CS5138 Malware Analysis, Spring 2023
Final Project Report

CS5138 Malware Analysis
Prof. Will Hawkins
04/28/2023
Student 1: Adel Alshappip
Student 2: Edwin Cervantes

Our tool provides several functionalities, including:

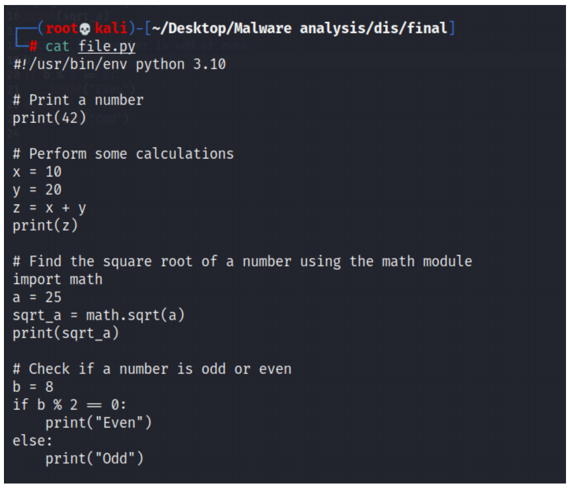
1- Generating a .pyc file from a given .py file and comparing it with another .pyc file with multiple disassembling options and give feedback.
2- Disassembling a .pyc file with multiple disassembling options.
3- Providing a user guide on how to use the tool and handling unexpected inputs.

Task 1:
The first task is that our tool can generate a .pyc file from a given file.py and compare it with another .pyc file to identify any differences in the code. It can print details about what is in command and what is the difference.The tool provides several disassembling options, including:
--d/--Disassemble: compares the disassembled bytecode
--R/--Read: compares the entire file in memory
--D/--Display: provides a human-readable interpretation of the bytecode instructions
--I/--Instructions: provides the bytecode instructions that the Python interpreter would execute when running the program

To use the tool, enter the command in the terminal as follows:
./disassembler.py --D file.py otherfile.pyc

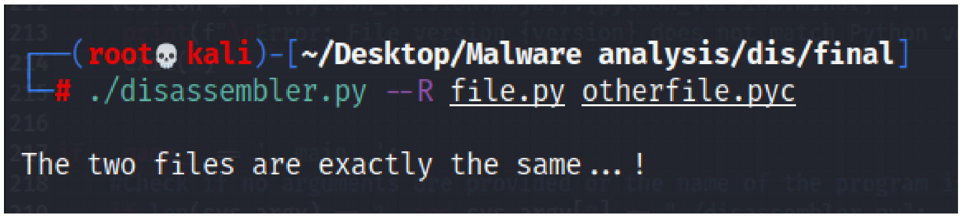
Here is an example with an explanation: we will use the option --D/--Display: provides a human-readable interpretation of the bytecode instructions.

Scenario 1:
Assume we have two files that are the same, file.py and otherfile.pyc.



```
┌──(root💀kali)-[~/Desktop/Malware analysis/dis/final]
└─# cat file.py
#!/usr/bin/env python 3.10

# Print a number
print(42)

# Perform some calculations
x = 10
y = 20
z = x + y
print(z)

# Find the square root of a number using the math module
import math
a = 25
sqrt_a = math.sqrt(a)
print(sqrt_a)

# Check if a number is odd or even
b = 8
if b % 2 == 0:
    print("Even")
else:
    print("Odd")
```

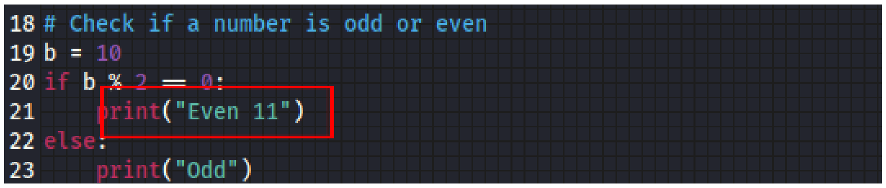
We want to compare them. We will use our tool to check.



```
┌──(root💀kali)-[~/Desktop/Malware analysis/dis/final]
└─# ./disassembler.py --R file.py otherfile.pyc

The two files are exactly the same...!
```

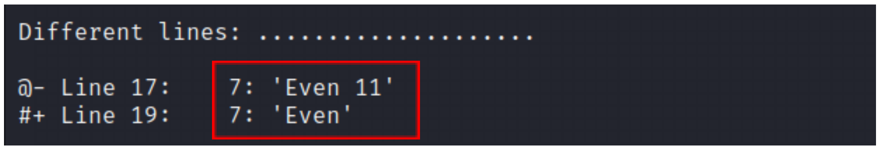
From the above screenshot, since the files are the same, the tool will print a message showing that the files are the same.

