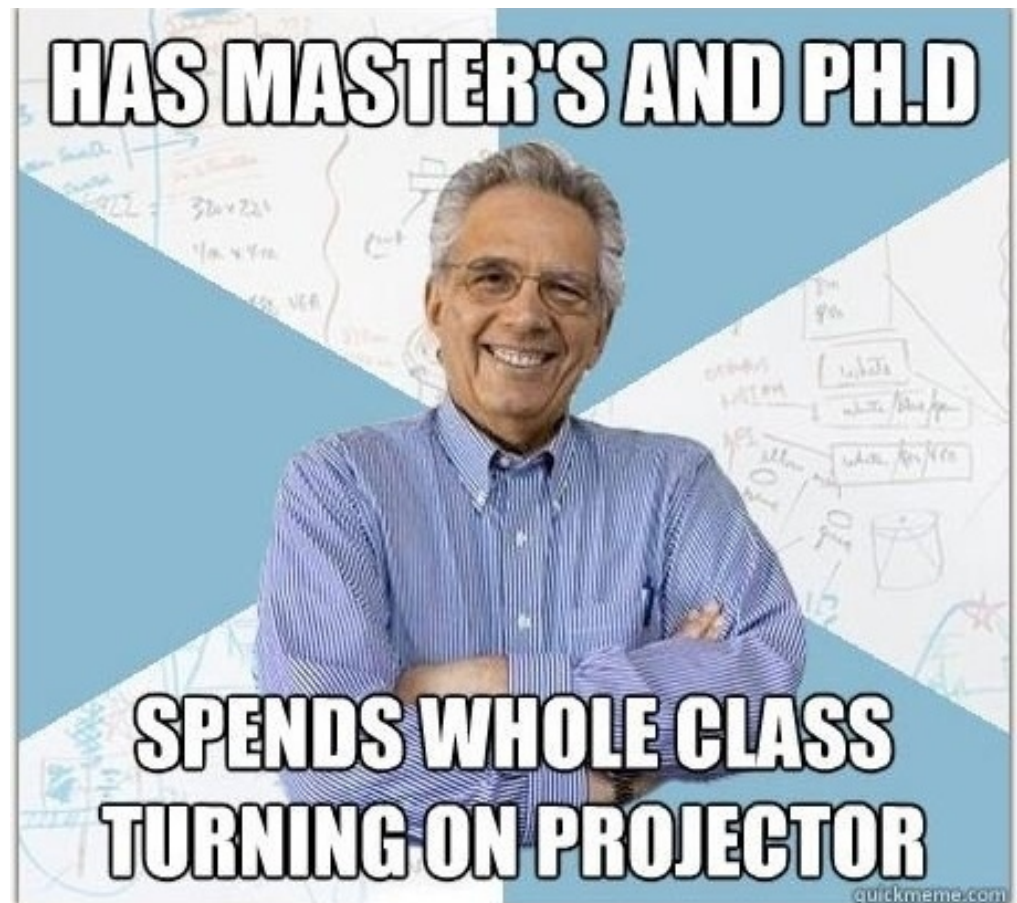


Spoils of the Hackathon

Leon Fedotov

Joey Geralnik

Itzik Kotler



Hackathon:

- The hackathon was really awesome
- Next "Spoils of the Hackathon" presentation could be given by YOU!
- Seriously: come, code, crack, create. It is really fun.

