RV Educational Institutions ®
RV College of Engineering ®

Autonomous Institution
Affiliated to Visvesvaraya Technological University, Belagavi

Approved by AICTE, New Delhi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS - CS235AI

Memory Scanner and Abnormality Detection

Submitted by

Vaibhav Soin	**1RV22CS221**
Sohan Varier	**1RV22CS200**

Computer Science and Engineering
2023-2024

INDEX

SYNOPSIS

Memory Scanner & Anomaly Detection is a tool engineered for scrutinizing the memory of Windows processes. Its primary objective is to unearth anomalies and irregularities that could signify security breaches, software bugs, or system instability.

The development of this project was rooted in a thorough understanding of Windows memory management and security principles. We did research to dissect memory management processes and identify potential vulnerabilities. Leveraging this knowledge, we used algorithms for memory scanning and anomaly detection, refining them to detect irregularities indicative of security breaches or software glitches. Through collaboration and testing, we ensured the tool's reliability and effectiveness, culminating in the creation of a robust solution for safeguarding Windows systems.

Looking ahead, "Memory Scanner & Anomaly Detection" holds significant potential for further enhancement. We envision integrating advanced technologies like machine learning to bolster anomaly detection capabilities and expand support for monitoring network activity and system processes. Additionally, adapting the tool to emerging technologies such as virtualization and cloud computing will enable it to adapt to evolving computing paradigms, ensuring continued effectiveness in fortifying system security. Moreover, we plan to explore additional anomalies and expand our code to provide comprehensive protection against a wider range of threats.

INTRODUCTION

Memory abnormalities can lead to system crashes, data corruption, and other critical issues, making it essential to identify and address them promptly. The memory scanner should be able to analyse the system's memory and identify potential problems such as memory leaks, buffer overflows, invalid memory accesses, and other memory-related vulnerabilities. This project is split into two parts:

PART 1:
Creating a memory scanner which can scan through an entire process and return values stored in each data block. Using this, we can locate an memory block in the process and modify its value. Furthermore, we can also scan for memory blocks with values greater or lesser than a reference value and dynamically update the list. This technique is commonly used in game cheat engines and it is imperative that we stop this from happening to our systems.

PART 2:
The second part of our project involves using the memory scanner built in part one to our advantage. We scan a process for anomalies and return warnings if any are found. We have extended to scanner to detect 3 anomalies as of now. They are:
1)Code Injection: Unauthorized insertion of malicious code into a process's memory space.
2)Buffer Overflow: Occurs when data exceeds the capacity of a buffer, potentially leading to memory corruption and exploitation.
3)DLL Injection: Technique used to force a process to load and execute malicious dynamic-link libraries (DLLs) into its address space.
Our scanner can detect these 3 anomalies and will give a warning if one of these also are detected.

SYSTEM ARCHITECTURE

The Windows system architecture is the foundation for the "Memory Scanner & Anomaly Detection" project, providing the infrastructure for memory management, process/thread management, system calls, and security subsystems. This project interacts with various components of the Windows architecture:

1. Kernel Mode and User Mode: Windows operates in two modes: kernel mode and user mode. Memory scanning and anomaly detection involve interactions with both modes to access and analyse memory contents securely.

2. Memory Management: Windows employs virtual memory management and protection mechanisms to allocate, manage, and protect memory resources. The project utilizes Windows memory management APIs to access and analyse memory contents effectively.

