Penetration Testing

Lab: Static Code Analysis

Professor: Anthony J Candeias

Objective:

XYZ Corporation has contracted you to perform a penetration test. In this engagement they would like to evaluate the security of the source code of a new application being built in python for the enterprise. You are to perform a static code analysis of the source of the applications. The goal is to identify vulnerabilities in the source code and report them to management on how to secure the application.

Applications Repos:

- Application 1
  - https://github.com/mpirnat/lets-be-bad-guys
- Application 2
  - https://github.com/fportantier/vulpy

Rubric:

Percentage	**Item**
20	Executive Summary ● Overview of the issues at a C-Level ● Business impact of the issues
20	Application 1 findings ● Vulnerabilities discovered
20	Application 2 Findings ● Vulnerabilities discovered
20	Recommendations ● Tactical recommendations (Patches, ACLs, etc.) ● Strategic recommendations (Security processes)
10	Methodology ● Overview of the approached used to complete the assignment ● Overview of the tools and scripts used ● For full credit of this section - explanation of automation use cases should be documented with custom scripts built for the assignment
10	Format of report and document ● Length of document each section should be a minimum of 1 page ● Reference need to be documented in a work cited page ● Format of the report should be professional ○ Refer back to Module 1 Pen Test Sample report for format and structure

Lab Guide

Overview: The following document is to provide step by step instructions on how to perform a static code analysis of python source code. This process leverages an open source tool Bandit which is located here. https://github.com/PyCQA/bandit

Python

Python is a scripting language derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages. It is powerful, easy to read and understand, and has a wide scope. Since it is such a popular language, there is a vast number of scripts available on places like Github to learn from and use.

The functionality of Python includes, and is not exclusive to:

- Interactive "shell"
- Basic types: numbers, strings
- Container types: lists, dictionaries, tuples
- Variables
- Control structures
- Functions & procedures
- Classes & instances
- Modules & packages
- Exceptions
- Files & standard library

A generic Python script is structured as follows:

1. “Modules” - Python source files or C extensions
   - Import, top-level via form, reload
2. Statements (if-else, loops, etc)
   - Control flow
   - Create objects
   - **Indentation matters (used instead of {})**
3. Objects
   - Everything is an object
   - Automatically reclaimed when no longer needed

In Python, an identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9). Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus **`Manpower`** and **`manpower`** are two different identifiers.

The following list of words are “Reserved” by Python’s architecture, meaning you cannot use them to define a variable or other things:

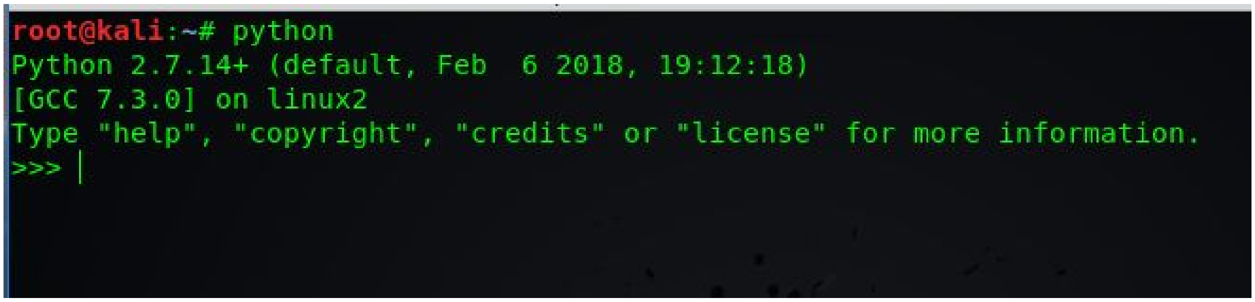
`and`	`exec`	`not`

assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

In Python, there is no need to declare a variable, you only need to initialize it. Everything is a “variable,” including functions, classes, and modules

Let’s start by using the interactive prompt . This environment allows you to test and run Python instructions. The only drawback of using the prompt is that your progress is not saved upon exiting. To open the prompt, execute the following command in the terminal:

```
python
```

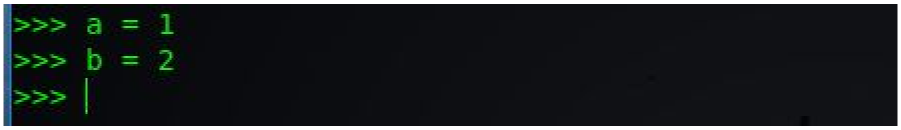


```
root@kali:~# python
Python 2.7.14+ (default, Feb  6 2018, 19:12:18)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Image 1. The interactive Python environment

Now that we are in the interactive window, let’s perform some basic operations, such as defining some variables. Execute the following two commands to set **`a = 1`**, and **`b = 2`**:

```
a = 1
b = 2
```

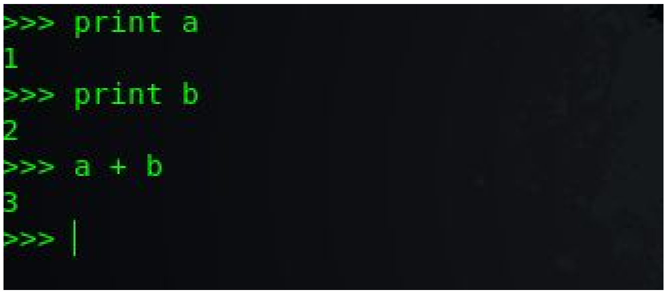


```
>>> a = 1
>>> b = 2
>>> |
```

Image 2. Defining variables

We can interact with these variables as well, such as printing them and adding them together. Execute the following commands:

```
print a
print b
a + b
```



```
>>> print a
1
>>> print b
2
>>> a + b
3
>>> |
```

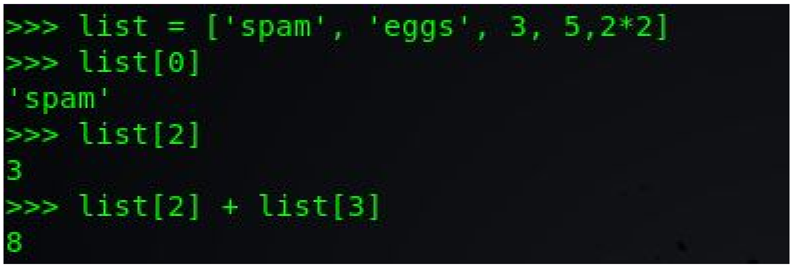
Image 3. Manipulating variables

We can also use lists (aka arrays) in Python. They can be heterogeneous, meaning they can contain both numbers, strings, and other variables. To create a list, execute the following command:

```
list= ['spam', 'eggs', 3, 5, 2*2]
```

As before, we can make the elements of the list interact with each other by calling the location (remembering that we start counting from zero). For example, if we were to access the third element our list, we would execute the command:

```
list[2]
```



```
>>> list = ['spam', 'eggs', 3, 5,2*2]
>>> list[0]
'spam'
>>> list[2]
3
>>> list[2] + list[3]
8
```

Image 4. Creating and manipulating items in a list

We can apply built in functions to lists, including:

- append(x)
- extend(x)
- insert(i,x)
- remove(x)
- pop([i]), pop()
- index(x)
- count(x)
- sort(x)

- reverse(x)

In Python, we have the following control structures: `if`, `elif`, `else`, `while`, `for`, `break`, and `continue`. They perform the same logical functions as in any other programming language. An example for loop is presented as follows:

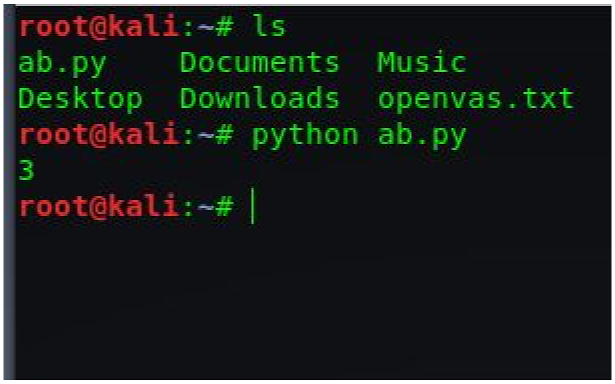
```
for n in range(2,10):
      for x in range(2,n):
            if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
      else:
            # loop fell through without finding a factor
            print n, 'is prime'
```

Again, the interactive mode is useful for exploring the power of Python. However, we would like to create scripts in an external editor (such as vim, Atom, TextWrangler, etc.) and run them in the terminal. Kali comes preinstalled with Leafpad, which is what we will use to create and edit our scripts.

Let's recreate our simple a+b demonstration using a script. Open Leafpad and write the following code (remember, a line with a "#" is commented out, and italicized for clarity):

