# Joshua Pardue

2-18-23

CS 405 Project Two: Security Policy Presentation

**https://youtu.be/GYMHjfm6PZE**

Complete this template by replacing the bracketed text with the relevant information.

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | This is a presentation for the Green Pace Security Policy. The Developer and narrator is me, Joshua Pardue. The following slides will define the specifics of the policy and detail the related particulars. |
| **2** | With defense in depth being a priority of this policy, Green Pace prioritizes security principles in all their applications which means that they are committed to maintaining a consistent approach and methodology to ensure that all policies are uniformly defined, implemented, governed, and maintained over time.  To achieve this goal, Green Pace has established a set of guidelines and protocols that are consistently followed by all members of their development team. These guidelines cover a range of topics such as access control, data encryption, vulnerability testing, and incident response. |
| **3** | The threats matrix lists standards and their respective severity from high to low. The matrix also defines the likelihood, cost, priority, and threat level to further understanding. |
| **4** | There are 10 core security principles that provide a comprehensive framework for establishing effective security strategies and maintaining the security of Green Pace systems and data.  1. Validate Input Data  2. Heed Compiler Warnings  3. Architect and Design for Security Policies  4. Keep It Simple  5. Default Deny  6. Adhere to the Principle of Least Privilege  7. Sanitize Data Sent to Other Systems  8. Practice Defense in Depth  9. Use Effective Quality Assurance Techniques  10. Adopt a Secure Coding Standard  These principles correlate with the Green Pace coding standards starting with   * Data Type *10*. *Adopt a Secure Coding* Standard is the principle correlated with this standard. Mistaking an integer's size for its precision can permit invalid precision arguments to operations such as bitwise shifts, resulting in undefined behavior. * Data Value 7. *Sanitize Data Sent to Other Systems* correlates with this standard as using the object representation of a floating-point value for comparisons can lead to incorrect equality results, which can lead to unexpected behavior. 10. Adopt a Secure Coding Standard could also be closely correlated with this standard. * String Correctness 1. Validate Input Data and *8. Practice Defense in Depth* could be correlated with this standard as incorrectly specified format strings can result in memory corruption or abnormal program termination. * SQL Injection 1. *Validate input data* correlates with this standard as failing to validate the parameters in library functions may result in an access violation or a data integrity violation. Such a scenario indicates a flaw in how the library is used by the calling code. However, the library itself may still be the vector by which the calling code's vulnerability is exploited * Memory Protection 7. Sanitize Data Sent to Other Systems, 8. Practice Defense in Depth, and 1. Validate Input Data could all potentially be correlated with this standard as providing invalid size arguments to memory allocation functions can lead to buffer overflows and the execution of arbitrary code with the permissions of the vulnerable process. * Assertions *2. Heed Compiler Warnings and 9. Use Effective Quality Assurance Techniques* correlates with this standard as static assertion is a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities at compile time. The absence of static assertions, however, does not mean that code is incorrect. * Exceptions *3. Architect and Design for Security Policies* and *10. Adopt a Secure Coding Standard* correlate with this standard as allowing the application to abnormally terminate can lead to resources not being freed, closed, and so on. It is frequently a vector for denial-of-service attacks. * Data Leak *7. Sanitize Data Sent to Other Systems* is correlated with this standard due to padding bits might inadvertently contain sensitive data such as pointers to kernel data structures or passwords. A pointer to such a structure could be passed to other functions, causing information leakage. * Error Handling 1*.Validate Input Data, 7. Sanitize Data Sent to Other Systems, 10. Adopt a Secure Coding Standard* all could potentially correlate with this standard as failing to detect error conditions can lead to unpredictable results, including abnormal program termination and denial-of-service attacks or, in some situations, could even allow an attacker to run arbitrary code. * Pointer Arithmetic *8. Practice Defense in Depth and 10. Adopt a Secure Coding Standard* are principles that can be correlated with this standard as using arrays polymorphically can result in memory corruption, which could lead to an attacker being able to execute arbitrary code. |
| **5** | The coding standards have been matched to the principles in the previous slide but in furthering understanding, these standards define a consistent and uniform approach to writing code and help to reduce the risk of introducing security vulnerabilities and weaknesses in the code. |
