# CS 405 Project Two Script Template

**CS-405-H7070-23EW2 Secure Coding**

**Richard Backscheider**

**Instructor: Ahlam Alhweiti**

**Southern New Hampshire University**

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Green Pace  Security Policy Presentation  Developer: Richard Backscheider  CS-405-H7070-23EW2  Secure Coding  Instructor: Ahlam Alhweiti  Southern New Hampshire University |
| **2** | Defense in Depth (DiD) is a cybersecurity strategy employing multiple layers of mechanisms to safeguard an organization's IT and infrastructure data (Delinea, n.d.). It integrates human expertise and technological resources through diverse security layers, addressing various attack vectors (Delinea, n.d.).. DiD’s advantages include effective threat counteraction and redundancy, minimizing the impact of individual security solutions on the entire network (CIS, 2021). |
| **3** | Effectively managing protection risks requires a systematic approach (Calle, 2022). Initiated by identifying potential threats using a risk management checklist, the process gains insights from past projects to anticipate and prevent unforeseen issues (Calle, 2022). A risk and control matrix is employed to address unpredictability, assigning responsibilities in advance and ensuring regular updates (Calle, 2022). Prioritization follows, involving the assessment and ranking of risks based on impact and likelihood (Calle, 2022). Steps include measuring probability, assessing impact, calculating total risk, and collaboratively updating the matrix with the team (Calle, 2022). This continuous effort ensures a dynamic and resilient risk management program, preventing unexpected risks and enhancing project outcomes (Calle, 2022). |
| **4** | To ensure secure application development, integrate secure coding practices and risk management into day-to-day operations and development processes (UC Berkley, n.d.). Regardless of the programming device, developers must follow secure coding requirements (UC Berkley, n.d.). The Application Security Training Program, mandated by MSSEI 6.2, covers essential areas like input validation, authentication, and cryptographic practices (UC Berkley, n.d.). The Application Security Testing Program verifies compliance, and Open Web Application Security Project (OWASP) Secure Coding Guidelines outline essential principles for both web and non-web applications (UC Berkley, n.d.). Additional resources, such as training courses and reading material, enhance proficiency in security coding (UC Berkley, n.d.). Integrate secure coding practices across all stages of the software development life cycle (SDLC) and perform automated application security testing (UC Berkley, n.d.). Development and testing environments should handle sensitive data (UC Berkley, n.d.). |
| **5** | The vulnerability ranking system assesses the vulnerability based on its potential impact and likelihood of exploitation (Seacord, n.d.). A severity level is initially assigned, considering effects on system confidentiality, integrity, and availability (Seacord, n.d.). The evaluation then explores the likelihood of exploitation, factoring vulnerability prevalence, ease of exploitation, and attacker motivation (Seacord, n.d.). Additionally, the system analyzes the vulnerability’s relevance to the current threat landscape, aligning with known and emerging threats (Seacord, n.d.). Lastly, the system assesses compliance with coding standards, prioritizing violations of high-priority standards (Seacord, n.d.). This comprehensive risk assessment is the foundation for prioritization and remediation efforts, ensuring a focused and practical approach to enhance codebase security (Seacord, n.d.). |
| **6** | **Encryption**   * **Encryption at rest:** Data encryption at rest provides a protective layer for stored information, encompassing various storage mediums like hard drives, phones, computers, and cloud assets (McVicker, 2022). Safeguarding this data involves employing encryption tools, implementing disk encryption, and ensuring security measures for mobile devices and computers (McVicker, 2022). * **Encryption at flight:** Encryption in transit pertains to safeguarding data while in motion (McVicker, 2022). It involves protecting data as it moves between devices within a network or traverses outside of a network (McVicker, 2022). Security measures for this type of encryption include email encryption, Data Loss Prevention (DLP) solutions, and robust network security features such as firewalls and authentication (McVicker, 2022). It is also imperative to consider and ensure the security of the transmission path that the data follows (McVicker, 2022). * **Encryption in use:** Encryption safeguards data generated, edited, or otherwise classified as active use (McVicker, 2022). Protecting this data involves implementing data control and protection measures before and during its utilization (McVicker, 2022). Managing access rights and identities is crucial in minimizing risks associated with this data (McVicker, 2022). |
