MALWARE ANALYSIS

HEADS OR TAILS?

Peter Ferrie
Microsoft, USA

The 'flip-side' to the section replacement technique described last month (see *VB*, August 2009, p.4) is the segment alignment technique. This technique is used by a virus which was named 'Coin' by its author, and is described here.

MISPLACED TRUST

The virus begins by searching for files within the current directory. The virus attempts to open and map each file that is found. If the mapping fails, the virus closes the file without attempting to unmap anything. However, as with the other viruses that were written by the same author, this virus is very trusting of the contents of the file. The virus checks for an ELF signature and several fields within the supposed ELF header, but without checking that the file is large enough to support the presence of these fields. A sufficiently small file will cause the virus code to crash. A truncated ELF file, or a file with a sufficiently large value in the e_phnum field, among other things, will also cause the virus to crash, since the code contains no bounds checking of any kind.

MISSING THE MARK

The virus is interested in executable ELF files for the *Intel* x86-based CPU, and whose ABI is not specified. The virus does not check for an infection marker, because the marker is actually the absence of something instead of the presence of something. This will be explained below.

For each such file that is found, the virus searches within the Program Header Table entries for two PT_LOAD entries in a row, with special characteristics. The virus requires that the first PT_LOAD entry has a physical address of zero, which is the file header, and which corresponds to the image base address. The second PT_LOAD entry must have a size in the file which is equal to the size in the memory.

If a file is found to be infectable, then the virus calculates the amount of slack space that exists between the end of the first loadable segment and the start of the next page in memory. The virus also calculates the amount of slack space that exists between the start of the next page in memory and the start of the second loadable segment. The file will be skipped if the space is too small for the virus body, if the first loadable segment is aligned exactly, or if the second loadable segment is purely virtual. The alignment condition also corresponds to the infection marker. That is, when a file is infected, the first loadable segment will be aligned exactly, thus leaving no room for a virus to be inserted using this technique.

MERRY-GO-ROUND

The virus rounds up the values for the physical and virtual sizes of the first loadable segment. If the second loadable segment does not start at the start of a page, then the virus rounds down the memory offset of the second loadable segment to the start of the page. The virus increases the physical and virtual sizes of the second loadable segment by the rounding amount applied to the memory offset of the second loadable segment. It then increases the file offset of the second loadable segment by the rounding amount that was applied to the physical and virtual sizes of the first loadable segment.

If any segment entries exist after the second loadable segment, and if any of the segment entries contain a physical offset which is greater than or equal to the file size of the first loadable segment, then the virus adjusts the physical offset of the segment by adding the combination of the rounding amounts that were applied to the physical and virtual sizes of the first loadable segment, and to the memory offset of the second loadable segment. After adjusting the Program Header Table, if necessary, the virus increases the file size by the combination of the rounding amounts that were applied to the physical and virtual sizes of the first loadable segment, and to the memory offset of the second loadable segment. The idea here is that if there is already enough space in memory to hold the virus body, then it is a simple matter to create a hole in the file that is also large enough to hold the virus body.

Then the virus attempts to remap the file. The assumption is that the operation will succeed, and the variable that holds the previous mapping is overwritten by whatever result is returned. In the event of a failure to map the file, the previous mapping still exists but the virus cannot unmap it because the original pointer has been lost. This is a minor bug in the code.

If the mapping is successful, then the virus moves all of the file contents that appear after the end of the first loadable segment to a new offset. The new offset is the old address plus the combination of the rounding amounts that were applied to the physical and virtual sizes of the first loadable segment, and to the memory offset of the second loadable segment.

DESTRUCTION PHASE 1

If the Section Header Table appears after the first loadable section, then the virus adjusts the pointer to the Section

Header Table by adding the combination of the rounding amounts that were applied to the physical and virtual sizes of the first loadable segment, and to the memory offset of the second loadable segment.

If any of the section header entries has a physical offset which is greater than or equal to the file size of the first loadable section, then the virus adjusts the physical offset of the section by adding the combination of the rounding amounts that were applied to the physical and virtual sizes of the first loadable segment, and to the memory offset of the second loadable segment.

While parsing the section header entries, the virus watches for a section header entry that is named '.dtors'. The '.dtors' section contains an array of functions to call during process termination. The list is terminated by a DWORD of zero. If the virus finds a section header entry that is named '.dtors', and if the first two bytes of the tail address are zero, then the virus assumes that all four bytes are zero.

TERMINAL SERVICES

The virus wants to replace the terminator entry with the address of the virus code. Of course, it is possible to have a destructor whose address happens to be on a 64KB boundary. This would result in the lower two bytes of the address being zero. In that case, the virus will overwrite that entry instead of appending an entry to the list. This is the most potentially serious bug in the code, but the condition seems so rare that it might almost never be encountered in the real world.

Furthermore, by simply replacing the terminator with the address of the virus code, an assumption is made that another zero can be found immediately afterwards, so that the process won't crash because of a bad pointer. If the terminator entry is found, then the virus copies itself into the cavity in the file and replaces the terminator entry with the address of the virus code.

Note that if no section header entry exists that is named '.dtors', the created (and empty) cavity will still remain in the file.

CONCLUSION

When we think about cavity infection, most of us probably think of existing cavities in the file, such that an infection results in no file size increase. 'Forcing' the cavity in this way is an interesting variation on the theme, but let's hope that the virus author has finished with his file format tricks now.