Technical Report:
Using Ghidra to Extract Malware Instruction Opcodes

1. Introduction

This report explains how we used Ghidra, an open-source reverse engineering framework developed by the NSA, to safely extract opcode sequences from malware samples. Instead of executing malware, we relied entirely on static analysis. The goal was to build datasets of instruction mnemonics for malware classification, research, and pattern recognition.

2. Purpose and Scope

- **Purpose**: Extract opcode sequences from malware binaries in a controlled, research-focused workflow.
- **Scope:** Static analysis only**,** the malware is never executed.
- **Usage**: Useful for malware clustering, feature extraction, signature development, and machine-learning datasets.

OpCodes such as **mov**, **push**, **call**, etc., provide useful low-level insight into how code behaves. When exported at scale, they can help identify similarities between malware families.

3. Safety, Legal, and Operational Precautions

Because actual malware samples were used, analysis took place in a strict defensive environment:

✅ Isolated virtual machine (no internet, no shared drives)

✅ Snapshotting enabled for rollback

✅ Samples stored in a read-only repository

✅ No execution, only static disassembly

✅ SHA-256 recorded and stored for each sample

These steps prevent accidental infection and ensure the work follows safe and ethical handling practices.

4. Tools and Environment

Component	Purpose
Ghidra	Disassembly, decompilation, and scripting
Python/Jython inside Ghidra	Automation for opcode extraction
CSV / .opcode output	Structured storage for analysis
Isolated VM (Windows/Linux)	Malware-safe working environment

5. Summary of the Workflow

1. Load the malware sample into Ghidra
2. Allow Ghidra's analyzers to run
3. Run a custom extraction script to collect opcodes
4. Export results to .opcode file
5. Use exported opcodes for research or machine-learning pipelines

This ensures reliable, repeatable results across many samples.

6. Output Format

The script produces a .opcode file (CSV-style) with columns:

Field	Description
addr	Address of the instruction
opcode	Instruction mnemonic (e.g., mov, push, call)
section_name	Memory section the instruction belongs to

7. Ghidra Automation Script (Concept Example)

Below is the custom opcode extraction script used during analysis.

It runs entirely inside Ghidra's scripting environment and does not execute any malware, it only reads the disassembly and exports instruction information.

```
# dump_opcodes.py

import os
import csv
import time
from ghidra.program.model.address import AddressSet
from ghidra.app.cmd.disassemble import DisassembleCommand

args = getScriptArgs()

try:
  outputs_folder = os.getcwd()
  results_folder = os.path.join(outputs_folder, 'results')
except Exception as exception:
  print("Exception: {}".format(exception))
  raise

filename = currentProgram.getName()
output_file = os.path.join(results_folder, filename + '.opcode')

try:
  start = time.clock()
  blocks = currentProgram.getMemory().getBlocks()

  if not blocks:
    raise Exception("Exception: no memory block")
```

```
opcodes = []
for block in blocks:
   section = block.getName()
   address_set = AddressSet(block.getStart(), block.getEnd())

   disassembled = DisassembleCommand(address_set, address_set, True)
   disassembled.applyTo(currentProgram)

   instructions = currentProgram.getListing().getInstructions(address_set, True)
   while instructions.hasNext():
      instruction = instructions.next()
      address = int(instruction.getAddress().getOffset())
      opcode = str(instruction).split(' ')[0]
      opcodes.append([address, opcode, section])

duration = time.clock() - start

if not opcodes:
   raise Exception("Exception: no instruction")

os.makedirs(results_folder, exist_ok=True)

with open(output_file, 'w') as csvfile:
   csvwriter = csv.writer(csvfile)
   csvwriter.writerow(['addr', 'opcode', 'section_name'])
```

```
        csvwriter.writerows(opcodes)

    print("Opcodes extracted for {}".format(filename))
    print("Duration: {:.2f}s".format(duration))

except Exception as exception:
    print("Exception in extraction: {}".format(exception))
```

8. Why This Script Was Useful

- Fully automated opcode extraction
- Works across any supported architecture in Ghidra
- Saves data in a researcher-friendly format
- Does not modify or execute the sample
- Consistent and repeatable across large datasets

This greatly reduced manual labor and allowed us to build datasets from many malware samples quickly and safely.

9. Limitations & Considerations

- Packed or encrypted samples may need unpacking before extraction
- Static opcodes do not show runtime behavior
- Cross-architecture normalization may be required for ML use

Even with these limitations, opcode extraction is a valuable first step in malware research and analysis.

10. Conclusion

Using Ghidra and automated scripting, we successfully built a pipeline to safely extract opcode sequences from malware samples. The resulting data can now be used for similarity analysis, machine-learning experiments, threat research, or academic study, all without ever executing malicious code.