Vimcryption

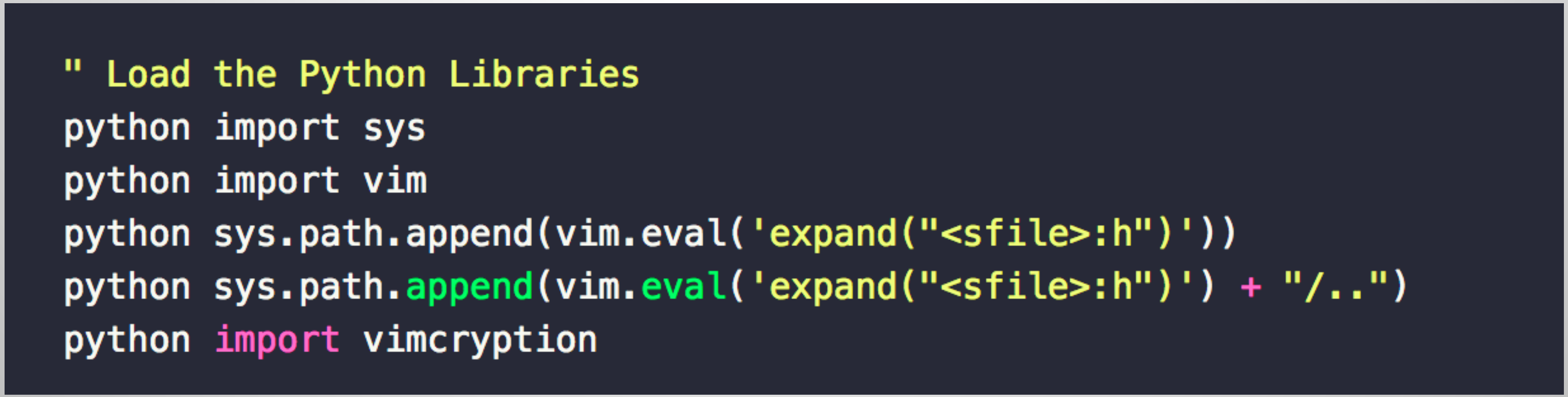
Thomas Manner and Miguel Nistal

Motivation: Vim Needs Encryption Options!

- Vim is one of the most popular editors of all time (https://stackoverflow.blog/2017/05/23/stack-overflow-helping-one-million-developers-exit-vim/)
- Very extensible (and with Python!)
- Vim currently doesn't have any good encryption options (pkzip, blowfish, blowfish2, or gnupg plugin)
- Vimcryption makes a good foundation for a long term open-source project

Building a Vim Plugin

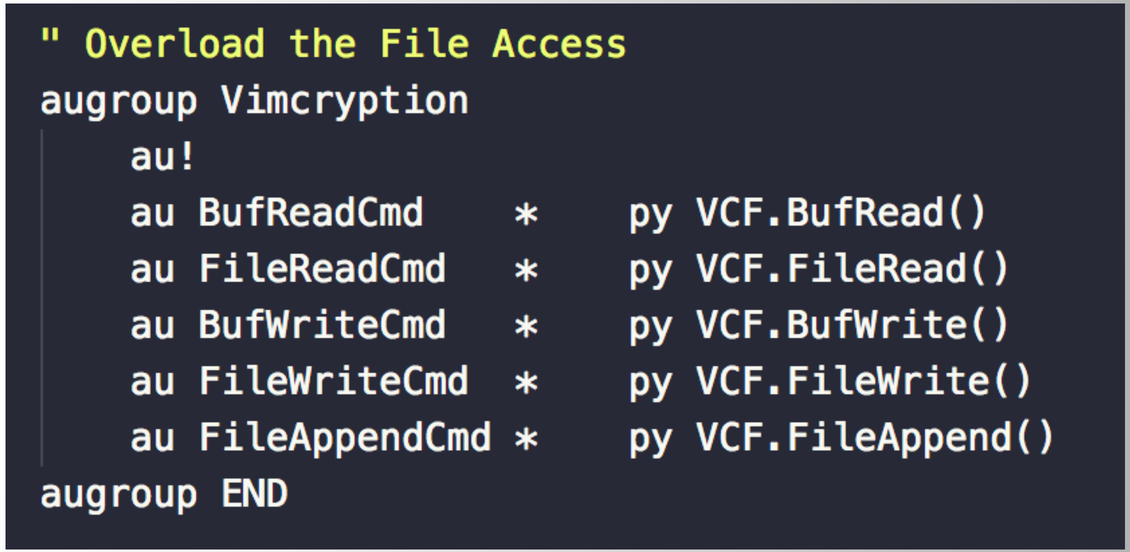
- Vim Plugins are written in VimScript with a common structure
- The "Pathogen-compatible" structure will allow us to release the plugin to be downloaded via package managers
- We took the approach of implementing the API in VimScript but offloading the heavy lifting to a Python backend
- This is done in Vim by calling the built-in Vim Python Interpreter which is launched at runtime and available globally to VimScript



```vim
" Load the Python Libraries
python import sys
python import vim
python sys.path.append(vim.eval('expand("<sfile>:h")'))
python sys.path.append(vim.eval('expand("<sfile>:h")') + "/..")
python import vimcryption
```

