WP6

DIGIT B1 - EP Pilot Project 645

Deliverable 2: Summary of the Evaluation of Results

KeePass Password Safe

Specific contract n°226 under Framework Contract n° DI/07172 – ABCIII

October 2016

DIGIT Fossa WP6 – Governance and Quality of Software Code – Auditing of Free and Open Source Software.

Deliverable 2: Summary of the evaluation of results - KeePass Code Review

Author:

everis

an NTT DATA Company

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Commission. The content, conclusions and recommendations set out in this publication are elaborated in the specific context of the EU – FOSSA project.

The Commission does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Commission nor any person acting on the Commission’s behalf may be held responsible for the use that may be made of the information contained herein.

Report Summary

Title	**KeePass Password Safe**		
Project Owner	KeePass Community		
DIGIT Sponsor	EU-FOSSA project		
Author	DIGIT		
Type	Public		
Version	V 0.5	**Version date**	10/10/2016
Reviewed by	EU-FOSSA Team	**Revision date**	08/11/2016
Approved by	European Commission - Directorate-General for Informatics (DIGIT)	**Approval date**	To be approved
		Nº Pages	**29**

Distribution list

Name and surname	Area	Copies
IT contacts	To be identified	To be identified
Communities	KeePass security Team	1

Contents

List of Tables

List of Figures

Acronyms and Abbreviations

AES	Advanced Encryption Standard
CWE	Common Weakness Enumeration
EU-FOSSA	Free and open Source Software Auditing project
FOSS	Free and Open Source Software
IDE	Integrated Development Environment
WP	Work Package

1 INTRODUCTION

1.1. Context

The security of the applications used nowadays has become a major concern for organisations, companies and citizens in general, as they are becoming a more common part of our daily lives, and are being used for business and leisure purposes alike. This information has become the most essential asset to protect, as it includes personal information, internal data, industrial property, etc.

From a security point of view, this new scenario presents many new challenges that need to be addressed in order to protect the integrity and confidentiality of the data managed by the applications and their users. Furthermore, their exposure to the Internet has made them a prime target, due to the value that this private and internal information has.

One of the advantages of Free and Open-Source Software (FOSS) is that its source code is readily available for review by anyone, and therefore it virtually enables any user to check and provide new features and fixes, including security ones. Also, from a more professional point of view, it allows organisations to review the code completely and find the vulnerabilities or weaknesses that it presents, allowing for a refinement of their security and in turn a safer experience for all the users of the applications.

1.2. Objective

The objective of this document is to provide, in a summarised format, the results of the code review ran on the **KeePass Password Safe** software. This goes with a set of recommendations focused on increasing the overall security level of the application. This review is carried out within the EU-FOSSA project, focusing on the security aspects of the software.

The objective of this code review is to examine the **KeePass Password Safe** software, focusing mainly on its security aspects, the risk that they pose to its users and the integrity and confidentiality of the data contained within.

KeePass is a free and open source software tool, which helps to manage passwords in a secure way. All passwords can be stored in one database, which is locked with one master key or a key file. Thus it is only necessary to remember one master password or select the key file to unlock the whole database.

The databases are encrypted using the Advanced Encryption Standard (AES) and Twofish encryption algorithms.

1.3. Scope

The scope of the project is as follows:

Application name	**KeePass Password Safe**				**Review start**	24/08/2016
Code review owner	European Commission - Directorate-General for Informatics (DIGIT)				**Review end**	23/09/2016
Objective	Security Code Review					
Num. Lines	84 622	**Version**	1.31	**Programming language**	C++	
Verification level	✓	1-Oportunistic	✓	2-Standard	✓	3-Advanced
Libraries	• MFC v 9.0 (out of the code review scope, as it is a Microsoft proprietary code.)					
Extensions/plugins	N/A					
Services required	N/A					
Result visibility	✓	Internal	✓	Restricted	✓	Public
Critical notification	During assessment / final report only			Dominik Reichl dominik.reichl@t-online.de		
Categories	Data/Input Management	✓	Error Handling / Information Leakage	✓	Specific C controls	✓
	Authentication Controls	✓	Software Communications	✓	Specific C++ controls	✓
	Session Management	✓	Logging/Auditing	✓	Specific JAVA controls	X
	Authorisation Management	✓	Secure Code Design	✓	Specific PHP controls	X
	Cryptography	✓	Optimised Mode Controls	✓		
Comments	The code review of the KeePass Password Safe includes: 1. KeePass v 1.31 Since version 1.21, KeePass has been developed and compiled using Visual Studio 2008 (with MFC 9.0)					

