Assignment 03

Software Reengineering

Code Analysis Assignment

***Submitted to*: Dr. Shahela Saif**

***Submitted by*:**

Malaika Zafar 21i-1110

Amna Shehzad 21i-1209

***Date of Submission*:**

25th November 2024

***Section*:**

SE-P

Introduction

Reverse engineering is a critical practice in software engineering that involves analyzing existing software to understand its architecture, functionality, and design. This project focuses on using freely available tools like Ghidra, IDA, APKTool, or dotPeek to reverse engineer a software repository.This report presents the static analysis conducted following code extraction, providing a detailed examination of the software's structure and other problems.

Reverse-Engineering Tool

The tool we selected for reverse-engineering in this assignment is called Ghidra. This tool is known as one of the best and most reliable softwares for Re-Engineering practices. It provides a wide variety of options that can be used for analysis of the software systems. Ghidra accepts .exe, dll and other forms of binary files.

License-free Software

KeePassXC is an open-source password manager that stores and manages passwords.

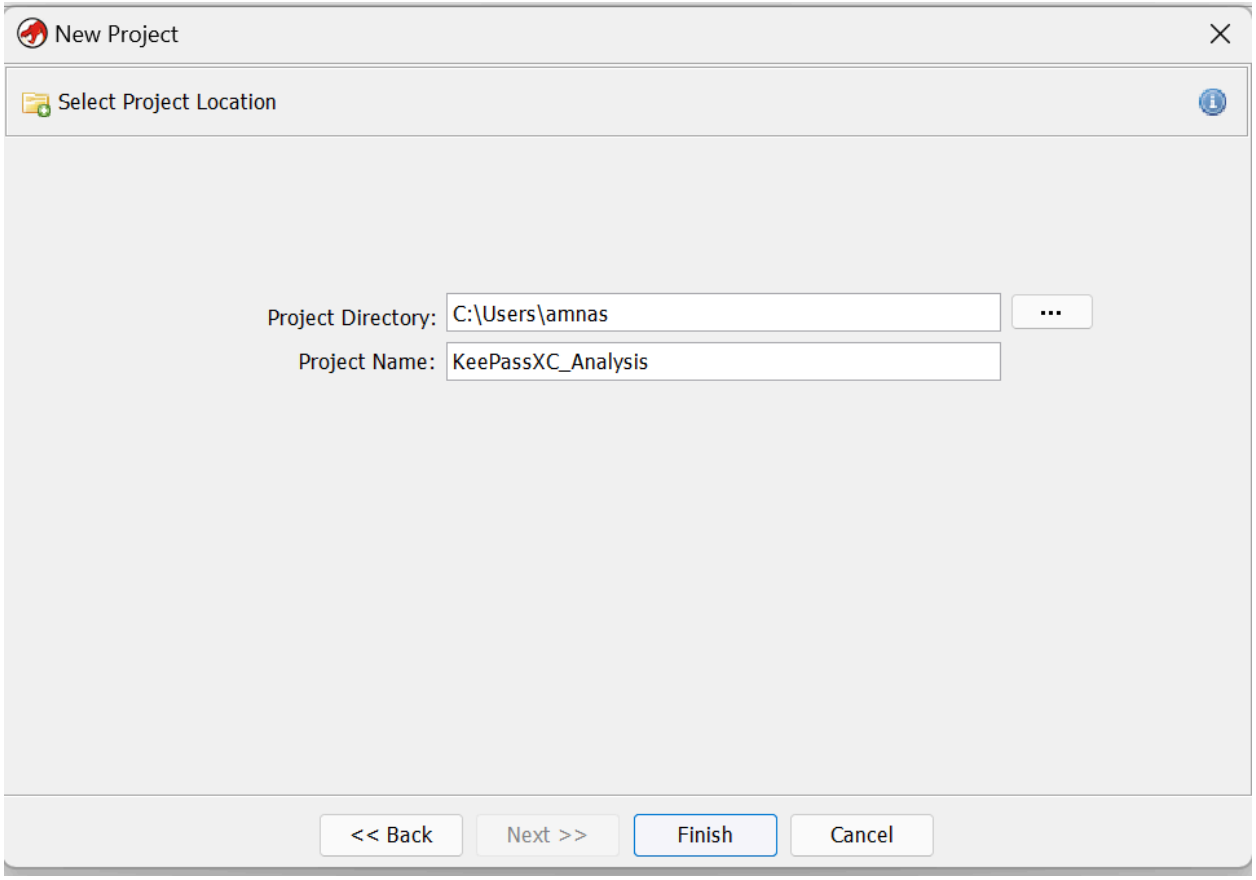
- It's a password generator utility that allows the creation of passwords with any combination of characters or easy to remember passphrases.
- It saves many different types of information, such as usernames, passwords, URLs, attachments, and notes in an offline, encrypted file that can be stored in any location, including private and public cloud solutions.
- It uses SHA-256 algorithm for encryption and storing in the databases whereas for authorization it uses hashing.
- It can be run on Windows, macOS, and Linux systems.

Project Setup

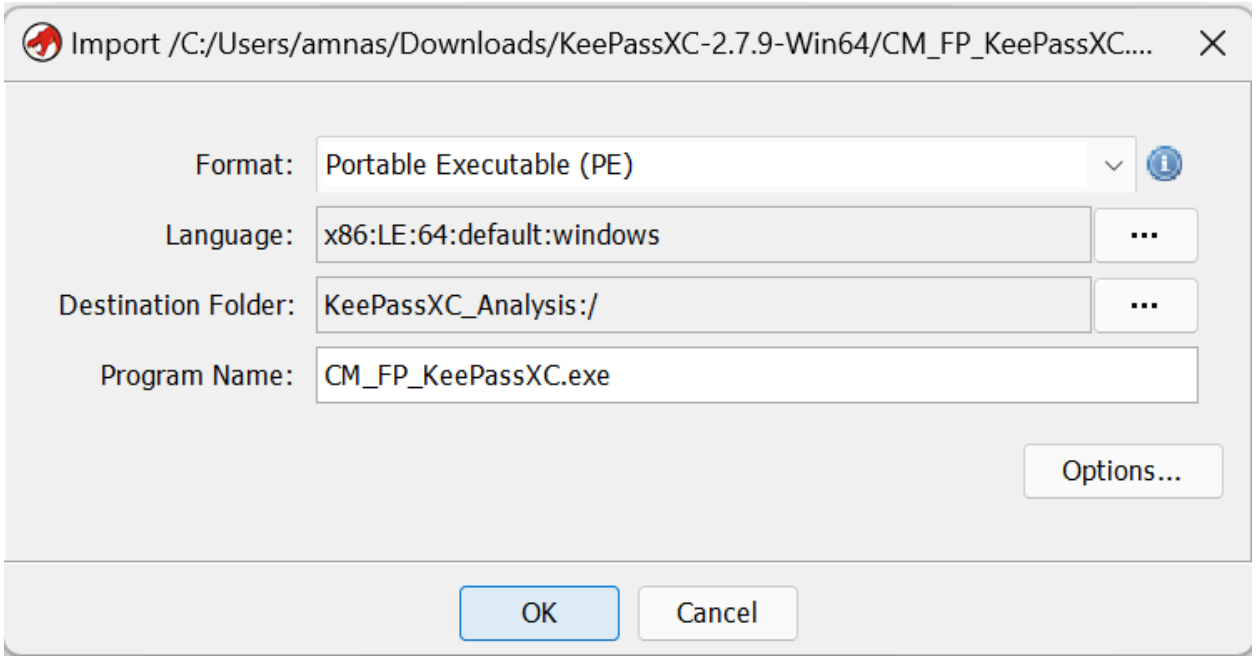
Pre-Requisites:

- Ghidra
- 7Zip
- KeePassXC installer

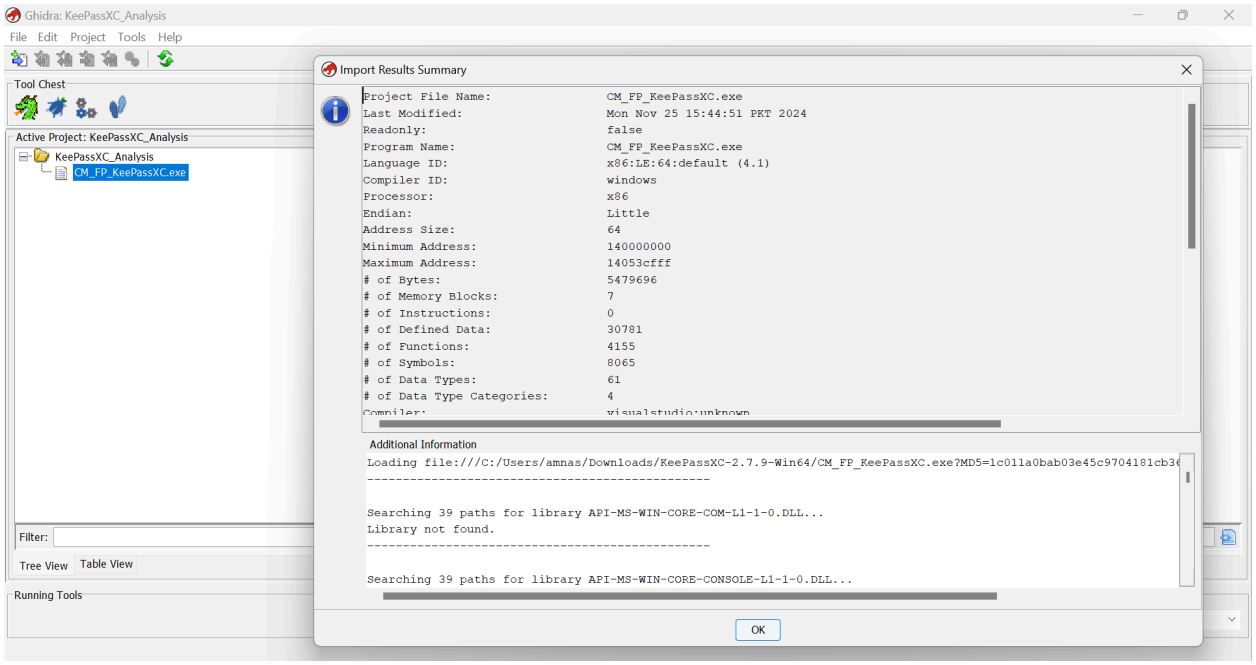
1. **Extract the KeePassXC Installer:**
    a. Use **7-Zip** or a similar extraction tool to unpack the KeePassXC installer.
    b. Locate and extract all key resources, including:
        i. The main executable file
        ii. Relevant dynamic-link libraries (DLLs) or supporting files.
    c. Organize these files into a dedicated folder for analysis.
2. **Launch Ghidra:**
   Run the **GhidraRun.bat** batch file to start the Ghidra interface.
3. **Create a New Project:**
    a. In Ghidra's project window, create a new project.
        i. Select **File > New Project** and specify a project name and location.



4. **Import the KeePassXC Executable:**
    a. In the Ghidra project window, select **File > Import** and select the extracted binary file.
    b. Follow the import prompts to ensure the binary is added to the project.
    c. Repeat the process for any additional DLLs or resources you plan to analyze.



5. **Run Initial Analysis:**
    a. Review the **Analysis Options** dialog to customize settings (e.g., enable decompiler analysis, memory block mapping, and symbol recovery).
    b. Click **Analyze** to start the process.



Analysis of Decompiled Code

1. Use of Qt Framework

KeePassXC application heavily relies on the QT Framework. This framework is for the development of cross-platform applications. KeePassXC utilizes this framework's UI utilities as well as for generating cryptographic hashes.

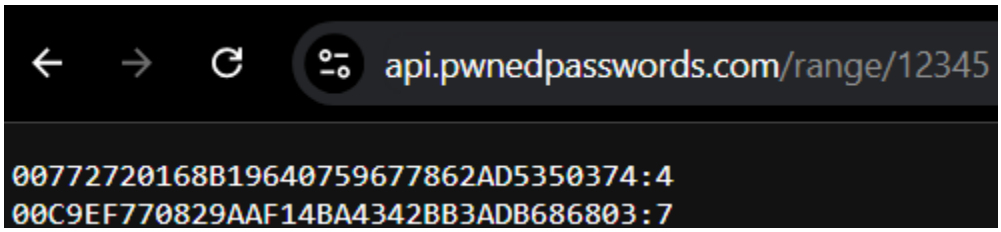
2. Use of Botan Framework

Botan is a widely used C++ library used for cryptography and TLS. KeePassXC uses Botan to generate public and private keys, identify and implement different encryption/ decryption and hashing algorithms.

3. Security Implications

Considering the fact that KeePassXC is a password manager, we decided to look into how it manages passwords and databases to ensure security. We identified multiple significant security measures taken by KeePassXC to secure user data such as passwords.

I. **"Have I been Pwned" (HIBP) api:**
This api allows us to search across multiple data breaches to see if a password has been compromised. The following url, with the first five characters of a hash sent in the {range}, returns a list of hash suffixes for passwords that have been breached in the past.
" https://api.pwnedpasswords.com/range/{range}"
Response:



KeePassXC uses this url to check whether a user's stored passwords have been compromised in the past and notifies the user to update potential weak passwords.

```
pQVar12 = (QString *)QString::left(local_c0,local_78,5);
local_c8 = QString::fromAscii_helper("https://api.pwnedpasswords.com/range/",0x25);
FUN_14000cf40(local_98,(QString *)&local_c8,pQVar12);
```

II. **Use of Asymmetric Algorithms**

RSA, ECDSA, and Ed25519 are implemented in KeePassXC to provide a robust defense through cryptographic diversity. This multi-algorithm strategy ensures that a compromise in any single algorithm won't undermine the password database's overall security.

a. RSA brings proven reliability, as it has been tried and tested for decades.
b. ECDSA delivers efficient security with compact keys
c. Ed25519 offers modern performance benefits.