| **6** | Encryption is essential for securing Green Pace sensitive data such as credit card details, personal identification information, and financial transactions. The three forms of encryption we use in our policy include:  Encryption in rest This refers to data that is stored in an encrypted form when it is not in use. Process of encoding sensitive data transmitted over a network to prevent unauthorized access. In REST, encryption is used to secure the communication between a client and a server, and to protect sensitive data, such as passwords, credit card numbers, and other confidential information, from being intercepted and read by unauthorized parties even if the device is stolen.  Encryption at flight When data is encrypted at flight, it helps to ensure that information remains confidential and secure during transmission. It is transformed into a code that is unreadable by anyone who does not have the key to decrypt it. This means that even if an attacker were to intercept the encrypted data, they would not be able to access the sensitive information contained within it.  Encryption in use Data that is encrypted while it is being processed by an application or service. This ensures that the data remains protected even if the device or network it is being processed on is compromised. It is often implemented with other security measures, such as password protection, two-factor authentication, and firewalls, to provide multiple layers of protection for sensitive information |
| **7** | Green Pace also uses Triple A policies including Authentication, Authorization, and accounting which are defined as:   * Authentication Determines who is trying to access the system and to ensure that only authorized users are able. Process of verifying the identity of a user, device, or system. Authentication is usually accomplished by requiring the user to provide a username and password. * Authorization helps to enforce security policies and to prevent unauthorized access to sensitive data. It is process of granting or denying access to resources or services based on the user's identity. After a user has been authenticated, authorization is used to determine what the user is allowed to do with respect to making changes or their level of access. * Accounting Used to monitor resource usage, to detect security incidents, and to audit compliance with security policies. This process keeps track of user activity and resource usage such as files accessed. It provides an avenue to provide a record of who has accessed a system, what they did, and when they did it. |
| **8** | The following unit tests for Green Pace are focused on testing individual units or components of a system or application. It is designed to test the functionality, security, and reliability of individual pieces of code or modules, and to ensure that they work correctly and securely when integrated into the larger system.  The slides consist of:   * Verify capacity is greater than a certain size * Verify resizing decreases the collection * Negative test to verify exception is thrown * Test collection size is proper * Negative test to remove an element and check collection |
| **9** | This is the CapacityIsGreaterOrEqual test.  This test verifies that capacity is greater than or equal to size for 0, 1, 5, 10 entries.  It tests the capacity of a collection by adding 11 entries to it and checking the capacity is greater than or equal to certain values using a series of ‘ASSERT\_TRUE’ statements |
| **10** | This is the ResizeDecrease test.  This test verifies resizing decreases the collection to zero.  It tests the behavior of the collection when it is resized to a smaller size using the resize method and ‘ASSERT\_TRUE’ checks the new size is less than the previous. |
| **11** | This is the OutOfRangeTest.  This is a negative test to verify the std::out\_of\_range exception is thrown when calling at() with an index out of bounds.  The purpose of the ‘at’ method is to throw an exception when an index is out of range. |
| **12** | This is the PardueTestOne which tests the behavior of a collection after calling the ‘pushback’ method to add an element. It verifies the collection can correctly add and retrieve elements along with its size can be properly updated after adding elements. It also ensures that the collection properly indexes elements starting from zero and that the element added to the collection has the expected value. |
| **13** | This is the PardueTestTwo which is a negative test because it is checking the size of the collection is not equal to 10 after removing an element using ‘pop\_back’ method. It verifies the size of the collection changes after removing an element and the test is considered negative as it verifies the implementation behaves as expected when the elements are removed from the collection. |
| **14** | Automation summary  Automation will be utilized to ensure and enforce coding standards in several ways corresponding with the Green Pace DevOps process of development, security, and operations which will complement its infrastructure with respect to pre-production and production phases. |