1.4. Deliverables

1 W*P6 - Deliverable 1: Code Review Results Report – KeePass Password Safe*

2 EXECUTIVE SUMMARY

This document is a high level report of the code review performed for the software KeePass Password Safe (version 1.31), where the assessment of the findings is explained, as well as the recommendations to improve the security of the code.

For technical details please see the complete *"KeePass Code Review Results Report"*[1]

This code review has been carried out following a manual review process aided by two open-source review tools:

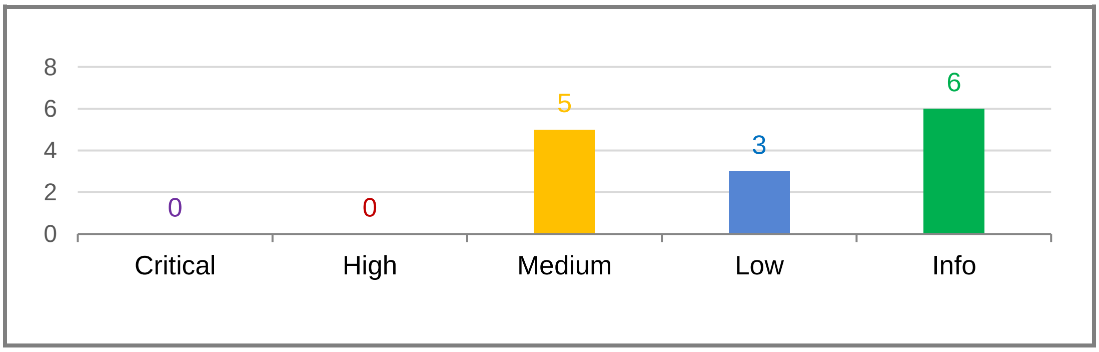
1. **CodeLite**: a Free Open-Source Integrated Development Environment (IDE) for C, it is one of the most used IDE for C and C++, quite easy to install and use.
2. **FlawFinder**: a Free Open-Source code review tool developed by David A. Wheeler, an expert in Free and Open Source Software and secure software development. This tool specialises in finding security flaws in C and C++.

The assessment of the findings pointed out by the code review has been performed form the attackers' point of view, where:

- The '**threat'** is related to the attacker;
- The '**vulnerability'** is related to the potential issue that may be caused and;
- The '**impact'** is related to the consequences of the attack being successful.

From a security point of view, KeePass Password Safe can be considered mature. This fact is corroborated by checking the results:

Figure 1: Risk Level



All of the findings can be solved easily without performing complex developments, and the risk of them being exploited is either low or not possible without modifying the source code itself.

[1] See the EU-FOSSA Community on Joinup: link

Furthermore, these vulnerabilities are hard to exploit. This makes it difficult to take advantage of the vulnerabilities in normal environments. However, in custom implementations this needs to be double-checked, as oversights or changes may make these vulnerabilities directly exploitable by attackers.

It is important to notice that this code review does not guarantee that all of the vulnerabilities are detected. Some security issues can remain undetected, therefore it is advisable to carry out other security tests to complement this code review.

As far as the he prioritisation is concerned, it is proposed according to their criticality: medium risk findings should be resolved in the short-term, low risk findings in the mid-term, and the informative ones in the long-term.

3 CODE REVIEW ENVIRONMENT

In order to carry out the code review and analysis, there was a need to develop a specific code review environment with the necessary tools (including both automated and manual tools).

