Streamlined Analysis of Application Crashes using WinDbg

A Dissertation Report Submitted in partial fulfilment of the requirements for the online internship offered by Centre of Excellence in Cyber Security, JNTUH

4 WEEK ONLINE INTERNSHIP

in

CYBERSECURITY and FORENSICS

Submitted by

Mr. Kamarajugadda Jyothi Phani Vaibhav (20SS1A0525)
Miss. Gadipelli Keerthi (21JJ1A0522)
Mr. Sanjeeb Kumar Sahoo (VP22CSCI0200082)
Mr. N Mithun Krishna (21D41A6246)
Miss. Gudapati Harshitha (20WH1A05D1)

Under the mentoring of

Mr. P. Radha Krishna Sastry,
Module Leader, E-Security team,
CDAC

CENTRE OF EXCELLENCE CYBERSECURITY
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
Kukatpally, Hyderabad - 500 085, Telangana, India

Streamlined Analysis of Application Crashes using WinDbg

.

CENTRE OF EXCELLENCE CYBERSECURITY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

Kukatpally, Hyderabad - 500 085, Telangana, India

CERTIFICATE

This certificate acknowledges the successful completion of the internship project titled "Streamlined Analysis of Application Crashes using WinDbg" by the team, comprising of Mr. Kamarajugadda Jyothi Phani Vaibhav, Miss. Gadipelli Keerthi, Mr. Sanjeeb Kumar Sahoo, Mr. N. Mithun Krishna, Miss. Gudapati Harshitha. This project is a vital component of the partial requirement for the Four Week Online Internship Program on "CYBER SECURITY & FORENSICS" held from August 28th to September 23rd, 2023. This commendable work is conducted at Centre of Excellence in Cyber Security, Jawaharlal Nehru Technological University Hyderabad, reflecting the team's dedication and expertise in the field of Cyber Security and Cyber Forensics.

Dr R. Sridevi
Professor in CSE &
Co. Ordinator, Centre of Excellence in Cyber Security, JNTUH

CENTRE OF EXCELLENCE CYBERSECURITY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

DECLARATION

We declare that the internship project work entitled "**Streamlined Analysis of Application Crashes using WinDbg**" submitted in Centre of Excellence Cyber Security, Jawaharlal Nehru Technological University Hyderabad, in partial fulfilment of the requirement for the award of the 4-week Online Internship in Centre of Excellence Cyber Security is a bonafide record of our own work carried out under the mentoring of **Mr. P. Radha Krishna Sastry, Module Leader, E-Security team, CDAC**

Kamarajugadda Jyothi Phani Vaibhav	**Gadipelli Keerthi**	**Sanjeeb Kumar Sahoo**	**N Mithun Krishna**	**Gudapati Harshitha**
20SS1A0525	21JJ1A0522	VP22CSCI0200082	21D41A6246	20WH1A05D1

ACKNOWLEDGEMENT

Abstract

The dissertation presents a comprehensive investigation and development of an automated system for analyzing and classifying memory dump files generated during software crashes. Memory dump files provide valuable insights into the causes of software crashes and can aid in debugging and resolving critical issues. However, the manual analysis of these files is time-consuming and error-prone, often requiring expertise in debugging tools like WinDbg.

The proposed system leverages automation to streamline the analysis process and classify memory dump files based on the exceptions they represent. The core components of the system include the WinDbg debugger, a Python script, and a predefined set of exception categories (e.g., Stack Overflow, Buffer Overflow, Use After Free, Pointer Related).

The dissertation discusses the implementation details of the automated system, including the integration of WinDbg, the structure of the Python script, and the methodology for exception classification. Additionally, it evaluates the system's performance in terms of accuracy and efficiency by analyzing a diverse set of memory dump files.

In conclusion, this research presents an innovative approach to automate the analysis and classification of memory dump files, facilitating the debugging process and contributing to the overall software development lifecycle. The system offers a valuable tool for software engineers and developers working to diagnose and resolve software crashes efficiently.

TABLE OF CONTENTS

1.INTRODUCTION

Software stability and reliability are paramount in today's digital landscape, where software applications underpin various aspects of our lives. However, despite rigorous development and testing processes, software crashes and unexpected failures remain inevitable occurrences. When these failures happen, diagnosing their root causes is critical for resolving issues and enhancing software quality.

One valuable source of information for diagnosing software crashes is memory dump files. These files capture the state of a software application at the moment of a crash, providing insights into the underlying problems. Analyzing memory dump files, however, can be a challenging and time-consuming task, often requiring expertise in debugging tools and manual intervention.

This dissertation addresses the need for efficient and automated memory dump file analysis and classification. The primary goal is to develop a system that streamlines the process of analyzing these files, identifies the type of exceptions they represent, and categorizes them accordingly. By automating this process, software developers and engineers can expedite the debugging and issue-resolution process, ultimately leading to improved software reliability.

The proposed system leverages the capabilities of the WinDbg debugger, a powerful tool for analyzing Windows memory dump files. Through a custom Python script, this system interacts with WinDbg, extracts essential information from the analysis reports, and classifies the dump files into distinct categories based on the type of exception encountered. The system aims to provide a user-friendly and efficient solution for handling memory dump files generated during software crashes.

This dissertation discusses the implementation and evaluation of the automated system, emphasizing its contribution to the software development lifecycle. The system's performance is assessed in terms of accuracy and efficiency, and its potential benefits in terms of time and resource savings are highlighted.

In summary, this dissertation addresses a critical aspect of software engineering by automating the analysis and classification of memory dump files. The developed system offers a valuable tool for software professionals, aiding them in identifying and resolving software crashes more efficiently, thereby enhancing software reliability and end-user satisfaction.

2.OBJECTIVE

1. **Memory Dump Analysis:** The primary objective of this research is to develop an automated system for the analysis of memory dump files generated during software crashes. The system aims to streamline the process of diagnosing the root causes of crashes, eliminating the need for manual analysis and reducing human intervention in the debugging process.

2. **Exception Type Classification:** The system seeks to accurately classify memory dump files into distinct categories based on the type of exception encountered. These categories include "Stack Overflow," "Buffer Overflow," "Pointer Related," and "UseAfterFree." The objective is to improve the efficiency of identifying the underlying issues in software crashes.

