WP6

DIGIT B1 - EP Pilot Project 645

Deliverable 1: Code Review Results Report

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 1: KeePass Code Review Results Report

Author:

Disclaimer

Deliverable 1: KeePass Code Review Results Report

Report Summary

Title	**KeePass Password Safe**		
Project Owner	KeePass Community		
DIGIT Sponsor	EU-FOSSA project		
Author	DIGIT		
Type	Public		
Version	V 0.3	**Version date**	10/10/2016
Reviewed by	EU-FOSSA Team	**Revision date**	02/11/2016
Approved by	European Commission - Directorate-General for Informatics (DIGIT)	**Approval date**	To be approved
		Nº Pages	**43**

Distribution list

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

Contents

List of Tables

List of Figures

Acronyms and Abbreviations

DG	Directorate General
EC	European Commission
FOSS	Free and Open Source Software
FOSSA	Free and open Source Software Auditing
GUI	Graphic User Interface
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. Applications are becoming a more common part of our daily lives, and are being used for business and leisure purposes alike. The information managed by these applications 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, the exposure of the applications to the Internet has turned them into 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 to find the vulnerabilities or weaknesses that it presents, allowing for a refinement of their security and ending up in a safer experience for all the users of the applications.

1.2. Objective

The objective of this document is to provide the results of the code review of **KeePass Password Safe** software. This review is carried out within the EU-FOSSA (Free and Open-Source Software Auditing) 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 that helps 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 to select the key file to unlock the whole database.

The databases are encrypted using the 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++**
Code Review Mode	✓ 1-Managed	✓ 2-Defined	✓ 3-Optimised		
Libraries	• MFC v 9.0 (It is not within the scope of the code review because this is a proprietary code from Microsoft)				
Extensions/plugins	N/A				
Services required	N/A				
Result visibility	✓ Internal	✓ Restricted	✓ Public		
Critical notification	During assessment	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 WP2 - *Deliverable 11: Design of the methods for performing the code reviews List of methods for communicating the results of code reviews*

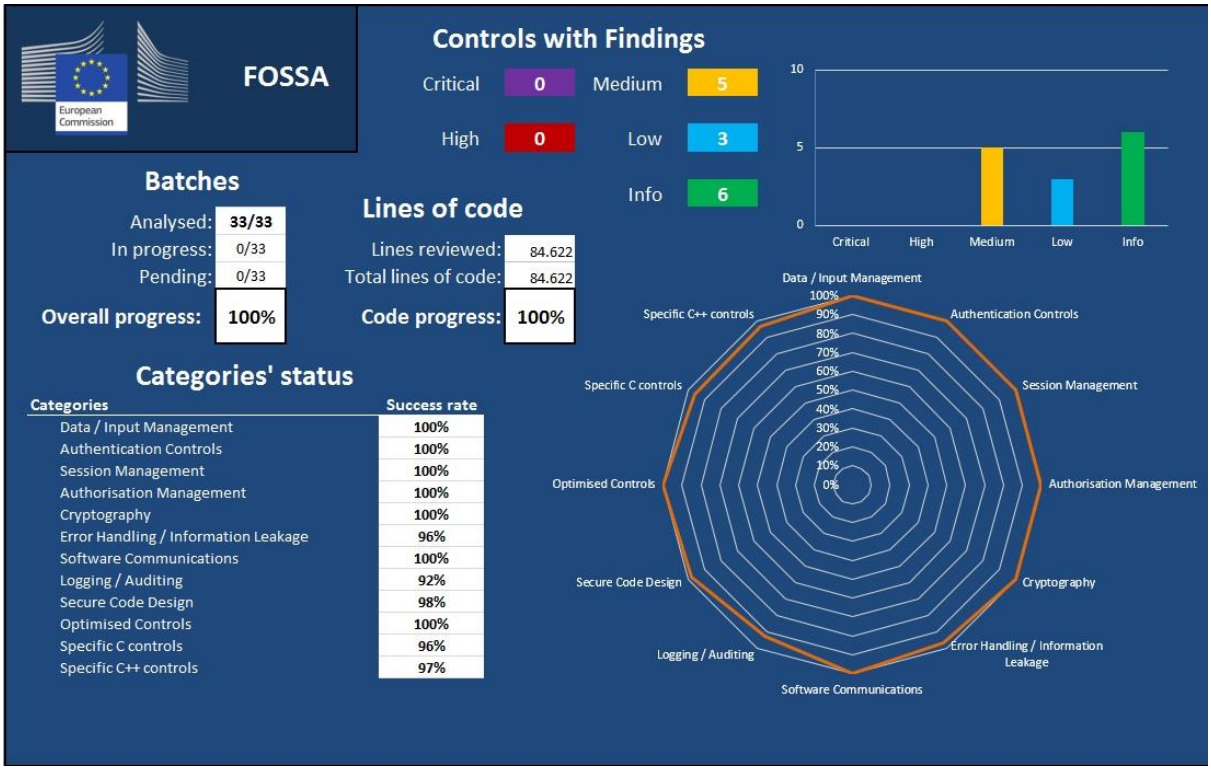
2 EXECUTIVE SUMMARY

The results of the tests and controls evaluated in this code review provided a number of relevant findings regarding the application reviewed. A general overview is depicted in Figure 1, which shows the findings and their impact on the categories included in the analysis.

84 622 lines were reviewed, comprising the totality of Lines of Code of the application. To optimise the process, the total was divided in 33 sets (or 'batches') of code, and distributed among the EU-FOSSA project code review team.

Figure 1 shows an overview of the results with the number of failed controls (findings) ordered by their assessment value.

Figure 1: General overview



In relation to the control categories, findings were discovered in these categories:

- **Error Handling / Information Leakage**
- **Logging / Auditing**
- **Secure Code Design**
- **Specific C controls**
- **Specific C++ controls**

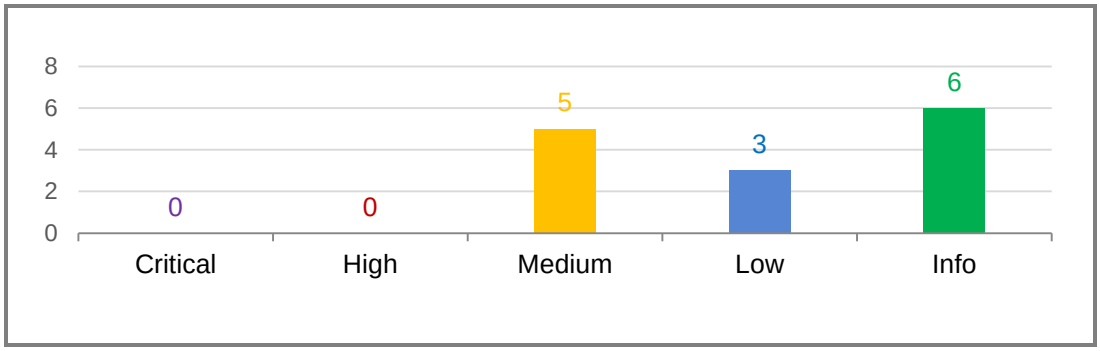
The total number of findings **(14)** can be considered as low when compared to the **218 controls** reviewed.

The remaining categories of controls successfully passed the review with no relevant findings.

No **critical or high-risk findings were detected**. Among the remaining findings, **five medium** and **three low** risk results were detected. The remaining six were of an **informative** nature.

A summary of the findings is depicted in Figure 2, which compares the failed controls found and indicates their distribution within the different risk levels.

Figure 2: Risk Level



This shows that the impact of the findings varies from one risk level to another. It is recommended to give priority to the following findings found during the code review:

Critical findings ✓

No critical vulnerabilities were found in this code review.

