Tools for Enforcing Secure Code

Daniel M Beck

UMGC – SDEV-360

July 27, 2020

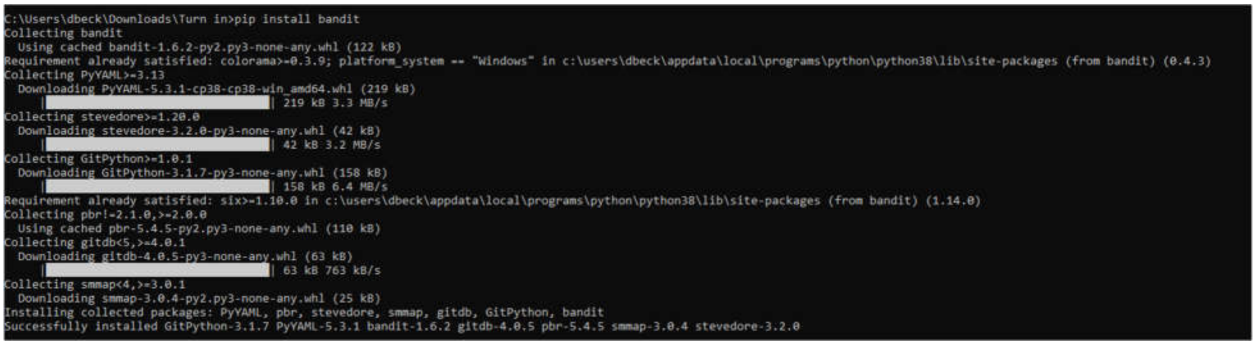
Tools for Enforcing Secure Code

Introduction - What is Security Testing?

With the greater need for secure software came tools that were developed to assist in the process of identifying vulnerabilities in code development. Robert Secord and Robert Schiela state, as one of the top 10 secure code practices, that businesses should “develop and/or apply a secure coding standard for your target development language and platform” (Seacord, R. & Schiela, R., 2018) to help reinforce the security of the developed code. For companies that use the Python development language, Bandit has been found to be a beneficial tool.

Bandit

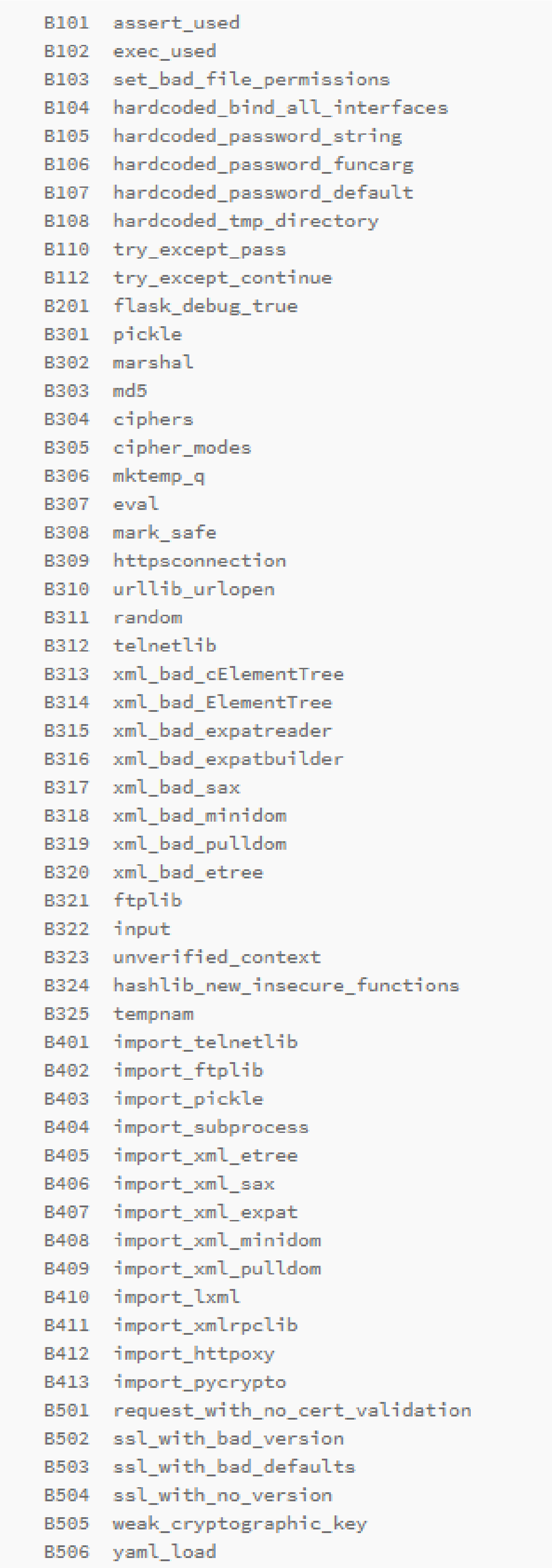
Bandit scans developers code for any known vulnerabilities and then provides explicit feedback about what has been found, the severity of the problem, and how confident it is in its discovery. Bandit is “great for catching issues like insecure configurations, known insecure module usage, hard-codes credentials, asserts, and much more” (Scott, 2020). A few reasons that Bandit is so popular, is that it is a free, open-source software that is configurable, and fast. Figure 1 shows how to install Bandit if the developer is running Python on their machine.