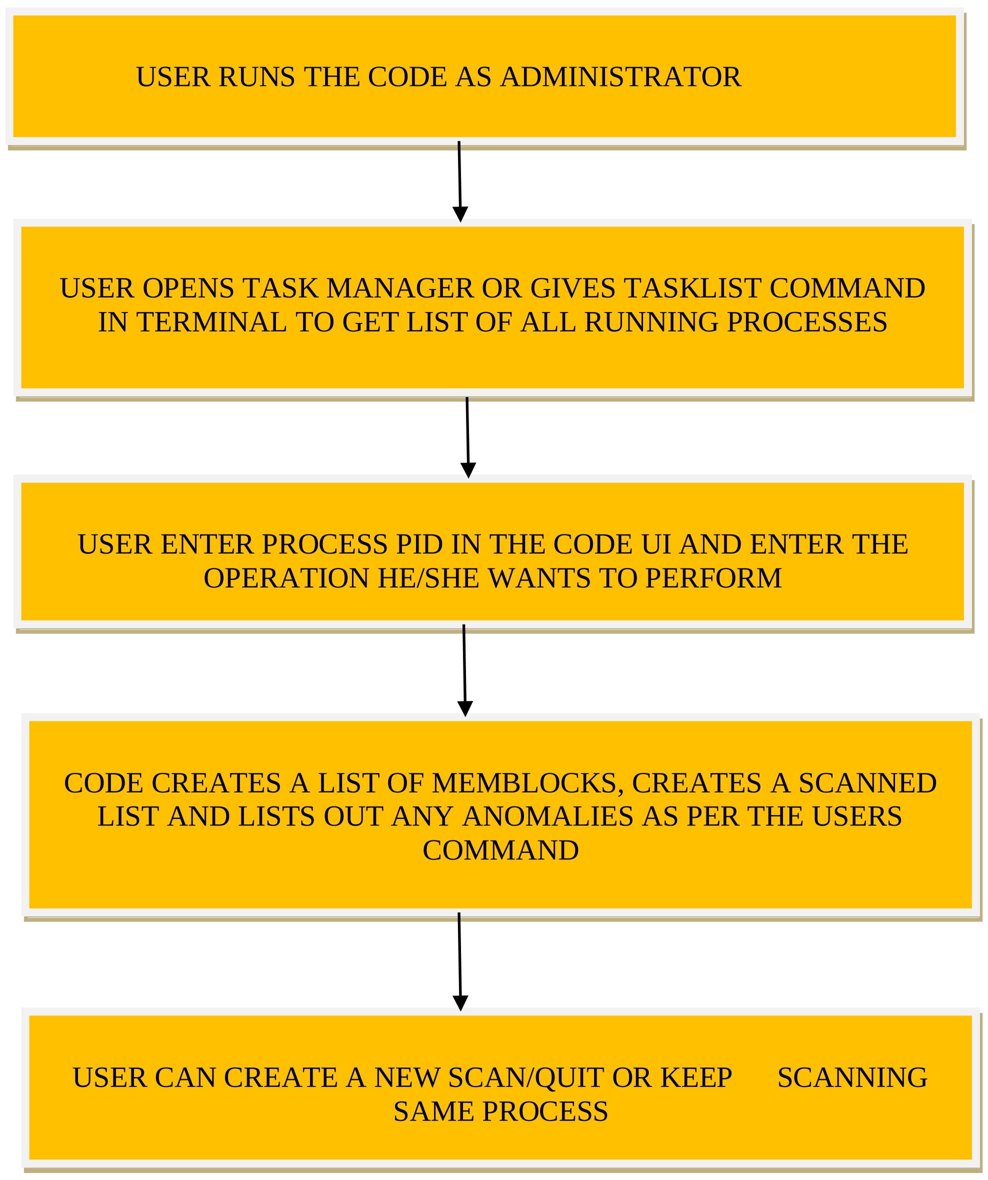
3. Process and Thread Management: Windows organizes tasks into processes and threads, each with its own virtual address space. The project interacts with processes and threads to scan memory regions, detect anomalies, and mitigate security threats.

4. System Calls and APIs: Windows provides a comprehensive set of system calls and APIs for interacting with system components and functionalities. The project utilizes these APIs to access system information, perform memory scans, and implement anomaly detection algorithms.

5. Security Subsystems: Windows incorporates security features like access control lists (ACLs), user account control (UAC), and Windows Defender. Integration with these subsystems enhances the project's ability to detect and mitigate security threats effectively.

By leveraging the features of the Windows system architecture, the project aims to provide users with a powerful tool for safeguarding their systems against memory-based attacks and anomalies.

METHODOLOGY



USER RUNS THE CODE AS ADMINISTRATOR

USER OPENS TASK MANAGER OR GIVES TASKLIST COMMAND IN TERMINAL TO GET LIST OF ALL RUNNING PROCESSES

USER ENTER PROCESS PID IN THE CODE UI AND ENTER THE OPERATION HE/SHE WANTS TO PERFORM

CODE CREATES A LIST OF MEMBLOCKS, CREATES A SCANNED LIST AND LISTS OUT ANY ANOMALIES AS PER THE USERS COMMAND

USER CAN CREATE A NEW SCAN/QUIT OR KEEP SCANNING SAME PROCESS

SYSTEM CALLS USED

The "Memory Scanner & Anomaly Detection" project on Windows utilizes several system calls and APIs to interact with the operating system and access system resources effectively. Some of the key system calls and APIs used in the project include:

1)OpenProcess: This system call is used to open a handle to a process, allowing the project to access and manipulate the memory of the specified process. It takes parameters such as the process identifier (PID) and desired access rights.