| **7** | **Triple-A Framework**  Authentication, authorization, and account (AAA) constitute a widely embraced security framework employed to regulate both network and application access (Mylonas, 2018). By enforcing stringent access and auditing policies, AAA intelligently manages access to computer resources, ensuring that only specific and legitimate users can interact with network and software application resources (Fortinet, n.d.).  The significance of AAA security extends across various methods of network access, historically setting a benchmark for security standards (Mylonas, 2018). While commonly associated with network protocols like RADIUS or Diameter, the AAA concept is broadly applied to enhance software application security (Fortinet, n.d.). This is particularly notable in domains such as Software as a Service (SaaS) products and microservice architectures(Fortinet, n.d.).   * **Authentication** involves verifying an individual’s identity through various methods such as static passwords, one-time passwords, certifications, and biometric credentials (GeeksforGeeks, 2021). These various forms of identification confirm that an individual is who they claim to be (GeeksforGeeks, 2021). * **Authorization** defines a user’s access rights and privileges, crucial to information and computer security (GeeksforGeeks, 2021). While authentication verifies an identity, authorization dictates the user’s permissible access, mitigating potential vulnerabilities by restricting interactions with sensitive data that may be unnecessary or defining the permissions granted during access (GeeksforGeeks, 2021). * **Accounting** involves tracking user activity within a system, noting timestamps, accessed resources, and data transfer information (GeeksforGeeks, 2021). This process is vital for forensic analysis and investigation when necessary (GeeksforGeeks, 2021). |
| **8** | **SQL Injection Coding Unit Testing**  The SQLInjection.cpp program fortifies itself against SQL injection by integrating regular expression patterns. These patterns identify and prevent SQL injection attempts by detecting keywords and operators such as ‘and,’ ‘or,’ and ‘=’ in queries. Upon detection, the program notifies the user and thwarts the injection by returning a “false” value. The enhanced code ensures the execution of only sanitized SQL queries, minimizing the risk of unauthorized access. Improved comments within the code enhance clarity about its functionality.  The C++ program industry demonstrates standard best practices, featuring explicit inline comments and apt naming conventions for enhanced readability. The developed C/C++ code illustrates a software design pattern approach.  The result is a robust validation of additional security for SQLite database, effectively safeguarding against potential SQL injection threats. The debugging summary provides a thorough and systematic overview, identifying specific bug types, detailing corrections, and offering screenshots of the application console output, compiler output, and the error list. |
| **9** | **Buffer Overflow Coding Unit Testing**  The Buffer Overflow code prevents overflow for all data types, adhering to C++ programming best practices with explicit comments and naming conventions. The project involves creating a new project, adding existing source files, and adjusting the user input buffer size to 21 characters. The comprehensive debugging summary covers specific bug types, accurately describes corrections, and provides relevant screenshots. The code focuses on preventing overflow by restricting user input using std::cin.getline, ensuring a maximum of 20 characters are read and stored in the buffer. The implementation demonstrates a straightforward software design pattern approach, including constant string initialization, user input handling, and console presentation. |
| **10** | Systematic debugging efforts help to address specific bugs in the code. The do\_even\_more\_custom\_applicaiton\_logic function was corrected to terminate after throwing a std::runtime\_error. A try-catch block was added in the do\_cutom\_applicaiton\_logic function to capture exceptions from do\_even\_more\_custome\_application\_logic, removing a redundant message.  The primary function now includes a try-catch block to handle the CustomeException thrown by do\_even\_more\_custom\_application\_logic. In the divide function, the error message for the std::invalid\_argument exception was improved for clarity. The do\_division function was corrected to catch the std::invalid\_argument exception thrown by divide.  To enhance exception handling, the catch blocks in the main function were reordered as per specifications, ensuring proper exception handling throughout the program. These corrections improved exception handling and provided helpful error messages during testing, debugging, and execution. |