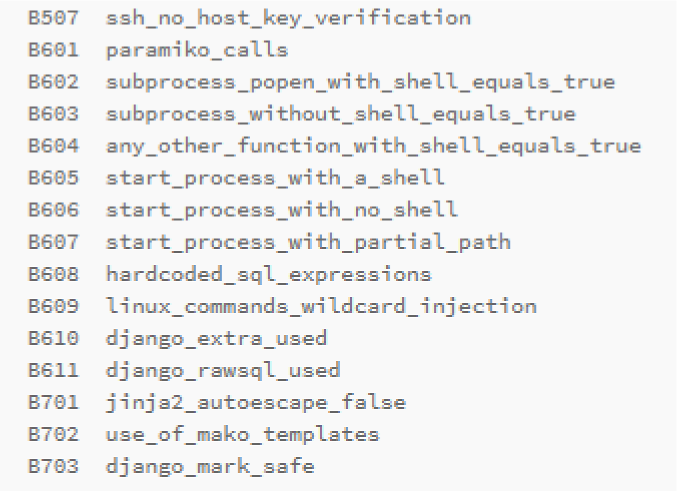
```
C:\Users\dbeck\Downloads\Turn in>pip install bandit
Collecting bandit
  Using cached bandit-1.6.2-py2.py3-none-any.whl (122 kB)
Requirement already satisfied: colorama>=0.3.9; platform_system == "Windows" in c:\users\dbeck\appdata\local\programs\python\python38\lib\site-packages (from bandit) (0.4.3)
Collecting PyYAML>=3.13
  Downloading PyYAML-5.3.1-cp38-cp38-win_amd64.whl (219 kB)
     |████████████████████████████████| 219 kB 3.3 MB/s
Collecting stevedore>=1.20.0
  Downloading stevedore-3.2.0-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 3.2 MB/s
Collecting GitPython>=1.0.1
  Downloading GitPython-3.1.7-py3-none-any.whl (158 kB)
     |████████████████████████████████| 158 kB 6.4 MB/s
Requirement already satisfied: six>=1.10.0 in c:\users\dbeck\appdata\local\programs\python\python38\lib\site-packages (from bandit) (1.14.0)
Collecting pbr!=2.1.0,>=2.0.0
  Using cached pbr-5.4.5-py2.py3-none-any.whl (110 kB)
Collecting gitdb<5,>=4.0.1
  Downloading gitdb-4.0.5-py3-none-any.whl (63 kB)
     |████████████████████████████████| 63 kB 763 kB/s
Collecting smmap<4,>=3.0.1
  Downloading smmap-3.0.4-py2.py3-none-any.whl (25 kB)
Installing collected packages: PyYAML, pbr, stevedore, smmap, gitdb, GitPython, bandit
Successfully installed GitPython-3.1.7 PyYAML-5.3.1 bandit-1.6.2 gitdb-4.0.5 pbr-5.4.5 smmap-3.0.4 stevedore-3.2.0
```

Figure 1, Installing Bandit with the command prompt

Figure 2 shows the list of tests that Bandit runs to discover vulnerabilities. These tests are created from known; common vulnerabilities that have been encountered.



```
B101  assert_used
B102  exec_used
B103  set_bad_file_permissions
B104  hardcoded_bind_all_interfaces
B105  hardcoded_password_string
B106  hardcoded_password_funcarg
B107  hardcoded_password_default
B108  hardcoded_tmp_directory
B110  try_except_pass
B112  try_except_continue
B201  flask_debug_true
B301  pickle
B302  marshal
B303  md5
B304  ciphers
B305  cipher_modes
B306  mktemp_q
B307  eval
B308  mark_safe
B309  httpsconnection
B310  urllib_urlopen
B311  random
B312  telnetlib
B313  xml_bad_cElementTree
B314  xml_bad_ElementTree
B315  xml_bad_expatreader
B316  xml_bad_expatbuilder
B317  xml_bad_sax
B318  xml_bad_minidom
B319  xml_bad_pulldom
B320  xml_bad_etree
B321  ftplib
B322  input
B323  unverified_context
B324  hashlib_new_insecure_functions
B325  tempnam
B401  import_telnetlib
B402  import_ftplib
B403  import_pickle
B404  import_subprocess
B405  import_xml_etree
B406  import_xml_sax
B407  import_xml_expat
B408  import_xml_minidom
B409  import_xml_pulldom
B410  import_lxml
B411  import_xmlrpclib
B412  import_httpoxy
B413  import_pycrypto
B501  request_with_no_cert_validation
B502  ssl_with_bad_version
B503  ssl_with_bad_defaults
B504  ssl_with_no_version
B505  weak_cryptographic_key
B506  yaml_load
```



