EX NO - 10 : Use Ghidra to disassemble and analyze the malware code.

AIM :

To perform static and basic dynamic analysis of a benign sample binary using Linux command -line tools (file, strings, readelf, objdump, strace, itrace, Isof) and document observable behaviours such as file I/0, dynamic symbol resolution, and network activity (if any).

PROCEDURE:

The project provides a hands-on guide for malware analysis using Ghidra, teaching user to dissect binaries, understand assembly and high -level behaviors, and identify malicious functionalities. It includes step-by-step tutorials, automation scripts, safe sample

Tools Used:

1. Ghidra (latest version)
2. Java Development Kit (JDK)
3. Virtualized environment (for safety)
4. Sample malware (in a controlled environment)
5. Windows/Linux operating system

Prerequisites:

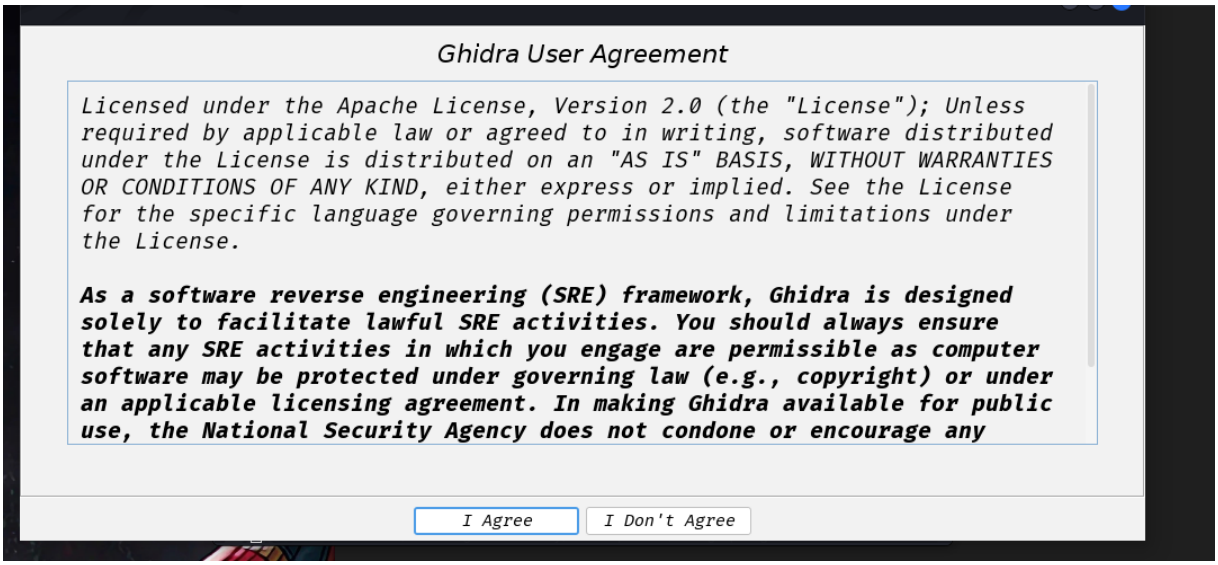
1. Isolated analysis environment (Virtual Machine)
2. Java Development Kit installed
3. Ghidra installed and configured
4. Test malware samples (safely contained)
5. Basic understanding of assembly language

Procedure:

1. Environment Setup

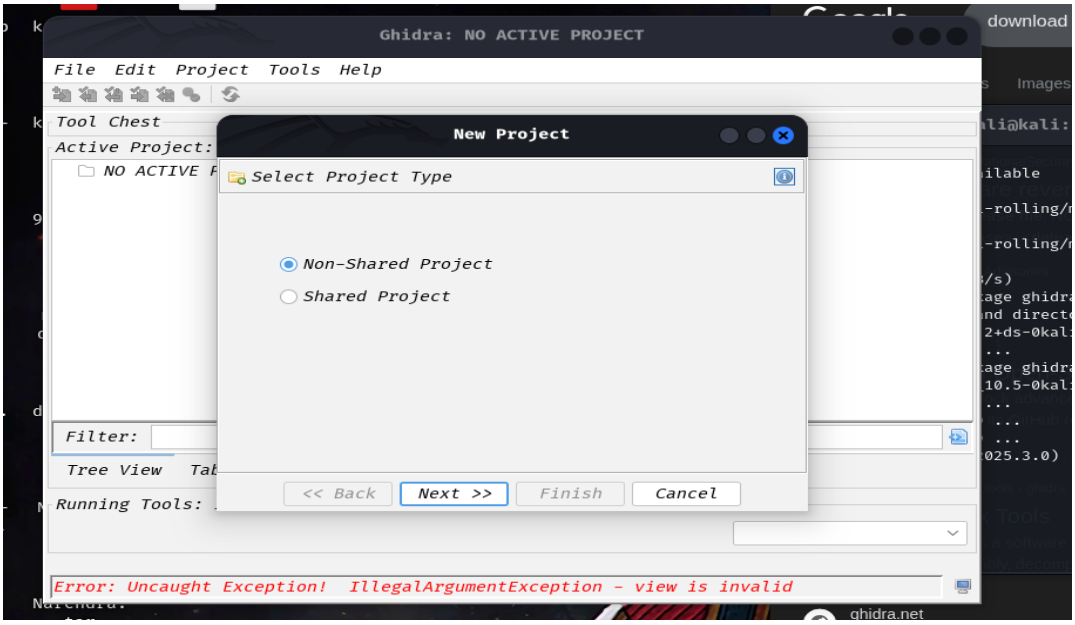
1. Install Java Development Kit

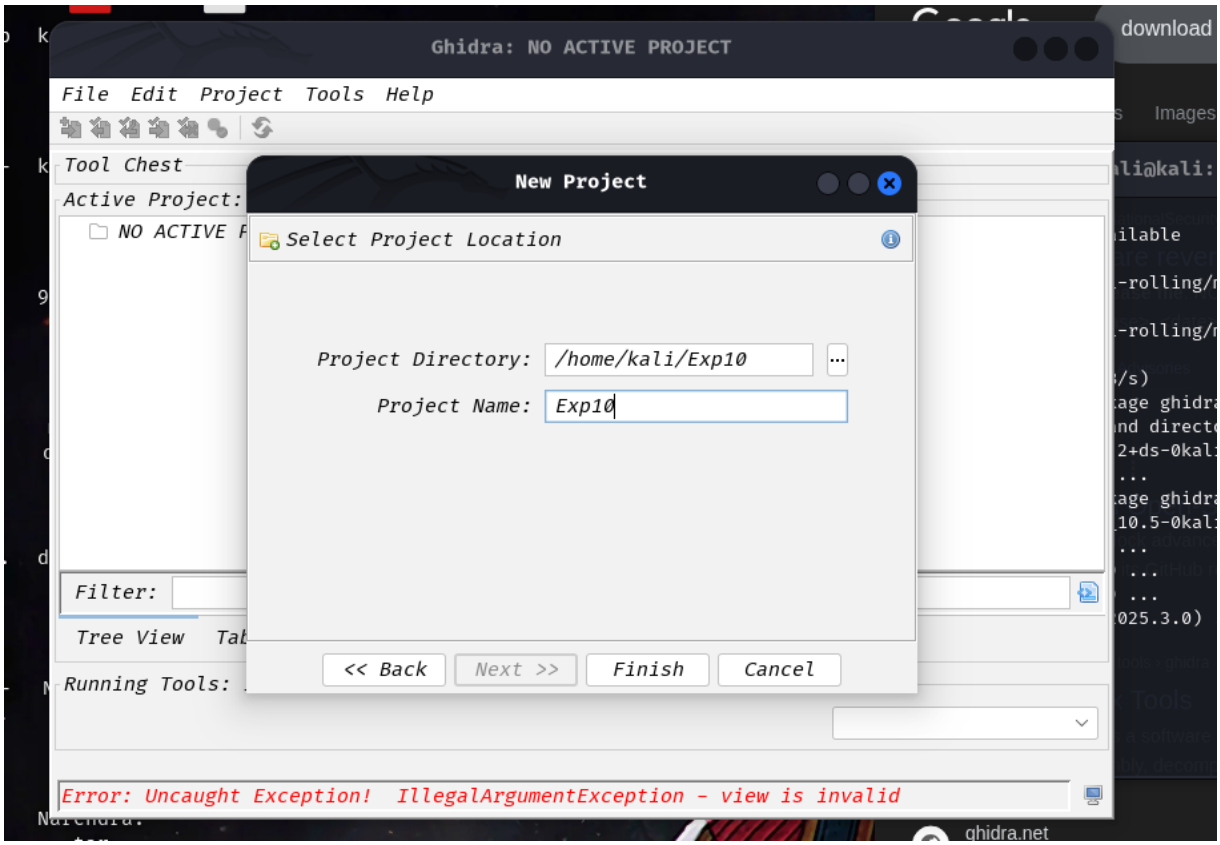
2. Download and install Ghidra
3. Configure isolated analysis environment
4. Verify Ghidra launches successfully