```
#set our variables
a = 1
b = 2

#printing the outcome of a + b
print a + b
```

Save the file as `ab.py`. Now, open a terminal (in the same location where our script is stored) and execute the command
`python ab.py`



```
root@kali:~# ls
ab.py    Documents  Music
Desktop  Downloads  openvas.txt
root@kali:~# python ab.py
3
root@kali:~# |
```

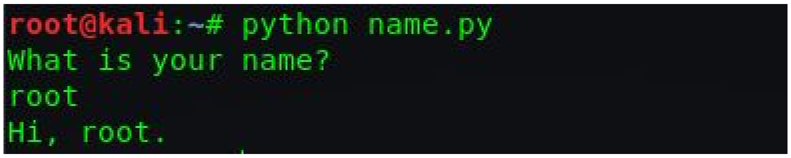
Image 5. Running the script ab.py

Now, let’s make a script that takes user input. The script we will write will return “Hi” for any name entered. Let’s make another script called `name.py`, and write the following code:

```
    #defines name and specifies to use raw input
    name = raw_input('What is your name?\n')

    #prints response from system
    print 'Hi, %s.' % name
```

Now, execute the command in the terminal, followed by your name (`root` in this example):

```
    python name.py
```



```
root@kali:~# python name.py
What is your name?
root
Hi, root.
```

Image 6. Running the script name.py

Having introduced the basics of coding in Python, we can turn to using it for penetration testing. Let’s examine the script PortScan.py provided in the *PenTestingScripts* folder. We will outline the following code step by step:

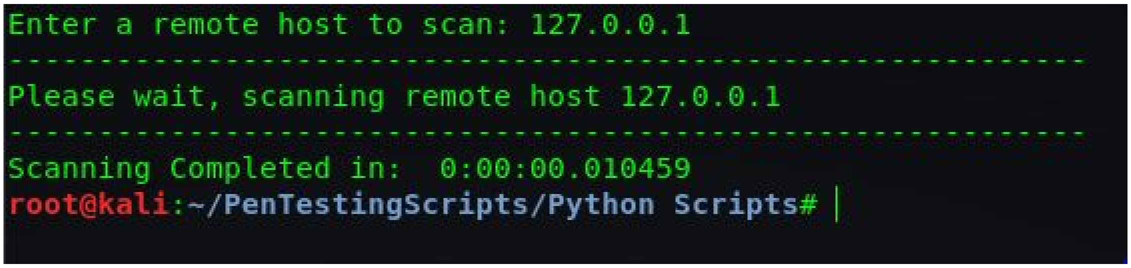
```
#!/usr/bin/env python
#importing libraries to use existing functions
#e.g. the socket library allows you to use the networking protocols
# built into python besides coding tcp/ip over again
import socket
import subprocess
import sys
from datetime import datetime

# Clear the screen
subprocess.call('clear', shell=True)

# Ask for input from the user
remoteServer    = raw_input("Enter a remote host to scan: ")
remoteServerIP  = socket.gethostbyname(remoteServer)

# Print a nice banner with information on which host we are about to
# scan
print "-" * 60
print "Please wait, scanning remote host", remoteServerIP
print "-" * 60
```

```
# Check what time the scan started
t1 = datetime.now()

# Using the range function to specify ports (here it will scans all
ports between 1 and 1024)

# We also put in some error handling for catching errors
# References the components of the socket library to have network
# communication
# Note, we can change the port range here (default is 1-1025)
# It also references the remote server which was defined by the user

try:
    for port in range(1,1025):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        result = sock.connect_ex((remoteServerIP, port))
        if result == 0:
            print "Port {}:       Open".format(port)
        sock.close()

# prompts the user if you break the script by pressing control+c
except KeyboardInterrupt:
    print "You pressed Ctrl+C"
    sys.exit()

# If you input the hostname and it cannot resolve the IP address
except socket.gaierror:
    print 'Hostname could not be resolved. Exiting'
    sys.exit()

# Error message if the IP cannot be reached
except socket.error:
    print "Couldn't connect to server"
    sys.exit()

# Checking the time again
t2 = datetime.now()

# Calculates the difference of time, to see how long it took to run
# the script
total =  t2 - t1

# Printing the information to screen
print 'Scanning Completed in: ', total
```

Let’s use this script to perform a port scan of the local address 127.0.0.1. While in the Python Scripts directory, execute the following command:

```
python PortScan.py
```

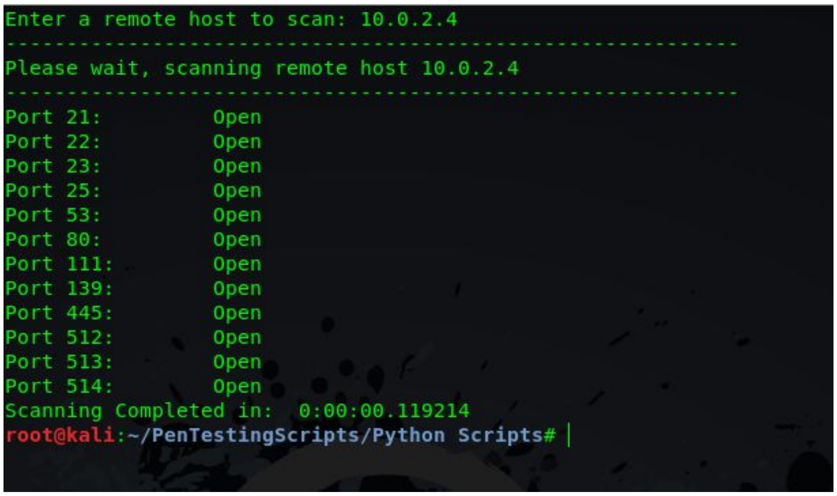
When prompted, enter 127.0.0.1 and hit enter.



```
Enter a remote host to scan: 127.0.0.1
------------------------------------------------------------
Please wait, scanning remote host 127.0.0.1
------------------------------------------------------------
Scanning Completed in:  0:00:00.010459
root@kali:~/PenTestingScripts/Python Scripts# 
```

Image 7. Running the script PortScan.py against the localhost

In our scenario, we have no open ports. If we were to scan our Metasploitable VM (address 10.0.2.4), we would return a much different result:

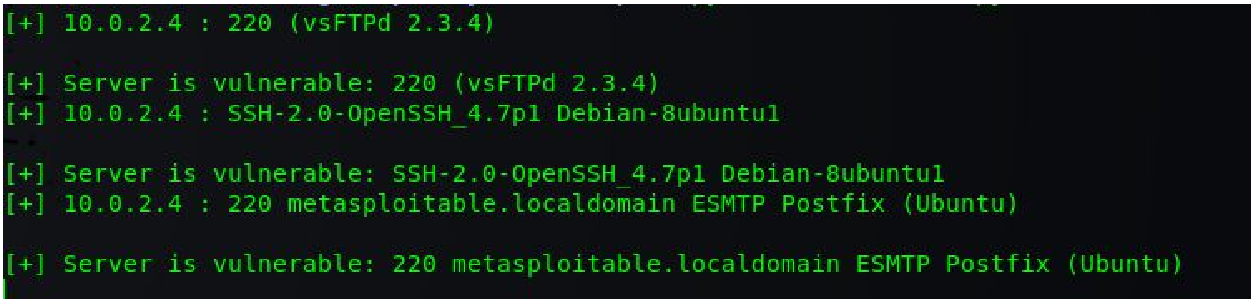


```
Enter a remote host to scan: 10.0.2.4
------------------------------------------------------------
Please wait, scanning remote host 10.0.2.4
------------------------------------------------------------
Port 21:         Open
Port 22:         Open
Port 23:         Open
Port 25:         Open
Port 53:         Open
Port 80:         Open
Port 111:        Open
Port 139:        Open
Port 445:        Open
Port 512:        Open
Port 513:        Open
Port 514:        Open
Scanning Completed in:  0:00:00.119214
root@kali:~/PenTestingScripts/Python Scripts# 
```

Image 8. Output after running PortScan.py against the Metasploitable VM

Another script available in the course directory is VulnScanner.py. As the name implies, this code will output a list of all possible vulnerabilities present on the target host. Note, this script takes a text file consisting of a list of the target(s) as a command line argument. Execute the following command (note, this may take a while):

```
python VulnScanner.py vuln-banners.txt
```



```
[+] 10.0.2.4 : 220 (vsFTPd 2.3.4)

[+] Server is vulnerable: 220 (vsFTPd 2.3.4)
[+] 10.0.2.4 : SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1

[+] Server is vulnerable: SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
[+] 10.0.2.4 : 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)