Together, these algorithms create a system that's both secure and widely compatible. User databases and passwords are stored in encrypted forms.

```
Botan::RSA_PrivateKey::RSA_PrivateKey((RSA_PrivateKey *)local_1e8,local_228,&local_308,1)

Botan::ECDSA_PrivateKey::ECDSA_PrivateKey((ECDSA_PrivateKey *)local_178,local_228,&local_378,
1);

Botan::Ed25519_PrivateKey::Ed25519_PrivateKey
          ((Ed25519_PrivateKey *)local_1e8,local_228,&local_2a8,1);
```

III. **Database lock**

KeePassXC not only encrypts the databases and passwords before saving them, it also allows users to lock the database using a master key. This helps to prevent unauthorized access to the database. In addition to these precautions, it also supports the use of a YubiKey as a form of two-factor authentication (2FA) to further enhance security. With YubiKey integration, users can ensure that even if their master key is compromised, an attacker cannot access the database without the physical YubiKey device.

4. Potential Vulnerabilities

Hashing Algorithm Support

It does not support AES-256 when it come to symmetric key decryption despite it being the most advanced encryption/ decryption method.

```
QMetaObject::tr((QMetaObject *)&DAT_1404d9ea0,local_res8,
                "AES-256/GCM is currently not supported",0,CONCAT44(uVar19,0xffffffff));
```

For backward compatibility, KeePassXC supports old password databases where the master password hash was derived using the MD5 algorithm. However, this poses a security concern as MD5 is widely considered insecure due to its vulnerability to collision attacks and brute force attempts.

```
else {
  bVar6 = QString::operator==((QString *)(param_1 + 0x28),"md5");
  if (bVar6) {
    if (*(int *)(*(longlong *)(param_1 + 0x20) + 4) < 8) {
      QMetaObject::tr((QMetaObject *)&DAT_1404d9ea0,local_res8,
                      "Cipher IV is too short for MD5 kdf",0,CONCAT44(uVar19,0xffffffff));
```

Information leakages:

```
    ,
    if (hashSeed != 0) {
      uVar3 = __acrt_iob_func(2);
      FUN_18001aff0(uVar3,
                    "qSetGlobalQHashSeed: forced seed value is not 0, cannot guarantee that the ha
                    ing functions will produce a stable value."
                    ,in_R8,in_R9);
    }
    DAT_1804ba900 = hashSeed & 0x7fffffff;
  }
```

```
2  puVar1 = (undefined8 *)FUN_18001afe0();
3  __stdio_common_vfprintf(*puVar1,param_1,hashSeedMessage,0,&local_res18);
4  return;
5}
```

The conditional check if (hashSeed != 0) introduces an implicit information flow, as the execution of the subsequent block of code depends on the value of hashSeed. This dependency means that the decision to log a message (qSetGlobalQHashSeed) indirectly reveals whether hashSeed is zero or non-zero. While the actual value of hashSeed is not directly exposed, the mere presence or absence of the log message could inadvertently disclose that the seed has been manually set, potentially leading to information leakage.

Credentials stored on clipboard

KeePassXC temporarily **stores user credentials** in the clipboard to autofill forms. While convenient, this approach introduces a potential security risk because clipboards are often shared resources in operating systems, which

might be accessible by other applications. For example, the Windows operating system ensures partial isolation of the clipboard, which means that applications with appropriate permissions can access clipboard data.

Shared Access:

The clipboard is a system-wide resource in most operating systems. Any application with appropriate permissions can access its contents.

No Isolation:

Many operating systems do not isolate clipboard contents by application, making it easier for third-party applications to snoop on data.

```
void setTextToClipboard(longlong param_1)

{
  QClipboard *this;
  QString *pQVar1;
  QString local_res8 [8];

  this = QGuiApplication::clipboard();
  pQVar1 = (QString *)
           QPlainTextEdit::toPlainText
                     (*(QPlainTextEdit **)(*(longlong *)(param_1 + 0x30) + 0xd0),local_res8);
  QClipboard::setText(this,pQVar1,0);
  QString::~QString(local_res8);
  return;
}
```

Memory Dump Vulnerability

- **Destructors** in programming languages like C++ are used to release resources allocated to objects when they are no longer needed.
- While the destructor **frees up memory for reuse**, it does not ensure that the sensitive data is cleared or overwritten in the memory.
- In the case of KeePassXC, the QByteArray objects have destructors that free memory when the object goes out of scope. This means that the **confidential data could still be present** in memory until overwritten by another process.
- Tools that capture memory dumps can potentially **recover** such data, posing a security risk.

```
QByteArray::operator=((QByteArray *)(p
QByteArray::~QByteArray(local_res8);
```

5. Positive Indicators:

Constant cleanups

The decompiled code shows that the system manages memory allocations and deallocations very regularly. Constructors, Destructors are used very frequently as well as allocation and deallocation functions. This means the system is taking precautionary measures to avoid memory leakages.

For example:

The following two screenshots give an example of how KeePassXC is continuously managing memory. In the first image, we can see that objects of type QByteArray are created, and in the second image this memory is deallocated and freed up for use.

```
FUN_1401de8b0(local_70,(QByteArray *)(param_1 + 0x30),(QObject *)0x0);
QByteArray::QByteArray((QByteArray *)&local_b8);
  QByteArray::~QByteArray((QByteArray *)&local_c8);
  QByteArray::~QByteArray((QByteArray *)&local_b8);
  FUN_1401de9e0(local_70);
}
else {
  bVar6 = QString::operator==((QString *)(param_1 + 0x28),"md5")
```

Preserves data integrity:

The system regularly uses semaphores and locks to avoid race conditions. This ensures that critical sections(ie shared resources) are accessed at a time which prevents data corruption.Using a synchronization mechanism like a semaphore ensures that database connections are limited, allowing processes to safely and efficiently share access to the database.

```
uVar3 = __scrt_acquire_startup_lock();
iVar2 = (int)CONCAT71((int7)((ulonglong)unaff_RBX >> 8),(char)uVar3);
if (DAT_1404e3508 != 1) {
  if (DAT_1404e3508 == 0) {
    DAT_1404e3508 = 1;
    iVar2 = _initterm_e(&DAT_14021c958,&DAT_14021c970);
    if (iVar2 != 0) {
      return 0xff;
    }
    _initterm(&DAT_14021b788);
    DAT_1404e3508 = 2;
  }
  else {
    bVar1 = true;
  }
  __scrt_release_startup_lock((char)uVar3);
```

Comparative Analysis:

In this section we will be analyzing the decompiled code extracted from Ghidra as well as the original code using the Understand tool to understand the changes that occurred as it helps to understand the decompiling capability of the tool Ghidra.