Scenario 2: We will modify the source code (file.py) a little bit by changing print if from print("Even 11") instead of print("Even"). We will then use our tool to catch the differences between the original file and the modified file.



```python
# Check if a number is odd or even
b = 10
if b % 2 == 0:
    print("Even 11")
else:
    print("Odd")
```

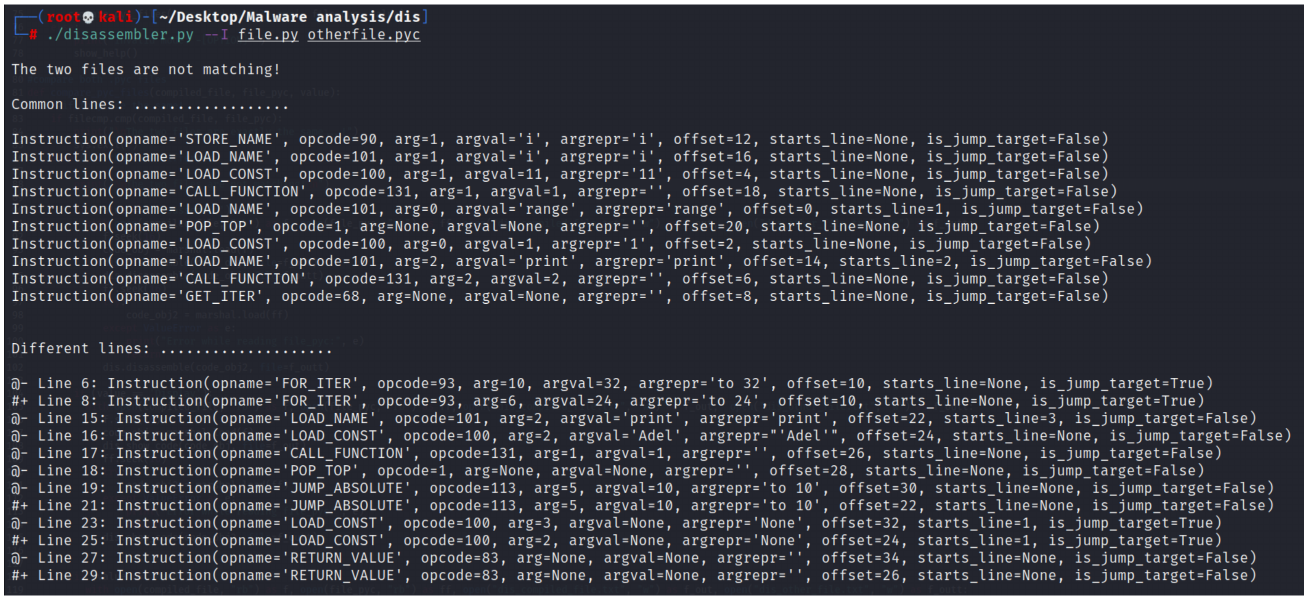
We will run our tool with the files again!



```
Different lines: ....................

@- Line 17:    7: 'Even 11'
#+ Line 19:    7: 'Even'
```

Great! Our tool caught the differences and printed a bunch of details comparing the files together in the terminal.

More comparing with other files:



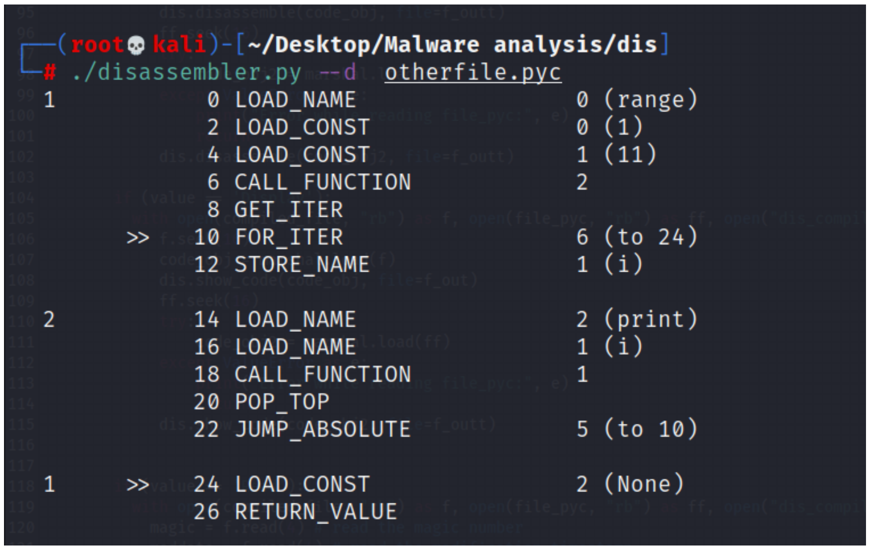
```
┌──(root💀kali)-[~/Desktop/Malware analysis/dis]
└─# ./disassembler.py --I file.py otherfile.pyc

The two files are not matching!

Common lines: ..................

Instruction(opname='STORE_NAME', opcode=90, arg=1, argval='i', argrepr='i', offset=12, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_NAME', opcode=101, arg=1, argval='i', argrepr='i', offset=16, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_CONST', opcode=100, arg=1, argval=11, argrepr='11', offset=4, starts_line=None, is_jump_target=False)
Instruction(opname='CALL_FUNCTION', opcode=131, arg=1, argval=1, argrepr='', offset=18, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_NAME', opcode=101, arg=0, argval='range', argrepr='range', offset=0, starts_line=1, is_jump_target=False)
Instruction(opname='POP_TOP', opcode=1, arg=None, argval=None, argrepr='', offset=20, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_CONST', opcode=100, arg=0, argval=1, argrepr='1', offset=2, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_NAME', opcode=101, arg=2, argval='print', argrepr='print', offset=14, starts_line=2, is_jump_target=False)
Instruction(opname='CALL_FUNCTION', opcode=131, arg=2, argval=2, argrepr='', offset=6, starts_line=None, is_jump_target=False)
Instruction(opname='GET_ITER', opcode=68, arg=None, argval=None, argrepr='', offset=8, starts_line=None, is_jump_target=False)


Different lines: ...................

@- Line 6: Instruction(opname='FOR_ITER', opcode=93, arg=10, argval=32, argrepr='to 32', offset=10, starts_line=None, is_jump_target=True)
#+ Line 8: Instruction(opname='FOR_ITER', opcode=93, arg=6, argval=24, argrepr='to 24', offset=10, starts_line=None, is_jump_target=True)
@- Line 15: Instruction(opname='LOAD_NAME', opcode=101, arg=2, argval='print', argrepr='print', offset=22, starts_line=3, is_jump_target=False)
@- Line 16: Instruction(opname='LOAD_CONST', opcode=100, arg=2, argval='Adel', argrepr="'Adel'", offset=24, starts_line=None, is_jump_target=False)
@- Line 17: Instruction(opname='CALL_FUNCTION', opcode=131, arg=1, argval=1, argrepr='', offset=26, starts_line=None, is_jump_target=False)
@- Line 18: Instruction(opname='POP_TOP', opcode=1, arg=None, argval=None, argrepr='', offset=28, starts_line=None, is_jump_target=False)
@- Line 19: Instruction(opname='JUMP_ABSOLUTE', opcode=113, arg=5, argval=10, argrepr='to 10', offset=30, starts_line=None, is_jump_target=False)
#+ Line 21: Instruction(opname='JUMP_ABSOLUTE', opcode=113, arg=5, argval=10, argrepr='to 10', offset=22, starts_line=None, is_jump_target=False)
@- Line 23: Instruction(opname='LOAD_CONST', opcode=100, arg=3, argval=None, argrepr='None', offset=32, starts_line=1, is_jump_target=True)
#+ Line 25: Instruction(opname='LOAD_CONST', opcode=100, arg=2, argval=None, argrepr='None', offset=24, starts_line=1, is_jump_target=True)
@- Line 27: Instruction(opname='RETURN_VALUE', opcode=83, arg=None, argval=None, argrepr='', offset=34, starts_line=None, is_jump_target=False)
#+ Line 29: Instruction(opname='RETURN_VALUE', opcode=83, arg=None, argval=None, argrepr='', offset=26, starts_line=None, is_jump_target=False)
```

Task 2:

Our tool is capable of disassembling a .pyc file with five different options:
Options:

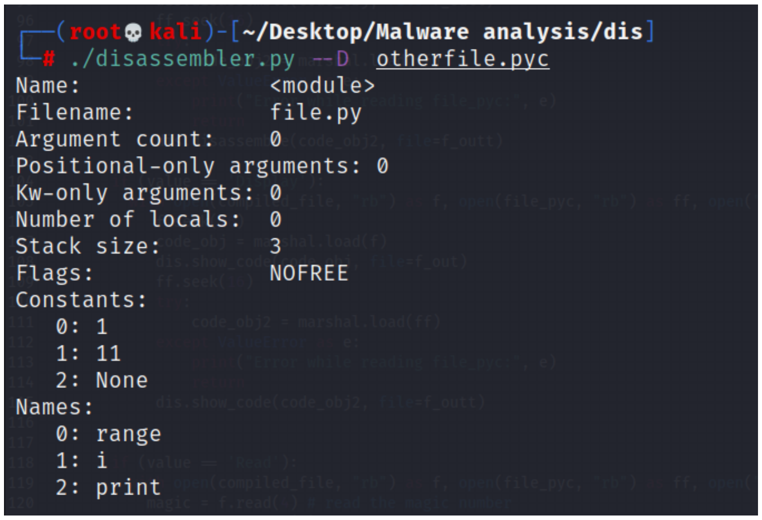
--d, --Disassemble	Disassemble bytecode from Python.
--r, --Read	Reads the entire file into memory.
--D, --Display	Displays a human-readable interpretation of the bytecode instructions.
--i, --Instructions	Bytecode instructions that the Python interpreter would execute when running the program.
--f, --Find	Find the offsets which are the start of lines in the source code.

To use this tool, run the following command: $./disassembler --[option] [filename].pyc
With option --d/--Disassemble



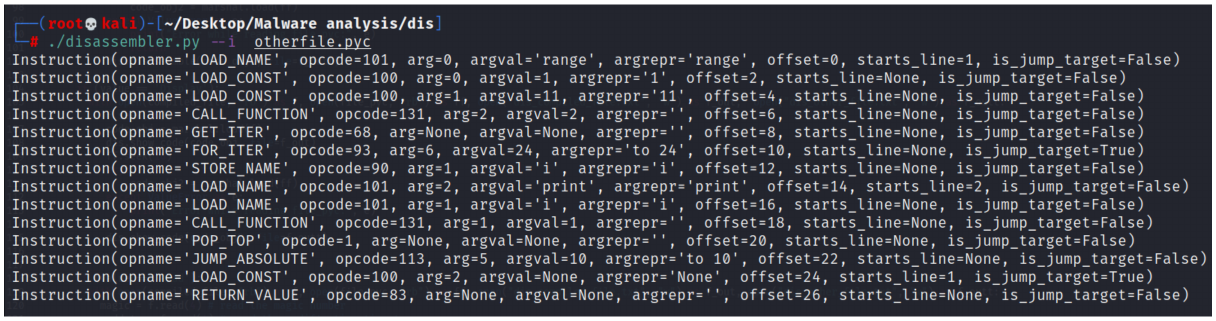
```
┌──(root💀kali)-[~/Desktop/Malware analysis/dis]
└─# ./disassembler.py --d otherfile.pyc
  1           0 LOAD_NAME                0 (range)
              2 LOAD_CONST               0 (1)
              4 LOAD_CONST               1 (11)
              6 CALL_FUNCTION            2
              8 GET_ITER
        >>   10 FOR_ITER                 6 (to 24)
             12 STORE_NAME               1 (i)

  2          14 LOAD_NAME                2 (print)
             16 LOAD_NAME                1 (i)
             18 CALL_FUNCTION            1
             20 POP_TOP
             22 JUMP_ABSOLUTE            5 (to 10)

  1     >>   24 LOAD_CONST               2 (None)
             26 RETURN_VALUE
```

With option --D, --Display



```
┌──(root💀kali)-[~/Desktop/Malware analysis/dis]
└─# ./disassembler.py --D otherfile.pyc
Name:              <module>
Filename:          file.py
Argument count:    0
Positional-only arguments: 0
Kw-only arguments: 0
Number of locals:  0
Stack size:        3
Flags:             NOFREE
Constants:
   0: 1
   1: 11
   2: None
Names:
   0: range
   1: i
   2: print
```

