Joakim Harbitz
Studentid: 120924

Assignment #4

Finding vulnerabilities in the SecureDesktop application

Strategy for the review

To be able to find security vulnerabilities in the source code of a program, it is crucial that one understand the source code it self.

The source consist of both Pascal- and Delphi files. Since I am not already familiar with any of the languages, I have read about them to get a basic understanding of how they work and also read the article «Shark Cage - Protecting User Interfaces Against Malware.pdf» to get a deeper understanding of the application.

Next step is to decide how to find vulnerabilities and security threats in the code. Because of my basic understanding of the languages, I though it would be best to find an automated tool that analyzes the code and outputs a report for me to go through. Another way would be to manually go through all the source code to see if for instance buffer overflows or unsecured functions exists.

The tools I have found for doing my source code analysis is:

- Pascal Analyzer
- Pascal Browser

Pascal Analyzer works by building large internal tables of identifiers and then collects information such as calls between subprograms. When the parsing is completed, extensive reports are produced. [1]

Pascal Browser will parse the source code it find and gather all sorts of information. This information is used to create the document collection that describes the source code. [2]

Finding vulnerabilities

Pascal Analyzer throws out 15 warnings. In the evaluation copy that I am using, the whole report is not shown. It will therefore be a guess by me when analyzing its report. Some of the warnings are listed below.

«Variables with absolute directive (0, was unknown)»

My guess is that the source code is referencing a hard coded path to some directory. Hard coded paths may be a vulnerability if necessary checks of the path before loading is not implemented. This may be a «way in» to malicious programs by taking control over the hard coded path and get malicious code loaded in the Secure Desktop's RAM location. [3]

«Classes with more than one destructor (0, was unknown)»

In this case, if the destructors are not destructing objects and variable «equally», some objects or variables may never be removed from RAM. This may result in a «dangling pointer» that do not point to a valid object of the appropriate type. This may be exploited my malicious code since «dangling references» do no longer resolve to a valid destination. [4]

Solution to vulnerabilities

The second vulnerability is the one I consider most dangerous. The solution will be to ensure that all of the destructors delete the necessary objects and variables in a safe way so nothing is left behind.

The absolute directive vulnerability can be solved by not making it static, and check that the directory is the one that the program really is looking for.

Sources

[1]: Pascal Analyzer - http://peganza.com/products_pal.htm
[2]: Pascal Browser - http://peganza.com/products_pab.htm
[3]: RAM - http://www.webopedia.com/TERM/R/RAM.html
[4]: Dangling pointer - http://electrofriends.com/qna/software-languages/c-cpp-faq/dangling-pointer/

Joakim Harbitz
Studentid: 120924

Report from Pascal Analyzer

```
*****************************************************************************
*                        Warnings Report for                                *
*C:\USERS\PC\DESKTOP\OBLIG 3 - SOFTSEC\SECUREDESKTOP-SOURCE\DMSCREENSHOT.PAS*
*                        08.10.2014 15:36:05                                *
*****************************************************************************

In this evaluation version, identifiers starting with J,K,L are excluded.
Also some report lines are displayed as "xxxxxx".

Interfaced identifiers that are used, but not outside of unit (13, was unknown):
-------------------------------------------------------------------------------

ApplicationDirectoryName : UnicodeString        Typed const, Interfaced SDCommon (15)

BackgroundBitmapFileName : UnicodeString        Typed const, Interfaced SDCommon (23)


The entire list is not shown in this evaluation version


Interfaced class identifiers that are public/published, but not used outside of unit (2, was unknown):
-----------------------------------------------------------------------------------------------------

GetBitmap                       Func, Method          dmScreenshot\TScreenshot (15)

ScreenCapture : Simple (unknown) ClassField             dmScreenshot\TScreenshot (10)


Variables that are referenced, but never set (0, was unknown):
--------------------------------------------------------------------------

        (none)


Variables that are set, but never referenced (0, was unknown):
--------------------------------------------------------------------------

        (none)


Local variables that are referenced before they are set (0, was unknown):
--------------------------------------------------------------------------

        (none)


Var parameters that are used, but never set (0, was unknown):
--------------------------------------------------------------------------

        (none)


Value parameters that are set (0, was unknown):
--------------------------------------------------------------------------

        (none)


Interfaces passed as parameters without "const" directive (0, was unknown):
--------------------------------------------------------------------------
```

(none)

Variables with absolute directive (0, was unknown):

(none)

Constructors/destructors without calls to inherited (0, was unknown):

(none)

Destructors without override directive (0, was unknown):

(none)

Classes with more than one destructor (0, was unknown):

(none)

Function result not set (0, was unknown):

(none)

Recursive subprograms (0, was unknown):

(none)

Dangerous Exit-statements (0, was unknown):

(none)

Dangerous Raise (0, was unknown):

(none)

Dangerous Label-locations inside for-loops (0, was unknown):

(none)

Dangerous Label-locations inside repeat/while-loops (0, was unknown):

(none)

Possible bad object creation (0, was unknown):

(none)

Bad thread-local variables (0, was unknown):

(none)

Instance created of class with abstract methods (0, was unknown):

(none)

Empty begin/end-blocks (0, was unknown):

(none)

Empty case labels (0, was unknown):

(none)

Short-circuited for-statements (0, was unknown):

(none)

Short-circuited if-statements (0, was unknown):

(none)

Short-circuited on-statements (0, was unknown):

(none)

Short-circuited repeat-statements (0, was unknown):

(none)

Short-circuited while-statements (0, was unknown):

(none)

Empty except-block (0, was unknown):

(none)

Joakim Harbitz
Studentid: 120924

Empty finally-block (0, was unknown):

(none)

Forward directive in interface (0, was unknown):

(none)

Empty subprogram parameter list (0, was unknown):

(none)

Ambiguous references in with-blocks (0, was unknown):

(none)

Classes without overrides of abstract methods (0, was unknown):

(none)

Local for-loop variables read after loop (0, was unknown):

(none)

For-loop variables not used in loop (0, was unknown):

(none)

Non-public constructors/destructors (0, was unknown):

(none)

Functions called as procedures (0, was 0):

(none)

Mismatch property read/write specifiers (0, was unknown):

(none)

Local variables that are set but not later used (0, was unknown):

(none)

Joakim Harbitz
Studentid: 120924

Duplicate lines (0, was unknown):

(none)

Duplicate class types in except-block (0, was unknown):

(none)

Redeclared identifiers from System unit (0, was unknown):

(none)

Identifier with same name as keyword/directive (0, was unknown):

(none)