For the manual code review, an IDE (Integrated Development Environment) was used:

codelite	**CodeLite**: a FOSS application that is light, user-friendly and has a high maturity level (version: 9). It is a cross-platform (supporting Windows, the major Linux distributions and Mac OS). It supports the following languages: • C • C++ • JavaScript • PHP One of the main reasons why it was chosen: its excellent support of C and C++ code. Source: http://www.codelite.org/

Alongside this IDE, an automated tool was also used to help complement the findings and potential results:

C/C++	**FlawFinder**: a FOSS automatic secure code review tool mainly focused on C and C++ code. It supports Linux and Unix-based operating systems mainly, although it can also be run on Windows when compiled using Cygwin. It is compatible with Common Weakness Enumeration (CWE), providing useful feedback on any finding. As a side note, this tool was developed by David A. Wheeler, an authority in the fields of secure software development and open-source software. Source: http://www.dwheeler.com/flawfinder/

4 SECURITY ASSESEMENT

There were a total of 10 batches with findings in 14 controls. These controls are grouped based on their overall risk level:

- **Medium Risk**
  - ***CBC-VMG-008***
  - ***CBC-MEM-005***
  - ***CBC-MSC-001***
  - ***CPP-MSC-001***
  - ***CBC-ENV-004***

- **Low Risk**
  - ***SCD-FWK-001***
  - ***SCD-VTY-002***
  - ***CBC-VMG-023***

- **Informational Risk**
  - ***LOG-CFG-004***
  - ***CPP-VMG-008***
  - ***CPP-OOP-001***
  - ***EHI-EHD-002***
  - ***CPP-VMG-007***
  - ***CPP-OOP-007***

After a detailed review and following information exchange with KeePass point of contact, it was determined that some of these findings are controlled within the code, so the risk is mitigated and they do not represent a security vulnerability. However, they are still mentioned here to consider in future developments. The findings are:

- ***CBC-MSC-001***
- ***CBC-ENV-004***
- ***CPP-MSC-001***
- ***EHI-EHD-002***
- ***CPP-VMG-007***
- ***CPP-OOP-007***

4.1. Medium Risk Findings

Table 1: Security Assessment of CBC-VMG-008

CBC-VMG-008	Ensure that floating-point conversions are within the range of the new type		**Medium**
Finding	In floating-point value conversions, if the destination type is smaller than the origin, it must be verified that the value can fit in the new type.	**Threat**	Medium
		Vulnerability	Low
		Impact	Medium
Detections	**File/s:**	**Line/s:**	
	`%root%\WinGUI\NewGUI\BCMenu.cpp`	2686, 2749	
Assessment	There are no error management controls of the return method GetUpperBound().Any errors in the type conversion must be controlled and managed. Thus the possible error or exceptions that this function can trigger must be controlled. • **Threat (Medium)**: to exploit this functionality, it is necessary to have access to the code. • **Vulnerability (Low)**: it is hard to find this vulnerability and to exploit it as well. It is also not publicly known. • **Impact (Medium)**: it can only affect local computers. The result of its occurrence is a loss of data integrity and precision. Related vulnerability code: **N/A.**		

Table 2: Security Assessment of CBC-MEM-005

CBC-MEM-005	Allocate sufficient memory for an object		**Medium**
Finding	It is necessary to guarantee that storage for strings has sufficient space available for character data and consequently to allocate sufficient memory for an object.	**Threat**	Medium
		Vulnerability	Medium
		Impact	Medium
Detections	**File/s:**	**Line/s:**	
	`%root%\WinGUI\PwSafe.cpp`	496	
Assessment	The '_tcslen' function is not capable of handling strings that are not \0-terminated. If such a string is passed without \0-termination, the function will execute an over-read and potentially cause the application to crash if no further controls are in-place. • **Threat (Medium):** to exploit this functionality, it is necessary to have access to the code. On the other hand, this finding can be detected using automatic tools. • **Vulnerability (Medium):** these functions do not have any control or filtering functionality to check the parameter received. So it can receive a non \0-terminated string. • **Impact (Medium)**: it can only affect local computers. Related vulnerability code: **CWE-126**.		

