

The logo graphic for the University of Guelph, featuring a stylized landscape with blue and white geometric shapes representing hills and a field.

UNIVERSITY OF GUELPH

# REPORT

Submission 3

## GROUP Members

LIONEL IRAKOZE

EDWARD HUA

LAWRENCE GBATSOMA

# Contents

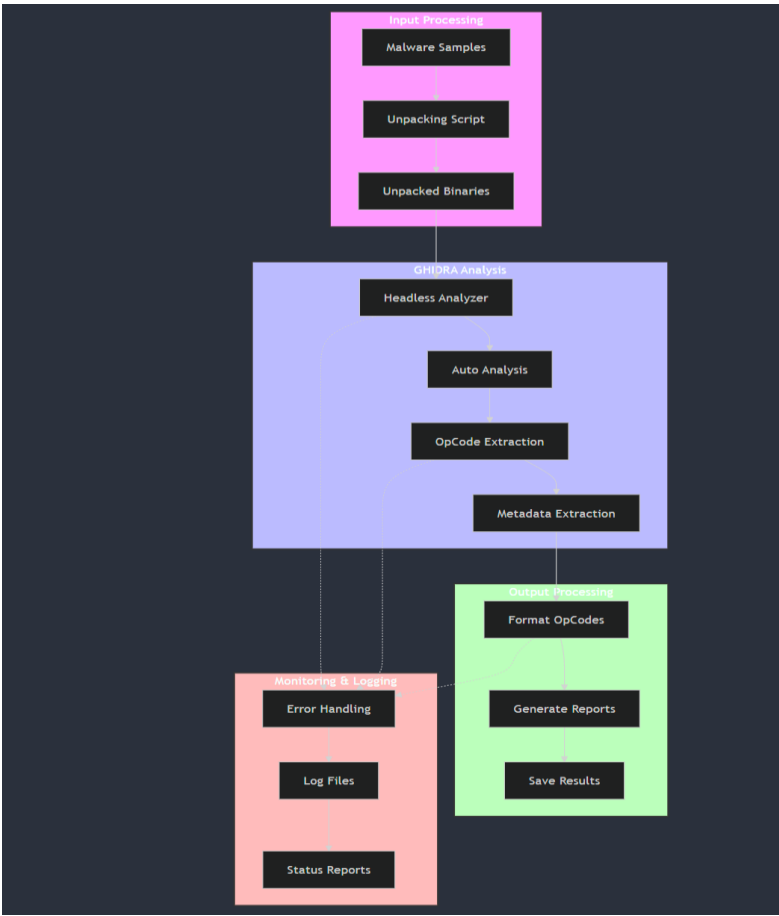
UNIVERSITY OF GUELPH.....	0
<b>Technical Report: Malware OpCode Extraction and Analysis .....</b>	<b>2</b>
<b>I.    Executive Summary.....</b>	<b>2</b>
<b>II.   System Architecture and Workflow .....</b>	<b>2</b>
<b>III.   Key Components .....</b>	<b>3</b>
<b>IV.   Implementation Overview.....</b>	<b>3</b>
<b>V.    Implementation Components .....</b>	<b>3</b>
<b>VI.   OpCode Extraction Process.....</b>	<b>5</b>
<b>VII.  Output Format .....</b>	<b>6</b>
<b>VIII. Conclusion .....</b>	<b>6</b>

# Technical Report: Malware OpCode Extraction and Analysis

## I. Executive Summary

This technical report documents the implementation and usage of automated tools for extracting operational codes (OpCodes) from malware samples associated with Project 4 APT groups. Our focus is on APT groups including G0010 Turla, G0024 Putter Panda, G0079 DarkHydrus, and other significant threat actors. The extraction process utilizes GHIDRA as the primary code reversing tool, with custom scripts developed to automate the extraction and analysis process. This implementation allows efficient analysis of malware operational characteristics while maintaining forensic integrity.

## II. System Architecture and Workflow



*[Opcode Extraction workflow]*

Our implementation follows a modular architecture designed for scalability and reliability. The diagram above illustrates the complete workflow of our malware analysis system, from initial sample processing through final output generation.

