Mandatory assignment 3 Software vulnerability

- AV: Jan Petter Berg Nilsen
- Studnr: 120505

Preface

This is an edited version of assignment 3, Software vulnerability. When I was about to send in the assignment I had some errors in my latex code. where my references should have been. So to get it sent inn at the right time, I just deleted the paths that gave me the errors, so I got the file to compile. I have talked to Hanno about this and he said "Kan du rette opp og lagre en fullstendig versjon i dag?" and in English for the non Norwegian reviewer, that would be "Can you correct and save a complete version today?". So now I am doing that. I also did one more round with spell check to correct my dyslexic errors, because there was a lot of them. This is so it would be easier to read and understand for you, the reviewer.

Intro

This is the third mandatory assignment in IMT3501 Software security. In this assignment we are going to look at some pascal source codes to find possible vulnerabilities, that potentially could be exploited by an attacker.

Secure Desktop applications source code, revolves around the article Shark Cage. I read the article before I started to do the analysis of the source code, so I was sure that I understood what the program was for and how it worked. It helped me understand the code a little bit more, even though it wasn't a good documentation of the program and I have never seen or programmed in pascal before.

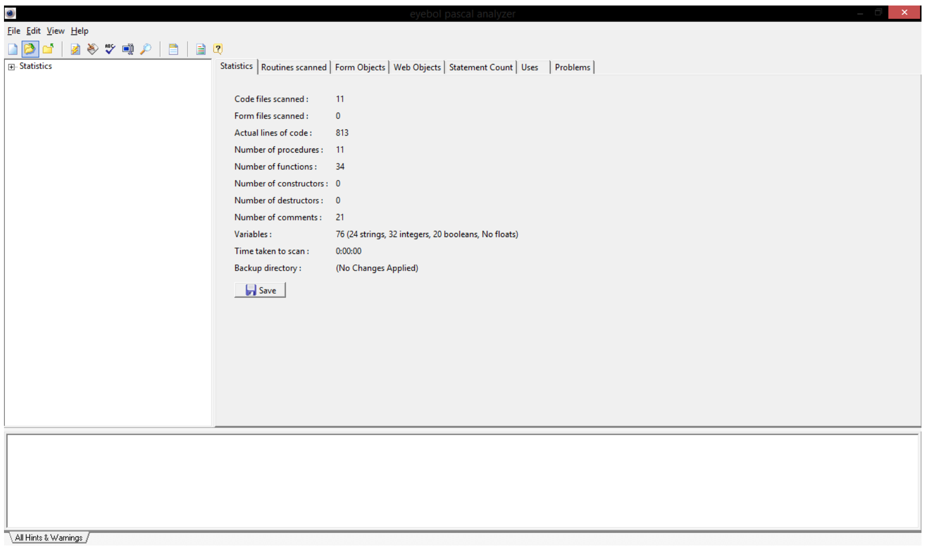
The pascal source code did not have much commenting in the code, so that made it harder to understand the code and to fine vulnerabilities for the applications. From this I could say that this is a potential vulnerability, because it would mean that the security scan of the code would not be optimal.

When doing the security scan for this application, I tried to follow the Microsoft guideline for a security analysis [1]. First I did an automatic scan where I used a variety of tools to scan the code, before I went through all the files in the repository, manually, to see if I found any vulnerabilities that way. The different tools that I used and possible vulnerabilities that I found, is listed below.

Tools

The different tools I used to analyse the code, found mainly errors which I don't see as a potential threat or vulnerability, but only errors that can be corrected to optimize the codes and the application. The reason why I didn't find any big errors with the tools, could be that I only used the free versions and it would most likely show more bigger errors if I had the paid edition. When I looked up the errors after the analysis, I also only got to see 2 or 1 results, depending on the tool. Here are some of the tools that I tried to gather information from, about the source code:

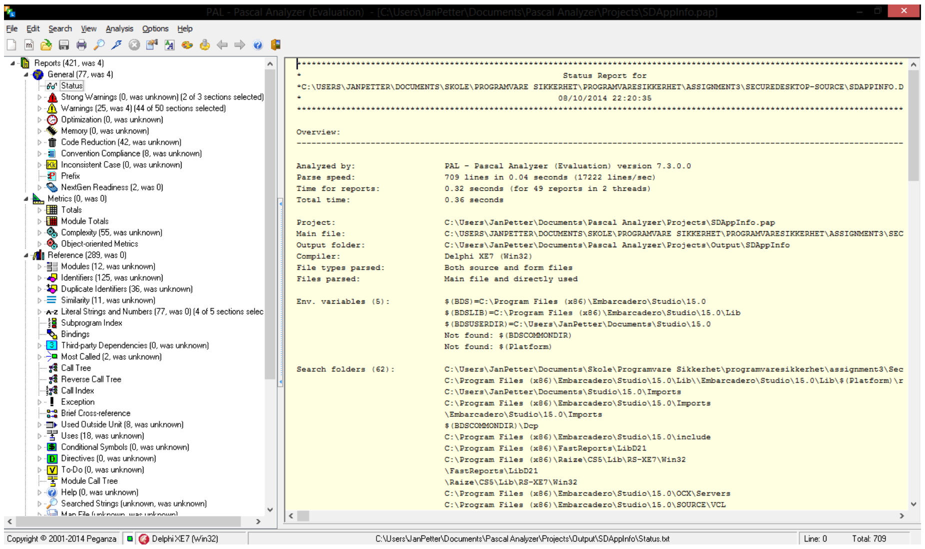
Eyebol is an analysing tool that helps finding errors and applying optimizations to the code, but also generating other information about the source code. Sadly this tool didn't give me many results, because I used the free version[2].



```
Code files scanned :       11
Form files scanned :       0
Actual lines of code :     813
Number of procedures :     11
Number of functions :      34
Number of constructors :   0
Number of destructors :    0
Number of comments :       21
Variables :                76 (24 strings, 32 integers, 20 booleans, No floats)
Time taken to scan :       0:00:00
Backup directory :         (No Changes Applied)
```

Figure 1: Eyebol

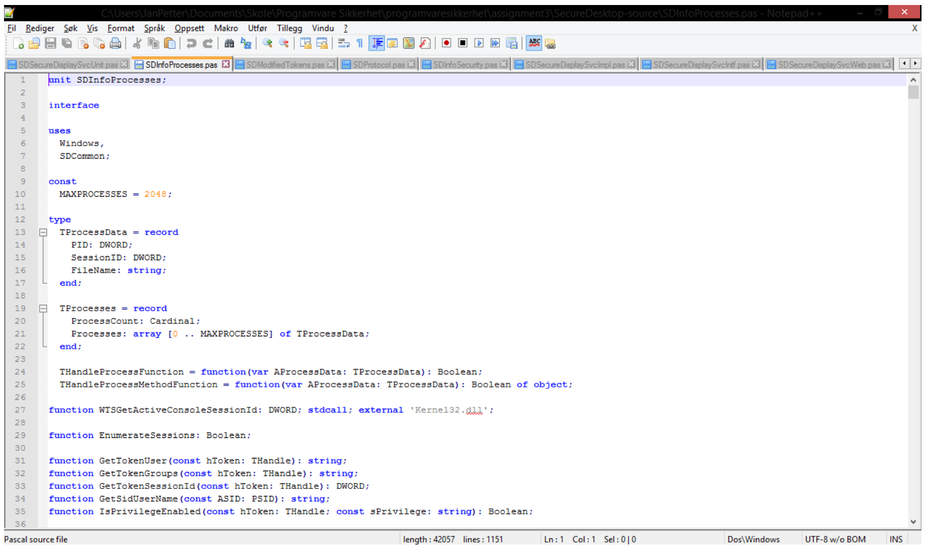
Pascal analyse 7 is a tool like Eyebol, but so much better, even when I used the free version from both of them. Pascal analyse, spots many types of programming bugs and anomalies. In my case it only gave me minor errors. Pascal analyse also generate a lot of information about the source code. Among the analysing tools, this was the best one, gave most results and was very easy to use[2].


