**7-2 Project Two: Presentation Script**

**Ryan DeBraal**

**CS-405-X6389 Secure Coding 21EW6**

**URL:** <https://www.youtube.com/watch?v=BvE5eyJgo7s>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | **INTRODUCTION**  Hello, my name is Ryan DeBraal.  I’m presenting our brand-new security policy. |
| **2** | **OVERVIEW: DEFENSE IN DEPTH**  Secure coding is at the heart of all the development.  Important to stay in sync with each other  Today we will review our principles and best practices.  Defense in depth requires many layers of redundant protection. |
| **3** | **THREATS MATRIX**  Cross section of standards. Mix between priorities and likelihood. A threat analysis should be done to prioritize High Priority *and* Likely threats. |
| **4** | **10 PRINCIPLES**   | **Principles** | Write a short paragraph explaining each of the 10 principles of security. | | --- | --- | | 1. ValidateInput Data | This principle describes the need to verify all input into the information system has been validated. For example, all fields are complete all data types are correct. | | 1. Heed Compiler Warnings | This principle describes the need for developers pay attention to what warnings the compiler throws and to correct their code appropriately. | | 1. Architect and Design for Security Policies | This principle describes the need to proactively code with security in mind. A developer should assume that every line of code they write could be used by a malicious user to breach the security of the system. | | 1. Keep It Simple | This principle describes the need to construct code in a planned and eloquent manner so that its complexity doesn’t become overwhelming. | | 1. Default Deny | This principle describes the need to block all inbound and outbound traffic that has not been expressly permitted by firewall policy. | | 1. Adhere to the Principle of Least Privilege | This principle describes the need to allowing users only enough access to perform the required job. | | 1. Sanitize Data Sent to Other Systems | This principle describes the need to eliminate unwanted characters from input by means of removing, replacing, encoding, or escaping the characters. This includes HTML, SQL, JavaScript, or any other input that could be used for injection or cross-site scripting (XSS) attacks. | | 1. Practice Defense in Depth | This principle describes the need to protect data under multiple redundant layers of security to avoid exploitation. | | 1. Use Effective Quality Assurance Techniques | This principle describes the need to perform rigorous quality assurance (QA) checks to effectively reduce the number of defects and exploitable code in a piece of software. | | 1. Adopt a Secure Coding Standard | This principle describes the need to prescribe to standards/rules/guidelines that, if used consistently, can help prevent security vulnerabilities. | |
| **5** | **CODING STANDARDS**   | Standard | Name of Standard | Applicable Principles | | --- | --- | --- | | STD-001-CPP | Do not cast to an out-of-range enumeration value [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value)] | Heed Compiler Warnings Keep It Simple | | STD-002-CPP | Use valid iterator ranges [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges)] | Heed Compiler Warnings Keep It Simple | | STD-003-CPP | Range check element access [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access)] | Architect and Design for Security Policies Keep It Simple Practice Defense in Depth Adopt a Secure Coding Standard | | STD-004-CPP | Sanitize data passed to complex subsystems [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/VOID+STR02-CPP.+Sanitize+data+passed+to+complex+subsystems)] | ValidateInput Data Architect and Design for Security Policies Practice Defense in Depth Use Effective Quality Assurance Techniques Adopt a Secure Coding Standard | | STD-005-CPP | Do not access freed memory [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory)] | Architect and Design for Security Policies Keep It Simple Default Deny Adhere to the Principle of Least Privilege Practice Defense in Depth Use Effective Quality Assurance Techniques Adopt a Secure Coding Standard | | STD-006-CPP | Use a static assertion to test the value of a constant expression [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/VOID+DCL03-CPP.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression)] | Heed Compiler Warnings Architect and Design for Security Policies Keep It Simple Use Effective Quality Assurance Techniques | | STD-007-CPP | Do not allow exceptions to transmit sensitive information [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/VOID+ERR12-CPP.+Do+not+allow+exceptions+to+transmit+sensitive+information)] | Architect and Design for Security Policies Sanitize Data Sent to Other Systems Practice Defense in Depth  Adopt a Secure Coding Standard | | STD-008-CPP | Close files when they are no longer needed [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed)] | Architect and Design for Security Policies Keep It Simple Practice Defense in Depth  Adopt a Secure Coding Standard | | STD-009-CPP | Do not use std::rand() for generating pseudorandom numbers [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC50-CPP.+Do+not+use+std%3A%3Arand%28%29+for+generating+pseudorandom+numbers)] | Keep It Simple | | STD-010-CPP | Do not destroy a mutex while it is locked [[link](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON50-CPP.+Do+not+destroy+a+mutex+while+it+is+locked)] | Architect and Design for Security Policies | |
| 6 | **ENCRYPTION POLICIES  Encryption in rest** If an attacker were to obtain a hard drive with encrypted data but not the encryption keys, the attacker would need to defeat the encryption to read the data.  **Encryption at flight** This protection is achieved by encrypting the data before transmission, authenticating the endpoints, and decrypting and verifying the data on arrival.  **Encryption in use** This type of encryption comes in many forms including CPU-based key storage (CPU registers hold encryption keys instead of RAM), enclaves (data is encrypted in RAM but seen as clear test by CPU), and homomorphic encryption (perform computations on its encrypted data without first decrypting it). |
| **7** | **TRIPLE-A POLICIES  Authentication** Authentication is based on each user having a unique set of login credentials for gaining network access.  **Authorization** After logging in to a system, for instance, the user may try to issue commands. The authorization process determines whether the user has the authority to issue such commands.  **Accounting** This can include the amount of system time, or the amount of data sent and received during a session. Anything anomalous (when compared to existing analytics) may be evidence of a data breach. |
| **8** | Unit Testing Explanation |
| **9** | Unit Testing Explanation |
| **10** | Unit Testing Explanation |
| **11** | Unit Testing Explanation |
| **12** | Unit Testing Explanation |
| **13** | Unit Testing Explanation |
| **14** | **AUTOMATION SUMMARY**  Continuous Integration and Continuous Deployment is an important practice in the SDLC.  In order to ensure our software is “healthy” we must be aware of its condition at every stage. |
| **15** | **TOOLS**  DevSecOps isn’t just a buzzword, it’s a way of operating a secure business.  Jira ticketing system to maintain user stories Git repo to store code Maven to implement automated builds  JUnit for functional testing  Selenium for UI testing Jenkins for automated deployment  AWS for hardware resources Docker for containers  Ansible for configuration  Splunk for logging |
| **16** | Risks and Benefits Explanation |
| **17** | Recommendations Explanation |
| **18** | Conclusions Explanation |
| **19** | References, Citations, Questions, and comments |