IMT3501 Software Security

Secure Desktop

Obligatory assignment # 3, 12HBISA

Anders Storsveen,120928

8. oktober 2014

Chapter 1

Introduction

This assignment consist of finding vulnerabilities in the SecureDesktop application. The objectives are as following:

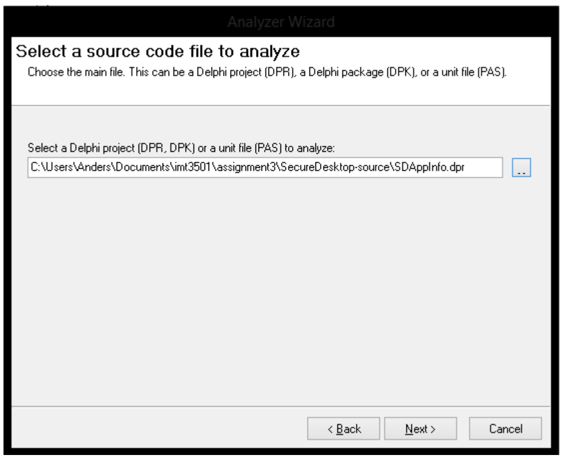
- What is your strategy for the review?
- How do you prioritise vulnerabilities found?
- How would you modify the application to remove the vulnerabilities?

After spending several hours online searching for and trying out numerous applications that could analyze pascal code, I decided to use the first program that I found. PAL - Pascal Analyzer from Peganza [1]. PAL is a utility program that analyzes, documents, debugs, and help optimize source code. It needs only the source code of the application for performing the analysis, unlike many other applications that analyzes a running program. PAL comes in two different modes; PAL.EXE with a GUI and PALCMD.EXE for a command-line analyzer which is useful when you want to automate the process of creating reports. Both processes uses the same engine and produces the same output. I used mostly the GUI version, but I also tried out the command-line version briefly. PAL functions with Pascal/Delphi Compilers from BP7 and later. I tried two different compilers, but at first glance i did not register huge differences. Therefore I used only the Delphi XE7 compiler for Win32 in my analysis.

> *PAL parses your source code in the same way as the compiler. It builds large data tables in memory and when the parsing is finished, produces an assortment of reports. These reports hold plenty of useful information that can help you error-proof your applications* [2].

1.1 Beginning a PAL-project

To analyze a set of source code with Pascal Analyzer, you must first create a project. When you start Pascal Analyzer for the first time you are presented with a wizard to help you get started. The wizard tells you, in fig. **??**, to select a source code file to analyze. This can be a Delphi project(DRP), a Delphi package (DPK), or a unit file(PAS). When you choose a Delphi project file, you can later include all files that is referenced within this project. Next you have to choose which compiler you want to use. Fig. **??** shows how you include the rest of the source code used in the analysis. Finally you choose where you want to save the output and are presented the window shown in fig. **??**. To view the source code directly in Pascal Analyzer, you can press F8. This is helpful because you can double click on any warning and be directed to the place in the source code this warning applies. This source window is for viewing only. At the bottom right of the source viewer, there are two navigation buttons. Use these buttons to go back or forward through the list of source locations.



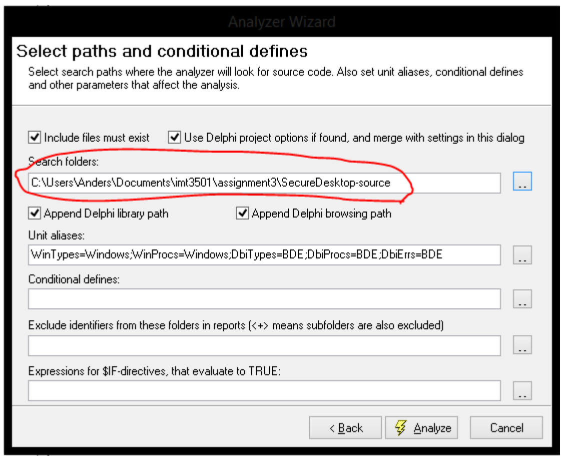
Select a source code file to analyze

Choose the main file. This can be a Delphi project (DPR), a Delphi package (DPK), or a unit file (PAS).

