# 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 Victoria Kaloudis and today I will be going over Green Pace’s security policy. |
| **2** | Our security policy approach leans towards a zero-trust policy. By utilizing the defense in depth model, we can ensure private data remains private. This illustration provides a visual representation of the defense-in-depth best practice of layered security. |
| **3** | Our secure coding standards are used as our guideline for all code written. With secure coding comes certain threats to the standards and system in general. With each coding standard, there are vulnerabilities that will likely happen and some that are unlikely to happen impacting each standard at either low or high priority. This chart displays our approach to prioritizing those vulnerabilities and impacts. |
| **4** | There are 10 core principles to follow while coding securely. Validating input data means to validate data from all data sources, especially untrusted ones. Using correct input validation can assist in eliminating most software vulnerabilities. External data sources are typically seen as untrustful. Some examples are command line arguments, network interfaces, environmental variables, and user-controlled files.  Heed compiler warnings means to Use the highest warning level available for your compiler when creating code. Utilizing static testing and dynamic testing will ensure the code has no vulnerabilities or security flaws.  Architect and Design for security policies means to Create and design software based on your security policies. Be mindful of the software architecture and design you have created in relation to your security policies.  Keep it simple means to Keep the code and design as simple as possible. This will help you in decreasing the number of errors, vulnerabilities, or security flaws found when configuring the design. More complex designs are more difficult to achieve high levels of quality and assurance.  Default deny means to Create a protection scheme based on access permissions. Default the access based on whether the user has the correct permission to access.  Adhere to the principle of least privilege means to Be mindful of the process execution using elevated permission. Try to utilize the least set of privileges necessary to complete any job. This will help in reducing the opportunities that attackers have to access elevated privileges.  Sanitize data sent to other systems means that All data passed to complex subsystems should be sanitized. Attackers might access them through SQL, injection, and command attacks. Some examples of complex subsystems are command shells, relational databases, and commercial off-the-shelf components.  Practice defense in depth means to manage risk in your code by creating multiple layers of defense strategies in case one-layer falls, there will be other layers to prevent a security flaw. This will create a treacherous path for attackers when they attack your system.  Use effective quality assurance techniques means to use Effective quality assurance techniques to support you in identifying and eliminating vulnerabilities in the code. Things such as dynamic analysis, static analysis, fuzz testing, penetration testing, and source code audits can help with determining vulnerabilities in your code.  Finally, Adopt a secure coding standard means to Either create a secure coding standard or use one already made for developing your code in the chosen language and platform. |
| **5** | Along with the core principles, are our 10 coding standards we work with based on risk assessment. The 10 coding standards are string correctness, SQL injection, memory protection, data types, containers, declarations and initializations, data value, exceptions, object-oriented programming, and assertions. Next to the standard are the specific names of each standard we are working with. The standards are organized based on severity levels. The highest severity with priority level P18 are string correctness, SQL injection, and memory protection. With string correctness, it is important to guarantee that storage for strings has sufficient space for character data and the null terminator. This threat is likely to occur, it is at a level 1 risk, and the remediation cost is around medium level. With SQL injection, it is important that you do not store an already-owned pointer value in an unrelated smart pointer. The threat is likely to occur, it is at a level 1, and the remediation cost is around medium level. With memory protection, it is important that you do not access freed memory. This threat is likely to occur, it is at a level 1 risk, and the remediation cost is around medium level. Next, Data type has a priority level of P12. With data type, it is important that you do not define a C-style variadic function. The threat is unlikely to occur, it is at a level 1 risk, and the remediation cost is around medium level. Next, containers has a priority level of P9. With containers, it is important to guarantee that library functions do not overflow. The threat is likely to occur, it is at a level 2 risk, and the remediation cost is high level. Next, declarations and initializations has a priority level of P6. With declarations and initializations, it is important that you do not modify the standard namespaces. The threat is unlikely to occur, it is at a level 2 risk, and the remediations cost is medium level. Next, data value has a medium severity with a priority level of P4. With data value, it is important to make sure you do not cast to an out-of-range enumeration value. The threat is unlikely to occur, it is at level 3 risk, and the remediation cost is around medium level. Next, Exceptions, object-oriented programming, and assertions have a low severity level. Exceptions has a priority level of P9. With exceptions, it is important to remember to handle all exceptions thrown before main() begins executing. The threat is likely to occur, it is at level 2, and the remediation cost is around low level. Object oriented programming has a priority level of P2. With object-oriented programming, it is important to remember you should not invoke virtual functions from constructors or destructors. The threat is unlikely to occur, it is at level 3, and the remediation cost is around medium level. And finally, assertions has a priority level of P1. With assertions, it is important to use a static assertion to test the value of a constant expression. The threat is unlikely to occur, it is at level 3, and the remediation cost is around high level. Based on these coding standards, we could adopt a secure coding standard. |