2. Project Creation

1. Create new project:
    - File → New Project
    - Select Non-Shared Project
    - Choose project location





2. Import malware sample:
    - File → Import File
    - Select malware sample
    - Configure import options

---

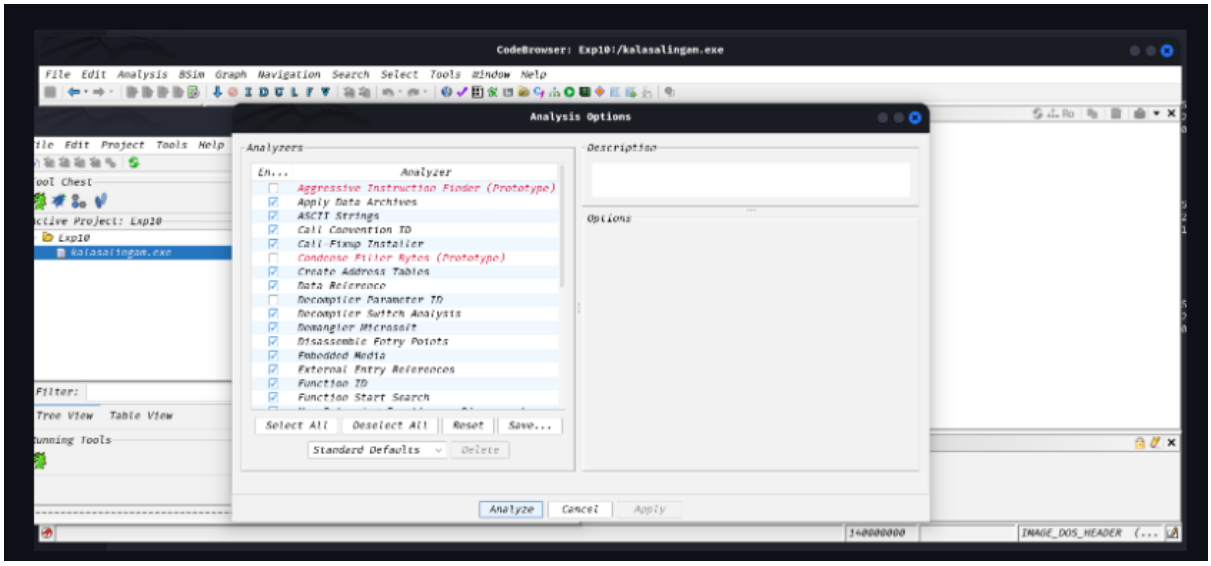
3. Initial Analysis

1. Double-click imported file to analyze
2. Wait for auto-analysis to complete
3. Review initial analysis results:
    - Functions identified
    - Strings detected
    - Entry points located

---

4. Code Analysis

1. Navigate through Program Trees:
    - Functions
    - Labels
    - Data Types



2. Examine Decompiled Code:
    - Use Decompiler window
    - Analyze function logic
    - Identify suspicious operations

