Malware Engineering: Coelioxys

UC3M

Fernando Vañó

March 9, 2016

1 Description

Coelioxys is a self-replicating parasite code based on Linux systems under the x86 architecture, although it could work on x86-64 with some little adjustments. The cause of the name is the brood parasitic bees named 'Coelioxys' which lays their eggs in the nests of other bees. The semantics of this will be understood after reading the report.

Since the gcc compiler is unsuitable in order to write a computer virus (it generates a lot of initializations and other stuff in addition to the dependency of the GOT and PLT) the code is completely written in x86 assembly from scratch. In this manner the total length is 638 bytes. It is based on a prior version, far more basic, written in C. This version (the C file) is attached like a pseudo-code file, but it shouldn't be compiled because it contains incorrect and unimplemented functions... It is only an informative map or a global idea of the main flow of Coelioxys.

This virus only affects ELF binary files of the x86 architecture. When a binary is infected, it contains the parasite code at the end of the ELF file and the entry point is changed in order to execute the parasite in a fist stage. Once the job is finished, the code jumps to the original entry point address so the flow can continue and the user don't notice any strange behaviour.

The parasite flow performs the following actions:

- Get the base address of himself (as the base of some relative offsets).
- Check if it has Root permissions.
- If it has root permissions, set the *setuid* of the file `/usr/bin/vi` and jump to the original entry point.
- If not, do the following actions:
    1. Open the file `/tmp/target` with R/W permissions.
    2. Check if it is a valid ELF file.
    3. Check if it is already infected (if so, finish).
    4. Save the original entry point of the target binary.
    5. Get the 'NOTE' segment (if it hasn't one, finish).
    6. Change the header of that segment in order to allocate the parasite code. Set it as a LOAD segment and change offsets to point to the shellcode. Also set the flags of the segment as RWE. The virtual address of this page will be 0x70000000.
    7. Save a backup of the original address of himself.
    8. Change his jump address to the original entry point of the target.
    9. Rewrite the entire ELF target, adding to the end the custom parasite.
    10. Restore the entry point with the previously saved address.
    11. Finally restore the stack frame and jump to the original `_start`.

Coelioxys can be detected very easily. It is important to aware that the segment code needs to have write permissions (in addition to read and execute, of course) because in the step 8 and 10 it modifies himself before being copied to the target binary. If the segment does not have write permissions, the process would end with a beautiful segmentation fault.

The vaccine of the virus is also provided in the attachment. It is a python script that removes totally the parasite (and the padding) resulting the final binary with the same length that the original (before being infected). The only difference is that the NOTE segment is not recovered, but in the majority of cases this won't be a problem.

For analyze the scope or the effects of the parasite we have to consider that this is only a proof of concept. The code only infects a unique file (`/tmp/target`) but it can be extended in order to infect all the ELF binaries of the system (evidently this would depend on the user executing the trigger and the permissions of this user on the system) and, in that case, it could be a real threat. If we suppose this is the real effect of coelioxys, it can be seen as a Cukoo laying their eggs in other nest.

On a given misconfigured system with various users, a given user could trigger the infection and leave the system. While other binaries are infected, and other users execute any of this binaries, the infection would expand within the entire filesystem. If a user with root permissions execute one of this infected files, the text editor `vi` (installed in almost all the Linux systems by default) would contain the setuid bit activated. When the user that triggered the infection returns at the next morning, it will found their little backdoor on the `vi` program, being able to edit any file of the entire system (for example `/etc/shadow`.

To sum up, I would like to emphasize that exists a 'popular belief' that the Linux systems (or the Unix systems in general) are free of virus and this operating systems cannot be affected by this type of malware. It is true that these systems enjoys of a higher level of security mechanisms and it is far more difficult to infect or compromise a host of these SO's compared with other massively used operating systems, but we should not trust in any compiled binary, it doesn't matter the source. It is obvious that the open source code is the real solution, if security is our quest.