| **15** | Tools  There are various tools associated with the Green Pace policy. Beginning in pre-production and formulation of a plan, the process will be analyzed for current processes, systems, and technology in determination of automation. The design phase will consist of defining the architecture and systems such as determining requirements, security design, performance design, and risk analysis. The build phase will include code development with testing, debugging, and particulars such as release management. This is followed by verification testing which will be implemented using the proper testing platform such as static analysis and vulnerability scanners such as Nessus or OpenVAS.  With an agile methodology in position, the production phase requires consistent collaboration with security settings and health checks from pre-production. Requirements will be designed and built with respect to authentication and authorization such as LDAP to manage user credentials and control access to resources. Tools will be utilized to monitor and detect issues and anomalies such as Application Performance Monitoring from New Relic or AppDynamics. Log Monitoring is a necessary option as well with Logstash to collect and analyze data from varying sources. Implementation of response criteria will include necessary tools from the design phase such as Web Application Firewalls or Cloud Security Tools to mitigate attacks on applications and infrastructure. The process then moves to maintaining and stabilizing the software making assurances it is reliable, stable, and performs as necessary. |
| **16** | Risks and Benefits  The use of automated security testing in general provides an avenue to test an application faster than manual testing. It allows for testing potential vulnerabilities quickly and efficiently. It also allows for better coverage as tests are executed over many test cases and scenarios. Another benefit is repeatability as this sort of testing can be done consistently which would ensure risky portions are identified and corrected. They eliminate the human component of error by providing consistent results as individuals often make mistakes by missing a vulnerability or not thoroughly testing. Automated testing ensures further accuracy and reliability in this manner.  The entire process of development should be constructed with considering security issues and vulnerabilities. If the program is designed with security in mind, these issues and vulnerabilities are easier spotted early in the process. With professionalism, quality of code, and best practices in mind, the code looks and performs more efficiently as well because patches and fixes do not have to be implemented after the completed product is rolled out. The security is built into the code and can be adjusted and tuned accordingly and not just an inefficient after thought. From a cost standpoint this plan makes sense also as it is easier for developers to address an issue early on in development versus spending time and money to backtrack and re-develop a certain portion. |
| **17** | Recommendations  While the Green Pace security process is reasonably failproof and thorough, areas lacking such as physical devices like laptops, phones, and hard drives can lead to operational disruptions. If a system is compromised, it can result in downtime, meaning individuals may not be able to access various systems and data necessary to complete their work. This can lead to decreased productivity, missed deadlines, and lost revenue. It could also disrupt a company’s supply chain, leading to delayed shipments, lost inventory, and a myriad of other negative circumstances. |
| **18** | Conclusions  In conclusion, there are a few recommendations that will ensure layering and security stays at an optimum level.  Use Strong Passwords - Strong passwords that include a mix of letters, numbers, and special characters can help prevent cyber-attacks. It is also important to use different passwords for different accounts and to change them regularly.  Keep Software Up to date - Keeping software up to date with the latest security patches can help prevent cyber-attacks. Hackers often exploit vulnerabilities in software, and software vendors regularly release patches to fix these vulnerabilities.  Install Antivirus and Firewall Software - Antivirus and firewall software can help prevent cyber-attacks by blocking unauthorized access to your computer or network and detecting and removing malware.  Train Employees - Cybersecurity training can help employees understand how to identify and prevent cyber-attacks. This can include training on phishing scams, social engineering tactics, and password best practices.  Use Two-Factor Authentication - Two-factor authentication (2FA) adds an extra layer of security to your accounts by requiring a second form of authentication, such as a code sent to your phone, in addition to a password.  Limit Access to Sensitive Data - Limiting access to sensitive data to only those who need it can help prevent cyber-attacks. This can include using role-based access control and regularly reviewing access permissions.  Backup Your Data - Regularly backing up your data can help prevent data loss in the event of a cyber-attack. Backups should be stored offsite and encrypted to protect against unauthorized access.  Implement Network Segmentation - Network segmentation involves dividing a network into smaller segments, each with its own security controls. This can help prevent the spread of a cyber-attack if one segment is compromised.  Conduct Regular Security Audits - Regular security audits can help identify vulnerabilities and gaps in cybersecurity defenses. Audits should include a review of policies, procedures, and technical controls. |
| **19** | References  Here is a list of the various references used throughout this presentation. Thank you. |