---

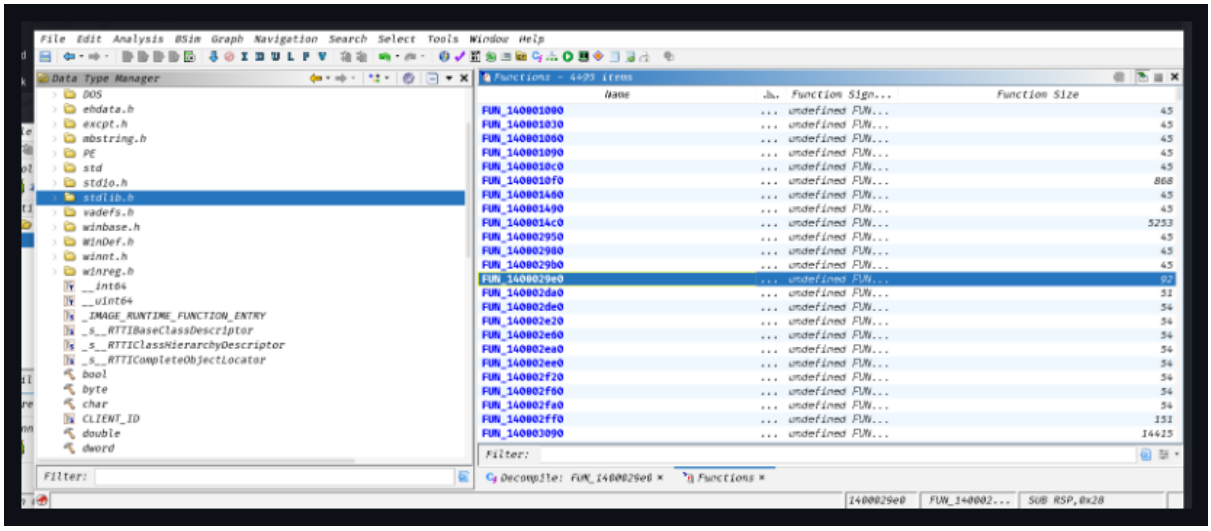
5. Function Analysis

1.Identify main functions:

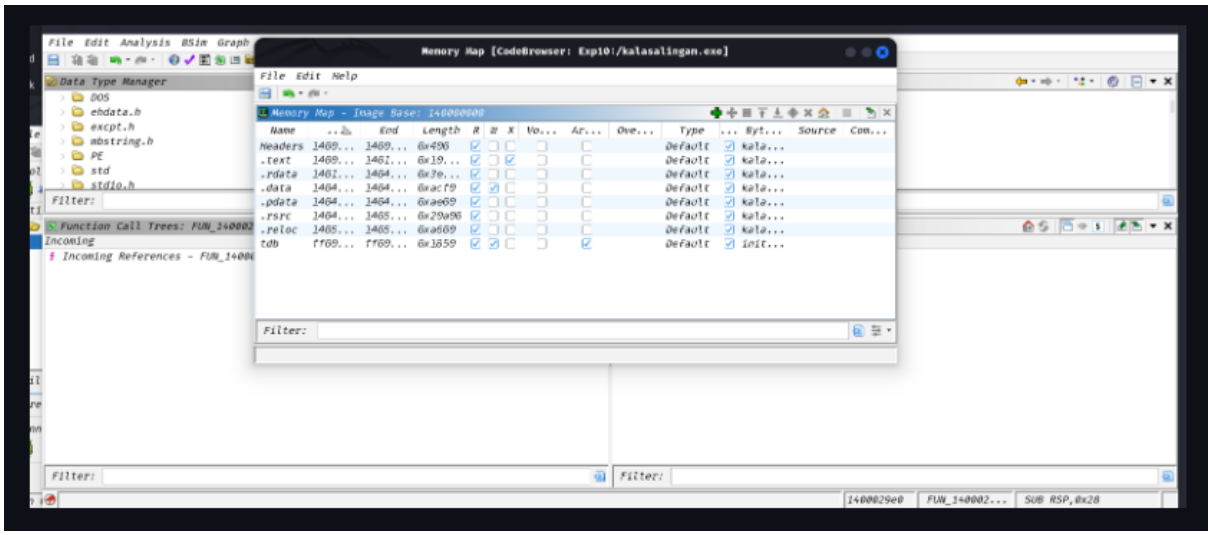
. Entry points

. Important API calls

. String references

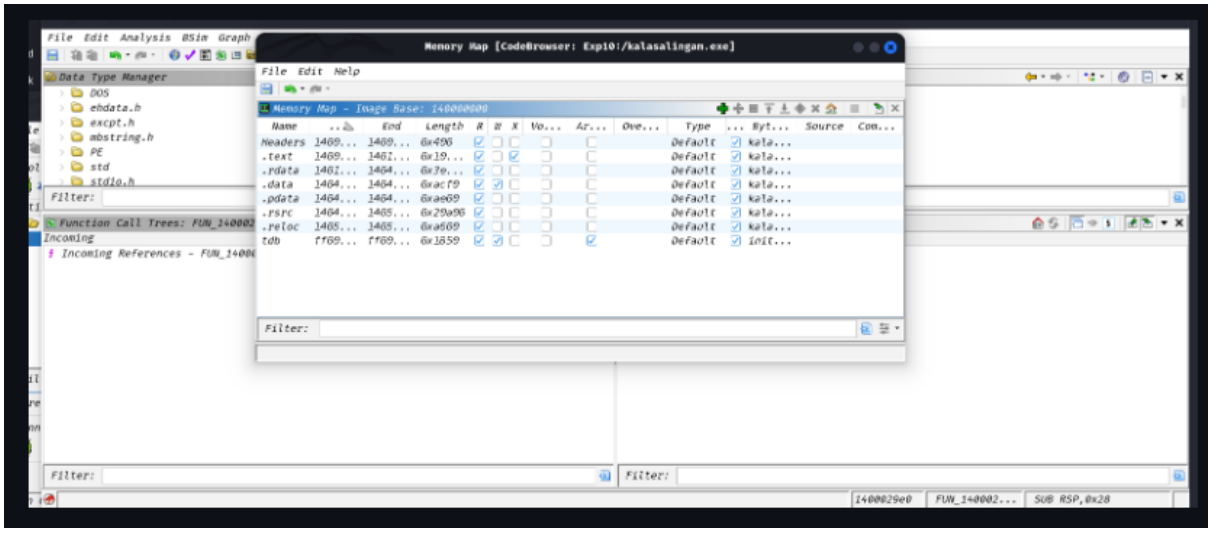


2. Analyze suspicious functions:
    - Network operations
    - File system operations
    - Registry modifications



6. String Analysis:

1. Examine string references:
    - URLs
    - File paths
    - Registry keys
    - Command strings



---

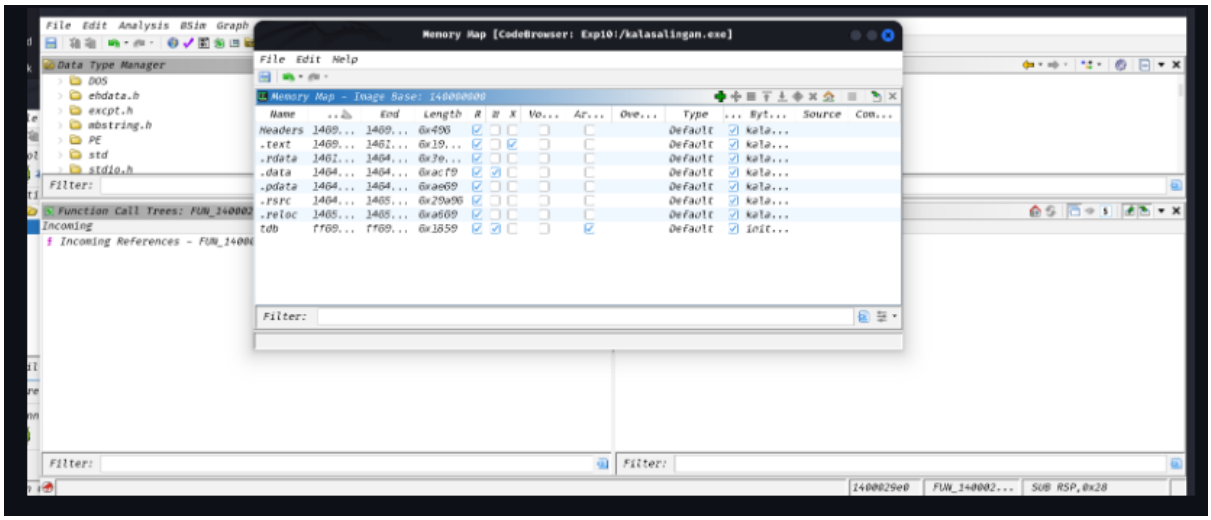
7. Cross-Reference Analysis

1. Track function calls:
    - Who calls this function
    - What this function calls

---

8. Memory Analysis

1. Review memory layout:
    - Sections
    - Segments
    - Permissions



Results:

The experiment successfully demonstrated:

1. Code Analysis Results:
    - Total functions analyzed:[☑]

    - Suspicious functions identified: [X]
    - Malicious behaviors detected: [X]
2. Malware Capabilities Identified:
    - Network communication methods
    - File system interactions
    - System modifications
    - Persistence mechanisms
3. Key Findings:
    - Primary malware functionality
    - Evasion techniques used
    - System impact assessment
    - Network indicators

Conclusion:

Ghidra proved to be a powerful tool for malware analysis and reverse engineering.
Key achievements:

1. Successfully disassembled and decompiled malware code
2. Identified malicious functions and system interactions
3. Extracted indicators of compromise for detection
4. Understood malware behavior and persistence mechanisms

Important: Always perform malware analysis in isolated virtual environments to prevent system compromise.
The tool is highly recommended for digital forensics professionals conducting malware investigation and threat analysis.
Best Practices Identified:

1. Systematic analysis approach
2. Documentation of findings
3. Safe handling procedures
4. Indicator extraction