5-16-2022

Investigation of Python Variable Privacy

Joshua Bartholomew

UNIVERSITY OF NEW HAVEN
HONORS PROGRAM

2021-2022 Honors Thesis

Investigation of Python Variable Privacy

Joshua Bartholomew

A thesis presented in partial fulfillment of the requirements of the Undergraduate Honors Program at the University of New Haven.

Thesis Advisor:	L Page	Digitally signed by Liberty Page Date: 2022.05.15 21:49:23 -04'00'
	(Signature)	
Department Chair:	Ali Golbazi	Digitally signed by Ali Golbazi Date: 2022.05.16 10:43:35 -04'00'
	(Signature)	
Honors Program Director:	Matthew Wranovix	Digitally signed by Matthew Wranovix Date: 2022.06.09 09:21:35 -04'00'
	(Signature)	

5-16-22

Date

Investigation of Python Variable Privacy

Joshua Bartholomew[1,2]
Advisor: Liberty Page[1,3]

[1] University of New Haven, West Haven, USA
[2] jbart9@unh.newhaven.edu
[3] lpage@newhaven.edu

Abstract. This study looks at the relative security of Python regarding private variables and functions used in most other programming languages. Python has only grown in popularity due to its simple syntax and developing capabilities. However, little research has been published about how secure Python code and programs compiled from Python code actually are. This research seeks to expose vulnerabilities in Python code and determine what must be done for these vulnerabilities to be exploited by hackers to abuse potentially sensitive information contained within the program.
The proposed methodology includes examining the private variable concept in other programming languages and conducting experiments to determine whether Python has any vulnerabilities specific to a lack of private variables and functions. Based on the findings of these experiments, further research will be needed to explore the range of vulnerabilities in Python code and how to protect against exploiting these vulnerabilities.

Keywords: Python · Private Variables · Secure Coding · Coding Vulnerabilities · Cybersecurity

1 Introduction

Due to the explosion of the computer industry, software development has become essential to the modern economy. As such, the United States Bureau of Labor Statistics [5] predicts the number of software developer jobs to jump twenty-two percent from 2020 to 2030. Even smartphones are computers and require the use of software as stated in [14]. Programmers often use programming languages to develop software to accomplish specific tasks provided to them based upon the needs of the computer according to IBM [4].

Whitney et al. [19] claims that Python is the number one programming language to date. However, Python does have one design difference from most other modern programming languages. Most programming languages allow the programmer to specify whether a variable or function is private or public, but Python does not allow variables to be defined with specific properties according to [1]. Not having private variables opens Python code up to numerous vulnerabilities and other dangers.

Because this language is so popular, it is critical that developers are aware of the weaknesses Python has, so they can make informed decisions when choosing a programming language for a particular project. This research explores and documents the specific vulnerabilities that Python has because it does not support private variables.

2 Background

Commonly used programming languages are usually object-oriented languages, which means that programmers write their code in a manner which "packages" related information and operations into units called objects. These objects are developed by what are called classes, which specify exactly what pieces of information an object will have "packaged" inside of it. Each piece of information is referred to as a variable, and each operation is called a function.

As mentioned, in 3rd generations languages besides Python, variables can be specified as private. With the use of private variables, only the code within the class of the variable can access it. Only other functions packaged within the class can read or change the value of that private variable. The only access for code outside of the class is through public functions within the class, which are under the complete control of the class writer so that the values cannot be read or altered in any way that the class is not specifically designed to allow.

Public variables in a class, on the other hand, are completely available for any part of the program to read, alter, or set to a new value. While these variables do provide easier access in other parts of the code, they have no safeguards to protect the values of these variables from being read or altered at an inappropriate time or in an insecure fashion. A lack of protection is especially dangerous when a program is written by multiple programmers. Unlimited access means that one programmer might alter variables in a manner that causes errors or may even perform operations that should be forbidden.

Python does not allow any variable to be specified as private, leaving all variables as public in every situation. This leaves programmers to wonder how secure Python is and to what attacks Python is vulnerable if it does not have the protection that comes from private variables.