High-risk findings ✓

No high risk vulnerabilities were found in this code review.

Medium-risk findings

The findings categorised as medium risk can have a reasonable impact at a technical and business level; their resolution is recommended although it does not need to be prioritised. The details of the vulnerabilities found are:

- **Ensure that floating-point conversions are within range of new type (id: CBC-VMG-008):** Any errors in the type conversion must be controlled and managed: There are no error management controls of the return method GetUpperBound().

- **Allocate sufficient memory for an object (CBC-MEM-005):** 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 will potentially cause the application to crash if no further controls are in-place. Related CWE: **CWE-126**.

- **Do not call the System() function (id: CBC-ENV-004)**: The use of the system() functions can result in exploitable vulnerabilities that would allow the execution of arbitrary system commands. **shellExecute**: This causes a new program to execute and it is difficult to use safely. This situation is controlled within the code. All hyperlink UI controls in KeePass have well-defined, fixed URLs. However the control and the findings are still in this report, under the section Findings Controlled Programmatically, due to its severity and to keep this in mind for future developments.

- **Do not use the rand() function to generate pseudorandom numbers (CBC-MSC-001): 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 alternative functions is recommended, such as 'random()'. Related CWE**: CWE-327.** This situation is controlled within the code and rand() is only used in situations where weak random numbers are sufficient. However the control and the findings are still in this report, under the section Findings Controlled Programmatically, due to its severity and to keep this in mind for future developments.

- **Do not use std::rand() for generating pseudorandom numbers (CPP-MSC-001): std::rand():**This function is not sufficiently random for security-related functions such as key and nonce creation. Related CWE: **CWE-327.** This situation is controlled within the code and std::rand() is only used in situations where weak random numbers are sufficient. However the control and the findings are still in this report, under the section Findings Controlled Programmatically, due to its severity and to keep this in mind for future developments.

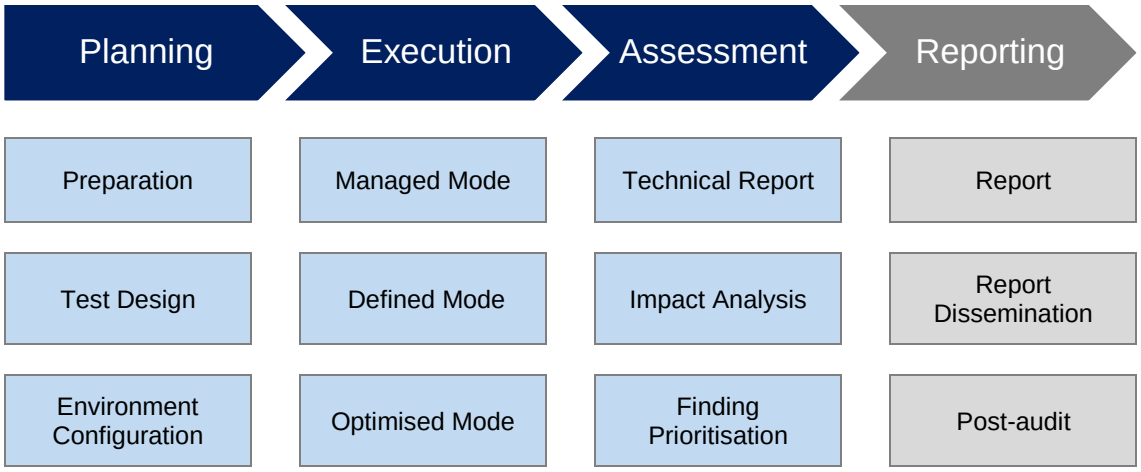
The rest of the findings categorised as either low or informative are still relevant and should be resolved as well. However, due to their low impact on the overall security, there is no need to fix them in the short term.

3 METHODOLOGY

The methodology followed to carry out the code review is summarised in Figure 3. This methodology covers from the initial planning phase to an optional post-audit support phase.

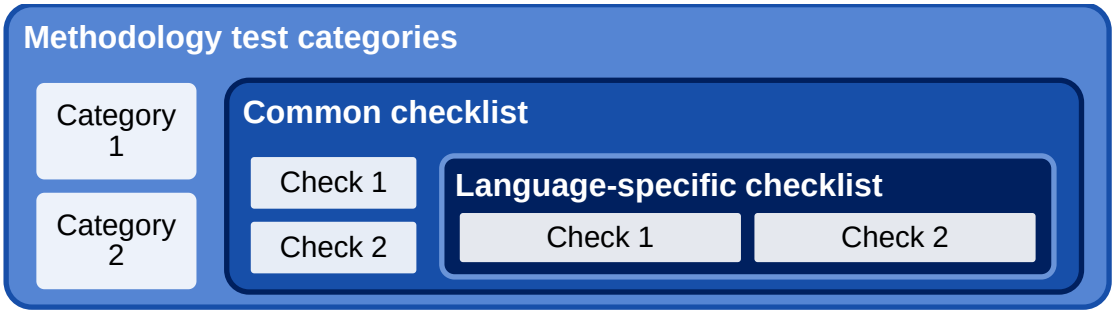
Furthermore, each one of these phases is divided into several mayor tasks.

Figure 3: Methodology phases



In the execution phase, a set of controls is checked by the code reviewers in order to properly verify the security and stability of the code. These controls and checks are grouped in a checklist presented in *Section 4.3. Overview of Results*, to facilitate the viewing of the findings.

Figure 4: Test category levels



As seen in Figure 4, there are two main groups of controls: the common ones (applicable regardless of the language of the code) and language-specific controls (for C, C++, JAVA or PHP). A combination of both should be used in any code review to ensure the most accurate results (explained in WP2 - Deliverable 11: *Design of the methods for performing the code reviews*).

3.1. Planning

The first phase of the methodology covers the information gathering activities needed in order to properly plan and carry out the code review. This includes the compilation of basic information about the code to be reviewed, an analysis of the applicable controls and the preparation of the testing environment if any specific requirements are demanded by the particularities of the code.

This information was obtained from the stakeholders requesting the code review and from the developers or IT maintainers where applicable. Once this phase is finished, all needs should have been met in order to start the test cases.

To further organise this phase, three main activities have been defined:

- **Preparation**: this activity comprises all the interviews, meetings and information gathering activities needed to properly define the scope, objectives and needs of the code to be reviewed.

- **Test Design**: once the scope, objectives and custom needs of the code have been identified, the next logical step is to establish the test cases that are going to be considered in order to achieve the objectives that have been set. This is reflected in the checklist, indicating those cases that are not applicable.

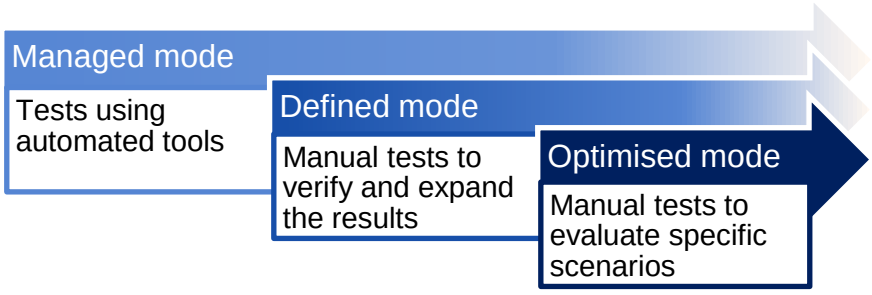
- **Environment Preparation**: before starting the next phase, it is necessary to ensure that the testing environment is prepared to carry out the tests selected during the previous activity. This includes the installation and configuration of the tools.

