

Lab 1: Malware Taxonomy

ITSC 303: Malware Analysis

Table of Contents

[Lab Outcomes 2](#_Toc491425557)

[Background Reading 2](#_Toc491425558)

[Introduction 2](#_Toc491425559)

[1.0 Scenario 2](#_Toc491425560)

[2.0 Identifying Python Executables 3](#_Toc491425561)

[2.1 File Size 3](#_Toc491425562)

[2.2 Strings 3](#_Toc491425563)

[2.3 Extraction 5](#_Toc491425564)

[3.0 Decompiling Python Byte Code 6](#_Toc491425565)

[3.1 Extracting Relevant Byte Code Files 6](#_Toc491425566)

[3.2 Decompiling Byte Code 6](#_Toc491425567)

[4.0 Analyzing Source Code 7](#_Toc491425568)

[4.1 Identifying Key Features 7](#_Toc491425569)

[4.1.1 Analyzing calc.exe 7](#_Toc491425570)

[4.1.2 Analyzing resume.pdf.exe 9](#_Toc491425571)

[Resources 10](#_Toc491425572)

Malware Analysis

Lab 1: Malware Taxonomy

Lab Outcomes

This lab will focus on the following outcomes:

* Decompile Python byte code using publicly available tools.
* Analyze malicious Python code.
* Classify malicious code based on behaviors using taxonomy.

Background Reading

* *The Art of* *Computer Virus Research and Defense* textbook.
* <http://www.py2exe.org/>
* <https://github.com/wibiti/uncompyle2>
* <https://github.com/matiasb/unpy2exe>

Introduction

This lab provides a high-level introduction to decompiling Python byte code, with the goal of classifying three provided samples based on their behaviours, using the taxonomy that was introduced in the Malware History and Taxonomy lecture.

Python byte code .pyc files are extracted from binaries generated using **py2exe**, and that byte code is then decompiled to provide the original Python source code for analysis. Once the source code is extracted, you will classify the malware based on key malicious features in the source code.

1. Scenario

An employee complained of having a slow computer. An investigation revealed that the computer was consuming a high percentage of CPU for a process labelled **calc.exe** and three suspicious samples were found in the Microsoft Windows **%temp%** directory. Your network administrator also noticed a large amount of HTTP network traffic containing file data coming from this machine.

You’ve been asked by your CISO to determine whether these samples are malicious, to identify whether it is possible for them to have spread elsewhere in your network, and to classify their malware type if they are indeed malicious.

1. Identifying Python Executables

Python is compiled into byte code, which is then interpreted as opcodes at run-time by the stack-based Python interpreter. Therefore, malware authors who write their malware using Python require an interpreter to be shipped with their compiled byte code. This is typically done with a self-extracting executable in order to ship all required files, including the interpreter, dependencies and compiled code. You can determine whether the binary uses Python by analyzing data, such as strings within these executables (to be covered in detail in the static analysis section of the course, by extracting the executable ourselves or by looking at the size of the executable.

## 2.1 File Size

The size of the executable is a main differentiating factor for Python-related binaries. Malware written in other languages, such as C, is substantially smaller because it is dynamically linked against the Windows API, and doesn’t have to ship required dependencies. The code has already been translated into machine instructions to be executed by the CPU, so there is no need to ship an interpreter.

C:\Users\Joshua Reynolds\Desktop\figure-1-binary-size.png

**Figure 1: The Large Binary File Size of calc.exe**

Used with permission from Microsoft.

## 2.2 Strings

Without going into detail about how strings are extracted from binaries, run **strings.exe** from Sysinternals against the **calc.exe** sample from our analysis environment. Strings will extract ASCII characters from the binary.

Using the findstr command, search for Python-related strings in the output:

>.\strings.exe calc.exe | findstr Python

Py\_SetPythonHome

more recent the version of Python needed to read the pickle

the imageop module has been removed in Python 3.0

-3 : warn about Python 3.x incompatibilities that 2to3 cannot trivially fix

-V : print the Python version number and exit (also --version)

Python %s on %s

Python %s

Intermediate representation of a Python parse tree.

abort() called from Python code didn't abort!

unbounded read returned more bytes than a Python string can hold

while calling a Python object

Internal error in the Python interpreter.