3. **Integration with WinDbg:** The research aims to integrate the developed system with the WinDbg debugger, a powerful tool for analyzing Windows memory dump files. This integration involves the automation of WinDbg commands and the extraction of essential information from analysis reports.

4. **User-Friendly Interface:** An important objective is to create a user-friendly interface for software developers and engineers. The system should be intuitive and easy to use, allowing users to analyze and categorize memory dump files without requiring in-depth knowledge of debugging tools.

5. **Performance Evaluation:** The research includes an evaluation of the system's performance in terms of accuracy and efficiency. The objective is to assess the system's ability to correctly identify exception types and expedite the debugging process. Performance metrics will be used to quantify the system's impact on software development.

6. **Resource and Time Savings:** The dissertation aims to demonstrate the potential resource and time savings achieved through the automation of memory dump file analysis. It will quantify the reduction in manual effort and the acceleration of issue resolution in software development.

7. **Enhancing Software Reliability:** Ultimately, the research objectives are aligned with the goal of enhancing software reliability. By automating the analysis and classification of memory dump files, the system contributes to improving software

quality and reducing downtime caused by crashes.

In summary, the objectives of this dissertation encompass the development of an automated system for memory dump file analysis, accurate exception classification, integration with WinDbg, user-friendliness, performance evaluation, resource and time savings, and the overall enhancement of software reliability. The research aims to provide a valuable tool for software professionals, addressing the challenges associated with software crashes and failures.

3.LITERATURE SURVEY

The analysis and debugging of software crashes and failures have long been critical challenges in the field of software development. This literature survey provides an overview of existing research, tools, and methodologies related to the automated analysis of memory dump files and the classification of exceptions.

1. **Memory Dump Analysis:**

1.1. WinDbg: The Windows Debugger (WinDbg) has been a fundamental tool for memory dump analysis. WinDbg provides a command-line interface for examining memory dumps, extracting stack traces, and diagnosing exceptions. It has been widely used by software developers and engineers to identify the root causes of crashes.

1.2. Debugger Scripting: Researchers and developers have extensively explored the automation of memory dump analysis using debugger scripting languages. Scripting languages like JavaScript and Python have been employed to create scripts that execute debugger commands, extract relevant information, and generate analysis reports.

2. **Exception Classification:**

2.1. Exception Types: Various exception types can lead to software crashes, including "Stack Overflow," "Buffer Overflow," "Pointer Related," and "UseAfterFree." Research has focused on developing algorithms and techniques to accurately classify memory dump files into these categories based on exception information and stack traces.

2.2. Pattern Recognition: Machine learning and pattern recognition approaches have been investigated to classify exceptions. These approaches often involve training models on labeled datasets to distinguish between different exception types automatically.

3. **Automation in Debugging:**

3.1. Automated Debugging Tools: Beyond memory dump analysis, automated debugging tools and techniques have gained attention. These tools aim to automate the process of identifying and fixing software defects, including memory-related issues. Some commercial and open-source solutions offer automated debugging capabilities.

3.2. Integration with Development Environments: Researchers have explored integrating automated debugging and analysis tools directly into software development environments, enhancing the debugging workflow for developers.

4. **User-Friendly Interfaces:**

4.1. Graphical User Interfaces (GUIs): The development of user-friendly graphical interfaces for memory dump analysis and debugging has been an area of interest. These interfaces aim to make the debugging process more accessible to developers with varying levels of expertise.

5. **Performance Evaluation:**

5.1. Accuracy and Efficiency: Evaluating the accuracy and efficiency of automated memory dump analysis tools is essential. Research often includes benchmarking against manual analysis and measuring the system's ability to correctly classify exceptions and expedite the debugging process.

6. **Software Reliability:**

6.1. Impact on Software Reliability: The ultimate goal of automated memory dump analysis is to enhance software reliability by identifying and resolving critical issues that lead to crashes. Studies have examined the impact of automated debugging tools on software quality and stability.

7. **Resource and Time Savings:**

7.1. Cost-Benefit Analysis: Several studies have conducted cost-benefit analyses to quantify the resource and time savings achieved through automation. These analyses help organizations understand the return on investment in automated debugging solutions.

In conclusion, the literature survey highlights the significance of automated memory dump analysis and exception classification in the context of software debugging. It underscores the diverse approaches and tools available to address these challenges, with a focus on improving software reliability and developer productivity. This research builds upon and contributes to this body of knowledge by developing an automated system for memory dump analysis and exception classification, aiming to provide a valuable resource for the software development community.

4. SYSTEM DESIGN

1. System Architecture:

The system architecture of the code is designed to analyze and categorize Windows dump files (.dmp) based on specific exception types. It comprises several key components and follows a structured workflow:

1.1. Dump Files Repository: The system begins by collecting dump files, which are memory snapshots of Windows processes that have encountered exceptions. These dump files are stored in the "dump_folder."

1.2. Exception Categorization: The primary objective of the system is to categorize these dump files into specific exception types:

1.2.1. Stack Overflow: These are categorized when the exception code "c00000fd (Stack overflow)" is encountered in the dump file analysis.

1.2.2. UseAfterFree: This category is determined when the dump file analysis identifies attempts to read from or write to specific memory addresses.

1.2.3. Pointer Related: This category includes dump files where attempts to write to memory addresses are detected.

1.2.4. Buffer Overflow: These dump files fall into the "Buffer Overflow" category when there are attempts to write to memory addresses.

1.3. Exception Folders: To maintain an organized structure, the system creates subfolders within the "dump_folder" for each exception type. These folders include "Stack Overflow," "UseAfterFree," "Pointer Related," and nested subfolders as needed, such as "UseAfterFree."

1.4. Analysis and Categorization Process: The core of the system lies in the analysis and categorization process for each dump file. This process involves:

1.4.1. Using Windbg (Windows Debugger), specified by the "windbg_path," to analyze each dump file. Executing specific debugging commands ("-c .ecxr;.exr -1;q") to extract information about the exception and saving the analysis output to a corresponding text file.

1.4.2. Identifying the exception type by parsing the analysis output. Exception types are determined based on predefined patterns, such as exception codes or memory access attempts.

1.5. Result Handling: After categorizing a dump file, the system moves the analysis result text file to the appropriate exception folder. This step helps maintain a well-organized structure for further analysis or reference.