3.2. Execution

The next phase covers the execution of the test cases selected for the code review in the previous phase, taking into consideration the scope, objectives and constraints set.

The execution process was divided into three sequential phases, each providing data as input for the next one, as depicted in Figure 5. All of them were carried out by the code review team, using both automated and manual tools.

Figure 5: Code review execution order



To further organise this phase, three main activities have been defined:

- **Managed mode**: this activity covers the execution of the automated tools selected for the analysis of the code. The following categories were analysed:
    - *Data/Input Management (DIM)*: The data entry points of an application, service or library are one of the weakest points of the code, and they must be controlled against unexpected values. The subcategories covered are as follows:
        - File Input / Output Management (FIM)
        - Data stream management (DSM)
        - Character encoding management (CEM)
        - Input validation and sanitisation (IVS)
        - Sensitive Data Management (SDM)
        - Entry point validation (EPV)
        - XML schema validation (XSV)
    - *Authentication Controls (AUT)*: It covers any aspect related to the process during which the code reviews and verifies the identity of another entity, such as a user. The subcategories covered are as follows:
        - Authentication verification (AUV)
        - Password policy usage (PPU)
        - Credential storage security (CST)
        - User account protection (UAP)
        - Password recovery process (PRP)
    - *Session Management (SMG)*: It covers all parts of the protection and management of user sessions once they are authenticated against the solution. The subcategories covered are as follows:
        - Session creation (SCP)
        - Session ID management (SID)
        - Session lifecycle (SLC)
        - Session logout (LGP)

- *Authorisation Management (ATS)*: This process is designed to ensure that when a user or entity correctly authenticates against the application, it gets the proper privileges assigned to it. The subcategories covered are as follows:
  - Access control system (ACS)
  - Privilege revision (PRV)
- *Cryptography (CPT)*: Covers all aspects related to the protection via encryption of the information and data in transit and at rest. The subcategories covered are as follows:
  - Credential protection at rest (CPR)
  - Cryptographic configuration (CRC)
- *Error Handling/Information Leakage (EHI)*: The information provided by the application errors, page metadata and sample content must be filtered to avoid a leakage of sensitive information. The subcategories covered are as follows:
  - Information leakage (INL)
  - Sample files (SFL)
  - Error handling (EHD)
- *Software communications (COM)*: it comprises those functions that manage and control network connections, including sockets and protocol functions. The subcategories covered are as follows:
  - HTTP Secure Management (HSM)
- *Logging/Auditing (LOG)*: The logs generated by an application are a superb source of information about its contents, workings and potential weaknesses. The subcategories covered are as follows::
  - Log configuration management (CFG)
  - Log generation (GEN)
  - Log sensitive information (LSI)
- *Secure Code Design*: There are several aspects related to the application itself and the technologies and frameworks used for its implementation. The subcategories are as follows:
  - Framework requirements (FWK)
  - Variable types / operations (VTY)
  - Expressions/Methods (EXM)

- **Defined Mode**: once the managed mode activity is finished, the code review team generates a set of results, by complementing these results with a full manual review of the applicable controls.

- **Optimised Mode**: The final part of the execution phase focuses on those sections of the application found to be most at risk, alongside several more specific tests that require further evaluation.

They are divided into the following subcategories:

- Concurrency (CCR)
- Denial of Service (DOS)
- Memory and resource management (MRM)
- Code Structure (COS)
- Role-privilege matrix (RPM)

The optimised mode covers the set of language-specific (C, C++, JAVA and PHP) controls, and other controls related to code unique particularities. The language specific controls for C (CBC) are divided into the following subcategories:

- PreProcessor (PRE)
- Variable Management (VMG)
- Memory Management (MEM)
- File I/O Management (FIO)
- Environment (ENV)
- Signal and Error Handling (SEH)
- Concurrency (CON)
- Miscellaneous (MSC)

The language specific controls for C++ (CPP) are divided into the following subcategories:

- Variable Management (VMG)
- Memory Management (MEM)
- File I/O Management (FIO)
- Exceptions and Error Handling (EEH)
- Object Oriented Programming (OOP)
- Concurrency (CON)
- Miscellaneous (MSC)

3.3. Assessment

This phase covers the analysis and evaluation of the findings identified in the previous phase, with the objective of validating and assessing their real risk considering their *Threat*, *Vulnerability* and *Impact* risk scores. Once these scores have been calculated, a prioritisation process is carried out to identify those findings that should be fixed in a timely manner. Finally, if the vulnerability is unknown and has not been reported before, the project owners might consider reporting it in a CVE, CWE, CVSS or similar system.

To further organise this phase, three main activities have been defined:

- **Technical Report Analysis**: review of the results from the previous phase, validating the findings and removing any incomplete, incorrect or false-positive results. As part of this task, the findings are classified based on their category.

- **Impact Analysis**: Once the findings have been properly validated and classified, the next step is to determine their *Threat*, *Vulnerability* and *Impact* risk scores:
    - **Threat factors**: skill required opportunity and dimension.
    - **Vulnerability factors**: ease of discovery, ease of exploitation and awareness.
    - **Impact factors**: confidentiality, integrity and availability.

  From the average result of these factors considered for the score, one of the following scores is given to the Threat, Vulnerability and Impact risks, based on the numeric results:

  0 to 3: Low **3 to 6: Medium** **6 to 9: High**

  Finally, the checklist is completed adding the global risk posed by the controls, which is calculated from the individual results (Threat, Vulnerability and Impact). Table 1 shows how to calculate the global risk taking into consideration the Impact and the Probability (Average value of both Threat and Vulnerability results).

Table 1: Global risk evaluation

	High	Medium	High	Critical
Impact	**Medium**	Low	Medium	High
	Low	Info	Low	Medium
		Low	**Medium**	**High**

Probability (Avg. Threat & Vulnerability)

The possible values are ***Critical***, ***High***, ***Medium***, ***Low*** or ***Info.*** If an issue is found, it is marked with an **X**; if no issues are found, it is marked with ✓, and if the control is not applicable it is marked with N/A.

- **Finding Prioritisation:** The prioritisation of the findings is based on their criticality, and the results are communicated as established in the initial phases of the project.

4 CODE REVIEW DETAILS

4.1. Initial Considerations

The application to review contained several particularities that needed to be identified in order to ensure the proper analysis of the code. This included characteristics such as frameworks or libraries implemented, and the different aspects of the modules in use.

The main focus of this code review was on the KeePass 1.31 application. In order to carry out the review, the application files were divided into groups, following the libraries and modules already defined by the application itself.

