PyFET: Forensically Equivalent Transformation for Python Binary Decompilation (Supplementary Materials to Paper)

Appendix

7. Additional Details for Preliminary Study

7.1. Representativeness of Decompilation Failures in Preliminary Study. We analyzed all the evaluated 77,022 errors in Section 5, including error messages/implicit error patterns and FETs that fixed the errors, to infer the root causes. Specifically, we assume that two errors have the same root cause if they have the same explicit/implicit error and are resolved by the same FET.

Root Causes. From the results we observe that the preliminary study's findings are valid for the entire dataset because (1) we didn't observe new errors and (2) the distributions of the entire dataset and the preliminary study are similar as shown in Table 10.

Modules/Rules. Table 11 shows the distribution of erroneous rules/modules across the five decompilers for the entire dataset. In brackets, we see the difference between the entire dataset and preliminary study data (Fig. 1-(b)). We see an average difference of 0.84%, showing that the errors are as diverse as in the entire dataset.

TABLE 10. Difference in Distributions of Root Causes of Preliminary Study Data and Entire Dataset.

Category	Preliminary	Entire dataset	Diff[1]
Missing Parsing Rules	44.5%	38.7%	-5.8%
Conflicting Parsing Rules	42.4%	39.4%	-3.0%
Unsupported instructions	11.9%	21.1%	9.2%
Implementation Bugs	1.2%	0.8%	-0.4%

1: Percentage difference between the two datasets.

TABLE 11. Difference in Distributions of Parse Errors of Preliminary Study Data and Entire Dataset.

Parse Errors	Uncompyle6	Decompyle3	Uncompyle2	Unpyc37	Decompyle++
Conditional[1]	16.1% (0.8%)	10.1% (1.9%)	34.6% (0.8%)	10.2% (1.2%)	38.1% (1.5%)
Boolean[2]	14.3% (1.2%)	2.2% (0.4%)	11.1% (1.1%)	33.9% (0.7%)	3.1% (1.4%)
Loop Block	20.2% (0.2%)	31.9% (1.2%)	42.4% (1.4%)	6.3% (0.1%)	27.2% (0.4%)
Except.[3]	31.1% (0.8%)	51.2% (2.2%)	0.0% (0.0%)	26.0% (1.0%)	9.4% (0.7%)
`with` Block	9.8% (0.4%)	3.6% (0.3%)	0.0% (0.0%)	0.0% (0.0%)	16.8% (1.7%)
Other	8.5% (1.1%)	1.0% (0.2%)	11.9% (0.6%)	23.6% (0.5%)	5.4% (1.5%)

1: Conditional Block. 2: Boolean Expression. 3: `try/except` Block

7.2. Identifying Incorrectly Changing Semantics. To identify cases for incorrectly changing semantics we leverage the original source (S_o) and it is decompiled source (S_d). We compile S_o and S_d to obtain binaries B_o and B_d, respectively. We then follow the below three steps:

1. *Identical Code*: We prune out samples if their B_o and B_d are identical. The comparison is done at the bytecode instruction-level.
2. *Harmless Additional Code*: We search and remove a few known additional code patterns (e.g., additional `del` in an exception block) in S_d at the source-level. We then compile the modified S_{d-new} to get B_{d-new}. If B_o and B_{d-new} are identical, it is a harmless additional code case.
3. *Incorrectly Omitted Code*: We follow the steps in Section-2.2.2 by instrumenting instructions in B_o, obtaining B_{o-new}. We decompile B_{o-new} to get S_{d-new}. If S_{d-new} misses any instrumented instructions, it is an incorrectly omitted case.

Finally, incorrectly changing code cases are samples that are not detected by Steps 1~3.

8. Additional Details for Design

8.1. Example of Applying a FET Rule. Fig. 15-(a) shows an example code snippet causing a decompilation error, due to a predicate containing more than 3 expressions followed by '`return None`'. Fig. 15-(b) presents a transformed code snippet that resolves the error by separating the expressions from the predicate (via a new variable '`temp_cond`'). In this example, the transformation rule shown in Fig. 6 achieves this on the program in Fig. 16.

Fig. 16-(a) is the target binary's CFG where ⓿ is the selected initial target block. Since transforming the initial block does not solve the decompilation error, PyFET tries additional blocks marked by ❶. Then, we get matching instructions from the offsets 8 to 38 as shown in Fig. 16-(b) by applying the regular expression shown in Fig. 6-(a) on the basic block with offsets 0~8 in Fig. 16-(a). Note that we use a red background to mark the matched patterns. The patterns in the bold text represent the patterns with transformation rules shown in Fig. 6-(c). Observe Fig. 16-(c) for the transformed control flow. Specifically, PyFET replaces `POP_JUMP_IF_FLASE` (at offset 8) with `JUMP_IF_FLASE_OR_POP` and `POP_JUMP_IF_TRUE` (at offsets 18 and 28) with `JUMP_IF_TRUE_OR_POP`. Note that when we change the jump instructions, we also change the jump targets from offset 40 to 38, which is omitted in Fig. 6-(c). Finally, at offset 38, we apply `RE-3`, which adds two instructions, `STORE_FAST` and `LOAD_FAST`. Observe that the addition shifts offsets of subsequent instructions.

9. Additional Details for Evaluation

9.1. Unsupported Instructions in `Decompyle++`. Fig. 17 shows the unsupported instructions in `Decompyle++`. The

```
if not_addr or s3_addr or http_addr or https_addr:
  return None
```

(a) Example Source Code causing a Decompilation Error.

```
tmp_cond = not_addr or s3_addr or http_addr or https_addr
if tmp_cond:
  return None
```

(b) Transformed Source Code

Figure 15. Source Code Representation of the Example.

second column shows the number of unsupported instructions for each version and the accumulated number of unsupported instructions including all the earlier versions. Observe that from Python 2.7 to 3.9, the number of unsupported instructions is accumulated from 2 to 23. We also observe that the decompilers have more explicit errors on newer Python version binaries (in Table 5) due to, in part, these increasing number of unsupported instructions.

TABLE 12. Statistics of Selected Application.

	Name	Stars	Size	# Files[1]	# Functions	SLOC[2]
1	youtube-dl[3]	111K	6.4 MB	870	3,377	124,827
2	keras[4]	55K	15.8 MB	630	11,337	180,444
3	ansible[5]	53K	37.7 MB	1,060	7,288	103,136
4	localstack[6]	41K	16.4 MB	380	4,861	62,114
5	rich[7]	38K	19.4 MB	175	1,581	25,536
6	openpilot[8]	35K	179.2 MB	329	2,086	31,191
7	pandas[9]	34K	50.2 MB	886	16,455	226,219
8	XX-Net[10]	31K	40.0 MB	939	21,424	258,724
9	cheat.sh[11]	29K	6.3 MB	40	269	3,455
10	black[12]	28K	5.6 MB	160	1,335	103,016

1: Python Source Files. 2: Sum of SLOC of the Python Files.
3: https://github.com/ytdl-org/youtube-dl. 4: https://github.com/keras-team/keras.
5: https://github.com/ansible/ansible. 6: https://github.com/localstack/localstack.
7: https://github.com/Textualize/rich. 8: https://github.com/commaai/openpilot.
8: https://github.com/pandas-dev/pandas. 10: https://github.com/XX-net/XX-Net.
11: https://github.com/chubin/cheat.sh. 12: https://github.com/psf/black

9.2. Selected Top 10 Applications. Table 12 shows the top 10 applications out of 100 Python projects (Appendix 4.2) in terms of popularity that have at least 10 SLOC per function and more than 100 functions (i.e., the application has more than 100 functions with a majority of at least 10 lines of code). Details of all 100 samples can be found on [2].