Select a Delphi project (DPR, DPK) or a unit file (PAS) to analyze:

C:\Users\Anders\Documents\imt3501\assignment3\SecureDesktop-source\SDAppInfo.dpr

Figur 1.1: Analyzer Wizard: select main source file



Select paths and conditional defines

Select search paths where the analyzer will look for source code. Also set unit aliases, conditional defines and other parameters that affect the analysis.

[✓] Include files must exist [✓] Use Delphi project options if found, and merge with settings in this dialog

Search folders:

C:\Users\Anders\Documents\imt3501\assignment3\SecureDesktop-source

[✓] Append Delphi library path [✓] Append Delphi browsing path

Unit aliases:

WinTypes=Windows;WinProcs=Windows;DbiTypes=BDE;DbiProcs=BDE;DbiErrs=BDE

Conditional defines:

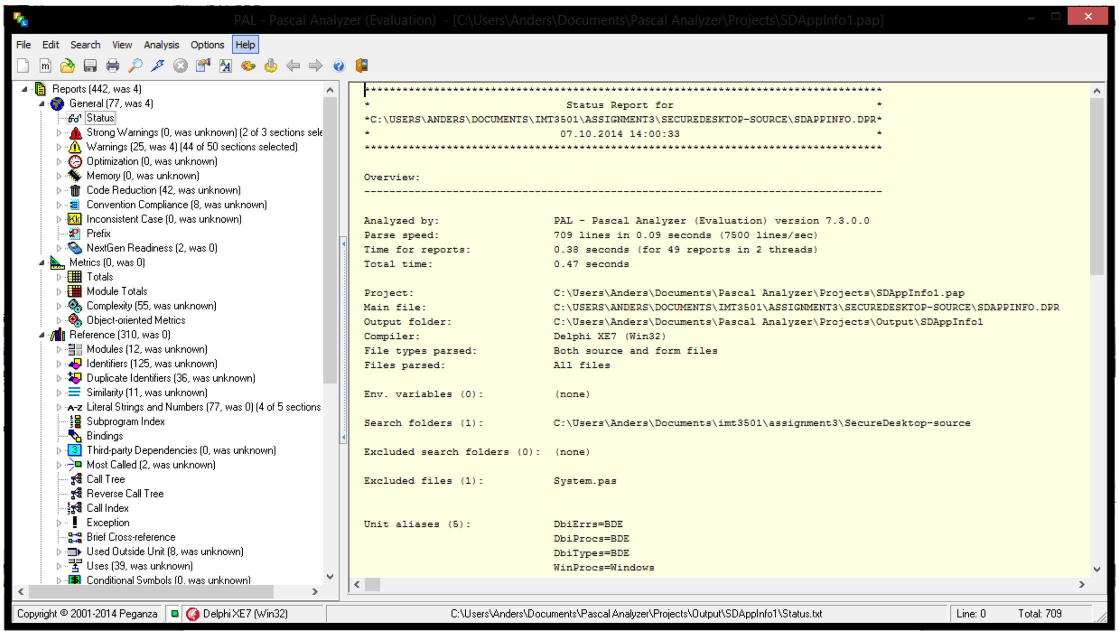
Exclude identifiers from these folders in reports (<+> means subfolders are also excluded)

Expressions for $IF-directives, that evaluate to TRUE:

Figur 1.2: Analyzer Wizard: include source files

1.2 Multi-projects

It is possible to combine multiple Pascal Analyzer projects (.pap-files) into one project(.pam-file). The includes projects are analyzed sequentially and the report contain mutual facts about these projects. However, it does not seem to include as detailed information as when I'm analyzing a single .dpr file for itself. The warning report only contains *Interfaced identifiers that are used, but not outside of unit* and *Interfaced class identifiers that are public/published, but not used outside of unit.* Because I'm using the Evaluation version of PAL, I can only read two entries in each category.