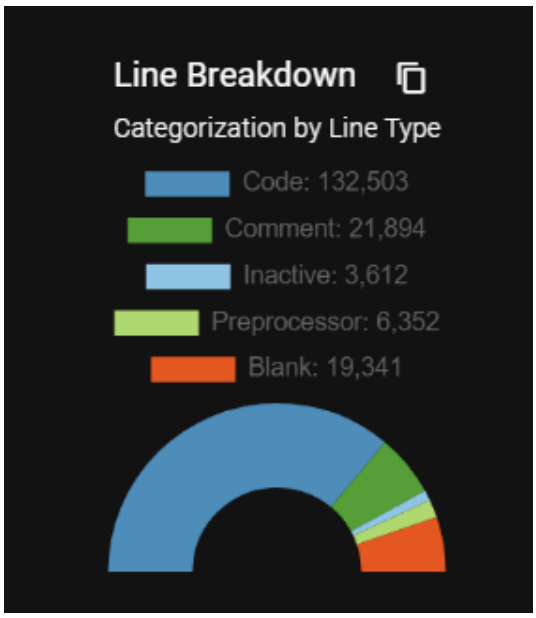
Analysis of code breakdown:

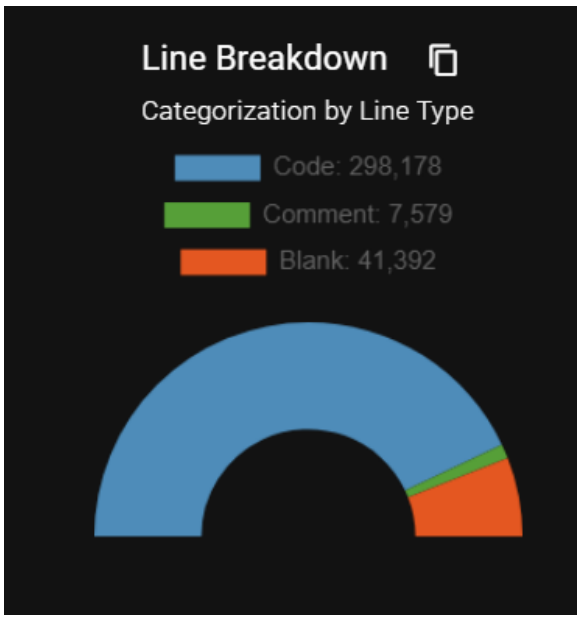
Project Metrics	
Files:	825
Program Units:	4931
Lines:	182663
Blank Lines:	19341
Code Lines:	132503
Comment Lines:	21894
Statements:	25740

Original code

Project Metrics	
Files:	1
Program Units:	7369
Lines:	346729
Blank Lines:	41392
Code Lines:	298178
Comment Lines:	7579
Statements:	139549

Decompiled code





Original code **Decompiled code**

Number of files

Source code:

In source code individual files with propers classes structure and function are added.In the decompiled file only the one file is extracted along with all of the functions and references to the classes added in it.

Code lines

Original code: 132,503

Decompiled code: 298,178

The original code has a lower number of code compared to the decompiled code which might be due to removal of preprocessor directives etc.

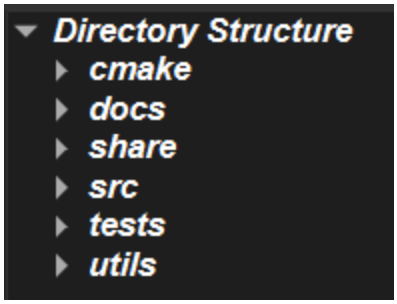
Blank lines:
Original code: 19,341
Decompiled code: 41,392
The number of blanks lines in the code is greater in decompiled code compared to original code.This might be due to the decompiled code formatting structure which can enhance readability.

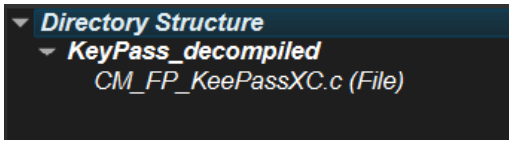
Comments:
Original code: 21,894
Decompiled code:7,579
The source code has more comments since during decompilation comments are removed.Ghidra then adds its own comments based on the code.

Inactive Code:
Original code: 3,612
The decompiled code had no inactive code.Ghidra primarily focuses on extracting executable instructions from the binary. Inactive code (such as comments, dead code, or code excluded) is removed.

Pre-processor directive:
Original code: 6,352
The decompiled code has no pre-processor directive.Preprocessor directives, which are handled at the compilation stage, are also not present in the binary output hence Ghidra does not consider them either.

Comparison of Entities





Original Code **Decompiled code**

Folder structure:

The decompiled code structure shows no likeness to the actual code structure since during it decompilation it does not consider the folder or file structure at all.It gathers all the function into a single file.Whereas in the original code you can see the multiple folders and files in it.

Naming convention:

In the original code, variables and functions are assigned meaningful and relevant names that reflect their actual functionality. However, in the decompiled code, the names lose their relevance and instead follow a standardized naming convention, typically assigned by Ghidra i.e. FUN_1401cd0.

Functional Comparison:

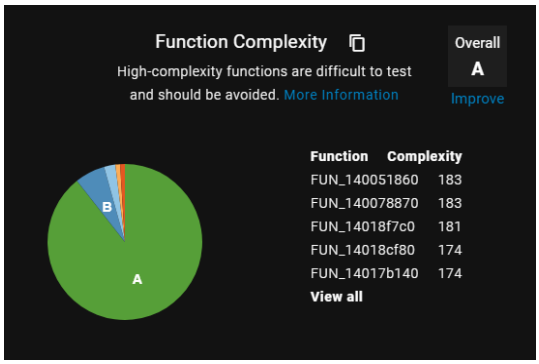
Number of functions:

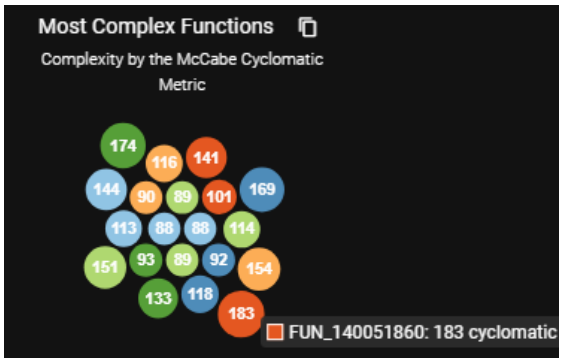
The number of functions (7,403) in the decompiled code is greater than the number of functions in the original code (4,793) because:

1. Function Splitting: During decompilation, Ghidra may split what were originally more complex, high-level functions in the source code into smaller, lower-level functions. This increases the total count of functions in the decompiled code.
2. Low-Level Code Representation: Decompiled code often includes more granular, low-level functions that are not present in the original code. For example, small helper functions or compiler-generated routines might be exposed in the decompiled output but are abstracted away in the original code.
3. Inlining and Optimizations: The decompiler might reveal multiple functions from the inlining and optimization processes done by the compiler. These optimizations could cause the decompiled code to show many more functions than in the original.