TABLE 13. Statistics of Decompilers

Decompiler	Python (version)	Total		Parsing	
		SLOC	Fn.*	SLOC	Fn.*
Uncompyle6	≤3.8	93,686	1,704	31,389	621
Unpyc37	3.7	3,068	535	3,064	407
Decompyle++	≤3.9	11,317	459	6,114	308

*: The number of functions.

9.3. Statistics of Decompilers. Table 13 shows that each decompiler varies in its size and structure (i.e., the number of functions). In addition, we observe that their designs also vary significantly. For instance, a tree-like data structure is used to maintain the block scope in `Unpyc37`, while `Decompyle++` uses a stack data structure. `Uncompyle6` parses using `SPARK` [3] while `Unpyc37` and `Decompyle++` implement their own parsers from scratch. To support multiple Python versions, `Uncompyle6` keeps separate parsing rules for each version while `Decompyle++` uses a unified switch case block with all bytecode instructions and all parsing rules combined (in file `ASTree.cpp` with 3,223 SLOC of switch case).

10. Additional Details for Case Studies

10.1. Evaluation of Python 3.9 to 3.8 Migrated Binaries. We present more details of how the Python 3.9 binaries migrated to 3.8 in Section 5.4.1 are further handled by PyFET. Specifically, after changing the file version, we find a total of 106,509 (104,761 explicit and 1,748 implicit errors) in `Uncompyle6` and 108,322 errors (106,616 explicit and 1,706 implicit errors) in `Decompyle3`. PyFET applies 21 and 19 FETs for the two respective decompilers. The top three FETs used are as follows.

- `R18.` Migrating Python 3.9 comparisons (`is`, `in`, `is not`, and `not in`) into 3.8, fixing 49.9% of errors.
- `R19.` Migrating Python 3.9 exception type comparisons to its 3.8 instruction, fixing 16.9% of errors.
- `R20.` Transforming instruction for raising an exception (i.e., the `raise` keyword), fixing 13.6% of errors.

The complete list of FETs can be found in Table 4, Table 9 and Table 14. The remaining 18 FETs account for 19.6% of the errors. PyFET resolves all errors to enable support for 3.9 binaries for the two decompilers without going through any development process for the decompilers. The transformed binaries are runnable on Python 3.8 environment, meaning that the transformation is high quality.

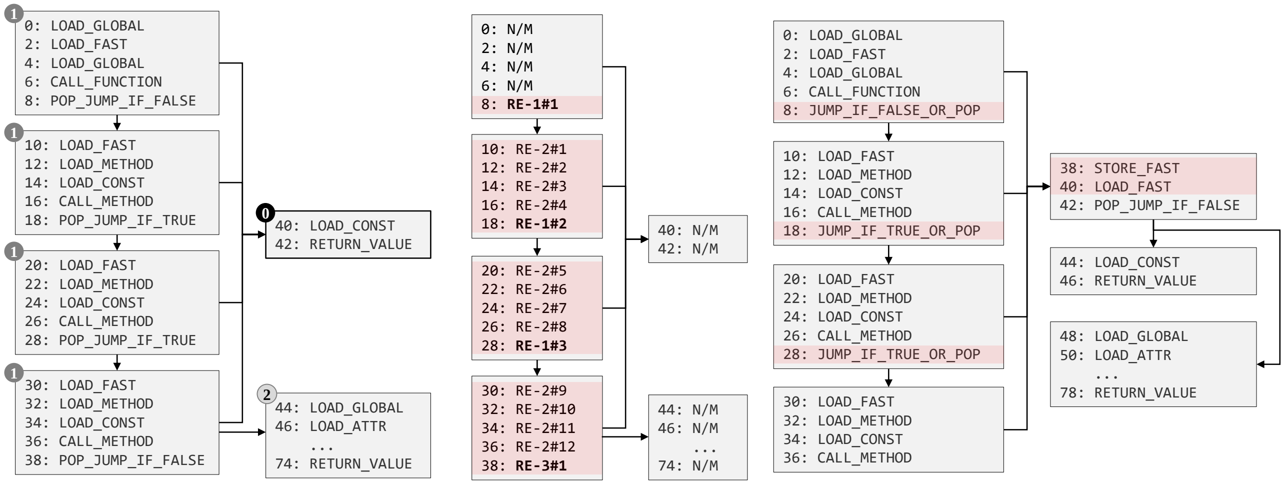
Differences in Instructions of Python 3.9 and 3.8. Table 15 shows instructions differences between Python 3.8 and 3.9 binaries.

FETs for Python 3.9 to 3.8 Migration. We list the FET rules used in Table 14 that help PyFET migrate Python 3.9 binaries to Python 3.8. Note that all of the FETs listed are essentially SETs.

10.2. Debugging `Unpyc37` and `Decompyle++`. We present details on debugging `Unpyc37` and `Decompyle++` in order to patch the decompilation errors outlined in Fig. 19.

Debugging `Unpyc37`. We aim to fix a bug to resolve the decompilation error in Fig. 19-(a). Note that since the decompiler does not provide any documentation except source code, we spend 10 hours just to understand the codebase. We find a solution that is removing 8 source code lines (2,508~2,516 in "`unpyc3.py`") that handles a sequence of boolean expressions with comparison operators (e.g., `<=`). While the patch fixes the error, it unfortunately introduces new implicit errors. Specifically, it makes the decompiler incorrectly decompile `or` to `and`, resulting in "`if c1 and c2`" where the desired outcome is "`if c1 or c2`".