PAL - Pascal Analyzer (Evaluation) - [C:\Users\JanPetter\Documents\Pascal Analyzer\Projects\SDAppInfo.pap]
File Edit Search View Analysis Options Help
Reports (421, was 4)
General (77, was 4)
Status
Strong Warnings (0, was unknown) (2 of 3 sections selected)
Warnings (25, was 4) (44 of 50 sections selected)
Optimization (0, was unknown)
Memory (0, was unknown)
Code Reduction (42, was unknown)
Convention Compliance (8, was unknown)
Inconsistent Case (0, was unknown)
Prefix
NextGen Readiness (2, was 0)
Metrics (0, was 0)
Totals
Module Totals
Complexity (55, was unknown)
Object-oriented Metrics
Reference (289, was 0)
Modules (12, was unknown)
Identifiers (125, was unknown)
Duplicate Identifiers (36, was unknown)
Similarity (11, was unknown)
Literal Strings and Numbers (77, was 0) (4 of 5 sections selec
Subprogram Index
Bindings
Third-party Dependencies (0, was unknown)
Most Called (2, was unknown)
Call Tree
Reverse Call Tree
Call Index
Exception
Brief Cross-reference
Used Outside Unit (8, was unknown)
Uses (18, was unknown)
Conditional Symbols (0, was unknown)
Directives (0, was unknown)
To-Do (0, was unknown)
Module Call Tree
Help (0, was unknown)
Searched Strings (unknown, was unknown)
Copyright © 2001-2014 Peganza
Delphi XE7 (Win32)
C:\Users\JanPetter\Documents\Pascal Analyzer\Projects\Output\SDAppInfo\Status.txt
Line: 0
Total: 709

```
***************************************************************************************************************
*                                                    Status Report for
*C:\USERS\JANPETTER\DOCUMENTS\SKOLE\PROGRAMVARE SIKKERHET\PROGRAMVARESIKKERHET\ASSIGNMENT3\SECUREDESKTOP-SOURCE\SDAPPINFO.D
*                                                   08/10/2014 22:20:35
***************************************************************************************************************

Overview:
---------------------------------------------------------------------------------------------------------------

Analyzed by:                 PAL - Pascal Analyzer (Evaluation) version 7.3.0.0
Parse speed:                 709 lines in 0.04 seconds (17222 lines/sec)
Time for reports:            0.32 seconds (for 49 reports in 2 threads)
Total time:                  0.36 seconds

Project:                     C:\Users\JanPetter\Documents\Pascal Analyzer\Projects\SDAppInfo.pap
Main file:                   C:\USERS\JANPETTER\DOCUMENTS\SKOLE\PROGRAMVARE SIKKERHET\PROGRAMVARESIKKERHET\ASSIGNMENT3\SEC
Output folder:               C:\Users\JanPetter\Documents\Pascal Analyzer\Projects\Output\SDAppInfo
Compiler:                    Delphi XE7 (Win32)
File types parsed:           Both source and form files
Files parsed:                Main file and directly used

Env. variables (5):          $(BDS)=C:\Program Files (x86)\Embarcadero\Studio\15.0
                             $(BDSLIB)=C:\Program Files (x86)\Embarcadero\Studio\15.0\Lib
                             $(BDSUSERDIR)=C:\Users\JanPetter\Documents\Studio\15.0
                             Not found: $(BDSCOMMONDIR)
                             Not found: $(Platform)

Search folders (62):         C:\Users\JanPetter\Documents\Skole\Programvare Sikkerhet\programvaresikkerhet\assignment3\Sec
                             C:\Program Files (x86)\Embarcadero\Studio\15.0\Lib\\Embarcadero\Studio\15.0\Lib\$(Platform)\r
                             C:\Users\JanPetter\Documents\Studio\15.0\Imports
                             C:\Program Files (x86)\Embarcadero\Studio\15.0\Imports
                             \Embarcadero\Studio\15.0\Imports
                             $(BDSCOMMONDIR)\Dcp
                             C:\Program Files (x86)\Embarcadero\Studio\15.0\include
                             C:\Program Files (x86)\FastReports\LibD21
                             C:\Program Files (x86)\Raize\CS5\Lib\RS-XE7\Win32
                             \FastReports\LibD21
                             \Raize\CS5\Lib\RS-XE7\Win32
                             C:\Program Files (x86)\Embarcadero\Studio\15.0\OCX\Servers
                             C:\Program Files (x86)\Embarcadero\Studio\15.0\SOURCE\VCL
```