2)VirtualQueryEx: This API retrieves information about a range of memory addresses within the address space of a specified process. It provides details such as the base address, size, state, and protection attributes of memory regions.

3)ReadProcessMemory: This API is used to read data from the memory of a specified process. It allows the project to inspect the contents of memory regions within a target process, which is essential for memory scanning and anomaly detection.

4)WriteProcessMemory: This API enables the project to write data to the memory of a specified process. It is used for tasks such as injecting code into a process or modifying memory contents during anomaly detection.

5)VirtualProtectEx: This API changes the protection attributes of a region of memory within the address space of a specified process. It is often used to modify memory protection settings to make memory regions writable or executable.

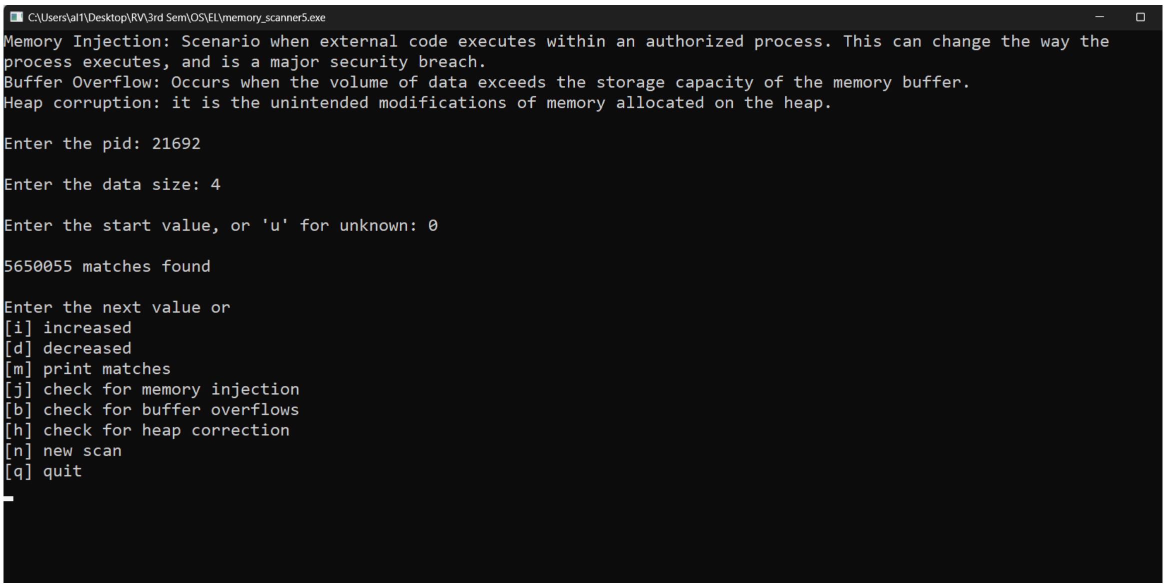
6)CreateToolhelp32Snapshot: This API creates a snapshot of the processes, threads, and modules running on the system. It allows the project to enumerate processes and gather information about them, which is useful for process identification and memory scanning.

7)Module32First/Module32Next: These APIs are used to iterate through the modules (DLLs) loaded into a process. They provide information about each module, including its base address, size, and file path. This information can be valuable for identifying loaded modules during anomaly detection.

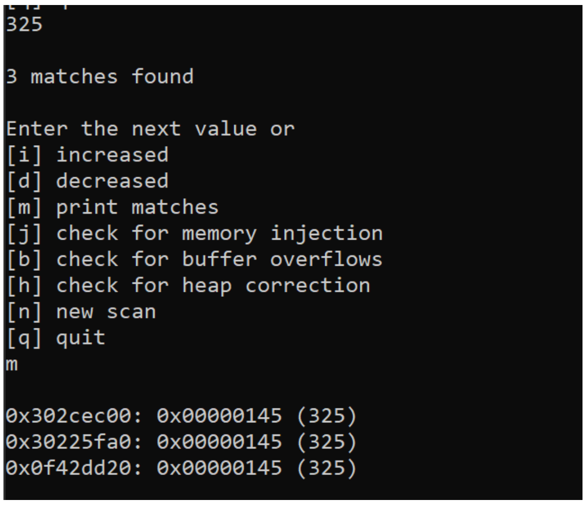
8)CloseHandle: This system call is used to close an open handle to a process or other system object. It ensures proper cleanup and resource management after interacting with system resources.