PAL - Pascal Analyzer (Evaluation) - [C:\Users\Anders\Documents\Pascal Analyzer\Projects\SDAppInfo1.pap]
File
Edit
Search
View
Analysis
Options
Help
Reports (442, was 4)
General (77, was 4)
Status
Strong Warnings (0, was unknown) (2 of 3 sections sele
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 (310, 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
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 (39, was unknown)
Conditional Symbols (0, was unknown)
Copyright © 2001-2014 Peganza
Delphi XE7 (Win32)
C:\Users\Anders\Documents\Pascal Analyzer\Projects\Output\SDAppInfo1\Status.txt
Line: 0
Total: 709

```
**********************************************************************************
*                              Status Report for                                 *
*C:\USERS\ANDERS\DOCUMENTS\IMT3501\ASSIGNMENT3\SECUREDESKTOP-SOURCE\SDAPPINFO.DPR*
*                             07.10.2014 14:00:33                                *
**********************************************************************************

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

Analyzed by:                 PAL - Pascal Analyzer (Evaluation) version 7.3.0.0
Parse speed:                 709 lines in 0.09 seconds (7500 lines/sec)
Time for reports:            0.38 seconds (for 49 reports in 2 threads)
Total time:                  0.47 seconds

Project:                     C:\Users\Anders\Documents\Pascal Analyzer\Projects\SDAppInfo1.pap
Main file:                   C:\USERS\ANDERS\DOCUMENTS\IMT3501\ASSIGNMENT3\SECUREDESKTOP-SOURCE\SDAPPINFO.DPR
Output folder:               C:\Users\Anders\Documents\Pascal Analyzer\Projects\Output\SDAppInfo1
Compiler:                    Delphi XE7 (Win32)
File types parsed:           Both source and form files
Files parsed:                All files

Env. variables (0):          (none)

Search folders (1):          C:\Users\Anders\Documents\imt3501\assignment3\SecureDesktop-source

Excluded search folders (0): (none)

Excluded files (1):          System.pas


Unit aliases (5):            DbiErrs=BDE
                             DbiProcs=BDE
                             DbiTypes=BDE
                             WinProcs=Windows
```

Figur 1.3: Pascal Analyzer: status report

Chapter 2

Analysis

Since I only have access to the free evaluation version of Pascal Analyzer, I can only see two entries in each category in the results. But it still give an indication of what kind of vulnerabilities and errors the code may contain. I will present the results for each Delphi project, even though many of the results overlap when the same files are included in more than one project. All definitions and recommendations for each entry is from peganza.com/palhelp [3].

2.1 SDAppInfo.drp

Loaded files include: dmScreenshot.pas, fmSDAppInfo.pas, SDAppInfo.dpr and SDCommon.pas

2.1.1 Warnings:

- **Interfaced identifiers that are used, but not outside of unit:**

  ApplicationDirectoryName : UnicodeString Typed const, Interfaced SDCommon (15)

  BackgroundBitmapFileName : UnicodeString Typed const, Interfaced SDCommon (23)

  This section lists all identifiers that are declared in the interface section of a unit, and that are used in the unit, but not outside the unit. You should declare these identifiers in the implementation section of the unit instead.

  Restrictions: Interfaced identifiers that are not used at all are not listed. These identifiers are already listed in the "Identifiers never used" section in the Code Reduction Report.

  Recommendation: Declare these identifiers in the implementation section of the unit, to avoid unnecessary exposure.

- **Interfaced class identifiers that are public/published, but not used outside of unit:**

  GetBitmap Func, Method dmScreenshot\TScreenshot (15)

  imgApplicationLogo : Simple (unknown) ClassField fmSDAppInfo\TfmAppInfo (16)

  This section lists all identifiers that are members of a class, and are declared with the public/published directive, but not used outside of the unit. Recommentation: Declare these identifiers with the private/protected directive instead.