Debugging `Decompyle++`. We aim to debug `Decompyle++` to handle the decompilation failure caused by Fig. 19-(b). Unlike other decompilers written in Python, `Decompyle++` is written in C/C++. We spent 6 hours in debugging to locate the code that handles the error-inducing statement: the `if` with `return` at line 3 in Fig. 19-(b). We notice that



```
0: LOAD_GLOBAL
2: LOAD_FAST
4: LOAD_GLOBAL
6: CALL_FUNCTION
8: POP_JUMP_IF_FALSE

10: LOAD_FAST
12: LOAD_METHOD
14: LOAD_CONST
16: CALL_METHOD
18: POP_JUMP_IF_TRUE

20: LOAD_FAST
22: LOAD_METHOD
24: LOAD_CONST
26: CALL_METHOD
28: POP_JUMP_IF_TRUE

30: LOAD_FAST
32: LOAD_METHOD
34: LOAD_CONST
36: CALL_METHOD
38: POP_JUMP_IF_FALSE

40: LOAD_CONST
42: RETURN_VALUE

44: LOAD_GLOBAL
46: LOAD_ATTR
    ...
74: RETURN_VALUE
```

(a) Target CFG (and instructions)

```
0: N/M
2: N/M
4: N/M
6: N/M
8: RE-1#1

10: RE-2#1
12: RE-2#2
14: RE-2#3
16: RE-2#4
18: RE-1#2

20: RE-2#5
22: RE-2#6
24: RE-2#7
26: RE-2#8
28: RE-1#3

30: RE-2#9
32: RE-2#10
34: RE-2#11
36: RE-2#12
38: RE-3#1

40: N/M
42: N/M

44: N/M
46: N/M
    ...
74: N/M
```

(b) Matched Regular Expressions

```
0: LOAD_GLOBAL
2: LOAD_FAST
4: LOAD_GLOBAL
6: CALL_FUNCTION
8: JUMP_IF_FALSE_OR_POP

10: LOAD_FAST
12: LOAD_METHOD
14: LOAD_CONST
16: CALL_METHOD
18: JUMP_IF_TRUE_OR_POP

20: LOAD_FAST
22: LOAD_METHOD
24: LOAD_CONST
26: CALL_METHOD
28: JUMP_IF_TRUE_OR_POP

30: LOAD_FAST
32: LOAD_METHOD
34: LOAD_CONST
36: CALL_METHOD

38: STORE_FAST
40: LOAD_FAST
42: POP_JUMP_IF_FALSE

44: LOAD_CONST
46: RETURN_VALUE

48: LOAD_GLOBAL
50: LOAD_ATTR
    ...
78: RETURN_VALUE
```

(c) Transformed CFG (and instructions)

Figure 16. Regular Expression Matching and Transformation Example (N/M: no matching, Highlighted lines: instructions matched in (b) and transformed in (c), A matched instruction is referred by 'RE-x#y' where the 'x' and 'y' represent the index of the pattern in the regular expression and the matched instance respectively).

Python Version	# Inst. (Acc.)*	Unsupported Instructions
2.7	2 (2)	BUILD_SET MAP_ADD
3.4	2 (4)	UNPAC_EX LOAD_CLASSDEREF
3.5	7 (11)	BUILD_TUPLE_UNPACK BUILD_MAP_UNPACK BUILD_LIST_UNPACK BUILD_MAP_UNPACK_WITH_CALL WITH_CLEANUP_START BEFORE_ASYNC_WITH GET_YIELD_FROM_ITER
3.6	2 (13)	CALL_FUNCTION_EX BUILD_TUPLE_UNPACK_WITH_CALL
3.8	2 (15)	BEGIN_FINALLY CALL_FINALLY
3.9	8 (23)	DICT_UPDATE LIST_TO_TUPLE LOAD_ASSERTION_ERROR WITH_EXCEPT_START JUMP_IF_NOT_EXC_MATCH DICT_MERGE STORE_ANNOTATION RERAISE

* The number of Unsupported Instructions (Accumulated).

Figure 17. Unsupported Instructions in Decompyle++

```
def error_1(...):
  return c1 or c2 \
         and not c3
```

(a) Error Inducing Code in Uncompyle6

Ref.	Description
R1	Dividing Logical Expression

(b) FET Rule Used in Uncompyle6

```
def error_1(...):
  FET_cond = c1 or c2
  return FET_cond \
         and not c3
```

(c) PYFET Fix for Error in Uncompyle6

```
def error_2(...):
  if c1 and c2 <= c3 <= c4:
     ...
```

(d) Error Inducing Code in Unpyc37

Ref.	Description
R1	Dividing Logical Expression

(e) FET Rule Used in Unpyc37

```
def error_2(...):
  FET_cond = c1 and c2 <= \
             c3 <= c4
  if FET_cond:
     ...
```

(f) PYFET Fix for Error in Unpyc37

```
def error_3(...):
  if c: return
  try: ...
  except: ...
```

(g) Error Inducing Code in Decompyle++

Ref.	Description
R16	Separating Control Structures

(h) FET Rule Used in Decompyle++

```
def error_3(...):
  if c: return
  FET_null()
  try: ...
  except: ...
```

(i) PYFET Fix for Error in Decompyle++

Figure 18. PYFET Fixes for Uncompyle6, Unpyc37, and Decompyle++

the decompiler *incorrectly skips an instruction* after `return` which is `SETUP_EXCEPT` (for `try-except`). As a result, it fails to recognize the following `try/except` block, causing the decompilation error. We notice that the skipping behavior is due to an extra invocation of `bc_next` function (line 1984 in "`ASTree.cpp`" for the case of `RETURN_VALUE`). Hence, we remove the line. This fixes the error and we do not observe particular side-effects yet.

```
if c1 and c2 <= c3 <= c4:
   ...
```

(a) Error Inducing Code in Unpyc37

```
 if c: return
 try: ...
```

(b) Error in Decompyle++

Figure 19. Decompilation Errors from Unpyc37 and Decompyle++

10.3. PYFET Rules Fixing Errors.

We show in Fig. 18 how PYFET fixes the errors in the three decompilers (`Uncompyle6`, `Unpyc37`, and `Decompyle++`).

11. Additional Case Studies

11.1. Decompiling Real-world Malware.

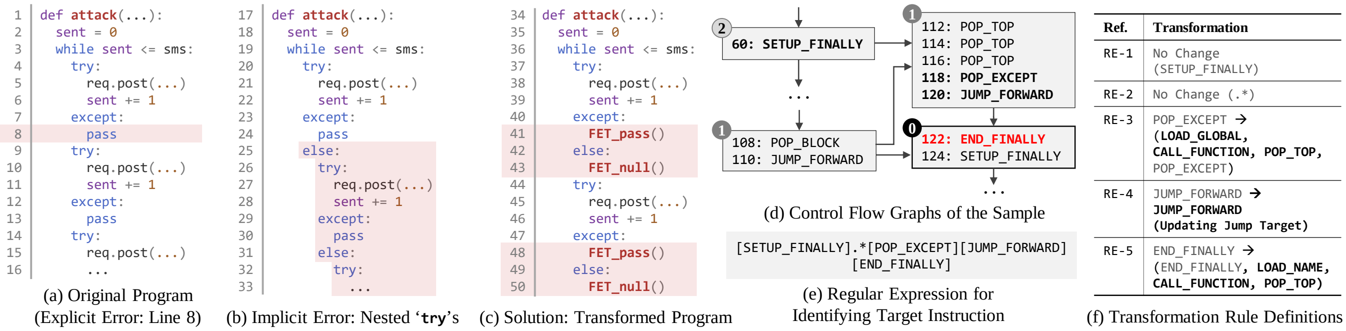
In our collected samples, we discover a malware sample that spams SMS messages. As it is `PyInstaller` compressed, we use `PyInstaller Extractor` [1] to extract `.pyc` files.