3 Related Work

This review will start with a section on why Python is so widely used and why it is still in wide use even though it has this potential weakness being discussed. The next section will discuss the uses of private variables including the cybersecurity concept of concealing information found in [2]. The third and final section of the review will consider some uses of Python to provide context of how Python is being implemented. This context both shows the usefulness of this research and attempts to provide some security to Python variables.

3.1 Python's Usefulness:

Python has been around for over three decades according to [3] and has been growing in popularity recently due to its easy-to-understand programming style and simplicity. Lindstrom [13] gives numerous accounts of how Python coding is much easier than most other languages, not requiring numerous lines of code to be written before your first program which simply prints out "Hello World" can be properly compiled and run. Python also has the functionality from being an interpreted language. An interpretive language is one which does not need to be completely written before the entire program is turned into computer code and run. In this language, the code can be run as it is written, so a programmer can execute a piece of code and read its output before writing other lines of code which further use those same variables and functions without having to restart the entire program. Python has many benefits when it comes to writing basic code and accomplishing a task, but the simplicity comes at the cost of the protections private variables and functions offer. There is a time and a place to use Python, and this research will shed light on when it should not be used.

This idea of functionality correlates with Westra's book [18] which reviews the ability of Python for modular programming. Modular programming simply means that pieces of code are organized into packets or "modules" so that they can be easily found and called at an appropriate time. With a large section of code kept together in a large packet, the module can be reused at an appropriate time easily. The author goes over how to use modular programming in Python to accomplish tasks through patterns such as "divide-and-conquer, abstraction, encapsulation, wrappers, and extensibility". The book ends with a look at how to test and then deploy modules once they have been written, including how to upload them to GitHub or even the Python Package Index, which is a well-used site for distributing Python modules.

What [18] teaches is near the center of this research. Modular programming is similar to object-oriented programming. And just as with the object-oriented programming style for Python, there is no way to guarantee protection of any value or function inside of a module in Python. There can be situations where a function should not be executed except under explicit circumstances, but it is impossible to prevent a programmer from calling any function within a module at any time.

Further research has proven that Python makes programming easier to learn and understand. One study by Jayal et al. [11] sought to determine whether starting a high-school student with learning Python to handle the underlying programming ideas in most languages before moving into Java to learn object-oriented coding would help students master programming faster. Teaching students with and without Python for a year yielded the results that starting with Python did improve students' understanding of programming concepts. The results were derived from quantitative analysis of the grades students received from the two years, with numerical results that showed an increase in the success rate when Python was used.

Nikula et al. [15] discusses teaching programming to students at the college level, and whether Python as a starting language is beneficial. The scope of the article looks at how different responsibilities of variables can be better taught using Python to give less experienced programmers a better grip of how they work. These responsibilities include "stepper", "fixed-value", "gatherer", and many others. The researchers concluded that the results supported their hypothesis after looking at examples of different computer science courses at three different colleges. All three colleges reported a quantitative analysis which showed either an increase in the number of students who successfully completed their program assignments or a decrease in the number of students who dropped out of or failed the course.

Although Python is a beneficial starting language for inexperienced programmers, it does not provide them a proper understanding of how every variable in every language has attributes such as the amount of space used to store the value and the type of data being stored such as a number or a string of text. This weakness causes problems when studying different languages. For example, Python does not have a constant variable attribute which prevents programmers from changing the value after the variable has been initialized, such as what C++ and Java has. Constant variables can easily be used for the "fixed-value" declaration, but Python does not have the ability to protect variables at all. While there are ways to get around this problem, they require greater effort than a constant declaration. Though there is no specific responsibility that would directly relate to "private" variables as constant variables, advanced situations do require that variables of many of these responsibilities be kept confidential, even from other programmers.

[15] also references that teaching a language with Python's unique programming syntax might cause early programmers to have problems when trying to learn the syntax of different languages which have similar syntaxes to each other. This difficulty might push these programmers to stick with the language they learned first, and continued use of Python without recognizing the security threats imposed by no private variables can be dangerous for the user of the software because it exposes data, which is why programmers must be informed about the lack of security for Python code.

