Lab: Multi-Format Malware Analysis
Static Analysis Across PE, ELF, APK, and Go Binaries

Ahmed Lekssays

January 6, 2026

Introduction

This lab provides hands-on experience in static analysis of malware across different executable formats: Windows PE files, Linux ELF binaries, Android APK packages, and cross-platform Go malware. You will learn to identify format-specific indicators, detect packing/obfuscation, and document findings in a structured manner.

Learning Objective

By completing this lab, students will:

- Master static analysis techniques for PE, ELF, APK, and Go formats
- Identify suspicious patterns and indicators of compromise
- Detect packed and obfuscated malware
- Practice using industry-standard analysis tools
- Develop professional malware analysis reporting skills

1 Prerequisites

1.1 Required Tools (All Ubuntu-based)

- **PE Analysis**: pev (readpe, pescan), pefile (Python library), hexdump, strings
- **ELF Analysis**: binutils (readelf, objdump, nm), file, strings
- **APK Analysis**: apktool, jadx
- **Go Analysis**: GoReSym, strings

- **General**: xxd, entropy calculators

1.2 Sample Files

All samples can be downloaded from the link below. Handle with care in isolated environments only.

- **Download URL**: `https://limewire.com/d/Am8H3#T6VStLCz68`
- **Archive Password**: `infected`

Important Note

CRITICAL SAFETY REQUIREMENTS:

- All analysis MUST be performed in isolated VMs with no network access
- Never execute samples on your host machine
- Use snapshots before analyzing each sample
- Verify file hashes before analysis

2 Part 1: Windows PE Analysis (25 points)

2.1 Sample Information

- **Sample ID**: PE-001
- **Filename**: `d551a474ecf0b7d1ba6dda319e3b77fdecf39489eaf14b7d8837002f2d31387b.exe`
- **SHA256**: `d551a474ecf0b7d1ba6dda319e3b77fdecf39489eaf14b7d8837002f2d31387b`
- **File Type**: Windows PE32 Executable

2.2 Deliverable 1.1: PE Header Analysis (8 points)

Analyze the PE header and provide your findings in a structured format.

Required Analysis:

1. Document basic file properties (size, hashes, machine type, subsystem)
2. Analyze the compilation timestamp and assess its validity (2 points)
    - Is the timestamp recent, old, or suspicious?

- Could it be forged? What would that indicate?

3. Examine the Entry Point location (3 points)
   - What section does the Entry Point RVA point to?
   - Is this typical or suspicious? Explain why.
   - What would it mean if the entry point was in .data or .rsrc?
4. Analyze ImageBase and DLL Characteristics (3 points)
   - Is the ImageBase standard or unusual?
   - What security features are enabled (ASLR, DEP, etc.)?
   - What do these settings tell you about the binary?

2.3 Deliverable 1.2: Section Analysis and Packing Detection (10 points)

Perform comprehensive section analysis and determine if the binary is packed.

Required Analysis:

1. Create a detailed section table including: name, virtual size, raw size, permissions, and entropy
2. Analyze each section for suspicious characteristics (5 points)
   - Which sections have high entropy (> 7.0)? What does this indicate?
   - Are there any sections with unusual permission combinations (e.g., WRITE+EXECUTE)?
   - Compare VirtualSize vs RawSize for each section. Identify and explain any significant discrepancies.
   - Are there sections with non-standard names? What might this mean?
3. Packing Assessment (5 points)
   - Based on entropy analysis, is this binary likely packed? Provide evidence.
   - What specific indicators support your conclusion?
   - If packed, can you identify the packer used? How?
   - What would be your next steps to analyze this binary further?

2.4 Deliverable 1.3: Import Analysis and Capability Assessment (7 points)

Analyze the import table to understand the malware's capabilities.

Required Analysis:

1. Identify and categorize suspicious imports (4 points)
    - Group imports by capability (networking, file operations, registry, process manipulation, etc.)
    - For each category, explain what the API functions suggest about malware behavior
    - Highlight any particularly dangerous or unusual API combinations
2. Malware Classification (3 points)
    - Based on the import analysis, what type of malware is this likely to be?
    - What specific capabilities does it possess?
    - Are there any imports that suggest anti-analysis techniques?
    - What is your confidence level in this assessment and why?

