ProcessFuzzyHash Volatility Plugin

ProcessFuzzyHash is a Volatility Plugin aimed at computing fuzzy hashes of processes in a Windows OS dump image. Fuzzy hashes are a subset of hashing functions that, contrary to other (cryptographic) hashing functions such as MD5, SHA-1, or SHA-256, try to preserve similarity between similar inputs (i.e., two similar inputs will generate a similar output). By Windows OS intrinsic characteristics, an instance of an executable file, i.e, a process, is likely to be different from other instance of the same executable. Consider for instance we want to detect whether a process dump belongs to the Google Chrome binary, i.e., "chrome.exe". If we compute the MD5 of the static file and the process dump file, they will differ. If we compute a fuzzy hash instead, the similarity between both hashes will be high (in our experiments around 90%, see webdiis.unizar.es/~ricardo/final-degree-projects/pfc/pfcs-finalizados/fuzzy-hashing-procesos). Hence, this plugin is useful to fingerprint processes in a Volatility dump. This plugin also allows the user to choose the parts of the process to be hashed. Following the Windows PE format, we allow to choose between the whole PE, the full process address space, specific PE (or section) headers, among others.

Motivation

This tool was developed by the undergraduate student Iñaki Abadía as part of the Final Degree Project of Bachelor on Computer Science at the University of Zaragoza. The goal of the dissertation was to evaluate different fuzzy hashing algorithms regarding similarities between Windows processes.
As part of the incident response phase, memory dumps are extracted from likely compromised machines and then processes hashes are calculated as a way to discard known processes from unknown ones. The hash functions usually used are cryptographic hashes (e.g. MD5, SHA, etc.). However, by the Windows OS underlying characteristics, hashes of different processes belonging to the same binary program will be completely different. Hence, fuzzy hashing algorithms become more suitable in these scenarios since they are less sensitive to minor input modifications, and thus, fuzzy hashes of different processes from the same binary program will have some similarity degree.
Besides detecting similar processes, this plugin can also be useful to identify malware samples coming from the same malware family. Malware are usually distributed in some sort of obfuscated way, normally through the use of packing, to evade signature-based detections. However, when the malware is executed its binary code is uncovered. Hence, different samples with different packings will have different hashes while samples reside on disk, but will share some similarity if fuzzy hashes are computed when malware samples are on memory (i.e., executed). Other uses of fuzzy hashing algorithms include spam filtering and copyright checking. In summary, ProcessFuzzyHash brings analysts a new innovative tool for detecting similar processes in a fast and easy manner.
In our opinion, this plugin is the best candidate for the Volatility contest since it provides a novel analysis technique using fuzzy hashing with fine-grain detail, specially focused on Windows processes. Furthermore, there not exists any other available tool similar to ours. Besides, it makes a valuable and outstanding contribution to the memory forensics field of work, since it provides analysts with a reliable tool for comparing process dump files. Hence, it makes easier the task of discarding legitimate/system processes and then put a spotlight on suspicious processes.

Description of how to use the plugin

The following code shows an execution of `ProcessFuzzyHash` with the `-h` option:

Options

```
-P: Process PID(s). Will hash given processes PIDs.
      (E.g. -P 252 | -P 252,452,2852)
-E: Process Name. Will hash process that match given string.
      (E.g. -N svchost.exe | -N winlogon.exe,explorer.exe )
-N: Process Name. Will hash processes that contain given string in the name.
      (E.g. -N svchost | -N winlogon,explorer )
-A: Algorithm to use. Aviable: ssdeep, sdhash, tlsh, dcfldd. Default: ssdeep
      (E.g. -A ssdeep | -A SSDeep | -A SSDEEP,sdHash,TLSH,dcfldd)
-S: Section to hash.
      Full process: "full"
```

```
      Full PE: "pe"
      PE section: "<pe-section>"
      PE header: "pe:< header>", "pe:header" for full header
      PE section header: "<pe-section>:header"
       (E.g. -S .text | -S .data,.rsrc | -S pe,.text:header |
             -S pe:NT_HEADERS | -S full)
-s: Hash strins instead of binary data.
-c: Compare given hash against generated hashes.
      (E.g. -c '3:elHLlltXluBGqMLWvl:6HRlOBVrl')
-C: Compare given hashes' file against generated hashes. File must contain
    one hash per line.
      (E.g. -C /tmp/hashfile.txt | -C hashfile.txt)
-H: Human readable values (Create Time)
-M: Multithreaded hashing.
-V: Keep hashed data on disk. Defaults to False.
--output-file=<file>: Plugin output will be writen to given file.
--output=<format>: Output formatting. [text,dot,html,json,sqlite,quick,xlsx]
```

