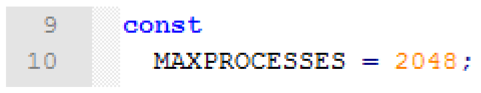
ASSIGNMENT 3

Katrine Borge, Kim Roger Torgersen
12HBISA, 12HBIDATA

Strategy:
In order to find possible vulnerabilities we plan to use known flaws in the pascal language along with what we have learned in this course and other courses. Our knowledge of pascal as a programming language is little to none so we will mostly use what this course has taught us. We will also do a search for any applications available to assist us in this task.

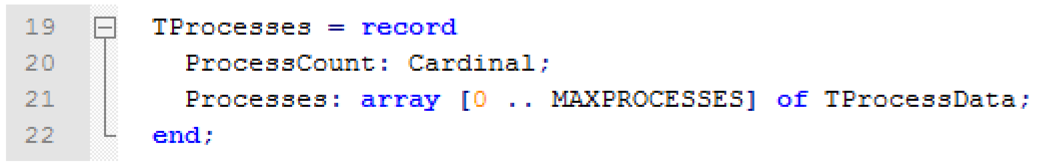
Software:
The software we found was Lazarus, but when we used this we found that the source code was not complete, thus making it harder to find software that can help us in the search for flaws. We chose to not use any more software after this since most software require the complete sourcecode in order to find flaws.

Vulnerability:
We found our first vulnerability in the file: SDInfoProcesses.pas on line 10, 21, 374-428(Function SDEnumerateProcesses(...))



```
 9  const
10    MAXPROCESSES = 2048;
```

Line 10 in this file creates a constant value of 2048.



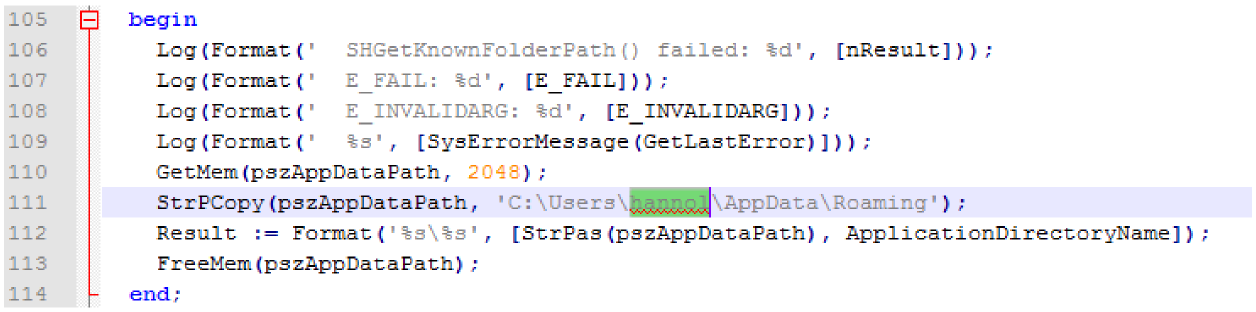
```
19    TProcesses = record
20      ProcessCount: Cardinal;
21      Processes: array [0 .. MAXPROCESSES] of TProcessData;
22    end;
```

In line 21 we see this value being used to create an array of Processes.
This array is used in the function SDEnumerateProcesses. The security flaw in this is the possibility of a buffer overflow as this function do not check if the array actually goes over 2048 items. Where as the user then can, though highly unlikely, create more than 2048 processes in the securedesktop application and thus work outside of the arrays set memory area.

Possible solution:

Although this a highly unlikely scenario, it might happen in a situation where someone wants to exploit this flaw.
A possible solution to it would be to add a check on how many item are in the array and stopping the user from creating more processes when it is full.
Another solution would be to create an array of arrays where as next array will be used when the previous one is full, this might not be the optimal solution as this will end up with alot of extra code and more overhead on the computer when a lot of processes is created.

Vulnerability 2:
File: SDCommon.pas on line 111.
In line 111 they use a hardcoded path to where they copy the data. It's hardcoded into the path of the user that made the program. This will make the program crash on any other computer because it wouldn't find the right path.

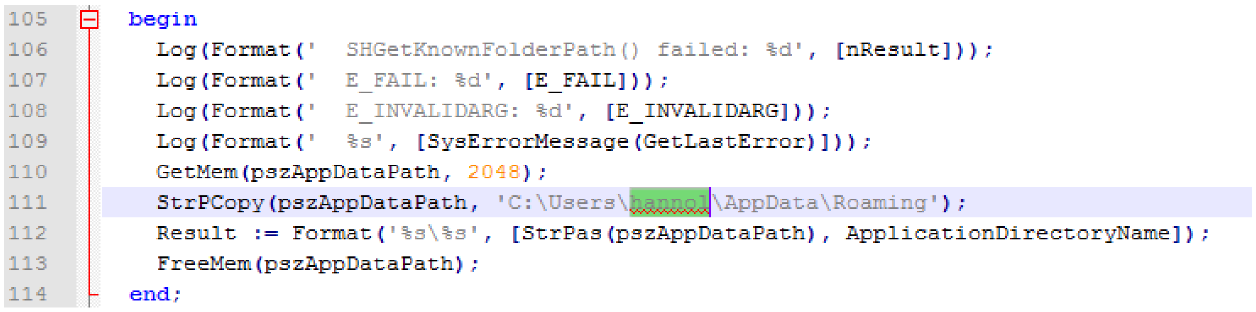


```
105  begin
106    Log(Format('  SHGetKnownFolderPath() failed: %d', [nResult]));
107    Log(Format('  E_FAIL: %d', [E_FAIL]));
108    Log(Format('  E_INVALIDARG: %d', [E_INVALIDARG]));
109    Log(Format('  %s', [SysErrorMessage(GetLastError)]));
110    GetMem(pszAppDataPath, 2048);
111    StrPCopy(pszAppDataPath, 'C:\Users\hannol\AppData\Roaming');
112    Result := Format('%s\%s', [StrPas(pszAppDataPath), ApplicationDirectoryName]);
113    FreeMem(pszAppDataPath);
114  end;
```

Solution:
The first thing is to not hardcode any path and instead use %appdata%, or other types of alternatives that works for the most common operation systems

Vulnerability 3:
File: SDLofFile:

Several places in the code they use the function “log”. This logs the data into an open log-file.



```
begin
  Log(Format('  SHGetKnownFolderPath() failed: %d', [nResult]));
  Log(Format('  E_FAIL: %d', [E_FAIL]));
  Log(Format('  E_INVALIDARG: %d', [E_INVALIDARG]));
  Log(Format('  %s', [SysErrorMessage(GetLastError)]));
  GetMem(pszAppDataPath, 2048);
  StrPCopy(pszAppDataPath, 'C:\Users\hanno1\AppData\Roaming');
  Result := Format('%s\%s', [StrPas(pszAppDataPath), ApplicationDirectoryName]);
  FreeMem(pszAppDataPath);
end;
```

Solution:
A way to solve the logging problem is to make some sort of encryption on the log, instead of having it in plaintext when opened in notepad. To see what’s really in the log-file you have to have the decryption key which should only be given to the people that really needs it.

References:

- Hoglund, G. and McGraw, G. (2004). Exploiting Software: How to Break Code. ISBN 0-201-78695-8. Library 005.8 Hog

- McGraw, G. (2006). Software Security: Building Security in. ISBN 0-321-35670-5. Library 005.8 McG

- Dowd, M., McDonald, J., and Schuh, J. (2006). The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. ISBN 0-321-44442-6. Library 005.8 Dow