Observed Explicit and Implicit Errors. We observe two different types of errors from two different decompilers. First, `Decompyle3` threw an explicit error that fails the decompilation process at line 8 of Fig. 20-(a). Second, as shown in Fig. 20-(b), `Uncompyle6` introduces an `else` block at the end of each `try` block (at lines 25 and 31), incorrectly translating the `try` blocks that are in the same level to the *nested* blocks in the decompiled program. Observe that three `try` blocks (lines 4~8, 9~13, and 14~16) are not nested in Fig. 20-(a), while the corresponding three `try` blocks in Fig. 20-(b) are nested: the second `try` (lines 26~31) is under the `else` block of the first `try` (lines 20~25), and the third `try` (lines 32~33) is under the second `try` block.

Solutions. While there are two different errors, we find that a single FET can resolve both of them. This is because they both are related to the `try-except` and `else`. Fig. 20-(c) shows a source code representation of the solution (i.e., desirable FET applied program). Observe that it makes 2 changes from line 8 in (a) to lines 41~43 in (c). First, `pass` is transformed to `FET_pass()` (line 41). Second, it introduces an `else` block with `FET_null()` (no-op) as shown in lines 42~43. Our solution may seem counter-intuitive as we solve

TABLE 14. PYFET'S TRANSFORMATION RULES FOR CONVERTING PYTHON VERSION 3.9 TO 3.8.

	Name	Original Stmt.	Transformation	Description
R18	Comparison Operator	[IS_OP, CONTAINS_OP]	[COMPARE_OP]	"is", "in", "is not", and "not in" operator equivalent.
R19	Exception Matching	[JUMP_IF_NOT_EXC_MATCH]	[COMPARE_OP][POP_JUMP_IF_FALSE]	Checks whether the exception matches.
R20	Raising Exceptions	[RERAISE]	[END_FINALLY]	Raises exception with raise.
R21	Nested try/ except	[POP_EXCEPT].+[JUMP_FORWARD].* [RERAISE][RERAISE]	[BEGIN_FINALLY].+[END_FINALLY] [POP_EXCEPT][JUMP_FORWARD][END_FINALLY]	Nested try/except in except block.
R22	Raising Exceptions in except	[SETUP_FINALLY].*[POP_BLOCK] [POP_EXCEPT].*[JUMP_FORWARD].* [RERAISE].*[RERAISE]	[SETUP_FINALLY].*[POP_BLOCK] [BEGIN_FINALLY].*[END_FINALLY] [POP_EXCEPT][JUMP_FORWARD][END_FINALLY]	Raising exception with raise in except block.
R23	Dictionary Operation	[DICT_MERGE]	[BUILD_MAP_UNPACK_WITH_CALL]	Merging two dictionaries.
R24	Dictionary Operation	[BUILD_MAP].+[DICT_UPDATE].+ [BUILD_CONST_KEY_MAP][DICT_UPDATE]	.+[BUILD_CONST_KEY_MAP] [BUILD_MAP_UNPACK]	Dictionary initialization stmt.
R25	Loading Assertion	[LOAD_ASSERTION_ERROR]	[LOAD_GLOBAL]	Loads AssertionError.
R26	Handling with Block	[SETUP_WITH].+[POP_BLOCK].* [WITH_EXCEPT_START] [POP_JUMP_IF_TRUE][RERAISE]	[SETUP_WITH].+[POP_BLOCK] [BEGIN_FINALLY][WITH_CLEANUP_START] [WITH_CLEANUP_FINISH][END_FINALLY]	with block implementation with internal cleanup.
R27	Only try/ finally	[SETUP_FINALLY].*[POP_BLOCK].* [JUMP_FORWARD].*[RERAISE]	[SETUP_FINALLY].*[POP_BLOCK] [BEGIN_FINALLY].*[END_FINALLY]	try/finally without except block.
R28	Variable Arguments	[LOAD_GLOBAL, LOAD_FAST].* [BUILD_LIST][LOAD_FAST] [BUILD_MAP][LOAD_FAST] .*[CALL_FUNCTION_EX]	[LOAD_GLOBAL, LOAD_FAST].* [BUILD_TUPLE][LOAD_FAST] [BUILD_TUPLE_UNPACK_WITH_CALL] [LOAD_FAST][CALL_FUNCTION_EX]	Variable number of arguments, with and without keywords, can be passed.
R29	Initialize Lists	[BUILD_LIST][LOAD_CONST]	[LOAD_CONST]+[BUILD_LIST]	List initialization
R30	Initialize Sets	[BUILD_SET][LOAD_CONST]	[LOAD_CONST]+[BUILD_SET]	Set initialization



```
1   def attack(...):
2     sent = 0
3     while sent <= sms:
4       try:
5         req.post(...)
6         sent += 1
7       except:
8         pass
9       try:
10        req.post(...)
11        sent += 1
12      except:
13        pass
14      try:
15        req.post(...)
16        ...
```

(a) Original Program (Explicit Error: Line 8)

```
17  def attack(...):
18    sent = 0
19    while sent <= sms:
20      try:
21        req.post(...)
22        sent += 1
23      except:
24        pass
25      else:
26        try:
27          req.post(...)
28          sent += 1
29        except:
30          pass
31        else:
32          try:
33            ...
```

(b) Implicit Error: Nested '**try**'s

```
34  def attack(...):
35    sent = 0
36    while sent <= sms:
37      try:
38        req.post(...)
39        sent += 1
40      except:
41        FET_pass()
42      else:
43        FET_null()
44      try:
45        req.post(...)
46        sent += 1
47      except:
48        FET_pass()
49      else:
50        FET_null()
```

(c) Solution: Transformed Program

(d) Control Flow Graphs of the Sample

```
[SETUP_FINALLY].*[POP_EXCEPT][JUMP_FORWARD]
[END_FINALLY]
```

(e) Regular Expression for Identifying Target Instruction

Ref.	Transformation
RE-1	No Change (SETUP_FINALLY)
RE-2	No Change (.*)
RE-3	POP_EXCEPT → (**LOAD_GLOBAL, CALL_FUNCTION, POP_TOP,** POP_EXCEPT)
RE-4	JUMP_FORWARD → **JUMP_FORWARD (Updating Jump Target)**
RE-5	END_FINALLY → (END_FINALLY, **LOAD_NAME, CALL_FUNCTION, POP_TOP)**

(f) Transformation Rule Definitions

Figure 20. Transformation (FET) for both Implicit and Explicit Errors

TABLE 15. NEW/REMOVED INSTRUCTIONS IN PYTHON 3.9 FROM 3.8

New	RERAISE, LOAD_ASSERTION_ERROR, IS_OP, CONTAINS_OP, LIST_TO_TUPLE, LIST_EXTEND, JUMP_IF_NOT_EXC_MATCH, DICT_MERGE, WITH_EXCEPT_START, DICT_UPDATE, SET_UPDATE
Removed	WITH_CLEANUP_START, WITH_CLEANUP_FINISH, BUILD_LIST_UNPACK, BUILD_MAP_UNPACK, BUILD_MAP_UNPACK_WITH_CALL, BUILD_TUPLE_UNPACK, BUILD_TUPLE_UNPACK_WITH_CALL, BEGIN_FINALLY, CALL_FINALLY, POP_FINALLY, END_FINALLY

the implicit error introducing else blocks by introducing our own else blocks. This is because the decompiler does not try to add a bogus else block when there is an existing else block.