Intercepting File Handling

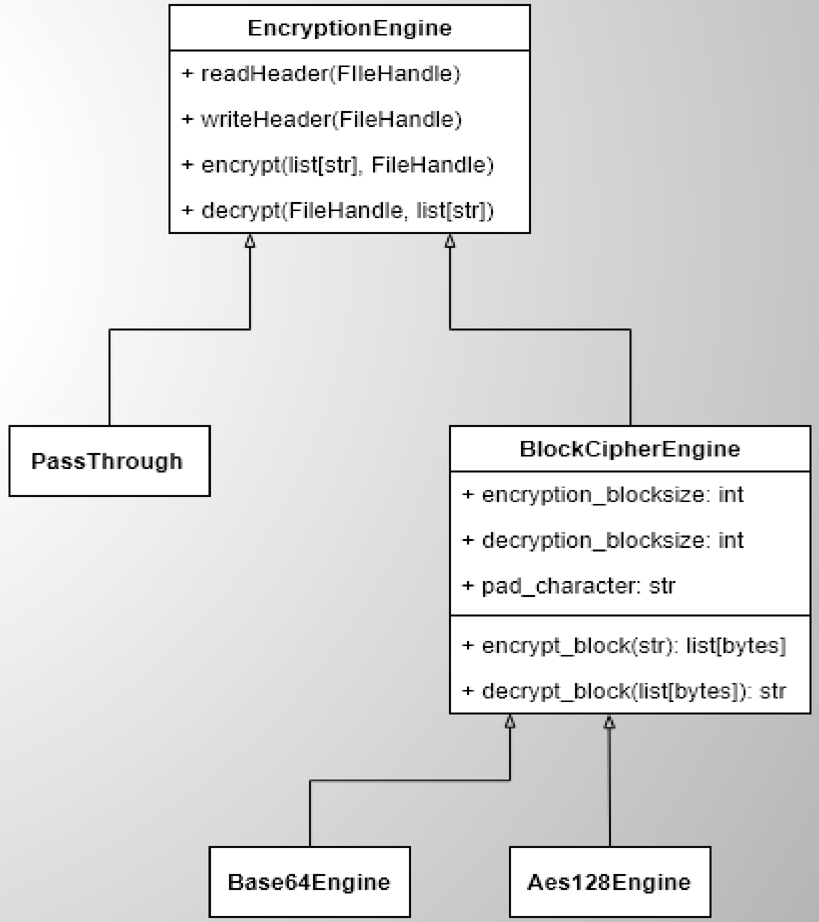
- For security, we need to ensure all disk reads and writes are done by Vimcryption
- Anytime the plugin loads, we disable swap, backup, undo history, and viminfo
- Vim has builtin support for overloading the rest of the file operations via "Cmd-Events"
- Cmd-Events are special callbacks which allow us to replace the built-in file operations with our own callbacks to the Vimcryption Python backend



```
" Overload the File Access
augroup Vimcryption
    au!
    au BufReadCmd    *    py VCF.BufRead()
    au FileReadCmd   *    py VCF.FileRead()
    au BufWriteCmd   *    py VCF.BufWrite()
    au FileWriteCmd  *    py VCF.FileWrite()
    au FileAppendCmd *    py VCF.FileAppend()
augroup END
```

