haystack Documentation

Release 0.42

Loïc Jaquemet

Jul 03, 2017

Contents

1 Summary: **3**

2 Packages: **5**

3 Contents: **7**

3.1 Installation 7
3.2 Usage 8
3.3 Capture a process memory to file 10
3.4 Making your own memory mappings handler 10

Python Module Index **13**

Useful links

- Code repository

CHAPTER 1

Summary:

Haystack is a framework dedicated to process heap analysis. The general idea is that process memory contains user data (the interesting stuff) allocated by the process and system metadata allocated by the kernel (in short) to manage allocation and de-allocation of user data (as on of many metadata present in there)

This framework assists its user in a programmatic interpretation of the system allocation metadata, so that the user can then concentrate on interpretation of the user-data itself.

This framework also provide a way to search user allocated memory for specific instance of user defined types such as C records. That mechanism is used internally to identify the system metadata records used by the memory allocator to manage allocation of user memory.

The framework also provide a way to reverse engineer the types of memory structure in use by a process. The reversed types will take into account linked list, pointers and other value constraints to propose a list of type definition.

CHAPTER 2

Packages:

The core package python-haystack is providing the base modules and classes to search for instance of C records in a process memory. Based on types definition (using python ctypes) and value constraints defined by the user, the package allows to search a process memory for such instances.

The additional package python-haystack-reverse is providing a set of tools to assist in reversing the types used by a process and recreate type definitions.

CHAPTER 3

Contents:

Installation

These procedures were tested on Ubuntu 16.04.

Install from PyPi

Install a virtual environment:

```
$ virtualenv v_haystack
$ source v_haystack/bin/activate
```

Install python-haystack:

```
(v_haystack) $ pip install haystack
```

Keeping it up to date

```
(v_haystack) $ pip install haystack --upgrade
```

Clone+Install from GitHub

Clone python-haystack:

```
$ git clone https://github.com/trolldbois/python-haystack.git
```

Setup a virtual environment:

```
$ virtualenv v_haystack
$ source v_haystack/bin/activate
```

Install python-haystack (won't work otherwise):

```
(v_haystack) $ cd python-haystack
(v_haystack) ~/python-haystack$ pip install -r requirements
(v_haystack) ~/python-haystack$ python setup.py install
```

Keeping it up to date

```
(v_haystack) $ cd python-haystack
(v_haystack) ~/python-haystack$ git pull
```

Usage

First you need to install python-haystack. Please refer to the *Installation* section of the documentation.

Then you need a process memory dump. Please refer to the *Capture a process memory to file* section of the documentation. We will name the process memory dump *memory.dmp* for the rest of this documentation.

Command line usage

A few entry points exists for different purposes

- `haystack-find-heap` allows to show details on Windows HEAP.
- `haystack-search` allows to search for instance of types
- `haystack-show` allows to show a specific formatted values of a type instance at a specific memory address

You can use the following URL to designate your memory handler/dump:

- `dir:///path/to/my/haystack/fump/folder` to use the haystack dump format
- `dmp:///path/to/my/minidump/file` use the minidump format (microsoft?)
- `frida://name_or_pid_of_process_to_attach_to` use frida to access a live process memory
- `live://name_or_pid_of_process_to_attach_to` ptrace a live process
- `rekall://` load a rekall image
- `volatility://` load a volatility image

API usage

`haystack.search.api.`**`load_record`**(*memory_handler*, *struct_type*, *memory_address*, *load_constraints=None*)

Load a record from a specific address in memory. You could use that function to monitor a specific record from memory after a refresh.

Parameters

- **`memory_handler`** – IMemoryHandler
- **`struct_type`** – a ctypes.Structure or ctypes.Union
- **`memory_address`** – long
- **`load_constraints`** – IModuleConstraints to be considered during loading

Returns (ctypes record instance, validated_boolean)

haystack.search.api.**output_to_json**(*memory_handler*, *results*)
Transform ctypes results in a json format :param memory_handler: IMemoryHandler :param results: results from the search_record :return:

haystack.search.api.**output_to_pickle**(*memory_handler*, *results*)
Transform ctypes results in a pickled format. To load the pickled objects, you need to have haystack in your path.

Parameters

- **memory_handler** – IMemoryHandler
- **results** – results from the search_record

Returns

haystack.search.api.**output_to_python**(*memory_handler*, *results*)
Transform ctypes results in a non-ctypes python object format :param memory_handler: IMemoryHandler :param results: results from the search_record :return:

haystack.search.api.**output_to_string**(*memory_handler*, *results*)
Transform ctypes results in a string format :param memory_handler: IMemoryHandler :param results: results from the search_record :return:

haystack.search.api.**search_record**(*memory_handler*, *record_type*, *search_constraints=None*, *extended_search=False*)
Search a record in the memory dump of a process represented by memory_handler.

The record type must have been imported using haystack functions.

If constraints exists, they will be considered during the search.

Parameters

- **memory_handler** – IMemoryHandler
- **record_type** – a ctypes.Structure or ctypes.Union from a module imported by haystack
- **search_constraints** – IModuleConstraints to be considered during the search
- **extended_search** – boolean, use allocated chunks only per default (False)

:rtype a list of (ctypes records, memory offset)

haystack.search.api.**search_record_hint**(*memory_handler*, *record_type*, *hint*, *search_constraints=None*, *extended_search=False*)
Search a record in the memory dump of a process, but only on the memory page containing the hinted address.

The record type must have been imported using haystack functions.

If constraints exists, they will be considered during the search.