Table 3: Security Assessment of CBC-ENV-004

CBC-ENV-004	Do not call system() function		**Medium**
Finding	The use of ‘system()’ functions can result in exploitable vulnerabilities, allowing the execution of arbitrary system commands.	**Threat**	Medium
		Vulnerability	Medium
		Impact	Medium
Detections	**File/s:**	**Line/s:**	
	`%root%\WinGUI\UpdateInfoDlg.cpp`	144	
	`%root%\WinnGUI\PwSafeDlg.cpp`	627, 635, 6418, 8710	
	`%root%\WinGUI\NewGUI\XHyperLink.cpp`	596	
Assessment	**shellExecute:** This causes a new program to execute and it is difficult to use safely. If the path it is not provided, the use of ‘system()’ functions to execute a command could potentially execute the wrong application with the same filename. It is recommended to use an alternative function that controls this eventuality. • **Threat (Medium)**: to exploit this functionality, it is necessary to have access to the code. On the other hand, this finding can be detected using automatic tools. • **Vulnerability (Medium)**: these functions do not have any control or filtering functionality, thus being able of potentially executing any command passed through them. • **Impact (Medium)**: it can only affect local computers, therefore remote programs cannot be accessed unless previously downloaded. Related vulnerability code: **CWE-78**. This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** the security of the code because **is not related to the main functionality of the software (encryption)**. However is still mentioned to create awareness about it.		

Table 4: Security Assessment of CBC-MSC-001

CBC-MSC-001	Do not use the rand() function to generate pseudorandom numbers		**Medium**
Finding	The **rand()** function should not be used to generate random numbers, as they are predictable due to the short cycle of numbers that it uses.	**Threat**	Low
		Vulnerability	Medium
		Impact	Medium
Detections	**File/s:**	**Line/s:**	
	`%root%\KeePassLibCpp\SysSpec_Windows\NewRandom.cpp`	74,76,78	
	`%root%\WinGUI\Util\WinUtil.cpp`	954	
Assessment	**rand():** the '**rand()**' function is no longer safe, as it does not provide enough entropy to be considered apt for security applications. The use of an alternative function is recommended, such as '**random()**'. • **Threat (Low)**: to exploit this functionality, it is necessary to have access to the code. Furthermore the attacker should have advanced coding and networks skills. On the other hand, this finding can be detected using automatic tools. • **Vulnerability (Medium)**: the weak entropy of the rand() function leads to predictable random numbers • **Impact (Medium)**: it is easier to guess the random number when using this function instead of other similar. Related vulnerability code: **CWE-327**. This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** the security of the code because **is not related to the main functionality of the software (encryption)**. However is still mentioned to create awareness about this function. The usage of **rand()** must be ceased in future developments.		

Table 5: Security Assessment of CPP-MSC-001

CPP-MSC-001	Do not use **std::rand()** to generate pseudorandom numbers	**Medium**	
Finding	Using the **std::rand()** function could lead to predictable random numbers.	**Threat**	Low
		Vulnerability	Medium
		Impact	Medium
Detections	**File/s:**	**Line/s:**	
	%root%\WinnGUI\PwSafeDlg.cpp	654	
Assessment	This function is not sufficiently random for security-related functions such as key and nonce creation. • **Threat (Low)**: to exploit this functionality, it is necessary to have access to the code. Furthermore the attacker should have advanced coding and networks skills. On the other hand, this finding can be detected using automatic tools. • **Vulnerability (Medium)**: the weak entropy of the std::rand() function leads to predictable random numbers • **Impact (Medium)**: it is easier to guess the random number when using this function instead of another similar one. Related vulnerability code: **CWE-76**. This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** the security of the code because **is not related to the main functionality of the software (encryption)**. However is still mentioned to create awareness about it. However is still mentioned to create awareness about this function and still mentioned in here. The usage of **std::rand()** must be ceased in future developments.		

4.2. Low Risk Findings

Table 6: Security Assessment of SCD-FWK-001

SCD-FWK-001	All frameworks and third party components are up-to-date		**Low**
Finding	**RegCreateKey**: this function is provided only for compatibility with 16-bit versions of Windows. Applications should use the **RegCreateKeyEx** function.	**Threat**	Medium
		Vulnerability	High
		Impact	Low
Detections	**File/s:**	**Line/s:**	
	`%root%\WinGUI\PwSafe.cpp`	328	
Assessment	The use of obsolete functions is discouraged unless strictly necessary due to legacy concerns. These functions are known and easily discoverable using automated tools. • **Threat (Medium)**: it is publicly known and detectable, but it can only be indirectly exploited. • **Vulnerability (High)**: deprecated functions usually have well-known flaws that can be exploited. • **Impact (Low)**: it only affects a limited part of the application. Related vulnerability code: **CWE-676**.		