3 Part 2: Linux ELF Analysis (25 points)

3.1 Sample Information

- **Sample ID**: ELF-001
- **Filename**: `981907e3f5ed07062b33b3e992d1f3412a2f3352208e92c1b58ff3c2387d50ae.elf`
- **SHA256**: `981907e3f5ed07062b33b3e992d1f3412a2f3352208e92c1b58ff3c2387d50ae`
- **File Type**: ELF 64-bit LSB executable

3.2 Deliverable 2.1: ELF Header and Format Analysis (8 points)

Analyze the ELF header and assess the binary's characteristics.

Required Analysis:

1. Document basic ELF properties (class, encoding, OS/ABI, machine architecture, entry point)
2. Binary Characteristics Assessment (4 points)
    - Is the binary stripped of symbols? What are the implications for analysis?

- Is this a Position Independent Executable (PIE)? How can you tell and why does it matter?
- Analyze the program headers - which segments are present and what are their permissions?

3. Security Features Analysis (4 points)
   - What security features are present or absent (Stack Canary, NX, PIE, RELRO)?
   - What does the presence/absence of these features tell you?
   - Are there any unusual or suspicious header values?

3.3 Deliverable 2.2: Section Analysis and Code Structure (10 points)

Examine the ELF sections to understand the binary's structure.

Required Analysis:

1. Document all sections with their types, sizes, and permissions
2. Critical Section Analysis (6 points)
   - Analyze .text section: What is its size and entropy? Any suspicious patterns?
   - Examine .data and .rodata: What types of data are stored? Any embedded resources?
   - Investigate .plt and .got: What do these reveal about dynamic linking?
   - Are there any non-standard sections? What might they contain?
3. Structural Assessment (4 points)
   - Compare this structure to typical benign ELF files. What's different?
   - Are there any sections that could hide malicious code or data?
   - Based on section analysis, can you infer anything about how this malware was compiled?

3.4 Deliverable 2.3: Symbol and Function Analysis (7 points)

Analyze symbols, imports, and strings to determine capabilities.

Required Analysis:

1. Symbol Table Analysis (3 points)
   - What functions are imported from shared libraries?
   - Categorize these by capability (networking, file I/O, process operations, etc.)

- Are there any particularly suspicious library functions?

2. String Analysis (2 points)
   - Identify interesting strings (URLs, file paths, commands, error messages)
   - What do these strings reveal about the malware's functionality?
   - Are there any indicators of C2 infrastructure or configuration?
3. Capability Assessment (2 points)
   - Based on symbols and strings, what is this malware designed to do?
   - What operating systems or environments does it target?
   - How sophisticated is this malware?

4 Part 3: Android APK Analysis (25 points)

4.1 Sample Information

- **Sample ID**: APK-001
- **Filename**: `cc93d01b68b59314a789c5355ac70b8e6965b9f64bb331b0337aac9d2da8aede.apk`
- **SHA256**: `cc93d01b68b59314a789c5355ac70b8e6965b9f64bb331b0337aac9d2da8aede`
- **File Type**: Android Application Package

4.2 Deliverable 3.1: Manifest Analysis and Permission Assessment (10 points)

Analyze the AndroidManifest.xml to understand the application's declared capabilities.

Required Analysis:

1. Document basic application information (package name, SDK versions, components)
2. Permission Risk Assessment (6 points)
   - List all requested permissions and categorize each as High/Medium/Low risk
   - For each HIGH risk permission, explain:
     - What capability it grants
     - Why it's dangerous in this context
     - Whether it's justified for the app's claimed functionality
   - Identify any permission combinations that are particularly concerning

- Are there any dangerous permissions requested without clear justification?

3. Component Analysis (4 points)
    - What Activities, Services, and Receivers are declared?
    - Analyze the intent filters - what can trigger this app?
    - Are there any exported components? Why is this concerning?
    - Based on the manifest, what is this app's stated vs likely actual purpose?

4.3 Deliverable 3.2: DEX Code Analysis (10 points)

Decompile and analyze the application's Java/Kotlin code using jadx.

Required Analysis:

1. Code Structure Analysis (4 points)
    - Document the main package structure
    - Identify the key classes and their relationships
    - Are there any obfuscated class/method names? What does this suggest?
2. Suspicious Code Identification (6 points)
    - Identify at least 3 suspicious methods/functions
    - For each, provide:
        - The class and method name
        - What the code does (explain the logic)
        - Why this behavior is malicious or suspicious
        - What data it accesses or exfiltrates
    - Include relevant code snippets or screenshots
    - Are there any anti-analysis or evasion techniques in the code?