Parameters

- **memory_handler** – IMemoryHandler
- **record_type** – a ctypes.Structure or ctypes.Union from a module imported by haystack
- **search_constraints** – IModuleConstraints to be considered during the search
- **extended_search** – boolean, use allocated chunks only per default (False)

:rtype a list of (ctypes records, memory offset)

haystack.search.api.**validate_record**(*memory_handler*, *instance*, *record_constraints=None*, *max_depth=10*)
Validate a loaded record against constraints.

Parameters

- **memory_handler** – IMemoryHandler
- **instance** – a ctypes record
- **record_constraints** – IModuleConstraints to be considered during validation

Returns

Capture a process memory to file

First of all, be prepared to face a need for elevated privileges.

On Windows, the most straightforward is to get a Minidump. The Windows task manager allows to capture a process memory to file. Alternatively the Microsoft Sysinternals suite of tools provide either a CLI (procdump.exe) or a GUI (Process explorer). Using one of these (with full memory dump option) you will produce a file that can be used with the `haystack-xxx` list of entry points using the `dmp://` file prefix.

While technically you could use many third party tool, Haystack actually need memory mapping information to work with the raw memory data. In nothing else, there is a dumping tool included in the pytahon-haystack package that leverage python-ptrace to capture a process memory. See the `haystack-live-dump` tool:

```
# haystack-live-dump <pid> myproc.dump
```

For live processes

- `haystack-live-dump` capture a process memory dump to a folder (haystack format)

For a Rekall memory dump

- `haystack-rekall-dump` dump a specific process to a haystack process dump

For a Volatility memory dump

- `haystack-volatility-dump` dump a specific process to a haystack process dump

Interesting note for Linux users, dumping a process memory for the same user can be done if you downgrade the "security" of your system by allowing cross process ptrace access:

```
$ sudo sysctl kernel.yama.ptrace_scope=0
```

Interesting note for Windows users, most processes memory can be dumped to a Minidump format using the task manager. (NB: I don't remember is the process memory mapping are included then)

Making your own memory mappings handler

If you have a different technique to access a process memory, you can implement the `haystack.abc.interfaces.IMemoryLoader` and `haystack.abc.interfaces.IMemoryMapping` interface for your favorite technique. Check out the Frida plugin for example.

Alternatively, if you can copy the process' memory mappings to file, you can "interface" with the basic, simple, haystack memory dump file format by doing the following: The basic format is a folder containing each memory mapping in a separate file :

- memory content in a file named after it's start/end addresses (ex: 0x000700000-0x000800000)
- a file named 'mappings' containing memory mappings metadata. (ex: mappings)

class haystack.abc.interfaces.**IMemoryLoader**
Parse a process memory _memory_handler from a storage concept, then identify its ITargetPlatform characteristics and produce an IMemoryHandler for this process memory dump

make_memory_handler()
Returns an instance of IMemoryHandler

class haystack.abc.interfaces.**IMemoryMapping**
Interface for a memory mapping. A IMemoryMapping should hold one of a process memory _memory_handler and its start and stop addresses.

read_array(*address*, *basetype*, *count*)
Reads the memory content at address <address> and returns an typed array.

Parameters

- **address** – long the virtual address.
- **basetype** – a ctypes class.
- **count** – long the size of the array.

Returns the memory content at address, in an array form

Return type (basetype*count) ctypes class

read_bytes(*address*, *size*)
Reads the memory content at address <address> and returns an array of bytes in a str.

Parameters

- **address** – long the virtual address.
- **size** – long the size of the array.

Returns the memory content at address, in an bytes string

Return type str

read_cstring(*address*, *max_size*, *chunk_length=256*)
Reads the memory content at address <address> and returns a python representation of the NULL terminated string.

Parameters

- **address** – long the virtual address.
- **max_size** – long the maximum size of the string.
- **chunk_length** – (optional) long the number of bytes read at each buffer read.

Returns the memory content at address, in an bytes string

Return type str

read_struct(*address*, *struct*)
Reads the memory content at address <address> and returns an ctypes record instance.

Parameters

- **address** – long the virtual address.
- **struct** – a ctypes class.

Returns the memory content at address, in an ctypes record form

Return type (struct) ctypes class

read_word(*address*)

Reads the memory content at address <address> and returns an word worth of it. Usually 4 or 8 bytes.

Parameters **address** – long the virtual address.

Returns the memory content at address, in an bytes string

Return type str

search(*bytestr*)

Search the memory for this particular sequence of bytes and iterates over the starting address of the results.

Parameters **bytestr** – bytes str, the sequence of bytes to look for.

Returns (iterator) long, the list of virtual address matching the byte pattern

Return type iterator, long, the starting virtual address of the match

Python Module Index

h

haystack.search.api, 8

Index

H

haystack.search.api (module), 8

I

IMemoryLoader (class in haystack.abc.interfaces), 11
IMemoryMapping (class in haystack.abc.interfaces), 11

L

load_record() (in module haystack.search.api), 8

M

make_memory_handler() (haystack.abc.interfaces.IMemoryLoader method), 11

O

output_to_json() (in module haystack.search.api), 9
output_to_pickle() (in module haystack.search.api), 9
output_to_python() (in module haystack.search.api), 9
output_to_string() (in module haystack.search.api), 9

R

read_array() (haystack.abc.interfaces.IMemoryMapping method), 11
read_bytes() (haystack.abc.interfaces.IMemoryMapping method), 11
read_cstring() (haystack.abc.interfaces.IMemoryMapping method), 11
read_struct() (haystack.abc.interfaces.IMemoryMapping method), 11
read_word() (haystack.abc.interfaces.IMemoryMapping method), 12

S

search() (haystack.abc.interfaces.IMemoryMapping method), 12
search_record() (in module haystack.search.api), 9
search_record_hint() (in module haystack.search.api), 9

V

validate_record() (in module haystack.search.api), 9