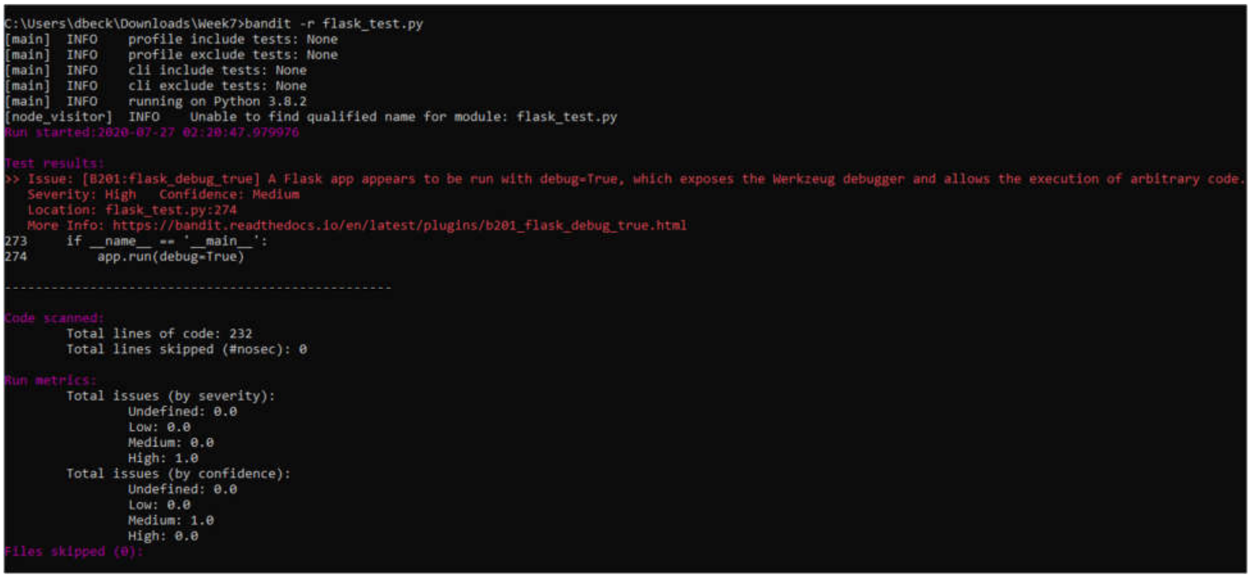
```
B507  ssh_no_host_key_verification
B601  paramiko_calls
B602  subprocess_popen_with_shell_equals_true
B603  subprocess_without_shell_equals_true
B604  any_other_function_with_shell_equals_true
B605  start_process_with_a_shell
B606  start_process_with_no_shell
B607  start_process_with_partial_path
B608  hardcoded_sql_expressions
B609  linux_commands_wildcard_injection
B610  django_extra_used
B611  django_rawsql_used
B701  jinja2_autoescape_false
B702  use_of_mako_templates
B703  django_mark_safe
```

Figure 2, Security tests that are executed when a file is run through bandit. Retrieved from pypi.org/project/bandit

The only disadvantage to using Bandit would be that it does not catch every vulnerability, so it would be good practice to use multiple tools to test the software.

Use of Bandit with Python Program that Runs a Simple Website with Flask

Another part of the simplicity of Bandit is how to run it. With the correct directory selected in the command prompt, typing “bandit -r filename.py” runs the selected code through Bandit. In a previous course, I wrote a project that runs a simple website, using Flask, that navigates through four pages. Figure 3 shows that running that file through Bandit caught an issue with test B201:flask_debug_true. This error occurred because as I was testing the website, it was easier to leave the code in debug mode so that I did not have to refresh the browser for every change. This would be an easy oversight when completing the code.