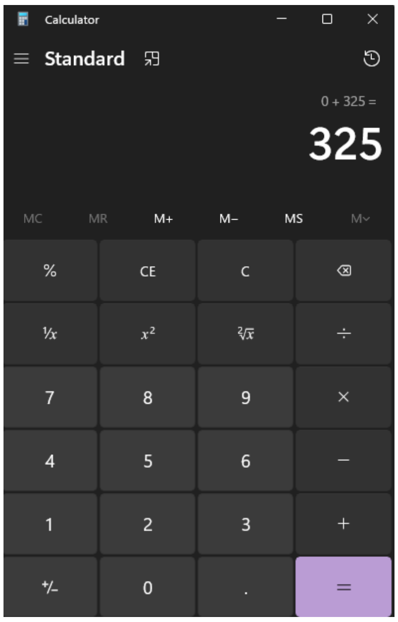
By leveraging these system calls and APIs, the "Memory Scanner & Anomaly Detection" project can effectively scan memory, detect anomalies, and interact with processes running on the Windows operating system, enhancing system security and reliability.

OUPUT/RESULTS

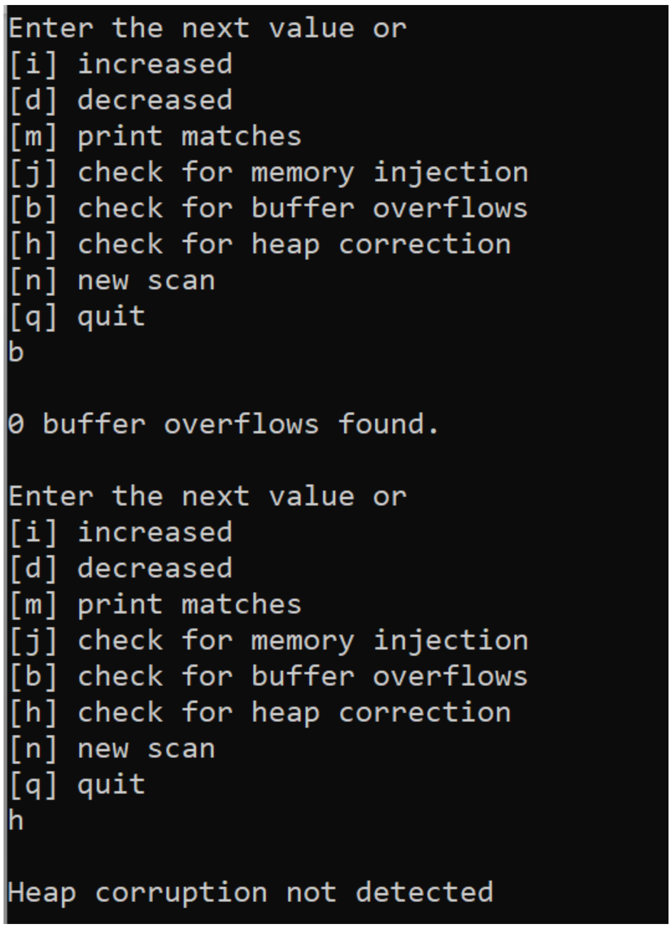

```
C:\Users\al1\Desktop\RV\3rd Sem\OS\EL\memory_scanner5.exe
Memory Injection: Scenario when external code executes within an authorized process. This can change the way the
process executes, and is a major security breach.
Buffer Overflow: Occurs when the volume of data exceeds the storage capacity of the memory buffer.
Heap corruption: it is the unintended modifications of memory allocated on the heap.

Enter the pid: 21692

Enter the data size: 4

Enter the start value, or 'u' for unknown: 0

5650055 matches found

Enter the next value or
[i] increased
[d] decreased
[m] print matches
[j] check for memory injection
[b] check for buffer overflows
[h] check for heap correction
[n] new scan
[q] quit
```

Output showing all the matches for given value of memory


```
325

3 matches found

Enter the next value or
[i] increased
[d] decreased
[m] print matches
[j] check for memory injection
[b] check for buffer overflows
[h] check for heap correction
[n] new scan
[q] quit
m

0x302cec00: 0x00000145 (325)
0x30225fa0: 0x00000145 (325)
0x0f42dd20: 0x00000145 (325)
```

