# CS 405 Project Two Security Policy Template

**Link to YouTube video presentation:** [**https://youtu.be/h7lAWP1wkLE**](https://youtu.be/h7lAWP1wkLE)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Good morning, everyone, my name is Connie Knupp, from the development team here at Green Pace. As many of you know we have been working to standardize our security policies to ensure consistent compliance with the industries best practices as we advance into the future. Today I will be presenting the newly updated security policy as well as providing implementation guidelines and recommendations for future security policy maintenance. |
| **2** | Standardized security is an essential element in a Defense in depth (DiD) strategy. Defense in depth is a layered approach to security that aims at mitigating risks with multiple overlapping layers of security. It is made increasingly crucial as cyber threats continue to develop. According to the United States Department of Homeland Security, technological advances combined with the growing risks related to artificial intelligence require unshakable diligence and increased versatility to face the cyber threats of today. (Shields ready, n.d.) As developers we are primarily responsible for the application security layer of DiD, although these policies can also touch on network security, host security and endpoint security layers. The policy covers items such as core security principles, coding standards, the triple A’s in security, and data encryption policies. |
| **3** | Priority of each threat considers the likelihood of the exploit being discovered and used by malicious actors, as well as the severity of the risk. Severity is lower for consequences like terminates the program, higher for undefined behavior, and highest if the attackers can use the flaw to run arbitrary code. The priority of the threats addressed in the coding standards section of the policy are itemized in the matrix seen on this slide. High priority threats will be the first to be targeted for correction. |
| **4** | The top 10 secure coding principles, as recommended by Carnegie Mellon University’s Software Engineering Institute (Seacord, 2018) and supported by the industry, are displayed with the coding standards that map to each principle.   1. **Validate input –** Treat all data from non-trusted sources as a potential threat until it is validated, this protects against many of high severity threats.   Standards such as ensuring adequate string storage, checking that no integer operation overflows, preventing SQL injection and detecting memory allocation errors all fall under this principle.   1. **Heed compiler warnings –** ignored warnings can lead to exploitable vulnerabilities.   Standards that ensure adequate string storage fall under this principle as the possibility of inadequate string storage will give a compiler warning.   1. **Architect and design for security policies –** Security is a core requirement and needs to be a primary consideration when planning and designing software architecture. Build security into the foundations to ensure integrity of the system. Practice modularization and abstraction with system components to easily accommodate a range of privileges or security authorization levels.    1. IDS-001-J (Prevent SQL injection)    2. MEM-001-CPP (Memory allocation)    3. MSC-002-CPP (Use assertions correctly) 2. **Keep it simple -** Increased complexity results in increased risk and a greater number of potential weaknesses vulnerable to exploitation. 3. **Default deny -** Protect system assets by defaulting to a denying access unless the access is explicitly granted. This aids in blocking attempts to gain entry or complete requests made from unauthorized parties/users. 4. **Adhere to the principle of least privilege -** Grant users the lowest access level with fewest permissions needed to accomplish their job functions. This principle limits the harm malicious actors can cause in the event they are able to access a user’s account. 5. **Sanitize data sent to other systems -** Ensure outgoing data is transmitted safely, conforms to requirements of the receiving system, is complete, is valid, and is free from harmful elements. This principle ensures the transmitted data does not cause damage to the receiving system.    1. IDS-001-J (Prevent SQL injection) 6. **Practice defense in depth –** As previously explained no single security layer is impenetrable, multiple layers of security are needed to ensure that when one fails another can stop the attack. 7. **Use effective quality assurance techniques -** Utilize effective QA measure such as JUnit tests, fuzz testing, penetration testing, and security reviews to identify weaknesses or flaws so they can be promptly corrected. 8. **Adopt a secure coding standard -** This will improve consistency and reduce risk of vulnerabilities. All standards map to this principle as they are all included in secure coding standards. |
| **5** | The 10 coding standards are ranked here according to the calculated priority.  1. MEM-001-CPP – Detect and handle memory allocation errors.  2. INT-001-CPP – Ensure that operations on signed integers do not result in overflow.  3. IDS-001-J – Prevent SQL injection.  4. STR-001-CPP – Guarantee that storage for strings has sufficient space for character data and the null terminator.  5. EXP-001-CPP – Do not read uninitialized memory.  6. STR-002-CPP – Use valid references, pointers, and iterators to reference elements of a basic\_string.  7. MSC-002-CPP – Value-returning functions must return a value from all exit paths.  8. ERR-001-CPP – Handle all exceptions.  9. FIO-001-CPP – Close files when they are no longer needed.  MSC-001-J – Do not use assertions to verify the absence of runtime errors. |
| **6** | For the risk assessment priority was calculated by multiplying severity (1- 4) times likelihood (1-3) times remediation cost (3-1). Here are the tables included in the security policy detailing the values for severity and likelihood. As you can see the highest severity is reserved for items that allow attackers to run arbitrary code. |
| **7** | Here is the table for the remediation cost. For severity and likelihood 1 is low and 3 or 4 is high, but the value for remediation cost is inverted, 3 is low and 1 is high. The priorities and levels are also detailed in a table that explains what values fall into which category. Low for example has a priority of 5 or less while the highest label has a value greater than or equal to 24. |
| **8** | The first encryption policy is encrypted at rest. Data is considered at rest when it is being stored (for example on a database) and not actively being utilized. Encryption at rest is a policy that states data at rest needs to be encrypted. This policy adds an additional layer of data in the event a malicious attacker gains access to the stored data. Adherence to encryption at rest requires the attacker would either need access to the encryption key or would have to spend a large amount of time using brute force to decrypt the data. This layer of protection makes it harder for attackers to steal sensitive information. Green Pace will utilize AES-256 to encrypt all data at rest and the security team will be responsible for creating and managing encryption keys.  The second encryption policy is encryption in flight. Data is considered in flight when it is in transit, either moving through the system, for example the account balance being pulled from the database to be displayed on the application, or moving outside of the system for example being sent to a partner application. Data in flight is vulnerable to interception or modification while in transit. Encryption in flight is a policy that states data in transit will be encrypted making it less vulnerable to attack. It includes utilizing protocols like transport layer security (TLS), virtual private networks (VPNs) and SSH file transport protocol (SFTP). Green Pace will utilize encryption in flight for all data in transit. The security team will be responsible for creating and managing keys |
| **9** | The final encryption policy is encryption in use. Data is considered in use when it is actively being utilized by the system. Encryption in use is a policy that states data in use will be encrypted to add an additional layer of security. Protecting data in use involves use of tools such as homomorphic encryption or secure multi-party computation to perform calculation on encrypted data bypassing the need to decrypt it. Encryption in use will be used by Green Pace for all sensitive data like passwords or account numbers. |
