Assigment 4

Joachim Hansen 120483

October 8, 2014

Readers

This report is written on the assumption that the reader have a good understanding of the course material.

Documentation

I have made illustration on how I have used the different static analysers and in addition to these figures, I have also explained the process in natrual language

Short on benefits and weaknesses by static analysis

+ Can run on incomplete code
+ can run in a big prosject
+ less time consuming than manual auditing (which requires an expert)
- Need users to apply rules. A security flaw not covered in a rule will not be discovered by the analyser.
- Users need to apply priority to rules
- Will get false negatives (false sense of security)

[1, pages 106,108-109]

Shark Cage article

I will refer to this article in this report, but it will not show up in the sources list in the end of the documet. This is becouse the article is missing information on the authors, year and so on.

Review strategy

My strategy for this review is to first understand the specification for the SharkCage application, gain a basic understanding for the Pascal programming language and prioritize the time I will spend on auditing each source file by its overall importance.

I also need to make decisions on how much time I will dedicate to different parts of this assignments, this is because I work alone and have no prior experience with auditing source code. This might reflect the end result.

prioritise tasks

I will classify the vulnerabilities found by the categories in table 1. The column costly to change is a catagory that both spesification and design falls under. This is becouse flaws here will ultimatly cause the need for alot of rework.[2, page 110]. Next catagory is the hign importance where the Cage Amin module belongs. I think that prioritizing this module over the others make sense, becouse this module is tasked with recording the shared secret, decide who will be granted accress to the trusted path and configuring (in other words: the more important stuff). This scheme is made so that I know how I should divide my time in the audeting process. Which scheme I will use (if any) to communicate the different severity levels of the SharkCage application security flaws, will be based upon feedback from the analyser.

Table 1: Vulnerabilities catagories

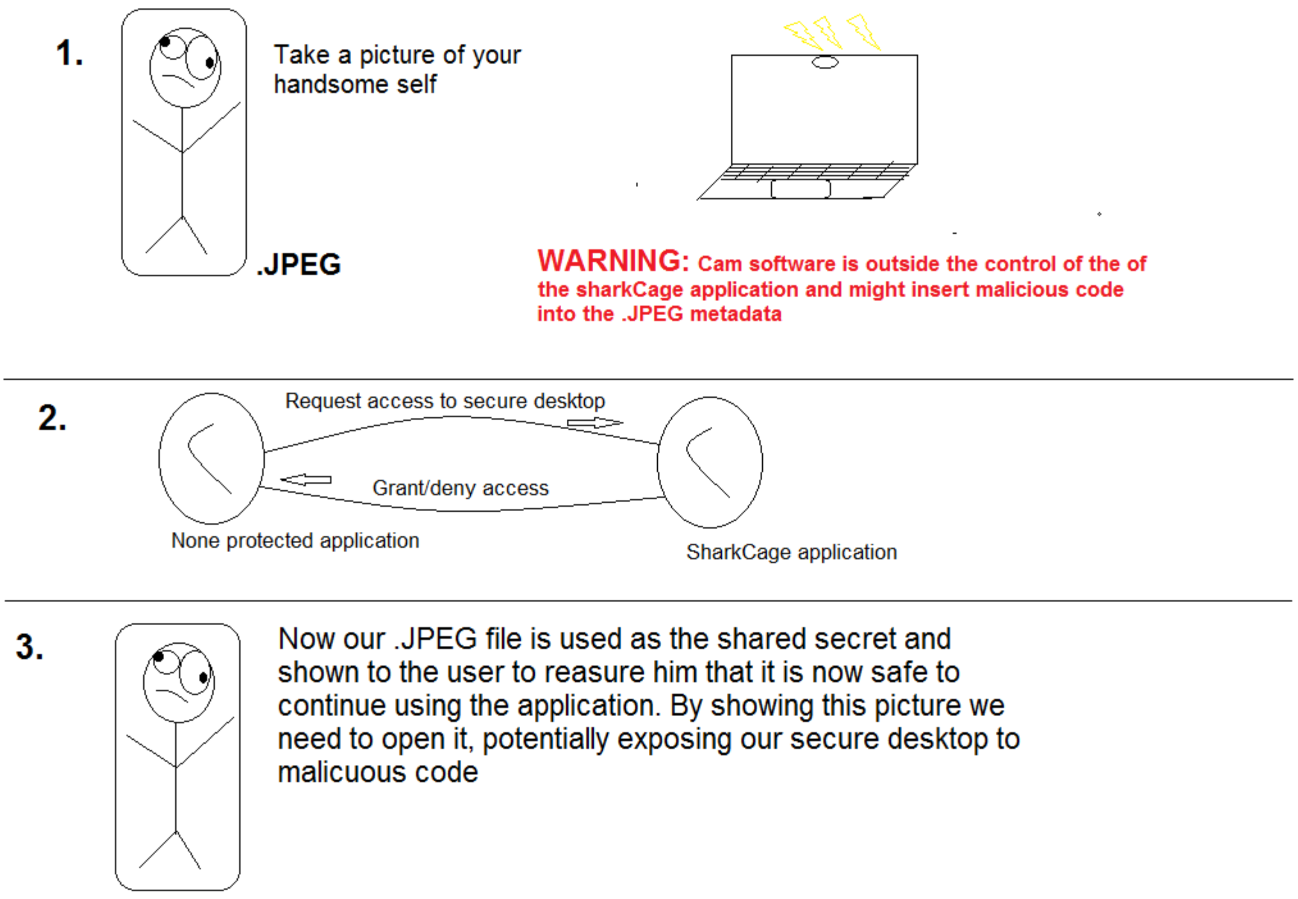
Costly to change	high importance	lower importance
Spesification and design(article)	Cage Admin	other modules