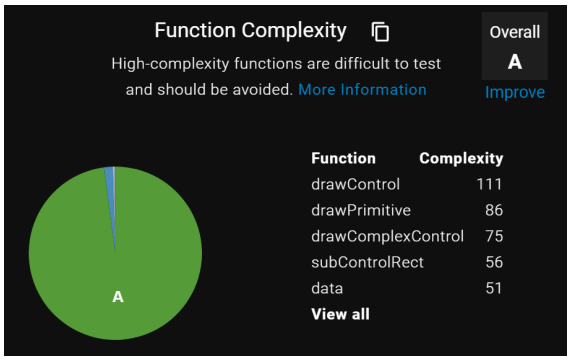
4. Code Reconstruction: Some code that was originally part of a single function may be reconstructed into multiple smaller functions by the decompiler, especially if the original code had a more compact or abstract structure.

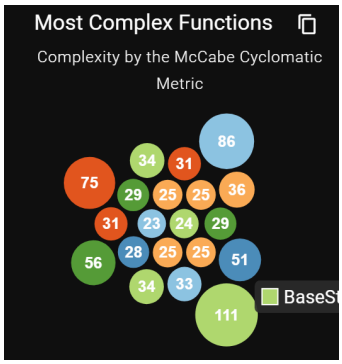
Functional complexity:





Decompiled code





Original code

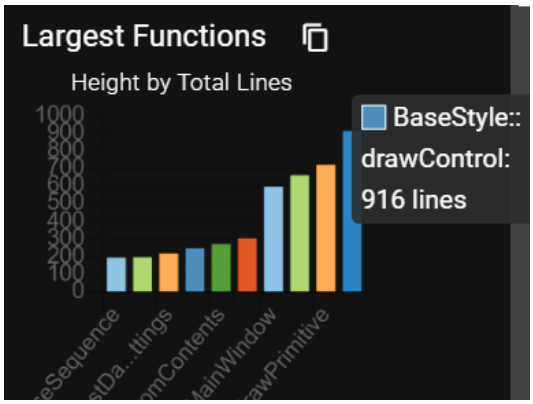
In the original code the most complex function is drawControl with a functional complexity of 111 whereas in the decompiled code the most complex function is FUN_140051860 with complexity of 183.It is quite noticeable that the functional complexity of decompiled code due to certain factors such as:

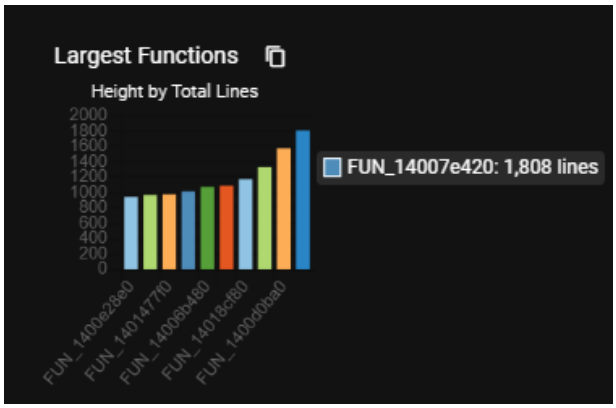
Loss of Abstraction: Decompiled code loses high-level abstractions and optimizations, making functions appear more complex than in the original source code.

Increased Granularity: Ghidra decompilation often breaks down complex operations into smaller steps, resulting in larger and more intricate functions.

Function Mapping Challenges: The decompiled function may combine code from multiple functions or parts of the program, leading to higher complexity in the decompiled output.

Largest function:





Original code **Decompiled code**

The number of lines for the largest function also changed after the decompilation.

Comparison of architecture and design:

Original code Architectural pattern:

- ➔ **Layered Architecture:**
  - ◆ While the original code documentation does not explicitly specify an architectural pattern, an analysis of the folder structure reveals a **layered architecture**. The design separates functionalities into distinct folders, each addressing specific responsibilities for example:
    - src/gui & src/cli:Handles the user interface with GUI and CLI .
    - src/core & src/crypto: Manages core password management, encryption, and business logic
    - src/format & src/streams: Focuses on database encryption and storage.
- ➔ **Modularity and Separation of Concerns:**
  - ◆ The folder structure exhibits clear modular design, with each directory handling specific functionality:
    - Password and database management in src/core and src/crypto.
    - Browser integration in src/browser.
    - Unit tests and integration tests in the tests directory.

➔ **Cross-Platform Support:**
- ◆ KeePassXC emphasizes cross-platform compatibility through cross-platform build tools (cmake) and configuration scripts (vcpkg).
- ◆ The modular separation of platform-specific utilities (e.g., keepassxc-kdewallet and keepassxc-keychain) ensures adaptability across various systems while maintaining shared logic for portability.

➔ **Security-Centric Design:**
- ◆ Security is a core focus, with reliance on cryptographic libraries like Botan for encryption and database security along with other algorithms with the help of different frameworks.
- ◆ The design incorporates robust cryptographic testing (tests/crypto) and ensures database integrity validation (tests/database), highlighting the commitment to security and reliability.

Decompiled code Architectural pattern:

Flat Structure:

- The decompiled code generated by Ghidra often lacks the original hierarchical or modular structure of the source code. All functions, regardless of their original organization, are extracted and presented in a single, flat file.
- Classes and objects are typically not represented, making the decompiled code appear function-centric and devoid of a logical architectural pattern.

Focus on Functions:

- Ghidra emphasizes extracting and listing individual functions rather than reconstructing the relationships or modular separation of the source code.
- Functions are grouped under a general function category, and connections to specific classes or files in the original source code may be missing or unclear.

Incomplete Representation of Object-Oriented Features:

- Polymorphism and virtual functions (via vtables) are partially identified, but the tool does not reconstruct full class structures.

- Classes without polymorphism or significant metadata are often misinterpreted as unrelated groups of functions.

Lack of Original File Organization:

- Ghidra does not preserve the original file structure of the source code (e.g., no distinct folders for modules, libraries, or layers).
- The output is a flattened representation, omitting the source's modular and layered architecture.