3.2 Private Variables/Object-oriented Programming:

Private variables are considered a cornerstone to object-oriented programming and are taught at the college level as can be read in Fischer's textbook [9], an example of a college programming textbook which teaches encapsulation and the importance of private variables. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them, which is a key attribute of object-oriented programming. Khan et al. [12] lists the benefits of object-oriented programming, including "objects, abstract data typing, inheritance, and polymorphism". All of these are principles accepted by and widely used in the coding community. The source continues with several coding examples to display how simply certain situations

can be implemented and how long it takes. One of these examples is a situation of a bank account being inherited by a savings account. Through inheritance, the savings account class contains all the code of a general bank account with some slightly altered functions and values specific for a savings account.

Specifically, [12] argues for the use of object-orientation because of its resemblance to the real world. A programming object can contain all the important information about an object in the real world in its variables. However, some information is inherently private and must be protected, for example a person's SSN and date of birth. If a coding language like Python does not enforce private variables, then it could be as dangerous as if the real-world object had as little protection as the code representation.

Object-oriented programming also is extremely useful in modular programming. Cavaiani [7] looks at the use of modular programming in Java, a common object-oriented programming language. This source is a guidance paper to provide instruction for newer programmers to learn object-oriented programming in Java. Java is one of the top programming languages used which incorporate encapsulation and other private variable features. The primary focus includes inheritance and searching for classes and methods from the Java Class Library.

The concept of inheritance and public classes and methods are key to this research, especially for what is hidden behind these methods and what the programmer can do in programming languages such as Java to greatly restrict what other coders can access. Java, like Python, has a list of built-in libraries, but downloading and incorporating new Java modules from other users into a program can be much more difficult than with Python modules.

"Object-Oriented Programming" by Dyke, Richard, and Kunz [17] looks at the use of object-oriented programming as early as 1989. This programming style is the industry standard as of today, and [17] explains the reasons for using this methodology. According to Booch [6], one of the guiding principles of object-oriented programming is encapsulation. Encapsulation hides the value or state of a structured data object inside a class, which withholds access. Without private variables, encapsulation is impossible to implement in a program.

3.3 Python Uses/Security Patch Attempts:

Python is also well-known for allowing users to easily create and distribute modules. Unfortunately, this level of distribution does lead to many potential threats listed by Rahman et al. [16]. [16] references several "security smells" found in thirty-one percent of Python Gists. Python Gists allow programmers to distribute code segments. These smells are lines of code that open the code to attacks and can range from having a password hard-coded into the code segment to not using encryption with network connections which opens the program to network attacks such as a man-in-the-middle attack where a hacker intercepts data between two parties without either party ever knowing.

Many of these "security smells" can be further exploited because of the lack of "private" variable protection. The most obvious example is the hard-coded password which the programmer can access through a variable call even if he

cannot get the value directly from the file. With one line of code, the programmer can list all the variables of a class object. If the hacker sees a variable name that appears promising, he can call that name and get the value, which in this case would be a password that the hacker can use in a different attack.

There are some common security methods to control access such as role-based access control models. Chou [8] looks at the role-based access control (RBAC) model of providing security for variables and other information contained within code. This model was modified for object-oriented programming specifically, so that it could be incorporated with most major object-oriented languages such as Java, which is used as an example in the paper. This method of coding allowed the authors to provide tight restrictions on the data being used by the program. The program would verify a user's credentials to determine any roles that the user possesses according to the information available to the computer. From these roles, the program can verify whether any of those roles grant the user permission to perform a certain task or if there is a need to block the user from doing something beyond what their roles permit.

[8] provides an example of access control in an object-oriented environment such as Python. The program provided is an example of security on the user level, but not on the code level. Many of the variables and methods are publicly accessible and exposed should a malicious line of code be introduced. There are also private methods which should not be made available to other users under any circumstances, but Python would not allow you to prevent anyone from calling these methods themselves and receiving the results returned. The most important of these methods is a password check method which returns whether a password is valid for an account. Hackers could call this method themselves in an attack to increase the speed of their attack.

