_LINUX (bool)
- RUNNING_FREEBSD (bool)
- RUNNING_OPENBSD (bool)
- RUNNING_MACOSX (bool)
- RUNNING_BSD (bool)
- HAS_PROC (bool)
- HAS_PTRACE (bool)

`signames` Module

Name of process signals.

SIGNAMES contains a dictionary mapping a signal number to it's name. But you should better use signalName() instead of SIGNAMES since it returns a string even if the signal is unknown.

FingerPrint.ptrace.signames.**getSignalNames**()
Create signal names dictionay (eg. 9 => 'SIGKILL') using dir(signal). If multiple signal names have the same number, use the first matching name in PREFERRED_NAME to select preferred name (eg. SIGIOT=SIGABRT=17).

FingerPrint.ptrace.signames.**signalName**(*signum*)
Get the name of a signal

```
>>> from signal import SIGINT
>>> signalName(SIGINT)
'SIGINT'
>>> signalName(404)
'signal<404>'
```

CHAPTER 4

Authors and Contributors

Fingerprint is an idea of Phil Papadopoulos and it is developed by Phil and Luca Clementi. This work is funded by NSF under the grant #1148473.

CHAPTER 5

Support or Contact

If you are having trouble with FingerPrint or if you need some help you can post an email on the Rocks mailing list npaci-rocks-discussion@sdsc.edu or pust an issue on github.

f

A

B

C

D

E

F

G

I

L

M

T

U

W

X