The distribution of software files and batches can be found in the following excel file:

Batches_and_files.xlsx

KeePass 1.31

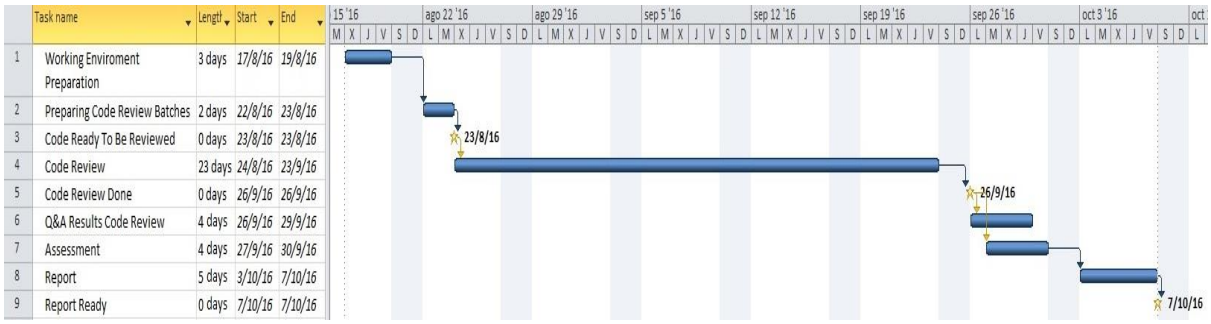
Batch	Files	Lines	Batch	Files	Lines
KeePassLibC	12	1.405	WinGUI_Util	14	2.146
KPLC_KeePassAPI (omitted)	0	0	WinGUI_Util_CmdLine	10	998
KeePassLibCpp	7	3.343	WinGUI_Util_SprEngine	4	481
KPLCpp_Crypto	20	6.279	WinGUI-B	8	2.342
KPLCpp_Crypto_SHA2 (omitted)	0	0	WinGUI-C	16	2.228
KPLCpp_DataExchange	4	1.951	WinGUI-D	10	2.552
KPLCpp_Details	4	1.913	WinGUI-E	8	2.595
KPLCpp_IO	6	532	WinGUI-F	9	1.424
KPLCpp_PasswordGenerator	8	1.174	WinGUI_Util-B	14	2.393
KPLCpp_SDK	15	1.828	WinGUI_Util-C	4	2.062
KPLCpp_SysSpec_Win	4	1.123	NewGUI-B	2	6.001
KPLCpp_Util	22	4.983	NewGUI-C	2	3.647

Batch	Files	Lines	Batch	Files	Lines
WinGUI	2	12.659	NewGUI-D	2	2.779
NewGUI	2	2.816	NewGUI-E	20	2.271
NewGUI_TaskbarListEx	3	179	NewGUI-F	10	1.797
NewGUI_TaskDialog	2	342	NewGUI-G	8	2.488
NewGUI_VistaMenu	2	135	NewGUI-H	14	3.033
WinGUI_Plugins	16	2.723			

4.2. Planning

The code review was performed following the planning stipulated at the beginning of the project and taking into consideration the tests selected and the size/complexity of the application to review. The final planning is detailed in Figure 6, including the dates and time required for each step.

Figure 6: Code review planning



4.3. Overview of Results

The controls used in the code review were generated within the EU-FOSSA project, derived from the proposals from two well-known software security authorities: the Application Security Verification Standard from OWASP, and the secure coding standard of C and C++ from 'the Carnegie Mellon Software Engineering Institute (SEI)'.

Not all the controls available from the set defined by EU-FOSSA are applicable to this code review. This is due to the fact that the EU-FOSSA control set contemplates a wide array of features and characteristics, such as a given functionality not present on the code review conducted. For example, the controls related to the Authentication category were not checked as this functionality is not present in KeePass.

Each control has a unique identifier, following the template below:

[CAT]-[SUB]-[###]

Legend:

[CAT] → Control category.
[SUB] → Control subcategory.
[###] → Control number.

4.3.1. General Findings

Table 2 shows a summary of all the general controls reviewed, and it gathers the set of controls of the methodology used in the code review project. Those controls that are out of the scope of this specific analysis have also been included, but marked as N/A. The findings are associated with the controls affected, and not with the number of detections therein. This means that if two detections were found in the same control, only one will appear in this table (as it is a reference). More details can be found in Section 4.4, which contains evidence of each finding identified as a potential issue (evidence row). However, these findings should be checked in the entire code, as there could more detections.

Table 2: Checklist general controls

ID	Control	Result	
DIM-FIM-001	Deletion of temporary files	✓	
DIM-FIM-002	File permissions at creation	✓	
DIM-FIM-003	Ensure that all files are closed after use	✓	
DIM-FIM-004	Usage of canonical path of files	✓	
DIM-FIM-005	Always check EOF on streams I/O operations	✓	
DIM-FIM-006	Updated file management	✓	
DIM-DSM-001	All data streams have to be closed after use	✓	
DIM-DSM-002	Get all valid data contained in a data stream	✓	
DIM-CEM-001	Correct format exchange of binary to string data	✓	
DIM-CEM-002	Normalise all string inputs	NA	
DIM-IVS-001	Data input validation	✓	
DIM-IVS-002	Data output validation	NA	
DIM-XSV-001	Review the XML schema, or DTD, used and its structure	NA	
DIM-XSV-002	Data is sanitised before constructing and sending it in XML format	NA	
AUT-AUV-001	The application uses a robust authentication verification process	NA	
AUT-PPU-001	The application makes use of a complex password policy	NA	
AUT-PPU-002	Password history is maintained	NA	
AUT-PPU-003	Passwords must expire after a set amount of time	NA	
AUT-CST-001	Protection of the password at rest	✓	
AUT-UAP-001	Number of login attempts is limited	NA	

ID	Control	Result	
AUT-UAP-002	Connections from uncommon locations are restricted	✓	
AUT-PRP-001	A password recovery process is defined	NA	
AUT-PRP-002	Password recovery process requires additional validation steps	✓	
AUT-PRP-003	User is warned of any password recovery attempts	NA	
SMG-SCP-001	Review controls in place to assign user privileges	✓	
SMG-SCP-002	Server keeps a list of all active identifiers and their data	NA	
SMG-SCP-003	Session cookies are protected and do not have sensitive data	NA	
SMG-SID-001	A unique ID is assigned to each individual user session	NA	
SMG-SID-002	Control active sessions at any time	NA	
SMG-SLC-001	Session timeouts are implemented	NA	
SMG-SLC-002	Privilege management	NA	
SMG-LGP-001	ID, assignations, privileges and resources are discarded on logout	✓	
SMG-LGP-002	Logout functionality should terminate the session and connection	✓	
ATZ-ACS-001	Use only trusted system objects for access authorisation decisions	NA	
ATZ-ACS-002	Authorisation rules and process	✓	
ATZ-PRV-001	Privileges and roles	NA	
ATZ-PRV-002	Privilege modification	✓	
CPT-CPR-001	Sensitive information is stored securely using encryption	✓	
CPT-CPR-002	Information stored is hashed to preserve its integrity	✓	
CPT-CRC-001	Review cryptographic configuration parameters	✓	
CPT-CRC-002	Management cryptographic keys	✓	
EHI-INL-001	Metadata leakage on any files accessible by the users	NA	
EHI-INL-002	Comments accessible in any client-side code files	NA	
EHI-INL-003	Internal routes and paths must not be shown as default routes	✓	
EHI-SFL-001	Sample files must be removed or filtered by the server	NA	
EHI-EHD-001	Application errors must be controlled in the GUI	✓	
EHI-EHD-002	Try-catch-finally block	X	Info
EHI-EHD-003	Correct Exception and Error Management	✓	
EHI-EHD-004	Object is restored to a previous state after an error or failure	NA	
EHI-EHD-005	Third-party services and libraries errors are controlled locally	✓	
COM-HSM-001	Avoid HTTP Response Splitting	✓	
COM-HSM-002	Prevent Directory Traversal	NA	
COM-HSM-003	HTTP Strict Transport Security	NA	
COM-HSM-004	Avoidance of redirects and forwards in webpages	NA	
LOG-CFG-001	Logs are properly configured	✓	