The closest attempt to creating private variables in Python is by Gaiman et al. [10], which references a prototype source material that altered the Python infrastructure using another programming language to create Sython. Sython is nearly identical to Python but includes the feature of private variables. The private variables do not work in the way being researched in this paper. Sython creates a style where variables are stored in a remote location with a running server waiting to receive commands from the user. The Sython user code can send commands to be done on these variables, which the server receives and executes. However, the user cannot request the value of any specified private variables. They can only send requests of how to alter it.

[10] explains an attempt to provide security to Python by creating a new language rooted in the Python coding style. This source explores how to provide privacy to Python variables, but in a completely different way. While Sython does offer protection from reading private variables, there appears no mechanism that prevents the client from altering the variable in any way they wish. If they know what value they want that variable to hold, they can send the command to change the value and the server end has no restrictions that stops it from following the instruction.

4 Apparatus

Table 1 lists the hardware and software used in this lab.

Table 1: List of Tools Used

ITEM/PART	VERSION	USAGE
MacBook Pro 2018	macOS Monterey	Primary machine used for testing
Python	3.8.9	Used to run Python scripts
Python IDLE	3.8.9	Used to run Python scripts
	Windows 10 / VirtualBox 6.1	Used to run tests requiring Windows
HxD	2.3	Used to look at active memory in Windows virtual machine
grep Unix command	(BSD grep, GNU compatible) 2.6.0-FreeBSD	Used to search through the Gigabytes of data that was produced during experimentation
xxd Unix command	xxd V1.10 27oct98 by Juergen Weigert	Used to interpret memory dump in Sprint #4

5 Methodology

Research into Python was conducted in sprints, with each sprint based upon the discoveries of previous sprints. Each sprint lasted approximately four weeks. The time was comprised of an initial design phase where the experiments were conceived and planned beforehand. Then the sprint continued with conducting each experiment and documenting the results for analysis, which was then used in the planning phase of the next sprint.

5.1 Planning Phase

This phase consisted of creative thinking regarding the implications of the assumed hypothesis regarding what can be done without the use of private variables. The current understanding of the Python infrastructure was analyzed to develop certain situations which would further reveal to what extent the Python variable and function infrastructure was exposed. There were numerous implications at each sprint and the most practical use had to be deduced before proceeding. A path had to be charted to decide what was deemed the most useful and practical conclusions that the potential experiments would provide.

There also had to be consideration of the amount of information that would be produced to properly plan for how the outputted data would be stored and analyzed. Some output from earlier sprints only consisted of a few dozen lines which could simply be outputted to the terminal or shell and be easily interpreted from there. Others produced gigabytes of data which had to be stored in text files and analyzed for any specific information that would be desirable. Once the amount of data was determined, the experiment had to be designed in a way in which the resulting data could be stored for future analysis in an efficient manner.

This phase of each sprint was given at least two weeks to ensure that enough time was allocated to thoroughly analyze the desired task and minimize the risk of overlooking a significant factor. As stated, there were numerous angles at which this hypothesis could be looked at, but not every angle could be covered before the due date of this thesis. The experiments that would produce the most usable data had to be prioritized to ensure the results would be most useful for both current programmers and future studies.

5.2 Execution Phase

This phase consisted of taking the plans made in the previous planning phase and implementing them in Python with whatever environment or tools were necessary. In each sprint, the Python script had to be written and executed to determine to what extent the hypothesis of this thesis was correct. There were multiple points at which the Python script had to be revised due to unforeseen situations.

The Python script was written and executed in Python IDLE for efficiency and ease of execution. The scripts were then stored in ordinary files with the '.py' extension to make it simpler to recognize and execute them when needed. These files were quite small so transferring them to online drives and virtual machines was relatively quick. Once the script was in the correct location, the execution time increased with each experiment. The first sprint had scripts that would only take a few moments to load and execute, but the rest of the sprints required hours of time for the scripts to finish execution, and then often only ended because of an error due to the length of time and amount of work the script was completing.