Applying FET. As shown in Fig. 20-(d), PYFET starts the process from the basic block 0 which contains the error-inducing instruction (122: END_FINALLY). One rule matched and was applied, but failed to resolve the error. Hence, PYFET looks for additional target blocks marked by 1. At this point, two rules matched, but none resolved the error. Then, PYFET looks for more target blocks and finds blocks marked by 2. Among them, the block containing instruction at 60 (SETUP_FINALLY) matches with a FET's regular expression shown in Fig. 20-(e). Specifically, the instructions at 60~122 are matched. Fig. 20-(f) shows the transformation rules. RE-3 and RE-5 add FET_pass() and FET_null() at lines 41 and 43 in Fig. 20-(c), respectively. RE-4 updates the argument of instruction at 110

(JUMP_FORWARD) to add the else block at line 42 in Fig. 20-(c). Note that there are 16 instances of the same patterns, hence we apply the same FET 16 times to resolve all the errors in the input binary.

Conclusion. From the decompiled source code, we find that the sample requests various APIs (e.g., api.sunlight.net and app-api.kfc.ru) to send an SMS to a victim. The implicit errors result in an incorrectly decompiled program with false dependencies between API requests for sending SMS messages. This is misleading because the dependencies imply that the malware sends or does not send an SMS message depending on the previous API's result.

Note that based on the recovered source code, we search publicly available repositories/reports. However, we did not find any contents that mention the sample.

Opcode mnemonic	Org. Opcode #	Modified Opcode #
RETURN_VALUE	83	**13**
UNARY_CONVERT	13	**83**
CALL_FUNCTION	131	111
DUP_TOP	4	64
MAP_ADD	147	161
BINARY_XOR	65	55
END_FINALLY	88	18

(a) Opcode Remapping

```
.*[#13]$
```

(b) Regular Expression that detects the Opcode Remapped Binary

Trans. Rule	Org. Opcode #
RETURN_VALUE	[#13]→**[#83]**
UNARY_CONVERT	[#83]→**[#13]**
CALL_FUNCTION	[#111]→**[#131]**
DUP_TOP	[#64]→**[#4]**

(c) Transformation Rules

Figure 21. Binary with Opcode Remapping (inSync, Python 2.7)

11.2. Opcode Remapped Python Binary (inSync).

Fig. 21-(a) shows how the opcode numbers are redefined. For example, the RETURN_VALUE instruction's opcode number is '83' in a default Python environment, while it is '13' in the modified druva's Python environment. When a decompiler processes a druva binary and encounters an opcode number '13', the decompiler will treat it as a UNARY_CONVERT instruction.

Detecting Opcode Remapped Binaries. To detect a binary with the remapped binaries, we use a regular expression shown in Fig. 21-(b), which is essentially finding a function's signature in the opcode remapped binaries. Note that UNARY_CONVERT in the normal Python environment has an opcode 13, which essentially means RETURN_VALUE in the modified environment. Hence, we are essentially looking for a binary containing a function that ends with RETURN_VALUE in the modified environment. In a usual Python binary, this will look for a function that ends with UNARY_CONVERT, which will very unlikely happen intuitively[9].

Applying Transformation Rules. Fig. 21-(c) shows a few transformation rules, which is a set of rules that simply translate the modified opcode numbers back to the original opcodes. For example, the druva Python environment changed the opcode for CALL_FUNCTION from 131 to 111. The transformation rule to rollback this modification is translating the opcode number 111 to 131 (i.e., [#111]→[#131]).

12. Examples of PyFET Transformations in Real-world Applications.

In this section, we present examples of our transformations for each rule, obtained from real-world applications (randomly chosen from the top 10 popular applications we presented in Table 12 (Section 9.2)). We provide 1,200 more raw samples of the examples on [2].

12.1. FET Rules R1~R16. We present two examples for each rule. For each example, we show the original code (on the left) and the transformed code (on the right) side by side. The differences (i.e., transformed code) are highlighted. Note that some transformations remove the code, which we visualize by highlighting a blank line. Fig. 22~Fig. 37 shows the examples for R1~R16, respectively.

12.2. FET Rules R17~R30. FET rules from R17 to R30 are for migrating Python 3.9 binaries to 3.8. Those rules transform bytecode instructions into semantically equivalent forms. They are syntactically equivalent transformations at the source level. They do not change source representations at all. Hence, we present examples of the target code in applications. The transformed code is identical to the target code, hence omitted. Fig. 38~Fig. 51 shows the examples for R17~R30, respectively.

[9] We have also not observed any functions ending with UNARY_CONVERT.

```
def main(...):
  py2_compat = (..)
  for path in sys.argv[1:]:
    for text in lines:
      if text.start(b‘from’)\
        or text == b‘__metaclass’ \
        or text == b‘annotations’ \
        or text == b‘__future__import’:
        future.append(text.decode())
      if missing:
        ...
```

(a) Original Program #1

```
def main(...):
  py2_compat = (..)
  for path in sys.argv[1:]:
    for text in lines:
      FET_cond = text.start(b‘from’)\
        or text == b‘__metaclass’ \
        or text == b‘annotations’ \
        or text == b‘__future__import’:
        future.append(text.decode())
      if FET_cond:
        future.append(text.decode())
      if missing:
        ...
```

(b) Transformed Output #1

```
def audio_dataset_from_directory(...):
  if sampling_rate is not None:
    if not isinstance(sampling_rate):
      raise ValueError()
  ...
  if labels is None or \
    label_mode is None or \
    sampling_rate is None or ragged:
    labels = None
    label_mode = None
  ...
```

(c) Original Program #2

```
def audio_dataset_from_directory(...):
  if sampling_rate is not None:
    if not isinstance(sampling_rate):
      raise ValueError()
  ...
  FET_cond = labels is None or \
    label_mode is None or \
    sampling_rate is None or ragged
  if FET_cond:
    labels = None
    label_mode = None
  ...
```

(d) Transformed Output #2

Figure 22. Example Transformations for `R1` `ansible` and `keras`

```
def placeholder(...):
  if tf.compat.v1.executing_eagerly():
    if sparse:
      spec = tf.SparseTensorSpec(\
      shape=shape, dtype=dtype)
    elif ragged:
      ragged_rank = 0
    else:
      spec = tf.TensorSpec(shape=shape,\
      dtype=dtype, name=name)
  ...
```

