Case Study with Supporting Media and Simulation Exercise

Title: Cybersecurity Risks in Healthcare

Real World Inspiration

In the United States, an estimated 350,000 people use insulin pumps. These medical devices allow patients with diabetes to manage their blood sugar in a convenient way, with neither frequent hospital visits nor daily insulin injections through needles (syringes or pens). Because devices are essential to the diabetics wearing them, if the pumps malfunction or are successfully attacked by a malicious actor, the wearer may become very sick or even die. The consequences of even a single medical device being hacked are too drastic; therefore, it is of great importance for the manufacturers of these devices to consider Cyber Informed Engineering (CIE) principles at every step of their design process.

If an unauthorized user gains access to a device such as an insulin pump, the hacker has several possible paths of harmful action. The outside source could take more of a read-only stance, where they merely monitor the patient's current data. The much more serious possibility, however, would be if the hacker uses their illicit access to change the commands being sent to the pump. Such commands may involve significantly increasing the amount of insulin released in the next dose. Similarly, a hacker might rapidly release successive insulin doses, likewise driving up the amount of insulin the patient receives.

As you could imagine, the aforementioned attacks have huge consequences for the user of the insulin pump. A patient with too much insulin runs the risk of insulin overdose. The symptoms follow those of hypoglycemia –when there is not enough sugar in the bloodstream–and include milder symptoms such as confusion, irritability, or dizziness as well as more severe cases featuring seizures or a loss of consciousness. The most severe cases risk coma and death [1]. Yet, even if the hacker does modify pump commands, monitoring the insulin pump's data is still harmful to the patient as their personal information has now been stolen; it is no longer confidential between themselves and their doctor or others, and the integrity of data is low. This lack of confidentiality now has serious implications for the patient's medical provider. It brings about the potential for medical malpractice and wrongful death lawsuits, leading to overall decreased trust in the organization and reputational damage.

In the case of an insulin pump, or any implantable medical device (IMD), the device works in conjunction with a 'programmer,' an external device that is responsible "for communicating wirelessly with an IMD and relaying data to a device used by clinicians or other health care providers." [2] Many insulin pumps currently on the market use certain technologies and settings to make insulin delivery easier on the patient. For example, many include glucose-monitoring technology to change amounts of insulin release based on glucose levels. They also include alarms triggered by low battery, low insulin reservoir, or out-of-range glucose levels. Most also connect wirelessly to phones or other technological devices [3]. To familiarize yourself more with the interface of an insulin pump, feel free to explore the interactive Medtronic insulin pump simulator linked below [4].

The aforementioned wireless connection is one of the main causes for concern regarding insulin pump security. For example, the FDA warned of the possibility for "an unauthorized person to gain access to a pump while it was pairing with other system components…" [5] This wireless connection is truly the main vulnerability. In the National Vulnerability Database [6], the majority of insulin pump related vulnerabilities focused on an unprotected wireless communication that allowed unauthorized users access to the insulin pump. There have been a few controlled cases within security conferences where professionals demonstrated the ability to connect to insulin pumps without knowing the devices' ID numbers. In 2012, Barnaby Jack infamously emptied an

insulin pump reservoir into a mannequin from a distance of 300 feet [7]! This was possible because the wireless channels lacked encryption and authentication [7]. The lack of security measures for the wireless communication channels of these insulin pumps allows hackers an easy way into the device. Once in, there is also the vulnerability of the broad setting ranges of the pumps. It creates the potential for hackers to release very large doses without being stopped.

CIE Principles

To combat these vulnerabilities, it is of much importance to start considering CIE principles at every step of the engineering process. The CIE principles encourage mechanical designs that minimize attack surfaces and cybersecurity risks in conjunction with the implementation of software and/or security controls. Rather than viewing cybersecurity purely as a software consideration, it can be much more beneficial for the hardware itself to additionally include design features that mitigate the risk of cyber attacks. In the case of an attack on an insulin pump, there are a few CIE principles that feel extremely relevant. **Take principle #6 'Active Defense'** for example. This principle asks the key question: *How do I proactively prepare to defend my system from any threat?* **Principle #2 'Engineered Controls',** which asks *How do I select and implement controls to reduce avenues for attack or the damage that could result?* **Principle #10 'Planned Resilience'** is another great one to keep in mind. Its key question is *How do I turn "what ifs" into "even ifs"?* These principles prioritize thinking creatively and proactively to include design features as solutions, and they would be great ones for you to think about as you go on to the lab simulation.

