Obligatory exercise #4

Jannis Schaefer (120914)

October 8, 2014

Task

- Find vulnerabilities in the SecureDesktop application. You find source code and an article in the repository.
- The source code has not been published, so you must not share it with people outside this course.
- What is your strategy for the review?
- How do you prioritise vulnerabilities found?
- How would you modify the application to remove the vulnerabilities?
- Document your process so that your results can be reproduced.

Solution

1 Strategy

Since I am working alone on this project, I will focus mainly on using tools for automated code analysis. In theory, this should reveal the most vulnerabilities in the time available to me.

After using the tools I will use the output to help me find some inputs the program takes from outside and trace them to estimate whether or not it may be possible to inject something.

2 Code analysis using tools

I identified Pascal Analyser [1] as a relevant tool, discarding many others for either being solely useful for metric analysis or not being able to work with partial source code. I was however unable to access full output due to only using an evaluation copy. Most critically for my usage was that report of critical errors seemed to be turned off.

Two warnings the tool spat out were two interfaced identifiers that were used but not outside of the unit (`ApplicationDirectoryName`, `BackgroundBitmapFileName`).

It further complained that many functions were called as procedures (for example `fmSDAppInfo.pas`, line 148).

In `SDInfoProcesses.pas`, there is an empty `begin`...`end`-block. The code seems to have been written for logging purposes, but only the call to the `Log`-function has been commented out, not the corresponding `if`-statement.

3 Manual code review

The first thing which I noticed was the lack of documentation and comments in the code. It is downright impossible to make good judgements of what the author(s) tried to achieve and whether or not they made any semantic mistakes.

I started tracing some of the strings which were mentioned in the warnings, and saw that `BackgroundBitmapFileName` indicates that one might try to inject an image file which is too large (or maybe too small) to achieve an buffer (under-/) overflow.

In `SDCommon.pas` in line 111, the %APPDATA% path of the specific user "hannol" is hard coded, which I suspect might result in unexpected behaviour in a production system.

The program does also interact with external binaries, which may well be compromised. I am however uncertain why most of those are interacted with and therefore reluctant to propose improvements.

I saw several examples of hard coding of values which were used multiple times in the program. Those should be made into constants to facilitate easier refactoring of code and to avoid typing mistakes.

4 Conclusion

My work was mostly hindered by not having access to the entire source code or compiled binaries and the total lack of commenting and documentation in the source code. I perceived it as a challenge to assess what the intention of vast parts of the code where. I also suspect that many of the choices of external files referenced were used for testing purposes, as this version of the source seems to be.

I could not identify any vulnerabilities which I would be able to exploit right away. But I suspect that some interactions between the code I was able to look at and the code I had no access to lead to possible exploits. I also believe there to be no specific checks to identify external entities before accessing them.

Bibliography

[1] Peganza. Pascal analyzer. `http://www.peganza.com/products_pal.htm`.