ID	Control	Result	
LOG-CFG-002	Logs register only the information needed for their purpose	NA	
LOG-CFG-003	Debug Logging	✓	
LOG-CFG-004	Logging exceptions	X	Info
LOG-GEN-001	Log generation must continue after a log system exception	NA	
LOG-LSI-001	Logs must not contain sensitive information, or else use hashes	NA	
LOG-LSI-002	User passwords and tokens must be omitted from logs	✓	
SCD-FWK-001	All frameworks and third party components are up-to-date	X	Low
SCD-VTY-001	Review operation on numeric values and bit collections	✓	
SCD-VTY-002	On division operations, check that the divisor does not equal zero	X	Low
SCD-VTY-003	Direct comparisons with NaN must not be carried out	✓	
SCD-VTY-004	Do not use floating-point variables as loop counters	✓	
SCD-EXM-001	Function return values are parsed and evaluated	✓	
SCD-EXM-002	Method arguments must fall within the established bounds	✓	
OPT-CCR-001	Ensure that instance locks are controlled	NA	
OPT-CCR-002	Do not use unsafe operations, expressions or methods in Threads	✓	
OPT-CCR-003	Thread pools must be controlled	✓	
OPT-DOS-001	Check DoS vulnerabilities on the application	NA	
OPT-MRM-001	Review the memory management process	✓	
OPT-MRM-002	Review resource management process	✓	
OPT-COS-001	Evaluate processes that call back to the code multiple times	NA	
OPT-COS-002	There is a clear separation between the application layers	NA	
OPT-RPM-001	Analyse role-privilege matrix used on the application	NA	

4.3.2. Language-Specific Findings

Table 3 contains a summary of all the language-specific controls reviewed; in this case only **C and C++ languages controls** apply (the controls for JAVA and PHP are not included). Those controls that are out of the scope for this specific analysis (because they do not apply) have also been included, but marked as N/A.

The findings are associated with the controls affected, and not with the number of detections therein. This means that if two detections were found on the same control, only one will appear in this table (as it is a reference). The details of the findings can be found in Section 4.4.

Table 3: Check-list language-specific controls

ID	Control	Result	
CBC-PRE-001	Do not create a universal character name through concatenation	✓	
CBC-PRE-002	Avoid side effects in arguments to unsafe macros	NA	
CBC-PRE-003	Do not use pre-processor directives in invocations of function like macros	✓	
CBC-VMG-001	Declare objects with appropriate storage durations	✓	
CBC-VMG-002	Declare identifiers before using them	✓	
CBC-VMG-003	Do not declare and identifier with conflicting linkage	✓	
CBC-VMG-004	Do not declare or define a reserved identifier	✓	
CBC-VMG-005	Use the correct syntax when declaring a flexible array member	✓	
CBC-VMG-006	Do not create incompatible declarations of the same function or object	✓	
CBC-VMG-007	Do not declare variables inside a switch statement before the first case label	✓	
CBC-VMG-008	Ensure that floating-point conversions are within range of new type	X	Medium
CBC-VMG-009	Preserve precision when converting integral values to floating-point type	NA	
CBC-VMG-010	Do not use object representations to compare floating-point values	✓	
CBC-VMG-011	Do not form or use out-of-bounds pointers or array subscripts	✓	
CBC-VMG-012	Ensure size arguments for variable length arrays are in a valid range	✓	
CBC-VMG-013	Do not subtract or compare two pointers that do not refer to the same array	NA	
CBC-VMG-014	Do not add or subtract an integer to a pointer to a non-array object	NA	
CBC-VMG-015	Guarantee that library functions do not form invalid pointers	NA	
CBC-VMG-016	Do not add or subtract a scaled integer to a pointer	✓	
CBC-VMG-017	Do not attempt to modify string literals	✓	
CBC-VMG-018	Guarantee that string storage has sufficient space for character data and the null terminator	NA	
CBC-VMG-019	Do not pass a non-null-terminated character sequence to a library function that expects a string	✓	
CBC-VMG-020	Cast characters to unsigned char before converting to larger	✓	
CBC-VMG-021	Do not confuse narrow and wide character strings and functions	✓	
CBC-VMG-022	Do not read uninitialised memory	✓	
CBC-VMG-023	Do not dereference null pointers	X	Low
CBC-VMG-024	Do not dereference null pointers	✓	

ID	Control	Result	
CBC-VMG-025	Variables must not be accessed using an incompatible type pointer	✓	
CBC-VMG-026	Prevent undefined behaviour when restrict-qualified pointers are used	NA	
CBC-VMG-027	Do not apply operands within the sizeof, _Alignof or _Generic functions	✓	
CBC-VMG-028	Ensure that unsigned and signed integer operations are managed correctly	✓	
CBC-MEM-001	Do not access freed memory	NA	
CBC-MEM-002	Free dynamically allocated memory when no longer needed	NA	
CBC-MEM-003	Allocate and copy structures containing a flexible array member dynamically	✓	
CBC-MEM-004	Only memory allocated dynamically should be freed	✓	
CBC-MEM-005	Allocate sufficient memory for an object	X	Medium
CBC-MEM-006	Do not modify the alignment of objects by calling realloc()	NA	
CBC-FIO-001	Exclude user input from format strings	NA	
CBC-FIO-002	Do not perform operations on devices that are only appropriate for files	NA	
CBC-FIO-003	Do not assume that fgets() or fgetws() returns a nonempty string when successful	NA	
CBC-FIO-004	Do not copy a FILE object	✓	
CBC-FIO-005	Do not alternately input and output from a stream without an intervening flush or positioning call	✓	
CBC-FIO-006	Reset strings or fgets() or fgetws() failure	NA	
CBC-FIO-007	Do not call getc(), putc(), getwc(), or putwc() with a stream argument that has side effects	NA	
CBC-FIO-008	Only use values for fsetpos() that are returned from fgetpos()	NA	
CBC-FIO-009	Avoid TOCTOU race conditions while accessing files	NA	
CBC-FIO-010	Do not access a closed file	✓	
CBC-ENV-001	Do not modify the object referenced by the return value of certain functions	✓	
CBC-ENV-002	Do not rely on an environment pointer following an operation that may invalidate it	✓	
CBC-ENV-003	All exit handlers must return normally	✓	
CBC-ENV-004	Do not call system()	X	Medium
CBC-ENV-005	Do not store pointers returned by certain functions	✓	
CBC-ENV-006	Ensure proper usage of the readlink() function	✓	
CBC-ENV-007	Do not call putenv() with a pointer to an automatic variable as the argument	✓	
CBC-ENV-008	Proper privilege revocation and relinquish process must be defined	NA	

ID	Control	Result	
CBC-SEH-001	Call only asynchronous-safe functions within signal handlers	NA	
CBC-SEH-002	Do not access shared objects in signal handlers	✓	
CBC-SEH-003	Do not call signal() from within interruptible signal handlers	NA	
CBC-SEH-004	Do not return from a computational exception signal handler	✓	
CBC-SEH-005	Set errno to zero before calling a library function known to set errno, and check errno only after the functions returns a value indicating failure	NA	
CBC-SEH-006	Do not rely on indeterminate values of errno	NA	
CBC-SEH-007	Detect and handle standard library errors	✓	
CBC-SEH-008	Detect errors when converting a string to a number	NA	
CBC-CON-001	Clean up thread-specific storage	✓	
CBC-CON-002	Do not destroy a mutex while it is locked	✓	
CBC-CON-003	Prevent data races when accessing bit-fields from multiple threads	✓	
CBC-CON-004	Avoid race conditions when using library functions and files	NA	
CBC-CON-005	Declare objects shared between threads with appropriate storage durations	NA	
CBC-CON-006	Avoid deadlock by locking in a predefined order	NA	
CBC-CON-007	Wrap functions that can spuriously wake up in a loop	NA	
CBC-CON-008	Do not call signal() in a multithreaded program	NA	
CBC-CON-009	Do not join or detach a thread that was previously joined or detached	NA	
CBC-CON-010	Do not refer to an atomic variable twice in an expressions	NA	
CBC-CON-011	Wrap functions that can fail within a loop	NA	
CBC-CON-012	Do not use the vfork() function	NA	
CBC-CON-013	Do not use signals to terminate threads	NA	
CBC-MSC-001	Do not use the rand() function for generating pseudorandom numbers	X	Medium
CBC-MSC-002	Properly seed pseudorandom number generators	NA	
CBC-MSC-003	Do not pass invalid data to asctime() function	✓	
CBC-MSC-004	Ensure that control never reaches the end of a non-void function	✓	
CBC-MSC-005	Do not treat a predefined identifier as an object if it might only be implemented as a macro	✓	
CBC-MSC-006	Do not call va_arg() on a va_list that has an indeterminate value	NA	
CBC-MSC-007	Do not violate constraints	NA	
CPP-VMG-001	Do not define a C-style variadic function	NA	
CPP-VMG-002	Overload allocation and deallocation functions as a pair in the same scope	NA	