Media Feature for the General Audience

For a custom media clip designed by faculty and students of the University of Pittsburgh, please click on the video file found here: **https://youtu.be/oW6uF47_zjc**

The media attached is an introduction into the dangers of cybersecurity in wearable medical devices to the users themselves as well as doctors and medical practitioners. Through the structure of a fictional news feature, the video explores what the vulnerabilities are in wearable medical devices that can be controlled remotely, what the risks are, and what is being put in place to protect them. The video also explores how cyber-informed engineering may lead to a more secure solution to those vulnerabilities rather than typical computer-science-based cybersecurity. It concludes open-ended with how these devices can continue to become safer for their users.

Future Policy Implications

Organizational policy decisions related to wireless insulin pumps rely on the requirements and guidance issued by various agencies. In the United States, regulatory responsibility for the cybersecurity of medical devices resides primarily with the Food and Drug Administration, or FDA. Via the Federal Food, Drug, and Cosmetics Act, this administration is authorized to oversee the production process of medical devices and conduct post-market surveillance [11]. The FDA works closely with the Cybersecurity and Infrastructure Security Agency (CISA), an organization that releases public advisories describing emergent cybersecurity vulnerabilities and exploits [11]. This collaboration has generated a comprehensive web of best practices for the manufacturing of wireless insulin pumps that serves to supplement mandatory regulations. However, the creation and release of medical devices also depends on laws not strictly focused on cybersecurity. Chief among these is the Health Insurance Portability and Accountability Act (HIPAA), which elaborates standards for the protection of patients' health information [11].

To varying degrees, these agencies and laws dictate the organizational policies of two main stakeholders: the companies responsible for manufacturing wireless insulin pumps, and the healthcare providers responsible for prescribing them to patients. The primary role for policy

experts within manufacturing companies, typically represented by a designated compliance department, is to bring products into accordance with FDA regulations and CISA recommendations to legally (and competitively) enter the market. On the pre-market side, this involves the assembly of a software Bill of Materials (SBOM), hardware end-of-life instructions, complete system specifications and diagrams, and risk management documentation [8]. FDA post-market guidance proves similarly robust; manufacturers are encouraged to maintain an up-to-date threat model supported by risk assessment tools such as the Common Vulnerability Scoring System [8]. Other recommended measures, such as participation in information-sharing organizations like H-ISAC, further help to clarify manufacturers' perceptions of the industry-wide vulnerabilities [8].

Comparatively, for compliance departments within healthcare providers, HIPAA takes on somewhat of a greater role than the FDA or CISA. For instance, HIPAA stipulations determine healthcare organizations' practices regarding the disclosure of protected health information to a medical device company representative [12]. However, perhaps a larger influence on healthcare provider policy rests with the procurement departments and committees of medical professionals responsible for choosing which insulin pumps to prescribe in the first place. Because a higher degree of dependence on wireless networks translates into a higher number of potential vulnerabilities to account for, medical professionals who prescribe medical devices overwhelmingly opt for the least connected options available. This policy successfully stifles any possibility of a cyberattack or resultant personal injury lawsuit but discourages the development of more efficient medical devices. Thus, medical device manufacturers are bound not only by the regulations imposed via the FDA, but the need to reassure and generate buy-in from healthcare networks.

Expand Your Understanding with a Laboratory Exercise

(a) Use Case

This use case illustrates the substantial damage a malicious actor can cause to a patient with an unsecured insulin pump and the extent to which hardware restrictions can mitigate this damage. As such, the laboratory seeks to highlight the engineered controls principle.

Caution: the values in this simulation are not reflective of a real diabetic. All values are fudge factors set to emulate semi-realistic experience.

(b) Program Introduction

In this exercise, we use Python. The Python simulation is created according to the object oriented programming paradigm, with separate objects representing an insulin pump system's primary components: the pump, a microprocessor, and a blood glucose sensor. Additional objects were made to represent the patient and an aggregate of all the previously mentioned objects, the full "system."

