Høgskolen i Gjøvik

Software Security

Obligatory exercise #4

SecureDesktop code review

Author:
120915 - Halvor Mydske Thoresen

October 8, 2014

1 Intro

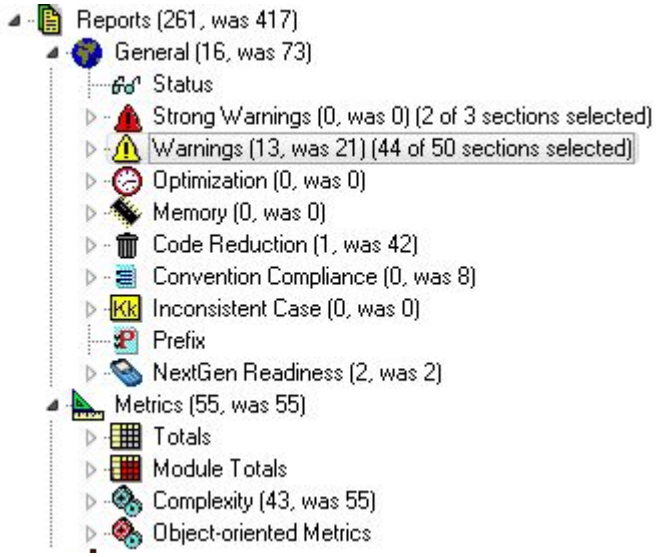
The task of this assignment was to find vulnerabilities in the source code for the SecureDesktop application and to write a code review.
Disclaimer: This paper is written without extensive knowlegde about Delphi/Pascal or any of the static analysis tools used.

2 Strategy

- Run tools for static code analysis
- Try to learn pascal
- Manual code review

3 Tools

The tools used for static code analyzis was CodeHealer[1], ICARUS[2] and Pascal analyser[3].
CodeHealer and ICARUS did not give anything to work with (Mostly because of parsing and general program errors).
Pascal analyser actually found some problems. However, because the evaluation verson of the program was used, not everything was shown.

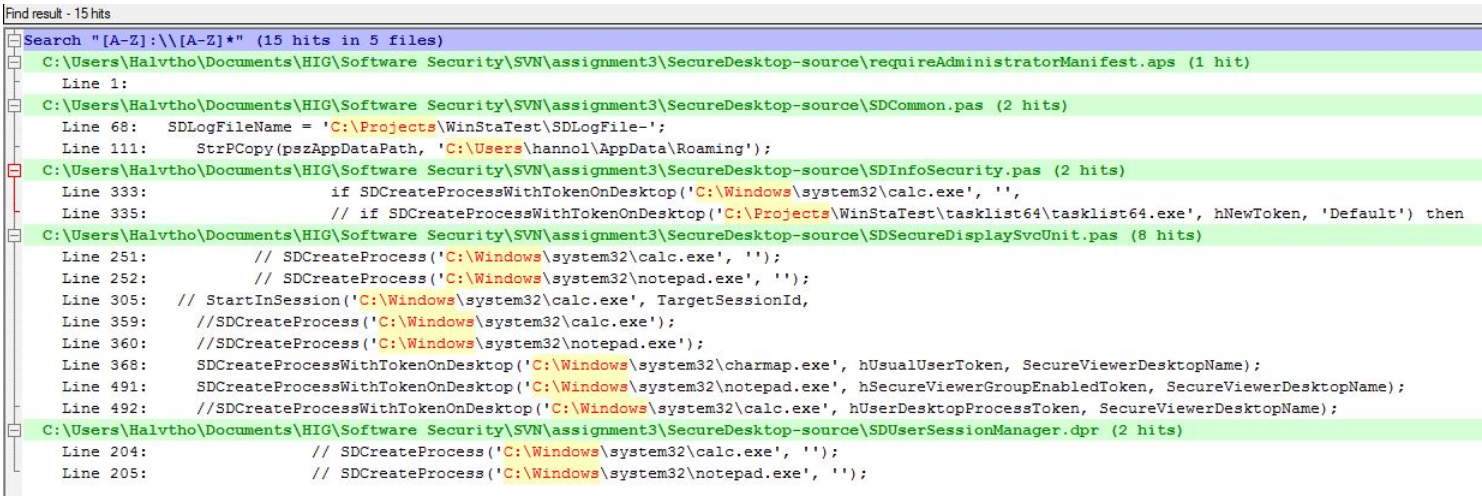


PAL did not find any major vulnerabilities, but it listed some minor problems like variables being declared and never used, and redundant code snippets.