In some cases, the custom code had to be copied into other Python executable scripts that had already been written, which were then executed using IDLE. This method allowed experimentation with normal Python scripts which further provided evidence for the use of this thesis in real-world applications.

The results were either printed to the screen or stored to a local text document and then searched using another Python script or via simple grep command. These results were then documented and used for designing future experiments in the next sprint.

6 Results

The following sections discuss what was explored in each sprint and the results of the experiments conducted.

6.1 Sprint #1

The initial stage of the study consisted of discovering and confirming how data is stored in Python during execution. After some research, it was confirmed that every variable and function in Python is stored in a dictionary style object. A dictionary object in Python is simply a list of variables, each assigned a string of text as its name. Functions are also stored in this dictionary linking the name of the function to where the code for the function exists. This allows developers to save data and functions based upon a specific name and recall the same data or function using that same name.

Further research revealed the highest-level dictionary can easily be accessed through a reference to a function called 'globals()'. Every piece of information within the entirety of the program can be traced through this dictionary. Therefore, if you know the names of the variables and functions, you can access any variable or function from any other point in the program. This does not even require foreknowledge when the code is written. The names can be passed into the program at any point and used as normal strings of text within the program which are then used as a reference for the dictionary.

There is also the matter of the common misconception among Python programmers that a variable or function can be made private by adding two underscores to the beginning of the name. Most of the Python programming community encourage the use of the double underscore on any variables or functions that should be considered private. While Python etiquette states that these are private and should be treated as such, the only protection provided is that the name is altered for any part of the program too far away from the code. However, the new name is still listed within a dictionary and is not altered to the point that the original name could not easily be recovered.

To prove the concepts of this sprint, a simple script was written to go through all of the variables in the 'globals()' dictionary, then look through all the dictionaries within those variables. These variables were then printed to the screen with their corresponding name, each line indented to make it easier to read. Below is the text for the program used in Listing 1.1 followed by the output results in Listing 1.2.

```
1 class myClass:
2     def __init__(self):
3         self.__x=5
4         self.y=10
5
6 myObj = myClass()
7
8
```

```
9
10 #VERSION 1 (Recursive)
11
12 def stealAll(location=globals(), output=print, indent='',
       endline=''):
13     myList = location.copy()
14     for name,obj in myList.items():
15         output(indent+str(name)+':'+str(obj)+endline)
16         if hasattr(obj, '__dict__') and not name=='
       __builtins__':
17             stealAll(location=obj.__dict__, output=output,
18                      indent='\t'+indent, endline=endline)
```

Listing 1.1: Python "stealAll" function version 1

```
1  __name__:__main__
2  __doc__:None
3  __package__:None
4  __loader__:<class '_frozen_importlib.BuiltinImporter'>
5    __module__:_frozen_importlib
6    __doc__:Meta path import for built-in modules.
7
8        All methods are either class or static methods to avoid
       the need to
9        instantiate the class.
10
11
12   _ORIGIN:built-in
13   module_repr:<staticmethod object at 0x7fc77c17f430>
14   find_spec:<classmethod object at 0x7fc77c17f460>
15   find_module:<classmethod object at 0x7fc77c17f490>
16   create_module:<classmethod object at 0x7fc77c17f4c0>
17   exec_module:<classmethod object at 0x7fc77c17f4f0>
18   get_code:<classmethod object at 0x7fc77c17f580>
19   get_source:<classmethod object at 0x7fc77c17f610>
20   is_package:<classmethod object at 0x7fc77c17f6a0>
21   load_module:<classmethod object at 0x7fc77c17f6d0>
22   __dict__:<attribute '__dict__' of 'BuiltinImporter' objects
       >
23   __weakref__:<attribute '__weakref__' of 'BuiltinImporter'
       objects>
24 __spec__:None
25 __annotations__:{}
26 __builtins__:<module 'builtins' (built-in)>
27 __file__:/Users/joshuabartholomew/OneDrive - University of
       New Haven/
28           Honors Thesis/Sprint 1 Code (Thief).py
29 myClass:<class '__main__.myClass'>
30   __module__:__main__
31   __init__:<function myClass.__init__ at 0x7fc77fd76ee0>
```

```
  __dict__:<attribute '__dict__' of 'myClass' objects>
  __weakref__:<attribute '__weakref__' of 'myClass' objects>
  __doc__:None
myObj:<__main__.myClass object at 0x7fc77fd57610>
  x:5
  y:10
stealAll:<function stealAll at 0x7fc77fa07ee0>
f:<_io.TextIOWrapper name='text.txt' mode='w' encoding='UTF-8
    '>
  mode:w
```

