Malware Analysis Final Project

Static Analysis CLI Tool

Report / Walkthrough

Khoa Do, Scott Hunt, Casey Schablein, Noah Trenaman

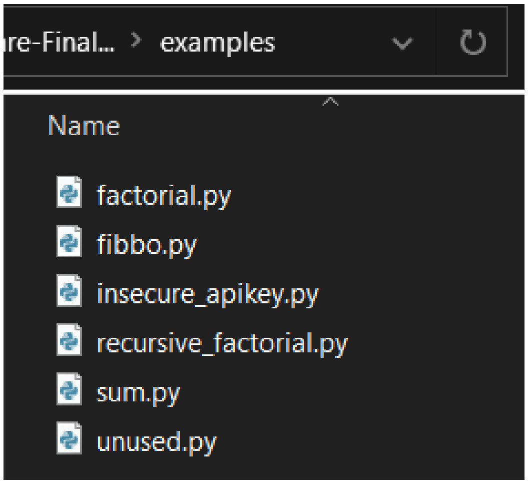
A Setup and Commands tldr is in the GitHub README: *https://github.com/khoaddo/Malware-Final-Project*

This document will mainly highlight each of the functions' services and expected outputs but will also go over setup. Since the tool was written in python, to run any of the commands you need to have python installed on your machine, so if you haven't done that already, do so now. You will also need to install the 'pandas' package.

Setup

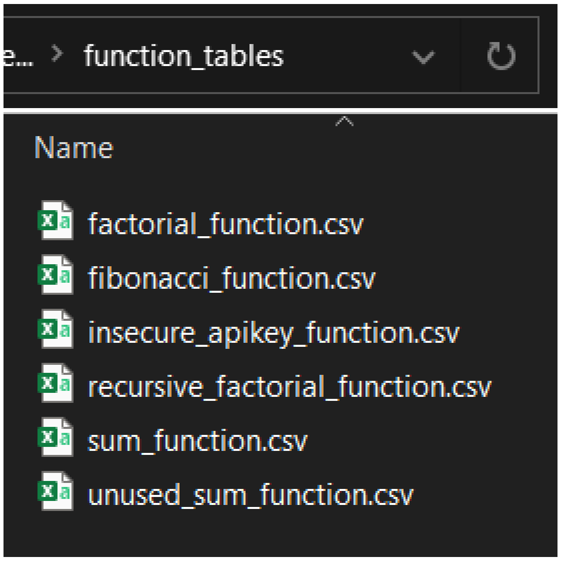
After extracting the zip file somewhere, you will see a few folders. One called '*examples'* and one called '*function_tables'*. Let's go over what each of them will contain.

As you can see below, 'examples' contains some premade sample scripts. For sake of simplicity, each script contains 1 function each, and is named respectively. The 'unused.py' script is a duplicate of 'sum.py' with the only difference of there being 3 unused variables declared at the beginning.



factorial.py
fibbo.py
insecure_apikey.py
recursive_factorial.py
sum.py
unused.py

Next up, the 'function_tables' folder. This contains a few csv files. These csv's contain the disassembled code of each function. Note that we did not make these ourselves, but they were generated by our program during runtime (don't believe me? Go ahead and delete them, run the program, and you will see).



factorial_function.csv
fibonacci_function.csv
insecure_apikey_function.csv
recursive_factorial_function.csv
sum_function.csv
unused_sum_function.csv

The program disassembles a given function using python's disassembly library, dis. We use this library in a function we created called function_table_to_bytecode_table. The function does 4 things:

- takes a given function,
- disassembles it into bytecode (using dis)
- iterates through each instruction to access all metadata,
- and finally returns a dataframe that represents the expanded disassembled instructions

We then use the dataframe this function returns to output into a human readable csv file. Once the csv files are saved into the 'function_table" folder, we can then analyze it by utilizing the rest of the main program.