2. System Components

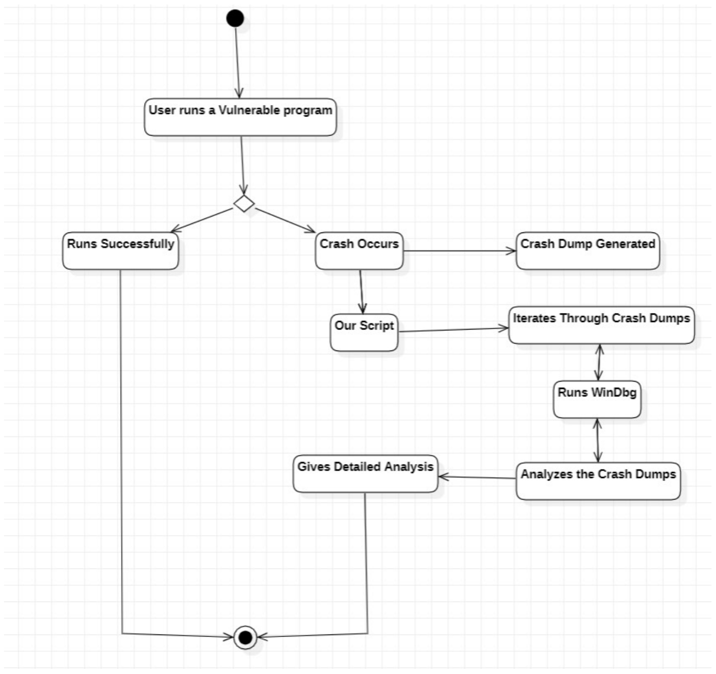
2.1. Windbg: The Windows Debugger is a crucial component that handles the analysis of dump files. It is invoked using the "windbg_path" configuration.

2.2. Dump Files: These are the raw input files stored in the "dump_folder." Each file represents a memory snapshot of a Windows process at the time of an exception.

2.3. Analysis and Categorization Module: This module is responsible for invoking Windbg, executing debugging commands, and parsing the analysis results. It also determines the exception type and moves the result to the appropriate folder.

2.4. Exception Folders: These folders serve as the categorized output repository, with each folder representing a specific exception type.

3. Data Flow:



4. Interactions:

The code you have provided interacts with multiple components, both internal and external, to achieve its goals. These interactions are vital for the functioning of the system and can be categorized as follows:

4.1. Interaction with the Operating System:

4.1.1. The code interacts with the operating system to access and manipulate files and directories.

4.1.2. It lists all files in a specified directory and identifies those with the '.dmp' file extension. These files are considered as input for further analysis.

4.2. Interaction with Windows Debugger (Windbg):

4.2.1. The code invokes the Windows Debugger (Windbg) tool using the path specified by the `windbg_path` variable.

4.2.2. It uses Windbg to analyze each dump file individually. The following interactions occur during this process:

4.2.3. The `-z` flag specifies the dump file to be analyzed.

4.2.4. The `-c` flag provides a series of commands to be executed within Windbg, including analyzing the exception context (`ecxr`), displaying the exception record (`exr -1`), and quitting (`q`).

4.2.5. The `-logo` flag specifies the path for saving the analysis results in a text file.

4.2.6. The analysis results are saved in files with names based on the input dump file names, appended with '_analysis.txt'.

4.3. Interaction with Exception Categorization:

4.3.1. After the analysis, the code categorizes the dump files based on the type of exception detected. The categories include:

1. Stack Overflow
2. Buffer Overflow
3. Pointer Related
4. Use After Free

4.3.2. It creates separate folders for each exception type and moves the analysis results into the respective folders based on the detected exception type.

5. Interaction with Exception Content:

The code reads the content of the analysis results files to determine the type of exception encountered. It searches for specific patterns or keywords within the content to identify the exception type.

6. Configuration and Paths:

Provide details about the configuration settings and paths used in the system. Explain why specific paths are chosen and what they represent.

7. Error Handling:

The code incorporates several error handling mechanisms to ensure the reliability and robustness of the analysis process. These mechanisms address various stages of execution and potential issues that may arise:

7.1. Error Handling for Input Validation:

7.1.1. The code starts by validating the input paths specified in the `windbg_path` and `dump_folder` variables. It ensures that these paths exist and are accessible.

7.1.2. If the specified paths are invalid or inaccessible, the code will raise an exception or error. The error message will indicate the nature of the issue, such as a missing file or directory.

7.2. Error Handling for File Operations:

7.2.1. The code interacts with the file system to list dump files, read their contents, and move analysis results. To ensure these file operations are successful:

7.2.2. It checks if each file in the `dump_folder` has the '.dmp' file extension before processing it.

7.2.3. It handles file read errors when attempting to open and read analysis result files.

7.2.4. It addresses potential issues that may occur when moving files between directories, such as permission errors or directory not found errors.

7.2.5. Any errors encountered during file operations will be reported along with a description of the issue. This includes errors related to opening, reading, or moving files.

7.3. Error Handling for External Tool Execution:

7.3.1. The code utilizes the Windows Debugger (Windbg) tool for dump file analysis. It constructs and executes a command to invoke Windbg for each dump file.
7.3.2. It employs error handling to manage potential issues that may arise during this external tool execution, such as:
7.3.3. Errors related to incorrect or inaccessible Windbg paths specified in the `windbg_path` variable.
7.3.4. Errors that may occur during the execution of Windbg commands.
7.3.5. Any issues related to the availability and compatibility of the Windbg tool.
7.3.6. If an error occurs during the execution of Windbg or any other subprocess, the code captures and reports the error, providing context for troubleshooting.

7.4. Error Handling for Exception Categorization:

7.4.1. The code categorizes exception types based on specific patterns within the analysis results. To ensure accurate categorization, it employs error handling to address situations where patterns cannot be found or identified.
7.4.2. If an analysis result does not match any predefined patterns, it will be categorized as 'Unknown,' and this information will be included in the report.

8. Dependencies:

8.1. Windows Debugger (Windbg):

8.1.1. The core component of the code is the Windows Debugger, commonly known as Windbg. It is an external tool provided by Microsoft for debugging and analyzing Windows applications.
8.1.2. The code heavily depends on the availability and correct configuration of Windbg, as it uses it to analyze dump files and extract critical information.
8.1.3. Ensure that the correct path to Windbg is specified in the `windbg_path` variable. The code won't function without a valid Windbg installation.

