yarp: carving fragmented registry files

Yet another registry parser, or yarp, is a library and tools to deal with Windows registry files [1]. Despite the name, yarp is not a simple registry parser.

The project started as an attempt to fully implement the Windows registry file format specification [2] and to provide features important to incident responders and forensic examiners. This article will highlight one of these features – the registry file carver (the *yarp-carver* tool). Here, the "registry file" means a primary file for a hive, an external file for a hive (a file created with the *RegSaveKeyEx* function or a similar function), or a backup file for a hive (a file created by a kernel in the *RegBack* directory), but not a transaction log file.

Carving registry files

The carver will locate, identify, and validate registry files, including fragmented ones. When a registry file is fragmented, the carver will extract the first fragment (the truncated registry file) and identify the truncation (fragmentation) point. The *yarp-print* and *yarp-timeline* tools can be used to examine a truncated registry file.

The following steps are performed by the tool when carving registry files:

1. Search for the "regf" signature at the start of each sector (since registry files are not smaller than an MFT record, the tool does not need to try every offset possible).
2. If the signature is found, read the candidate base block (4096 bytes in length) from the same offset.
3. Validate the candidate base block: the base block must contain expected and reasonable values in corresponding fields. For example, the hive bins data size field must contain a value which is multiple of 4096 bytes, but is not too large, and the file type field must indicate a primary file.
4. If the base block is valid, move 4096 bytes forward, then read the specified (in the base block) number of bytes as the hive bins data.
5. Identify the truncation point, if the registry file is fragmented:
    a) First, walk through the hive bins – all hive bins must be chained together using the offset and size fields, the first hive bin must contain the offset field set to 0, each subsequent hive bin must contain the offset value equal to the offset value of a preceding hive bin plus the size of that hive bin. A hive bin with an invalid or unexpected offset value or an invalid size (not a multiple of 4096 bytes) should be treated as a truncation point indicator. A missing hive bin should be treated as a truncation point indicator too.
    b) Next, if a truncation point indicator is found, check the cells of the last valid hive bin. Since fragmentation can take place inside the last hive bin, check the last hive bin even if no truncation point indicator has been found at the previous step. To do this, walk through the cells of a given hive bin and locate an invalid cell, if it is present: an invalid

cell is too large, or is crossing the hive bin boundary, or its size is not a multiple of 8 bytes.

c) If all cells of the last hive bin look valid (and no truncation point indicator was found before), then the registry file is not fragmented. If all cells of the last valid hive bin look valid, then the registry file is fragmented at a boundary of a hive bin (use the offset of the missing or invalid hive bin as a truncation point). Otherwise, the registry file is fragmented in the middle of a hive bin (use the offset of the invalid cell as a truncation point).

d) Since hive bins may contain large cells, there is an additional check: if cells preceding an invalid cell in a given hive bin contain the “regf” signature (indicator of another registry file) or the “hbin” signature (indicator of a hive bin from another registry file), use the location of that signature as a truncation point.

6. Adjust (round down) the truncation point according to the sector size (which is assumed to be 512 bytes), because fragmentation occurs at a cluster boundary (and a cluster contains one or more sectors).

7. Since the offset of the registry file and its real size (up to the truncation point, if fragmented) are known, the tool can copy the data to a file and skip to the next offset. If the base block contains the file name of the registry file, this file name will be used for the output file.

Carving registry fragments

The carver will locate and validate individual (freestanding) hive bins and individual sets of hive bins. These registry fragments can be examined using the *yarp-print* and *yarp-timeline* tools.

The following steps are performed by the tool when carving registry fragments:

1. Search for the “hbin” signature at the start of each sector.

2. If the signature is found, read the candidate hive bin header (32 bytes in length) from the same offset.

3. Validate the candidate hive bin header: the hive bin header must contain valid values in the offset and size fields (but the offset field is not required to be set to 0, because this hive bin could be located in the middle of a fragmented registry file).

4. If the hive bin header is valid, read and validate the cells (using the same approach as described above).

5. If all cells of the hive bin look valid, try to read the next hive bin right after the current one. If the next hive bin is valid and can be chained to the current one, treat these hive bins as a single registry fragment (and repeat this step to extend the current registry fragment). If the next hive bin is valid, but its offset field contains an unexpected value, treat that hive bin as belonging to a separate registry fragment (and repeat this step to extend the new registry fragment). If an invalid cell is found in the hive bin, treat this hive bin as the last one in the

corresponding registry fragment; identify and adjust (round down) the truncation point using the rules defined above.

6. Copy the registry fragment to a file and skip to the next offset.

Carving compressed registry files and fragments

The carver will locate, identify, and extract NTFS-compressed registry files and fragments (using the LZNT1 algorithm). NTFS-compressed registry files can be found in restore points created by Windows XP.

The following layouts of compressed clusters are supported:

- Compressed clusters belonging to the same file are stored compactly, with no space between them.
- Compressed clusters belonging to the same file are stored with gaps between runs, the slack space (one or more clusters in a gap) can be allocated by another file (the number of compressed clusters in the run plus the number of clusters in the gap after this run is expected to be equal to the size of a compression unit in clusters).

The following values are assumed:

- Compression unit size: 16 clusters.
- Cluster size: 4096 bytes.

