MALWARE ANALYSIS

MAKING A HASH OF THINGS

Peter Ferrie
Microsoft, USA

File format tricks abound in ELF files. One of these was described in last month's issue of *Virus Bulletin* (see *VB*, July 2009, p.4). In that trick, a particular section of the file was overwritten by virus code. A variation of that technique is described here.

MISPLACED TRUST

In contrast to the 'Caveat' virus, which overwrites the '.note.ABI-tag' section of ELF files, the 'Hasher' virus (so-named by its author) is interested in the '.hash' section. The virus begins by searching for files within the current directory. When a file is found, the virus attempts to open and map it. If the mapping process fails, the virus closes the file without attempting to unmap anything.

However, the virus is very trusting of the contents of the file. The first three variants of the virus all assume that the file is in ELF format without verifying this fact. A field inside the supposed ELF header is used, without checking that the file is large enough to support the field's presence. A sufficiently small file will cause the code to crash. A truncated ELF file, or a file with a sufficiently large value in the e_shnum field, among other things, will also cause the virus to crash, since the code contains no bounds checking of any kind. The .D variant of the virus requires that a file is at least 1,024 bytes long, but this is insufficient to avoid crashes when pointers reach outside of the file.

THE MAKER'S MARK

The virus is interested in ELF files for the *Intel* x86-based CPU. At this point the .C and .D variants of the virus check whether the file is infected already, while the .A and .B variants perform this check later. The infection marker for the .C and .D variants is the last byte of the e_ident field being set to 1. This has the effect of inoculating the file against a number of other viruses, since a marker in this location is quite common. The .C and .D variants set this value in the file immediately. This has the effect of preventing the files from being examined again, in case an error occurs while infecting them. In addition, the .D variant requires that the ABI is either for *Linux* or is not specified.

For each such file that is found, the virus searches within the Section Header Table entries for the SHT_HASH entry. If the SHT_HASH entry is found, then with the exception of the .D variant, the virus checks if the section is large enough to hold the virus body. The file cannot be infected by any of the first three variants if the section is too small.

HASH COOKIES

At this point, the .A and .B variants check if the file is infected already. The infection marker for the .A variant is the number of hash buckets being set to one. This is a legal value, but it effectively disables the hashing mechanism. The infection marker for the .B variant is the first byte in the hash section being a 'push' instruction.

The hash table exists to improve the performance of locating symbols. Instead of searching linearly through the symbol table, the hash table allows the searching to be achieved using perhaps only a few comparisons. The hash table consists of an array of buckets, which is a collection of pointers whose number ideally corresponds to the number of unique hashes in the symbol table. However, the number can be made arbitrarily smaller than that, which saves space.

To find a symbol, its hash value is calculated (the hashing algorithm is published in the file format specification), and the bucket is indexed by using the hash value modulo the number of buckets. A bucket is simply a starting point for searching within a particular chain. The number of chains corresponds exactly to the number of symbols in the file. If either a bucket entry or a chain entry of zero is encountered, then the symbol does not exist in the file. In the most extreme case, the number of buckets can be set to one, in which case the entire chain might be searched for a match, as for the case where no hash table exists at all.

A HOLE IN THE BUCKET

The .A variant of the virus disables the lookup by setting the number of buckets to one, and the number of chains and the first bucket entry to zero. This corresponds to a single empty bucket, and thus no symbols. The virus code is appended immediately after the end of this new hash table, since the table is no longer usable. As a result of the change, symbol lookup no longer works for an infected file, but the file remains executable as before. The entrypoint of the file is altered to point directly to the virus code.

The .B variant of the virus alters the characteristics of the Section Header Table entry, by replacing the SHT_HASH entry with a SHT_NULL entry. As a result of the change, the hash table seems no longer to exist in the file, and thus the entire table becomes available for the virus. The virus code is placed over the top of the hash table, and the entrypoint of the file is altered to point directly to the virus code.