Please report this to the Python maintainer, along with the traceback,

the Python version, and the hardware/OS platform and version.

Base class for all standard Python exceptions that do not represent

BaseException.message has been deprecated as of Python 2.6

line is longer than a Python string can hold

requested number of bytes is more than a Python string can hold

Python int too large to convert to C int

Python int too large to convert to C long

string is too long for a Python string

join() result is too long for a Python string

unichr() arg not in range(0x10000) (narrow Python build)

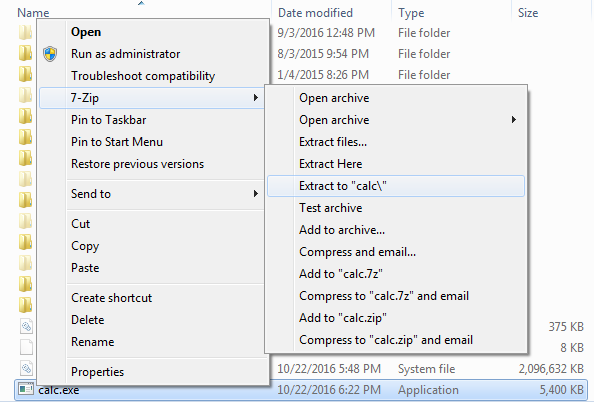
decoded result is too long for a Python string

-snip-

As you can see in this example output, there are a number of references to Python-related functionality.

## 2.3 Extraction

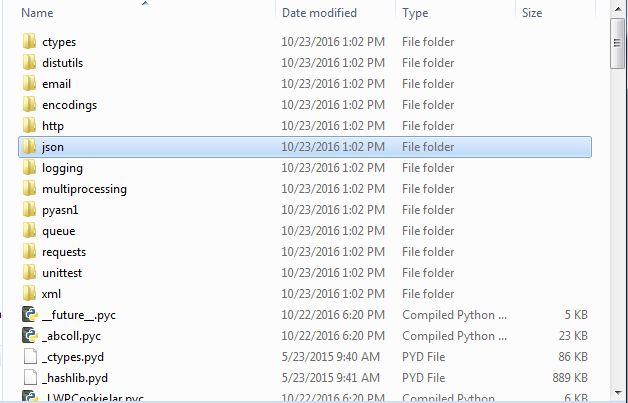
You can use an archive tool, such as 7-Zip, to extract the archive components of the py2exe self-extracting archive. This will provide insight into what files the archive contains.



**Figure 2: 7-Zip Extract Archive**

Used with permission from Microsoft.

The result is a large number of folders containing .pyc and .pyd files. These files are the dependencies for the malware in question.



**Figure 3: Extracted Archive Files**

Used with permission from Microsoft.

1. Decompiling Python Byte Code

## 3.1 Extracting Relevant Byte Code Files

To recover the relevant Python source code, you must first extract the relevant .pyc files using **unpy2exe**. For the purpose of this exercise unpy2exe has been packaged as an executable, although the implementation is done in Python, and can be found on [GitHub](https://github.com/matiasb/unpy2exe) (https://github.com/matiasb/unpy2exe). This can be installed on any system that supports Python.

As a preliminary example, extract the relevant .pyc files from the calc.exe example:

>.\unpy2exe.exe calc.exe

Magic value: 78563412

Code bytes length: 4477

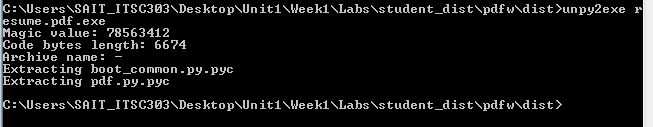
Archive name: -

Extracting boot\_common.py.pyc

Extracting calc.py.pyc

Two files have been extracted. Do this for all provided samples for this lab.





