**Project Two: Security Policy Presentation**

Alex Wells

December 10, 2023

[Video Presentation (Youtube Link)](https://www.youtube.com/watch?v=ikCJSGYSbw0)

# CS 405 Project Two Script Template

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | **Hello everyone. My name is Alex Wells and I’m a software security specialist with green pace. Today I will walk you through a presentation on our secure coding standards.** |
| **2** | **Defense in Depth! This is the practice of creating a mutli-layered security system. This has the benefit of not having just one point of failure. This can encompass secure code, encryption policies, mutli-factor authentication, physical security and so much more. If there’s one thing you take away from todays presentation it should be this.** |
| **3** | **The most likely standard violations are:**  **Overflows**  **SQL injection**  **Accessing Free Memory and**  **Not making assert messages clear**  **The highest priority are**  **SQL injection**  **Accessing free memory**  **The lowest priority are.**  **Not making assert messages clear**  **Handle all exceptions.**  **Close files when they are no longer needed**  **Note: Low priority does not mean they don’t need to be implemented and adhered to, just that other standards should be first**  **The more unlikely issues are**  **Modifying the standard namespace**  **Out of bound element access**  **Not closing files when they’re not longer needed.** |
| **4** | **Here are our ten principles to keep in mind when developing secure code:**   * **Validate Input Data** * **Heed Compiler Warning** * **Architect and Design for Security Policies** * **Keep It Simple** * **Default Deny** * **Adhere to the Principle of Least Privilege** * **Sanitize Data Sent to Other Systems** * **Practice Defense in Depth** * **Use Effective Quality Assurance Techniques** * **Adopt a Secure Coding Standard**   **By keeping these in mind we can create a policy of security first and develop clean working code to the best of our ability.** |
| **5** | **Here are our 10 coding standards that should be adopted to minimize potential vulnerabilities in the code.**   * **STD-001-CPP: Do not modify the Standard Namespace.** * **STD-002-CPP: Ensure that operations on signed integers do not result in overflow.** * **STD-003-CPP: Range check element access.** * **STD-004-CPP: Sanitize and tokenize all input to check for potential SQL injection.** * **STD-005-CPP: Do not access free memory.** * **STD-006-CPP: Make assert messages clear to avoid confusion and aid in debugging.** * **STD-007-CPP: Handle All Exceptions.** * **STD-008-CPP: Guarantee that container indices and iterators are withing the valid range.** * **STD-009-CPP: Value-returning functions must return a value from all exit paths.** * **STD-010-CPP: Close files when they are no longer needed.**   **By adopting this standards we can prevent common errors and vulnerabilities from occurring.** |
| **6** | **There are 3 main types of encryption, Encryption in rest, encryption at flight, and encryption in use. These are all focused on different states of data and how to protect it in those states.**  **Encryption in rest is encryption used to protect data that is stored on a disk or other back up media. We will use this to protect the database of passwords and other store information. This can be done through salting and hashing the data or by requiring a secure key to access it.**  **Encryption at flight is used to protect data as it is being transferred between locations, usually server to client or vice versa. This is achieved through various measures such as checksum verifications, private/public keys and session keys.**  **Encryption in use is meant to protect the data as it’s being used by its intended recipient. This can be achieved through multi-factor authentication or single sign on systems. It can be further reduced by following the principle of least privilege and to default deny ensuring that the user does not have access to further data.** |
| **7** | **Triple A: Authentication, Authorization, and Accounting**  **Authentication is having the user prove they are who they say they are, usually through a log-in system with a username and password. You can make this more secure by requiring Multi-Factor Authentication. New users must be added to the authentication service in order to access the system at all.**  **Authorization is about giving access to users to access information. This means that there should be multiple layers of security access and privileges in order to be able to best define what each user should have access to.**  **Accounting is about logging changes and accesses made to the system to be reviewed later. This should log all files accessed by users, all changes made to the database, and many other events. To be effective accounting just can’t be logged, it has to be reviewed otherwise you just create data for the sake of having it.** |
| **8** | **Now we’re going to move on to some unit testing.**  **This first test tests whether an exception is thrown if you resize a collection to be larger than INT\_MAX, this fails because collection->size is governed by an integer so it overflows. This is testing handling of overflow errors.** |
| **9** | **This second test tests access to a collection by its index, the at() method is range checked for us so when a developer tries to access by an index that is out of range it should throw an exception and as you can see it does.** |
| **10** | **This third test tests whether a function that is going to overflow will throw an std::overflow\_error. This was a custom function that added 1 to an integer already at INT\_MAX so it should always throw and as you can see it does.** |
| **11** | **This is the DevSecOps pipeline, its shows the process from pre-production to production and back again as code moves along the pipeline.** |
| **12** | **The DevSecOps pipeline integrates security into the standard development pipeline. This ensures that every step of the way has security in mind so it is at the forefront.**  **We will implement automation tools to ensure that code is compliant with our coding standards. Automation tools are used for the enforcement and compliance to the standards. This will be added to both pre-production and production parts of the DevSecOps pipeline. This will ensure all code created is compliant with the standards.** |
| **13** | **There are always risks and benefits to implementing changes to a development pipeline. However, we can’t afford to wait.**  **If we act now, we can ensure that we create compliant and safe code in line with our standards. The largest cost will be set up, retraining, and retooling our development pipeline.**  **If we wait, we might let an issue slip through the cracks and be in production code, which can leave a vulnerability that can be exploited and cost the company time and money to fix as well as possible reputational damage.**  **REMEMBER: The earlier a problem is caught, the cheaper it is to remedy.** |
| **14** | **This policy does not cover networking standards so more work will need to be done to ensure that vulnerabilities do not exist in connecting to servers and other 3rd party sources.** |
| **15** | **In conclusion, all the outlined standards should be implemented as soon as feasible to prevent vulnerabilities. This will ensure that we create vulnerability free code. In addition to peace of mind, this protects our investment in the software, both time and money, as well as our reputational considerations if a breach occurs and it was easily preventable.** |