# CS 405 Project Two Script

Complete this template by replacing the bracketed text with the relevant information.

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Malcolm McGee, and this is a presentation of the Green Pace security Policy |
| **2** | We will be beginning with defense in depth.  Our Security policy was implemented to ensure that every layer within our organization is secure. Through leveraging Defense in Depth (DiD) best practices we established a robust multi-layered security framework. This framework includes preventative measures such as access control, encryption, and authentication. With our approach we can limit the damage that could ensue from a security vulnerability being discovered and exploited through this Defense in Depth implementation. |
| **3** | Our threats matrix is broken down into four sections. We have Likely, Priority, Low Priority, and Unlikely. Two of our priorities will be SQL injection which is STD-004-CPP and Memory Management which is STD-008-CPP. We can also implement automation to help detect some of these threats. For SQL injection we can create unit tests to see if the input is sanitized properly to prevent sql injection. |
| **4** | Next, I want to move on to our 10 principles. I will cover a couple of them here. For heed compiler warnings when we are developing code, we want to ensure that we listen to any warnings our compiler throws and work towards implementing them.  Whenever we are giving a user privilege to make a change we should adhere to the principle of least privilege. Meaning if they are needing the ability to edit something on their account, we don’t need to give them ability to edit any account, but instead just limit it to their details. |
| **5** | Listed here are our coding standards. We have them in order of prioritization. The order was created by which ones could cause the most damage to user data that is stored if not followed. Because of this we have SQL injection at the top. Not following this standard will allow for users to inject SQL commands and access data in our database. Next, we have memory management which if isn’t followed could allow for someone to crash the program and get data from that. For memory protection we have the ability of leaking data. As we go down the impact of these coding standards are less dramatic, but we should still follow them to create a program that is safe and usable for our end users. |
| **6** | **For our encryption policies we have Encryption in rest. Which is while data is being stored the encryption at rest policy would want to ensure that it is encrypted. This can be applied by encrypting data in our databases or on the user device with whatever encryption algorithm is best for that occasion from things like Advanced Encryption Standard (AES), Secure Hash algorithm (SHA) and others. The reason we should implement encryption at rest is because it helps protect the data in the case that a bad actor can get it. If someone can hack our database or mine the data from the user’s device having a second layer of protection from encryption would help protect that data and not allow for someone to use it. Encryption at flight will be when we are transmitting data from point A to point B we can send the data in a plaintext format, but that isn’t secure. Instead, the data should be encrypted while it is being sent. In a lot of applications, we will be sending data over the network or to servers and to protect it the data should be encrypted. This will help safeguard the data from unauthorized individuals intercepting it and using it. Applying this practice allows us to mitigate the damage from things like man-in-the-middle attacks that are trying to intercept data we are sending. We also have encryption while at use which is Through utilizing device features like Secure enclaves, we can perform operations on data while it is encrypted. Encryption, while it is in use, would be to have the data secure while we are using it through different approaches like the use of secure enclaves. The reason we should use this is because some attack methods like buffer overflow or exploits allowing us to read some data from RAM if that data isn’t encrypted then bad actors will be able to use it and gather more.** |
| **7** | For our triple- a policies we have authentication, authorization, and accounting.  Authentication is a more common policy, and it is ensuring that the person that is intending to log into your system for example has the proper login credentials to access the system. Using authentication, we can ensure that the user of the system is supposed to have access. Authentication could go beyond just login credentials like using 2FA for the user, biometrics, NFC cards, and much more. This helps ensure that only allowed users can access the system protecting it from unauthorized access.  Authorization is the process of limiting what actions certain users can perform. The way this can be implemented is through adding distinct roles and allowing those roles to have specific functions. Then we can assign those roles to new users. This can also tie into the principle of least privilege giving the user only the privileges that are necessary for the role they are doing. This helps protect us from disgruntled ex-employees who may want to attack the system prior to leaving, or in the case of a standard end-user account being hacked them not having unnecessary privileges to the database that is now given to some bad actor.  Accounting is keeping a log of user actions and interactions in a system. This can be used by making a record of every change that a user in the system does with rolling backups of a set time frame. The reason why it is so that in the case that a user makes a mistake, or someone makes an unauthorized change admin will be able to see when it occurred and roll back the changes to a previous state. It also allows for the discovery of a data breach in the case that the admin sees a user download a lot of data and the user didn’t authorize it. It could lead them to discover that the user's account was compromised and spot the data breach to help remedy the issue and notify end users. |
