MALWARE ANALYSIS

FLYING SOLO

Peter Ferrie
Microsoft, USA

Continuing his series of analyses of viruses contained in the EOF-rRlf-DoomRiderz virus zine, Peter Ferrie looks at a virus named 'Pilot'.

The term 'pilot' in the sense of a television programme can be likened to a proof-of-concept for a proposed series. A 'pilot' in the sense of computer viruses might be an appropriate term for a technique that could become common in the future. At least, that's one conclusion that might be drawn from the virus whose author named it 'Pilot'. (In fact, the virus author named it 'PiLoT', intending to refer to the PLT, as explained below.)

RESOLVE TO WORK HARDER

In the case of viruses for the *Intel* x86-based *Linux* platform, it is common to see the use of 'int 0x80' instructions to call the system functions. However, in this virus there are no 'int 0x80' instructions. Instead, the virus resolves the function addresses dynamically, in much the same way as most viruses for the *Windows* platform do.

The general principle of address resolution is to find the base address of the interesting external file (for example, kernel32.dll in *Windows* and libc in *Linux*). On the *Windows* platform, it is a trivial matter to walk a series of in-memory structures to find the one that refers to the kernel32.dll file (though the current most common method relies on an undocumented field in one of those structures, and thanks to a minor change related to that field, the technique does not work on the most recent version of *Windows*). On the *Linux* platform, some searching is required, since there is no equivalent direct pointer to the libc file.

GET IT. 'GOT' IT? GOOD.

The virus begins by examining the Procedure Linkage Table (PLT). Specifically, the virus examines the value at PLT+8. The PLT is ultimately an array of jumps to imported functions, however it contains additional instructions that are used by the linker to resolve the addresses dynamically. It begins with a push of an absolute indirect address, followed by a jump through another absolute indirect address (subsequent entries have a different format – a jump through another absolute indirect address, followed by a push of an immediate value, and ending with a relative jump to the first entry in the PLT). The first entry in the PLT jumps to the dynamic linker if its presence is required. Subsequent entries jump to the other functions used by the host process.

The source of the address for the jump is the Global Offset Table (GOT)+8. The size of the push instruction is six bytes and the address for the jump is two bytes into the jump instruction. Thus, the value at PLT+8 is an address within the GOT. The GOT is a table of pointers, and the value at GOT+8 is a pointer to the _resolve symbol, which points to the dynamic linker. If the dynamic linker is not required (because the symbols were all resolved before the process started) then the value at this location will be zero.

ELVES VS TROLLS

The virus retrieves the value at GOT+8. If the value is zero, then the virus retrieves the value at GOT+16 and trusts that this value is a pointer within the libc file. If the value at GOT+8 is not zero, then the virus page-aligns this value, and uses it as a starting point for a search within memory. The virus searches backwards in memory, page by page, looking for the dynamic linker's ELF header. The virus recognizes the header when it finds the 'ELF' signature at the start of a page, and a value that describes the file as 32-bit class, data in LSB format, and version 1 header format.

The virus contains no exception handling, so there is a risk that, depending on the section layout, a gap exists in memory between the starting location of the search and the ELF header. If such a gap exists, then the virus will cause a segmentation fault, which will cause the process to be terminated.

Once the dynamic linker's ELF header has been found, the virus searches within the Program Header Table entries for the PT_LOAD entry with the lowest virtual address and the PT_DYNAMIC entry, which the virus assumes will always exist. If the PT_DYNAMIC entry is found, then the virus is interested in its virtual address.

The virus converts the virtual address of the PT_DYNAMIC entry into a file offset, and then searches within the dynamic linking array for an entry which has the DT_PLTGOT tag. It is also assumed that this search will always be successful. The associated pointer references the GOT of another file. The virus retrieves this pointer, and then retrieves a value from within that GOT, at offset 16. This value is assumed to point into libc.