### III. Key Components

#### *3.1. Input Processing Module*

Handles encrypted malware samples

Maintains chain of custody

Validates file integrity

#### *3.2. GHIDRA Analysis Engine*

Performs automated binary analysis

Extracts OpCodes and metadata

Handles multiple architectures

#### *3.3. Output Processing Module*

Standardizes output format

Generates detailed reports

Maintains analysis logs

#### *3.4. Monitoring System*

Real-time error detection

Performance monitoring

Status reporting

### IV. Implementation Overview

#### *4.1. Tools and Environment*

GHIDRA

Decompiler version: DEV

Analysis engine: Advanced

Script support: Jython 2.7.2

#### *4.2. Development Environment*

Python/Jython for scripting

Windows 10/11 operating system

7-Zip for archive handling

Custom batch scripts for automation

#### *4.3. Required Resources*

Minimum 16GB RAM

64-bit operating system

SSD storage recommended

Multi-core processor

### V. Implementation Components

The implementation consists of three main components:

#### *5.1. Unpacking Script (unpack.bat)*

The unpacking script serves as the initial entry point, handling. It handles secure archive extraction with the predefined passwords “infected”, manages the directory structure to keep the initial file organization by APT group and some basic integrity checks

```

unpack.py > ...
1  import os
2  import zipfile
3  import shutil
4  import logging
5  from pathlib import Path
6  import sys
7  from datetime import datetime
8
9  class MalwareUnpacker:
10     def __init__(self, source_dir, extract_dir, zip_password="infected"):
11         self.source_dir = Path(source_dir)
12         self.extract_dir = Path(extract_dir)
13         self.zip_password = zip_password.encode()
14         self.setup_logging()
15
16     def setup_logging(self):
17         timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
18         log_file = f"unpacking_log_{timestamp}.txt"
19
20         logging.basicConfig(
21             level=logging.DEBUG,
22             format='%(asctime)s - %(levelname)s - %(message)s',
23             handlers=[
24                 logging.FileHandler(log_file),
25                 logging.StreamHandler(sys.stdout)
26             ]
27         )
28
29     def unzip_file(self, zip_path, extract_path, password=True):
30         try:

```

*[Sample code in the unpack.py file]*

## 5.2. OpCode Extraction Script (opcode\_extractor.py)

This core component interfaces directly with Ghidra's API to manage the extraction process and handles different architectures as well as ensuring consistent output formatting.

```

opcode_extractor.py > extract_opcodes
25  def extract_opcodes():
26
27      print("APT Directory: {}".format(apt_group))
28      print("Malware hash: {}".format(original_file))
29      print("Output directory: {}".format(output_dir))
30      print("Writing opcodes to: {}".format(output_file))
31
32      # Get listing and monitor
33      listing = program.getListing()
34      monitor = ConsoleTaskMonitor()
35
36      with open(output_file, 'w') as f:
37          # Write header information
38          f.write("# Opcode extraction for malware analysis\n")
39          f.write("# APT Group: {}\n".format(apt_group))
40          f.write("# Malware Hash: {}\n".format(original_file))
41          f.write("# Executable format: {}\n".format(program.getExecutableFormat()))
42          f.write("# Processor: {}\n".format(program.getLanguage().getProcessor().toString()))
43          f.write("# Creation date: {}\n".format(program.getCreationDate()))
44          f.write("# Format: <Address> | <Bytes> | <Mnemonic> | <Full Instruction>\n\n")
45
46      # Process all instructions
47      print("Starting instruction processing...")
48      instruction_count = 0
49      instructions = listing.getInstructions(True)
50      while instructions.hasNext() and not monitor.isCancelled():
51          insn = instructions.next()
52          if insn:
53              # Get instruction components
54              addr = insn.getAddress()
55              bytes_str = " ".join([format(b & 0xFF, '02x') for b in insn.getBytes()])
56              mnemonic = insn.getMnemonicString()
57              full_insn = insn.toString()
58
59              # Write formatted instruction
60              f.write("{} | {} | {} | {}\n".format(addr, bytes_str, mnemonic, full_insn))
61              instruction_count += 1
62
63      # Progress indicator every 1000 instructions
64      if instruction_count % 1000 == 0:
65          print("Processed {} instructions...".format(instruction_count))
66
67      print("Successfully extracted {} opcodes to {}".format(instruction_count, output_file))
68      return True
69

