DearleyPY- Augmentation of Python Decompilation

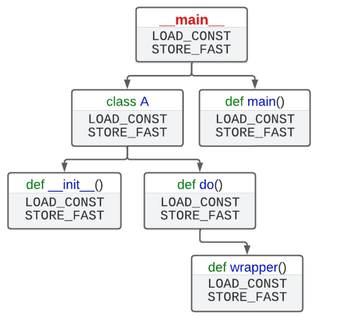
Jonsonn School Undergraduate Research Scholar Awards

David Wank, The University of Texas at Dallas

Background

```
 0 # A class definition with
 1 # nested functions
 2
 3
 4 class A:
 5     def __init__(self):
 6         ...
 8     def do(self):
 9         def wrapper():
10             ...
11
12 def main():
13     ...
14
15
16 if __name__ == "__main__":
17     main()
18
```

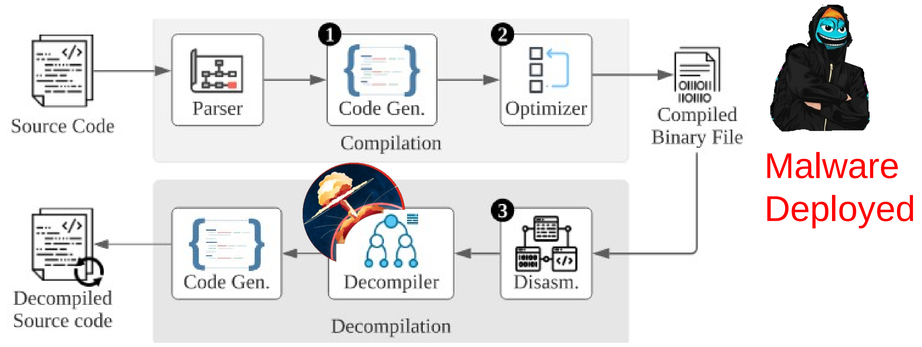
(a) Python source code



(b) Nested code object (CO) structure

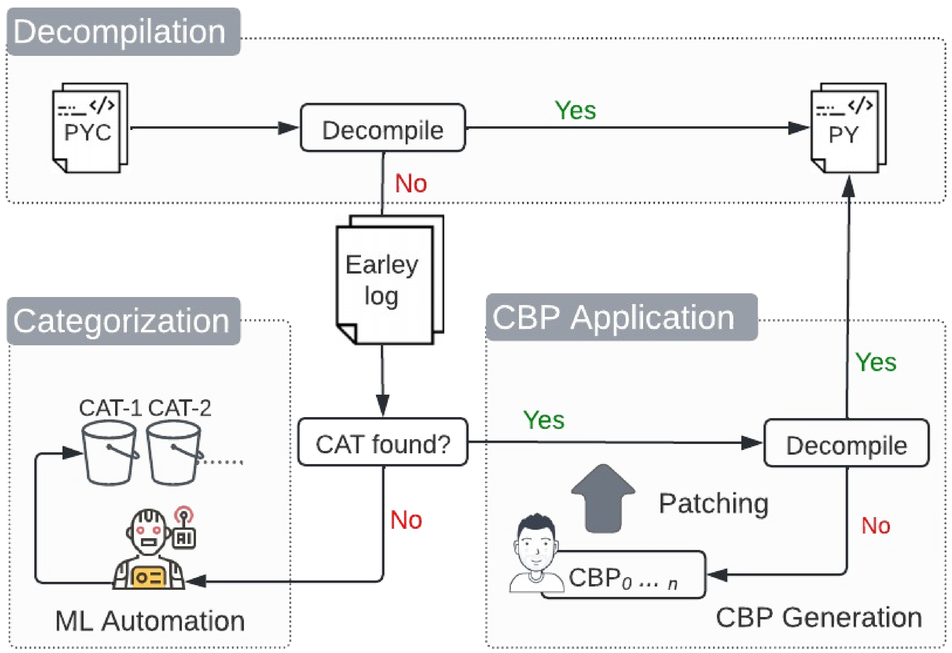
- Python compiles to bytecode and is stored in a hierarchy
- Python bytecode may execute without the source code

Problem Statement



- Malicious actors may distribute malware as Python bytecode in a way that thwarts existing analysis tools which decompile code
- Can we improve the analysis tools to handle edge cases?

Proposed Solution



- DearleyPY augments existing decompilers, enabling end-users with little knowledge of decompilation details to produce patch rules in an end-to-end process
- DearleyPY uses ML to automate significant portions of the analysis process, reducing workload on the end-user

Results

1	`try:` `  return 0`	Replacing	`try:` `  CBP_RETURN = 0`

3	`if a and b and c and d:`
	`  foo()`
		Dividing	`CBP_tmp = a and b and c`
			`if CBP_tmp and d:`
			`  foo()`

4	`if a:`
	`  for ...:`
	`  foo()`
	`elif b:`
	`  bar()`
		Uncoupling	`if a:`
			`  for ...:`
			`  foo()`
			`if not a and b:`
			`  bar()`

5	`x = range(10)`
	`comp = (val for val`
	`  in x)`
		Differentiating	`x = range(10)`
			`comp = (val for val`
			`  in CBP_x)`

6	`for foo in range(10):`
	`  if bar:`
	`  break`
		Replacing	`for foo in range(10):`
			`  if bar:`
			`  CBP_BREAK = 1`

7	`return 0` `unreachable_foo()`	Deleting	`return 0`

(a) Original Statements (b) Patching (c) Equivalent Statements

- DearleyPY has a suite of built-in rules that can manipulate Python bytecode in various ways to resolve decompilation errors
- Users add patches similar to these to resolve errors they encounter

Conclusion

- We conducted the first-ever large-scale decompilation failure study, analyzing 1,635,411 cases in total and discovering seven major failure categories.
- We created DearleyPY, a suite of tooling for root cause analysis, binary patching, reduction of failures and more.