- **Functions called as procedures:**

  fmSDAppInfo.TfmAppInfo.DisplayAppInfo at fmSDAppInfo (134)

  fmSDAppInfo.TfmAppInfo.RemoveAppBar at fmSDAppInfo (148)

  This section lists locations in the source code where functions are called as procedures, that is without using the result value. Maybe this is a coding error, and the function should really be called as a function instead.

- **Redeclared identifiers from System unit:**

  PI : Simple (unknown) Var, Local fmSDAppInfo\TfmAppInfo\StartApp (199)

  This section lists identifiers that use the same name as an identifier from the System.pas unit for the compiler target. Although allowed, at least it is a source for confusion when maintaining the code.

2.1.2 Code Reduction:

- **Identifiers never used:**

  CloseTimer : Simple (unknown) ClassField fmSDAppInfo\TfmAppInfo (11)

  CloseTimerTimer Proc, Method fmSDAppInfo\TfmAppInfo (19)

  This is a list of all identifiers that are declared but never used. The Delphi compiler (from Delphi 2) also reports this if warnings ($W+) have been turned on during compilation. Most often, you can remove these identifiers. If you remove any identifier, make sure your code still compiles and works properly. A wise habit is to first comment out these declarations, and remove them entirely when you have validated that the code still compiles and works as intended. Also, note that if a subprogram is not used, does not necessarily indicate that it is not needed at all. If it is part of a general unit, the subprogram could very well be used in other applications.

  Identifiers (parameters, local variables etc) related to subprograms that are not used, are not reported.

  Constructors/destructors are not examined by this section. Also parameters to event handlers, or methods that are referenced in form files, are not reported as unused. The reason is to avoid unnecessary warnings.

Also unused methods of a class that are implemented through interfaces are not reported. In this case, the class has no choice but to implement these methods.

- **Local identifiers that possibly are set more than once without referencing in-between:**

  BlendBitmap : Simple (unknown) Var, Local dmScreenshot\TScreenshot\GetDimmed (45)

  This is a list of all local identifiers that are set (assigned) more than once without referencing in-between. They are referenced in unknown fashion, and the parser cannot determine whether they are set or just referenced in these locations. You can probably delete all but the last assignment.

- **Functions called only as procedures (result ignored):**

  DisplayAppInfo Func, Method fmSDAppInfo\TfmAppInfo (27)

  RemoveAppBar Func, Method fmSDAppInfo\TfmAppInfo (29)

  These functions may possibly better be implemented as procedures, because the result is never used.

- **Functions/procedures (methods excluded) only called once:**

  SDAppInfoApplicationExecutable Func, Interfaced SDCommon (52)

  SDAppInfoApplicationName Func, Interfaced SDCommon (51)

  The code in these functions/procedures could possibly be included inline instead, avoiding an unnecessary call.

- **Methods only called once from other method of the same class:**

  DisplayAppInfo Func, Method fmSDAppInfo\TfmAppInfo (27)

  GetBitmap Func, Method dmScreenshot\TScreenshot (15)

  These methods are never called from the outside. The code in these methods could possibly be included inline instead, avoiding an unnecessary call.

2.1.3 Convention Complience

- **Class fields that are not declared in the private section:**

  CloseTimer : Simple (unknown) ClassField fmSDAppInfo\TfmAppInfo (11)

  imgApplicationLogo : Simple (unknown) ClassField fmSDAppInfo\TfmAppInfo (16)

  This is a list of all class fields that are not declared in the private section of a class.