4.4 Deliverable 3.3: Resource and Native Library Analysis (5 points)

Examine embedded resources and native components.

Required Analysis:

1. Resource Analysis (2 points)
    - Are there any suspicious files in assets/ or res/raw/?
    - Look for embedded executables, scripts, or encrypted data
    - Analyze any suspicious resources - what might they contain?

2. Native Library Analysis (3 points)
    - Are there native libraries in the lib/ folder? For which architectures?
    - Extract and analyze any .so files
    - What functions do they export? Why use native code?
    - Does the use of native libraries suggest an attempt to hide functionality?

5 Part 4: Bonus - Go Malware Analysis (20 bonus points)

5.1 Sample Information

- **Sample ID**: GO-ELF-001
- **Filename**: `ef1ef1954560b13d5c13e2142210d187bcfa9bb86690e0ff8d6de70bf5c8b4f7.elf`
- **SHA256**: `ef1ef1954560b13d5c13e2142210d187bcfa9bb86690e0ff8d6de70bf5c8b4f7`
- **File Type**: Linux ELF executable (Go-compiled)

Bonus Challenge

This is a challenging bonus section worth 20 extra points. Go malware is increasingly common and presents unique analysis challenges due to its large binary size, embedded runtime, and symbol stripping.

5.2 Deliverable 4.1: Go Binary Identification and Characteristics (6 points)

Identify and characterize the Go binary.
Required Analysis:

1. Go Binary Confirmation (2 points)
    - Prove this is a Go binary using specific string patterns
    - Extract the Go version used for compilation
2. Binary Characteristics Analysis (4 points)
    - What is the binary size? How does this compare to typical malware?
    - Why are Go binaries typically so large? What are the implications for analysis?
    - Is this binary cross-compiled? What evidence supports your conclusion?
    - What OS and architecture was this compiled for?

5.3 Deliverable 4.2: Go Runtime and Symbol Analysis (7 points)

Analyze the Go runtime and attempt to recover symbols.
Required Analysis:

1. Symbol Recovery (3 points)
    - Use GoReSym or similar tools to recover function names
    - How many functions were identified? Are symbols stripped?
    - What does the presence/absence of symbols tell you?
2. Go Runtime Analysis (4 points)
    - Identify key Go runtime structures (goroutines, schedulers, garbage collector)
    - How does the Go runtime complicate static analysis?
    - Can you identify any main.* functions? What do their names suggest?
    - Compare this to analyzing C/C++ malware - what's harder? What's easier?

5.4 Deliverable 4.3: Capability Analysis and Classification (7 points)

Determine the malware's functionality despite Go's obfuscation.
Required Analysis:

1. Function Analysis (3 points)
    - Identify main.* functions that suggest malicious behavior
    - What capabilities can you infer from function names?
    - Are there any functions related to networking, encryption, or persistence?
2. String and Capability Analysis (2 points)
    - Extract and analyze interesting strings (URLs, commands, paths)
    - What do these strings reveal about functionality?
    - Can you identify any configuration data or C2 addresses?
3. Malware Classification (2 points)
    - Based on your analysis, what type of malware is this?

- What are its primary capabilities?
- Why did the authors choose Go for this malware?
- How would you continue analyzing this if you had more time?

6 Final Deliverable: Comprehensive Analysis Report (25 points)

Submit a professional malware analysis report (5-8 pages) containing:

6.1 Report Structure and Requirements

6.1.1 Executive Summary (5 points)

- Brief overview of all samples analyzed
- Key findings for each sample (2-3 sentences per sample)
- Overall threat assessment and severity ratings
- Critical recommendations for detection and mitigation

Evaluation Criteria:

- Can a non-technical manager understand the threats?
- Are the key findings clearly prioritized?
- Is the summary concise yet comprehensive?

6.1.2 Technical Analysis (10 points)

- Consolidated findings from Parts 1-3 (or 1-4 if doing bonus)
- For each sample, include:
  - Format-specific analysis results
  - Identified capabilities and behaviors
  - Detection evasion techniques observed
  - Indicators of sophistication
- Cross-format comparison and insights:
  - What similarities exist across different formats?

- What format-specific techniques were used?
- Which sample was most sophisticated and why?