Listing 1.2: Python "stealAll" function version 1 output

6.2 Sprint #2

The second sprint comprised of using the basic concept proven in Sprint #1 by implementing the code in other Python programs to determine what data could be extracted.

During this process, the "stealAll" function had to be revised to allow the extraction of data from applications much larger than the simple program used in the first sprint. In Listing 1.3, you can see the text of the improved version which not only could read through more data, but could also be used to search the outputted data for a specific keyword to save in the later stage.

```
#VERSION 3 (find)
def stealAll(output=print, indent='\t', endline='', find=''):
    location = globals().copy()
    myList = list(location.keys())
    myList.reverse()
    while myList:
        name = myList.pop()
        data = name.split('\5')
        if len(data) > 10:
            continue
        evalValue = 'location'
        for x in data:
            if '[' in evalValue:
                evalValue = evalValue + '.__dict__'
            evalValue = evalValue + '["' + x+ '"]'
        obj =eval(evalValue)
        new_output = "".join([indent for x in range(0, len(
    data)-1)])
        +str(data[-1])+':'+str(obj)+endline
        if not find or find in new_output:
            output(new_output)
        if hasattr(obj, '__dict__') and not name=='
    __builtins__':
            newlist = list(obj.__dict__.keys())
            newlist.reverse()
```

```
24                for x in newlist:
25                    myList.append(name+"\5"+x)
```

Listing 1.3: Python "stealAll" function version 3 with search feature

All three programs allowed the extraction of a significant amount of data stored in a local text file. Every application only terminated because of a error from running the program for so long on so much data. All of the data can be seen in Table 2 below.

Table 2: List of Applications Tested

APPLICATION	DATA EXTRACTED	SIGIFICANT FINDS
API Integrator by article author	57.59 GB	Found Multiple API Keys
SongList by article author	15.52 GB	Found Google file URL
OpenLP, a free open-source software application (GNU General Public License)	255 MB	Default data such as username, password, and previous searches

As can be seen, there is a significant amount of data that can easily be acquired, and the program only stopped because of miscellaneous errors likely caused by the duration of the program to extract this much data. This data then had to be sifted through using search methods such as the grep Unix command or the Python script find feature added in version 3 of the "stealAll" function as seen in Listing 1.3.

One of the key pieces of information found in every application and can even be seen in the example of Listing 1.2 is the file name and directory of the Python script. Within this directory can be found not only the username of the account where the file is located but also the list of folders which can be used if the file system can be accessed in some other manner.

API Integrator In the first application, the API keys found could easily be used to access data from secure websites. Further, the specific website that was used could also be acquired by the URL and the name of the class the variables were found in. Beyond that, nearly 60 gigabytes of data in raw text format was produced by the application including variable names and corresponding data. Most of this data came from the imported classes used for web uploads and other related operations.

SongList The second application was a more simple application that linked to a file on a personal Google drive to download data. This link was found within the outputted data, though the authorization keys could not be found. The total data amounted to over 15 gigabytes, again most of which was data found within the Google API imports.

