Introducing Secure Design by Scripting in an Undergraduate Microcontroller Based Design Course

Kalyan Mondal
Gildart Haase School of Computer Sciences and Engineering
Fairleigh Dickinson University
Teaneck, NJ, USA
mondal@fdu.edu

Angela Elias-Medina
Gildart Haase School of Computer Sciences and Engineering
Fairleigh Dickinson University
Teaneck, NJ, USA
aaelimed@student.fdu.edu

***Abstract*—This paper discusses a systematic approach to revising a second undergraduate course on microprocessor system design to improve student learning outcomes by introducing scripting-based design with a security mindset. The current course is based upon using the Dragon 12-Plus development system, which requires using compiled C code, and does not offer any on-board security features. The updated course has the intended outcomes of gaining design and technical skills on multiple microcontroller-based design platforms and introduce "security mindset" for networked systems. We introduce a Project Based Learning (PBL) approach, and the focus of the course is on hands-on activities where the students work on multiple design projects using C and MicroPython. The course hardware platform of Dragon 12-Plus is augmented with a small form factor pyboard, which is used to acquire sensor data and transmit securely for simple data analytics. We introduce three new laboratories, including one on data security using MicroPython. We also outline necessary changes to undergraduate engineering programming course sequence. Additionally, mapping of these new labs to CAE-CD KUs and the NICE Framework Specialty Areas is included.**

Keywords—microcontrollers, data security, project based learning, pyboard, MicroPython

I. Introduction

The use of microcontrollers in industrial and net-centric electronic devices has become ubiquitous over the past several years. Electrical and computer engineers need to have an understanding of microcontroller hardware and software systems, their networking abilities, and be able to develop systems with reduced vulnerabilities. Since many such designs end up being connected to the Internet, the students must develop a "security mindset" to avoid these smart devices being hacked. The students should also be able to utilize the secure data transmission, aggregation, and analysis capabilities to design more applications that are useful.

At our college, an undergraduate microprocessor based design course is presented in two-semesters with the first semester course (EENG2287 Microprocessor System Design I) focusing on microprocessor architecture and its programming using the associated assembly language. The second course, EENG3288 Microprocessor System Design II, deals with microcontroller based system design [1] using a high-level language, such as C. Students in the second course learn about a relatively complex 16-bit microcontroller (NXP HCS12 in our case) architecture. They master an application development system (Dragon 12-Plus in our case), and an Integrated Development Environment (IDE) (CodeWarrior in our case) that allows I/O pin level programming in C [2-3]. Typically, students learn how to program parallel ports to control display systems, use real-time interrupts to control external I/O devices, use timers to generate arbitrary waveforms and measure them, use A/D converters to acquire sensor (e.g., temperature, pressure, etc.) data, and use PWM to control DC motors. This is atypical of many colleges, where different microcontrollers and IDEs are used for similar student learning outcomes. The students not only learn how to pursue pin level programming for simpler display devices, but also get an understanding of how to develop C functions for re-use in their programs. In particular, functions for initializing complex displays like LCD and keyboards are discussed and form the foundation of developing device drivers at the hardware level. Finally, students develop application programs using both pin level and function based programming. It is worth noting that students find considerable difficulty developing and debugging an application mainly due to two reasons: (a) develop the full C program and compile to find errors and (b) understand the operation of the interface properly and code to apply proper voltage levels on pins at proper timings. We feel that the reason (a) takes students a lot of time since the compiler messages are quite cryptic. If the students could develop a line of code and try before moving to the next line code, more localized debugging will be possible.

In real world, the engineering graduates may have to use a different microcontroller and its development system than what they used in their colleges and have to spend a considerable amount of time self-learning system interfacing and programming requirements. Some authors [4] have proposed learning multiple development systems with pluggable microcontrollers or even a combination of

This project was supported by the National Security Agency (NSA) under Grant/Cooperative Agreement entitled 'Cybersecurity Workforce Education - CNAP Initiatives' No. H98230-17-1-0321. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation herein.

reconfigurable platform (e.g. CPLD/FPGA) and a microcontroller in the undergraduate program. Thus introducing a second microcontroller and its IDE into the existing course will be pedagogically beneficial to the students.