8.2. Python Libraries:

8.2.1. The code is written in Python, a versatile programming language. Although not explicitly mentioned, it relies on the Python standard library, which includes essential modules for file and directory operations, subprocess execution, and string manipulation

8.2.2. Ensure that a compatible version of Python is installed on the system where the code will run.

8.3. Operating System:

8.3.1. The code is designed to run on a Windows operating system, as it utilizes Windows-specific tools like Windbg. It may not function correctly on other operating systems.

8.4. File System Access:

8.4.1. The code interacts with the file system to perform various tasks, such as listing dump files, reading their contents, and moving analysis results. Therefore, it requires appropriate permissions to access, read, and write files and directories.

8.4.2. Ensure that the code is executed with the necessary file system permissions to prevent permission-related errors.

9. Scalability and Performance:

Scalability and Performance is a crucial aspect of any software system, as it determines how well the code can adapt to growing demands and changing requirements. In the case of the provided code, which deals with analyzing dump files, scalability considerations are important to ensure its effectiveness and efficiency as the volume of dump files or the complexity of analysis scenarios increases.

9.1. Handling Large Datasets:

As the number of dump files to be analyzed grows, the code should be able to efficiently handle large datasets without a significant decrease in performance. It's essential to optimize data structures and algorithms to ensure that analysis remains responsive even with a substantial increase in input data.

9.2. Parallel Processing:

To enhance scalability, consider implementing parallel processing techniques. This allows the code to distribute the analysis of multiple dump files across multiple CPU cores or even multiple machines, significantly reducing analysis time for large-datasets.

9.3. Resource Management:

Scalability should also consider resource utilization, such as memory and CPU usage. The code should be designed to use system resources judiciously and release them when analysis is complete to accommodate additional requests or tasks.

9.4. Load Balancing:

In scenarios where multiple users or applications access the code concurrently, load balancing mechanisms can be implemented to distribute incoming requests evenly. This prevents overloading the code and ensures fair allocation of resources.

5.PROPOSED WORK

The proposed work is centered around the development and enhancement of a system for analyzing memory dump files generated during software crashes. Memory dump files contain valuable information about the state of a program at the time of the crash, and analyzing them can help identify the root causes of software failures, including exceptions such as stack overflow, buffer overflow, and pointer-related issues.

The main objective of this proposed work is to create a robust and automated system that performs the following tasks:

1. **Automated Memory Dump Analysis:** Develop a script or software tool that automates the analysis of memory dump files. This tool will be capable of processing a collection of memory dump files and extracting relevant information about the crash.

2. **Exception Type Detection:** Implement algorithms and rules for detecting different types of exceptions within the memory dump files. The primary exception types of interest include:
   2.1. Stack Overflow: Identifying instances where the program's call stack has exceeded its allocated memory.
   2.2. Buffer Overflow: Detecting situations where a program writes beyond the boundaries of a buffer.
   2.3. Pointer Related: Recognizing issues related to invalid or corrupted memory pointers.
   2.4. UseAfterFree: Identifying attempts to access memory that has already been deallocated.

3. **Output Generation:** Create analysis reports for each memory dump file processed. These reports will include details about the detected exception type, relevant memory addresses, and any additional context that may help diagnose the issue.

4. **Organized Exception Folders:** Develop a file management system that organizes memory dump files into folders based on their exception types. This will help in efficient categorization and retrieval of crash data.

5. **User-Friendly Interface:** If applicable, design a user-friendly interface that allows users to interact with the tool easily. This can include options for selecting memory dump files and initiating the analysis process.
6. **Error Handling and Robustness:** Implement error handling mechanisms to ensure that the tool can gracefully handle unexpected scenarios or issues that may arise during the analysis process.

Methodology

The proposed work will involve the following steps:

1. **Environment Setup:** Configure the analysis environment, including specifying the paths to essential tools like WinDbg (Windows Debugger) and the directory containing memory dump files.

2. **Memory Dump Analysis:** Develop a script or software component that executes WinDbg with the appropriate commands to analyze each memory dump file. Extract relevant information and generate analysis reports.

3. **Exception Detection:** Implement algorithms to parse the analysis reports and identify the specific exception types based on predefined patterns and rules.

4. **Report Generation:** Create detailed analysis reports for each memory dump file, including the detected exception type and relevant information. Save these reports as text files.

5. **File Organization:** Move the analysis reports to appropriate folders based on the identified exception type. Maintain an organized directory structure for categorized crash data.

6. **User Interface (Optional):** If applicable, design a user interface to facilitate user interaction and provide a convenient way to initiate the analysis process.

7. **Testing and Validation:** Thoroughly test the system using various memory dump files to ensure accurate exception detection and report generation. Validate the tool's effectiveness in categorizing crashes.

Expected Outcomes

Upon successful implementation of the proposed work, the following outcomes are expected:
1. An automated system capable of efficiently analyzing memory dump files and categorizing exceptions.

2. Detailed analysis reports for each memory dump file, aiding in the diagnosis of software crashes.

3. Well-organized exception folders for easy retrieval and management of crash data.

4. Improved efficiency in identifying and addressing software issues related to exceptions, leading to enhanced software quality and reliability.