Execution

Running the program is simply running the main.py file in the project's root directory:

```
python main.py
```

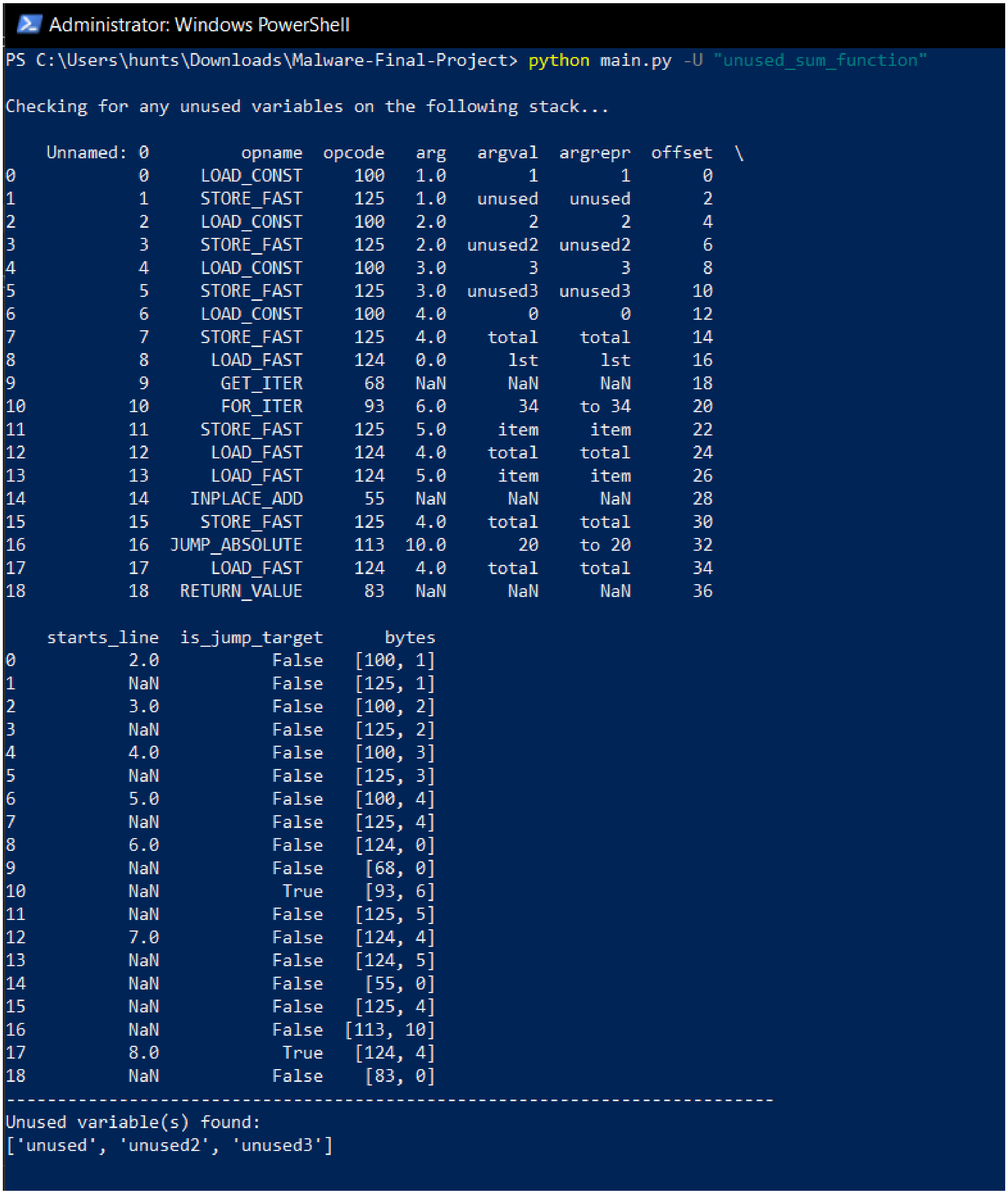
To choose which type of analysis to do, you need to provide 2 additional arguments. The first argument specifies which analysis to do. In the projects current state, there are 3 to choose from for this argument. However, in theory, this project could go open source where more functionalities could be added on indefinitely. The 3 current options are: U, -C or -R.

- -U: Checks if there are any unused variables on the stack
- -C: Gathers all the constant values on the stack
- -R: Checks if a function is recursive, returns the name if so

The second argument for running main.py is simply the name of the function you want disassembled/analyzed, i.e. "sum_function". So a full command to run would have the structure as follows:

```
python main.py -U "unused_sum_function"
```