| **6** | The three encryption polices we work with at Green Pace are Encryption in flight, encryption at rest, and encryption in use. Encryption in rest refers to encryption of data that is currently being stored (Dinic et al. 2022). This should be used as many people are not using software 24/7. For example, username and password combinations are not used all the time. They are stored in a database in the system. The encrypted data is in rest and called upon when the user wants to log into their account. Encryption at flight refers to encryption of data that is currently in movement from one source to the next (Dinic et al. 2022). This should be used to create a secure network to move information from one computer to another computer, or even from network to network. For example, sending an instant message from one coworker to another would use encryption in flight. Encryption in use refers to the encryption of data that is currently in use (Dinic et al. 2022). This should be used anytime the software is accessed, sent, processed, etc. It is the most vulnerable time for data as it is immediately available. Creating secure code during this stage is critical for protection against threats. |
| **7** | Here at green pace, we also use a triple A policy, authentication, authorization and accounting. Authentication is used to determine who the user is (Fortinet et al. n.d.). When a user registers, or becomes a new user, the username and password are encrypted and saved in a system. The authentication step tests the username and password you type in against the encrypted data to ensure you are who you say you are. Multifactor authentication is a common extra security feature that many companies are using to prevent authentication breaches. Authorization refers to the level of authorization that a user has (Fortinet et al. n.d.). Certain users might have different levels of authorization like those in the IT department would have more authorization than a sales rep might have. Having various levels of authorization help make a system more secure as exploiters need to hack more people to determine who has the appropriate authorization. Accounting refers to the logs that are kept in the system to determine where and what each user accessed during their connection (Fortinet et al. n.d.). This is especially good for IP address checking. If the system notices an unusual IP address being used, they can let IT know there might be a breach. Also, if a user changes anything in a database, there can be a notification system set up to let IT know. If a hacker tries to decrypt data, IT can get a notification. Even if a file is accessed by an unknown user, having an accounting system will allow for a more secure infrastructure. |
| **8** | Unit testing is a great dynamic testing tool to utilize when securely coding. Dynamic testing executes the code and looks for potential errors, bugs, etc. At Green Pace, we conduct many unit tests to ensure our code is up to standard. Unit testing is a software test used to test parts of your code (normally units) and ensure that each unit is operating as expected. There are two types of tests, Positive and negative. Positive tests prove that functionality works when tested. Negative tests proves that an error or exception happens when providing bad data. Throughout this project, we conducted 16 unit tests. I will be going more in depth with 2 positive and 2 negative tests in the next few slides. |
| **9** | The first unit test was testing that a collection is empty when created. This is a positive test making sure that the collection is in fact empty and returning a size of 0. As you can see, the unit test results show that this unit test passed. |
| **10** | The second unit test is testing to verify adding a single value to an empty collection. This is a positive test as it is adding one entry to the collection and testing the return value. As you can see, the unit test results show that this unit test passed. |
| **11** | The third unit test is testing to verify the out of range exception is thrown when calling at() with an index out of bounds. This is a negative test as we tested whether an error or exception was found. As you can see, the unit test results show that this unit test passed. |
| **12** | The fourth unit test was testing to verify the removal of an element in the collection. This is a negative test as the collection size will not equal 20. As you can see, the unit test results show that this unit test passed. |
| **13** | Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Both pre-production and production are extremely important for the DevSecOps process. The pre-production phase should focus on creating a plan to assess priorities and possible regulatory changes, designing an architecture that has secure best practices built in, and building secure infrastructure following those guidelines. The production phase should focus on maintaining the system that was built by performing health checks and penetration testing, monitor and detecting potential vulnerabilities, building a secure response catalogue to vulnerabilities, and to maintain and stabilize those fixes so the system can run without problems. |