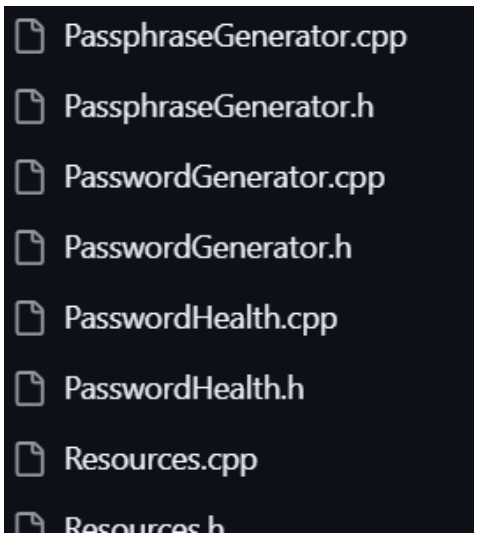
Reconstruction by Ghidra

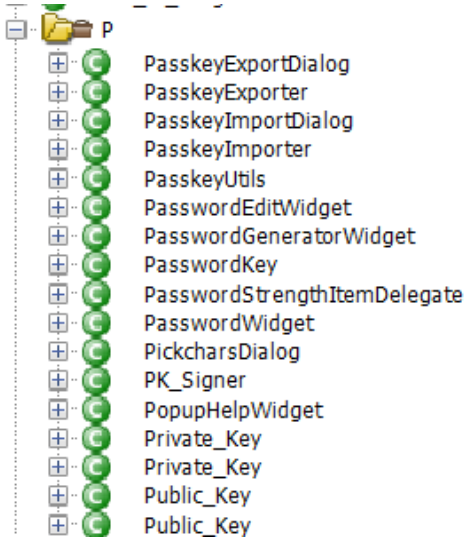
No Representation of File structure:
Ghidra does not provide a direct representation of the original file structure within a program. Once a program is compiled into a binary, the compiler removes all traces of the source file organization, such as `.cpp` and `.h` files or logical groupings into modules. Ghidra, therefore, focuses on analyzing the binary data, such as functions, instructions, and data segments. It does not retain or visualize the higher-level organization of the code file, such as file boundaries, module grouping, or the way the file's contents are logically partitioned. As a result, the overall structure of the source code, including file divisions and hierarchical relationships between different code sections, is not preserved or represented in the tool's interface.

Incorrect Representation of classes:
When analyzing classes, Ghidra's approach is often incomplete or lacks accuracy. During compilation, high-level class structures, including member variables and inheritance hierarchies, are typically flattened.Ghidra attempts to group identified functions under respective classes, but it often lacks a clear representation of the class structures itself.Ghidra typically groups functions under their respective classes, but the full class structure, including member variables, inheritance hierarchies, and object layout, is not explicitly shown. In some cases, Ghidra may even fail to recognize certain classes altogether, which can lead to gaps in the class map. Consequently, the tool may only list functions without providing the broader

context of the classes they belong to, resulting in an incomplete view of the program's object-oriented design.

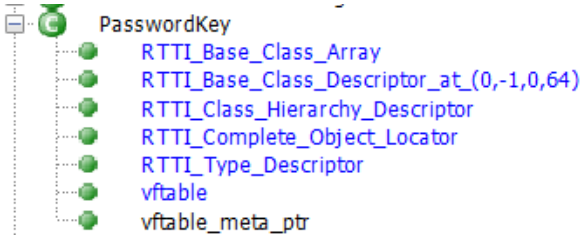
Missing Classes:

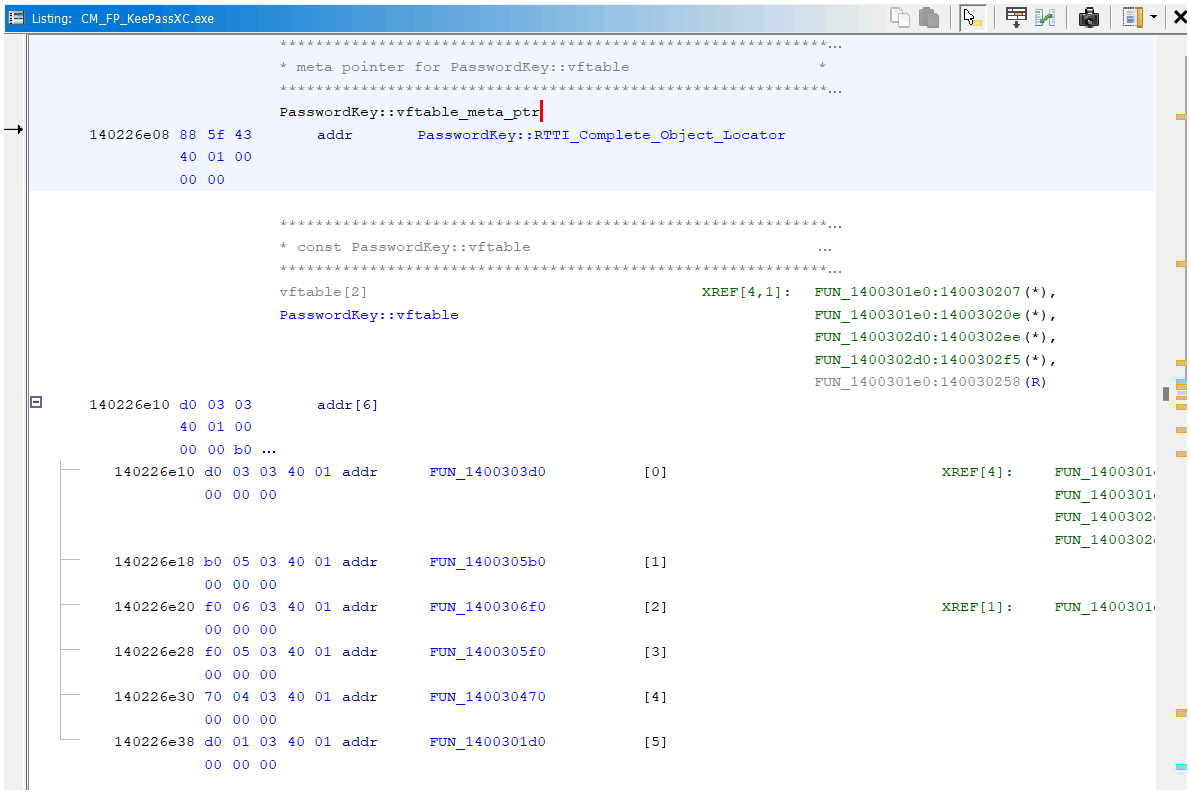




Classes that consist of only static methods or simple data structures are typically optimized by the compiler into standalone functions or raw memory layouts. This makes it difficult for Ghidra to reconstruct these classes during analysis. Ghidra treats them as plain data structures or groups of functions rather than as full-fledged object classes. In these cases, the individual functions are identified and classified, but the connection to the class itself may not be made, resulting in an incomplete class map

```
class PasswordKey : public Key
```





Polymorphic Classes (Vtables):

In case of polymorphism , not all the structure of classes is missed. In vftables(Virtual Function Tables) it tries to map them to virtual functions, but it doesn't always connect them to specific classes cleanly.

For us to get better understanding we need to manually infer and reconstruct class using vtable,function patterns and memory analysis. Manually map vtable entries to function names and class members by using Ghidra's Listings option and inspecting function signatures.

Unintuitive Function and Variable Names:

When analyzing binaries, Ghidra assigns generic names to functions and variables because the compiler removes symbolic names during the build process. However, these names often lack meaningful context related to the actual purpose of the functions or variables, making it difficult to keep track of the system. Typically, the names consist of a prefix that defines the type (e.g., function, label, or data), followed by their location in the binary. While this naming convention helps identify and locate entities within the binary, it does not provide descriptive or intuitive names that reflect their functionality.

Prefix	Meaning
FUN_	Function entry point or subroutine.
LAB_	A label, typically a branch or jump destination
DAT_	Data reference, such as global or static variables
unk_	Unknown type or structure, yet to be classified.
sub_	Subroutine or function, alternative to `fun_`.
loc_	Local variable or local label in assembly code.
str_	String data in memory (e.g., ASCII or Unicode strings).
seg_	Segment identifier, referring to specific memory sections.