```

*[sample code found in the opcode\_extractor.py file]*

### 5.3. Execution Script (extract\_opcodes.bat)

This script coordinates the overall process by managing the Ghidra headless analyzer, by handling script execution, managing logging, and coordinating error recovery.

```
1  REM Configuration
2
3
4  REM Configuration
5  set "GHIDRA_PATH=C:\Users\MCTI Student\Downloads\ghidra"
6  set "PROJECT_PATH=C:\Users\MCTI Student\Desktop\Sub 3\GhidraProject"
7  set "USER_SCRIPTS_DIR=%USERPROFILE%\ghidra_scripts"
8  set "MALWARE_DIR=C:\Users\MCTI Student\Desktop\Sub 3\Unpacked_Samples"
9  set "LOG_DIR=C:\Users\MCTI Student\Desktop\Sub 3\logs"
10 set "TEMP_DIR=C:\Users\MCTI Student\Desktop\Sub 3\temp"
11
12 echo Starting malware analysis process...
13 echo =====
14 echo Configuration:
15 echo GHIDRA_PATH: %GHIDRA_PATH%
16 echo PROJECT_PATH: %PROJECT_PATH%
17 echo USER_SCRIPTS_DIR: %USER_SCRIPTS_DIR%
18 echo MALWARE_DIR: %MALWARE_DIR%
19 echo LOG_DIR: %LOG_DIR%
20 echo =====
21 echo.
22
23 REM Create necessary directories
24 if not exist "%LOG_DIR%" mkdir "%LOG_DIR%"
25 if not exist "%TEMP_DIR%" mkdir "%TEMP_DIR%"
26 if not exist "%USER_SCRIPTS_DIR%" mkdir "%USER_SCRIPTS_DIR%"
27 if not exist "%PROJECT_PATH%" mkdir "%PROJECT_PATH%"
28
29 REM Create main log file
30 set "MAIN_LOG=%LOG_DIR%\main_process.log"
31 echo Analysis started at %date% %time% > "%MAIN_LOG%"
32
33 REM Copy script to Ghidra user scripts directory
34 copy /Y "opcode_extractor.py" "%USER_SCRIPTS_DIR%" >nul
35
```

*[Sample code in the extract\_opcodes.bat file]*

## VI. OpCode Extraction Process

### 6.1. Initialization Phase

Environment validation

Directory structure verification

Log file initialization

Resource allocation

### 6.2. Malware Processing Phase

File integrity checking

Malware unpacking

Initial classification

Architecture detection

### 6.3. Analysis Phase

GHIDRA loading

Auto-analysis execution

OpCode extraction

Metadata collection

### 6.4. Output Generation Phase

Data formatting

Report generation

File organization

Error checking

## VII. Output Format

### 7.1. Opcode sample output

```
Unpacked_Samples > 02 PITYTIGER (MITRE ID G0011) > opcodes > 8df17aa2a351c09c3e331fdb6cb3947eb06f057b2ff17eed5d89b148d1380a4a.opcode X unpack.py extract_opcodes.b
1 | Opcode extraction for malware analysis
2 | # APT Group: 02 PITYTIGER (MITRE ID G0011)
3 | # Malware Hash: 8df17aa2a351c09c3e331fdb6cb3947eb06f057b2ff17eed5d89b148d1380a4a
4 | # Executable format: Portable Executable (PE)
5 | # Processor: x86
6 | # Creation date: Sun Nov 03 21:04:20 EST 2024
7 | # Format: <Address> | <Bytes> | <Mnemonic> | <Full Instruction>
8 |
9 | 00402fdc | 55 | PUSH | PUSH RBP
10 | 00402fdd | 8b ec | MOV | MOV EBP,ESP
11 | 00402fdf | 6a ff | PUSH | PUSH -0x1
12 | 00402fe1 | 68 78 71 40 00 | PUSH | PUSH 0x407170
13 | 00402fe6 | 68 78 3e 40 00 | PUSH | PUSH 0x403e78
14 | 00402feb | 64 a1 00 00 00 00 50 64 89 25 | MOV | MOV EAX,FS:[0x258964500000000]
15 | 00402ff5 | 00 00 | ADD | ADD byte ptr [RAX],AL
16 | 00402ff7 | 00 00 | ADD | ADD byte ptr [RAX],AL
17 | 00402ff9 | 83 ec 58 | SUB | SUB ESP,0x58
18 | 00402ffc | 53 | PUSH | PUSH RBX
19 | 00402ffd | 56 | PUSH | PUSH RSI
20 | 00402ffe | 57 | PUSH | PUSH RDI
21 | 00402fff | 89 65 e8 | MOV | MOV dword ptr [RBP + -0x18],ESP
22 | 00403002 | ff 15 b8 70 40 00 | CALL | CALL qword ptr [0x0080a0c0]
23 | 00403008 | 33 d2 | XOR | XOR EDX,EDX
24 | 0040300a | 8a d4 | MOV | MOV DL,AH
25 | 0040300c | 89 15 a4 cd 60 00 | MOV | MOV dword ptr [0x00a0fdb6],EDX
26 | 00403012 | 8b c8 | MOV | MOV ECX,EAX
27 | 00403014 | 01 e1 ff 00 00 00 | AND | AND ECX,0xff
28 | 0040301a | 89 0d a0 cd 60 00 | MOV | MOV dword ptr [0x00a0fdc0],ECX
29 | 00403020 | c1 e1 08 | SHL | SHL ECX,0x8
30 | 00403023 | 03 ca | ADD | ADD ECX,EDX
31 | 00403025 | 89 0d 9c cd 60 00 | MOV | MOV dword ptr [0x00a0fdc7],ECX
32 | 0040302b | c1 e8 10 | SHR | SHR EAX,0x10
33 | 0040302e | a3 98 cd 60 00 6a 01 e8 e9 | MOV | MOV [0xe9e8016a0060cd98],EAX
```

