**Chris Bull**

**2/22/2025**

**Project Two: Security Policy Presentation**

**https://youtu.be/9f-4Q2Ba1CQ**

**Project 2 Script**

**Title Slide – Green Pace Security Policy Presentation**  
Today, I will be presenting our security policy framework for Green Pace, focusing on the implementation of a Defense in Depth strategy. My name is Christopher Bull, and this presentation outlines our approach to protecting company data at rest, in transit, and during active use. It will also touch upon various coding standards to be followed by our developers here at Green Pace.

In this PowerPoint, we will discuss the underlying principles of our security policies, examine key coding standards, explore our encryption and triple-A policies (Authentication, Authorization, and Accounting), and review our unit testing and automation strategies. This overview is intended to ensure that our security measures not only comply with industry standards but also support the company’s long-term operational integrity and data protection goals.

**Overview – Defense in Depth**  
Defense in Depth is a layered security strategy designed to reduce the overall risk of a security breach by implementing multiple lines of defense. In this slide, you can see a diagram that illustrates the many layers of protection that can be deployed across our departments. Each layer acts as a barrier so that if one is compromised, the others continue to protect our systems, which is a core concept of defense in depth.

Green Pace will be implementing the practice of Defense in Depth across all departments. We will be implementing policies to protect our data when it is stored on our computers, when it is sent between computers, and when it is being accessed on our computers.

Later in the presentation, we will discuss which tools we can use to automate the detection of these coding vulnerabilities.

**Threats Matrix**  
This slide is meant to provide an overview of potential security risks that our organization may encounter. Here, we have organized various types of threats into a clear table format, with explanations summarizing each risk and its potential impact. This kind of approach allows us to assess and prioritize vulnerabilities based on their severity and likelihood of them occurring.

In an ideal world, all standards should be followed. Realistically, the “high priority” and “likely” cells contain the most important standards to adhere to.

**10 Principles**  
This slide outlines the ten core principles that form the foundation of our security policy. These principles ensure that our software development practices align with the best practices in security. Each principle is a guideline meant to minimize vulnerabilities during the development process and maintain system integrity.

To give a summary of each principle:

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Ensure that all input data is validated to prevent malicious input, such as SQL injection or intentional buffer overflow. This helps protect systems from unexpected behavior or security breaches caused by poorly validated data. |
| 1. Heed Compiler Warnings | Pay attention to compiler warnings and resolve them as they can highlight vulnerabilities or coding mistakes. Ignoring them may lead to security issues or instability in the program. |
| 1. Architect and Design for Security Policies | Design systems with security in mind from the start by incorporating security policies into the architecture. This approach ensures that vulnerabilities are minimized during the development phase. Catching the bugs early can be a huge time save for the project as a whole. |
| 1. Keep It Simple | Avoid unnecessary complexity in program code and system design. Simple systems are easier to develop, debug, and maintain, leading to less hidden vulnerabilities. |
| 1. Default Deny | Set all permission systems in the program to deny access by default. In doing this, each user will have restricted access unless explicitly granted access. |
| 1. Adhere to the Principle of Least Privilege | Grant users and systems the minimum access necessary to perform their tasks. This limits the potential damage from accidental misuse or malicious activities. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data before sending it to other systems to prevent injection attacks and ensure compatibility. Proper sanitization protects against exploiting vulnerabilities in the target system. |
| 1. Practice Defense in Depth | Implement multiple layers of security so that even if one layer fails, others will still protect the system. There is a balance that can be struck with this practice. Implement enough defenses to ensure safety without hurting user experience or program performance. |
| 1. Use Effective Quality Assurance Techniques | Incorporate multiple layers of testing, like code reviews, static analysis, and penetration testing, to help identify and fix vulnerabilities early in the development phase. This principle helps remove bugs and exploits before the program is released to production. |
| 1. Adopt a Secure Coding Standard | Follow secure coding standards to ensure consistency and reduce the risk of vulnerabilities being present in the program. This helps contribute to the building of secure and bug-free software. |

**Coding Standards**  
Here, I present our set of coding standards, each of which has been carefully chosen to help prevent specific security vulnerabilities. These standards are intended to guide our developers to program in a way that improves the overall quality and security of our software.

I ranked each of these coding standards with a likelihood of happening as well as a priority labeled as “P” and a number. The higher the number, the lower the priority. As an example, STD-001-CPP has a priority of P12 (lower priority) and a likelihood of “probable”. STD-006-C has a priority of P1 (high priority) and a likelihood of “unlikely”.

**Encryption Policies**

**Encryption at rest** involves protecting data stored on physical media such as hard drives and databases. As an example, our use of Windows Bitlocker at Green Pace ensures that data remains protected even if the physical device is compromised. This policy is important for maintaining the confidentiality and integrity of the information stored on our computers.

**Encryption in flight** refers to the protection of data as it moves across networks. By utilizing protocols such as TLS 1.2 or higher, along with secure email encryption, we ensure that sensitive information remains confidential during its transmission. This helps minimize the risk of interception or unauthorized access during data transfer.

**Encryption in use** involves protecting data while it is actively being processed. Our policy mandates that sensitive documents are decrypted only when absolutely necessary and are immediately re-encrypted once processing is complete. This helps minimize the window of vulnerability and ensures that data is protected even during active operations (like a computer user editing the file).

**Triple-A Policies (Authentication, Authorization, Accounting)**

These slides focus on the Triple-A security policies that we have implemented at Green Pace.

**Authentication** refers to the process where a user provides information about who they are. The data presented is typically information about the user or information that only the user knows. Most of the time, this information is a password that only the user knows. In addition to this, a user can set up multi factor authentication where they must authenticate themselves with multiple forms of authentication. An example of this would be a password and then a prompt on the user’s cell phone.