**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 3.2 Decompiling Byte Code

Now that you have extracted all relevant Python byte code files, you can decompile them to their original source code using the **uncompyle2** tool. For the purpose of this exercise uncompyle2 has been packaged as an executable, although the implementation is done in Python, and can be found on [GitHub](https://github.com/wibiti/uncompyle2) (https://github.com/wibiti/uncompyle2). This can be installed on any system that supports Python.

>uncompyle2.exe calc.py.pyc

# 2016.10.30 19:56:32 Mountain Daylight Time

# Embedded file name: calc.py

from multiprocessing import Process

import subprocess

import os

import time

import re

import requests

import base64

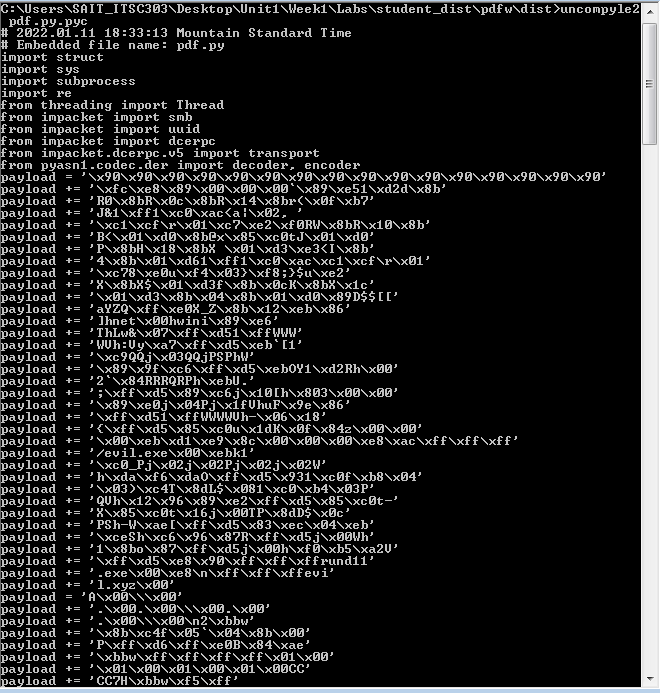
class Evil:

C2 = 'http://evil.xyz/files'

-snip-

Perform this for all relevant extracted .pyc files and redirect the source code to files for further analysis.







**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Analyzing Source Code

## Identifying Key Features

Now that you have recovered the original source code, you can analyze it to extract the key functionality of the malware. Start with the calc.exe sample to uncover its true intentions.

### 4.1.1 Analyzing calc.exe

1. Within the Evil Python class, what is the class variable referencing?

**It is referencing a link to evil.xyz/files**

1. Describe what the \_\_init\_\_() method is doing. How would this malware be classified in relation to the discussed taxonomy?

**it launches the REAL calc.exe program to distract the victim**

**It is a trojan, as it is trying to appear harmless.**

1. Describe what the steal\_hives method is doing. How would this malware be classified in relation to the discussed taxonomy?

**It copies the SAM, SECURITY, and SYSTEM hives, and sends them to the attacker.**

**It is Information stealing.**

1. Describe what the steal\_files method is doing. How would this malware be classified in relation to the discussed taxonomy?

**It is copying and stealing files on the host**

**It is information stealing.**

1. What file extensions are being referenced by the re.match call?

**doc, docx, txt, xls, xlsx, db, png, jpg and jpeg.**

**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 4.1.2 Analyzing resume.pdf.exe

1. What vulnerability is this malware exploiting?

**It exploits a bug in server service called the Named Pipe Vulnerability.**

1. Describe the get\_ips() method.

**It runs the ipconfig command and gets the ips from the output.**

1. What is being accomplished in the for loop when \_\_name\_\_ == ‘\_\_main\_\_’? How would this malware be classified in relation to the discussed taxonomy?

**It is attempting to spread to other hosts.**

**It is a Worm**

1. Can you find ASCII strings in the exploit payload? (**Hint:** Look for the class variable value from calc.exe.) How would this malware be classified in relation to the discussed taxonomy?

**Yes you can find ASCII in the payload  
It is more than likely a Worm**

**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Resources

Szor, P. (2005). *The art of computer virus research and defense*. Toronto: Addison-Wesley.