### 7.2. Log Output Sample

```
(AutoImporter)
INFO IMPORTING: loaded 0 additional files (HeadlessAnalyzer)
INFO ANALYZING all memory and code: file:///C:/Users/MCTI Student/Desktop/Sub 3/temp/sample.bin (HeadlessAnalyzer)
INFO Skipping POB processing: missing POB information in program metadata (PdbUniversalAnalyzer)
INFO Packed database cache: C:\Users\MCTI Student\AppData\Local\ghidra\packed-db-cache (PackedDatabaseCache)
INFO Applied data type archive: windows_vs12_64 (ApplyDataArchiveAnalyzer)
WARN Decompiling 0040699c, pcode error at 004069aa: Unable to resolve constructor at 004069aa (DecompileCallback)
WARN Decompiling 004032fc, pcode error at 00403f70: Unable to resolve constructor at 00403f70 (DecompileCallback)
WARN Decompiling 004032fc, pcode error at 004032a4: Unable to resolve constructor at 004032a4 (DecompileCallback)
WARN Decompiling 004032fc, pcode error at 004032b1: Unable to resolve constructor at 004032b1 (DecompileCallback)
INFO -----
INFO ASCII Strings 1.052 secs
INFO Apply Data Archives 3.046 secs
INFO Call Convention ID 0.025 secs
INFO Call-Fixup Installer 0.017 secs
INFO Create Address Tables 1.778 secs
INFO Create Address Tables - One Time 0.190 secs
INFO Create Function 0.046 secs
INFO Data Reference 0.035 secs
INFO Decompiler Parameter ID 2.335 secs
INFO Decompiler Switch Analysis 1.172 secs
INFO Demangler Microsoft 0.051 secs
INFO Disassemble Entry Points 0.279 secs
INFO Embedded Media 0.072 secs
INFO External Entry References 0.000 secs
INFO Function ID 0.119 secs
INFO Function Start Search 0.011 secs
INFO Non-Returning Functions - Discovered 0.027 secs
INFO Non-Returning Functions - Known 0.006 secs
INFO POB Universal 0.003 secs
INFO Reference 0.226 secs
INFO Scalar Operand References 0.016 secs
INFO Shared Return Calls 0.017 secs
INFO Stack 0.182 secs
INFO Subroutine References 0.032 secs
INFO WindowsResourceReference 1.041 secs
INFO x86 Constant Reference Analyzer 0.279 secs
INFO -----
INFO Total Time 12 secs
INFO -----
(AutoAnalysisManager)
INFO REPORT: Analysis succeeded for file: file:///C:/Users/MCTI Student/Desktop/Sub 3/temp/sample.bin (HeadlessAnalyzer)
INFO SCRIPT: C:\Users\MCTI Student\ghidra_scripts\opcode_extractor.py (HeadlessAnalyzer)
Starting opcode extraction...
Processing program: sample.bin
Program path: C:\Users\MCTI Student\Desktop\Sub 3\temp\sample.bin
Found matching file in: 07 TA459(MITRE ID G0062)
API Directory: 07 TA459(MITRE ID G0062)
Malware hash: 0d219aa54b1d417da61bd4aed5eeb53d6cb91b3287d53186b21fed450248215
Output directory: C:\Users\MCTI Student\Desktop\Sub 3\Unpacked_Samples\07 TA459(MITRE ID G0062)\opcodes
Writing opcodes to: C:\Users\MCTI Student\Desktop\Sub 3\Unpacked_Samples\07 TA459(MITRE ID G0062)\opcodes\0d219aa54b1d417da61bd4aed5eeb53d6cb91b3287d53186b21fed450248215.opcode
Starting instruction processing...
```

## VIII. Conclusion

The implementation successfully automates the complex process of OpCode extraction from various malware samples. The system demonstrates strong handling of different architectures and maintains high accuracy in output generation.