For the last several years, microcontroller-based embedded systems are being designed to be networkable and get connected to the Internet as part of the deployment in smart devices. Although microcontrollers like HCS12 are networkable using Serial Communication Interface (SCI) and Controller Area Network (CAN) interfaces built-in the HCS12 chip, the Dragon 12-Plus development board does not offer the Internet connectivity. The popular Arduino platform with inexpensive WiFi - enabled ESP8266 microcontroller has provided a simple solution for Internet connectivity and is liked equally by the hobbyists and self-learners. Both require C language programming.

These Internet-ready devices, also known as Internet of Things (IoT), may have vulnerabilities that can be exploited by malicious intruders to breach privacy and/or cause serious damage to the owners of such devices. Many newer microcontroller devices, e.g., NXP Kinetis K8x, provide on-chip cyclic redundancy check (CRC), random number generation (RNG), and symmetric cryptographic encryption/decryption capabilities whereby some of these risks can be minimized. However, these also are C based complex devices.

Another important shift in paradigm has revolutionized the development of IoTs. Single-board credit-card sized Raspberry Pi or similar single-board computers running a full blown operating system like Linux and with WiFi capability has made IoT development a lot easier. This is because of simpler scripting based software that makes an app development a breeze. One has to keep in mind that running a full operating system (OS) does not come without some security risks and overheads. The OS and associated libraries need to be patched, and these single board computers have a much larger attack surface than a simple microcontroller.

In this paper, we outline the changes to the EENG3288 course at our college by making judicial changes to the curriculum, introducing new laboratory equipment and developing new labs to circumvent several issues discussed above. We also discuss the introduction of the Project Based Learning (PBL) paradigm [5] in this course update process. Other colleges can readily adapt these concepts with minor changes.

The current ten specific outcomes of the course include EAC-ABET and ETAC-ABET student outcomes. With the re-design of the course, we will add the following intended student learning outcomes.

Outcome 5.1: Design and program using a second development system based on a current generation 32-b microcontroller and a modern scripting language, e.g., MicroPython.

Outcome 5.2: Use MicroPython for pin level programming to control a display subsystem.

Outcome 5.3: Develop a timer based display controller system.

Outcome 5.4: Gather sensor data, encrypt, and transmit over a network.

Outcome 6.1: Pursue team design projects (PBL), as specified, extending the knowledge gained from lab projects.

II. Design of the Course Contents

Both Electrical Engineering and Electrical Engineering Technology majors take the EENG3288 Microprocessor System Design II course. A majority of them complete two programming courses prior to taking this course, namely, ENGR1204 Programming Languages in Engineering and ENGR3200 Advanced Engineering Programming. Students learn solving engineering problems in the ENGR1204 course using MATLAB scripting language. In ENGR3200, the students learn, using the compiled language C++, to solve engineering problems. Apart from procedural programming in ENGR3200, they get introduced to the object-oriented programming, exception handling, and input data validation. These students join the EENG3288 class with some basic concepts of secure software development using C++. In EENG3288, the students develop C code on CodeWarrior and download the executable onto the flash ROM of Dragon 12-Plus for execution.

A. *Introducing a Second Platform – pyboard into EENG3288*

As discussed earlier, introduction of a second development platform in EENG3288 is highly desirable and will help satisfy the intended Outcomes 5.1 through 5.4.

Based on our experience in developing labs for a graduate course, EENG7709 Embedded Systems [6], we chose to introduce a Python-based microcontroller development system, namely pyboard, in the undergraduate EENG3288 course. Although EENG7709 uses Raspberry Pi single-board computer based labs, the Raspberry Pi appears to be a rather complex platform for simple I/O pin level programming, sensor data gathering, and sensor data transmission used in the undergraduate EENG3288 course. Since applications developed for Raspberry Pi run under the Linux OS, the system and power overheads are larger than those on the pyboard. MicroPython runs bare-metal on the pyboard, and essentially provides a Python operating system. The built-in pyb module contains functions and classes to control the peripherals available on the board, such as UART, I^2C, SPI, ADC and DAC. It connects to the PC over USB, giving a USB flash drive to save MicroPython scripts, and a serial Python prompt (a REPL) for instant programming. This allows anyone to instantly type and execute MicroPython commands, just as one would when running Python on the PC.

The C-language based program development and debugging required for the Dragon 12-Plus can be relatively more time consuming and tedious. On the other hand, network interfaces and most on board features on pyboard are similar to those extended by Dragon 12-Plus, except that the pyboard is a lot more powerful.