[+] Server is vulnerable: 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)
```

Image 9. Output from the script VulnScanner.py targeting Metasploitable

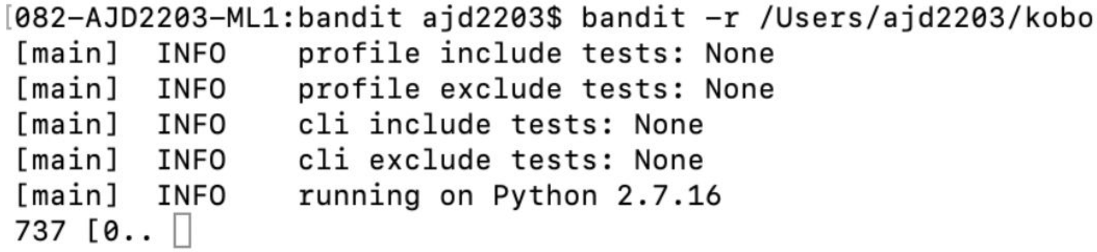
The `vuln-banners.txt` file is a list of only a few vulnerabilities. You are encouraged to add any other vulnerable banners you find to it. Note, the `VulnScanner.py` file only scans ports 21, 22, 25, 80, 110, and 443. You can add any other ports you would want to scan.

Bandit Instructions

1. Git clone the bandit repo locally on a Linux system
   a. **git clone https://github.com/PyCQA/bandit.git**
2. Navigate to the new bandit directory and run the install script
   a. **sudo python setup.py install**
3. For this request we need to scan multiple repos, I created a new directory and git cloned all the repos into the single directory
   a. Git clone all the repos you want to scan into one directory
4. Now we have our source code in a directory to review

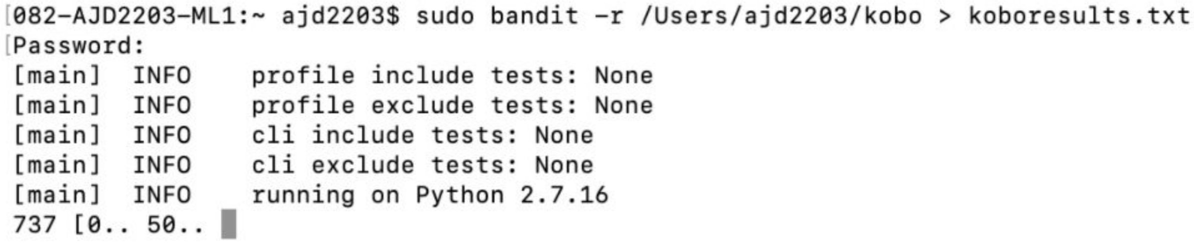
Usage Instructions

1. We need to run bandit against the directory with the source code
   a. **bandit -r {~/your_repos/project}**
      i. Remove brackets
   b.



```
[082-AJD2203-ML1:bandit ajd2203$ bandit -r /Users/ajd2203/kobo
 [main]  INFO    profile include tests: None
 [main]  INFO    profile exclude tests: None
 [main]  INFO    cli include tests: None
 [main]  INFO    cli exclude tests: None
 [main]  INFO    running on Python 2.7.16
 737 [0..
```

2. The results will populate, save those to provide a summary overview to the requestor
3. Its recommended to output the results into a text file for easier review and to use sudo as well
   a.



```
[082-AJD2203-ML1:~ ajd2203$ sudo bandit -r /Users/ajd2203/kobo > koboresults.txt
[Password:
 [main]  INFO    profile include tests: None
 [main]  INFO    profile exclude tests: None
 [main]  INFO    cli include tests: None
 [main]  INFO    cli exclude tests: None
 [main]  INFO    running on Python 2.7.16
 737 [0.. 50..
```

Bandit Results

```
Test results:
>> Issue: [B605:start_process_with_a_shell] Starting a process with a shell, possible injection
detected, security issue.
   Severity: High   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-docker/mongo/backup-to-s3.py:70
   More Info:
https://bandit.readthedocs.io/en/latest/plugins/b605_start_process_with_a_shell.html
69
70        os.system("{backup_command}|s3cmd put --multipart-chunk-size-mb={chunk_size} -
s3://{bucket}/{filename}".format(
71            backup_command=BACKUP_COMMAND,
72            bucket=AWS_BUCKET,
73            chunk_size=CHUNK_SIZE,
74            filename=filename
75        ))

--------------------------------------------------
>> Issue: [B404:blacklist] Consider possible security implications associated with subprocess
module.
   Severity: Low   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-docker/postgres/backup-to-s3.py:10
   More Info:
https://bandit.readthedocs.io/en/latest/blacklists/blacklist_imports.html#b404-import-subprocess
9         import smart_open
10        import subprocess
11        import sys