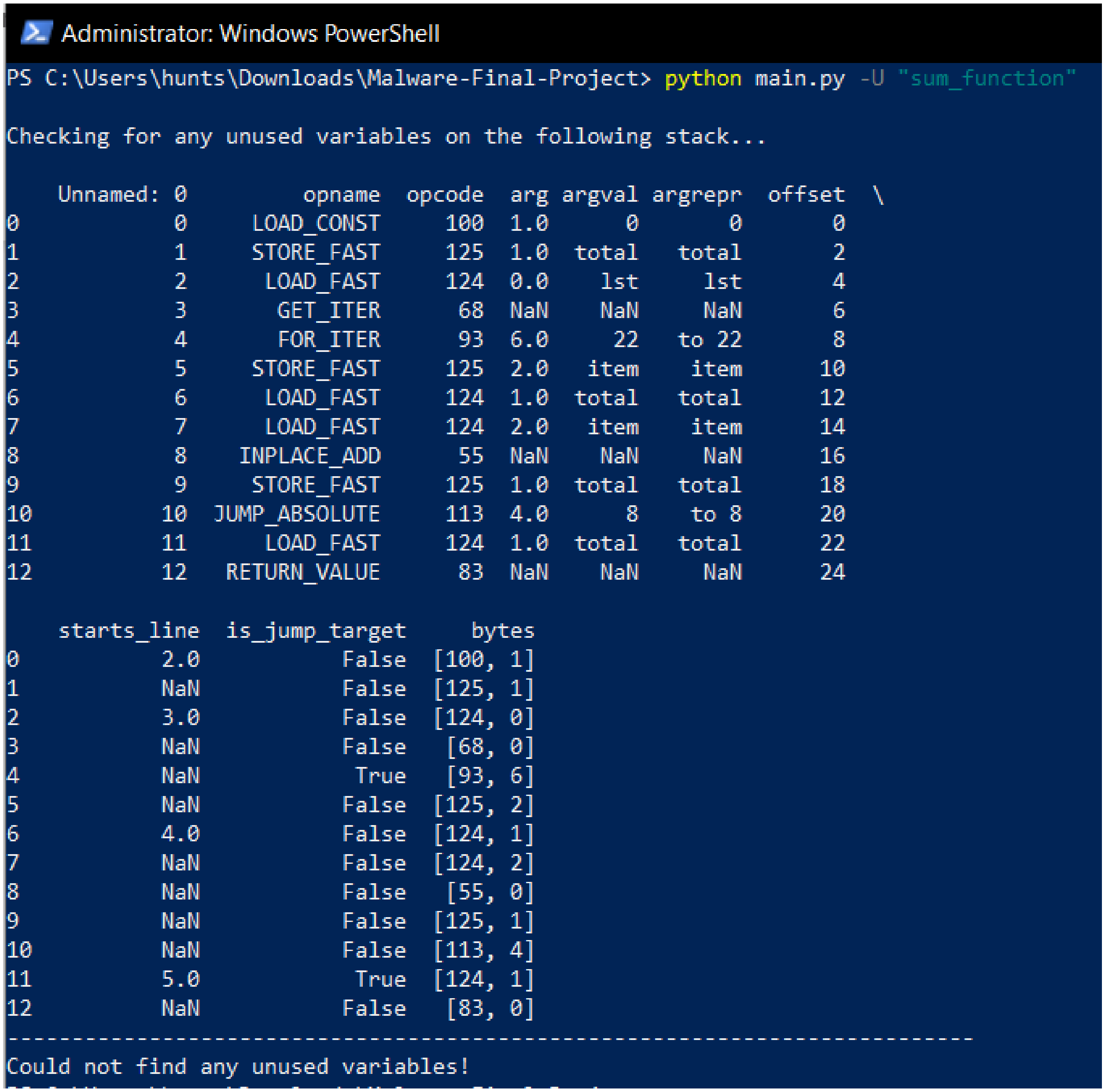
This is a real command you could run for this project and the output is as follows:



```
PS C:\Users\hunts\Downloads\Malware-Final-Project> python main.py -U "unused_sum_function"

Checking for any unused variables on the following stack...

    Unnamed: 0         opname  opcode   arg   argval  argrepr  offset  \
0            0     LOAD_CONST     100   1.0        1        1       0
1            1     STORE_FAST     125   1.0   unused   unused       2
2            2     LOAD_CONST     100   2.0        2        2       4
3            3     STORE_FAST     125   2.0  unused2  unused2       6
4            4     LOAD_CONST     100   3.0        3        3       8
5            5     STORE_FAST     125   3.0  unused3  unused3      10
6            6     LOAD_CONST     100   4.0        0        0      12
7            7     STORE_FAST     125   4.0    total    total      14
8            8      LOAD_FAST     124   0.0      lst      lst      16
9            9       GET_ITER      68   NaN      NaN      NaN      18
10          10       FOR_ITER      93   6.0       34    to 34      20
11          11     STORE_FAST     125   5.0     item     item      22
12          12      LOAD_FAST     124   4.0    total    total      24
13          13      LOAD_FAST     124   5.0     item     item      26
14          14    INPLACE_ADD      55   NaN      NaN      NaN      28
15          15     STORE_FAST     125   4.0    total    total      30
16          16  JUMP_ABSOLUTE     113  10.0       20    to 20      32
17          17      LOAD_FAST     124   4.0    total    total      34
18          18   RETURN_VALUE      83   NaN      NaN      NaN      36

    starts_line  is_jump_target      bytes
0           2.0           False   [100, 1]
1           NaN           False   [125, 1]
2           3.0           False   [100, 2]
3           NaN           False   [125, 2]
4           4.0           False   [100, 3]
5           NaN           False   [125, 3]
6           5.0           False   [100, 4]
7           NaN           False   [125, 4]
8           6.0           False   [124, 0]
9           NaN           False    [68, 0]
10          NaN            True    [93, 6]
11          NaN           False   [125, 5]
12          7.0           False   [124, 4]
13          NaN           False   [124, 5]
14          NaN           False    [55, 0]
15          NaN           False   [125, 4]
16          NaN           False  [113, 10]
17          8.0            True   [124, 4]
18          NaN           False    [83, 0]
---------------------------------------------------------------------------
Unused variable(s) found:
['unused', 'unused2', 'unused3']
```