ID	Control	Result	
CPP-VMG-003	Do not recursively re-enter a function during the initialization of one of its static objects	NA	
CPP-VMG-004	Destructors and deallocation functions must be declared noexcept	✓	
CPP-VMG-005	Do not define an unnamed namespace in a header file	NA	
CPP-VMG-006	Do not cast to an out-of-range enumeration value	✓	
CPP-VMG-007	Guarantee that container indices and iterators are within the valid range	X	Info
CPP-VMG-008	Guarantee that library functions do not form invalid iterators	X	Info
CPP-VMG-009	Use valid iterator ranges	✓	
CPP-VMG-010	Do not subtract iterators that do not refer to the same container	NA	
CPP-VMG-011	Do not use an additive operator on an iterator if the result would overflow	NA	
CPP-VMG-012	Do not use pointer arithmetic on polymorphic objects	NA	
CPP-VMG-013	Guarantee that storage for strings has sufficient space for character data and the null terminator	✓	
CPP-VMG-014	Do not pass a null pointer to char_traits::length	✓	
CPP-VMG-015	Use valid references, pointers, and iterators to reference elements of a basic_string	✓	
CPP-VMG-016	Range check element access	✓	
CPP-VMG-017	Do not delete an array through a pointer of the incorrect type	NA	
CPP-VMG-018	Do not rely on side effects in unevaluated operands	✓	
CPP-VMG-019	Do not access a cv-qualified object through a cv-unqualified type	✓	
CPP-VMG-020	Do not cast pointers into more strictly aligned pointer types	NA	
CPP-VMG-021	Do not cast or delete pointers to incomplete classes	NA	
CPP-VMG-022	Use offsetof() on valid types and members	NA	
CPP-VMG-023	A lambda object must not outlive any of its reference captured objects	NA	
CPP-VMG-024	Do not access the bits of an object representation that are not part of the object's value representation	✓	
CPP-VMG-025	Do not rely on the value of a moved-from object	✓	
CPP-MEM-001	Properly deallocate dynamically allocated resources	NA	
CPP-MEM-002	Detect and handle memory allocation errors	NA	
CPP-MEM-003	Explicitly construct and destruct objects when manually managing object lifetime	NA	
CPP-MEM-004	Provide placement new with properly aligned pointers to sufficient storage capacity.	NA	
CPP-MEM-005	Honor replacement dynamic storage management requirements	NA	

ID	Control	Result	
CPP-MEM-006	Do not store an already-owned pointer value in an unrelated smart pointer	NA	
CPP-MEM-007	Avoid using default operator 'new' for over-aligned types	✓	
CPP-EEH-001	Do not call std::terminate(), std::abort(), or std::_Exit()	✓	
CPP-EEH-002	Do not use setjmp() or longjmp()	NA	
CPP-EEH-003	Do not reference base classes or class data members in a constructor or destructor function-try-block handler	✓	
CPP-EEH-004	Catch handlers should order their parameter types from most derived to least derived	✓	
CPP-EEH-005	Honor exception specifications	NA	
CPP-EEH-006	Guarantee exception safety	✓	
CPP-EEH-007	Do not leak resources when handling exceptions	NA	
CPP-EEH-008	Constructors of objects with static or thread storage duration must not throw exceptions	NA	
CPP-EEH-009	Exception objects must be nothrow copy constructible	NA	
CPP-EEH-010	Catch exceptions by lvalue reference	NA	
CPP-OOP-001	Do not invoke virtual functions from constructors or destructors	X	Info
CPP-OOP-002	Do not slice derived objects	NA	
CPP-OOP-003	Do not delete a polymorphic object without a virtual destructor	✓	
CPP-OOP-004	Write constructor member initialisers in the canonical order	NA	
CPP-OOP-005	Do not use pointer-to-member operators to access non-existent members	NA	
CPP-OOP-006	Honor replacement handler requirements	NA	
CPP-OOP-007	Prefer special member functions and overloaded operators to C Standard Library functions	X	Info
CPP-CON-001	Ensure actively held locks are released on exceptional conditions	NA	
CPP-CON-002	Do not speculatively lock a non-recursive mutex that is already owned by the calling thread	NA	
CPP-MSC-001	Do not use std::rand() for generating pseudorandom numbers	X	Medium
CPP-MSC-002	Ensure your random number generator is properly seeded	NA	
CPP-MSC-003	Obey the one-definition rule	✓	
CPP-MSC-004	Do not modify the standard namespaces	NA	
CPP-MSC-005	Value-returning functions must return a value from all exit paths	✓	
CPP-MSC-006	Do not return from a function declared [[noreturn]]	✓	
CPP-MSC-007	A signal handler must be a plain old function	NA	

4.4. Detailed Results

This chapter defines in detail each of the tests carried out, including the checks performed, the results obtained and any relevant evidence. Each control includes three scores: *Threat*, *Vulnerability* and *Impact*.

There were a total of 14 controls with findings. These controls belong to the following categories and sub-categories:

- **Error Handling / Information Leakage**
  - *Error Handling (**1 info**)*
- **Logging / Auditing**
  - *Log Configuration Management (**1 info**)*
- **Secure Code Design**
  - *Framework Requirements (**1 low**)*
  - *Variable types / operations (**1 low**)*
- **Specific C Controls**
  - *Variable Management* (**1 medium**, **1 low**)
  - *Memory Management* (**1 medium**)
  - *Environment* (**1 medium**)
  - *Miscellaneous* (**1 medium**)
- **Specific C++ Controls**
  - *Variable Management* (**2 info**)
  - *Object-Oriented Programming* (**2 info**)
  - *Miscellaneous* (**1 medium**)

The following sections describe in detail the findings, checks, results, evidences and recommendations.

4.4.1. Logging / Auditing

4.4.1.1. Log Configuration Management

Table 4: LOG-CFG-004 findings

LOG-CFG-004	Logging exceptions		**Info**
Description	Exceptions must be logged in a proper manner in case they are not going to be thrown.	**Threat**	**Low**
		Vulnerability	**Low**
		Impact	**Low**
Checks	**1**	Exceptions are logged after being handled.	**X**
Results	There is no logging functionality implemented on the catch(…) block; therefore any exception captured is not logged, nor is any trace left of this event recorded		

Evidence	**%root%\KeePassLibCpp\Details\PWFindImpl.cpp** (Lines 51-60)
	
	try
	{
	if(bCaseSensitive == FALSE)
	spRegex.reset(new boost::basic_regex<TCHAR>((LPCTSTR)strFind,
	boost::regex_constants::icase));
	else
	spRegex.reset(new boost::basic_regex<TCHAR>((LPCTSTR)strFind));
	}
	catch(...) { return DWORD_MAX; }		

Recommendation / Specific Solution	**Recommendation:** Log any exception captured that will not be thrown to have a record of the event.		