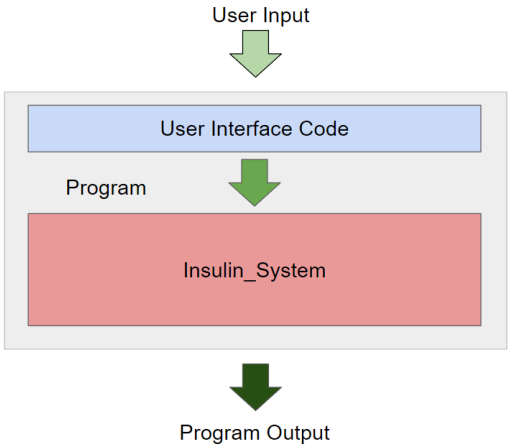

Figure 1: (Above) Overall simulation structure. Green arrows represent data flow, with earlier steps in lighter shades of green.

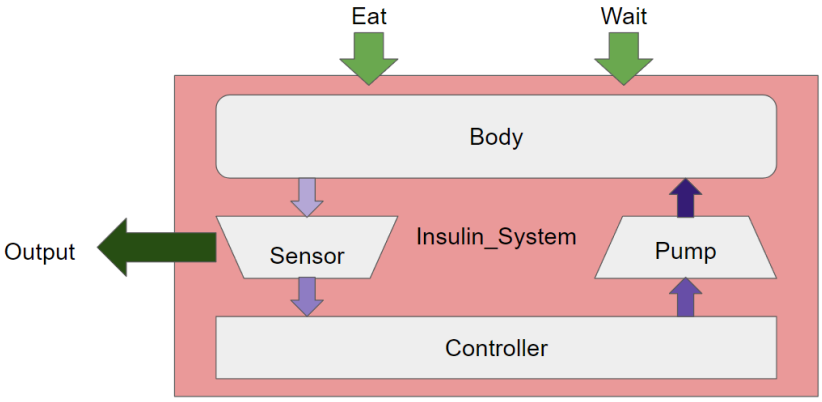

Figure 2: (Above) Arrangement of objects within the Insulin_System object. Green arrows represent data flow into and out of the Insulin_System object, while purple arrows represent data flow within the Insulin_System object.

The Body object attempts to simulate bodily response to carbohydrate and insulin intake through a set of differential equations found in [13]. However, those differential equations were (heavily and) arbitrarily modified to suit this simulation, with more details are available in the code file itself. Similarly, while the Controller object calculates insulin boluses based on a PID algorithm described in [14] and [15] (the blood glucose target of 120 mg/dL from [15] is not used here).

(c) Simulation Set-Up

Please make a copy of the provided Python notebook (Python 3.10 or higher is needed to run the code). Read through the provided objects and run each block after you finish reading it. Please proceed to examine the “Insulin Pump Simulation v2” block. This block presents you as a type-1 diabetes patient equipped with a fully automatic insulin pump system.

(d) Tasks of this Exercise

Task 1: Observe the Simulation’s Automated Response to Carbohydrate Intake

Navigate to the Colab toolbar (or Jupyter notebook toolbar), select the “Runtime” dropdown menu, and click “Run all.” This should immediately navigate you to the relevant simulation.

You should be presented with a list of choices. Except for “Wait,” “Eat,” “Log In,” “Device Information,” and “Exit,” the other commands require a higher-than-default level of system access.

Please use the “Wait” and “Eat” commands to simulate an individual’s food intake for at least a 24 hour period. For example, you could eat 800 calories for breakfast, wait 360 minutes until lunch, eat 400 calories for lunch, wait 360 minutes for dinner, eat a large, 1200 calorie dinner, and sleep (wait) for 720 minutes. Once you have simulated eating for a personally satisfactory duration, enter the “Exit” command to view graphs of blood glucose and insulin levels for up to the past 2000 minutes.