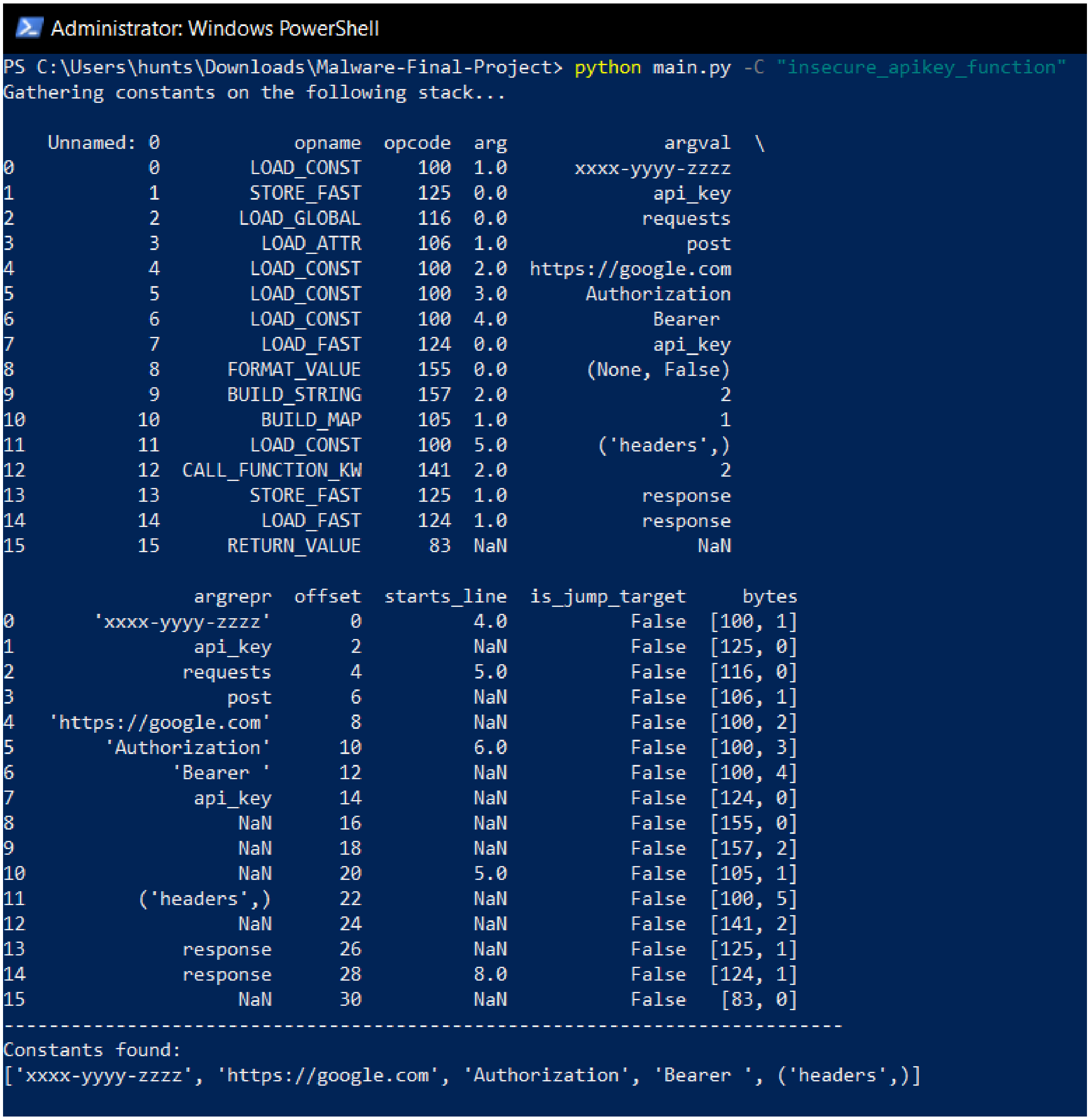
It successfully found all 3 unused variables in the function!