| **10** | The triple A’s of security are authentication, authorization, and accounting. Authentication is the security process that attempts to ensure the user is who they claim to be. Green Pace will use multifactor authentication for all users who are attempting to access the system. The first factor for authentication will be either be logging in with matching username and password or matching biometrics (fingerprint, facial recognition) if enabled on the device. The second factor for authentication will be a one-time use code that will be sent via text message to the user on their registered number. The passwords will be required to be more than 14 characters with at least 1 uppercase, 1 lowercase, 1 symbol and 1 number. One time use codes will be 6 digits long and only good for 5 minutes after the code has been requested. If the credentials fail to match the user can attempt again. If the user fails to match more than 3 times the user will receive a notification via email and/or text message and the account will be locked until the user completes a secure account unlock process. Once authenticated the username can be logged to track all actions initiated by the user. User login credentials will always be protected by adherence to the encryption policies.  Authorization is the process of determining what activities and resources an authenticated user is allowed to utilize. At Green Pace the users will be authorized for the minimum privileges necessary to accomplish each users’ tasks. If elevated permissions are needed temporarily the request will be approved by administration and the security team prior to being granted. The special permission will only remain in effect as long as required before being dropped to the user’s normal authorization level. Green Pace will have a default deny mentality that will block a user from functionality unless explicitly granted by the user’s authorization level. Adherence to this policy minimizes the damage a bad actor can cause if they gain access to the system via stolen credentials. The addition of new users and changes to users’ level of access will be logged and monitored for excessive changes that may indicate the system has been infiltrated by bad actors. Suspected compromised accounts will have permissions removed until safety has been ensured by the security team.  Accounting refers to logging and tracking the services and resources each user has accessed or attempted to access. Logs will be monitored for unusual activity. Particularly logs will include user login attempts, addition of new users, changes to users’ level of access, which files are accessed by the user, and any changes made to the database. The logs will be included the time of request, if the request was successful and the duration of the activity of session. Successful accounting is essential to a secure system, in the event of an attack this allows the security team to analyze how the breech occurred, what the attackers gained access to, how to stop the attacker, and how to mitigate future risk via the method used by the attacker. |
| **11** | Creating unit tests is an important part of the software design process. Unit tests assist developers in finding errors earlier in the process which means there is less cost associated with correcting the error.  An additional benefit is Unit tests can be rerun after any software updates to verify the new code does not break existing functionality.  Tests can be positive, in that they test the code works as intended or negative. Negative tests verify errors or exceptions are thrown when expected and are used to ensure exceptions are handled with grace. Tests are pass or fail, examples of a passed and failed test are displayed here. |
| **12** | This is an example of a unit test to ensure a collection is empty on creation. It is a positive test that verifies the code works as intended and the collection is empty therefore the size is zero when the collection is created. The results indicate a successful test. |
| **13** | This is an additional example of a positive test, even though it expects a false result it does not throw an error or exception. Expectations indicate items that should have a specific result (indicates proper setup for test) but do not indicate the tests results. Assertions indicate whether the code ran as intended and indicate test results. This test to ensure one item can be added to an empty collection was successful. |
| **14** | Here is an example of a negative unit test. This test expects the standard out of range exception as it attempts to access an out of bounds index. As you can see it uses the Assert Throws function to indicate an exception should be thrown. This test completed with a ok status so it passed. |
| **15** | A second example of a negative test, this one expects the length\_error exception to be thrown as the program attempts to reserve more than the max size. Like the preceding examples the results were ok and the test passed. |
| **16** | The focus on security needs to occur in all stages of development as illustrated in the image above. In the assess and plan stage developers need to analyze the threat landscape as well as the regulations that may apply to the project. When updates are needed to respond to new threats developers need to assess and plan carefully how they can correct software flaws and mitigate the threats. In the design and build phase developers need to plan for security by following best practices which include utilizing test-driven designs. It also includes strict adherence to the secure coding standard adopted by Green Pace.  It is vital to add automated testing like unit tests and static testing in the verify and testing phase of software development. The verify and testing phase should also include having the system analyzed by security experts to uncover additional vulnerabilities prior to release. As illustrated in the DevSecOps image security remains a priority and is a major component of the post release phases. In the transition and health check phase security settings are verified and penetration testing is completed. During the monitor and detect phase the system’s many layers of defenses are monitored for abnormalities so that any intrusion is detected as quickly as possible. In the event of a threat steps are taken during the respond phase to block the attack and roll back the system if needed to return to safe operations. During maintain and stabilize the system is assessed against its established security baseline and steps are taken to return to baseline following an attack. |
| **17** | The DevSecOps pipeline is a continuous loop for software development that focuses on security at every stage. Tools can be used in several stages and are easily visualized with green text on the slide currently displayed. In the verify and testing stages tools like static analysis and unit tests can be utilized. In the transition and health check phase tools are used for intrusion testing. In the monitor and detect phase SIEM tools (security information and event management) tools can be effective. |
| **18** | The risks of waiting longer for security far outweigh the risks of implementing now. They include loss of trust from customers if we are exploited. Cost of attack. Increased cost and production time when errors are found later in development. Risks of implementation include temporary struggles with learning curve for developers and code may become more complex. Benefits include less risk to the company as well as to the customers. Lower cost associated with errors found sooner in the development cycle. Professional growth and improved quality in production code. |
| **19** | This policy was intended as a starting reference. I recommend expanding it to include standards for emerging threats and existing threats not yet covered. There are standards listed for many languages on the website for Carnegie Mellon Unversitity’s software engineering institute. I recommend taking the steps listed on this slide.  Implement encryption where it is missing so existing servers meet standards.  Answer any questions that are brough forward by developers learning new policy.  Evaluate existing software projects to determine if updates to improve security are needed.  Example: Are all systems multifactor authentication?  Create incident response procedures  Who is notified?  Guidelines for when to shutdown and/or rollback  Follow security policy in all projects moving forward.  Provide continuing education on secure practices as the industry progresses into the AI inundated future. |
| **20** | In conclusion moving forward Green Pace will adopt secure coding best practices and standards outlined in the security policy. This includes:   * + 1AAA   + Encryption standards     - Rest     - Flight     - In Use   + Defense in depth   + Top 10 security principles   + Coding standards depicted   Green Pace commits to continually improving our security policies and adhering to industry best practices. |