Consider:

- How is a patient’s blood glucose level related to their insulin level?
- There are three levels of low blood glucose, or hypoglycemia, with the least significant level beginning at 70 mg/dL (70 units in the simulation) [9]. A hungry adult is expected to have a blood glucose level between 80 and 130 mg/dL, and an insulin pump should aim to lower blood sugar after a meal to less than 180 mg/dL within 2 hours of the meal [10]. Does the provided insulin pump simulation maintain the patient at a reasonable blood sugar level? Consider modifying the insulin pump’s insulin delivery logic to achieve better patient outcomes. Modifying the insulin pump’s logic requires re-coding the Controller class’s auto_cycle method.

Task 2: Exploiting Improper Authentication

The pump is purposefully designed to mimic Jay Radcliff’s 2011 insulin pump hacking exhibit [7]: when not logged in, the user can still query the pump for identifying information that contributes to a formulaic password. In this case, the pump returns a device identifier which is appended to a root password “SHURE_pump_”.

Re-execute the “Insulin Pump Simulation v2” code block and use the “Device Information” option to find the pump’s identification number. Try to configure one of the options which requires logging in to change before and after logging in.

Consider:

- Which cyber-informed engineering principles best relate to this vulnerability?
- How might this vulnerability be addressed?
- Is it absolutely possible to eliminate improper authentication?

Task 3: Examine the Power of Logging-In

The insulin pump simulation’s password is “SHURE_pump_000000001.” This password enables the user to use all of the simulation’s menu’s options. In task 1, the insulin pump user’s only options were eating and waiting: this limited autonomy is rare and might only occur for young children whose guardians are responsible for managing the insulin pump. Comparatively, task 3 posits you as a patient managing his/her own insulin pump to demonstrate the dangers associated with access to insulin pump controls.

Please rerun the “Insulin Pump Simulation v2” code block and login (on each code block rerun) before proceeding with the following subtasks.

First, you may attempt to induce hyperglycemia in the insulin pump user. This may be accomplished by setting the basal rate to 0.0 and disabling the pump’s AI. Then, proceed to wait and eat as in task 1. For illustrative purposes, it is advised to eat at least 2000 calories to ensure the body’s initial insulin pool fails to prevent hyperglycemia.

Next, you may attempt to induce hypoglycemia. While it remains crucial to log in for this task, it is unnecessary to change the basal rate or disable the pump’s AI. Rather, while continuing to feed the subject as usual, rapidly deliver a succession of large insulin boluses (20, 30 insulin units).

Consider the following:

- If a malicious actor managed to remotely login to the current insulin pump system, they could push the pump user into either hyper- or hypoglycemia. How might this be prevented?
- Which cyber-informed engineering principles (best) address this vulnerability?
- Suppose that, given the physical capability to transmit and receive information, it is impossible to prevent malicious actors from obtaining remote control of the insulin pump’s full functionality. Would it be worthwhile for insulin pumps to retain wireless capability?

Task 4: Code an Engineered Controls Countermeasure.

It’s time to put your considerations into motion! In accordance with the cyber-informed engineering principle of “engineered controls,” please modify the provided “Pump_Pro” class to mitigate the dangers associated with malicious remote control.

The “Pump_Pro” class is located in the code block after the “Insulin Pump Simulation v2” code block. You may assume the “max_bolus,” “max_basal,” and “delay” instance variables never deviate from the values presumed by the class constructor. In a physical system, these variables would be stored in a removable read-only micro-SD (ROM, read-only memory) card (or something similar) which a licensed healthcare provider configures and provides to the end user.

In the corresponding physical system, the ROM card would serve as a boot drive for a supervisory microprocessor built into the pump. The code you write in the Pump_Pro class is also baked into the ROM, but the supervisory microprocessor would also have some read and write enabled memory for any necessary data storage.

Although your contribution to this task is purely software engineering, because it is software for a hardware modification to the insulin pump system, the countermeasure as a whole may be categorized as either an engineered control and/or an active defense.