OpenLP The last application was a program published free and open source for the use of presenting song lyrics in churches. This application was chosen because of the familiarity the author has with the application and the potential data that could have been acquired. The default credentials as well as searches were found within the application. Lastly, this application only produced 255 megabytes of data as opposed to the gigabytes of data from the other two applications because of its increased complexity.

6.3 Sprint #3

The third sprint them moved on to taking the data extracted from the program and looking at the memory of the related program while the application was still running. If you refer back to Listing 1.1 again, you can see that even the functions of the application are printed with their relative memory address. These memory addresses can be mapped to the volatile memory of the machine.

To complete this research, the API Program was transferred to the Windows 10 virtual machine and run actively with HxD opening the related memory to the Python application. This was done because it was the best available option to look at the memory directly related to the application while the application was active.

The results of this experiment showed that the memory addresses from the data output were valid links to active memory and could therefore be used to locate functions in memory and even be used for reverse engineering of the script.

6.4 Sprint #4

The fourth and final sprint looked at the memory of the computer after the script had been completed and terminated. This was done by creating a memory dump from the MacBook Pro once the API Application was terminated. This was then interpreted using the hex analyzer xxd Unix command.

Unfortunately, the memory address of the data output for the functions were virtual memory addresses. Virtual memory addresses are abstract links to where the actual data is being stored in the memory, called the physical memory address. The only way a virtual memory address can be translated to the physical memory address is through a table used by the program which can be difficult to extract. Therefore, the virtual memory addresses were useless without finding the table or other means to connect the virtual and physical memory addresses.

However, during the analysis of the memory dump, there was found several fragments of the data produced by the "stealAll" function which even included the API keys mentioned before in several locations which could easily be stolen and abused. Not only this, but one could also find the names of variables, functions, and many other custom names which can give a useful insight to the structure and usage of a program. Similar to how variables are linked to names related to what data they store, classes and modules are usually named based upon what type of information they contain. These class and module names can provide insight into what data is contained by the class and module.

7 Conclusion/Discussion

Programming securely has become a necessity in today's technological age with the cybersecurity threats that exist. This need implicates the requirement of vetting all primary methods of developing programs to ensure that any data being handled in the program is properly maintained and protected.

Python is widely used in the programming community, but lacks the ability to protect data through private variables. The danger therefore lies in using Python to develop programs that work with sensitive data that hackers wish to access. Therefore, programmers must be made aware of these vulnerabilities and be provided with better ways of protecting data.

8 Future Work

There are an infinite number of future works that can be derived from this research. The foremost is methodology for protecting data from the access discussed in this paper. The options are various, though a likely possibility is to obfuscate the data so it can only be read by the program that knows how to interpret the variable values. However, this leads to the problem of protecting the code that deciphers the data, since functions and their operations can also be accessed by methodology in this research. Another option to protect the data is to obfuscate the variable names. One of the primary means of deciphering data from this research is by looking at the name of the variable

Another potential future work is the development of tools that can access Python scripts and produce any valuable data found. The Python scripts developed for this research have some functionality for searching through the found variables, but that ability is limited. Better methods to extract the valuable information can certainly be developed and implemented in future works.

A third area of future research is the research into methods of injection-style attacks toward Python scripts and how this research can be used once a payload can be sent into another executing script. Python code does not need to be complete before it is executed, which allows lines of Python code not originally written in the program to be inserted into the program during execution, potentially by those with malicious intent. The better we understand attacks on Python scripts, the better we can protect against future attacks.

The last avenue of future research discussed here is the potential for the forensic analysis of computers that recently executed Python scripts. Authorities may reach computers after a Python script has been executed and the files deleted beyond recovery. With future memory analysis tools based upon this research, authorities can take a memory dump of a computer and search through it for any pieces of the Python script that had been executed. This can provide essential evidence for the prosecution of criminals or other related information to an investigation.

