# CS 405 Project Two Script Template

Ashley Santo

October 15, 2022

Project 2 Security Policy Presentation

[**https://youtu.be/ptCSRwE8ix0**](https://youtu.be/ptCSRwE8ix0)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello! My name is Ashley and today I’ll be giving a presentation on the security policy at Green Pace. |
| **2** | This security policy defines the secure coding principles and standards that should be followed in order to prevent or delay attacks. This will support the defense in depth best practice because it adds more layers to security. Just to summarize what Defense in Depth is, basically it is a series of security measures layered to protect the data. Each layer acts as a fail-safe should the layer before become compromised. This helps stop or at least slow attacks until security employees can go in and prevent the attack from happening. |
| **3** | The coding standards within this policy go through a risk assessment to determine how critical they are. This helps indicate how disastrous the consequences may be if the standard is ignored.  The threats matrix table shows some ways this is assessed. How likely or unlikely are they to lead to vulnerabilities and how severe are the vulnerabilities. There are more measures that go into risk assessment such as cost to correct for example, but these are the more major deciding factors on whether something should be corrected. There are automatic tools that can analyze the code base and determine how critical the error, message or warning is. This can help developers catch issues they might not see or that the IDE they are working in didn’t find. |
| **4** | These are the 10 principles for this policy:  One, validate input data. Validating input from all untrusted data sources can eliminate many exploitable vulnerabilities. There should be suspicion of most external, untrusted data sources such as command line arguments, network interfaces, and user-controlled files.  Two, heed compiler warnings. Code should be compiled using the highest warning level available for the complier being used and eliminate warnings by editing the code.  Three, architect and design for security. Design a software architecture with security policies in mind. The software should enforce these policies.  Four, keep it simple. Developers should try to keep the overall design as simple as possible since the more complex designs can increase the chances that errors will occur once implemented, configured and used.  Five, default deny. Access should be permitted only to the highest level needed for a user to perform their necessary actions. Access should be denied by default with permissions granted on a needed basis.  Six, adhere to the principle of least privilege. In order to prevent an attacker from having the opportunity to have higher privileged access to code, all processes should execute using the least number of privileges need to complete the request. If an elevated privilege is granted, it should only be for as little time as possible and the privilege should be revoked as soon as it is done.  Seven, sanitize data sent to other systems. All data that is put into complex subsystems such as command shells, relational databases, and more should be sanitized so that attackers aren’t able to exploit the unused functionality through SQL, command or other injection attacks.  Eight, practice defense in depth. There should be multiple defense strategies so that if one layer fails, another is there to either prevent an attack or at a minimum, slow down the attacker. By having multiple layers, it can stop any exploits or at least minimize the consequences of the exploit.  Nine, use effective quality assurance techniques. Having good quality assurance techniques can help in identifying and removing vulnerabilities. This can involve multiple kinds of testing.  Ten, adopt a secure coding standard that should be created and applied for the target languages and platforms being used. |
| **5** | The ten coding standards are:   1. Do not attempt to create a std::string from a null pointer 2. Do not store an already-owned pointer value in an unrelated smart pointer 3. Properly deallocate dynamically allocated resources 4. Make sure that integer conversions do not cause lost or misinterpreted data 5. Valid references, pointers, and iterators should be used to reference elements in a container 6. The order of evaluation should not be depended on for side effects. 7. All exceptions thrown should be handled before the main() function begins it’s execution 8. Do not alternate input and output from a stream without flushing or an intervening positioning call 9. Virtual functions can’t be invoked from constructors or destructors. 10. Use static assertion techniques to test expressions in code   The system of prioritization used starts with the standards that have high severity and high priority. For instance, the first three all are considered high priority and should be corrected as soon as possible to avoid the severe consequences. After that is moving onto those with high severity but lower priority. Once we get to the low severity standards, we prioritize those with higher priorities and likelihoods and then look at the cost to correct. |
| **6** | Next, is the encryption policies. Data at rest refers to any data residing in computer storage in any digital form. This data type is currently inactive and not moving between devices or network points. So, no app, service, tool or anything else is actively using this type of info. At rest encryption is a security practice of encrypting the store data to prevent unauthorized access. Data is scrambled into ciphertext and a decryption key is required to see it. This is crucial because most of this type of data contains private and valuable information.  Encryption in flight refers to any data that is being transmitted over a network. All data that goes over an internal network or the internet is vulnerable to being intercepted. Encrypting in flight means that the data is encrypted before transmission, the computer system endpoints are then authenticated and the data is decrypted and verified on arrival.  In use encryption guarantees that sensitive data is never left unsecured, regardless of its location or lifecycle stage (at rest/in flight). The data is always encrypted so even if there is a data breach, data loss is prevented. It also analyzes data requests in real time and blocks suspicious requests. |