1) *Pyboard and MicroPython*

The inexpensive pyboard is a powerful microcontroller based board with many useful features including a hardware random number generator. MicroPython supported on the pyboard is a subset of the Python 3 programming language optimized to run on microcontrollers. Over the last few years, Python has become the number 1 choice for programming and embedded application development. MicroPython [7] being a subset of Python allows simple tweaking to port applications developed using Python.

The MicroPython organization provides detailed tutorials on running the scripts over pyboard. These together with many blogs and application notes helped in developing the labs described later in this paper and referenced therein.

2) *Integrating Python into the Undergraduate Curriculum*

Starting Spring 2019, an introduction to Python in addition to MATLAB scripting started in ENGR1204 as shown in Figure 1. The following course, ENGR3200, will introduce object oriented programming concepts of Python starting the Fall 2019 semester. In the Spring 2020 semester, MicroPython will be introduced in EENG3288, enabling the Outcomes 5.1 through 5.4.

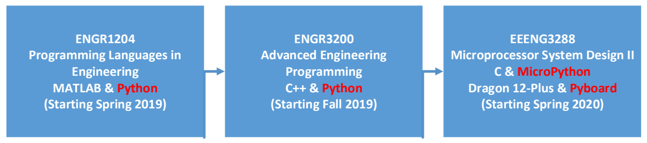

Fig. 1. Introducing Python in the Undergraduate Curriculum

B. *Developing New Labs on pyboard*

As mentioned earlier, the pyboard was chosen to cover Outcome 5.1. We developed the following three new labs to cover Outcomes 5.2 through 5.4.

1. *Lab 1: Seven-segment display of patterns by pin programming*

This lab uses a 4-digit 4D75 7-segment display unit with leftmost through right digits denoted as DSP1, DSP2, DSP3, and DSP4. The pin connections between the display device and the pyboard are to be done properly and the pyboard pins need to be set to the 'out' direction. Obviously, the displays will turn off when a logical 'high' (i.e., high voltage level) is applied to the corresponding cathode pins.

We chose to display the digit '9' on DSP2 and '6' on DSP3 as shown in Figure 2. The segment pins are multiplexed between all four devices. So we need to make sure that the segments corresponding to the digit '9' (a, b, c, f, and g) on DSP2 and those corresponding to the digit '6' (c, d, e, f, and g) on DSP3 are refreshed at a high enough rate to provide a steady display using the principle of "persistence of vision". So a delay of 8.5 ms (Python `timer.sleep(.0085)`) that corresponds to ~118 Hz was used.

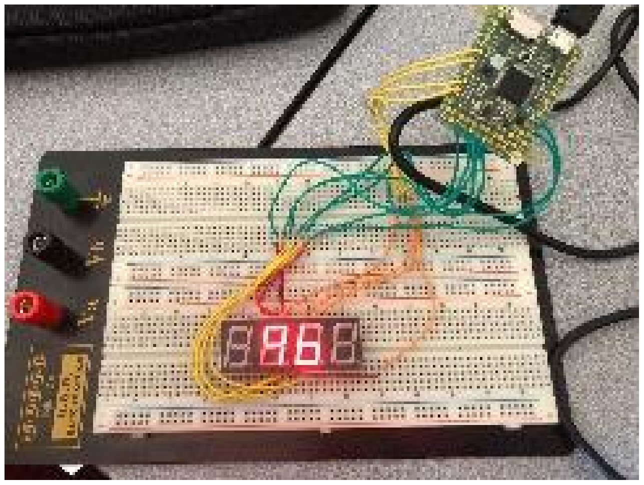

Fig. 2. Pyboard display of '96' on 7-segment displays

```
# Script to display 9 on DSP2 & 6 on DSP3
# -------------- Setup Functions ------------ #
def set_output(pinArr,pinName,pinDict):
 "Sets pins for output and stores values in a dictionary."
 for i in range(len(pinArr)):
 pinDict[pinName[i]] = Pin(pinArr[i],pyb.Pin.OUT_PP)
```

Three setup functions are needed to declare specific pyboard pin direction as output (set output), turning all displays off (seg7_off) and turning a digit display on one of the display device (seg7_disp). All of them use the dictionary feature of Python to save and retrieve Pin, Segment, and Digit information. One of the setup functions is listed above for reference.