(a) Original Program #1

```
def placeholder(...):
  if tf.compat.v1.executing_eagerly():
    if sparse:
      spec = tf.SparseTensorSpec(\
      shape=shape, dtype=dtype)
    elif ragged:
      ragged_rank = 0
    if not ragged and not sparse:
      spec = tf.TensorSpec(shape=shape,\
      dtype=dtype, name=name)
  ...
```

(b) Transformed Output #1

```
def get_detail_sentence(...):
  if not CP.notCar:
    sentence_builder = “...”
    ...
  else:
    if CP.carFingerprint == "COMMA":
      return ...
    ...
```

(c) Original Program #2

```
def get_detail_sentence(...):
  if not CP.notCar:
    sentence_builder = “...”
    ...
  if CP.notCar:
    if CP.carFingerprint == "COMMA":
      return ...
    ...
```

(d) Transformed Output #2

Figure 23. Example Transformations for `R2` in `keras` and `openpilot`

```
def calculate_hash_from_tarball(...):
  ...
  while True:
    chunk = await tar.read(1024)
    if not chunk:
      break
    tar_hash.update(chunk)
  return tar_hash.hexdigest()
```

(a) Original Program #1

```
def calculate_hash_from_tarball(...):
  ...
  while True:
    chunk = await tar.read(1024)
    if not chunk:
      break

  return tar_hash.hexdigest()
```

(b) Transformed Output #1

```
def find_answers_by_keyword(...):
  ...
  for answer_dict in answer_dicts:
    answer = ans_dict.get(‘answer’, ’’)
    if match(...):
      answers_found.append(ans_dict)
    if len(answers) > CONFIG[‘search’]:
      break
    answers_found.append(...)
  ...
```

(c) Original Program #2

```
def find_answers_by_keyword(...):
  ...
  for answer_dict in answer_dicts:
    answer = ans_dict.get(‘answer’, ’’)
    if match(...):
      answers_found.append(ans_dict)
    if len(answers) > CONFIG[‘search’]:
      break

  ...
```

(d) Transformed Output #2

Figure 24. Example Transformations for `R3` `ansible` and `cheat.sh`

```
def check_params(...):
  for params_name in params:
    for fn in legal_params_fns:
      if has_arg(fn, params_name):
        res = self.sk_params.copy()
        res.update(...)
        if not fn:
          break

      else:
        ...
```

(a) Original Program #1

```
def check_params(...):
  for params_name in params:
    for fn in legal_params_fns:
      if has_arg(fn, params_name):
        res = self.sk_params.copy()
        res.update(...)
        if not fn:
          break
        FET_null()
      else:
        ...
```

(b) Transformed Output #1

```
def pretty_flags(...):
  for i in range(32):
    if flags & flag:
      names_pretty_flags.append(...)
      flags ^= flag
      if not flags:
        break

    else:
      ...
```

(c) Original Program #2

```
def pretty_flags(...):
  for i in range(32):
    if flags & flag:
      names_pretty_flags.append(...)
        flags ^= flag
        if not flags:
          break
        FET_null()
    else:
      ...
```

(d) Transformed Output #2

Figure 25. Example Transformations for `R4` `keras` and `XX-Net`

```
def getPublicKey(...):
  if not os.path.isfile():
    return None
  with open(PERSIST_pubkey_my + \
    '/comma/id_rsa.pub') as f:
  return f.read()
```

(a) Original Program #1

```
def getPublicKey(...):
  if not os.path.isfile():
    return None
  f = open(PERSIST_pubkey_my + \
    '/comma/id_rsa.pub’)
  return f.read()
```

(b) Transformed Output #1

```
def save_file(...):
  mode = "a" if append else "w+"
  if not isinstance(content, str):
    mode = mode + "b"
    ...
  with open(file, opener=_opener if \
    permissions else None) as f:
    f.write(content)
  return f.flush()
```

(c) Original Program #2

```
def save_file(...):
  mode = "a" if append else "w+"
  if not isinstance(content, str):
    mode = mode + "b”
    ...
  f = open(file, opener=_opener if \
    permissions else None)
  f.write(content)
  return f.flush()
```

(d) Transformed Output #2

Figure 26. Example Transformations for `R5` in `openpilot` and `localstack`

```
def create_metric_coverage_docs(...):
  ...
  for service in sorted(simp.keys()):
    ...
    implemented_ops = {operation[0]:\
      operation[1] for operation in \
      details.items() if not operation_1}
    ...
```

(a) Original Program #1

```
def create_metric_coverage_docs(...):
  ...
  for service in sorted(simp.keys()):
    ...
    implemented_ops = foo()
    ...
def foo():
  return {operation[0]: operation[1] \
    for operation in details.items() \
    if not operation_1}
```

(b) Transformed Output #1

```
def ensure_running(...):
  with self._lock:
    cmd = (...)
    if self._preload_modules:
      ...
      data = {x: y for x, y in \
        data.items() if x in keys}
...
```

(c) Original Program #2

```
def ensure_running(...):
  with self._lock:
    cmd = (...)
    if self._preload_modules:
      ...
      data = foo()
...
def foo():
  return {x: y for x, y in data.items() \
    if x in keys}
```

(d) Transformed Output #2

Figure 27. Example Transformations for `R6` in `localstack` and `XX-Net`

```
def delete_function_event_invoke_cfg(...):
  region = LambdaRegion.get()
  try:
    function_arn = func_arn(function)
    ...
  except Exception as e:
    return
```

(a) Original Program #1

```
def delete_function_event_invoke_cfg(...):
  region = LambdaRegion.get()
  FET_raise = 0
  try:
    function_arn = func_arn(function)
    ...
  except Exception as e:
    FET_raise = 1
  else:
    FET_null()
  if FET_raise == 1:
    return
```

(b) Transformed Output #1

```
def safe_getattr(...):
  try:
    attr_name = safe_to_getattr(obj)
  except Exception as error:
    return (error, None)
```

(c) Original Program #2

```
def safe_getattr(...):
  FET_raise = 0
  try:
    attr_name = safe_to_ getattr(obj)
  except Exception as error:
    FET_raise = 1
  else:
    FET_null()
  if FET_raise == 1:
    return (error, None)
```

(d) Transformed Output #2

Figure 28. Example Transformations for `R7` in `localstack` and `rich`

```
def append(...):
  if isinstance(text, str):
    sanitized_text = strip_control(text)
    ...
  elif isinstance(text, Text):
    _Span = Span
    for style in styles:
      ...
      self._length += len(text)

  else:
    ...
```

(a) Original Program #1

```
def append(...):
  if isinstance(text, str):
    sanitized_text = strip_control(text)
    ...
  elif isinstance(text, Text):
    _Span = Span
    for style in styles:
      ...
      self._length += len(text)
      FET_null()
  else:
    ...
```