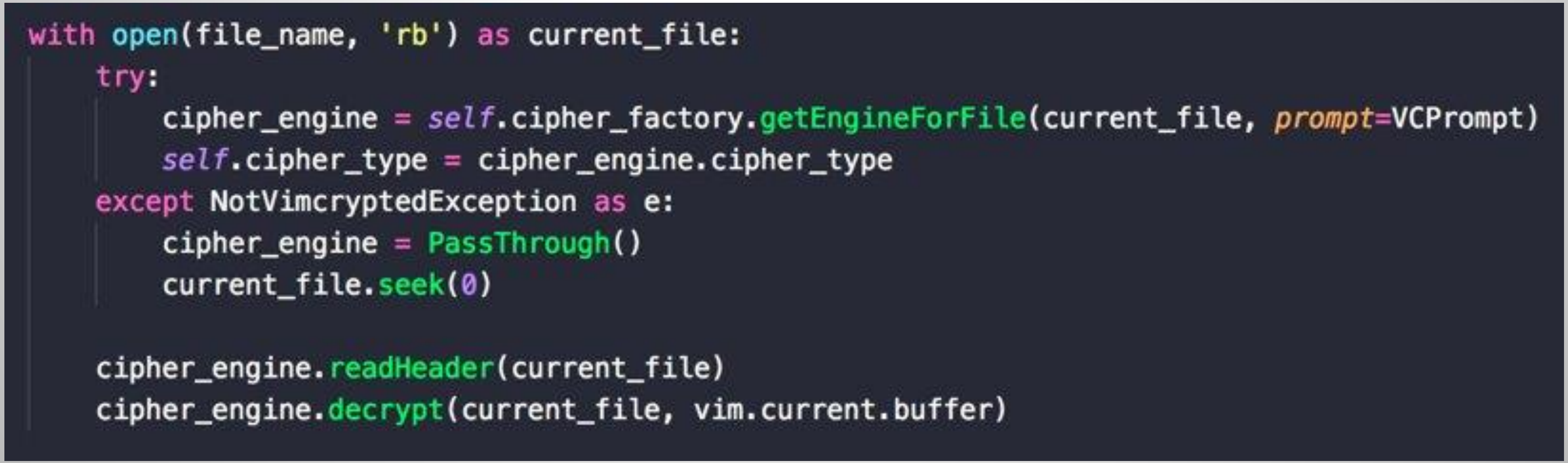
The Extensible Encryption Engine

- To keep up with the cryptographic arms race, any encryption plug in needs to be easy to update and extend.
- Engine Architecture
  - EncryptionEngine – Abstract Base Class
    - encrypt
    - decrypt
  - BlockCipherEngine – Abstract Base Class
    - encrypt_block
    - decrypt_block


EncryptionEngine
+ readHeader(FIleHandle)
+ writeHeader(FileHandle)
+ encrypt(list[str], FileHandle)
+ decrypt(FileHandle, list[str])
PassThrough
BlockCipherEngine
+ encryption_blocksize: int
+ decryption_blocksize: int
+ pad_character: str
+ encrypt_block(str): list[bytes]
+ decrypt_block(list[bytes]): str
Base64Engine
Aes128Engine

Using the Encryption Engine

- We used the Factory design pattern to dynamically get the correct engine based on the meta-data in the file header
- Once we have the engine, the code to decrypt a file is only two lines!
- Eventually we'll bind readHeader into the Engine and reduce it to one