Evaluation Criteria:

- Depth of technical understanding demonstrated
- Quality of comparative analysis
- Clear explanation of findings
- Proper use of technical terminology

6.1.3 Indicators of Compromise (5 points)

Create a comprehensive IOC table for each sample:

Sample	IOC Type	Value/Description
PE-001	File Hash	SHA256: d551...
PE-001	Mutex	Specific mutex name
PE-001	Registry Key	HKLM\Software\...
ELF-001	File Hash	SHA256: 9819...
ELF-001	File Path	/tmp/.hidden_file
APK-001	Package Name	com.malicious.app
APK-001	C2 Domain	malicious-c2.example.com

Include behavioral IOCs, network IOCs, and host-based IOCs.
Evaluation Criteria:

- Completeness of IOC coverage
- Actionability of IOCs (can they be used for detection?)
- Proper categorization and formatting

6.1.4 Methodology and Tools (3 points)

- List all tools used for each analysis phase
- Describe your analysis workflow and decision-making process
- Document challenges encountered and how you overcame them
- Discuss any limitations in your analysis

Evaluation Criteria:

- Is the methodology clearly documented and reproducible?
- Are tool choices justified?
- Does the student demonstrate critical thinking about the analysis process?

6.1.5 Detection and Mitigation Recommendations (2 points)

- Provide specific detection rules (YARA signatures, network signatures, etc.)
- Suggest mitigation strategies for each malware type
- Recommend security controls to prevent similar attacks

Evaluation Criteria:

- Are recommendations practical and actionable?
- Do they address the specific threats identified?

7 Submission Requirements

7.1 Submission Link

Please submit your final work via the following link:
https://forms.gle/FSR7ieCEK8JXpW4KA

7.2 File Organization

Submit a single ZIP file named `LastName__FirstName__Lab2.zip` containing:

```
|-- Report/
|   ‘-- LastName_FirstName_Report.pdf
|-- Part1_PE/
|   |-- analysis_notes.txt
|   |-- section_table.csv or .xlsx
|   |-- import_analysis.txt
|   ‘-- screenshots/
|       |-- header_analysis.png
|       |-- entropy_graph.png
|       ‘-- imports.png
|-- Part2_ELF/
|   |-- analysis_notes.txt
|   |-- section_table.csv or .xlsx
```

```
|   |-- symbols_analysis.txt
|   `-- screenshots/
|-- Part3_APK/
|   |-- manifest_analysis.txt
|   |-- permissions_table.csv or .xlsx
|   |-- code_analysis.txt
|   `-- screenshots/
|       |-- suspicious_method1.png
|       |-- suspicious_method2.png
|       `-- suspicious_method3.png
|-- Part4_BONUS_Go/ (if applicable)
|   |-- identification.txt
|   |-- symbol_recovery.txt
|   |-- capability_analysis.txt
|   `-- screenshots/
`-- README.txt
```

7.3 README.txt Format

Include a README with:

- Your name and student ID
- Date of analysis
- VM/environment used (Ubuntu version, tools installed)
- Verification that all file hashes match provided samples
- Any special notes or issues encountered
- Time spent on each part (optional but helpful)

8 Grading Rubric

8.1 Detailed Grading Criteria

Excellent (90-100%):

- Deep analysis with clear explanations of why findings are significant
- Demonstrates understanding beyond tool output
- Makes insightful connections between different indicators

Component	Points	Focus
Part 1: PE Analysis	25	Analysis depth, packing detection
Part 2: ELF Analysis	25	Symbol interpretation, capability assessment
Part 3: APK Analysis	25	Permission risk, code understanding
Final Report	25	Professional quality, IOCs, recommendations
Total	**100**	
Bonus: Go Analysis	**+20**	Go-specific techniques, advanced analysis

- Professional report with actionable recommendations
- All deliverables complete with thorough documentation
- Critical thinking evident throughout

Good (80-89%):

- Solid analysis with adequate explanations
- Shows understanding of key concepts
- Makes some connections between indicators
- Well-organized report with good recommendations
- Most deliverables complete with good documentation

Satisfactory (70-79%):

- Basic analysis completed but limited depth
- Understanding of concepts but minimal explanation
- Few connections made between findings
- Report needs better organization or recommendations
- Some deliverables incomplete or superficial

Needs Improvement (¡70%):

- Significant gaps in analysis or understanding
- Mostly tool output without interpretation
- No connections between findings
- Poor report quality or missing sections
- Multiple deliverables incomplete or incorrect