| **11** | Debugging strands is a vital process in software engineering, encompassing the systematic identification and correction of errors, commonly referred to as bugs, within a software system (GeeksforGeeks, 2023). Its significance lies in preventing disruptions to software functionality that could result in suboptimal performance or inaccurate outcomes (GeeksforGeeks, 2023). Despite being labor-intensive, debugging is indispensable for ensuring the accurate operation of software systems (GeeksforGeeks, 2023).  Within the debugging process, code inspection is employed, manually examining source code to identify potential bugs or errors (GeeksforGeeks, 2023). Debugging tools, such as debuggers, trace, and profilers, efficiently identify and resolve bugs (GeeksforGeeks, 2023). Unit testing scrutinizes individual units or components to uncover bugs, while integration testing reveals interaction errors between components (GeeksforGeeks, 2023). System testing assesses the entire system for bugs, and monitoring observes software for unusual behavior (GeeksforGeeks, 2023). Logging and recording events and messages is a valuable tool in bug identification (GeeksforGeeks, 2023).  Debugging is an iterative process, often requiring multiple attempts to identify and resolve all bugs (GeeksforGeeks, 2023). Establishing a well-defined process for reporting and tracking bugs is crucial for effective bug management (GeeksforGeeks, 2023). Additionally, debugging is pivotal in software engineering, employing strategies like code inspection, debugging tools, unit testing, integration testing, system testing, monitoring, and logging in an iterative process to eliminate bugs from a software system (GeeksforGeeks, 2023). |
| **12** | **Automation Summary**   * **Collaboration** – Designing a collaborative code review tool is crucial for efficient communication and teamwork among developers, leveraging diverse skills to enhance code quality (CodeSee, n.d.). The tool effectively manages conflicts during the review process, leading to the adoption of optimal solutions and higher-quality, maintainable code (CodeSee, n.d.). Collaboration also facilitates learning from experiences and mistakes, promoting continuous improvement and professional growth among team members. * **Security** – A secure static code analysis tool is vital for maintaining software security, quality, and reliability (CodeSee, n.d.). These tools assess source code without execution, identifying potential vulnerabilities and coding violations (CodeSee, n.d.). Detecting security flaws in the development phase allows for timely issue resolution, reducing the risk of critical vulnerabilities in production and minimizing remediation costs (CodeSee, n.d.). Static analysis enforces consistent coding practices, resulting in cleaner, more maintainable code and improving developer’s skills and security awareness (CodeSee, n.d.). Some tools also incorporate behavioral analysis, adding an extra layer of security by identifying anomalous actions and enhancing the software’s resilience against attacks (CodeSee, n.d.). * **Integration** – A code review tool with robust integrations is essential for an efficient development workflow (CodeSee, n.d.). It should ideally be web-based or self-managed, offering flexibility to adapt to changing demands (CodeSee, n.d.). Robust integrations enable customization for diverse project requirements, seamlessly integrating with popular repositories like Git (CodeSee, n.d.). Integration ensures effective version control, change tracking, and smooth handling of branching and merging operations c.). Additionally, integration with deployment tools, especially in CI/CD pipelines, allows the tool to provide real-time feedback, helping developers catch and address issues early, and reducing the risk of deploying faulty code (CodeSee, n.d.). * **Metrics and Analytics** – A versatile code review tool is crucial for addressing diverse development needs, project requirements, and team preferences (CodeSee, n.d.). It can be customized to track relevant metrics, focusing on improving code quality (CodeSee, n.d.). Additionally, the tool should support various programming languages, frameworks, and libraries to cater to various projects and teams (CodeSee, n.d.). * **Visualization** – Effective code visualization is crucial in a code review tool, enhancing comprehension, efficiency, communication, and early issue identification (CodeSee, n.d.). Through visual representations like UML diagrams, flowcharts, call graphs, and code maps, reviewers can swiftly grasp code relationships, communicate feedback efficiently, and save time by concentrating on key areas during the review process (CodeSee, n.d.). |
| **13** | A DevOps pipeline streamlines collaboration between developers and operations professionals through automated processes and tools for code creation and deployment to a production environment (Hall, n.d.). While specifics can vary, standard components include build automation, continuous integration, automated testing, validation, and reporting (Hall, n.d.). Manual integrations allows human intervention in the DevOps pipeline (Hall, n.d.). A distinguishing characteristic is the emphasis on continuity, involving continuous integration, continuous delivery/deployment (CI/CD), continuous feedback, and continuous operations (Hall, n.d.). This approach ensures ongoing functionality rather than relying on one-off tests or scheduled deployments (Hall, n.d.).  DevOps tools such as:   * **Clang Clang** offers a language front-end and tooling infrastructure for C language family languages (C, C++, Objective C/C++, OpenCL, CUDA, and RenderScript)within the LLVM project (LLVM, n.d.). Clang includes both a GCC-compatible compiler driver (clang) and an MSVC-compatible compiler driver (clang-cl.exe) (LLVM, n.d.). * **Parasoft’s** C/C++ test is a fully integrated software testing solution   designed for embedded safety-critical industries, featuring automated  testing capabilities tailored for high-velocity Agile DevOps environments  (Parasoft, 2023). It tightly integrates with C and C++ IDE’s, CI/CD  pipeline, and containerized deployments, enabling early defect  detection and automatic enforcement of compliance with industry  standards (Parasoft, 2023).   * **CodeSonar** is a static code analysis solution designed to identify and comprehend quality and security defects within source code or binaries (CodeSecure, 2023). It simplifies the integration of Static Application Security Testing (SAST) into your development process by supporting over 100 compilers and compiler versions (CodeSecure, 2023). Additionally, it offers various integrations with popular development tools and IDEs, along with a comprehensive whole-program analysis that uncovers issues overlooked by other tools (CodeSecure, 2023). |
| **14** | **Risks and Benefits**  **Benefits**   * **Reduce Risk and Cost** – Secure coding and development offer a crucial advantage by mitigating risks and reducing the expenses linked to software vulnerabilities (Gray, 2023). In 2020, the average cost of a data breach, as per IBM, was $3.86 million, with an average of 289 days to identify and address vulnerabilities early in the software development lifecycle (SDLC), preventing their exploitation by attackers (Gray, 2023). This proactive approach saves time, money, and resources in addressing security incidents and avoids potential legal, regulatory, and reputational consequences (Gray, 2023). * **Improves Quality and Performance** – Secure coding and development not only enhance software security but also elevate the quality and performance of applications (Linkedin.com, 2023). Adhering to standards like OWASP Top 10, ISO/IEC 27034, and NIST SP800-53 ensures code consistency, readability, maintainability, and compliance with best practices (Linkedin.com, 2023). This approach facilitates more accessible code review, testing, debugging, and updates, minimizing errors and bugs (Linkedin.com, 2023). Furthermore, secure coding practices improve functionality and usability, aligning applications with security expectations and requirements (Linkedin.com, 2023). * **Gains Trust and Reputation** – Another advantage of secure coding and development lies in building trust and reputation in the competitive software market (Dorsett, 2023). Emphasizing the security of applications showcases a dedication to customers, partners, and regulators, fostering confidence and loyalty (Dorsett, 2023). This commitment distinguishes an organization from competitors who may not adhere to secure coding practices (Dorsett, 2023). Additionally, creating a secure and professional environment through secure coding practices can attract and retain talented developers who value these standards (Dorsett, 2023).   **Risks:**   * Delaying action in response to the 2013 Adobe data breach posed significant risks (BBC, 2013). A prolonged response time increases the potential exposure of sensitive customer information, escalating the likelihood of unauthorized access and data compromise (BBC, 2013). Such delays carried the risk of eroding user trust in Adobe’s ability to protect their data and could result in lasting reputational damage (BBC, 2013). Financially, a protracted response incurred higher costs associated with investigating the breach, implementing security enhancements, and potential legal consequences (BBC, 2013). Additionally, the prolonged vulnerability exposed Adobe to regulatory scrutiny and penalties for non-compliance with data protection regulations (BBC, 2013). A swift and decisive response was crucial to containing the breach’s impact, protecting affected users, and mitigating the multifaceted risks associated with the security incident (BBC, 2013). |
| **15** | To bolster software security, focus on regular code reviews and static analysis tools for early vulnerability detection. Cultivate a security-first mindset among developers, fostering a culture prioritizing proactive measures against evolving threats. Adherence to secure coding standards, encompassing input validation and authentication practices, offers a structured defense against potential attacks. Incorporating these practices into the development lifecycle enhances overall software security, reducing the risk of breaches. |
| **16** | Secure coding standards serve as vital guidelines for developers, aiming to mitigate security risks in software development through rules governing coding practices, data handling, input validation, and access controls. Key aspects involve validating inputs, implementing robust error handling, and ensuring secure authentication and authorization. Adherence to industry coding and naming conventions, avoidance of hardcoded passwords, utilization of encryption for sensitive data, and keeping dependencies up-to-date are highlighted practices. Developers should actively engage in regular code reviews and utilize static code analysis tools to detect and resolve security vulnerabilities early. It is crucial to adopt a security-first mindset, stay informed about evolving threats, and contribute to constructing resilient software that defends against potential attacks, adopting a defense-in-depth approach. |