Since the “regf” signature in the beginning of a registry file and the “hbin” signature in the beginning of a registry fragment cannot be compressed, these signatures (in their literal form according to the LZNT1 algorithm) are used to locate the compressed registry data. If compressed clusters belonging to the same file are stored with gaps between runs, the carver will also examine the corresponding slack space.

When extracting a compressed registry file, hive bins and their cells are not checked. Thus, the fragmentation status of an extracted registry file is unknown.

When extracting a compressed registry fragment, only the header of the first hive bin is checked. The fragmentation point of an extracted registry fragment is unknown. An extracted registry fragment cannot be larger than a compression unit.

Reconstructing fragmented registry files

By default, the carver will attempt to reconstruct fragmented registry files. Reconstruction of NTFS-compressed fragmented registry files is not supported.

The reconstruction process is done in two stages:

- The brute-force stage (used to reconstruct registry files consisting of 2, 3, and 4 fragments).
- The incremental stage (used to reconstruct registry files consisting of 150 fragments or less).

Brute-force stage

Each truncated registry file identified and validated by the carver before is treated as the first fragment of an original registry file and concatenated with a different number of eligible registry fragments, then a resulting candidate hive is validated. All possible combinations of registry fragments are tried until the validation process succeeds for a candidate hive (this means that the reconstruction process succeeded) or there are no more registry fragments left (this means that the original registry file could not be reconstructed).

First, it is assumed that an original registry file consisted of two fragments (and the first one is already known – a truncated registry file itself), so the reconstruction process tries to find a single registry fragment to fill the missing part. Then (if the previous step did not succeed), it is assumed that an original registry file consisted of three fragments, so the reconstruction process tries to find two remaining registry fragments to fill the missing parts: one fragment is placed in the middle of a candidate reconstructed registry file, another one – at the end of that file. Finally, it is assumed that an original registry file consisted of four fragments, so the reconstruction process tries to locate three remaining registry fragments: two fragments will be placed in the middle of a candidate reconstructed registry file, while the fourth one – at the end of that file.

A registry fragment is considered eligible if it starts with a hive bin containing an expected value in the offset field. An expected value in the offset field is calculated as a value in the offset field of the last hive bin in a preceding registry fragment plus the reported size of that hive bin (taken from its offset field). If a registry fragment is going to be used in the middle of a registry file, this fragment must not reach the hive bins data size. If a registry fragment is going to be used at the end of a registry file, this fragment should reach or even exceed the hive bins data size (to account possible remnant data in the original registry file). Here, the “remnant data” means data between the end of the hive bins data and the end of the registry file (such data may be missing from a registry file, if Windows truncated this registry file during the shrinking process, when the number of allocated hive bins is being reduced by discarding hive bins at the end of the registry file).

If a registry file being reconstructed is truncated somewhere in the middle of a hive bin, a subsequent fragment is used with a margin. A margin is data located right before the selected subsequent fragment, its size is equal to the missing part at the end of the current registry fragment. This allows the reconstruction process to deal with registry files truncated in the middle of a hive bin.

The validation process involves walking through the whole tree of keys and values (if something goes wrong, the hive is considered invalid: for example, the hive is considered invalid when a child key node does not point to a proper parent key node, or when there is no valid record in a cell at a requested offset, or when there is no valid cell at a requested offset), comparing the number of allocated unreferenced cells against a threshold (to skip hives with an implausible number of allocated unreferenced cells), comparing the size of remnant data against a threshold (to skip hives with an implausible size of remnant data).

Incremental stage

The brute-force stage becomes slower when a greater number of fragments is considered. So, there should be means of reconstructing a fragmented registry file other than trying all possible combinations of registry fragments.

The incremental stage is achieving this by successively picking the largest eligible registry fragment until the hive bins data size is reached or exceeded. First, the reconstruction process will determine an expected offset for the next registry fragment; then, a list of such registry fragments is built, the largest registry fragment is chosen from that list and appended to the truncated registry file. If the hive bins data size isn't reached at this point, the operation is repeated for the new expected offset (and a new registry fragment will be appended to the candidate registry file). If the hive bins data size is reached or exceeded, the resulting candidate registry file is validated in the same way as described above. If the hive bins data size is not reached after 149 registry fragments were applied to a truncated registry file (treated as the first fragment, so a final registry file cannot contain more than 150 fragments), the reconstruction of that registry file stops.

There is another mode of operation, which is very similar to the described above (both modes of operation are utilized by the carver). The difference is in the first iteration: when picking the second fragment (which is appended to the original truncated registry file), the second largest suitable registry fragment is chosen (instead of the largest one); other iterations remain the same.

Test results

According to the tests, approximately 10-25% of fragmented registry files can be reconstructed using the *yarp-carver* tool. This result includes registry files from allocated space and registry files that cannot be reconstructed at all (because of missing registry fragments).

It was found that a truncation point for a registry file fragmented within a large cell can be identified incorrectly using the validation method described above. The reason is that the validation algorithm does not examine the data stored in a cell. Such a large cell usually contains value data, thus a reconstructed registry file may occasionally contain corrupted value data. If such a large cell contains a subkeys list or a key values list and the registry file is truncated within active records of this list, then the reconstruction process for that registry file will likely fail.

Also, the carver cannot identify registry fragments that do not contain a hive bin header, such chunks of registry data cannot be extracted and used in the reconstruction process.

References

1. Yet another registry parser, URL: https://github.com/msuhanov/yarp.
2. Windows registry file format specification, URL: https://github.com/msuhanov/regf.