If there is a function that has no unused variables, the output will look like this:



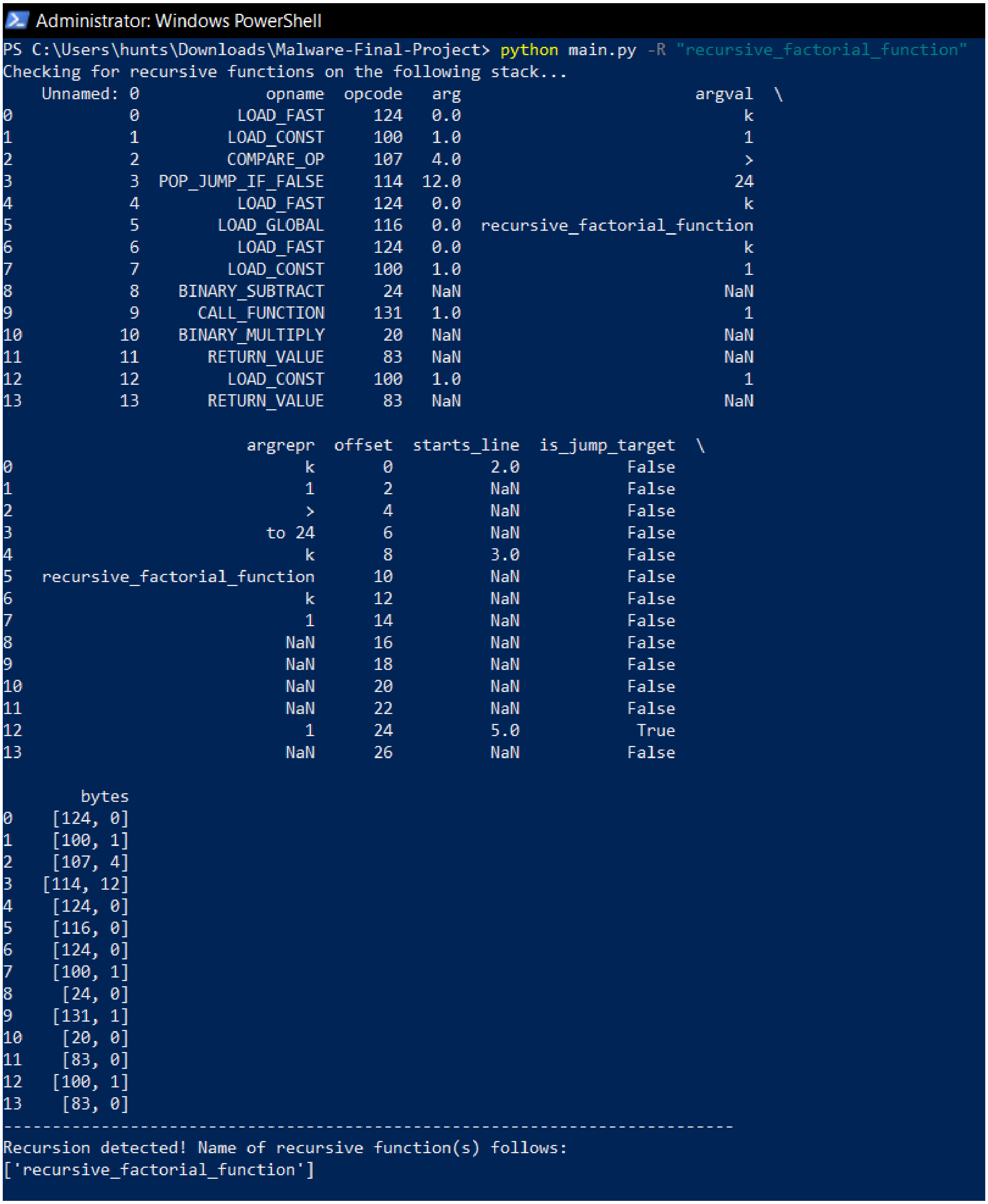
```
PS C:\Users\hunts\Downloads\Malware-Final-Project> python main.py -U "sum_function"

Checking for any unused variables on the following stack...

    Unnamed: 0         opname  opcode  arg argval argrepr  offset  \
0            0     LOAD_CONST     100  1.0      0       0       0
1            1     STORE_FAST     125  1.0  total   total       2
2            2      LOAD_FAST     124  0.0    lst     lst       4
3            3       GET_ITER      68  NaN    NaN     NaN       6
4            4       FOR_ITER      93  6.0     22   to 22       8
5            5     STORE_FAST     125  2.0   item    item      10
6            6      LOAD_FAST     124  1.0  total   total      12
7            7      LOAD_FAST     124  2.0   item    item      14
8            8    INPLACE_ADD      55  NaN    NaN     NaN      16
9            9     STORE_FAST     125  1.0  total   total      18
10          10  JUMP_ABSOLUTE     113  4.0      8    to 8      20
11          11      LOAD_FAST     124  1.0  total   total      22
12          12   RETURN_VALUE      83  NaN    NaN     NaN      24

    starts_line  is_jump_target     bytes
0           2.0           False  [100, 1]
1           NaN           False  [125, 1]
2           3.0           False  [124, 0]
3           NaN           False   [68, 0]
4           NaN            True   [93, 6]
5           NaN           False  [125, 2]
6           4.0           False  [124, 1]
7           NaN           False  [124, 2]
8           NaN           False   [55, 0]
9           NaN           False  [125, 1]
10          NaN           False  [113, 4]
11          5.0            True  [124, 1]
12          NaN           False   [83, 0]
---------------------------------------------------------------------------
Could not find any unused variables!
```