Table 7: Security Assessment of SCD-VTY-002

SCD-VTY-002	On division operations, check that the divisor does not equal zero		**Low**
Finding	The size of the 'lpstrText' variable is not controlled against invalid or zero values.	**Threat**	Low
		Vulnerability	Low
		Impact	Medium
Detections	**File/s:**	**Line/s:**	
	%root%\WinGUI\NewGUI\BCMenu.cpp	1011	
Assessment	In division operations, the values must be checked to ensure that no invalid values are operated and that no value is divided by zero. • **Threat (Low)**: the attacker needs access to the code and specific skills to exploit this vulnerability. • **Vulnerability (Low)**: it is hard to find and to exploit this vulnerability, but it is a wrong coding practice. • **Impact (Low)**: it only affects in the cases that the IpstrText function returns a 0 value. Related vulnerability code: **N/A.**		

Table 8: Security Assessment of CBC-VMG-023

CBC-VMG-023	Do not read uninitialised memory		**Low**
Finding	The 'szTitle' variable is not initialised before accessing its content. The 'm_value' variable is not initialised before accessing its content.	**Threat**	Low
		Vulnerability	Low
		Impact	Low
Detections	**File/s:**	**Line/s:**	
	%root%\WinGUI\Util\SendKeys.cpp	585	
Assessment	Local, automatic variables assume unexpected values if they are read before they are initialised. • **Threat (Low)**: the attacker needs to have access to specific resources and must have advanced computer skills to exploit this flaw. • **Vulnerability (Low)**: it is hard to discover and to exploit. • **Impact (Low)**: can lead to unexpected behaviour when accessing the unexpected values of a non-initialised variables. Related vulnerability code: **N/A.**		

4.3. Informational Risk Findings

Table 9: Security Assessment of EHI-EHD-002

EHI-EHD-002	Try-catch-finally block			**Info**
Finding	The '**finally**' statement should always be present, and used to release system resources and perform other clean actions. If any of these additional actions within the finally block can throw exceptions, these need to be captured within a new **try-catch-finally** block.	**Threat** **Vulnerability** **Impact**	Low Low Low	
Detections	**File/s:**		**Line/s:**	
	%root%\WinGUI\Util\SessionNotify.		65	
Assessment	Those programming languages that have the 'try-catch-finally' structure have to be used correctly. The 'finally' statement should always be present, and used to release system resources and perform other clean actions. • **Threat (Low)**: users cannot directly take advantage of this vulnerability. • **Vulnerability (Low)**: risk of memory exhaustion or of leaving a component in an undefined state. • **Impact (Low)**: can cause an application to freeze or even crash. Related vulnerability code: **N/A.** This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** the security of the code because **is not related to the main functionality of the software (encryption)**. However is still mentioned to create awareness about it.			

Table 10: Security Assessment of CPP-VMG-007

CPP-VMG-007	Guarantee that container indexes/iterators are within a valid range		**Info**
Finding	The 'pos' variable, used to access array positions, is manually incremented, and no range controls are in-place to ensure that the value remains valid and within bounds. A misuse of this variable can lead to an improper behaviour, even a program crash.	**Threat**	Low
		Vulnerability	Low
		Impact	Low
Detections	**File/s:**	**Line/s:**	
	%root%\KeePassLibCpp\Details\PwFileImpl.cpp	294, 299, 305	
Assessment	Ensuring that array references are within the bounds of the array is almost entirely the responsibility of the programmer when using standard template library vectors. • **Threat (Low)**: the index used to go through the array is not commonly obtained from direct user input. • **Vulnerability (Low)**: the lack of length control can be exploited to cause a lack of memory or even a crash of the application. • **Impact (Low)**: it would only affect a section of the code and it would be complex for it to cause severe damages. Related vulnerability code: **N/A.** This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** the security of the code because **is not related to the main functionality of the software (encryption)**. However is still mentioned to create awareness about it.		