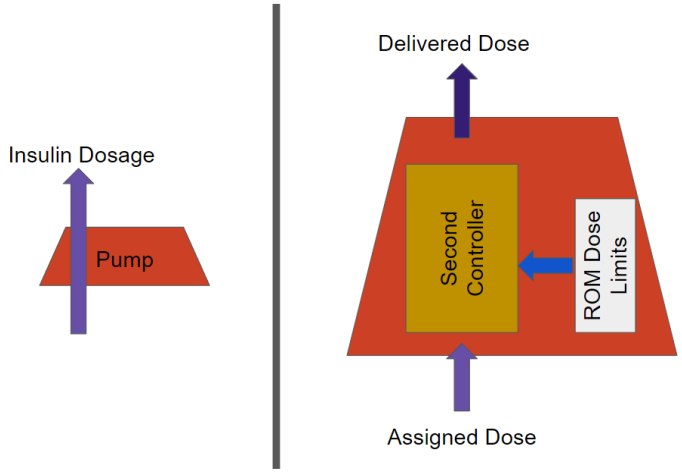

Figure 3: (Left) Regular Pump object. (Right) Pump_Pro object with an engineered controls countermeasure.

As noted in the code document, it is recommended to create three insulin dosage restrictions:

1. Limit the maximum insulin basal rate to 2.0 units per hour
2. Limit the maximum bolus dose to 30.0 units
3. Enforce a three-hour cooldown period between insulin bolus-es

Please test your coding against the hyperglycemia and hypoglycemia routines from task 3. Remember to run the "pump class with restrictions" and "insulin system equipped with Pump_Pro" code blocks before the "Insulin Pump Simulation with CIE Protection" code block containing your modified version of the insulin pump simulation!

Consider:

- Were the engineered controls successful against hyperglycemia (high blood sugar condition)?
- Were the engineered controls successful against hypoglycemia (low blood sugar condition)?
- How could the countermeasures be improved?
- This task specifically addresses the "engineered controls" principle. What would be a countermeasure that addresses the "active defense" principle?

Further Reading and Useful Public Video Links

For further reading, the following references would be suitable to explore at your convenience.

[1] Morales-Brown, Peter. "Insulin Overdose: Dosage, Symptoms, and Treatment." *Medical News Today*, MediLexicon International, 2 Aug. 2023, www.medicalnewstoday.com/articles/317300.

[2] M. Rushanan, A. Rubin, D. Kune, and C. Swanson, "Sok: Security and Privacy in Implantable Medical Devices and Body Area Networks," *IEEE Symposium on Security and Privacy*, May 2014. ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6956585&tag=1.

[3] Cleveland Clinic medical. “Types of Insulin Pumps.” *Cleveland Clinic*, 11 Dec. 2023, my.clevelandclinic.org/health/articles/insulin-pumps.

[4] *Medtronic*, us.medtronicvirtualpump.com/gTl7X5011Iy6IjOlU/VirtualDemoPump/MiniMed_780G_EC/5C/MMT-1884/.

[5] *FDA Warns of Cybersecurity Risk with Certain Medtronic Insulin Pumps* | *Reuters*, 21 Sept. 2022, www.reuters.com/business/healthcare-pharmaceuticals/fda-warns-cybersecurity-risk-with-certain-medtronic-insulin-pumps-2022-09-20/.

[6] Booth, H. , Rike, D. and Witte, G. (2013), The National Vulnerability Database (NVD): Overview, ITL Bulletin, National Institute of Standards and Technology, Gaithersburg, MD, [online], https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915172

[7] Klonoff, David C. “Cybersecurity for connected diabetes devices.” *Journal of Diabetes Science and Technology*, vol. 9, no. 5, 16 Apr. 2015, pp. 1143–1147, https://doi.org/10.1177/1932296815583334.

[8] Synopsis. “Securing Connected Medical Devices for FDA Submissions.” *Synopsis*, 2021, www.synopsys.com/content/dam/synopsys/sig-assets/whitepapers/wp-securing-connected-medical-devices-fda.pdf

[9] Skoler, Eliza, and Matthew Garza. “Low Blood Sugar – Hypoglycemia 101.” *DiaTribe*, diaTribe, 2 Feb. 2022, diatribe.org/diabetes-management/low-blood-sugar-hypoglycemia-101.

[10] Hoskins, Mike. “Helping You Understand ‘normal’ Blood Sugar Levels.” *Healthline*, Healthline Media, 15 Sept. 2022, www.healthline.com/health/diabetes/normal-blood-sugar-level#target-glucose-goals.

[11] United States Government Accountability Office. “Medical Device Cybersecurity: Agencies Need to Update Agreement to Ensure Effective Coordination.” *GAO*, Dec. 2023, www.gao.gov/assets/d24106683.pdf.