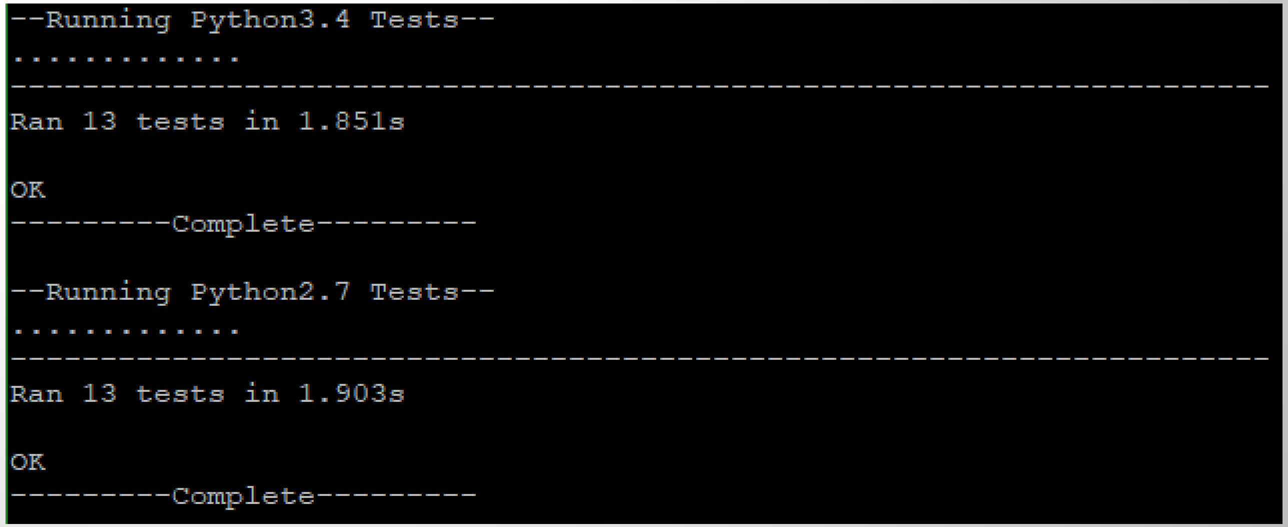
```python
with open(file_name, 'rb') as current_file:
    try:
        cipher_engine = self.cipher_factory.getEngineForFile(current_file, prompt=VCPrompt)
        self.cipher_type = cipher_engine.cipher_type
    except NotVimcryptedException as e:
        cipher_engine = PassThrough()
        current_file.seek(0)

    cipher_engine.readHeader(current_file)
    cipher_engine.decrypt(current_file, vim.current.buffer)
```

Efficient Text Processing with Generators

- Text files can be larger than the memory space of our machine.
- Encryption library doesn’t need to know about everything in the stream at once.
- Streaming allows more flexibility:
  - Network data streams like ftp and ssh
  - Terminal input and output (tty)
  - Live log files
- Only processes the next character or block.