Possible design flaw

The SharkCage article talks about a possible design implementation where the integrated camera on your laptop will provide the means for the protected application and SharkCage to share a secret. While this sounds great on paper, the integrated camera on your laptop is also controlled by software/code that is outside the SharkCage control.

Let's imagine that the authors have the intension to have the cam driver process and the protected application kept in separate desktop (isolated from each other). After the picture of your handsome self-have be captured by the cam, the rights to this file have been limited to sharkCage and the protected application. So far so good right? Well what if our trust in the cam software was misplaced and this software inserted malicious code into the picture file (e.g. JPEG). This scenario is within the realm of possibility, as inserting malicious code in metadata has been done before[3]. In figure 1 I try to illustrate how this scenario might play out.

Figure 1: Design flaw: step by step


1.
Take a picture of your handsome self
.JPEG
2.
Request access to secure desktop
Grant/deny access
None protected application
SharkCage application
3.

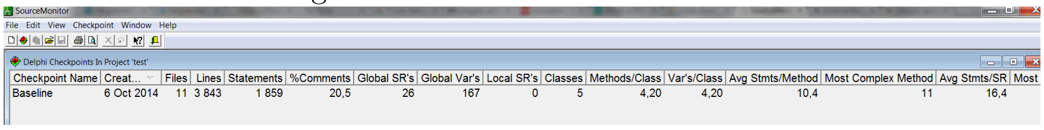
WARNING: Cam software is outside the control of the of the sharkCage application and might insert malicious code into the .JPEG metadata

Now our .JPEG file is used as the shared secret and shown to the user to reasure him that it is now safe to continue using the application. By showing this picture we need to open it, potentially exposing our secure desktop to malicuous code

Use of the source Monitor analyser

This program scans though the selected Pascal files (*ps) and outputs a baseline containing multiple different metrics on the code e.g. how many lines of comment, how many statements etc. Later baselines can be compared with previous ones and provide the programmers with information relevant to the quality of the software being developed (see figure 2). This information alone do not provide me with a lot of information regarding security flaws, but I will use this in combination with other analysers.[4]