[12] Office for Civil Rights. “FAQ 490.” *U.S. Department of Health and Human Services*, 28 Dec. 2022, www.hhs.gov/hipaa/for-professionals/faq/490/when-may-a-covered-health-care-provider-disclose-protected-health-information-without-authorization/index.html.

[13] Al Ali, Hannah, et al. “Examining Type 1 Diabetes Mathematical Models Using Experimental Data.” *International Journal of Environmental Research and Public Health*, vol. 19 no. 2, 2022, https://doi.org/10.3390/ijerph19020737.

[14] Cinar, Ali. “Automated Insulin Delivery Algorithms.” *Diabetes Spectrum*, vol. 32 no. 3, 2019, https://doi.org/10.2337/ds18-0100.

[15] Thomas, Andreas, et al. “Algorithms for Automated Insulin Delivery: An Overview.” *Journal of Diabetes Science and Technology*, vol. 16 no. 5, 2021, https://doi.org/10.1177/19322968211008442.

Authors

Karlynn Riccitelli, University of Pittsburgh Class of 2026. Interests include the intersection of English and Computer Science, in combination with Digital Media and User Experience Design.

Naomi Taylor, University of Pittsburgh Class of 2025. Interests include film direction, cinematography, and photography.

Casey Withers, University of Pittsburgh Class of 2025. Interests include applied statistics, comparative politics, and data science.

Lambert Zhang is an engineering science major at the University of Pittsburgh with a concentration in engineering physics. He is expected to graduate in Spring 2025 and appreciates the feeling of accomplishment associated with assembling devices.

Appendix

Run All Command:

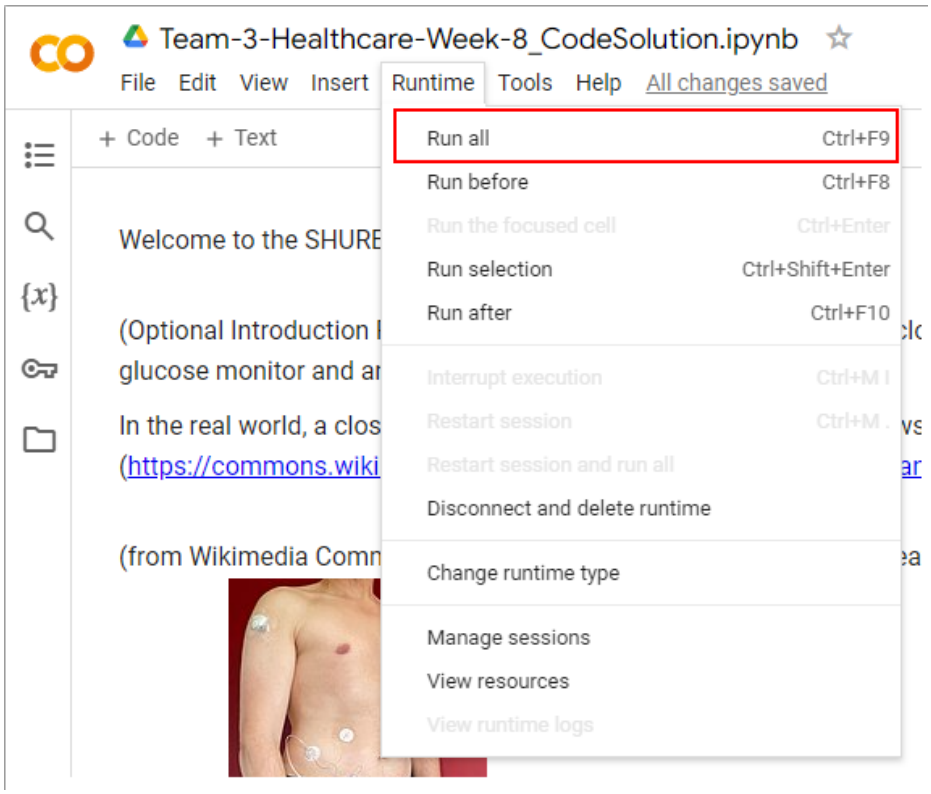

Figure 10: The "Run all" command, circled with a red box.