The overall digit '96' display script is straightforward and is available from the author upon request.

In the lab, we will ask students to run the supplied program. After familiarization, they will be asked to modify it to display different digits on the four 7-segment displays. We can also assign more complicated displays including changing patterns without changing any connectivity.

2. *Lab 2: Timer cycle based traffic light controller*

This lab involves cycling through three external LEDs lighting in the sequence: RED -> RED -> RED -> RED. This is followed by the GREEN -> GREEN -> GREEN -> GREEN sequence. It is followed by YELLOW. Each LED stays ON for a specified interval.

The MicroPython script involves declaring a "Semaforo(object)" class in which the initialization of the displays and the timers and an interrupt service routine need to be defined. Following this, the main program starts by turning on one of the displays followed by instantiating the previously declared “Semaforo()” class. Figure 3 shows the hardware interconnection for this lab.

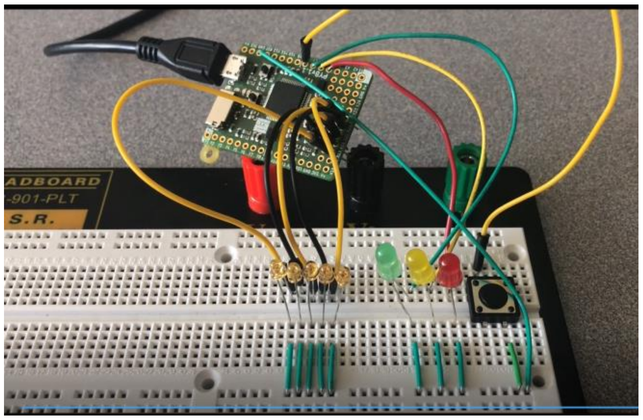

Fig. 3. Pyboard with external LEDs for Traffic Light Controller design

Within a class declaration of “Semaforo(object)”, an initialization function called “_init_” needs to be defined. Apart from initializing the LED devices counter called “count”, a timer cycle counter called “cyclecount” also needs to be initialized to zero. A timer object has to be initialized as Timer 2 to have a frequency close to 1 Hz. By passing the parameter values of prescaler=10 and period=10000000, the timer will trigger interrupts at the 84 MHz / 10 / 10000001 ~ 0.84 Hz rate. An interrupt service routine called “signal” has to be used for timer callback. So the top part of the MicroPython code reads as follows:

```
# Timer Based Simple Traffic Light Controller

# Assuming both RED & GREEN LEDs stay ON 4 times as long
as YELLOW

import pyb, micropython

micropython.alloc_emergency_exception_buf(100)

class Semaforo(object):

 def __init__(self):

 self.led = [1,2,3,1] # R,G,Y,R

 self.count = 0 # Initialize led count

 tim = pyb.Timer(2,prescaler=10,period=10000000)

# default single period

 self.cyclecount = 0 # Initialize cycle count

 tim.callback(self.signal)
```

Next, we need to define the interrupt service routine “signal”. The coding is done in a long hand manner with specific cycle counts. First, increment CYCLECOUNT. Then check if CYCLECOUNT = 4 to turn GREEN LED ON, else if CYCLECOUNT = 8 then turn YELLOW LED ON, else if CYCLECOUNT = 9 then turn RED LED ON. Finally, reset all counters to restart the cycle. The code is available from the author by request.

The program can be started and run in a simple manner. In the lab, students will be first running the supplied program. After familiarization, they will be asked to modify it to turn on varying number of LEDs in a specified sequence of variable durations. Since there is no external control or data transfer, security risks posed by this application is minimal.

3. *Lab 3: Encrypting temperature/pressure sensor data for porting & processing*

This is the most complicated of the three labs. Here we will only outline important aspects of this application and the script. Actual script will be provided on request. The basic requirement is to collect a set of temperature readings at regular intervals, save them in a .csv file after encryption, transmit to the PC for decrypting and displaying.

Sensor: This lab uses an external DHT-11 temperature/pressure sensor. As shown in Figure 4, Pin 1 (VCC) of the sensor is connected to 3V3, Pin 2 (DATA) is connected to Y9, and Pin 4 (GND) is connected to GND on the board. In order to use the DHT-11 effectively, we used the dht library created by polygontwist on GitHub to set up the sensor. The file “dht.py” needs to be copied to the Pyboard for the sensor to work. Figure 4 shows the overall interconnection of pyboard, DHT-11, and other subsystems for this exercise.