Figure 2: Source monitor: Baseline

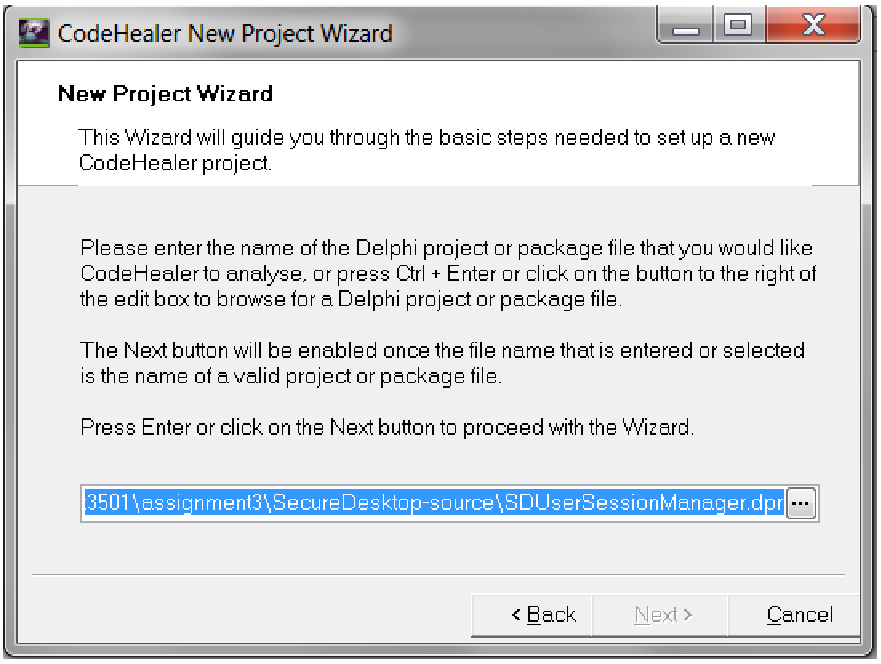

SourceMonitor
File Edit View Checkpoint Window Help
Delphi Checkpoints In Project 'test'

Checkpoint Name	Creat...	Files	Lines	Statements	%Comments	Global SR's	Global Var's	Local SR's	Classes	Methods/Class	Var's/Class	Avg Stmts/Method	Most Complex Method	Avg Stmts/SR	Most
Baseline	6 Oct 2014	11	3 843	1 859	20,5	26	167	0	5	4,20	4,20	10,4	11	16,4	

Code healer

The application that keeps on giving (do not work). See the wonderfull application in figure 3. As you can see the wizard do not work :) [5]

Figure 3: Code Haler (The healer that got ebola)



New Project Wizard

This Wizard will guide you through the basic steps needed to set up a new CodeHealer project.

Please enter the name of the Delphi project or package file that you would like CodeHealer to analyse, or press Ctrl + Enter or click on the button to the right of the edit box to browse for a Delphi project or package file.

The Next button will be enabled once the file name that is entered or selected is the name of a valid project or package file.

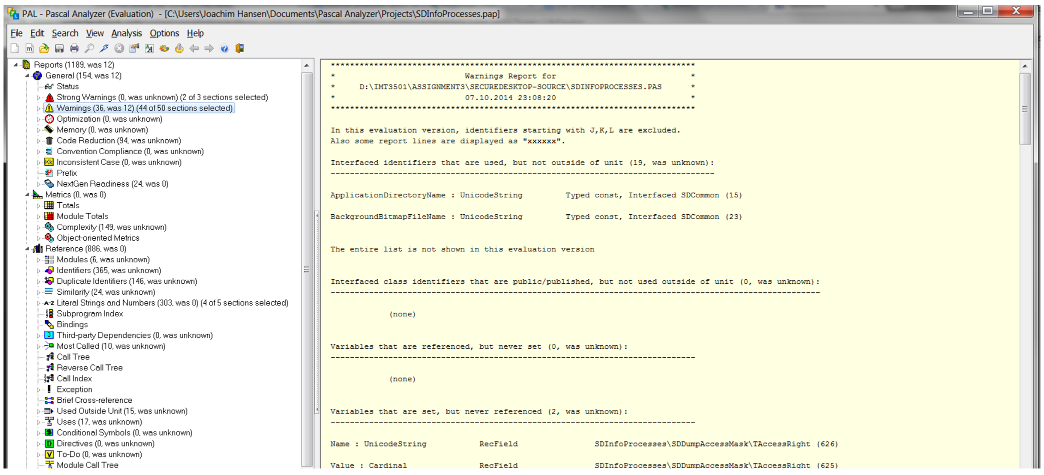
Press Enter or click on the Next button to proceed with the Wizard.