(b) Transformed Output #1

```
def build(...):
  ...
  if self.cell.format == "channels_first":
    ch_dim = 1
  elif self.cell.format == "channels_last":
    for step in output:
      ch_dim = self.rank + 1

  else:
    ...
```

(c) Original Program #2

```
def build(...):
  ...
  if self.cell.format == "channels_first":
    ch_dim = 1
  elif self.cell.format == "channels_last":
    for step in output:
      ch_dim = self.rank + 1
      FET_null()
  else:
    ...
```

(d) Transformed Output #2

Figure 29. Example Transformations for `R8` in `rich` and `keras`

```
def bfs(...):
  ...

  while queue:
    node = queue.popleft()
    print(node)
    if dest is node:

      break

  return False

```

(a) Original Program #1

```
def bfs(...):
  ...
  FET_cond = queue
  while FET_cond:
    node = queue.popleft()
    print(node)
    if dest is node:
      FET_cond = queue
      break
    FET_cond = queue
  return False

```

(b) Transformed Output #1

```
def resolve_apis (...):
  ...

  while stack:
    resolve_service = stack.pop()

    if service in result:

    continue
      ...

  ...
```

(c) Original Program #2

```
def resolve_apis (...):
  ...
  FET_cond = stack
  while FET_cond:
    resolve_service = stack.pop()

    if service in result:
      FET_cond = stack
      continue
      ...
    FET_cond = stack
    ...
```

(d) Transformed Output #2

Figure 30. Example Transformations for `R9` in `jieba` and `localstack`

```
def _maybe_build(...):
  ...
  for input_list in compute_dtype:

    try:
      dtype = input_list[0].dtype

    except AttributeError:
      pass
    else:
      if isinstance(values): continue
  ...
```

(a) Original Program #1

```
def _maybe_build(...):
  ...
  for input_list in compute_dtype:
    FET_else = 0
    try:
      dtype = input_list[0].dtype
      FET_else = 1
    except AttributeError:
      pass
    if FET_else == 1:
      if isinstance(values): continue
  ...
```

(b) Transformed Output #1

```
def lib2to3_parse(...):
  ...
  for grammar in grammars:
    ...

    try:
      ...

    except ParseError as pe:
      ...
    else:
      lineno, column = te.args[1]
      errors[grammar] = InvalidInput(...)
      continue
  ...
```

(c) Original Program #2

```
def lib2to3_parse(...):
  ...
  for grammar in grammars:
    ...
    FET_else = 0
    try:
      ...
      FET_else = 1
    except ParseError as pe:
      ...
    if FET_else == 1:
      lineno, column = te.args[1]
      errors[grammar] = InvalidInput(...)
      continue
  ...
```

(d) Transformed Output #2

Figure 31. Example Transformations for `R10` in `keras` and `jieba`

```
class wrapper(*arg, **kwargs):
  ...
```

(a) Original Program

```
class wrapper(FET_one_star_arg, \
  FET_two_star_kwargs):
  ...
```

(b) Transformed Output

```
class get_flashvar(x, *arg, **kwargs):
  ...
```

(c) Original Program

```
class get_flashvar(x, FET_one_star_arg, \
  FET_two_star_kwargs):
  ...
```

(d) Transformed Output

Figure 32. Example Transformations for `R11` in `pandas` and `youtube-dl`

```
...
from keras.optimizers.optimizer_v2 import
(
  gradient_descent as gradient_des_v2,
  adamax as adamax_v2
)


...
```

(a) Original Program #1

```
...
from keras.optimizers.optimizer_v2 import
(
  gradient_descent as gradient_des_v2
)
from keras.optimizers.optimizer_v2 import
(
  adamax as adamax_v2
)
...
```

(b) Transformed Output #1

```
...
from .a.b.c.subprocess import (
  LongNameAndAllOfItsLetters1 as let1,
  LongNameAndAllOfItsLetters2 as let2
)


...
```

(c) Original Program #2

```
...
from .a.b.c.subprocess import (
  LongNameAndAllOfItsLetters1 as let1
)
from .a.b.c.subprocess import (
  LongNameAndAllOfItsLetters2 as let2
)
...
```

(d) Transformed Output #2

Figure 33. Example Transformations for `R12` in `keras` and `jieba`

```
def test_rotation(...):
  ...
  expected_files = {"rlog", "qlog"}
  ...
```

(a) Original Program #1

```
def test_rotation(...):
  ...
  expected_files = FET_set("rlog", "qlog")
  ...
```

(b) Transformed Output #1

```
def _parse_options(...):
  ...
  search_options = {'insensitive',\
  'word_boundaries', 'recursive',}
  ...
```

(c) Original Program #2

```
def _parse_options(...):
  ...
  search_options = FET_set('insensitive',\
  'word_boundaries', 'recursive',)
  ...
```

(b) Transformed Output #1

Figure 34. Example Transformations for `R13` in `openpilot` and `cheat.sh`

```
def visit_default(...):
  ...
  if isinstance(node, Node):
    ...
    for child in node.children:
      yield from self.visit(child)
    ...
```

(a) Original Program #1

```
def visit_default(...):
  ...
  if isinstance(node, Node):
    ...
    for child in node.children:
      FET_yield_from(self.visit(child))
    ...
```

(b) Transformed Output #1

```
def fixture_snapshot(...):
    ...
    sm.add_transformer(SNAPSHOT_,\
    priority=2)
    yield from sm
    ...
```

(c) Original Program #2

```
def fixture_snapshot(...):
    ...
    sm.add_transformer(SNAPSHOT,\
    priority=2)
    FET_yield_from(sm)
    ...
```

(d) Transformed Output #2

Figure 35. Example Transformations for `R14` in `jieba` and `localstack`

```
def __init__(...):
    ...
    self._points = {ilabel:stack_copy\
(_start_point) for ilabel in ilabels}
    ...
```

(a) Original Program #1

```
def __init__(...):
    ...
    self._points = [(ilabel,stack_copy\
(_start_point)) for ilabel in ilabels]
    self._points = dict(self._points)
    ...
```

(b) Transformed Output #1

```
def _repr_mimebundle_(...):
    ...
    if include:
        data = {k: v for (k, v) in \
        data.items() if k in include}
```

(c) Original Program #2

```
def _repr_mimebundle_(...):
    ...
    if include:
        data = [(k, v) for (k, v) in \
        data.items() if k in include]
        data = dict(data)
```

(d) Transformed Output #2

Figure 36. Example Transformations for `R15` in `jieba` and `rich`

```python
def can_connect(...) :
    ...
    if error_classes is None:
        ...
        return False

    try:
        _path = f"__{rands(10)}__.pickle"
    except error_classes:
        return False
    else:
        return True
    ...
```

(a) Original Program #1

```python
def can_connect(...):
    ...
    if error_classes is None:
        ...
        return False
    FET_null()
    try:
        _path = f"__{rands(10)}__.pickle"
    except error_classes:
        return False
    else:
        return True
    ...
```