Next, let’s check to see what happens for constants with -C! We have created a function that has insecure api keys for this one, just to prove they are accessible, so let’s use that in the command:



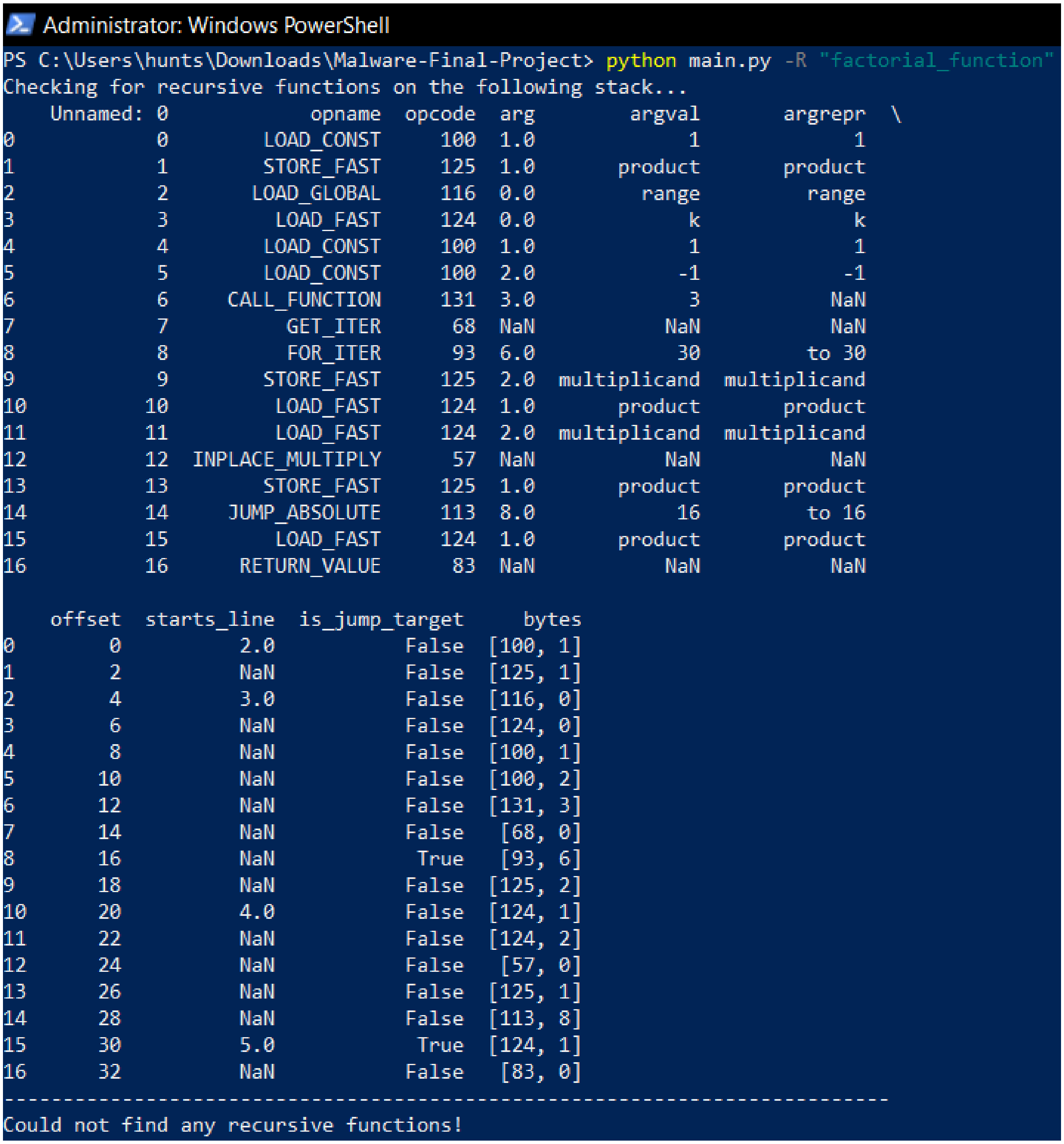
```
PS C:\Users\hunts\Downloads\Malware-Final-Project> python main.py -C "insecure_apikey_function"
Gathering constants on the following stack...

    Unnamed: 0            opname  opcode  arg               argval  \
0            0        LOAD_CONST     100  1.0       xxxx-yyyy-zzzz
1            1        STORE_FAST     125  0.0              api_key
2            2       LOAD_GLOBAL     116  0.0             requests
3            3         LOAD_ATTR     106  1.0                 post
4            4        LOAD_CONST     100  2.0   https://google.com
5            5        LOAD_CONST     100  3.0        Authorization
6            6        LOAD_CONST     100  4.0              Bearer 
7            7         LOAD_FAST     124  0.0              api_key
8            8      FORMAT_VALUE     155  0.0        (None, False)
9            9      BUILD_STRING     157  2.0                    2
10          10         BUILD_MAP     105  1.0                    1
11          11        LOAD_CONST     100  5.0         ('headers',)
12          12  CALL_FUNCTION_KW     141  2.0                    2
13          13        STORE_FAST     125  1.0             response
14          14         LOAD_FAST     124  1.0             response
15          15      RETURN_VALUE      83  NaN                  NaN

                 argrepr  offset  starts_line  is_jump_target     bytes
0       'xxxx-yyyy-zzzz'       0          4.0           False  [100, 1]
1                api_key       2          NaN           False  [125, 0]
2               requests       4          5.0           False  [116, 0]
3                   post       6          NaN           False  [106, 1]
4   'https://google.com'       8          NaN           False  [100, 2]
5        'Authorization'      10          6.0           False  [100, 3]
6              'Bearer '      12          NaN           False  [100, 4]
7                api_key      14          NaN           False  [124, 0]
8                    NaN      16          NaN           False  [155, 0]
9                    NaN      18          NaN           False  [157, 2]
10                   NaN      20          5.0           False  [105, 1]
11          ('headers',)      22          NaN           False  [100, 5]
12                   NaN      24          NaN           False  [141, 2]
13              response      26          NaN           False  [125, 1]
14              response      28          8.0           False  [124, 1]
15                   NaN      30          NaN           False   [83, 0]
---------------------------------------------------------------------------
Constants found:
['xxxx-yyyy-zzzz', 'https://google.com', 'Authorization', 'Bearer ', ('headers',)]
```