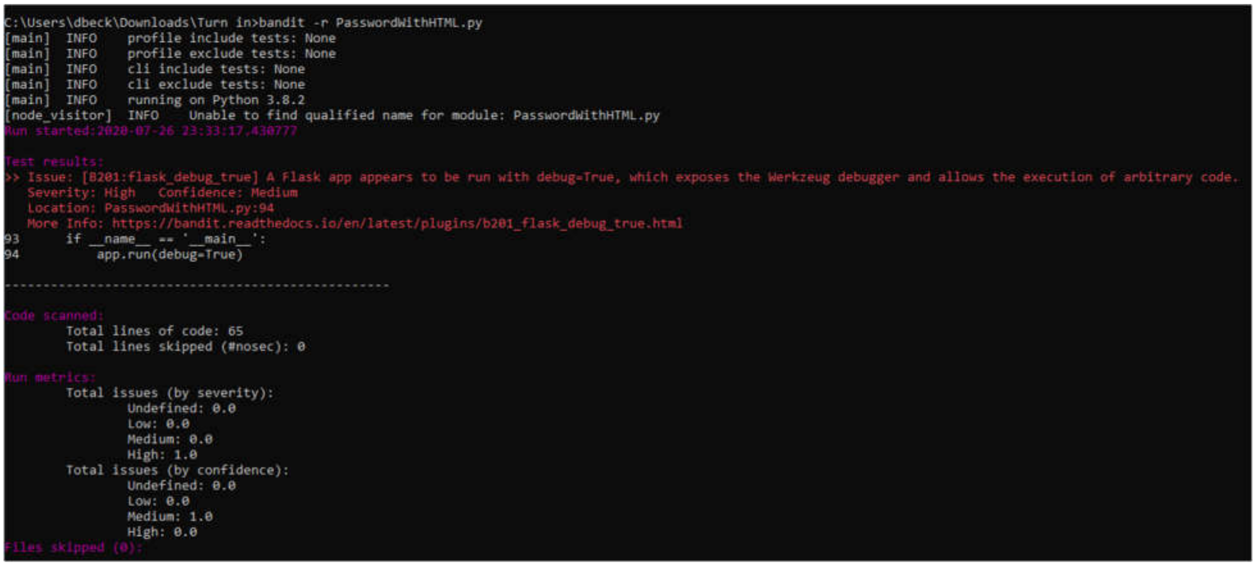


```
C:\Users\dbeck\Downloads\Week7>bandit -r flask_test.py
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.8.2
[node_visitor]  INFO    Unable to find qualified name for module: flask_test.py
Run started:2020-07-27 02:20:47.979976

Test results:
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
   Severity: High   Confidence: Medium
   Location: flask_test.py:274
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b201_flask_debug_true.html
273     if __name__ == '__main__':
274         app.run(debug=True)

--------------------------------------------------

Code scanned:
        Total lines of code: 232
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0.0
                Low: 0.0
                Medium: 0.0
                High: 1.0
        Total issues (by confidence):
                Undefined: 0.0
                Low: 0.0
                Medium: 1.0
                High: 0.0
Files skipped (0):
```

Figure 3, Running Bandit on code that runs a simple website

Use of Bandit with Python Program that Checks Password Complexity

Another project from a previous course that I ran through the Bandit tool was a program that asks the user for a password, then checks that the password is complex enough and that it did not match any common passwords. Figure 4 shows that running the password program through Bandit caught the issue with test B201:flask_debug_true again. Using this tool while attending this course seem like it would have been useful as it looks like a habit was being created of not setting debug mode to false before completing the project.



```
C:\Users\dbeck\Downloads\Turn in>bandit -r PasswordWithHTML.py
[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 3.8.2
[node_visitor]  INFO    Unable to find qualified name for module: PasswordWithHTML.py
Run started:2020-07-26 23:33:17.430777

Test results:
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
   Severity: High   Confidence: Medium
   Location: PasswordWithHTML.py:94
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b201_flask_debug_true.html
93      if __name__ == '__main__':
94          app.run(debug=True)

--------------------------------------------------

Code scanned:
        Total lines of code: 65
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0.0
                Low: 0.0
                Medium: 0.0
                High: 1.0
        Total issues (by confidence):
                Undefined: 0.0
                Low: 0.0
                Medium: 1.0
                High: 0.0
Files skipped (0):
```

Figure 4, Running Bandit on code that checks complexity of password

Conclusion

Bandit, and other tools like it, are growing exceedingly important to utilize since they “uncover vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders” (Software Testing: Security Testing, 2019). This tools also ensure that the software system and application are free from any threats or risks that can cause a loss of data or leak. Using this tool with any Python code focuses on finding all possible loopholes and weaknesses of the code which might result into the loss of information or the corruption of programs.

References

Bandit. (2019, July 1). Retrieved July 28, 2020, from https://pypi.org/project/bandit/

Seacord, R. & Schiela, R. (2018, May 02). Top 10 Secure Coding Practices. Retrieved July 26, 2020, from https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices

Scott, A. (2020, February 25). Top Python Tools for Developing Secure, Quality Code. Retrieved July 27, 2020, from https://levelup.gitconnected.com/top-python-tools-for-developing-secure-quality-code-4b3f5ec1e2de

Software Testing: Security Testing. (2019, May 10). Retrieved July 28, 2020, from https://www.geeksforgeeks.org/software-testing-security-testing/