Software Security Assignment 3 - Code Review

Mats Authen

October 8, 2014

Introduction

About the assignment

This is the task we where given for the 3rd obligatory assignment:

- Find vulnerabilities in the SecureDesktop application. You find source code and an article in the repository.
- The source code has not been published, so you must not share it with people outside this course.
- What is your strategy for the review?
- How do you prioritise vulnerabilities found?
- How would you modify the application to remove the vulnerabilities?
- Document your process so that your results can be reproduced.
- You may work in groups up to 3 people, but that should be reflected in the number of tools you use and in how many potential vulnerabilities you address.

About the authors

For this assignment, I cooperated with Tommy Andre Evensen (*120484*). We worked together on choosing a methodology, and discovering vulnerabilities. We wrote the rest individually, including prioritising of vulnerabilities, and how we chose to document our findings.

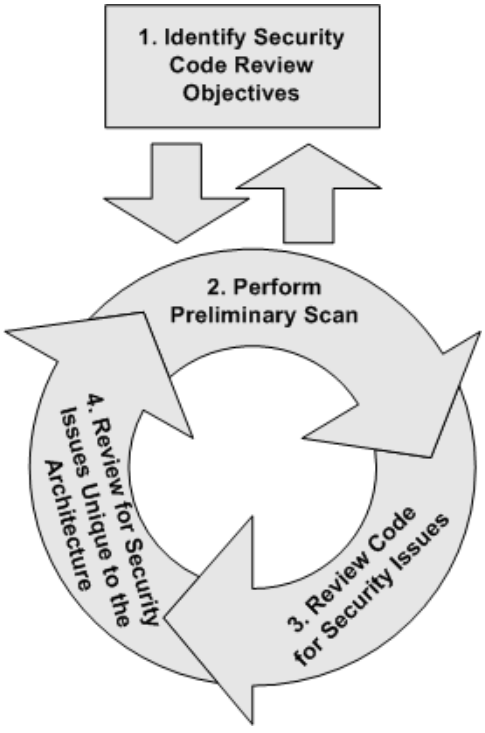
Solution

Method

We decided on Microsoft's guide to Perform a Security Code Review for Managed Code. The steps of the methodology is:

1. Identify security code review objectives
2. Perform preliminary scan
3. Review code for security issues
4. Review for security issues unique to the architecture

Because all attempts to find a tool for step 2 has failed (both us and apparently everyone else), we decided to skip this step. Instead we went straight for step 3, which is manual review.
Here is a graphical representation of the methodology:



Vulnerability prioritisation

I decided to grade the vulnerabilities on a scale from 1 to 10, with 1 being a bad coding practice, and 10 being a severe vulnerability that is both easy to exploit and has big consequences, like easy and effortless privilege escalation.

Findings

Potential heap overflow

Where?	SDCommon.pas, line 247-292 (SDReadFromMailslot function)
Description	The length of Buffer is protected by the SetLength function, but as it is a char array, it's basically a null terminated string. Because of this, if an attacker can inject a string with a null terminator %00 in the middle of it in the file that is read from on line 262, that will count as the end of the string, and everything else on the buffer will be placed on the heap, but not subject to the length check.
Outcome	If this can be used to do a heap overflow attack, an attacker can use this to inject shellcode, or another form of malicious payload into the heap. The result can be total control to the attacker. However, I should point out that I am uncertain if it can actually be exploited this way.
Fix	I'm unsure if the ReadFile function will work with a pascal string instead, but if it will, the pascal string type should be used instead of the null terminated string.
Vulnerability score	7

Manipulate config directory and file

Where?	SDCommon.pas, line 90-116 (ConfigDirectory function)
Description	There is a bug in the SHGetKnownFolderPath system call, that could mean it will only work correctly of compiled on an x64 architecture system. On an x86 architecture system it might not update correctly. Also this system call won't work at all on Windows 8 as it will return an empty string.
Outcome	An attacker with escalated privileges could use this to take control of the config directory and/or brake the code. The fact that the attacker need to already have escalated privileges limits the severity of this vulnerability.
Fix	Use a different system call to get the config directory.
Vulnerability score	5

Open verbose logging

Where?	Everywhere something is logged using the Log() function
Description	Throughout the code, messages and errors are logged extensively to files located at "C:/Projects/WinStaTest/", which is readable by all users.
Outcome	The logs can give an attacker extensive knowledge of the application, because the logs so often contain information of which functions ran successfully/produced an error. An example is on line 106 in SDCommon.pas, where a log entry will be created if SHGetKnownFolderPath fails, *with* a description of SHGetKnownFolderPath and the result error (*which in turn can lead an attacker to exploit the previous vulnerability*).
Fix	Remove, or at least limit the logging in the final version of the application.
Vulnerability score	3

Hard coded paths

Where?	SDCommon.pas, SDInfoProcesses.pas
Description	Several places in the code, one can find hard coded paths, that might prove troublesome. A notorious example can be found on line 90 in SDCommon.pas, where this line can be found: StrPCopy(pszAppDataPath, 'C:/Users/hannol/AppData/Roaming');. This will most certainly create a problem if the current user is not named "hannol".
Outcome	If prerequisite folder structure is not present, this could cause errors and crashes.
Fix	Use environment variables for locations instead, like %APPDAT% or %USERPROFILE%.
Vulnerability score	2

No commenting

Where?	Everywhere
Description	There are no comments (*except for when code is being commented out*). No comments to explain what is going on. True, the code does come with a quite extensive documentation-pdf, but it doesn't help that much when you are reading the code.
Outcome	Anyone needing to understanding the code will have a much harder time doing so. **NOTE:** This is not a vulnerability, it is simply bad practice.
Fix	Make atleast one comment per function, explaining what it is meant to do, and a short description of how.
Vulnerability score	1

PChar

Where?	In most of the files, but only possibly problematic in SDInfoProcess.pas and SDModifiedTokens.pas
Description	PChar is a null terminated string, therefore effectively a C-string, which means that the wrong use of it could be disastrous. One example is in SDInfoProcess.pas, on line 156 in the GetSidUserName function. In this case, UserName is a PChar (*line 146*). UserName is allocated a space of 2049 bytes (*line 153*), then it fills the string with the windows username using the LookupAccountSid call. The problem is, it doesn't include any type of check to make sure that the username is not longer than 2048 characters. This exact thing is also done in many other functions, and it is also used to get the domain name in the exact same way.
Outcome	If an attacker can make LookupAccountSid (*or other calls used throughout the code*) return a username (*or domain name*) longer than 2048 characters, it is a heap overflow vulnerability, and that is pretty bad. But it should be noted that the Windows username and domain name has their length limited by two constants, UNLEN for the username, which is 256, and DNLEN for the domain name, which is 15. But it's still bad practice to not check the values before they are placed into the variables.
Fix	Use a different call than LookupAccountSid, although I don't know if one exists. But because of UNLEN and DNLEN, this is most likely not necessary.
Vulnerability score	3

Sources

1. Microsoft - How To: Perform a Security Code Review for Managed Code (.NET Framework 2.0)
   Link

2. Microsoft - Preventing the exploitation of user mode heap corruption

vulnerabilities
Link

3. Hiddenfeatures.net - Inconsistencies when accessing the registry on Windows x64
Link