Last but not least is the recursion checker, -R. A successful detection will look something like this…:



```
PS C:\Users\hunts\Downloads\Malware-Final-Project> python main.py -R "recursive_factorial_function"
Checking for recursive functions on the following stack...
    Unnamed: 0             opname  opcode   arg                        argval  \
0            0          LOAD_FAST     124   0.0                             k
1            1         LOAD_CONST     100   1.0                             1
2            2         COMPARE_OP     107   4.0                             >
3            3  POP_JUMP_IF_FALSE     114  12.0                            24
4            4          LOAD_FAST     124   0.0                             k
5            5        LOAD_GLOBAL     116   0.0  recursive_factorial_function
6            6          LOAD_FAST     124   0.0                             k
7            7         LOAD_CONST     100   1.0                             1
8            8    BINARY_SUBTRACT      24   NaN                           NaN
9            9      CALL_FUNCTION     131   1.0                             1
10          10    BINARY_MULTIPLY      20   NaN                           NaN
11          11       RETURN_VALUE      83   NaN                           NaN
12          12         LOAD_CONST     100   1.0                             1
13          13       RETURN_VALUE      83   NaN                           NaN

                         argrepr  offset  starts_line  is_jump_target  \
0                              k       0          2.0           False
1                              1       2          NaN           False
2                              >       4          NaN           False
3                          to 24       6          NaN           False
4                              k       8          3.0           False
5   recursive_factorial_function      10          NaN           False
6                              k      12          NaN           False
7                              1      14          NaN           False
8                            NaN      16          NaN           False
9                            NaN      18          NaN           False
10                           NaN      20          NaN           False
11                           NaN      22          NaN           False
12                             1      24          5.0            True
13                           NaN      26          NaN           False

        bytes
0    [124, 0]
1    [100, 1]
2    [107, 4]
3   [114, 12]
4    [124, 0]
5    [116, 0]
6    [124, 0]
7    [100, 1]
8     [24, 0]
9    [131, 1]
10    [20, 0]
11    [83, 0]
12   [100, 1]
13    [83, 0]
---------------------------------------------------------------------------
Recursion detected! Name of recursive function(s) follows:
['recursive_factorial_function']
```

…while a non-recursive function will output something like this:



```
PS C:\Users\hunts\Downloads\Malware-Final-Project> python main.py -R "factorial_function"
Checking for recursive functions on the following stack...
    Unnamed: 0            opname  opcode  arg        argval       argrepr  \
0            0        LOAD_CONST     100  1.0             1             1
1            1        STORE_FAST     125  1.0       product       product
2            2       LOAD_GLOBAL     116  0.0         range         range
3            3         LOAD_FAST     124  0.0             k             k
4            4        LOAD_CONST     100  1.0             1             1
5            5        LOAD_CONST     100  2.0            -1            -1
6            6     CALL_FUNCTION     131  3.0             3           NaN
7            7          GET_ITER      68  NaN           NaN           NaN
8            8          FOR_ITER      93  6.0            30         to 30
9            9        STORE_FAST     125  2.0  multiplicand  multiplicand
10          10         LOAD_FAST     124  1.0       product       product
11          11         LOAD_FAST     124  2.0  multiplicand  multiplicand
12          12  INPLACE_MULTIPLY      57  NaN           NaN           NaN
13          13        STORE_FAST     125  1.0       product       product
14          14     JUMP_ABSOLUTE     113  8.0            16         to 16
15          15         LOAD_FAST     124  1.0       product       product
16          16      RETURN_VALUE      83  NaN           NaN           NaN

    offset  starts_line  is_jump_target     bytes
0        0          2.0           False  [100, 1]
1        2          NaN           False  [125, 1]
2        4          3.0           False  [116, 0]
3        6          NaN           False  [124, 0]
4        8          NaN           False  [100, 1]
5       10          NaN           False  [100, 2]
6       12          NaN           False  [131, 3]
7       14          NaN           False   [68, 0]
8       16          NaN            True   [93, 6]
9       18          NaN           False  [125, 2]
10      20          4.0           False  [124, 1]
11      22          NaN           False  [124, 2]
12      24          NaN           False   [57, 0]
13      26          NaN           False  [125, 1]
14      28          NaN           False  [113, 8]
15      30          5.0            True  [124, 1]
16      32          NaN           False   [83, 0]
---------------------------------------------------------------------------
Could not find any recursive functions!
```

Notes, Limitations and The Future

For sake of time, we manually added the functions you can disassemble. Currently, in order to add more you would need to change the code each time. In the future, this would and SHOULD be abstract enough to input any function without the need to specify in the code.