Unit Testing the Engine

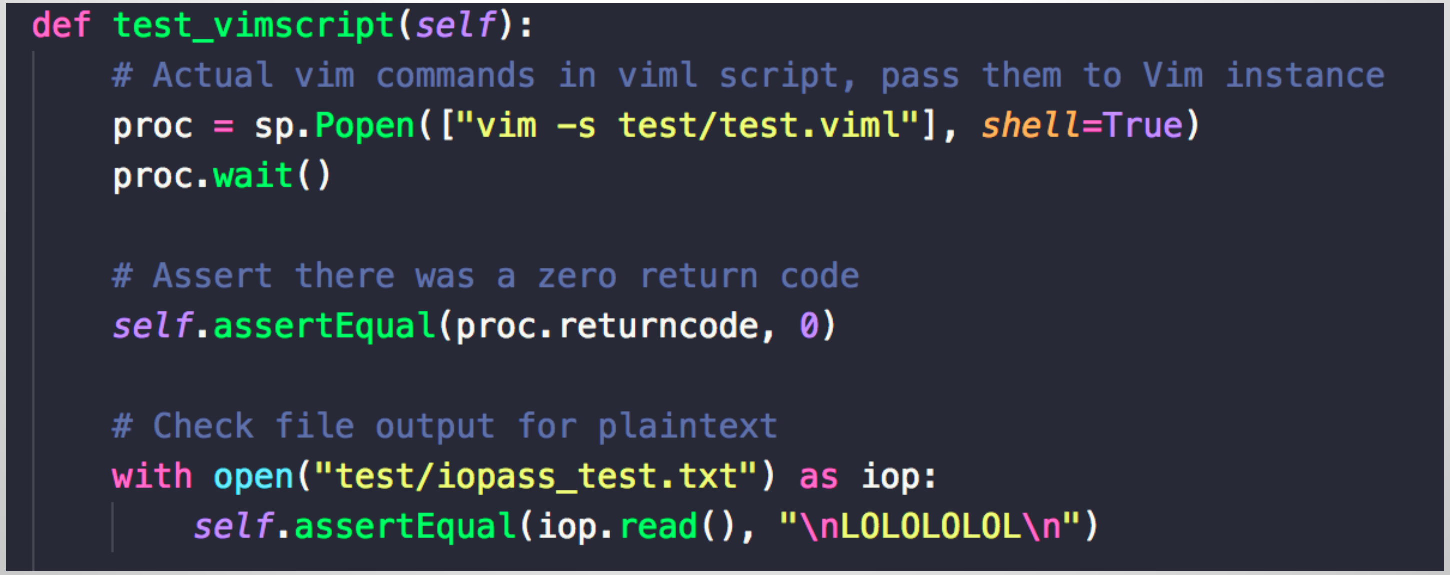
- To ensure stability and to prevent functional regression, we set up unittests for the EncryptionEngine library. The tests define and check all engine interfaces to guarantee that passing library code will work with VIM on the first try.
- Suites
  - TestPassThrough
  - TestBase64Engine
- Tests
  - test_encrypt_str
  - test_encrypt_list
  - test_decrypt_str
  - test_decrypt_list
  - test_readHeader
  - test_writeHeader



```
--Running Python3.4 Tests--
.............
----------------------------------------------------------------------
Ran 13 tests in 1.851s

OK
---------Complete---------

--Running Python2.7 Tests--
.............
----------------------------------------------------------------------
Ran 13 tests in 1.903s

OK
---------Complete---------
```

Extending the Testing to Vim Integration

- To provide integration testing for the plugin, we needed a way to automate the actual usage of the plugin in a Vim instance
- Vim has an argument "-s" which will execute characters in a file as if they were the result of user keystrokes
- We were then able to automate the foreground testing we would've done, and do all the verification from a Python based test function



```python
def test_vimscript(self):
    # Actual vim commands in viml script, pass them to Vim instance
    proc = sp.Popen(["vim -s test/test.viml"], shell=True)
    proc.wait()

    # Assert there was a zero return code
    self.assertEqual(proc.returncode, 0)

    # Check file output for plaintext
    with open("test/iopass_test.txt") as iop:
        self.assertEqual(iop.read(), "\nLOLOLOLOL\n")
```

Demo: Let's Vimcrypt!

Final Thoughts

- Project will be available via GitHub and installable via Pathogen and Vundle! Look for the project on vimawesome.com!
- Learned a lot about the practical implications of security when dealing with File System Operations and text editing
- We want to continue development, eventually supporting asymmetric key encryption so you can easily share files with your friends
- We've also found that Python lacks a generally accepted encryption library to match the built-in cryptographic hashlib – This is a need we can solve, maybe we'll get a PEP!

Thank you!

Resources

- VimScript API:
  - https://devhints.io/vimscript
- VimScript CmdEvent Reference:
  - http://vimdoc.sourceforge.net/htmldoc/autocmd.html
- VimScript Python Reference:
  - http://vimdoc.sourceforge.net/htmldoc/if_pyth.html
- VimScript Tutorial:
  - http://learnvimscriptthehardway.stevelosh.com/
- Python Generators:
  - https://wiki.python.org/moin/Generators
- Python Unit Testing:
  - https://docs.python.org/3/library/unittest.html