| **14** | Automation plays a key role in the DevSecOps lifecycle as it can fast-track many steps of the cycle. For example, static and dynamic testing are great automation tools to utilize during the build and verify and test stages. Static testing tests the code for vulnerabilities without executing the code. The test results will give you a list of vulnerabilities and connect you to the National Institute of Standards and Technology (NIST) database for more details. One tool we use often for statis testing is Cppcheck. This is a common tool used for static testing with C and C++ coding languages. Its main purpose is to detect bugs, undefined behavior, and dangerous code. Dynamic testing tests the code by executing it. It will give you a list of potential vulnerabilities within the system and some tips on how to address them. They should be in use at every step of the process. An example of dynamic testing includes unit testing. Unit testing is a great automated way to test the security level of your program and determine how much of your program is covered. |
| **15** | With all decisions comes a list of risks and benefits. Specifically, you should adopt our security policy now rather than waiting to adopt it later. If you choose to wait, you face some potentially major problems. The problems could be higher risk and more severe problems. In the back of your mind, you are always thinking about the what ifs? Those come with adapting or remodeling your programs. The risks you run by waiting include, but are not limited to, data breaches, remediation costs, and loss of trust with users. Data breaches can cause all sorts of problems for you, especially if the information is extremely private. For example, if someone has their social security number saved in your database and you have a data breach, their social security number could be leaked to the dark web, causing identity theft. Also, with data breaches comes remediation costs. If a class action lawsuit is filed by your users and a judge grants them money, your company must pay the money to your users. And finally, you run the risk of losing trust with your users. They are trusting you to keep their information private. If you lose their trust, you can lose a lot more than just money. However, there is one key solution for waiting and that is to act now! Acting now can only benefit you in the future. It will help ensure your data is secure so there are no data breaches, it will help eliminate vulnerabilities in your program that are risky for data breaches, and you can only gain trust with your users. It’s a win win situation for all parties. |
| **16** | Based on our analysis, it is recommended that you take action now. It is important to remain up to date on new vulnerabilities. Accessing the NIST database and reviewing new bugs and vulnerabilities will support your desire for more secure coding. Also, understanding what you are up against in terms of hackers can only benefit your team. There are three types of hackers, white hat, grey hat, and black hat. White hat hackers hack into systems to show room for improvement. For example, a hacker hacks into a bank’s system and steals some money. They then provide a report to the bank with the issues they easily found and provides some ways to remedy the situation. Grey hat hackers have multiple “hats”. Some look for vulnerabilities in a system and request payment to fix them for a company. While some are more ethically driven and will expose an injustice. An example could be Adrian Lamo who hacked into the New York Times and reported on Chelsea Manning's theft of classified documents to the FBI. While what Adrian did was good for the country, he did not have permission to hack into the new York times. Black hat hackers have more malicious intent with their hacking. They are the types of hackers that will hold a hospital hostage until a payment is made. Understanding what is out there can help you build your security system further. Finally, it is our recommendation to adopt a zero trust approach when coding. Expecting the unexpected is impossible, but setting your self up for success is not. When you assume nothing is safe (in terms of coding) you are building your system to handle all types of threats. Having multiple layers of defense is key in case one goes down, there is another right there. |
| **17** | In conclusion, All coding standards are important for the entire cycle. Validating input data helps with eliminating as many vulnerabilities as possible. Heeding compiler warnings will help with testing and executing the code after every line is programmed. Building a system architecture and designing for security polices will ensure your system is safeguarding private information with no vulnerabilities. Keeping the code simple ensures it’s easy to read and understand so other developers can make changes if need be. Defaulting denial and adhering to the principle of the least privilege are both key in maintaining secure connections after the build phase by restricting access to important features based on who needs the access and who does not. Sanitizing data sent to other systems ensures that data is safely sent to the next step with no vulnerabilities to combat exploits of that data. To help do this, you should practice defense in depth by building multiple layers of defense. It may seem redundant, but if one layer goes down, there will be more layers of defense to take its place. Using effective quality assurance techniques is extremely important. Static and dynamic testing was mentioned quite often in this presentation as they are great resources for ensuring the program is secure enough to handle exploits of sensitive information. Adopting a zero trust approach to coding can only benefit your team. Sensitive information would be locked behind many doors as opposed to just one door. Not only are you building for future data vulnerabilities, you are also saving the company time and money. Finally, none of this could be possible without the adoption of a secure coding standard. Having a guideline for the DevSecOps lifecycle will help with designing your code using secure best coding practices. It will help with preventing problems down the line. |
| **18** | This concludes my presentation, and I am happy to answer any questions you might have. Thank you. |
| **19-20** | Here are the list of references used throughout the presentation. |