4.4.2. Secure Code Design

4.4.2.1. Framework Requirements

Table 5: SCD-FWK-001 findings

SCD-FWK-001	All frameworks and third party components are up-to-date		**Low**
Description	All frameworks and components used are kept up-to-date including all existing patches and security hotfixes. Latest version is not needed but must be patched at least.	**Threat**	Medium
		Vulnerability	High
		Impact	Low
Checks	**1**	Framework components are kept up-to-date.	**X**
	2	Third-party components are kept up-to-date.	**N/A**
Results	**RegCreateKey**: This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the **RegCreateKeyEx** function. There is various evidence of this function within the code.		
Evidence	**%root%\WinGUI\PwSafe.cpp** (Line 328) LONG l = RegCreateKey(HKEY_CLASSES_ROOT, _T(".kdb"), &hBase);		
Recommendation/ Specific Solution	**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.		

4.4.2.2. Variable types / operations

Table 6: SCD-VTY-002 findings

SCD-VTY-002	On division operations, check that the divisor does not equal zero		**Low**
Description	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**
		Vulnerability	**Low**
		Impact	**Medium**
Checks	**1**	The fields of a division are checked for invalid values.	**X**
	2	Controls to ensure that no operation is done if the divisor equals zero.	N/A
Results	The size of the 'lpstrText' variable is not tested against invalid or zero values.		
Evidence	**%root%\WinGUI\NewGUI\BCMenu.cpp** (Line 1011) size.cx += 3*(size.cx/(LONG)wcslen(lpstrText));		
Recommendation/ Specific Solution	**Recommendation:** Check the 'lpstrText' variable to ensure that no invalid or zero values are received.		

4.4.3. Specific C Controls

4.4.3.1. Variable Management

Table 7: CBC-VMG-008 findings

CBC-VMG-008	Ensure that floating-point conversions are within range of new type		**Medium**
Description	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**
Checks	**1**	A check must be defined to validate that the value fits in the smaller destination type.	✓
	2	Any errors in the type conversion must be controlled and managed.	X
Results	There are no error management controls of the return method GetUpperBound().		
Evidence	**%root%\WinGUI\NewGUI\BCMenu.cpp** (Lines 2686, 2749) int numSubMenus = (int)m_SubMenus.GetUpperBound();		
Recommendation / Specific Solution	**Recommendation:** There must be a control within the code to check the return method **GetUpperBound** in order to manage possible errors or exceptions.		

Table 8: CBC-VMG-023 findings

CBC-VMG-023	Do not read uninitialised memory		**Low**
Description	Local, automatic variables assume unexpected values if they are read before they are initialised	**Threat**	**Low**
		Vulnerability	**Low**
		Impact	**Low**
Checks	**1**	Always initialise variables before accessing their content.	X
Results	The 'szTitle' variable was not initialised before accessing its content. The 'm_value' variable was not initialised before accessing its content.		

Evidence	**%root%\WinGUI\Util\SendKeys.cpp** (Line 585)
	TCHAR szTitle[300];
	if (::GetWindowText(hwnd, szTitle, sizeof(szTitle)/sizeof(TCHAR)))
	bMatch |= (_tcsstr(szTitle, wtitle) != 0);		

Recommendation/ Specific Solution	**Recommendation:** Always initialise variables prior to accessing their content. In other case it will lead to an unexpected behaviour.		

4.4.3.2. Memory Management

Table 9: CBC-MEM-005 findings

CBC-MEM-005	Allocate sufficient memory for an object		**Medium**
Description	It is necessary to guarantee that storage for strings has sufficient space available for character data and consequently allocate sufficient memory for an object.	**Threat**	**Medium**
		Vulnerability	**Medium**
		Impact	**Medium**

Checks			
	1	The length of string storage arrays must not equal zero.	✓
	2	Validate string operations to ensure that they are controlled and cannot result in an overflow.	X
	3	Arguments passed to functions must match the expected format and size.	✓

Results	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. Related CWE: **CWE-126**.
Evidence	**%root%\WinGUI\PwSafe.cpp** (Line 496) if((_tcslen(tszBuf) > 0) && (tszBuf[0] != _T('-'))) return TRUE;
Recommendation / Specific Solution	**Recommendation:** 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.

4.4.4. Specific C++ Controls

4.4.4.1. Object-Oriented Programming

Table 10: CPP-OOP-001 findings

CPP-OOP-001	Do not invoke virtual functions from constructors or destructors		**Info**
Description	Do not directly or indirectly invoke a virtual function from a constructor or destructor that attempts to call into the object under construction or destruction.	**Threat**	**Low**
		Vulnerability	**Low**
		Impact	**Low**
Checks	**1**	Virtual functions are not called within destructors or constructors in inherited classes. Overrides are either invoked or make use of the qualified ID.	**X**
Results	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: the 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.		
Evidences	**%root%\WinGUI\Util\ShutdownBlocker.cpp** (Line 60) CShutdownBlocker::~CShutdownBlocker()		
Recommendation / Specific Solution	**Specific Solution:** Call a nonvirtual, private member function from constructors, or destructors instead of calling a virtual function		

4.4.5. Findings controlled programmatically

After the feedback from the community the following issues are controlled within the code. Despite this situation, these controls are included under this section to keep them in mind for future development.

During the code review, several findings were identified. After a detailed review and following information exchange with KeePass point of contact, it was determined that these findings are controlled within the code. For this reason, the findings were moved to a separate section, as the risk of using this code is mitigated.

Before deciding to change them, 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.

4.4.5.1. Error Handling

Table 11: EHI-EHD-002 findings

EHI-EHD-002	Try-catch-finally block			**Info**
Description	For those programming languages that have the 'try-catch-finally' structure, each of its section has to be used correctly.		**Threat**	**Low**
			Vulnerability	**Low**
			Impact	**Low**
Checks	**1**	The 'finally' statement should always present, used to release system resources, and for other clean actions.		**x**
	2	Those operations that can throw exceptions have to be conducted at the beginning of the 'try' section.		✓
	3	Generic exceptions will never be used or cached. In case of multiple exceptions, different 'catch' sections will be added.		✓
Results	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, this need to be captured within a new try-catch-finally block. This issue is controlled within the KeePass code. However, due to the severity of the control is still mentioned in here.			
Evidence	**%root%\WinGUI\Util\SessionNotify.cpp** (line 65) try { m_lpWTSUnRegisterSessionNotification(m_hTarget); } catch(...) { ASSERT(FALSE); } // RPC cancelled, exception 0x71A			
Recommendation / Specific Solution	**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, this need to be captured within a new try-catch-finally block.			

4.4.5.2. Specific C controls: Environment

Table 12: CBC-ENV-004 findings

CBC-ENV-004	Do not call system()		**Medium**
Description	The use of 'system()' functions can result in exploitable vulnerabilities, allowing the execution of arbitrary system commands.	**Threat**	**Medium**
		Vulnerability	**Medium**
		Impact	**Medium**
Checks	**1**	Avoid the use of 'system()' functions when passing an unsanitised or improperly sanitised command string originating from a tainted source.	X
	2	Avoid the use of 'system()' functions if a command is specified without a path name.	X
	3	Avoid the use of 'system()' functions if a relative path to an executable is specified and the control over the current working directory is accessible.	✓
	4	Avoid the use of 'system()' functions to specify executable programs.	N/A
	5	Check that a command processor is not invoked by 'system()' functions.	N/A
Results	**shellExecute**: This causes a new program to execute and it is difficult to use it safely. If the path it is not provided, using '**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. Related CWE: **CWE-78**. This issue is controlled within the KeePass code. However, due to the severity of the control is still mentioned here.		
Evidence	**%root%\WinGUI\UpdateInfoDlg.cpp** (Line 144) ShellExecute(NULL, NULL, PWM_HOMEPAGE, NULL, NULL, SW_SHOW); **%root%\WinGUI\PwSafeDlg.cpp** (Lines 627, 6418) ShellExecute(NULL, NULL, PWM_HOMEPAGE, NULL, NULL, SW_SHOW); (Line 635) ShellExecute(NULL, NULL, PWM_URL_DONATE, NULL, NULL, SW_SHOW); (Line 8710) ShellExecute(m_hWnd, NULL, tszFile.c_str(), NULL, NULL, SW_SHOW); **%root%\WinGUI\NewGUI\XHyperLink.cpp** (line 596) HINSTANCE result = ShellExecute(NULL, verb, url, NULL,NULL, showcmd);		
Recommendation / Specific Solution	**Recommendation:** Where more control 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**.		