| **8** | For the unit testing this one is testing for out of range. In this example we have a collection that is empty and are attempting to access what value that may be stored at item 1. Since there is no item 1 the test throws an out-of-range message, but it if wasn’t implemented properly we could’ve accessed data stored in memory. |
| **9** | In this test I want to ensure that we can add an entry to an empty collection. We want to make sure that the collection updates its size to 1 representing that an item was added to the collection. |
| **10** | In this unit test we want to make sure that if we resize a collection the size is updated properly. We are starting with an empty collection and increasing the size to 10. |
| **11** | With this unit test we are verifying that the size check does work. We are doing this by adding 5 entries to an empty collection. Afterwards we are testing to ensure that the collection is of a size of 5 which shows that the size check is working. |
| **12** | When working on our policy automation is also going to play a large role. To streamline the enforcement of our standards, the establishment of a centralized policy repository is important. This repository is an integral component of our assessment and planning procedures, serving as the designated storage for our organizational policies. This will include adherence to regulations, release requirements, and organizational guidelines. This repository will also play a large part in allowing us to set up and implement policy automation and enforcement. Automated tools will be employed to assess risks, and issue alerts, and notifications. The automation of compliance not only enhances operational efficiency but also mitigates the need for repetitive tasks. We will also be able to automate the transition and health check phases. Using automated penetration testing aids in minimizing false alarms and streamlining the resolution of identified issues. We can extend this automation in many ways like through establishing a system for logging and storing data in a dedicated database. This database serves as a critical tool for identifying vulnerabilities and preventing potential attacks. Various security measures, such as signature checks, ensuring data integrity, and employing multiple security layers, will be implemented. We can also do things like automate the creation of save points. These backups will serve as secure backups allowing us to swiftly revert to a previous stable version in the event of an attack or system malfunction. |
| **13** | The DevSecOps pipeline is a strategy used by Green Pace in the implementation of DiD. Our strategy will consist of planning in the pre-production phase, designing, building secure code, verifying, and testing the system. While we are in the production phase, we maintain the program and respond to any threats.  For this DevSecOps pipeline we are also using automation and some of this comes in when we are doing Source code management and using static application security testing.  We also use continuous integration where code is built, and automated tests are then run on that code to ensure the quality and security of it.  We will also be utilizing Continuous Deployment where code is deployed to various environments, and we are able to use Dynamic application security testing. |
| **14** | For risks and benefits we have a benefit in not deploying the strategy in that it will save us time and money in the short term. We are also able to release and iterate on products at a faster speed when we aren’t focusing on designing security in with the program.  The other risk is that with the use of automation the automation will only be as good as what we develop. We need to ensure that we are creating thorough automation tests that are detecting a wide variety of errors and not just limited to how we think a user may be interacting with our program.  For benefits we will be saving money in the long term. If we deployed unsecure code that code would be exploited sooner or later, and we would be stuck trying to implement security afterwards. We will never be able to get this same layer of security with designing our products with security in mind if we just bolted on security in the end due to a vulnerability being discovered.  We also have a risk to our reputation by deploying code that wasn’t tested. This damage could limit the companies that may wish to work with us. |
| **15** | While we did try to create a security policy that had a lot of depth, the code and tests are limited by those who are creating it. We should also investigate hiring third party security firms to perform security audits on our system regularly. This will help us find areas where the automation may be failing and improve it for the future.  We also hope that a breach never occurs, but we should also plan how we will respond if or when it does occur. We need to have plans in place for communication to those affected, how we will work to resolve the issue, and any other steps we will want to take.  This policy is also only as good as our standards in development. We can’t afford to become lax in our coding practices and not working to create defense in depth. This also ties back into having a third-party audit us routinely. This will help us see if we are not implementing our standards normally and work to provide feedback as well as work to improve.  Lastly, we need to take in all the feedback from our tests and implement solutions for them. In the future we need to run more tests as we develop our systems and ensure that any feedback we get from the tests and 3rd party security auditors are properly implemented into our system. |
| **16** | Whenever we are designing any type of software, we should be approaching it with a defense in depth mentality. We should create multiple layers of security and test those layers.  We need to ensure that we always have security be a part of our design process and never as an after thought for the product or service we are working on.  We should also work with third parry to help validate that we aren’t missing things with our tests and catching security vulnerabilities prior to that code being deployed.  We should also have separate teams or developers check the unit tests and other tests that someone implemented. This allows for us to get more input and help us find other security tests we should implement that may have been missed. Thanks for your time! |
| **17** |  |