Menu Options:

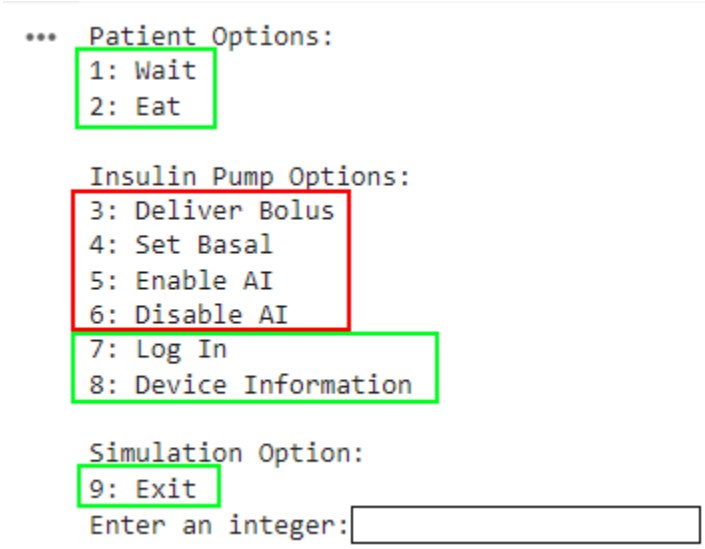

Figure 11: A screenshot of the simulation's user options. Options which can be used without logging in are denoted in green, while options which require logging in are surrounded with red.

Insulin Delivery Algorithm:

```
def auto_cycle(self, subject, sensor, pump): #insulin delivery mathematics
  sensor.measure(subject) #measure values

  #math for PIE algorithm
  current_sugar = sensor.send()[1] #get current blood sugar value
  #calculate current error
  current_error = current_sugar - 130.0 #120 mg/dL is a target value taken from the second paper mentioned in the above text box
  #update integral of errors
  self.error_integral += current_error
  #find rate of change
  rate_of_change = current_error - self.previous_error
  #calculate dosage via PIE algorithm with 120 mg/dL target blood glucose; coefficients are arbitrary
  dosage = 0.00075 * current_error + 0.000005 * self.error_integral + 0.05 * rate_of_change
  #update previous error variable
  self.previous_error = current_error

  if not(self.enable): #manual insulin delivery; just deliver basal rate (boluses delivered via other command)
    dosage = self.basal_rate / 60.0 #need to divide by 60 b/c dosage is total bolus for 1 minute but basal_rate is in insulin units per hour

  #print(f"{0.00075 * current_error}, {0.000005 * self.error_integral}, {0.05 * rate_of_change}, {dosage}") #troubleshooting

  self.cycle(subject, sensor, pump, dosage) #tell pump to deliver insulin
```

Figure 12: A screenshot of the method within the simulation's controller class that automatically calculates an appropriate insulin dose.

Region to Add Restrictive Code:

```
#pump class with restrictions
class Pump_Pro(Pump):
  def __init__(self, arg_str = "Infusion System", max_bolus = 30.0, max_basal = 0.0333, delay = 360):
    #default value of max_basal is 0.0333 or approximately 2 units/hour divided by 60 minutes per hour

    super().__init__(arg_str) #calling the parent class contructor. In c++, this would be accomplished
    #in this case, the above is needlessly more complicated than "self.label = arg_str" . . . but Googl

    self.maximum_bolus = max_bolus #maximum allowable single insulin dose
    self.maximum_basal = max_basal #dosages above this level are considered bolus-es
    self.delay = delay #time after a bolus before another bolus is allowed

    self.allow_bolus = True #can a bolus be delivered right now?
    self.previous_bolus = 0.0 #how long has it been since the last bolus delivery?

  def deliver_insulin(self, subject, amount): #obfuscate the parent class's version of this method
    #add restrictive logic here

    #end restrictive logic here
    subject.timestep(amount)

  #mutators
  def set_max_bolus(self, max_bolus):
    self.maximum_bolus = max_bolus

  def set_max_basal(self, max_basal):
    self.maximum_basal = max_basal

  def set_delay(self, delay):
    self.delay = delay

  #accessors
```

Figure 13: A screenshot with the place to write restrictive logic boxed in red ink.