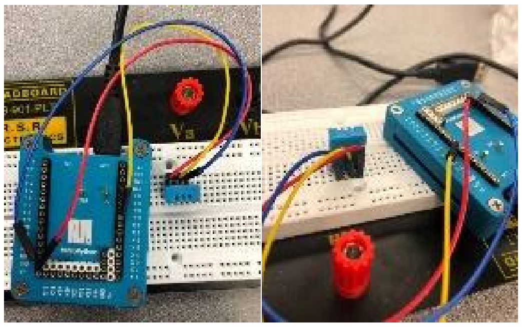

Fig. 4. A Complete Setup for Lab 3

Real Time Clock (RTC): This clock class tracks the date and time. To set up the RTC, datetime() function must be used with the format (year, month, day, weekday, hour, minute, second, millisecond). The weekday is specified by the numbers 1-7, with Monday corresponding to 1 and Sunday to 7. RTC is a 24-hour clock, so the hour must be given using military time. Datetime() can also be used to get the current date and time. The script fragment is listed below:

```
import pyb

import dht

dht.init() # Initialize sensor

# Get values for date and time.

print('Enter numerical values for date and time.')

year,month = int(input('Year: ')), int(input('Month: '))

day,wday = int(input('Day: '))

, int(input('Weekday (1-7;Mon-Sun): '))

hour,minute = int(input('Hour (24hr/day): '))

, int(input('Minute: '))

second = int(input('Second: '))
```

```
# Set up RTC with given values
rtc = pyb.RTC()
rtc.datetime((year,month,day,wday,hour
 ,minute,second,0))
# Write into a file - sample.csv
def dataLog():
 dt = rtc.datetime()
 T = dht.temp()
 log = open('sample.csv','a')
 log.write('{0}-{1:02d}-{2:02d}
{4:02d}:{5:02d}:{6:02d}'.format(*dt))
 log.write(', {}\n'.format(T))
 log.close()
 print(dt, T)
dataLog()
```

Both the time data from the RTC and temperature data from the DHT-11 sensor are encrypted. A simple way to implement encryption is by using the XOR Cipher. The concept of implementation is to first define an encryption key (say 19) and then to perform XOR operation of this key with the characters in the String, which you want to encrypt.

```
key = 19
def encrypt(string):
 enc_string = ''
 for i in range(len(string)):
  num = ord(string[i])
  encNum = num^key
  encChar = chr(encNum)
  enc_string += encChar
 return enc_string
```

To decrypt the encrypted characters (enc_string), we have to perform XOR operation again with the defined key.

```
def decrypt(enc_string):
 string = ''
 for i in range(len(enc_string)):
  num = ord(enc_string[i])
  decNum = num^key
  decChar = chr(decNum)
  string += decChar
 return string
```

Next a *.csv file needs to be opened in which the new data in csv format has to be encrypted and appended. To keep the size of this file from growing indefinitely, the earliest encrypted data needs to be discarded from the file.

The temperature readings on a particular day from the program is shown in Figure 5. The increase in temperature was initiated by blowing hot air from a hair dryer onto the DHT-11 sensor.

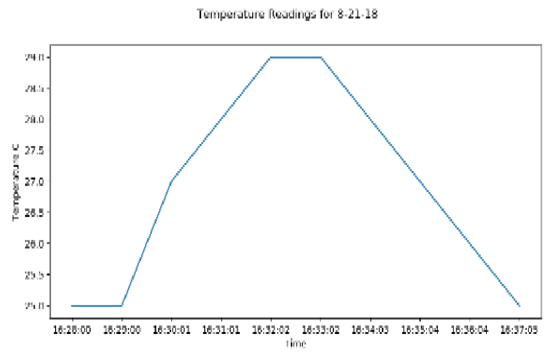

Fig. 5. Temperature readings from DHT-11 securely transmitted and received

For the class assignment, variations in some of the parameters will be specified. One of the most important one will be to use a different encryption/decryption algorithm discussed in the class. Also other sensors will be used to capture pressure or other physical quantities.