3501\assignment3\SecureDesktop-source\SDUserSessionManager.dpr

PAL Pascal Analyser

This software seems to be everything that I could wish for. It has extensive tecnical documentation of what its different levels of warings means, call tree, classes, third party dependencies and so on[6]. The only problem here is that it's not free. The evaluation program only provides a small subset of the information the full version would show and this is far from enough to get any use of the program. My only remaining choice seems to be to audit the code manually or using grep (which is free). Becouse I refuse to use money on buying software to pass this assigment (see figure 4). [7]

Figure 4: PAY 39.99


PAL - Pascal Analyzer (Evaluation) - [C:\Users\Joachim Hansen\Documents\Pascal Analyzer\Projects\SDInfoProcesses.pap]
File Edit Search View Analysis Options Help

```
Reports (1189, was 12)
  General (154, was 12)
    Status
    Strong Warnings (0, was unknown) (2 of 3 sections selected)
    Warnings (36, was 12) (44 of 50 sections selected)
    Optimization (0, was unknown)
    Memory (0, was unknown)
    Code Reduction (94, was unknown)
    Convention Compliance (0, was unknown)
    Inconsistent Case (0, was unknown)
    Prefix
    NextGen Readiness (24, was 0)
  Metrics (0, was 0)
    Totals
    Module Totals
    Complexity (149, was unknown)
    Object-oriented Metrics
  Reference (886, was 0)
    Modules (6, was unknown)
    Identifiers (365, was unknown)
    Duplicate Identifiers (146, was unknown)
    Similarity (24, was unknown)
    Literal Strings and Numbers (303, was 0) (4 of 5 sections selected)
    Subprogram Index
    Bindings
    Third-party Dependencies (0, was unknown)
    Most Called (10, was unknown)
    Call Tree
    Reverse Call Tree
    Call Index
    Exception
    Brief Cross-reference
    Used Outside Unit (15, was unknown)
    Uses (17, was unknown)
    Conditional Symbols (0, was unknown)
    Directives (0, was unknown)
    To-Do (0, was unknown)
    Module Call Tree
```