--------------------------------------------------
>> Issue: [B602:subprocess_popen_with_shell_equals_true] subprocess call with shell=True
identified, security issue.
   Severity: High   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-docker/postgres/backup-to-s3.py:87
   More Info:
https://bandit.readthedocs.io/en/latest/plugins/b602_subprocess_popen_with_shell_equals_true
.html
86            process = subprocess.Popen(
87                BACKUP_COMMAND, shell=True, stdout=subprocess.PIPE)
88            while True:
89                chunk = process.stdout.read(CHUNK_SIZE)
```

```
--------------------------------------------------
>> Issue: [B605:start_process_with_a_shell] Starting a process with a shell, possible injection
detected, security issue.
   Severity: High   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-docker/redis/backup-to-s3.py:72
   More Info:
https://bandit.readthedocs.io/en/latest/plugins/b605_start_process_with_a_shell.html
71
72        os.system("{backup_command} && s3cmd put --multipart-chunk-size-mb={chunk_size}"
73                  " /srv/backups/{source} s3://{bucket}/{filename}"
74                  " && rm -rf /srv/backups/{source}".format(
75            backup_command=BACKUP_COMMAND,
76            bucket=AWS_BUCKET,
77            chunk_size=CHUNK_SIZE,
78            filename=filename,
79            source=DUMPFILE
80        ))

--------------------------------------------------
>> Issue: [B404:blacklist] Consider possible security implications associated with subprocess
module.
   Severity: Low   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-install/helpers/cli.py:5
   More Info:
https://bandit.readthedocs.io/en/latest/blacklists/blacklist_imports.html#b404-import-subprocess
4         import re
5         import subprocess
6         import sys

--------------------------------------------------
>> Issue: [B322:blacklist] The input method in Python 2 will read from standard input, evaluate
and run the resulting string as python source code. This is similar, though in many ways worse,
then using eval. On Python 2, use raw_input instead, input is safe in Python 3.
   Severity: High   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-install/helpers/cli.py:51
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b322-input
50                text = cls.get_message_with_default(message, default)
51                input_ = input(cls.colorize(text, color))
52

--------------------------------------------------
```

```
>> Issue: [B603:subprocess_without_shell_equals_true] subprocess call - check for execution
of untrusted input.
   Severity: Low   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-install/helpers/cli.py:77
   More Info:
https://bandit.readthedocs.io/en/latest/plugins/b603_subprocess_without_shell_equals_true.html
76              if polling:
77                  process = subprocess.Popen(command, stdout=subprocess.PIPE, cwd=cwd)
78                  while True:

--------------------------------------------------
>> Issue: [B603:subprocess_without_shell_equals_true] subprocess call - check for execution
of untrusted input.
   Severity: Low   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-install/helpers/cli.py:88
   More Info:
https://bandit.readthedocs.io/en/latest/plugins/b603_subprocess_without_shell_equals_true.html
87                  try:
88                      stdout = subprocess.check_output(command, universal_newlines=True,
cwd=cwd)
89                  except subprocess.CalledProcessError as cpe:

--------------------------------------------------
>> Issue: [B404:blacklist] Consider possible security implications associated with subprocess
module.
   Severity: Low   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-install/helpers/command.py:6
   More Info:
https://bandit.readthedocs.io/en/latest/blacklists/blacklist_imports.html#b404-import-subprocess
5       import time
6       import subprocess
7
8       from helpers.cli import CLI

--------------------------------------------------
>> Issue: [B603:subprocess_without_shell_equals_true] subprocess call - check for execution
of untrusted input.
   Severity: Low   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-install/helpers/command.py:367
   More Info:
https://bandit.readthedocs.io/en/latest/plugins/b603_subprocess_without_shell_equals_true.html
366             command.extend(args)
367             subprocess.call(command, cwd=config.get("kobodocker_path"))
```

```
368

--------------------------------------------------
>> Issue: [B603:subprocess_without_shell_equals_true] subprocess call - check for execution
of untrusted input.
   Severity: Low   Confidence: High
   Location: /Users/ajd2203/kobo/kobo-install/helpers/command.py:382
   More Info:
https://bandit.readthedocs.io/en/latest/plugins/b603_subprocess_without_shell_equals_true.html
381             command.extend(args)
382             subprocess.call(command, cwd=config.get("kobodocker_path"))

--------------------------------------------------
```