CarnivoreLE

(Live Edition)

Description

CarnivoreLE is a triage tool primarily intended for forensically-sound previews and data capture on live suspect computers. The suspect computer must be either IBM PC or Apple Macintosh architecture equipped with microprocessor(s) supporting AMD 64 bit. The suspect operating system must be a 64-bit version of GNU/Linux (2.6.24 kernel or above), Microsoft Windows (version 6.0 Vista or above), or OS X/MacOS 10.7 or above. Issues may be encountered with the following:

- GNU/Linux distributions other than those based on RPM or DEB and/or with kernel versions below 2.6.24.
- Windows versions lower than 6.1 “Windows 7”.
- OS X versions lower than 10.7.

Minimum hardware resources sufficient to run any of the above operating systems as prescribed by their respective manufacturers are also sufficient to run CarnivoreLE.

CarnivoreLE is designed with speed, forensic integrity, and automation as priorities. It takes advantage of a command line interface (CLI) to maintain the smallest memory footprint possible. When ran from a properly configured external device, i.e. Carnivore Portable, and with access to root or administrator privilege, data gathering modules called *reapers* each provide dialog and options to the examiner. All operations are logged and maintained in the evidence store. CarnivoreLE is written in C and makes use of the fastest technologies such as Snappy compression, Advanced Forensic Format version 4 (AFF4) formating, and PMEM imagers. Applications written in the C language are very fast on all three systems. The C language, being both a low- and mid-level language, has been used consistently since the late 1960’s to build programs that are arguably faster than comparable programs written in any other language. Automation to enhance ease of use and reduction of user error is consistent throughout.

Carnivore Portable is a rugged external disk drive equipped with a 1.7” 2 TiB solid state drive (SSD) bearing one logical partition and one extended partition with four volumes:

Partition 1 – Primary - CARNPM
    5.4 GiB, FAT32, contains live disk CarnivorePM
Partition 2 – Extended
    Partition 5 – WINTOOLS
        4.3 GiB, NTFS, contains live tools for Windows including CarnivoreLE. The `carnivore` or `carnivore.exe` executable file must be accompanied by the `config`, `docs`, and `thirdparty` source code directories. See Building below.

Partition 6 – MACTOOLS

4.3 GiB, HFS+, contains live tools for Macintosh including CarnivoreLE

Partition 7 – LINTOOLS

4.3 GiB, Ext4, contains live tools for GNU/Linux including CarnivoreLE

Partition 8 – EVIDENCE

1.9 TiB, exFAT, output area for CarnivoreLE and CarnivorePM

Carnivore Portable is equipped with USB 3.0 (backward compatible), Firewire 800, and eSATA ports. It meets or exceeds current US military specifications for ruggedness and submersibility. In the absence of a branded Carnivore Portable unit, any external storage disk can be substituted provided that it can be partitioned, formatted, and connected sufficiently. Substitute units should be used at the practitioner's own risk with the understanding that resource specifications less than that outlined above will be slower, less efficient, less storage space, etc.

Programming

The repo includes two main branches as follows:

master

develop

Additional branches will come and go as needed. The hierarchy can largely be ignored except to pay attention to which branch you're on. All three platforms are represented in each branch. If you intend to make general pull requests, only those sent from the develop branch will be accepted. The master branch represents the latest stable version. Developer pulls and pull requests can be made from develop and any special branch other than master if you expect to work in that branch.

Also, any changes for one platform (i.e. Linux) that should reasonably be expected to pertain to the other platforms (i.e. Windows and/or Macintosh) should be included in those as well at the development stage. Read the instructions on contributions in both repo and https://positronikal.github.io/.

CarnivoreLE is written in C with the exception of occasional scripting. When development moves outside the C source code, preferred scripting should be in bash for Linux/Macintosh or batch/Powershell for Windows. These scripting methods are likely to be the most ubiquitous across all targeted operating systems. Python being an obvious first choice, doesn't come with Windows natively and its behavior can't be predicted with accuracy across Linux distributions and Macintosh and therefore isn't practical for this application. Code::Blocks (C::B) was used as the IDE and all files necessary for configuration are included in the repos. Originally, the various implementations of CarnivoreLE were written on native platforms, but this was changed at one point in favor of GNU/Linux as the development environment for both Linux and Windows implementations. C::B is used for Macintosh as well, but only on a OS X/MacOS computer.

Version control is maintained via git from the CarnivoreLE root directory, not the individual implementations:

1. Fork the CarnivoreLE repo and clearly note that it is a fork. GitHub does this, but theirs isn't always noticeable. Don't create a fork of a fork, then expect to pull request to this origin.
2. Clone your fork to a local repo.
3. Make changes in the local repo using the preferred branch (not master).
4. Push to your fork.
5. Pull request from your fork.