| **7** | Then, we have our Triple A policies. Authentication refers to a method of identifying a user. This is normally done by having the user input a valid username and password before access is granted into the network. Each user has unique credentials for gaining access and the server compares the entered credentials with credentials in a stored database. If there’s a match, network access is granted but if there is not a match, network access is denied.  Authorization is the process of enforcing policies where user permissions are decided. It determines what types of activities, resources or services a user is permitted to use. This usually goes hand in hand with authentication where once authentication is proven, authorization to specific processes is given.  Accounting monitors the resources a user consumes during network access. This can include the amount of data sent and received during a session or logging the amount of time spent in the system. |
| **8** | Unit testing is a great way for developers to see if their code is functioning correctly as they are writing it. There are positive tests where we verify the code is working as it should. Or negative tests to make sure it responds correctly to failures. We can have negative tests that are written to pass or fail. This is an example of a simplistic coding vulnerability of whether two integers equal each other and if added together, would they come out to the specified value.  The first test is a positive test of making sure the variable x equals y. As we can see, it passed the test. |
| **9** | The second and third tests are negative tests. One is supposed to fail while the other should pass to confirm it caught the failure. The check is to confirm x does not equal y.  Test one uses ASSERT\_FALSE when x equals y. While the program expected a false result, it was actually true and therefore failed.  Test two uses ASSERT\_FALSE when x doesn’t equal y. The program passes this test because it’s confirming that x doesn’t equal y in this case as it was written. |
| **10** | This last unit test example checks to see if the variable z is the sum of x and y. The test verifies this is true. |
| **11** | This is a diagram of the DevSecOps automation summary. There should be security automation at the verify and test stage where code is tested for vulnerabilities, errors, bugs, or warnings. Monitor and detect is another automated stage where logs and analytics are documented by the system to track any issues. Respond is automated where if an attack is to occur, how should the system handle it to block the attack. |
| **12** | To go into more detail, DevSecOps stands for development, security, and operations. It’s an approach to automation and platform design that integrates security as a shared responsibility throughout the entire development lifecycle. The idea is that security should be collaborative and not just done by one team at the end of the lifecycle. This way, security is implemented from end to end where application and infrastructure security is planned from the start. It also means automating security gates to keep the workflow from being slowed down for checks along the way. The main idea is that security should be handled sooner rather than later. |
| **13** | Acting quickly or waiting to respond to vulnerabilities is dependent on the risk assessment. High risk vulnerabilities should be handled right away while low risk vulnerabilities can probably wait. But, if the cost to fix the issue is high, it might get pushed back behind issues that are low cost to fix. This is where the strategy is lacking in that every issue comes with its own risk assessment and certain factors outweigh others. The risks could be that even though a high-risk item is low cost to fix, it might not have the same severity in consequences as a high-cost vulnerability that is also high risk.  Each risk should be assessed to determine the overall severity of the consequences should an attacker try to exploit it. From there, determine the cost and likelihood ignoring it for will lead to exploitable code. |
| **14** | One of the major gaps is that not every security risk is going to come from vulnerable code. Sometimes, an attacker will utilize social engineering or other tactics to gain access. An example of this is the Uber security breach that occurred on September 15, 2022. Uber stated that the employee’s account was compromised and most likely, their information was on the dark web. This allowed the attacker to gain this information and then spam the two-factor authentication until the employee accepted and the attacker was given access. From there, they were able to access other employee accounts and grant elevated permissions to gain access into Uber’s code base and other internal systems.  Due to this, my recommendation for future policies is to have staff training and policies on what to look out for with other potential methods attackers may use. There is more to security than just vulnerable code. There are physical and social security risks that can be used to access restricted areas in the network. We must always be diligent to protect every aspect of the network. |
| **15** | In conclusion, following the policies, principles and standards listed here are going to help prevent potential breaches in security. Security should always be considered a top priority regardless of the lifecycle stage the software is in. It’s up to everyone on a team to work together to prevent vulnerabilities from being exploited. New standards should be added as they become relevant or the risk of creating a specific vulnerability is higher. Technology is ever-changing and we must keep policies and standards to match the newest potential hazards. |
| **16** | These are the references used if you want to learn more. |