Table 11: Security Assessment of CPP-OOP-007

CPP-OOP-007	Prefer special member functions and overloaded operators to C Standard Library functions	**Info**	
Finding	The ‘**memset(…)**’ function should not be used to initialise objects, as it may not properly initialise the value representation of the object. Improper initialisation leads to class invariants that do not apply in later uses of the object.	**Threat**	Low
		Vulnerability	Low
		Impact	Medium
Detections	**File/s:**	**Line/s:**	
	%root%\WinGUI\NewGUI\BtnST.cpp	503	
	%root%\WinGUI\NewGUI\CBMenu.h	71	
Assessment	Several C standard library functions perform byte wise operations on objects. • **Threat (Low)**: the attacker needs special access or specific resources and must have advanced coding skills to exploit this flaw. • **Vulnerability (Low)**: it is hard to find and to exploit this vulnerability. • **Impact (Medium)**: the improper initialisation leads to class invariants that do not apply in later uses of the object. It can lead to an application malfunction. Related vulnerability code: **N/A.** This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** the security of the code because **is not related to the main functionality of the software (encryption)**. However is still mentioned to create awareness about it.		

Table 12: Security Assessment of LOG-CFG-004

LOG-CFG-004	Logging exceptions	**Info**	
Finding	There is no logging functionality implemented in the catch(…) block; therefore any exception captured is not logged, nor is any trace left of this event recorded	**Threat**	Low
		Vulnerability	Low
		Impact	Low
Detections	**File/s:**	**Line/s:**	
	%root%\KeePassLibCpp\Details\PWFindImpl.cpp	From 51 to 60	
Assessment	Exceptions must be logged in a proper manner in case they are not to be thrown. • **Threat (Low)**: users cannot directly take advantage of this vulnerability. • **Vulnerability (Low)**: it is hard to discover and its exploitation is theoretical • **Impact (Low)**: its exploitation does not directly damage the system. Related vulnerability code: **N/A.**		

Table 13: Security Assessment of CPP-VMG-008

<table>
<tr><th>CPP-VMG-008</th><th colspan="2">Guarantee that library functions do not form invalid iterators</th><th colspan="2">Info</th></tr>
<tr><td rowspan="3">Finding</td><td colspan="2" rowspan="3">**Memory operations:** Memory operations done using **memcpy**, are used several times without checking the size of the source and destination.
The function does not verify if the destination container is able to hold the element to be copied via **memcpy(…)**.</td><td>Threat</td><td>Low</td></tr>
<tr><td>Vulnerability</td><td>Medium</td></tr>
<tr><td>Impact</td><td>Low</td></tr>
<tr><td rowspan="2">Detections</td><td colspan="2">File/s:</td><td colspan="2">Line/s:</td></tr>
<tr><td colspan="2">%root%\WinGUI\AddEntryDlg.cpp</td><td colspan="2">1071</td></tr>
<tr><td>Assessment</td><td colspan="4">Copying data into a container that is not large enough to hold the original data will result in a buffer overflow.
• **Threat (Low)**: the code would need to be modified directly in order to exploit this vulnerability, although it is discoverable with automated tools
• **Vulnerability (Medium)**: this vulnerability entails the known risk of losing the integrity of the memory locations being managed within the function (or those accessed by it).
• **Impact (Low)**: it is complex to exploit this vulnerability, but the lack of a size control for arrays in the code can result in an overflow.
Related vulnerability code: **N/A.**</td></tr>
</table>