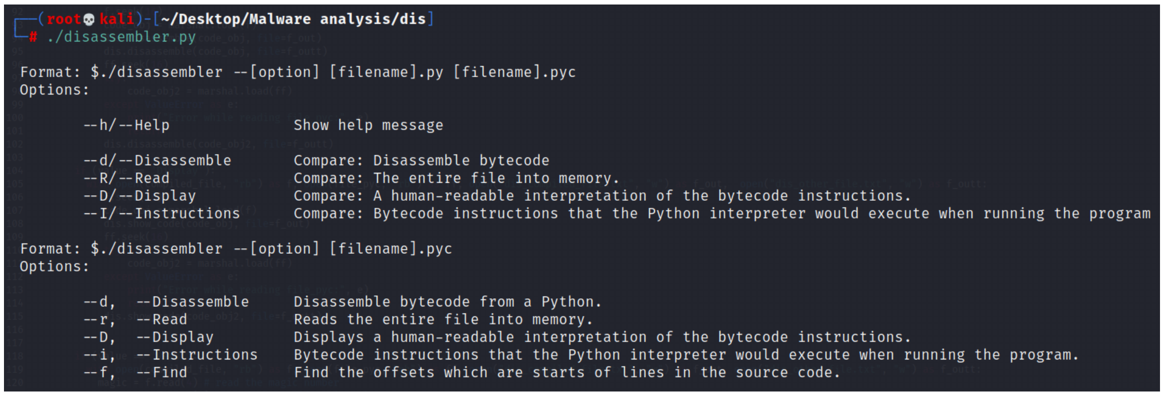
With option --i, --Instructions



```
┌──(root💀kali)-[~/Desktop/Malware analysis/dis]
└─# ./disassembler.py --i  otherfile.pyc
Instruction(opname='LOAD_NAME', opcode=101, arg=0, argval='range', argrepr='range', offset=0, starts_line=1, is_jump_target=False)
Instruction(opname='LOAD_CONST', opcode=100, arg=0, argval=1, argrepr='1', offset=2, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_CONST', opcode=100, arg=1, argval=11, argrepr='11', offset=4, starts_line=None, is_jump_target=False)
Instruction(opname='CALL_FUNCTION', opcode=131, arg=2, argval=2, argrepr='', offset=6, starts_line=None, is_jump_target=False)
Instruction(opname='GET_ITER', opcode=68, arg=None, argval=None, argrepr='', offset=8, starts_line=None, is_jump_target=False)
Instruction(opname='FOR_ITER', opcode=93, arg=6, argval=24, argrepr='to 24', offset=10, starts_line=None, is_jump_target=True)
Instruction(opname='STORE_NAME', opcode=90, arg=1, argval='i', argrepr='i', offset=12, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_NAME', opcode=101, arg=2, argval='print', argrepr='print', offset=14, starts_line=2, is_jump_target=False)
Instruction(opname='LOAD_NAME', opcode=101, arg=1, argval='i', argrepr='i', offset=16, starts_line=None, is_jump_target=False)
Instruction(opname='CALL_FUNCTION', opcode=131, arg=1, argval=1, argrepr='', offset=18, starts_line=None, is_jump_target=False)
Instruction(opname='POP_TOP', opcode=1, arg=None, argval=None, argrepr='', offset=20, starts_line=None, is_jump_target=False)
Instruction(opname='JUMP_ABSOLUTE', opcode=113, arg=5, argval=10, argrepr='to 10', offset=22, starts_line=None, is_jump_target=False)
Instruction(opname='LOAD_CONST', opcode=100, arg=2, argval=None, argrepr='None', offset=24, starts_line=1, is_jump_target=True)
Instruction(opname='RETURN_VALUE', opcode=83, arg=None, argval=None, argrepr='', offset=26, starts_line=None, is_jump_target=False)
```

Task3:
When the user runs the tool with "./disassembler.py", it will display a list of available options to guide the user on how to use it. Additionally, the tool will provide a help guide in case any of the following errors occur:

- File not found.
- Incorrect file extension, for example using ".exe" instead of ".pyc".
- More arguments than the specified limit.
- Fewer arguments than the specified limit.
- Invalid option name.



```
┌──(root💀kali)-[~/Desktop/Malware analysis/dis]
└─# ./disassembler.py

 Format: $./disassembler --[option] [filename].py [filename].pyc
 Options:

        --h/--Help              Show help message

        --d/--Disassemble       Compare: Disassemble bytecode
        --R/--Read              Compare: The entire file into memory.
        --D/--Display           Compare: A human-readable interpretation of the bytecode instructions.
        --I/--Instructions      Compare: Bytecode instructions that the Python interpreter would execute when running the program

 Format: $./disassembler --[option] [filename].pyc
 Options:

        --d,  --Disassemble     Disassemble bytecode from a Python.
        --r,  --Read            Reads the entire file into memory.
        --D,  --Display         Displays a human-readable interpretation of the bytecode instructions.
        --i,  --Instructions    Bytecode instructions that the Python interpreter would execute when running the program.
        --f,  --Find            Find the offsets which are starts of lines in the source code.
```