Gjøvik University College

Software Security

Assignment #4

Code Review

Victor Rudolfsson - 120912

October 8, 2014

1 Strategy

As usual when I come across code, the first thing I want to find out is whether the code is operational. As such I proceeded to attempt compiling the code using Lazarus and *fpc*, which lead to a dead end.

I realized I needed to take another approach, and proceeded with static code analysis using Pascal Analyzer, however ... Delphi being a proprietary language, many of the tools used in conjunction with it seem to be as well - this was the case with Pascal Analyzer. Other tools I attempted to use was CodeHealer (which provided no information I considered useful in this case) and SourceMonitor (which showed a large amount of global variables – this may turn out an issue).

I got a decent idea of the code structure, but with files missing and the code being incomplete, I may only be touching on the tip of the iceberg. Especially with Pascal Analyzer not *"showing complete results"* as it was only an evaluation version, I realized I had to proceed going through the code.

2 Code Review

As interesting as the PDF about SharkCage was, this code had me stumped. Going through code that makes use of a third party library I'm very unfamiliar with, written in a language I'm not only unfamiliar with but have never had a look at before proved rather difficult and confusing. As such, the vulnerabilities I may have stumbled upon I'm not entirely sure of - especially since I could not compile the program and test my theories.

I had a hunch that there would be a buffer overflow somewhere in the code, it felt like with so many *GetMem()* calls, it felt like it would be highly relevant. I could not, however, locate one with certainty (partly because I am too unfamiliar with Delphi, and Pascal).

However, one of the flaws I (may have) found was that the program executes external .exe's without verifying their integrity (i.e via a signature or hash digest) at any point. Although it assumes these .exe files to be located in the same directory as the executing

program, if the integrity of one of these executables is compromised, these will be run in a privileged state – a malicious attacker could in such a scenario place an executable that runs malicious code and then passes the arguments onto the original (now renamed) executable, and running in a privileged mode, it may get access to the newly created desktop and monitor user behavior while having the user believe he's running in a secure mode. This could be resolved by verifying the integrity of these files at least once, and abort if these files have been modified.

Another potential vulnerability is in the *ConfigDirectory* function on line 111, in *SDCommon.pas*, where a failure of KnownFolder causes a fallback to a hardcoded path doesn't exist on all computers. Perhaps by causing such a failure, one could create this location, causing a world-writeable configuration folder where an attacker could freely modify the configuration. Using a dynamic fall-back path (in this case, *%APPDATA%*) would have resolved this issue; although this path would be writeable regardless.

One such configuration option, would be the URL to the SOAP server (also found in SDCommon.pas on line 123, in the SDServiceRequestURL function). Making sure vital configuration options cannot be changed by outsiders could be done by placing the configuration in a path not accessible by unprivileged users.

Furthermore, as I am not entirely sure what checks is done when the logo image for an application is shown, if an extremely large file named with the application key (e.g *LogoAPPKEY.4d.png*) is found that is in reality a stream, or a very large file, this may cause a crash or an overflow.

Finally, it would have been very helpful if this code at least was commented better (maintenance "vulnerability"? ;)), and perhaps had less verbose logging (or made this optional) to prevent too much insight in it when it actually is compiled and run. I feel like I, not knowing Delphi, could have got more information from this program when it's actually running than looking through the code.