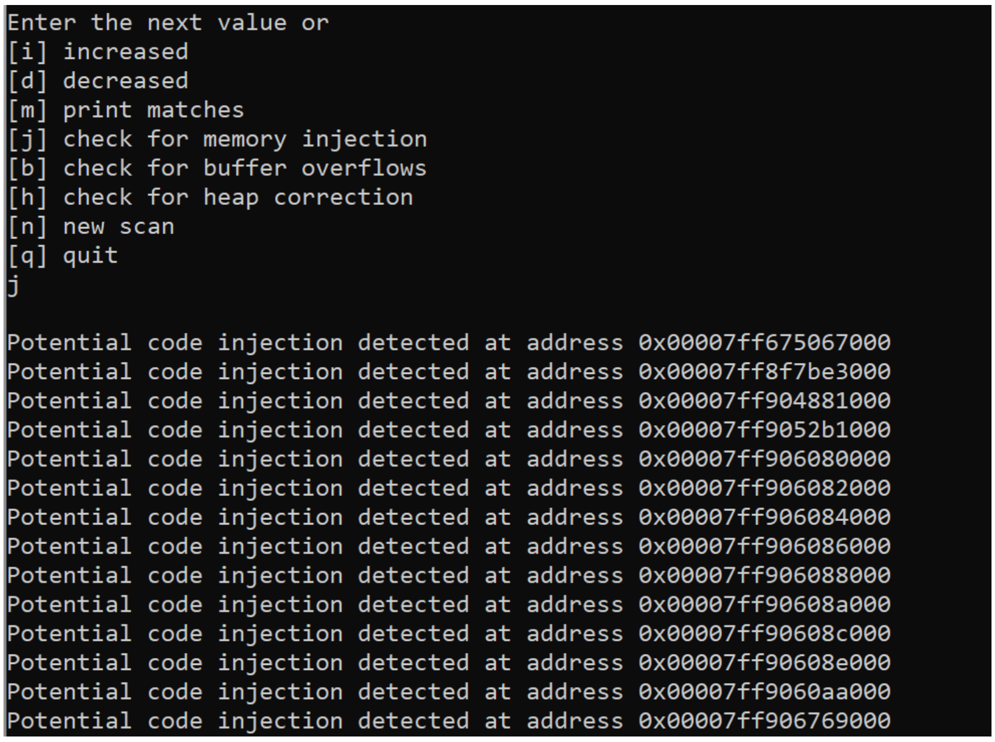


Output after finding narrowing down the memory block based on the value at that memory



```
Enter the next value or
[i] increased
[d] decreased
[m] print matches
[j] check for memory injection
[b] check for buffer overflows
[h] check for heap correction
[n] new scan
[q] quit
b

0 buffer overflows found.

Enter the next value or
[i] increased
[d] decreased
[m] print matches
[j] check for memory injection
[b] check for buffer overflows
[h] check for heap correction
[n] new scan
[q] quit
h

Heap corruption not detected
```

Checking for Buffer Overflows and Heap Corruption



```
Enter the next value or
[i] increased
[d] decreased
[m] print matches
[j] check for memory injection
[b] check for buffer overflows
[h] check for heap correction
[n] new scan
[q] quit
j

Potential code injection detected at address 0x00007ff675067000
Potential code injection detected at address 0x00007ff8f7be3000
Potential code injection detected at address 0x00007ff904881000
Potential code injection detected at address 0x00007ff9052b1000
Potential code injection detected at address 0x00007ff906080000
Potential code injection detected at address 0x00007ff906082000
Potential code injection detected at address 0x00007ff906084000
Potential code injection detected at address 0x00007ff906086000
Potential code injection detected at address 0x00007ff906088000
Potential code injection detected at address 0x00007ff90608a000
Potential code injection detected at address 0x00007ff90608c000
Potential code injection detected at address 0x00007ff90608e000
Potential code injection detected at address 0x00007ff9060aa000
Potential code injection detected at address 0x00007ff906769000
```

Checking for code injections

CONCLUSION

In conclusion, the "Memory Scanner & Anomaly Detection" project on Windows presents a robust solution for monitoring and analyzing memory activity in real-time. By utilizing a combination of system calls and APIs, the project can effectively scan memory regions, detect anomalies, and take appropriate actions to maintain system integrity and security.

Through the implementation of memory scanning algorithms and anomaly detection techniques, the project provides valuable insights into potential threats such as code injection, buffer overflow, and DLL injection. By detecting these anomalies early, the project helps prevent malicious activities and ensures the stability and reliability of the system.

Moreover, the project's modular and extensible architecture allows for further enhancements and customization to address evolving security threats. Future iterations of the project could include additional anomaly detection algorithms, support for detecting other types of memory-based attacks, and integration with security monitoring systems for proactive threat mitigation.

Overall, the "Memory Scanner & Anomaly Detection" project serves as a valuable tool for system administrators, security analysts, and developers in identifying and mitigating memory-related vulnerabilities, thereby enhancing the overall security posture of Windows-based systems.