pytroj



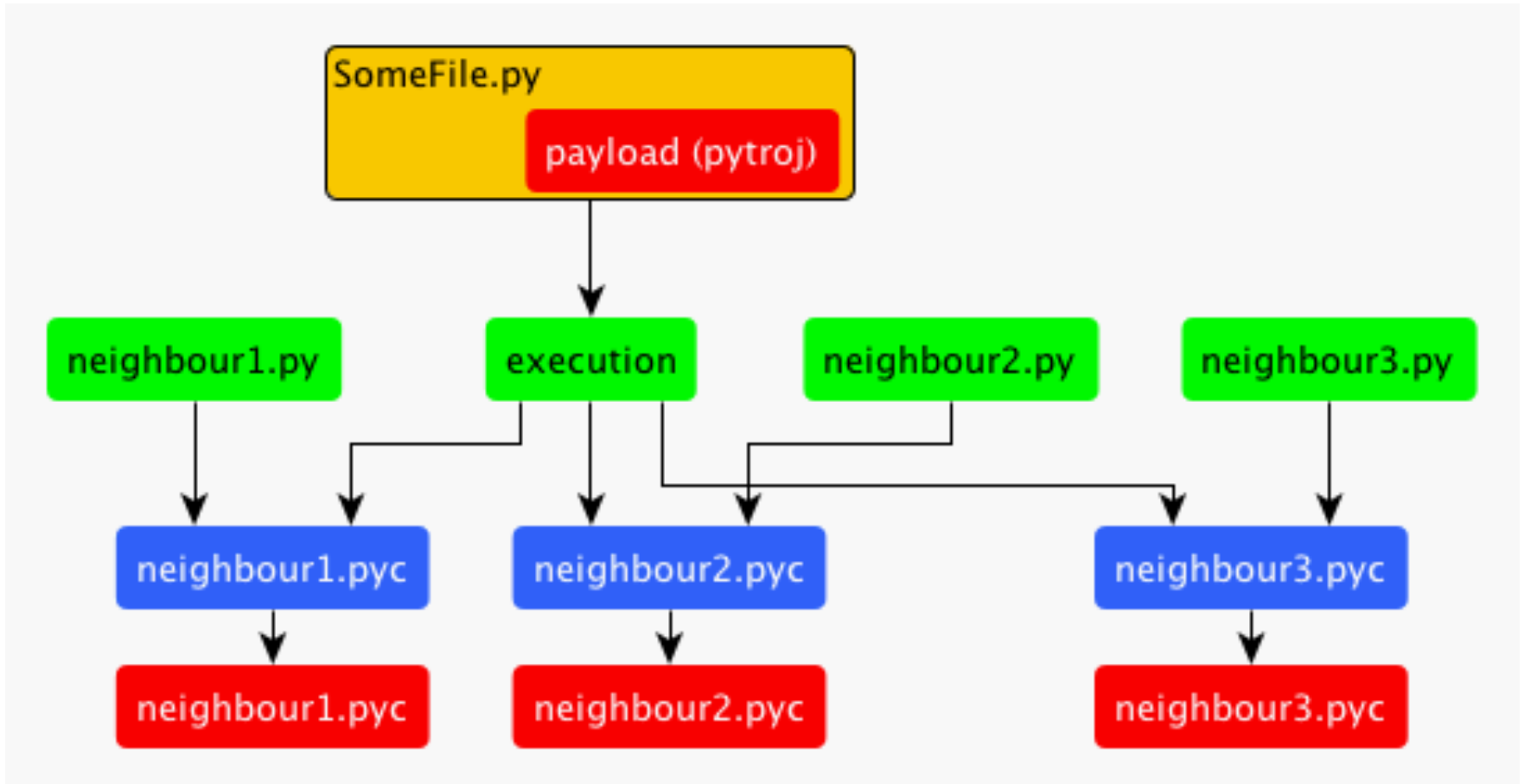
Python's execution

- Python file is interpreted.
- Bytecode is stored in a file with the same name but with the pyc extension.
- The file is signed with the original file's system timestamp and executed.
- Next time you run the same file, python will look for a pyc file and check it's timestamp against the original.
- If they match we are good to go and python only executes the bytecode without reinterpretation.

Goals of pytroj

- Self infecting payload.
- Once an infected file is executed pytroj looks for pyc files and prepends itself to them.
- Then the infected file executes its original bytecode.
- If at any time the .py file is updated, the .pyc will be automatically recompiled. This is intentional: the updated code must run or pytroj will be easily detectable.
- However, because pytroj infects all .pyc files on each run it should quickly be reinfected

Flow



How?

- pytroj opens the file it's been executing from.
- Reads the bytecode, and finds itself with a string trick - we bound out payload with two strings.
- then it looks for pyc files, and uses the same technique to inject what it found into the files.
- saves the files without touching the timestamp that “signs” the pyc file.


Getting started

- Code must be able to reproduce.
- Quine? No, we cheat.
- Program reads itself, locates malicious part, and copies to other .pyc files.

Tampering with pyc files

- Python marshal module reads pyc files (less magic number and timestamp) and returns code object.
- Code object is read only :-).
- Open source to the rescue! [Byteplay](#) reads new Code object that can be converted to marshal code objects and are easy to work with.
- Include byteplay in pytroj
- Payload is now ~30KB.

Restructuring - but first!

- Can't count the number of times I entered ``rm *.pyc`` to remove infected pyc files and start again -_-
- All it would take is one mistake to enter ``rm *.py`` instead 
- Summary: use version control. Plus, [github](https://github.com) is free hosting.

Restructuring - payload size

- How can we decrease size of payload?
- Once again, open source to the rescue.
- [pyminifier](#) takes input file, zips it, converts result to base64, produces output file with:
`exec zlib.decompress(base64.b64decode("base64_program_here"))`
- Minifying byteplay brings payload size down to ~7KB.
- For comparison, .pyc files in my /usr/lib/python2.7 range between 100B and 170KB, with most being above 10. 7KB will usually not be noticeable.
- can be gunzipped and minimized further but we wanted to keep the logic open and easy to understand.

Automatically detecting length of exploit

- Simple hack - begin and end code with:
signature = "DC9723"
- Gets translated into:
0 LOAD_CONST 0 ('DC9723')
3 STORE_NAME 0 (signature)
- if data.code[1][1] == signature:
 #Code is already infected
 return
- To find exploit size, iterate over own code until you come to the second signature, easy!

Live example!



Less than legal next steps:

- Infect package on [pypi](#). Installing the package with `pip install infected_django` could infect the whole system.
- Malicious code instead of our harmless payload could do anything: keylog, spyware, etc.

linx

- <https://github.com/jgeralnik/Pytroj>
- <http://news.ycombinator.com/item?id=3039439>
- <http://www.symantec.com/connect/blogs/python-has-venom>
- <http://www.google.co.il/search?q=pytroj>

FIN



Python Tail Recursion