# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, thank you for joining me today for our Security Policy presentation, I will be introducing a new proposed plan for Green Pace in efforts to comply with best practice security standards. |
| **2** | As you can see by the following diagram there is A LOT of thought that goes into proper security management, we won’t be covering all of these today, but we will be going over adopting a few basic measures to ensure we are doing our due diligence to have best practice secure coding. |
| **3** | I have outlined in the security policy 10 coding principles we must ensure we are implanting that can lead to dire consequences should we face an attack. Please see the security policy for these practices, this includes ways to prevent SQL injections, ensuring proper data types and values are used, and employing proper string correctness. |
| **4** | The 10 principles of secure coding that we must practice are validating input, with some examples from the policy, heeding compiler warnings, ensuring our architecture and design include not allowing access to freed memory, for example, keeping it simple, and default deny. We should not be giving access where it is not needed! |
| **4.5** | Along with this there is the principle of least privilege, ensuring we only give access where it is needed to perform its daily operations and that’s it. Sanitize Data, as we know this is commonly how SQL Injections happen if we are not careful. Practicing Defense in depth, which for those of you that do not know, is using multiple layers of security should one fail. It is a safety net to ensure more damage to the system is not done. Using effective quality assurance techniques is rather simple, we must employ unit testing, exceptions, and static analyses when coding our projects. These are all ensuring you are using secure coding practices. |
| **5** | Here you will see the same coding standards we have mentioned before, but with some more detail. These are all just some examples of what you can do to practice due diligence in your best coding standard practices. Things such as returning values when exiting a value-returning function, not allowing tainted variables that can be used in privileged operations, or accessing freed memory which can cause crashes. |
| **5.5** | Here we have more of the previous examples, using static assertions, not throwing exceptions across boundaries, and not calling the system function. |
| **6** | Onto encryption, data should always be encrypted when in flight, rest, and use. We will get to each of these individually. Data in Flight. This is data that is being transmitted between systems. We must ensure that this data is encrypted so bad actors can’t intercept it and see sensitive information as it is being sent. Data at rest is stored while not being used, this can include passwords and usernames or other private information. Data is use is exactly how it sounds, this is when it is being processed, with proper encryption we won’t need to decrypt it while it’s being used. |
| **7** | The other important set of three we must talk about is our Triple-A policy when it comes to users. Authentication, authorization, and accounting. We must authenticate a user, this is verifying they are who they say they are using proper usernames and passwords, or other various login methods. Authorization is the principle of least privilege and default deny in action. After authenticating the user we only allow them to do what they need to perform the necessary tasks using roles. Accounting is all about logging and monitoring activity so that we can see suspicious behaviors and attempts in the system, while also giving staff the ability to audit our system. |
| **8** | Unit tests. Here you can see a screenshot of various kinds of unit tests we practiced with our developers so that they can understand how to properly test their code. |
| **9** | While in the previous slide you saw results of those tests, here are some examples of unit testing we might employ in our system. |
| **10** | All of this is necessary for DevSecOps to incorporate security to ensure we are complying with federal and government regulations as well as keeping the trust of our shareholders. This graphic illustrates the endless cycle our developers must run through to keep up with the times. |
| **11** | New vulnerabilities are always being discovered and it is our duty to try and stay on top of these issues. It all starts with planning, leads to designing and building our security, and as we deploy our product we have to test, monitor, and maintain our code and systems. We can do this through multiple tools that help us with static code analysis through use of software like SonarQube, scanning our dependencies through use of tools like OWASP, and active monitoring through security information and event management, or SEIM, with systems like Splunk that analyzes data in real-time. |
| **12** | Should we enact this policy now or later? There are inherent risks and benefits to both. Should we do it now we can start mitigating any damage and vulnerabilities now. However this will take some time and quite a few resources so that we can prevent any future attacks. Waiting until later could open us up for vulnerabilities now and increases our chances of an attack happening, but we’ll be able to fine tune our methods and plan better. We also need to assess the current state of our security practices and identify gaps that the new policy will address. We also need to assess the risk of the impact on ongoing projects as well as future products we will be starting soon. There is a way we can start now, by slowly phasing it in so we can minimize the disruption and implement security now that will impact us severely should we not focus on these prospects. Areas that we are most vulnerable, and in the meantime we can start training and bringing awareness to our team that a new policy will be coming so they aren’t surprised by the sudden change.  We need to talk about taking action in the now. This will enhance our security, ensure we are compliant with legal regulations, and we will improve our developers’ coding practices. However, there will be a cost and possible resistance to this new change. We’ll need the proper training, tools, and develop due process. We are hoping that all of our teams will embrace the new policy, but if they don’t there may need to be some management to help with the change. As we’ve discussed, delaying action will increase our vulnerability to attacks and we could possibly be fined for being out of compliance with regulations. We know this policy isn’t perfect and there may be gaps, but we can take in feedback as we introduce the new policy to hear from fellow developers and their opinions on what can be improved. This security policy doesn’t cover what training will look like or how we will allocate resources while continuing to work on current projects, but we have hope that it will come together nicely. |
| **13** | Something we must address in an updated policy is the incomplete coverage of this one. This includes how we will deal with incidents as they occur and the physical security measures we must take. What does that mean? In the original graphic it displayed examples of some physical security measures, items such as Access Control panels, CCTVs, this policy itself, and alarms should there be an intrusion. |
| **14** | So to recap, we have to employ best secure coding practices and security in general right from the beginning. How do we do this? With Defense in Depth, where we can utilize multiple layers of security. What does that entail? Implementing secure practices, proper encryption, a triple A architecture, and DevSecOps automation. Thank you for listening, have a great day. |