III. Additional Discussion Topics and Outcomes to be Covered in EENG3288

The following additional discussion topics need to be added in EENG3288 to introduce the pyboard, MicroPython and the labs described in this paper. Within brackets, we also indicate the KU topics and outcomes as well as NICE Framework categories and specialty areas [8] met by the following discussions and activities.

- Introduction to the pyboard and STM32F405RGT6 microcontroller architecture [*CAE-CD EBS KU Outcome 1, Topic 1*].
- Introduction to MicroPython.
- Mechanism of turning LEDs ON by applying proper voltages.
- Pin programming and Lab 1 [*NICE Framework SA SP.DEV*].
- Python switch, callbacks, and interrupts [*NICE Framework SA SP.DEV*].
- Python classes and objects [*NICE Framework SA SP.DEV*].
- The Timers and timer cycle based programming and Lab 2 [*CAE-CD EBS KU Topic 5, NICE Framework SA SP.DEV*].
- Internet and Security – Data encryption, decryption and Lab 3 [*EBS KU Topic 9, HFS KU Topic 6b, NICE Framework SA OM.DTA*].

In order to satisfy the requirement specified in Outcome 6.1, we will assign multiple projects to multiple teams of three students. The projects will conform to the PBL approach and be assigned at the mid-point of the course to enhance student-learning outcomes. We will divide each project into three sub-projects so that each student in a team can complete a sub-project and integrate the project. Many projects are listed at the end of each chapter of [2]. The instructor will specify the projects based upon these and other resources.

IV. Conclusion

This paper presents a set of new microcontroller laboratory exercises based upon scripting in MicroPython and using a credit card sized pyboard platform. These exercises were developed to fulfill three shortcomings in our current course: (a) enhance student learning by introducing a second hardware platform and its corresponding IDE; (b) enhance student programming experience through scripting in one of the emerging language, namely, Python; and (c) develop "security mindset" among developers of Internet of Things (IoT) apps on microcontrollers. We understand that all these additional activities may become overwhelming for some students. Therefore, we envisage adopting proposed enhancements in steps. For example, we already introduced learning elements of the Python language in the two preceding programming language courses. We will continue introducing other enhancements in due course. We will continue to assess the proposed new outcomes and plan to present those results to our peers. In this paper, we outlined three lab exercises using this pyboard platform and other hands-on lab exercise developments are underway. We have uploaded all the developed labs to the Cybersecurity Library - CLARK platform (www.clark.center) per NSA grant requirements. In future, all details including the program files will be available for download by the educators. We foresee Python based application development to become more prevalent in future and instructors from different colleges are welcome to use our lab exercises readily in their courses.

References

[1] K. Mondal, "Teaching an embedded system course to electrical engineering and technology students", *Proc. ASEE Mid-Atlantic Symposium*, Farmingdale State Univ, NY, 2011.

[2] H. Hwang, The HCS12/9S12 – *An Introduction to Software and Hardware Interfacing*. 2nd Edition. Delmar Cengage Learning, NY, 2010.

[3] R. Haskell and D. Hanna, *Learning by Example Using C Programming the Dragon 12-Plus Using CodeWarrior*. LBE Books, MI, 2008.

[4] F. Salewski, D. Wilking, and S. Kowalewski: "Diverse hardware platforms in embedded systems lab courses: A way to teach the differences", *ACM SIGBED Review* vol. 2, no. 4, pp. 70-74, 2005.

[5] M. Rodriguez-Sanchez, A. Torrado-Carvajal, J. Vaquero, S. Borromeo, and J. Hernandez-Tamames, "An Embedded Systems Course for Engineering Students Using Open-Source Platforms in Wireless Scenarios", *IEEE Transactions on Education*, vol. 59, no. 4, pp. 248–254, 2016.

[6] A. Rao, D. Clarke, M. Bhadiyadra. and S. Phadke,."Development of an Embedded System Course to Teach the Internet-of-Things", *IEEE STEM Education Conference*, ISEC, Princeton, 2018, pp. 154-160.

[7] M. Tollervey, *Programming with MicroPython - Embedded Programming with Microcontrollers and Python*. O'Reilly Media, Inc., CA, 2018.

[8] W. Newhouse, S. Keith, and G. Witte, *NIST Special Publication 800-181 National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework*. US Department of Commerce, 2017.