4 Manual code review

4.1 Hardcoded path

I decided to search through all the project files using the notepad++ regex function and see if I could find any hardcoded paths.



Hardcoding paths is generally a bad idea and might open up for an attacker to do something with the path that is being pointed to.

4.2 URL

A URL is requested from the .ini file. This might be a vulnerability if the attacker can change which url is being executed.

```
118  function SDServiceRequestURL: string;
119  var
120    INI: TINIFile;
121  begin
122    INI := TINIFile.Create(Format('%s\%s', [ConfigDirectory, ConfigFileName]));
123    Result := INI.ReadString(INISection_ServiceRequest, INIIdent_RequestURL, '');
124    INI.Free;
125  end;
```

4.3 Possible buffer overflow

Because of my suboptimal pascal skills I can not say for sure, but there might an opening for a bufferoverflow attack ot lines 257 to 270 in the SDCommon.pas file.
I do not know how arrays, seLenght or the ReadFile function works, but I am adding this just incase there migth be a problem. What if AMailSlot is

larger than the allocated buffer?

```
257   if GetMailslotInfo(AMailslot, nil, cNextMessageSize, @cMessagesCount, nil) then
258   begin
259     if (cNextMessageSize <> MAILSLOT_NO_MESSAGE) then
260     begin
261       SetLength(Buffer, cNextMessageSize div SizeOf(Char));
262       if ReadFile(AMailslot, Buffer[0], cNextMessageSize, cBytesRead, nil) then
263       begin
264         Log(Format('ReadFile(%d) succeeded, %d bytes read.', [cNextMessageSize, cBytesRead]));
265
266         // for nIndex := Low(Buffer) to High(Buffer) do
267         // begin
268         // Log(Format('%.3d: "%s" (%.2x)', [nIndex, Buffer[nIndex], Ord(Buffer[nIndex])]));
269         // end;
270         AMessage := string(Buffer);
```

4.4 Image load

The program loades a image without checking for size. What would happen if the file was replaced with a much larger file?

```
93  function TfmAppInfo.DisplayAppInfo(const AKey: Integer): Boolean;
94  begin
95    imgApplicationLogo.Picture.LoadFromFile(SDAppInfoFullLogoFileName(AKey));
96    //Log(Format('+imgApplicationLogo.Picture.LoadFromFile(%s)',
97    //  [SDAppInfoFullLogoFileName(AKey)]));
98
```

```
320   function SDAppInfoLogoFileName(const nApplicationKey: Integer): string;
321 begin
322     if (nApplicationKey >= 0) then
323   begin
324       Result := Format('Logo%.4d.png', [nApplicationKey]);
325     end
326     else
327   begin
328       Result := 'invalid.png';
329     end;
330 end;
331
332   function SDAppInfoFullLogoFileName(const nApplicationKey: Integer): string;
333 begin
334     Result := Format('%s\%s', [ConfigDirectory, SDAppInfoLogoFileName(nApplicationKey)]
335   //Log(Format('nApplicationKey: %d', [nApplicationKey]));
336     //Log(Format('LogoFileName(%d): %s', [nApplicationKey, SDAppInfoLogoFileName(nAppli
337     //Log(Format('ConfigDirectory: %s', [ConfigDirectory]));
338     //Log(Format('Result: %s', [Result]));
339 end;
340
```

4.5 Comments and documentation

There is a distinct lack of comments in the code, which will make it hard for maintenance and other code reviews later on. This might allow for potential vulnerabilities to go unnoticed for å long time.

5 Vulnerability priority

First of all, the hardcoded paths should be fixed.
Next, the code should be commented and documented.
When both the points above is completed, then the rest should be looked at.

References

[1] CodeHealerGroup. Codehealer download. http://www.socksoftware.com/downloads.php. [Online; accessed 7-October-2014].

[2] Peganza. Icarus download. http://www.peganza.com/downloads.htm. [Online; accessed 7-October-2014].

[3] Peganza. Pascal analyzer download. http://www.peganza.com/downloads.htm. [Online; accessed 7-October-2014].

[4] Campwood Software. Sourcemonitor version 3.5. http://www.campwoodsw.com/sourcemonitor.html. [Online; accessed 7-October-2014].