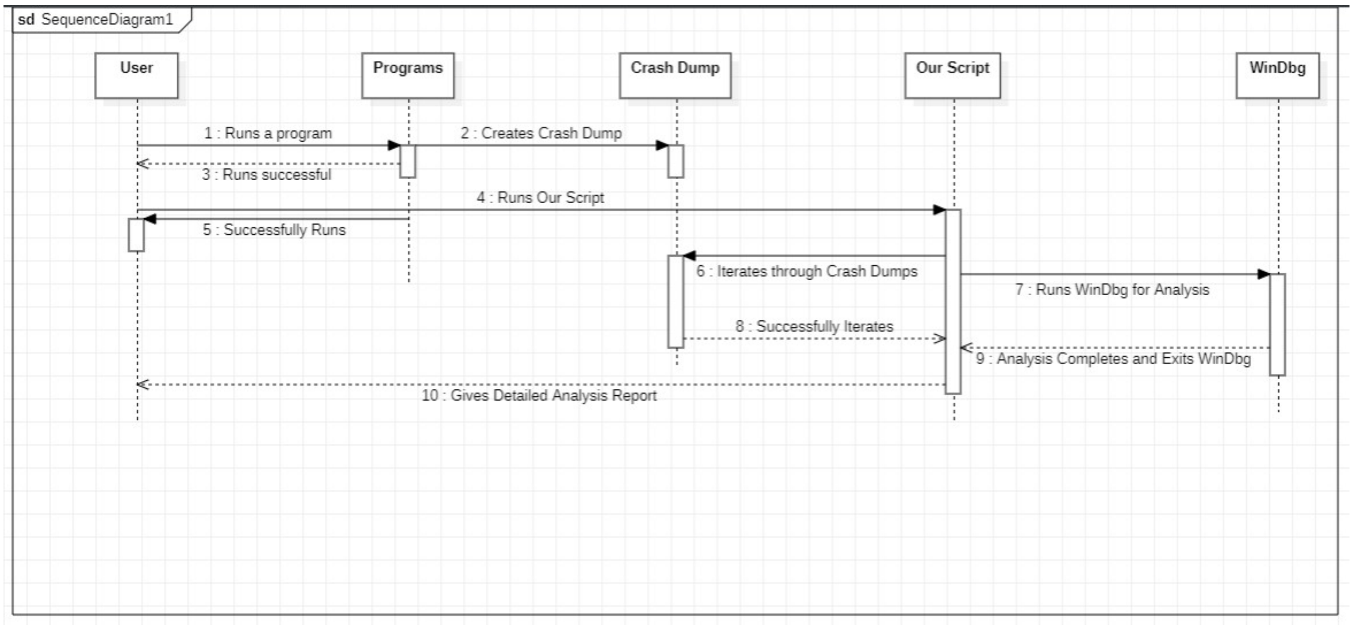
5. The potential for future enhancements and integration into software development and debugging workflows.

The proposed work aims to develop this system that streamlines the analysis of memory dump files, enabling the automatic detection and categorization of exceptions. This system will be a valuable tool for software developers and engineers, helping them identify and address critical issues that lead to software crashes. The organized categorization of crash data will facilitate more effective debugging and ultimately contribute to the improvement of software quality.

6. IMPLEMENTATION

The implementation of the system involves creating a script or program that can automate the process of analyzing memory dump files, detecting exception types, generating analysis reports, and organizing the files into categorized folders. Below is an overview of the key steps and components of the implementation:

1. Sequence Diagram:


sd SequenceDiagram1
User
Programs
Crash Dump
Our Script
WinDbg
1 : Runs a program
2 : Creates Crash Dump
3 : Runs successful
4 : Runs Our Script
5 : Successfully Runs
6 : Iterates through Crash Dumps
7 : Runs WinDbg for Analysis
8 : Successfully Iterates
9 : Analysis Completes and Exits WinDbg
10 : Gives Detailed Analysis Report

2. Configuration Setup:

We used Microsoft Visual Studio for making our project. You need to change the Properties of the Project File in Microsoft Visual Studio and do the following steps for Disabling Mitigations:

1. ASLR : Linker→Advanced→Change Randomized Base Address flag to /DYNAMICBASE:NO from drop down list
2. DEP : Linker→Advanced→Change DEP flag to /NXCOMPAT:NO from drop down list
3.strcpy() Unsafe : C/C++→Preprocessor→Add CRT_SECURE_NO_WARNINGS to Preprocessr Definitions
4. Buffer Overrun Exception : C/C++→Code Generation→Change Security Check to Disable Security Check (/GS-) from drop down list
5. Basic Runtime Check Exception : This disables the Run-Time Check which throws the Exception - Stack around the variable was corrupted. To disable: C/C++→Code Generation→Change Basic Runtime Checks to Default.
6. Or alternative for point 3 is using #define_CRT_SECURE_NO_WARNINGS,

even before #include<stdio.h>

Set the paths for the following components:

2.2. `windbg_path`: The path to the WinDbg (Windows Debugger) executable.

2.3. `dump_folder`: The directory containing memory dump files to be analyzed.

3. Create Exception Folders:

Create folders for organizing memory dump files based on detected exception types. These folders include:

1. `"Stack Overflow"`
2. `"Buffer Overflow"`
3. `"Pointer Related"`
4. `"UseAfterFree"`

4. Exception Detection Function:

Implement the `move_file_based_on_exception` function, which takes the path to an analysis report file as input and determines the type of exception based on the content of the file. The function recognizes the following exception types:

1. Stack Overflow
2. Buffer Overflow
3. Pointer Related
4. UseAfterFree

5. Iterate Through Dump Files:

Loop through the memory dump files in the specified `dump_folder`.

6. Analysis and Report Generation:

For each memory dump file, create an output analysis report file. The report file's name is derived from the input dump file's name.

Execute WinDbg using subprocess with the necessary commands to analyze the memory dump file (`-z` for loading the dump file, `-c` for commands to be executed, and `-logo` for specifying the output file).

7. Exception Type Detection:

Use the `move_file_based_on_exception` function to determine the type of exception detected in the analysis report.

8. Report Output and File Organization:

Save the analysis report in the same directory as the memory dump file.

If an exception is detected, move the analysis report to the corresponding exception folder within the `dump_folder`.

Print messages indicating the progress and actions taken for each memory dump file.

9. Error Handling:

Implement error handling to capture and handle any subprocess errors that may occur during analysis.

10. User Feedback:

Provide informative console output messages to keep the user informed about the analysis progress, report generation, and file movements.

Testing and Validation:

Thoroughly test the implementation using a variety of memory dump files, including those with different exception types, to ensure accurate detection and categorization.

Validate the system's ability to create organized exception folders and produce analysis reports.

