Obligatory Assignment #4:

Brage Celius - 120920

8. oktober 2014

1 Strategy

For this assignment I have decided on a strategy of approach:

1. Get an overview of the code and read the article.
2. Search for, and use, static code analysis tools.
3. Interpret the results and present solutions.
4. Search the code manually to locate more possible vulnerabilities.

2 Overview of the code

The code is written in Pascal/Delphi and contains three main Delphi Project files(.dpr). In addition, there are 11 Pascal Unit files, 1 Resource Script, 1 manifest file and a readme.txt.

3 Tools Used

The source code is written in Pascal/Delphi and tools for analyzing this language were not abundant. However i have found some tools that I use in this assignment:

- ICARUS
- Code Healer
- Pascal Analyzer (trial version)

3.1 Icarus Results

ICARUS gave me a general overview of the code. By analyzing the results given by ICARUS it becomes clear that a lot of source code is not available to us.

```
Module SDCommon uses:

  Units used in implementation:

       DateUtils source not found
       IniFiles source not found
       SysUtils source not found
       ShlObj source not found
       ActiveX source not found
       KnownFolders source not found
       AccCtrl source not found
       AclApi source not found
```

Figur 1: Snippet of results, ICARUS run on SDUserSessionManager.dpr

3.2 Code Healer Results

For some reason Code Healer did not would not let me open any of the ".dpr" files when creating a new project. The "New Project Wizard" did not see them as valid .dpr files and would not let me continue. However, i tried disabling the "New Project Wizar" and import the files manually. This way I was able to open the files. Running an analysis on the imported files gave me a lot of warnings of files not being available. I tried to ignore all these errors as the specified files were unavailable to us, but the end result was that Code Healer exited with parsing errors. See figure .

Analysis results

Check description	Line	Column	Module
Check failures (0) * Parsing errors *			
Parsing errors (3)			
Unable to find or open unit 'System'	1	18	SDAppInfo
Unable to find or open unit 'System'	1	27	SDSecureDisplaySvc
Unable to find or open unit 'System'	1	29	SDUserSessionManager

Figur 2: Snippet of results, Code Healer run on all .dpr files.

3.3 Pascal Analyzer Results

Pascal Analyzer was by far the best tool among the three i tried. It provided detailed information about the source code, and sorted them in categories making it easy to interpret. Running Pascal Analyzer on all the Delphi Project Files did not return any major vulnerabilities. It did return some warnings, like unreferenced identifiers and scope of variables and public functions. As best practice,

variable scopes should be defined as narrow as possible and functions should be private unless there is a good reason for them to be public. However, these can not be necessarily concluded to be true as there are bits of the source code not available to us. I will not address these issues further as they are not security vulnerabilities.

4 Manual Search of the code

Manually analyzing the entire code is a huge task, especially since it is poorly commented. Looking through the entire code would require too much work, therefore I have selectively searched the code based on results from Pascal Analyzer and interesting keywords.

Lack of Commenting

The thing that struck me first was the lack of commenting throughout the code. This could pose a problem for the ones maintaining or further developing the code, making it time consuming or in the worst case introducing vulnerabilities that could have been avoided.

Executables and hard coded paths

I searched the source code for lines containing ".exe" or ":\" and found that the application executes both .exe files that are not originally part of the application, like "calc.exe" and "notepad.exe", as well as .exe files that are part of the application like "SDAppInfo.exe". It does so through hard coded paths. An attacker could potentially imitate or alter these files, and thereby execute unwanted programs. To solve this problem you could implement keys or other form of authentication between programs or integrity checks. Also, if the specified external .exe file is missing for some reason, this would prevent proper execution of the program. To solve this, one could implement internal versions of the executables that are provided with the software.

5 Conclusion

Using Pascal Analyzer i was able to find several coding practices that should be improved upon, like unreferenced variables and public functions that could be private. However these errors might be false positives as bits of the source code are commented out and there are pieces being referenced that are not included in our version of the source code. Furthermore, commenting code is critical in regards to maintenance and should be greatly improved upon in this source code. Additionally, hard codes values can pose a security risk, especially when they are executables that are easily replaced or altered. These values should be made dynamic if possible, or if not there should be some kind of authentication or integrity check upon execution of the file. One possible solution is to alter the

external executables into versions that authenticate themselves to the program that is executing them.

Referanser

[1] Peganza
http://www.peganza.com/index.htm

[2] Pascal Analyzer Documentation
http://www.peganza.com/Files/PAL.PDF

[3] Code Healer Group
http://www.socksoftware.com/

[4] Microsoft about .rc files.
http://msdn.microsoft.com/en-us/library/windows/desktop/aa380599(v=vs.85).aspx