As shown, the plugin allows the user to choose in a range of fuzzy hashing algorithms with a full set of personalization. The plugin activity is mainly divided into computation of hashes or comparison of hashes. Note that there are some rules regarding the use of parameters:

```
- If -P and -N provided, -N will be ignored.
- If -N and -E provided, -E will be ignored.
- Full process (-S full) and full PE/PE section cannot be hsahed at the same time.
- Supported PE header names (pefile): DOS_HEADER, NT_HEADERS, FILE_HEADER,
                                      OPTIONAL_HEADER, RICH_HEADER, HEADER
- Hashes' file given with -C must contain one hash per line.
- If -c and -C given, the comparation will be between them. No new hashes will be
generated.
- Params -c and -C can be given multiple times (E.g. vol.py (...) -c <hash1> -c
<hash2>)
```

Example: ssdeep

For the sake of space and since all fuzzy hashing behave in a similar fashion, we only consider ssdeep algorithm as running example. As told before, the plugin activity is mainly divided into 1) generation of hashes and 2) comparison of hashes. In the following, we show a trace execution example for each activity.

Generation of hashes

Consider that we are interested, for instance, in computing the hash of the full loaded PE of the Service Host process:

```
$ python vol.py --plugins=$PLUGINS_DIR -f $MEMDUMP --profile=Win7SP1x64
processfuzzyhash -A ssdeep -N svchost -S pe
Volatility Foundation Volatility Framework 2.6

Name        PID  Create Time        Section Algorithm Hash
svchost.exe 608  131(...)8750 pe      SSDeep    384:Nvv(...)YnYEbvKSK
svchost.exe 720  131(...)6250 pe      SSDeep    384:Nvv(...)TmnYEbvK/K
svchost.exe 776  131(...)3750 pe      SSDeep    384:Nvv(...)b7q+f5Tmn+0C4xUERK
svchost.exe 864  131(...)1250 pe      SSDeep    384:Nvv(...)b7q+f5T4xUEbvKOfK
svchost.exe 904  131(...)7500 pe      SSDeep    384:Nvv(...)kTYvVeZMmn+xUEbvKkOK
...
```

Now let's say we're interested in the NT_HEADERS section from Windows Login Service and Windows Explorer. Here's what we should do:

```
python vol.py --plugins=$PLUGINS_DIR -f $MEMDUMP --profile=Win7SP1x64
```

```
processfuzzyhash -A ssdeep -N winlogon,explorer -S pe:NT_HEADERS
Volatility Foundation Volatility Framework 2.6

Name         PID  Create Time        Section       Algorithm Hash
winlogon.exe 444  131500462048593750 pe:NT_HEADERS SSDeep    3:eln:el
explorer.exe 1224 131500498124726250 pe:NT_HEADERS SSDeep    3:eln:el
```

Now, consider that we want the code section from Windows Services:

```
python vol.py --plugins=$PLUGINS_DIR -f $MEMDUMP --profile=Win7SP1x64
processfuzzyhash -E services.exe -S .text:header
Volatility Foundation Volatility Framework 2.6

Name         PID Create Time        Section      Algorithm Hash
services.exe 480 131500462049062500 .text:header SSDeep    3:il8Hm9XlltFn:iSG9/
```

Comparation of hashes

Consider that we want to compare a fuzzy hash from a Service Host instance to all Service Host instances on a memory dump ($MEMDUMP):

```
python vol.py --plugins=$PLUGINS_DIR -f $MEMDUMP --profile=Win10x64_15063
>  processfuzzyhash -A ssdeep -S pe -N svchost -c '384:llc0KQICSTWvlunowG\
>  hPsqEjLS7KHhJdAWER/ZqJtKxg7AgADGBdWKEVwKlJh8xR:lMfTWvtwsbEnUYFR4Jh8JjRWv2'
Volatility Foundation Volatility Framework 2.6

Hash A                   Hash B                           Algorithm Score
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWv2  ssdeep    100
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWw2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWo2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWS2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWp2  ssdeep    98
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWK2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWc2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWn2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWt2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRW+t2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWG2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWQL2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWp2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWH2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRW0G2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWae2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWG2  ssdeep    97
```

Finally, now we want to do the same as before, but comparing with a set of hashes saved on a file from a previous execution:

```
python vol.py --plugins=$PLUGINS_DIR -f $MEMDUMP --profile=Win10x64_15063
>  -C $HASHFILE -c '384:llc0KQICSTWvlunowGhPsqEjLS7KHhJdAWER/ZqJtKxg7AgA\
>  DGBdWKEVwKlJh8xR:lMfTWvtwsbEnUYFR4Jh8JjRWv2'
Volatility Foundation Volatility Framework 2.6

Hash A                   Hash B                           Algorithm Score
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWv2  ssdeep    100
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWw2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWo2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWS2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWp2  ssdeep    98
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWK2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWc2  ssdeep    97
```

```
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWn2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWt2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRW+t2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWG2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWQL2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWp2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWH2  ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRW0G2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWae2 ssdeep    97
384:llcwGhP(...)E8JjRWv2 384:hlclunowGhP(...)EnUY8JjRWG2  ssdeep    97
```