Table 14: Security Assessment of CPP-OOP-001

<table>
<tr><th>CPP-OOP-001</th><th colspan="2">Do not invoke virtual functions from constructors or destructors</th><th colspan="2">Info</th></tr>
<tr><td rowspan="3">Finding</td><td colspan="2" rowspan="3">**CShutdownBlocker** is declared as a virtual function in the header file.</td><td>Threat</td><td>Low</td></tr>
<tr><td>Vulnerability</td><td>Low</td></tr>
<tr><td>Impact</td><td>Low</td></tr>
<tr><td rowspan="2">Detections</td><td colspan="2">File/s:</td><td colspan="2">Line/s:</td></tr>
<tr><td colspan="2">%root%\WinGUI\Util\ShutdownBlocker.cpp</td><td colspan="2">60</td></tr>
<tr><td>Assessment</td><td colspan="4">A virtual function is invoked from a constructor within an inherited class.
Attempting to call a derived-class function from a base class under construction is dangerous: th
e derived class has not had the opportunity to initialise its resources, which is why calling a virtual function from a constructor does not result in a call to a function in a more derived class.
• **Threat (Low)**: it needs special access and skills to get to the vulnerability
• **Vulnerability (Low)**: it is hidden and hard to exploit.
• **Impact (Low)**: it can lead to an unexpected behaviour.
Related vulnerability code: **N/A.**</td></tr>
</table>

5 RECOMMENDATIONS

5.1. Details

The code review has evaluated the security level of the application analysed and identified vulnerabilities and weaknesses that can put it at risk.

In this section, for each finding a corresponding recommendations is given to help increase the overall security level of the application.

Table 15 shows the recommendations that should be implemented for each of the findings described and assessed in Section 4.

Table 15: Controls with Findings and Recommendations/Specific Solutions

Controls with Findings	Recommendation/Specific Solution
CBC-VMG-008	**R01_CBC-VMG-008** **Recommendation:** There must be a control within the code to check the return method **GetUpperBound** in order to manage possible errors or exceptions.
CBC-MEM-005	**R02_CBC-MEM-005** The '**_tcslen**' function is not capable of handling strings that are not \0-terminated. The code must have controls to ensure that the string is passed with \0-termination, or add \0 at the end of the string if necessary..
CBC-ENV-004	**R03_CBC-ENV-004** This issue is controlled programmatically within the KeePass code. Before deciding to change it, one must take into account the risk of adding more complexity to the code, and ensure that the mitigation of the risk that is provided via the code is maintained. Where more control is required on what will be executed use **ShellExecuteEx** instead of **ShellExecute**. **ShellExecuteEx** provides additional functionality. If you don't require any of the functionality provided by **ShellExecuteEx**; keep it simple and stick with **ShellExecute**.

Controls with Findings	Recommendation/Specific Solution
CBC-MSC-001	**R04_CBC-MSC-001** This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** the security of the code because **is not related to the main functionality of the software (encryption)**. However is still mentioned to create awareness about this function and as an informational issue. The usage of **rand()** must be ceased in future developments. Before deciding to change it, one must take into account the risk of adding more complexity to the code, and ensure that the mitigation of the risk that is provided via the code is maintained. **Recommendation:** The **rand()** function does not provide enough entropy. The usage of other functions such as '**random()**' is recommended.
CPP-MSC-001	**R05_CPP-MSC-001** This issue is controlled programmatically within the KeePass code. The issue in this case **does not affect** at all the security of the code because **is not related to the crucial functionality of the software (encryption)**. However is still mentioned to create awareness about this function and as an informational issue. The usage of **std::rand()** must be ceased in future developments. Before deciding to change it, one must take into account the risk of adding more complexity to the code, and ensure that the mitigation of the risk that is provided via the code is maintained. **Recommendation:** The **std::rand()** function is not sufficiently random for security-related functions. Instead it is recommended to implement a code such as: std::default_random_engine engine; engine.seed(n); std::uniform_int_distribution<> distribution; auto rand = [&](){ return distribution(engine); }