STASH THE HASH

The .C variant of the virus requires that the size of the .hash section is large enough to hold both the number of chains and the virus body. This would be a rare occurrence, but the virus author included the technique for completeness. If the section is large enough, then the virus reduces the number of buckets by the size of the virus body in dwords. There is a bug in this code, which is that the virus forgets to include room for at least one bucket. The new bucket number is checked against a value that is less than zero, but it should be checked against a value that is less than one. (Interestingly, the virus author included an overview document which describes the technique, and the document included an algorithm written in C which contains the correct check. It seems that the bug was introduced when the virus author ported the algorithm to assembly language.) As a result, the number of buckets can be reduced to zero, in which case a divide-by-zero error will occur when the virus is building the new bucket list. Given that a 'bucket list' is also a list of things to do before the end of one's life, this bug is rather appropriate. If the list is empty, the process dies.

If the list is valid, then the virus erases the existing hash table entirely, and creates a new one in its place. The number of chains remains the same, but the placement of the chains is altered according to the new number of buckets. For each symbol, the hash value is created, and the corresponding bucket entry (the hash value modulo the number of buckets, as described above) is examined. If the entry is empty, then the hash value becomes the bucket value. If the bucket value exists already, then the chain is walked until the end is found, after which the hash value is appended to the chain. Once the bucket list has been created, the virus body is appended to the hash table, and the entrypoint of the file is altered to point directly to the virus code.

KICK THE BUCKET

The .D variant of the virus searches the Section Header Table for the SHT_HASH and SHT_DYNAMIC entries. Both of them must exist in order for the virus to infect the file. The .D variant also requires that there are at least nine buckets in the hash table. The reason for this is because the .D variant intends to reduce the size of the hash table by 32 bytes (which corresponds to eight buckets) and because at least one bucket must exist (as described above). If the hash table contains at least nine buckets, then the .D variant reduces the number of buckets by eight, and then erases and rebuilds the hash table in the same way as for the .C variant. The size of the hash table is then reduced by 32 bytes in the Section Header Table.

Once the hash table modifications have been made, the .D variant of the virus makes further adjustments to the Section Header Table entries. The second and following sections, up to and including the hash table section, have their memory and file offsets increased by 32 bytes. The contents of those sections are also moved down in the file by 32 bytes. An implicit assumption exists here, which is that the section is legally movable. This is not the case for code and data sections, since they might contain direct references to each other which would also need to be adjusted. Thus, if the hash table appears after code or data sections, then the resulting infected file will no longer run.

Next, the .D variant of the virus examines the Program Header Table. Another assumption is made here, which is that the Program Header Table exists. If the Program Header Table does not exist, then the .D variant will crash. If any entry in the Program Header Table corresponds to one of the moved sections, then the .D variant will increase the entry's memory and file offset by 32 bytes. Also, if any entry in the dynamic segment corresponds to one of the moved sections, then the .D variant will increase the entry's memory offset by 32 bytes.

PHaT CODING

After making the appropriate adjustments to the Program Header Table, the .D variant of the virus examines the Program Header Table again. The lowest non-zero virtual address of all of the entries, and the last PT_LOAD entry, is saved for later. If the PT_PHDR entry is seen, then the .D variant increases its memory and file size by 32 bytes. Once all of the Program Header Table entries have been examined, the .D variant of the virus moves all of the sections after the last PT_LOAD entry down in the file by 32 bytes. The .D variant then inserts a new PT_LOAD entry into the newly created gap, whose file offset begins at the current end of the file. The virtual address of the entry is set to two pages below the previously lowest virtual address, taking into account the amount by which the file exceeds a multiple of four kilobytes. Two pages are required for the virus code, because even though the virus code is less than four kilobytes long, the new size of the file might exceed another multiple of four kilobytes, resulting in the virus code extending beyond the boundary of one page. The entrypoint of the file is altered to point directly to the virus code, and then the virus code is appended to the file.

CONCLUSION

The addition of a new section header is an interesting technique, since it has long been thought that files are packed too tightly for space to be found. While not ground breaking in any way, this virus does show that one should be careful about received wisdom.