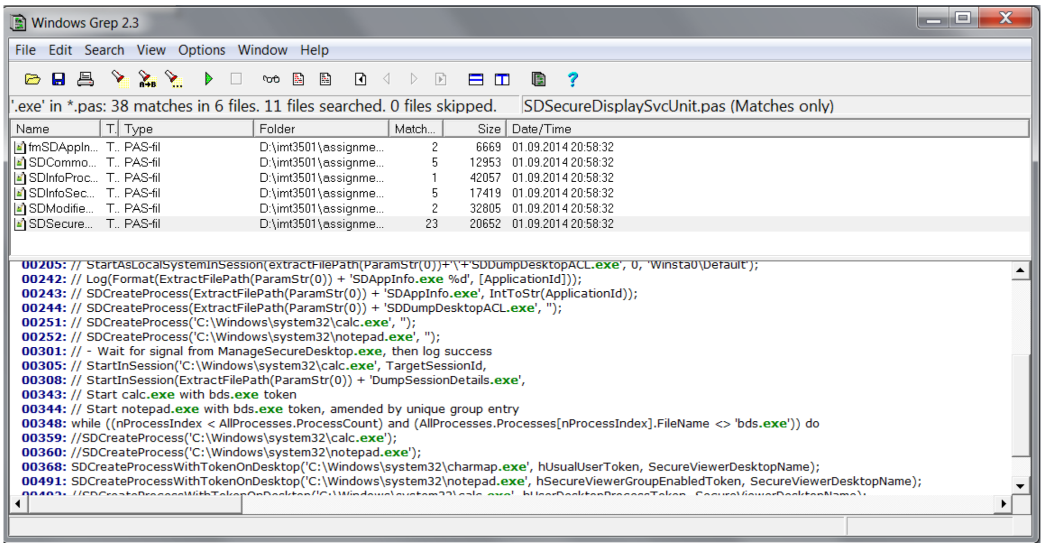
```
****************************************************************************
*                           Warnings Report for                            *
*     D:\IMT3501\ASSIGNMENT3\SECUREDESKTOP-SOURCE\SDINFOPROCESSES.PAS      *
*                           07.10.2014 23:08:20                            *
****************************************************************************

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 (19, 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 (0, was unknown):
------------------------------------------------------------------------------------------------------

            (none)


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

            (none)


Variables that are set, but never referenced (2, was unknown):
----------------------------------------------------------------------------

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

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

Seach for spesific strings

I will use an application called Windows grep which gives me the utilities like that off the UNIX command with the same name (without 'windows' off course). "problably the simplest and most straightforward approach to static analysis is the UNIX utility grep" - [quote] Gery McGraw in the book Software secuity [1, page 110]. With this application you are also able to use regex statements (like in grep - e in UNIX) With this application you are also able to use regex statements (like in grep - e in UNIX). In windows grep you are presented with a wizard that helps you choose which sets of folders and files extensions you want to search in. The program will then show which folders and files matched your query, you can then click each individual files and see what in the file matched and get the line number. I my case I used the assignment folder and .PS extension. I seach first for .exe files to see if the program is depended on other prosesses. Several programs is used by the SharkCage application e.g. calc.exe and notepad.exe. The shark cage apllication is not only trusting these executable files, but any third party API they might use (notion of transetive trust) [8, page 19]. See figure 5.My seccond query was to seach for .dll, I did this becouse dll injection is a common infection vector [9] see figure 6. I could also have used a regex expression to check for hard coded values as well as if functions actually gets called in the code.By just taking a quick look at the code, you will see that there is a lot of hard coded values. This can result in unexpeted results if not MR pro programmer is very displinied. There is also alot of hard coded paths which could be exploited (see figure 7)

Gary Mcgraw also describes the downside with the use of grep: ”Grep is rather lo-fi becosue it dosen’t understand anything about the files it scans. Comments, string literals, declerations, and function calls are all just part of the stream of characters to be matched against” [1, page 110]

Figure 5: .exe

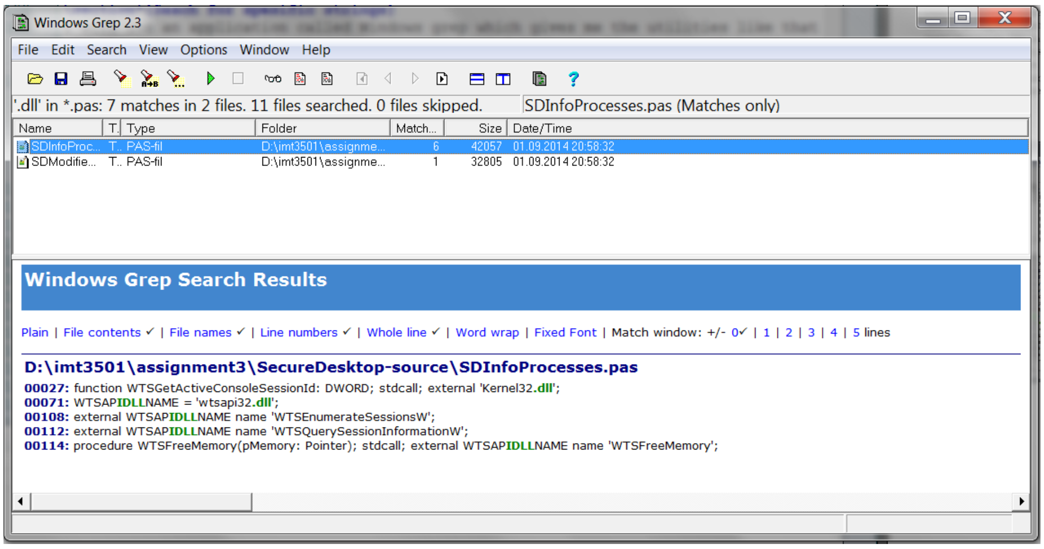


'.exe' in *.pas: 38 matches in 6 files. 11 files searched. 0 files skipped. SDSecureDisplaySvcUnit.pas (Matches only)

Name	T.	Type	Folder	Match...	Size	Date/Time
fmSDAppIn...	T..	PAS-fil	D:\imt3501\assignme...	2	6669	01.09.2014 20:58:32
SDCommo...	T..	PAS-fil	D:\imt3501\assignme...	5	12953	01.09.2014 20:58:32
SDInfoProc...	T..	PAS-fil	D:\imt3501\assignme...	1	42057	01.09.2014 20:58:32
SDInfoSec...	T..	PAS-fil	D:\imt3501\assignme...	5	17419	01.09.2014 20:58:32
SDModifie...	T..	PAS-fil	D:\imt3501\assignme...	2	32805	01.09.2014 20:58:32
SDSecure...	T..	PAS-fil	D:\imt3501\assignme...	23	20652	01.09.2014 20:58:32