Controls with Findings	Recommendation/Specific Solution
EHI-EHD-002	**R06_EHI-EHD-002** This issue is controlled programmatically within the KeePass code. Before deciding to change it, one must take into account the risk of adding more complexity to the code, and ensure that the mitigation of the risk that is provided via the code is maintained. **Recommendation:** The '**finally**' statement should always be present, and used to release system resources and to perform other clean actions. If any of these additional actions can throw exceptions, these need to be captured within a new **try-catch-finally** block.
SCD-FWK-001	**R07_SCD-FWK-001** **Specific Solution:** The usage of deprecated functions is discouraged. ○ **RegCreateKey**: this function is provided only for compatibility with 16-bit versions of Windows. Applications should use the **RegCreateKeyEx** function.
SCD-VTY-002	**R08_SCD-VTY-002** **Recommendation:** Check the '**lpstrText**' variable to ensure that no invalid or zero values are received.
CBC-VMG-023	**R09_CBC-VMG-023** **Recommendation:** Always initialise variables prior to accessing their content. In other case it will lead to an unexpected behaviour.
CPP-VMG-007	**R10_CPP-VMG-007** This issue is controlled programmatically within the KeePass code. Before deciding to change it, one must take into account the risk of adding more complexity to the code, and ensure that the mitigation of the risk that is provided via the code is maintained. **Recommendation:** Set controls in place to ensure that the values used in indexes or iterators remain within the valid range. There must be controls in place to ensure that the values used in indexes or iterators are within the valid range.

Controls with Findings	Recommendation/Specific Solution
CPP-OOP-007	**R11_CPP-OOP-007** This issue is controlled programmatically within the KeePass code. Before deciding to change it, one must take into account the risk of adding more complexity to the code, and ensure that the mitigation of the risk that is provided via the code is maintained. **Recommendations:** The behaviour of **std::memset()** can be avoided with other options: • **std::memset** may be optimised if the object modified is not accessed again for the rest of its lifetime. • Defining an assignment operator that is used instead. • Replacing the call to this function with a default-initialised copy-and-swap operation called **clear()**. • Defining an equality operator that is used instead.
LOG-CFG-004	**R12_LOG-CFG-004** **Recommendation:** Log any exception captured that will not be thrown to have a record of the event.
CPP-VMG-008	**R13_CPP-VMG-008** **Recommendation:** Set controls in place to ensure that the destination container can address the element to be copied without losing integrity in memcopy() operations
CPP-OOP-001	**R14_CPP-OOP-001** **Specific Solution:** Call a nonvirtual, private member function from constructors, or destructors instead of calling a virtual function

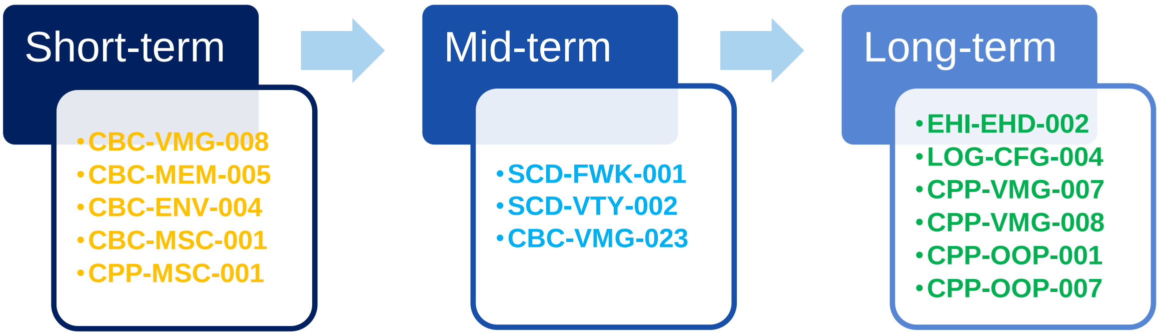
5.2. Prioritisation

Once the severity of the findings found during the code review has been determined, the following step in the methodology includes a prioritisation process and an action plan definition. This allows the stakeholders and project owners to identify the most urgent findings that need to be solved, allowing the planning of the fixes as part of the standard development cycle.

For this purpose, the following priority sets have been established. The main consideration is to solve the Medium findings identified during this code review in the short-term. The low findings should be targeted in the mid-term, and finally the Informative findings do not require any priority.

Thus, the following graph has been generated:

Figure 2: Priority levels



Short-term
•CBC-VMG-008
•CBC-MEM-005
•CBC-ENV-004
•CBC-MSC-001
•CPP-MSC-001
Mid-term
•SCD-FWK-001
•SCD-VTY-002
•CBC-VMG-023
Long-term
•EHI-EHD-002
•LOG-CFG-004
•CPP-VMG-007
•CPP-VMG-008
•CPP-OOP-001
•CPP-OOP-007