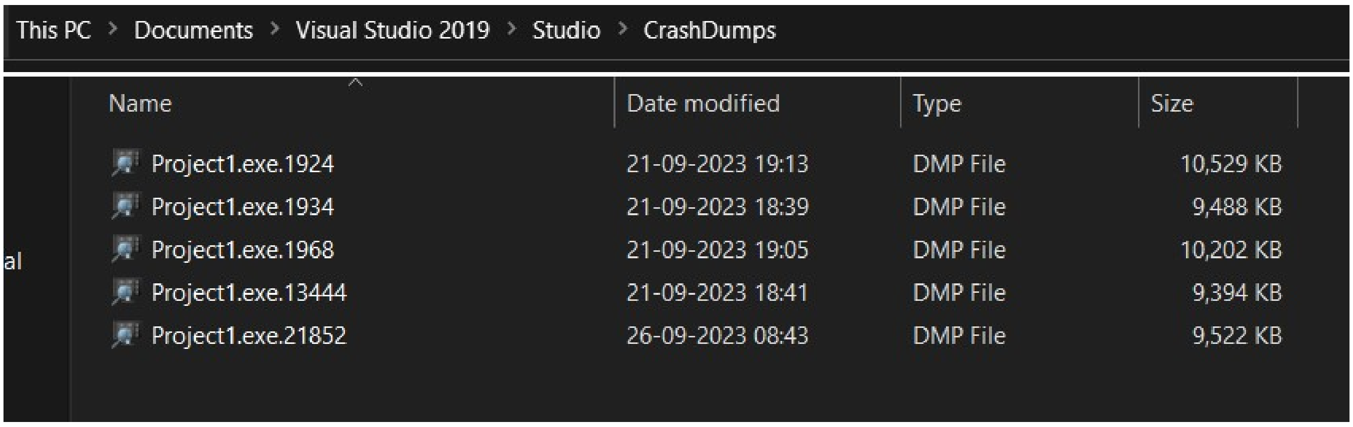
The below is the Script made by our team for Streamlined Analysis of Application Crashes using WinDbg System.

Script :

```python
#Make sure to add your own paths and OBSERVED_ADDRESS

import os
import subprocess
import re
import shutil

# Define paths and folders
windbg_path = r" ADD YOUR windbg.exe PATH "
dump_folder = r" ADD YOUR CrashDumps PATH "
output_path = r" ADD YOUR Report PATH "
analysis_file = r"analysis.txt"

# Create the output folder if it doesn't exist
os.makedirs(output_path, exist_ok=True)

# Function to create a folder name without special characters
def clean_folder_name(name):
    return re.sub(r'[()\' ]', '', name)

#Add your Exceptions here
def get_exception_type(content):
    if 'ExceptionCode: c00000fd (Stack overflow)' in content:
        return 'StackOverflow'
    elif 'Attempt to read from address' in content:
        return 'UseAfterFree'
    elif 'Attempt to write to address  OBSERVED_ADDRESS' in content:  # Replace with
your observed Address
        return 'PointerRelated'
    elif 'Attempt to write to address' in content:
        return 'BufferOverflow'
    else:
        return None

# Process dump files and generate output files in the corresponding folders
for dump_file in os.listdir(dump_folder):
    if dump_file.endswith('.dmp'):
        dump_file_path = os.path.join(dump_folder, dump_file)

        # Generate the output file path with the exception name
        output_file = os.path.splitext(dump_file)[0] + '_analysis.txt'
        output_file_path = os.path.join(output_path, output_file)

        cmd = [
            windbg_path,
            '-z', dump_file_path,
            '-c', '.ecxr;.exr -1;q',
            '-logo', output_file_path
        ]

        try:
            subprocess.run(cmd, check=True)
            print(f"Analysis for {dump_file} saved as {output_file}")

            # Read the content of the analysis file
            with open(output_file_path, 'r') as file:
                content = file.read()

            # Get the exception type from the content
            exception_type = get_exception_type(content)
            print(exception_type)

            # Update the folder name based on the exception type
            if exception_type:
                folder_name = clean_folder_name(exception_type)
            else:
                folder_name = "Unknown"

            # Move the output file to the corresponding folder
            destination_folder = os.path.join(output_path, folder_name)
            os.makedirs(destination_folder, exist_ok=True)
            new_file_path = os.path.join(destination_folder, output_file)
            shutil.move(output_file_path, new_file_path)

        except subprocess.CalledProcessError as e:
            print(f"Error analyzing {dump_file}: {e}")

# Concatenate analysis information into the analysis.txt file
analysis_file_path = os.path.join(output_path, analysis_file)
with open(analysis_file_path, "w") as analysis_file:
    for root, folders, _ in os.walk(output_path):
        for folder in folders:
            analysis_file.write(f"'{folder}':\n\n")
            folder_path = os.path.join(output_path, folder)
            for _, _, files in os.walk(folder_path):
                for file_name in files:
                    if file_name.endswith('_analysis.txt'):
                        dmp_file_name = file_name.replace('_analysis.txt', '.dmp')
                        analysis_file.write(f"File: {os.path.join(dump_folder,
dmp_file_name)}\n")
            analysis_file.write("\n\n")

print(f"\nAnalysis completed. Results saved in {output_file_path}")
```

7. RESULTS

These are the inputs for our system:

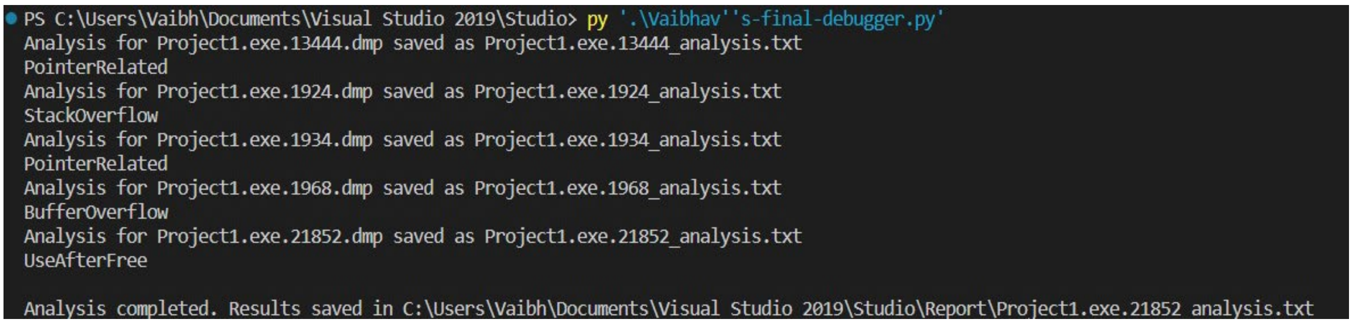


This PC › Documents › Visual Studio 2019 › Studio › CrashDumps

Name	Date modified	Type	Size
Project1.exe.1924	21-09-2023 19:13	DMP File	10,529 KB
Project1.exe.1934	21-09-2023 18:39	DMP File	9,488 KB
Project1.exe.1968	21-09-2023 19:05	DMP File	10,202 KB
Project1.exe.13444	21-09-2023 18:41	DMP File	9,394 KB
Project1.exe.21852	26-09-2023 08:43	DMP File	9,522 KB