4.4.5.3. Specific C Controls: Miscellaneous

Table 13: CBC-MSC-001 findings

CBC-MSC-001	Do not use the 'rand()' function to generate pseudorandom numbers		**Medium**
Description	The 'rand()' function should not be used to generate random numbers, as they are predictable due to the short cycle of numbers that uses.	**Threat**	**Low**
		Vulnerability	**Medium**
		Impact	**Medium**
Checks	**1** The 'rand()' function is not used.		**X**
Results	**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()'. It could be suitable for situations where weak random numbers are sufficient. Related CWE: **CWE-327**. This issue is controlled within the KeePass code. However, due to the severity of the control is still mentioned in here.		

Evidence	**%root%\KeePassLibCpp\SysSpec_Windows\NewRandom.cpp** (Line 74,76,78)
	
	ww = (WORD)(rand());
	
	%root%\WinGUI\Util\WinUtil.cpp (Line 954)
	
	const DWORD dwTest = dwOffset + rand();		

Recommendation / Specific Solution	**Recommendation:** The **rand()** function does not provide enough entropy. The usage of other functions such as '**random()**' is recommended.		

4.4.5.4. Specific C++ Controls: Variable Management

Table 14: CPP-VMG-007 findings

CPP-VMG-007	Guarantee that container indexes/iterators are within a valid range		**Info**
Description	It is almost entirely the responsibility of the programmer to ensure that array references are within the bounds of the array when using standard template library vectors.	**Threat**	**Low**
		Vulnerability	**Low**
		Impact	**Low**
Checks	**1**	There are controls in place to ensure that the values used in indexes or iterator are within the valid range.	**X**
Results	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 can lead to an improper behaviour, even a program crash. This issue is controlled within the KeePass code. However, due to the severity of the control is still mentioned here.		

Evidence	**%root%\KeePassLibCpp\Details\PwFileImpl.cpp** (Lines 294, 299, 305)
	p = &pVirtualFile[pos];
	..
	p += 2; pos += 2;
	..
	p += 4; pos += 4;		

Recommendation / Specific Solution	**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.		

Table 15: CPP-VMG-008 findings

CPP-VMG-008	Guarantee that library functions do not form invalid iterators		**Info**
Description	Copying data into a container that is not large enough to hold the original data will result in a buffer overflow.	**Threat**	**Low**
		Vulnerability	**Low**
		Impact	**Low**
Checks	**1**	Code ensures that the destination container can hold all the elements being copied to it.	**X**
Results	**Memory operations**: Memory operations done using **memcpy**, are used several times without checking the size of source and destiny. The function does not verify if the destination container is able to hold the element to be copied via **memcpy**(…). This issue is controlled within the KeePass code. However, due to the severity of the control is still mentioned here.		

Evidences	**%root%\WinGUI\AddEntryDlg.cpp** (Line 1071)
	e.pOriginalEntry = m_pOriginalEntry;
	memcpy(e.uuid, m_pOriginalEntry->uuid, 16);		

Recommendation / Specific Solution	**Recommendation:** Set controls in place to ensure that the destination container can address the element to be copied without losing integrity in **memcopy()** operations		

4.4.5.5. Specific C++ Controls: Object-Oriented Programming

Table 16: CPP-OOP-007 findings

CPP-OOP-007	Prefer special member functions and overloaded operators to C Standard Library functions		**Info**
Description	Several C standard library functions perform byte wise operations on objects.	**Threat**	**Low**
		Vulnerability	**Low**
		Impact	**Low**

Checks			
	1	Do not use std::memset(…) to initialise an object of nontrivial class type as it may not properly initialise the value representation of the object.	X
	2	Do not use std::memcpy(…) (or related byte wise copy functions) to initialise a copy of an object of nontrivial class type, as it may not properly initialise the value representation of the copy.	✓
	3	Do not use std::memcmp(…) (or related byte wise comparison functions) to compare objects of nonstandard-layout class type, as it may not properly compare the value representations of the objects	✓

Results	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 not kept in later uses of the object. This issue is controlled within the KeePass code. However, due to the severity of the control is still mentioned here.

Evidence	**%root%\WinGUI\NewGUI\BtnST.cpp** (Line 503)
	SHELLEXECUTEINFO csSEI;
	memset(&csSEI, 0, sizeof(csSEI));
	
	%root%\WinGUI\NewGUI\CBMenu.h (Line 71)
	memset(this, 0, sizeof(MENUITEMINFO));

Recommendation / Specific Solution	**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.

4.4.5.6. Specific C++ Controls: Miscellaneous

Table 17: CPP-MSC-001 findings

CPP-MSC-001	Do not use std::rand() for generating pseudorandom numbers		**Medium**
Description	Using std::rand() function could lead to predictable random numbers.	**Threat**	**Low**
		Vulnerability	**Medium**
		Impact	**Medium**
Checks	**1**	Use strong PRNG algorithms instead of std::rand() function.	**X**
Results	This function is not sufficiently random for security-related functions such as key and nonce creation. Related CWE: **CWE-327**. This issue is controlled within the KeePass code. However, due to the severity of the control is still mentioned in here.		
Evidence	**%root%\WinnGUI\PwSafeDlg.cpp** (line 654) srand((unsigned int)time(NULL));		
Recommendation / Specific Solution	**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); }		

5 TECHNICAL CONCLUSIONS

It is important to highlight that KeePass 1.31 is maintained for legacy versions due to the fact that this version is widely used within the EC.

The most relevant aspect to consider, and the first that stands out, is the nature of the findings. Considering that the C++ language is based on the C language, it can be determined that most of the findings are language-specific, instead of common general.

The focus of the code review was, at first, on the core part of KeePass, specifically on the functionality related to the encryption algorithms. It is a critical section from a security point of view, followed by the Graphical User Interface (GUI). The GUI is the 'visible' part of the application that interacts with the user, and it usually undergoes security audits via pentesting and vulnerability assessments.

Another interesting aspect to highlight is the programming language used in this code. It is written in C++, and compiled using Visual Studio, which is a complex language from a security point of view. It provides a very high level of flexibility and customisation, especially when compared with other modern languages used for software development. Compilation using Visual Studio entails that the code takes advantage of the libraries and frameworks provided from this proprietary IDE, such as MFC 9 in this case.

The fact that the C and C++ languages allow direct access to memory represents a stronger effort to control errors and exceptions in the code. Furthermore, C++ provides exception management to handle memory issues.

To conclude, the code review confirmed that the code **has a good level from a security point of view,** with only a few findings, none of which were critical or high-risk in nature. It is important to highlight that these findings cannot be directly considered security flaws that can be exploited, given that 'Security' is a set of layers and, therefore, several risky findings are necessary to compromise the software.

Nevertheless, the KeePass community has provided a new version amending the controls and comments within this Technical Report.