```
00205: // StartAsLocalSystemInSession(extractFilePath(ParamStr(0))+'\'+'SDDumpDesktopACL.exe', 0, 'Winsta0\Default');
00242: // Log(Format(ExtractFilePath(ParamStr(0)) + 'SDAppInfo.exe %d', [ApplicationId]));
00243: // SDCreateProcess(ExtractFilePath(ParamStr(0)) + 'SDAppInfo.exe', IntToStr(ApplicationId));
00244: // SDCreateProcess(ExtractFilePath(ParamStr(0)) + 'SDDumpDesktopACL.exe', '');
00251: // SDCreateProcess('C:\Windows\system32\calc.exe', '');
00252: // SDCreateProcess('C:\Windows\system32\notepad.exe', '');
00301: // - Wait for signal from ManageSecureDesktop.exe, then log success
00305: // StartInSession('C:\Windows\system32\calc.exe', TargetSessionId,
00308: // StartInSession(ExtractFilePath(ParamStr(0)) + 'DumpSessionDetails.exe',
00343: // Start calc.exe with bds.exe token
00344: // Start notepad.exe with bds.exe token, amended by unique group entry
00348: while ((nProcessIndex < AllProcesses.ProcessCount) and (AllProcesses.Processes[nProcessIndex].FileName <> 'bds.exe')) do
00359: //SDCreateProcess('C:\Windows\system32\calc.exe');
00360: //SDCreateProcess('C:\Windows\system32\notepad.exe');
00368: SDCreateProcessWithTokenOnDesktop('C:\Windows\system32\charmap.exe', hUsualUserToken, SecureViewerDesktopName);
00491: SDCreateProcessWithTokenOnDesktop('C:\Windows\system32\notepad.exe', hSecureViewerGroupEnabledToken, SecureViewerDesktopName);
```

Figure 6: .dll



```
'.dll' in *.pas: 7 matches in 2 files. 11 files searched. 0 files skipped.    SDInfoProcesses.pas (Matches only)
```

Name	T.	Type	Folder	Match...	Size	Date/Time
SDInfoProc...	T..	PAS-fil	D:\imt3501\assignme...	6	42057	01.09.2014 20:58:32
SDModifie...	T..	PAS-fil	D:\imt3501\assignme...	1	32805	01.09.2014 20:58:32

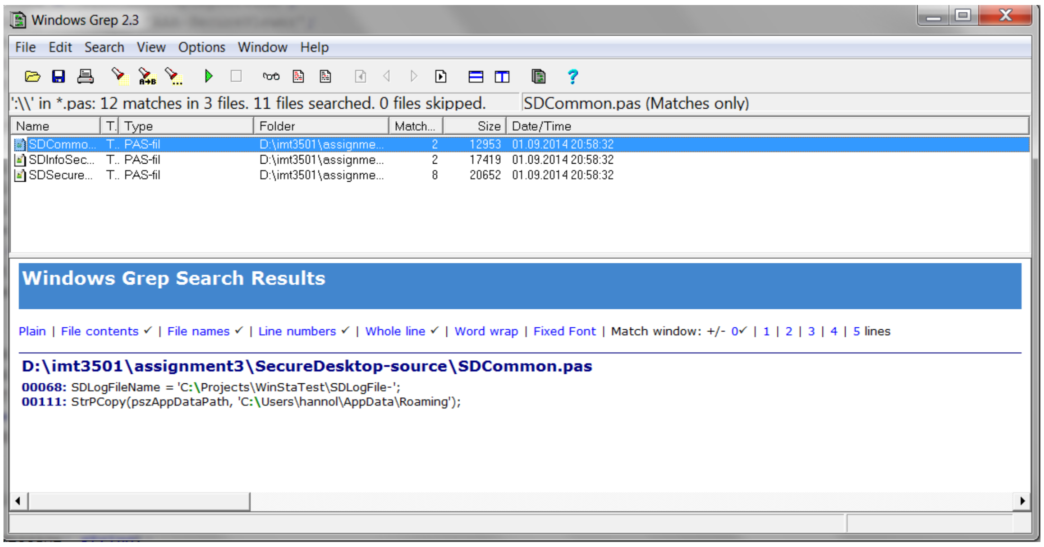
Windows Grep Search Results

Plain | File contents ✓ | File names ✓ | Line numbers ✓ | Whole line ✓ | Word wrap | Fixed Font | Match window: +/- 0✓ | 1 | 2 | 3 | 4 | 5 lines

D:\imt3501\assignment3\SecureDesktop-source\SDInfoProcesses.pas