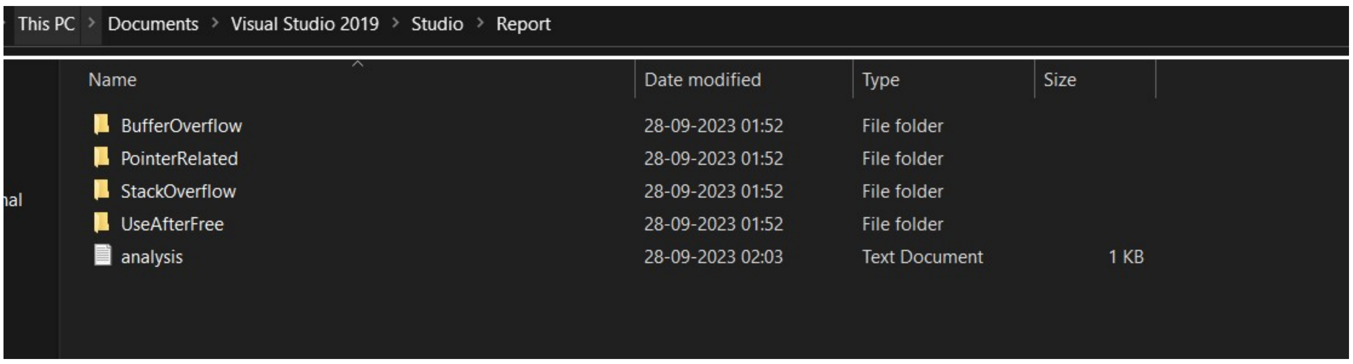
These are the outputs for our system:

At Terminal:



```
PS C:\Users\Vaibh\Documents\Visual Studio 2019\Studio> py '.\Vaibhav''s-final-debugger.py'
Analysis for Project1.exe.13444.dmp saved as Project1.exe.13444_analysis.txt
PointerRelated
Analysis for Project1.exe.1924.dmp saved as Project1.exe.1924_analysis.txt
StackOverflow
Analysis for Project1.exe.1934.dmp saved as Project1.exe.1934_analysis.txt
PointerRelated
Analysis for Project1.exe.1968.dmp saved as Project1.exe.1968_analysis.txt
BufferOverflow
Analysis for Project1.exe.21852.dmp saved as Project1.exe.21852_analysis.txt
UseAfterFree

Analysis completed. Results saved in C:\Users\Vaibh\Documents\Visual Studio 2019\Studio\Report\Project1.exe.21852_analysis.txt
```

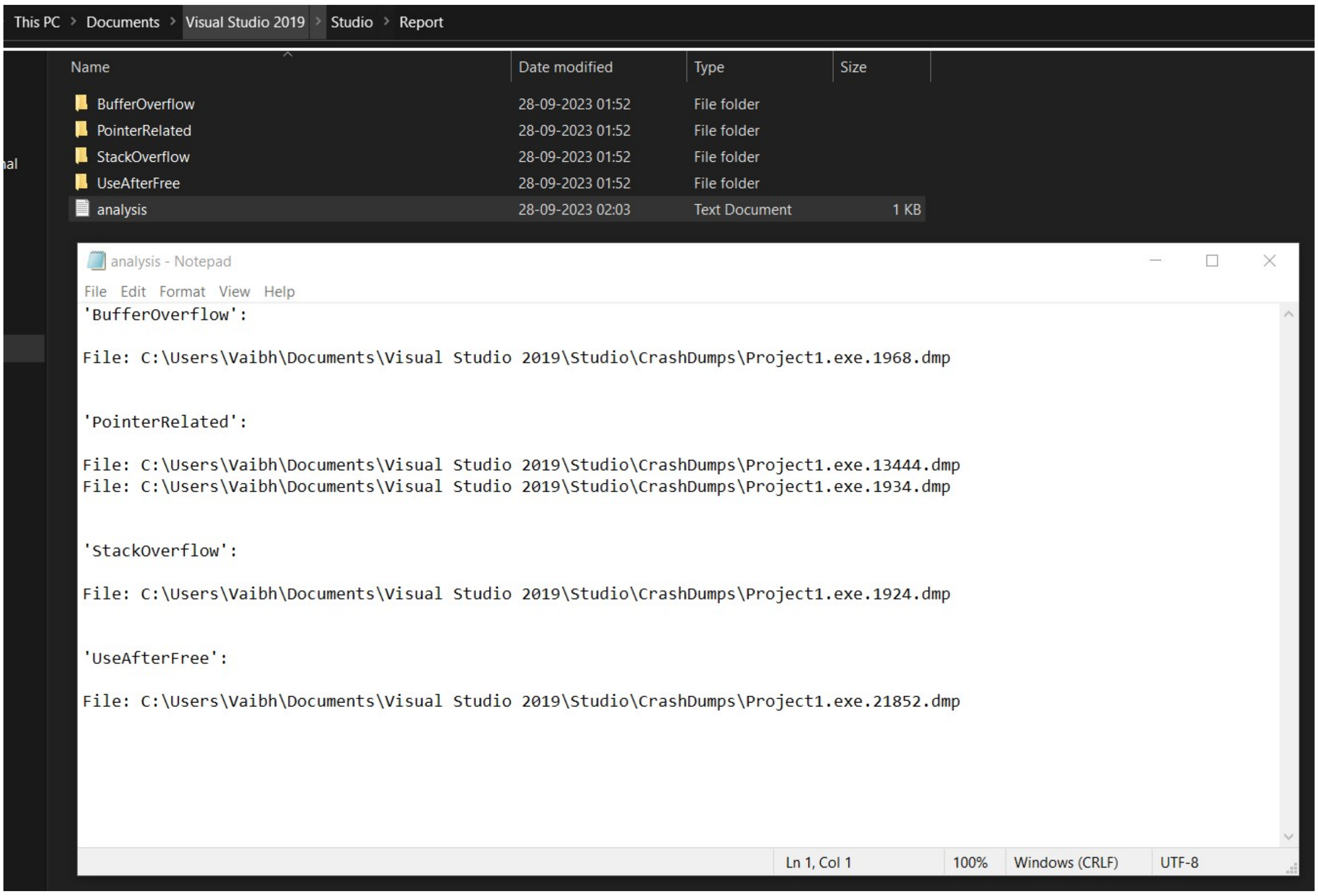
At Output Path:



This PC › Documents › Visual Studio 2019 › Studio › Report

Name	Date modified	Type	Size
BufferOverflow	28-09-2023 01:52	File folder	
PointerRelated	28-09-2023 01:52	File folder	
StackOverflow	28-09-2023 01:52	File folder	
UseAfterFree	28-09-2023 01:52	File folder	
analysis	28-09-2023 02:03	Text Document	1 KB

Inside Analysis:


This PC > Documents > Visual Studio 2019 > Studio > Report

Name	Date modified	Type	Size
BufferOverflow	28-09-2023 01:52	File folder	
PointerRelated	28-09-2023 01:52	File folder	
StackOverflow	28-09-2023 01:52	File folder	
UseAfterFree	28-09-2023 01:52	File folder	
analysis	28-09-2023 02:03	Text Document	1 KB

analysis - Notepad
File Edit Format View Help

```
'BufferOverflow':

File: C:\Users\Vaibh\Documents\Visual Studio 2019\Studio\CrashDumps\Project1.exe.1968.dmp


'PointerRelated':

File: C:\Users\Vaibh\Documents\Visual Studio 2019\Studio\CrashDumps\Project1.exe.13444.dmp
File: C:\Users\Vaibh\Documents\Visual Studio 2019\Studio\CrashDumps\Project1.exe.1934.dmp


'StackOverflow':

File: C:\Users\Vaibh\Documents\Visual Studio 2019\Studio\CrashDumps\Project1.exe.1924.dmp


'UseAfterFree':

File: C:\Users\Vaibh\Documents\Visual Studio 2019\Studio\CrashDumps\Project1.exe.21852.dmp
```

8.CONCLUSION

The Presented System provides a robust and automated solution for the analysis of crash dump files generated by software applications. The purpose of this System's script is to process these crash dump files, extract critical information about the exceptions, and organize the results for further investigation and analysis.

The key features and outcomes of this script can be summarized as follows:

1. Automated Analysis: The script automates the analysis of crash dump files using the Windows Debugger (WinDbg) tool. It iterates through a specified folder containing dump files, analyzes each file, and generates detailed analysis reports.

2. Exception Classification: It identifies and classifies exceptions based on the content of the analysis reports. The script distinguishes between various types of exceptions such as Stack Overflow, Use-After-Free, Pointer Related, and Buffer Overflow.

3. Organized Output: The script generates analysis reports with file names indicating the exception type, making it easier to identify the nature of the issue. These reports are organized into respective folders, ensuring a structured and accessible storage of results.

4. Summary Report: A summary report, named 'analysis.txt,' is created to provide an overview of the analyzed dump files. It lists each exception type and the associated dump file paths for further reference.

5. Error Handling: The script incorporates error handling mechanisms to address any issues that may arise during the analysis process, ensuring the reliability of the tool.

6. User-Friendly: The script is designed to be user-friendly, with clear print statements indicating the progress of the analysis. Users can easily track the analysis of each dump file and access the final results.

In summary, this script offers an effective solution for automating the analysis of crash dump files, enabling software developers and analysts to efficiently investigate and address software issues. Its organized output structure and exception classification provide valuable insights into the root causes of crashes, ultimately contributing to the improvement of software quality and reliability.

9.FUTURE SCOPE

While the current implementation of the code effectively fulfills its intended purpose, there are several areas where future enhancements could be considered to further improve its functionality, maintainability, and efficiency:

a. **User - Friendly Interface:**
Enhance the user interface to provide a more user-friendly experience. Consider building a graphical user interface (GUI) or a web-based front-end to facilitate easier interaction with the code.

b. **Customizable Exception Handling:**
Allow users to define custom exception types and actions based on specific criteria. This flexibility can make the code more adaptable to different use cases.

c. **Integration with Monitoring Systems:**
Enable integration with system monitoring and alerting tools to provide real-time notifications when exceptions occur. This can help administrators respond promptly to critical issues.

d. **Distributed Processing:**
Enhance the code to support distributed processing, allowing the analysis of multiple dump files simultaneously. This can significantly improve scalability and performance.

e. **Logging and Reporting:**
Implement comprehensive logging and reporting capabilities, including detailed information about analyzed dump files, detected exceptions, and actions taken. This can assist in auditing and troubleshooting.

f. **Security Enhancements:**
Strengthen security measures, such as implementing user authentication and access controls, to protect against unauthorized access to sensitive data and functions.

g. **Continuous Integration and Deployment (CI/CD):**
Implement CI/CD pipelines to automate testing and deployment processes. This ensures that updates and enhancements are delivered to production seamlessly.

h. **Cross-Platform Compatibility:**
Make the code cross-platform compatible, allowing it to run on different operating systems. This can broaden its applicability and user base.

i. **Machine Learning Integration:**

Explore the integration of machine learning models to predict and prevent exceptions based on historical data. This can proactively address potential issues.

j. **Community Collaboration:**

Encourage collaboration with the developer community by open-sourcing the code. This can lead to valuable contributions and enhancements from a wider range of experts.

k. **Performance Optimization:**

Continuously optimize the code for performance, especially when handling a large number of dump files. Implement caching and parallel processing techniques.

l. **Compliance and Standards:**

Ensure that the code complies with relevant industry standards and regulations, especially in contexts where it handles sensitive or regulated data.

10. References

[1] https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

[2] https://helpx.adobe.com/in/xd/kb/how-to-generate-crash-dump-on-windows-machine.html.

[3] https://www.youtube.com/playlist?list=PLhx7-txsG6t6n_E2LgDGqgvJtCHPL7UFu

[4] https://www.youtube.com/watch?v=ZGH9vk-gPxM

[5] https://techcommunity.microsoft.com/t5/ask-the-performance-team/understanding-crash-dump-files/ba-p/372633

To Table of Contents