(b) Transformed Output #1

```python
def uses_keras_history(...):
    ...
    if getattr(tensor, "_keras_history",\
        None) is not None:
        return True

    try:
        new_tensors_to_check.extend(...)
    except AttributeError:
        pass
    ...
```

(c) Original Program #2

```python
def uses_keras_history(...):
    ...
    if getattr(tensor, "_keras_history",\
        None) is not None:
        return True
    FET_null()
    try:
        new_tensors_to_check.extend(...)
    except AttributeError:
        pass
    ...
```

(d) Transformed Output #2

Figure 37. Example Transformations for R16 in keras and rich

```python
def can_omit_invisible_parens(...):
  ...
  assert len(line.leaves) >= 2, "Stranded delimiter"
```

(a) Program Template # 1 (R17)

```python
def _get_cache_fn(...):
  ...
  assert re.match(r'^[a-zA-Z0-9_.-]+$', key), 'invalid
                key %r' % key
  return os.path.join(...)
```

(b) Program Template # 2 (R17)

Figure 38. Example for R17 in jieba and youtube-dl

```python
def remove_color(...):
  for text, style, control in segments:
        if style:
            colorless_style = cache.get(style)
            if colorless_style is None:
                  ...
```

(a) Program Template #1 (R18)

```python
def _server_time (...):
  if self.__server_time is not None:
        return self.__server_time
  ...
```

(b) Program Template #2 (R18)

Figure 39. Example for R18 in rich and youtube-dl

```python
def _get_signature(...):
  try:
      ...
  except ValueError:
      _signature = "(...)”
  ...
```

(a) Program Template #1 (R19)

```python
def _html_wrapper(...):
  try:
      ...
  except FileNotFoundError:
      print("ERROR: %s" % cmd)
      raise
  ...
```

(b) Program Template #2 (R19)

Figure 40. Example for R19 in rich and cheat.sh

```python
async def asyncSetUp(...):
  events.append('asyncSetUp’)
  self.addAsyncCleanup(self.on_cleanup)
  raise MyException()
```

(a) Program Template #1 (R20)

```python
def get_callable(...):
  ...
  if not obj:
   raise ImportError(f'Could not import "{str}”’)
```

(b) Program Template #2 (R20)

Figure 41. Example for R20 in XX-Net and pandas

```python
def get_server_version_from_running_container(...):
  try:
     container_name = get_main_container_name()
     ...
  except ContainerException as e:
     try:
        img_name = get_docker_image_to_start()
     except ContainerException:
        ...
```

(a) Program Template #1 (R21)

```python
def load(...):
  ...
  try:
     json.load(cachef)
  except ValueError:
     try:
        file_size = os.path.getsize(cache_fn)
     except (OSError, IOError) as oe:
        ...
```

(b) Program Template #2 (R21)

Figure 42. Example for R21 in localstack and youtube-dl

```python
def lambda_function_or_layer_arn (...):
  ...
  try:
     alias_response = client.get_alias(...)
     version = alias_response["FunctionVersion"]
  except ContainerException as e:
     ...
     raise Exception(msg)

```

(a) Program Template #1 (R22)

```python
def compute_output_shape(...):
  ...
  try:
     if self.data_format == "channels_last":
        ...
     ...
  except ValueError as e:
     raise ValueError(...)
  ...
```

(b) Program Template #2 (R22)

Figure 43. Example for R22 in localstack keras

```python
def __init__(...):
  ...
  MIMENonMultipart.__init__(self, 'application',
_subtype, policy=policy, **_params)
```

(a) Program Template #1 (R23)

```python
def floats(...):
 ...
 st.floats(**kwargs, allow_nan=allow_nan,
           allow_infinity=allow_infinity)
 ...
```

(b) Program Template #2 (R23)

Figure 44. Example for R23 in XX-Net and openpilot

```
_AXIS_TO_AXIS_NUMBER: dict[Axis, int] = {
    **NDFrame._AXIS_TO_AXIS_NUMBER,
    1: 1,
    "columns": 1,}
```

(a) Program Template #1 (R24)

```
def get_callable(...):
  ...
  top_dict_ = {k: v for k, v in data.items() if not \
isinstance(v, dict)}
  ...
```

(b) Program Template #2 (R24)

Figure 45. Example for R24 in pandas and openpilot

```
def test_create_and_update_secret(...):
  ...
  assert len(secret_arn.rpartition("-")[-1]) == 6
  ...
```

(a) Program Template #1 (R25)

```
def test_header_split (...):
  for inp in unchanged.strip().splitlines():
      assert inp == _add_section_name(inp)
  ...
```

(b) Program Template #2 (R25)

Figure 46. Example for R25 in localstack and cheat.sh

```
...
with open('update/LATEST_VERSION', 'w') as f:
    f.write(version)
...
```

(a) Program Template #1 (R26)

```
def readinto(...):
  with memoryview(b) as view, view.cast("B") as byte_view:
    data = self.read(len(byte_view))
  ...
```

(b) Program Template #2 (R26)

Figure 47. Example for R26 in youtube-dl and XX-Net

```
def _switch_region(...):
  ...
  try:
      config.DEFAULT_REGION = region
      yield
  finally:
      config.DEFAULT_REGION = previous_region
```

(a) Program Template #1 (R27)

```
def test_rich_print(...):
  ...
  try:
      console.file = output
      ...
  finally:
      console.file = backup_file
```

(b) Program Template #2 (R27)

Figure 48. Example for R27 in localstack and rich

```
def fire(self, *args, **kwargs):
  ...
  h(*args, **kwargs)
```

(a) Program Template #1 (R28)

```
def f (*inps, **kwargs):
  ...
  return result(self, inp, shape, *inps, **kwargs)
```

(b) Program Template #2 (R28)

Figure 49. Example for R28 in ansible and openpilot

```
def test_device_fell(...):
 ...
 msg.sensorEvents[0].acceleration.v = [10.0, 0.0,
0.0]
 ...
```

(a) Program Template #1 (R29)

```
def selinux_initial_context(...):
 ...
 self._selinux_initial_context = [None, None, None]
 ...
```

(b) Program Template #2 (R29)

Figure 50. Example for R29 in openpilot and ansible

```
class TestLambdaAPI(...):
 ...
 TAGS = {"hello", "world", "env", "prod"}
```

(a) Program Template #1 (R30)

```
def set_floatx(...):
 ...
 accepted_dtypes = {"float16", "float32", "float64"}
 ...
```

(b) Program Template #2 (R30)

Figure 51. Example for R30 in localstack and keras

References

[1] "PyInstaller Extractor," https://github.com/extremecoders-re/pyinstxtractor, 2022.

[2] "PyFET repository: Correctness and Impact of PyFET." 2022, https://github.com/pyfet-pyc/src/tree/main/Correctness_and_Impact_PyFET.

[3] "SPARK: Scanning, Parsing, and Rewriting Kit," 2009, http://pages.cpsc.ucalgary.ca/~aycock/spark/.