References

1. 6. data types and variables, https://python-course.eu/python-tutorial/data-types-and-variables.php
2. Cyber security principles, https://www.cyber.gov.au/acsc/view-all-content/advice/cyber-security-principles
3. General python faq¶, https://docs.python.org/3/faq/general.html
4. What is software development?, https://www.ibm.com/topics/software-development
5. Software developers, quality assurance analysts, and testers : Occupational outlook handbook (Sep 2021), https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm
6. Booch, G.: Object-oriented analysis and design with applications. The Benjamin/Cummings series in object-oriented software engineering, Benjamin/Cummings Pub. Co. (1994), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=cat04460a&AN=unh.143970&site=eds-live&scope=site
7. Cavaiani, T.P.: Object-oriented programming principles and the java class library. Journal of Information Systems Education **17**(4), 365 – 368 (2006), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=mfi&AN=23720681&site=eds-live&scope=site
8. Chou, S.C.: Embedding role-based access control model in object-oriented systems to protect privacy. The Journal of Systems and Software **71**(1-2), 143–161 (04 2004), http://unh-proxy01.newhaven.edu:2048/login?url=https://www-proquest-com.unh-proxy01.newhaven.edu/scholarly-journals/embedding-role-based-access-control-model-object/docview/229598408/se-2?accountid=8117, copyright - Copyright Elsevier Sequoia S.A. Apr 2004; Document feature - tables; charts; references; Last updated - 2021-09-12; CODEN - JSSODM
9. Fischer, A.: Exploring C++ (2014), http://docplayer.net/23245803-Exploring-c-alice-e-fischer-university-of-new-haven-january-2009-revised-to-september-5-2012-copyright-c-by-alice-e.html
10. Gaiman, M., Simha, R., Narahari, B.: Privacy-preserving programming using sython. Computers & Security **26**(2), 130 – 136 (2007), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0167404806001386&site=eds-live&scope=site
11. Jayal, A., Lauria, S., Tucker, A., Swift, S.: Python for teaching introductory programming: A quantitative evaluation. ITALICS: Innovations in Teaching & Learning in Information & Computer Sciences **10**(1), 86 – 90 (2011), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edo&AN=66717664&site=eds-live&scope=site
12. Khan, E., Al-A'ali, M., Girgis, M.: Object-oriented programming for structured procedural programmers. Computer **28**(10), 48 – 57 (1995), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.467579&site=eds-live&scope=site

13. Lindstrom, G.: Programming with python. IT Professional, IT Prof **7**(5), 10 – 16 (2005), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.1516083&site=eds-live&scope=site
14. Lundberg, J.C.: When is a phone a computer. Washington Journal of Law, Technology & Arts **8**(4), 473 – 486 (2013), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edshol&AN=edshol.hein.journals.washjolta8.28&site=eds-live&scope=site
15. Nikula, U., Sajaniemi, J., Tedre, M., Wray, S.: Python and roles of variables in introductory programming: Experiences from three educational institutions. Journal of Information Technology Education **6**, 199 – 214 (2007), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=eric&AN=EJ807663&site=eds-live&scope=site
16. Rahman, M.R., Rahman, A., Williams, L.: Share, but be aware: Security smells in python gists. 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), Software Maintenance and Evolution (ICSME), 2019 IEEE International Conference on pp. 536 – 540 (2019), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8919248&site=eds-live&scope=site
17. Ten Dyke, Richard, P., Kunz, J.C.: Object-oriented programming. IBM Systems Journal **28**(3), 465 (1989), http://unh-proxy01.newhaven.edu:2048/login?url=https://www.proquest.com/scholarly-journals/object-oriented-programming/docview/222408366/se-2, name - IntelliCorp; Copyright - Copyright International Business Machines Corporation 1989; Last updated - 2021-09-10; CODEN - IBMSA7
18. Westra, E.: Modular Programming with Python. Community Experience Distilled, Packt Publishing (2016), http://unh-proxy01.newhaven.edu:2048/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1242579&site=eds-live&scope=site
19. Whitney, L., Staff, T., Wolber, A., Pernet, C., Alexander, M., Combs, V.: The best programming languages to learn in 2022 (Apr 2022), https://www.techrepublic.com/article/the-best-programming-languages-to-learn-in-2022/