2.2 SDSecureDisplaySvc.drp

Loaded files include: SDCommon.pas, SDInfoProcesses.pas, SDInfoSecurity.pas, SDModifiedTokens.pas, SDProtocol.pas, SDSecureDisplaySvcImpl.pas, SDSecureDisplaySvcIntf.pas, SDSecureDisplaySvcUnit.pas and SDSecureDisplaySvcWeb.pas

2.2.1 Warnings:

- **Interfaced identifiers that are used, but not outside of unit:** see SDAppInfo: section 2.1.1.
- **Interfaced class identifiers that are public/published, but not used outside of unit:** see SDAppInfo: section 2.1.1.
- **Variables that are set, but never referenced**

  Name : UnicodeString RecField SDInfoProcesses\SDDumpAccessMask\TAccessRight (626)

  Value : Cardinal RecField SDInfoProcesses\SDDumpAccessMask\TAccessRight (625)

  This is a list of all variables that are set but never referenced. Either these variables are unnecessary or something is missing in the code, because it is meaningless to set a variable and then never reference, or use it.

  Restrictions: Variables marked with the absolute directive are not examined. These identifiers shadow another variable in memory, and are changed whenever the other variable changes.

  Recommendation: Examine why these variables are set, but never referenced.

- **Empty finally-block:**

  SDInfoProcesses (414)

```
411            try
412              // GetTokenUser(hToken)
413              AProcesses.Processes[nPIDIndex].SessionID := GetTokenSe
414            finally
415            end;
```

- **Functions called as procedures:** see SDAppInfo: section 2.1.1.
- **Identifier with same name as keyword/directive:**

  Name : UnicodeString RecField SDInfoProcesses\SDDumpAccessMask\TAccessRight (626)

  This section reports identifier with names that conflict with keywords/directives. Although allowed, at least it is a source for confusion when maintaining the code.

2.2.2 Inconsistent Case:

- **Inconsistent case for different identifiers with same name:**

  GroupName : PChar Var, Local SDInfoProcesses\GetTokenGroups (273)

  groupname : LPCWSTR ValParam SDModifiedTokens(78)

  This is a list of all identifiers with the same name, but that are declared with different case:

2.3 SDUserSessionManager.drp

loaded files include: SDCommon.pas, SDInfoProcesses.pas, SDModifiedTokens.pas and SDUserSessionManager.dpr

2.3.1 Warnings:

- **Interfaced identifiers that are used, but not outside of unit:** see SDAppInfo: section 2.1.1.

- **Variables that are set, but never referenced:** see SDSecureDisplaySvc: section 2.2.1.

- **Empty begin/end-blocks:**

  SDInfoProcesses (405)

```
403             begin
404               // Log('GetModuleBaseName() succeeded.');
405             end;
```

  Recommendation: The whole block should have been removed/commented away, instead of only what is inside the block.

- **Empty finally-block:** see SDSecureDisplaySvc: section 2.2.1.

- **Functions called as procedures:** see SDAppInfo: section 2.1.1.

- **Duplicate lines:**

  SDUserSessionManager(285) Log(")

  This section lists locations in the source code where a line is duplicated, that is when two lines immediately following each other, have the same content. Recommendation: The check is done without case-sensitivity, so check first if the duplicate is there for a reason.

- **Redeclared identifiers from System unit:** see SDAppInfo: section 2.1.1.

- **Identifier with same name as keyword/directive:** see SDSecureDisplaySvc: section 2.2.1.

2.4 Conclusion

A static code analysis tool like Pascal Analyzer does not see semantic errors, but it is good to find syntax errors in the code. Most of the errors found using Pascal Analyzer is relatively easy to fix, and most of them are not really necessary to address. The most prominent vulnerability in the code, is the lack of comments. This makes it hard to manually review the code, since it is hard to differentiate developers intentions with actual functionality of the code.

Referanser

[1] Pascal analyzer. [Online]. Available: http://www.peganza.com/products_pal.htm

[2] Pascal analyzer. Peganza. [Online]. Available: http://www.peganza.com/Files/PAL.PDF

[3] Pagnza palhelp. [Online]. Available: http://www.peganza.com/palhelp/index.html?warnings_report.htm