The source code layout is as follows:

`-CarnivoreLE` = Root directory contains subdirectories, GNU standard documents, and C::B project files.

- `docs/` = Application documentation.

`-LIN/`

- `bin/` = Compiled binaries. Only exists if compiled using C::B.

`-Debug/`

`-Release/`

- `config/` = Configuration files.
- `include/` = Header files.
- `obj/` = Object files. Only exists if compiled using C::B.

`-Debug/`

`-Release/`

- `src/` = Source code files.
- `thirdparty/` = Third party applications.

`-carnivore.cbp`

`-carnivore.depend`

`-carnivore.layout`

`-MAC/` = OS X / MacOS implementation

- `bin/` = Compiled binaries. Only exists if compiled using C::B.

`-Debug/`

`-Release/`

- `config/` = Configuration files.
- `include/` = Header files.

- obj/ = Object files. Only exists if compiled using C::B.
    - Debug/
    - Release/
- src/ = Source code files.
- thirdparty/ = Third party applications.
    - carnivore.cbp
    - carnivore.depend
    - carnivore.layout

- WIN/ = Windows implementation
- bin/ = Compiled binaries. Only exists if compiled using C::B.
    - Debug/
    - Release/
- config/ = Configuration files.
- include/ = Header files.
- obj/ = Object files. Only exists if compiled using C::B.
    - Debug/
    - Release/
- src/ = Source code files.
- thirdparty/ = Third party applications.
    - carnivore.cbp
    - carnivore.depend
    - carnivore.layout

- .git/
- ATTRIBUTION
- AUTHORS
- BUGS
- CONTRIBUTING
- COPYING
- README.md
- USING
- VERSION

`-.gitignore`

Building

Building is done using C::B configured with GCC, Clang, and MINGW compilers for GNU/Linux, Mac, and Windows respectively. All settings for both DEBUG and RELEASE should auto-populate when the project file (carnivore.cbp) is opened in the IDE.

Substitute units for Carnivore Portable require some manual changes. Rugged external hard disk drives are highly recommended and with the type of connectivity necessary, i.e. USB 3.0, eSATA, Firewire, etc. The drive should be partitioned into at least four (4) partitions with the following labels/properties:

1. `LINTOOLS` = 4 GB, ext4
2. `MACTOOLS` = 4GB, HFS+
3. `WINTOOLS` = 4GB, NTFS
4. `EVIDENCE` = *, ExFAT

* = Remainder of drive space. Carnivore Portable ships with an `EVIDENCE` partition of ~ 1.9 TB.

Each of the `_TOOLS` partitions should have a `CarnivoreLE` directory with the following contents:

`-config/`

`-uthashes-___.txt (lin, mac, or win)`

`-utlist-___.txt (lin, mac, or win)`

`-docs/`

`-carnivore.1 (UNIX man page)`

`-CarnivoreLE Manual.pdf`

`-thirdparty/` (**)

`-hashdeep` (Linux and Macintosh) or `hashdeep64.exe` (Windows)

`-linpmem` (Linux) or `osxpmem.zip` (Macintosh) or `winpmem.exe` (Windows)

`-carnivore` (Linux or Macintosh) or `carnivore.exe` (Windows)

** = Windows must also have `handle64.exe` in the `thirdparty` directory. This file cannot be distributed by this repo per the licensing agreement. It must be manually obtained from this link and stored in the `thirdparty` directory:

https://docs.microsoft.com/en-us/sysinternals/downloads/handle

Operation Cycle and Usage

Using CarnivoreLE requires Carnivore Portable or a substitute unit and presumes the suspect computer, i.e. *Device Under Test* or DUT, is in a running state. Quick start steps are as follows:

1. Connect Carnivore Portable using the fastest cable transfer method available from the DUT and allow it to be auto-recognized/auto-mounted. Do not allow the DUT to perform any optimization checks or modifications to Carnivore Portable. On Windows systems, some modification is unavoidable, therefore the goal is to protect the Carnivore Portable bootloader as much as possible be opting out of any dialogs presented by the DUT.
2. Open a terminal window and navigate to the tools partition appropriate to the DUT, i.e. `WINTOOLS`, `MACTOOLS`, or `LINTOOLS` and run `carnivore`:

   Windows (run cmd.exe as administrator): `> carnivore`

   Mac and Linux: `$ ./carnivore`
3. Follow the onscreen prompts until the application exits. Linux and Mac will prompt for `sudo` password as necessary.
4. Leave Carnivore Portable connected and reboot the DUT into CarnivorePM to review output from CarnivoreLE or perform additional preview/triage examinations.
5. Shutdown DUT, remove Carnivore Portable and reconnect to Carnivore Workstation in a lab environment for further examination/analysis.
6. All reports, logs, and other output will be located in the session's evidence directory.