**References**

Ballman, A. (2025a, July 21). *Err51-CPP. handle all exceptions*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

Ballman, A. (2025b, July 8). *MSC52-CPP. value-returning functions must return a value from all exit paths*. Carnegie Mellon University: Software Engineering Institute. . <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC52-CPP.+Value-returning+functions+must+return+a+value+from+all+exit+paths>

Long, F. (2023, January 10). *MSC60-J. Do not use assertions to verify the absence of runtime errors.* Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/java/MSC60-J.+Do+not+use+assertions+to+verify+the+absence+of+runtime+errors>

Mariani, F. (2025a, August 6). *Str31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>

Mariani, F. (2025b, July 2). *Str52-CPP. Use valid references, pointers, and iterators to reference elements of a basic\_string*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR52-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+basic_string>

Mohindra, D. (2025, August 6). *IDS00-J. Prevent SQL Injection*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

Pincar, J. (2025a, July 25). *Exp53-CPP. do not read uninitialized memory*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

Pincar, J. (2025b, July 2). *Fio51-CPP. close files when they are no longer needed*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

Pincar, J. (2025c, June 25). *Mem52-CPP. detect and handle memory allocation errors*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors>

Seacord, R. (2025, May 5). *INT32-C. ensure that operations on signed integers do not result in overflow*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow>

Seacord, R. (2018, May 2). *Top 10 secure coding standards.* Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow>

United States Government. (n.d.). *Shields ready*. Cybersecurity and Infrastructure Security Agency CISA. https://www.cisa.gov/shields-ready