At this point, the virus performs the routine again, beginning with the search for the ELF header, and ending with the search for the DT_PLTGOT tag. The result is

that the virus recovers the required values for the libc file: a pointer to the dynamic linking array, the adjustment to convert a virtual address to a file offset, and a pointer to the GOT.

STRING THEORY

Given these values, the virus searches the dynamic linking array for the entries whose tags are DT_STRTAB, DT_SYMTAB, and DT_HASH. At last, the virus has all that it needs to resolve arbitrary symbols. The virus retrieves the addresses of the open, lseek, mmap, close, munmap, mprotect, readdir, opendir and closedir APIs, which are needed to infect files, and places the addresses on the stack. The resolution is achieved by hashing the name of the API, indexing through the bucket list (see *VB*, August 2009, p.4) to find the starting point in the list, and then comparing the names in the list until a match is found.

The virus allocates two pages of memory for itself using read/write attributes, copies itself to the first page, then changes the attributes of that page to read/execute. This allows the virus to work on systems that enforce the write^exec exclusion. That is, any given page can be writable or executable, but not both at the same time. The virus copies the API addresses from the stack into the second page, then transfers control to the first page.

I LIKE TO MOVE IT MOVE IT

In order to restore the PLT (see below), the virus changes the attributes for the page in which it exists to read/write, and does the same for the following page. By always marking two pages, despite the fact that the virus is smaller than a page, the virus does not need to worry about the offset of the PLT. Since the paging API requires an aligned base as a starting address, the virus must either place itself at exactly such an aligned address (which might require moving the PLT, and thus everything around it, too – a very complicated operation, though the virus author demonstrated that a similar thing can be done, in his Crimea virus [see *VB*, February 2008, p.4]), or the size of the marking must be increased appropriately (which is the case here) in case the PLT spans two pages. However, there is an implicit assumption here – that the PLT is no larger than 8KB, which is equivalent to 512 functions. While the vast majority of files will not import nearly as many functions, we have seen such extreme examples on the *Windows* platform. It is certainly possible that such files could exist on the *Linux* platform, too. In that case, the virus will cause a segmentation fault while rebuilding the PLT, which will cause the process to be terminated.

The virus then builds a new PLT, beginning with the second entry, by placing the indirect absolute jump, the push and the relative jump once for each of the symbols. The appropriate values for each are filled in as the PLT is constructed. After the PLT has been restored, the virus changes the attributes for the two pages to read/execute. This is a potential bug, since if the PLT did not span two pages, then the attributes for the next page might originally have been something other than read/execute. Thus, by changing the attribute to read/execute, an incompatibility might be introduced that will cause the process eventually to crash.

Finally, the virus is ready to search for files to infect.

THE MAKER'S MARK

The virus is interested in files that are at least 84 bytes long, in ELF format for the *Intel* x86-based CPU, and not infected already. The infection marker 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.

For each such file that is found, the virus searches within the Section Header Table entries for an entry that is named '.plt'. If the .plt entry is found, then the virus checks if the section is large enough to contain the first entry and the virus body. If the section is too small, then the file will not be infected, however the infection marker is not added, so such a file could be examined repeatedly in the future.

If the section is large enough, then the virus examines each of the entries in the PLT, to ensure that the addresses are arranged in increasing order. This is required because an out-of-order table cannot be reconstructed by the routine described above. If all goes well, then the virus overwrites the PLT with the virus body, and saves some important values in the code (the GOT pointer, the PLT-specific relocation-table pointer, the number of PLT entries and the original entrypoint). The virus changes the host entrypoint to point directly to the virus code, and then sets the infection marker.

CONCLUSION

As we can see, the PLT is another cavity, but not *just* another cavity. Unlike others, the contents of the PLT must be restored before the host can run. This benefits us, too – a virus cannot be heavily entrypoint-obscuring if it uses the PLT as a cavity, because the host cannot call any external functions until the PLT is restored.