Figure 2: Pascal analyse 7

When doing the manual searching process for vulnerability I used notepad++[?].



```
unit SDInfoProcesses;

interface

uses
  Windows,
  SDCommon;

const
  MAXPROCESSES = 2048;

type
  TProcessData = record
    PID: DWORD;
    SessionID: DWORD;
    FileName: string;
  end;

  TProcesses = record
    ProcessCount: Cardinal;
    Processes: array [0 .. MAXPROCESSES] of TProcessData;
  end;

  THandleProcessFunction = function(var AProcessData: TProcessData): Boolean;
  THandleProcessMethodFunction = function(var AProcessData: TProcessData): Boolean of object;

function WTSGetActiveConsoleSessionId: DWORD; stdcall; external 'Kernel32.dll';

function EnumerateSessions: Boolean;

function GetTokenUser(const hToken: THandle): string;
function GetTokenGroups(const hToken: THandle): string;
function GetTokenSessionId(const hToken: THandle): DWORD;
function GetSidUserName(const ASID: PSID): string;
function IsPrivilegeEnabled(const hToken: THandle; const sPrivilege: string): Boolean;

```

Figure 3: notepad++

I also tried other types of tools, like Pascal Browser 2 and Icarus [2] [3]. They gave some information about the source code, but I feel that the other tools mentioned above, helped me more to analyse the different source codes. Other tools that I tried but gave me error, and did not work with the source codes, were Codehealer and DUnit Xtreme Unit Testing for Delphi[4] [5]. For the Delphi I got the error that it was missing files from the code, so this could mean that we didn't get all the source files for the application.

Vulnerability

I didn't find any big vulnerabilities for either doing the automatic analysis or the manual analysis. The manual analysing process was very hard, because it was many lines of code, was poorly commentated and wasn't very well documented. As mentioned in the intro, I would say that bad commenting in the code is no good practise and would make a vulnerability analysis much more difficult, so that it would be easier to over see some potential vulnerabilities. That way the analysis of the source code would not be as optimized as it could be. When the code is poorly commentated it would also mean that it is harder to maintain, and potential vulnerabilities could occur.

Here are some examples of errors that I found, when doing the automated search with the different tools:

- Redeclared identifiers from System unit in the files fmSDAppInfo line 199.
- Variables that are set, but never referenced in file SDInfoProcesses line 625 and 626.
- Interfaced identifiers that are used, but not outside of unit in file SDCommon line 15 and 23.
- Redeclared identifiers from System unit in SDUserSessionManager line 34.

Most of these errors could easily be fixed and I really do not look at them as a type of vulnerability. But they could also occur because of some missing files, that I talked about earlier in this rapport, and also could be cleared up with better documentation.

When I did the manual search, I only found that the program used some hard coded values and paths, like it does on the file SDcommand.pas on line 111. This could be used to manipulate the program and make it crash. This could be fixed by using const/constant values in the program.

I also see that some parts of the source code, like SDInfoSecurity.pas on line 200, execute exe and other file types that are not a part of the original program. This could mean that someone could manipulate the program to run malicious software, that could potentially be a vulnerability. What we can do to decrease the probability of an attack like this to occur, is to make some sort of a checking code or authentication, like a sort of code that is implemented to check the exe file before it starts running. This will only make sure that the right executable files can execute inside the program.

Bibliography

[1] Microsoft: Perform a Security Code Review for Managed Code, http://msdn.microsoft.com/en-us/library/ff649315.aspx

[2] Analysis tools: Pascal analyse 7, Eyebol and Pascal Browser 2, http://www.torry.net/pages.php?id=50

[3] Analysis tool: Icarus, http://delphi.about.com/od/devutilities/ss/icarus-delphi-unit-uses-clause-analyzer.htm

[4] Analysis tool: Codehealer, http://www.socksoftware.com/codehealer.php

[5] Analysis tool: DUnit Xtreme Unit Testing for Delphi, http://dunit.sourceforge.net/

[6] Manually analysis tool: Notepad++, http://notepad-plus-plus.org/