At Green Pace, all users will be required to have a strong password of at least 12 characters that contain an uppercase, lowercase, and symbol to meet this requirement.

**Authorization** is when users are granted specific permission based on their role within the organization. This tiered access prevents accidental or malicious misuse of sensitive data. To implement this at Green Pace, we have ensured that only system administrators have the ability to modify user permissions. On top of this, only the HR manager will have the ability to add new users to the system. Standard users will not have the ability to perform either of these actions.

Accounting involves tracking user activity, which provides logs for auditing and detecting potential security incidents. Specifically, accounting provides information about a user’s activity on the system. At Green Pace, all user logins will have their IP address, location, and computer being used tracked in an auditing database that only the system administrator can access.

**Unit Testing**  
To demonstrate how unit tests can be used to check the secure coding standards provided in this presentation, I have created four unit tests for the STD-005-CPP – “Do not access freed memory” coding principle.

The first unit test checks to make sure that deleting a pointer and setting it to a null pointer properly deallocates it in memory. I did this by making a pointer and then deleting it and setting it to a null pointer. My test case succeeds if these two lines succeed. I have provided a screenshot of the test case succeeding.

The second unit test simply checks to make sure smart pointers are functioning as intended. It creates a smart pointer and checks that it points to the correct value that we defined. If it does this with no errors, the test case succeeds.

The third unit test checks to make sure that not setting a deleted pointer to a null pointer will result in a dangling pointer. It does this by only deleting a pointer and not setting it to a null pointer. Trying to interact with a dangling pointer often results in program crashes, so I simulated this scenario by checking to make sure that the pointer is not a null pointer. If it isn’t, then the test succeeds because we know that it would be considered a dangling pointer.

The last unit test checks if deleting a pointer twice will cause undefined behavior. In my personal testing, this caused the program to crash. To recreate this as a test case, I deleted a pointer and set it to a null pointer. From my experience, I know that attempting to delete a deleted null pointer causes the program to crash, so I made my unit test succeed if the pointer was a null pointer after deleting it. This test case should succeed every time in its current form because we are manually setting the pointer to a null pointer in the line before the check.

**Automation Summary**  
“This slide summarizes the role of automation in our security strategy, specifically within the DevSecOps pipeline. Automation is important for integrating security into every phase of the software development lifecycle. Tools such as IriusRisk for threat modeling, OWASP dependency-check for vulnerability scanning, and others like JBroFuzz and OWASP ZAP during testing help us maintain DevSecOps security standards.

By automating these processes, we reduce the possibility of human error and ensure continuous monitoring and rapid response to security threats. This proactive approach speeds up the development cycle and enhances overall program security by consistently validating and improving our defenses.

In the planning phase, a tool like IriusRisk can be used to help collaborate with teammates. IriusRisk helps a team collaboratively build threat modeling for a project.

Next, in the build phase, code is consistently scanned for any potential vulnerabilities and exploits with a tool like OWASP dependency-check. This tool will help automate the vulnerability scanning process by looking for utilized libraries with public vulnerabilities.

During the testing phase, a build artifact is created (a functioning early version of the program). It is tested using tools like JBroFuzz, OWASP ZAP, and Arachi which help detect issues with user authentication, authorization, SQL injection, and API-related endpoints.

Lastly, once the program is deployed, tools like Osquery, Falco, and Tripwire will help determine whether an application is functioning as intended while live in a production environment. One way they do this is by purposely messing up one part of the program to make sure other parts remain functional. This is an important aspect of a program to test because programs can face many types of unexpected issues once deployed to a production environment.

DevSecOps saves time spent on implementing security features and fixing vulnerabilities at the end of development, saves money spent on fixing vulnerabilities, and is easier for developers to implement security into the program as they develop. It comes with the downsides of costing more money upfront to build a DevSecOps team and coordinate the plan, having a more complex development cycle, and potentially needing more time to complete a project.

Overall, the benefits of DevSecOps far outweigh the risks for larger projects. Sometimes it may make sense for a smaller project to stick to the traditional software development lifecycle

**Recommendations**  
Based on our analysis of the current security measures, several weak points in the company security policy have been found. The company authentication policy could benefit from some more depth. As of right now, if a user’s password is compromised, their entire account is compromised. We have seen this be an issue in recent cyber-attacks like the Colonial Pipeline attack. In this attack, the company’s lack of MFA on user account led to them being an easy target.

Another policy that can be improved upon is the accounting policy. Green Pace currently logs data about where the user account is being logged in from. This is great for surface level issues, but lacks logging for other important features within a system.

Lastly, the encryption at rest policy could also use some work. It is common for computers to be left unlocked and open while a user is not at their desk. This is not a scenario that BitLocker file encryption can help with. There should be another layer of protection to help in this exact scenario.

**Conclusions**  
To improve the authentication policy at Green Pace, multi factor authentication (MFA) should be required for all employees to prevent unauthorized access to company accounts. This can be required as an app on every employee’s cell phone that verifies it is them trying to sign in when signing in from a new location.

To enhance the accountability policy, logging should be implemented to log when employees access sensitive data within the company system. This way, if a user account is compromised, a system administrator can see a record of all the actions the account performed while compromised which can lead to a much easier recovery.

To enhance the encryption at rest policy, all sensitive documents should be encrypted with the option to be decrypted on command with a key/password. A program can be used to password-protect important user documents so that the password needs to be entered each time they’re opened. This helps prevent a user opening sensitive files on a computer that was left unlocked.