Steps Unique to OS X / MacOS

Imaging requires specific ownership and permissions to be passed to the operating system. Allow this by following these steps:

1. In Finder, select the `MACTOOLS` volume.
2. Choose `File` > `Get Info`.
3. Click the disclosure triangle next to `Sharing & Permissions`.
4. If necessary, click the lock icon and enter a username and password that can use `sudo`.
5. Uncheck the "`Ignore ownership on this volume`" checkbox. You DO NOT want to ignore ownership.

Loading the kernel extension (`kext`) requires user approval and will generate a "`System Extension Blocked`" notification. When this occurs, follow these steps:

1. Open `System Preferences` > `Security & Privacy`.
2. If necessary, click the lock icon and enter a username and password that can use `sudo`.

3. Select “`App Store & identified developers`” button.
4. Select “`Allow`” for Adam Sindelar's `kext`.

Native Utility Verification

CarnivoreLE makes calls to libraries, headers, and utilities found on the native DUT system. This alleviates the need for the practitioner to bring this same software preloaded on Carnivore Portable. The disparity between UNIX utilities and the concept of nested external references on Windows systems, i.e. libraries that reference other libraries or header files that reference other header files, makes guaranteed true static compilation sometimes next to impossible. Further, some utilities on Windows and Macintosh systems are proprietary and porting them from system to system may violate their respective terms of use. Even if that wasn’t an issue, these utilities could not be distributed as part of a Carnivore system and would have to be uniquely harvested from native systems by the field practitioner, then added to the CarnivoreLE system in order to be useful.

For these reasons and likely others, we have chosen to use native utilities following two requirements. First, that each utility is called by the full expected path and not by a path variable and, second, that each properly named utility found at the full path location matches by hash value one of a reference set of known values using both Message Digest 5 (`MD5`) and Secure Hash Algorithm 256 (`SHA256`). Hashing is accomplished in the very early stages of CarnivoreLE execution and provides annunciation to the practitioner of the results so a decision can be made as to whether or not to run a given module if the hash verification fails. CarnivoreLE does not make moral decisions as to whether a given action is good or bad, but leaves it up to the practitioner to decide. If a utility fails hash verification, the practitioner should have articulable reasons for running the dependent module anyway.

CarnivoreLE relies on `hashdeep` in the utverify module to accomplish this. It ships with three populated reference sets – `uthashes-lin.txt`, `uthashes-mac.txt`, and `uthashes-win.txt`. These reference sets were created on known and trusted native systems (GNU/Linux, Macintosh, and Windows) and are in a format specific to `hashdeep`. To expand the sets so that other versions of the same utilities can be added, the practitioner can run `hashdeep` themselves on their own known and trusted systems that have `hashdeep` installed. `hashdeep` is available in the repositories of most major GNU/Linux distributions. Source code can be obtained from here:

http://md5deep.sourceforge.net/

The utilities to be audited are provided to `hashdeep` via one of three lists respective of each operating system – `utlist-lin.txt`, `utlist-mac.txt`, and `utlist-win.txt`. These are simply text files with the full path and filename of each utility written one per line and all three ship with CarnivoreLE. The command to generate the `uthashes-*.txt` file is:

GNU/Linux:

```
$ hashdeep -f utlist-lin.txt > uthashes-lin.txt
```

Macintosh:

```
$ hashdeep -f utlist-mac.txt > uthashes-mac.txt
```

Windows:

```
> hashdeep64.exe -f utlist-win.txt > uthashes-win.txt
```

Take precautions not to overwrite the original or last updated `uthashes-*.txt file`. It is recommended that the new results be compared with those from the respective `uthashes-*.txt` and deduplicated. Both the `uthashes-*.txt` and `utlist-*.txt` files can be found in CarnivoreLE's `config` directory. New or modified versions of these files must be placed in this directory and bear the same name as the replaced original or they will not be recognized by CarnivoreLE.

Once CarnivoreLE useage against a DUT is complete, an output report will be located in the session's evidence directory.

Return values for utverify are as follows:

0 Success
: Note that the program considers itself successful even when it encounters read errors, permission denied errors, or finds directories when not in recursive mode.

1 Unused hashes
: Under any of the matching modes, returns this value if one or more of the known hashes was not matched by any of the input files.

2 Unmatched inputs
: Under any of the matching modes, returns this value if one or more of the input values did not match any of the known hashes.

64 User error
: Such as trying to do both positive and negative matching at the same time.

128 Internal error
: Such as memory corruption or uncaught cycle.