```
00027: function WTSGetActiveConsoleSessionId: DWORD; stdcall; external 'Kernel32.dll';
00071: WTSAPIDLLNAME = 'wtsapi32.dll';
00108: external WTSAPIDLLNAME name 'WTSEnumerateSessionsW';
00112: external WTSAPIDLLNAME name 'WTSQuerySessionInformationW';
00114: procedure WTSFreeMemory(pMemory: Pointer); stdcall; external WTSAPIDLLNAME name 'WTSFreeMemory';
```

Figure 7: Hard coded paths



```
':\' in *.pas: 12 matches in 3 files. 11 files searched. 0 files skipped.    SDCommon.pas (Matches only)
```

Name	T.	Type	Folder	Match...	Size	Date/Time
SDCommo...	T..	PAS-fil	D:\imt3501\assignme...	2	12953	01.09.2014 20:58:32
SDInfoSec...	T..	PAS-fil	D:\imt3501\assignme...	2	17419	01.09.2014 20:58:32
SDSecure...	T..	PAS-fil	D:\imt3501\assignme...	8	20652	01.09.2014 20:58:32

Windows Grep Search Results

Plain | File contents ✓ | File names ✓ | Line numbers ✓ | Whole line ✓ | Word wrap | Fixed Font | Match window: +/- 0✓ | 1 | 2 | 3 | 4 | 5 lines

D:\imt3501\assignment3\SecureDesktop-source\SDCommon.pas

```
00068: SDLogFileName = 'C:\Projects\WinStaTest\SDLogFile-';
00111: StrPCopy(pszAppDataPath, 'C:\Users\hannol\AppData\Roaming');
```

What to modify? (conclusion)

I would replace absolute paths with relative paths, and hard coded values with CONSTANT. I recommend dropping the possibility of using the integrating camera to generate the shared secret. And perhaps the use a CAPTICA instead, that the SharkCage application controls(the shark cage application also mention this technology) . The program should not place blind trust on .dll and .exe file extensions.

Bibliography

[1] G. MCGRAW, *Software security bilding security in*, 2012.

[2] I. Sommerville, *Software engineering*, 2011.

[3] V. Diaz. (2013). Malware in metadata, [Online]. Available: `http://securelist.com/blog/research/58196/malware-in-metadata/`.

[4] C. Software. (2014). Source monitor, [Online]. Available: `http://www.campwoodsw.com/sourcemonitor.html`.

[5] C. healer group. (2014). Code healer for delphi, [Online]. Available: `http://www.socksoftware.com/`.

[6] Peganza. (2014). Pascal analyser, [Online]. Available: `http://www.peganza.com/Files/PAL.PDF`.

[7] ——, (2014). Pascal analyser info, [Online]. Available: `http://www.peganza.com/products_pal.htm`.

[8] M. Dowd, J. McDonald, and J. Schuh, *The art of security assesment*, 2007.

[9] A. Malik. (2014). Dll injection and hooking, [Online]. Available: `http://securityxploded.com/dll-injection-and-hooking.php`.