Example:

```
undefined8 FUN_1401df1c0(longlong param_1,QString *param_2)
```

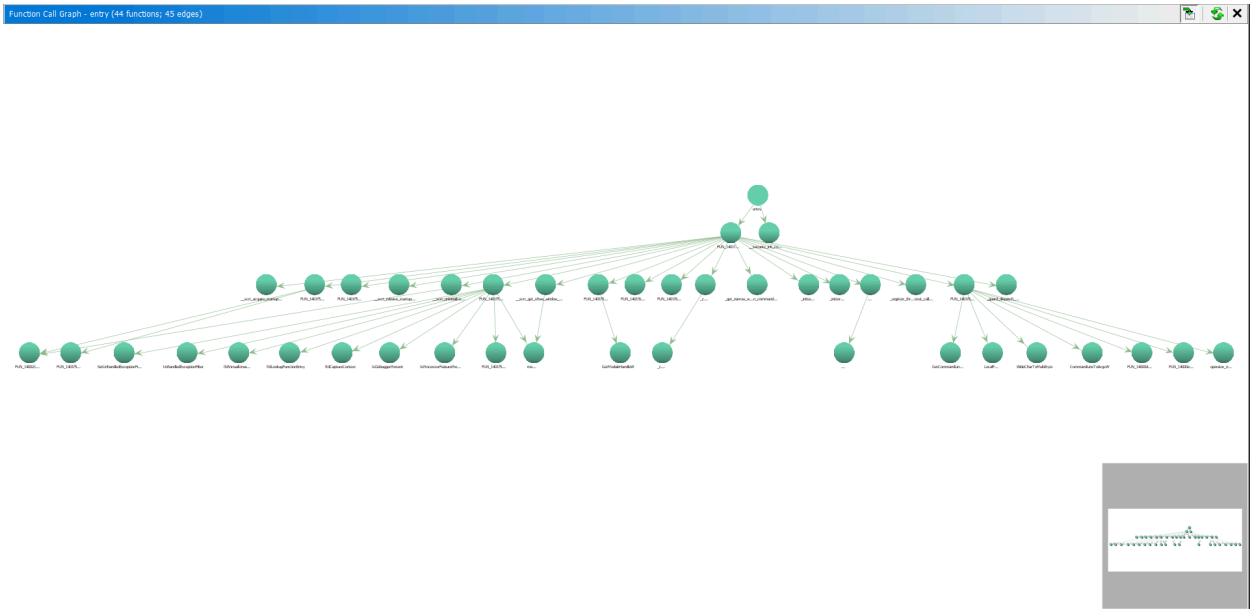
However, Ghidra allows users to rename variables and functions as well as add comments in the decompiled code to aid in the reverse engineering process. By utilizing this functionality, we don't have to figure out what a function does everytime we come back to it, instead we can leave various types of comments and rename variables and functions which act as reminders for us.

```
void lookup_hash_table(longlong param_1,undefined4 *param_2,undefined8 param_3)
```

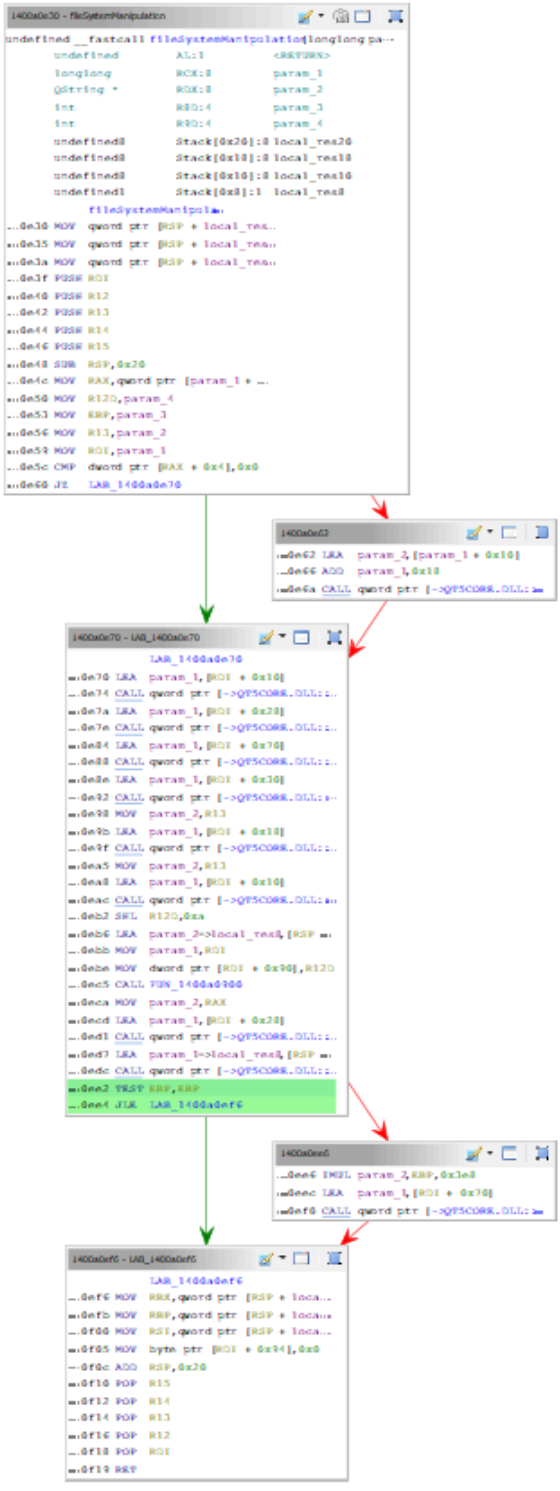
Appendix

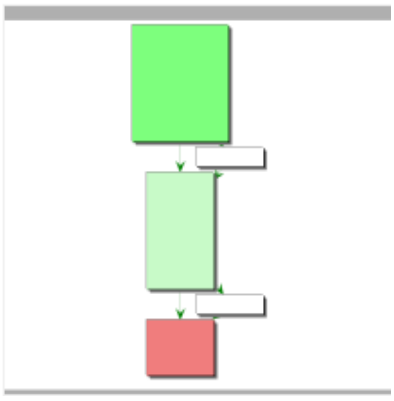
Ghidra:

Function call graph from entry point

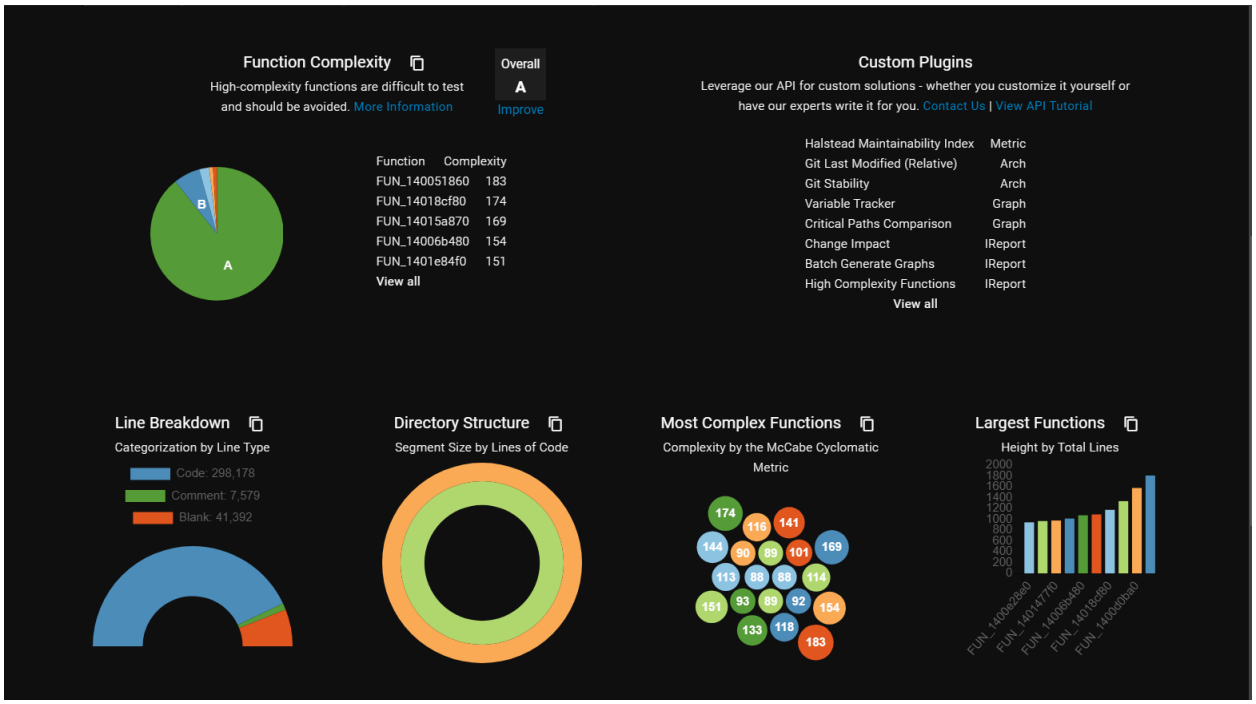


Function graph of overall system:


undefined __fastcall fileSystemManipulation(longlong pa...
undefined AL:1 <RETURN>
longlong RCX:8 param_1
QString * RDX:8 param_2
int R8D:4 param_3
int R9D:4 param_4
undefined8 Stack[0x20]:8 local_res20
undefined8 Stack[0x18]:8 local_res18
undefined8 Stack[0x10]:8 local_res10
undefined1 Stack[0x8]:1 local_res8
fileSystemManipula...
...0e30 MOV qword ptr [RSP + local_res...
...0e35 MOV qword ptr [RSP + local_res...
...0e3a MOV qword ptr [RSP + local_res...
...0e3f PUSH RDI
...0e40 PUSH R12
...0e42 PUSH R13
...0e44 PUSH R14
...0e46 PUSH R15
...0e48 SUB RSP,0x20
...0e4c MOV RAX,qword ptr [param_1 + ...
...0e50 MOV R12D,param_4
...0e53 MOV RBP,param_3
...0e56 MOV R13,param_2
...0e59 MOV RDI,param_1
...0e5c CMP dword ptr [RAX + 0x4],0x0
...0e60 JZ LAB_1400a0e70
1400a0e62 LEA param_2,[param_1 + 0x10]
...0e66 ADD param_1,0x18
...0e6a CALL qword ptr [->QT5CORE.DLL::...
1400a0e70 - LAB_1400a0e70
LAB_1400a0e70
...0e70 LEA param_1,[RDI + 0x10]
...0e74 CALL qword ptr [->QT5CORE.DLL::...
...0e7a LEA param_1,[RDI + 0x28]
...0e7e CALL qword ptr [->QT5CORE.DLL::...
...0e84 LEA param_1,[RDI + 0x70]
...0e88 CALL qword ptr [->QT5CORE.DLL::...
...0e8e LEA param_1,[RDI + 0x30]
...0e92 CALL qword ptr [->QT5CORE.DLL::...
...0e98 MOV param_2,R13
...0e9b LEA param_1,[RDI + 0x18]
...0e9f CALL qword ptr [->QT5CORE.DLL::...
...0ea5 MOV param_2,R13
...0ea8 LEA param_1,[RDI + 0x10]
...0eac CALL qword ptr [->QT5CORE.DLL::...
...0eb2 SHL R12D,0x4
...0eb6 LEA param_2=>local_res8,[RSP ...
...0ebb MOV param_1,RDI
...0ebe MOV dword ptr [RDI + 0x90],R12D
...0ec5 CALL FUN_1400a0900
...0eca MOV param_2,RAX
...0ecd LEA param_1,[RDI + 0x28]
...0ed1 CALL qword ptr [->QT5CORE.DLL::...
...0ed7 LEA param_1=>local_res8,[RSP ...
...0edc CALL qword ptr [->QT5CORE.DLL::...
...0ee2 TEST RBP,RBP
...0ee4 JLE LAB_1400a0ef6
1400a0ee6 IMUL param_2,RBP,0x1e8
...0eec LEA param_1,[RDI + 0x70]
...0ef0 CALL qword ptr [->QT5CORE.DLL::...
1400a0ef6 - LAB_1400a0ef6
LAB_1400a0ef6
...0ef6 MOV RBX,qword ptr [RSP + loca...
...0efb MOV RBP,qword ptr [RSP + loca...
...0f00 MOV R12,qword ptr [RSP + loca...
...0f05 MOV byte ptr [RDI + 0x94],0x0
...0f0c ADD RSP,0x20
...0f10 POP R15
...0f12 POP R14
...0f14 POP R13
...0f16 POP R12
...0f18 POP RDI
...0f19 RET



Understand Report for Decompiled code:



Understand Report for Original code:


Most Complex Functions
Complexity by the McCabe Cyclomatic Metric
86
34
31
75
29
25
25
36
31
23
24
29
56
28
25
25
51
34
33
111
Largest Functions
Height by Total Lines
1000
900
800
700
600
500
400
300
200
0
parseSequence
testDa... tings
sizeFromContents
MainWindow
drawPrimitive
Most Complex Classes
Complexity by the Average McCabe Cyclomatic Metric
23
14
7
8
6
6
6
9
7
5
5
7
15
7
6
6
10
8
7
25
Largest Classes
Height by Total Lines
3500
3000
2500
2000
1500
1000
500
0
EditEntityWidget
BrowserService
TestGuiFdoSecrets
TestCli
DatabaseWidget
Most Complex Files
Complexity by the Average McCabe Cyclomatic Metric
25
10
10
12
8
7
8
14
8
7
7
9
14
8
7
8
14
10
9
14
Largest Files
Height by Total Lines
25000
20000
15000
10000
5000
0
BrowserService.cpp
zxcvbn.c
MainWindow.cpp
TestGui.cpp
BaseStyle.cpp