**CS 405 Project Two Script Template**

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

|  |  |
| --- | --- |
| **Slide Number** | **Narrative** |
| **1** | |  |  | | --- | --- | | * ValidateInput Data | Input validation is essential to protecting applications. There are many methods of database injections that can take place. Certain characters should be ommitted to prevent such intrusions. | | * Heed Compiler Warnings | Compiler warnings can contain errors that can be known to contain security defects or other issues that could create larger problems within the application. Hackers can use these to bypass some metrics of security. You should always compile code with the highest warning level available. You should also minimize and remove them from your code. | | * Architect and Design for Security Policies | There is neccesity to design/architect your software to be able to enforce security policies. Modularizing permission is an example of this where dividing the permissions different parts of the system can promote better security. | | * Keep It Simple | Complicated designes in software can lead to many complications. There is increased risks of erros within the system the more complicated it is. Errors can arise from a variety of areas within the software | | * Default Deny | Permissions should be the main branch concerning access within the software. By default, access should be denied unless the proper permissions have been granted. | | * Adhere to the Principle of Least Privilege | A security best practice that requires limiting privileges to the minimum necessary to perform a job or task. This means that users, accounts, and processes should only have the access rights and permissions needed to complete their specific tasks, and no more. | | * Sanitize Data Sent to Other Systems | All data in transit should be protected. Attackers can use methods such as sql injectiong or console commands. similar to input validation but more apparent in complex systems where unused functionality within the software is utilized by hackers. | | * Practice Defense in Depth | Having multiple layers of defense built into a software can help thwart attackers attempts. If there is a breach at one level, the other, non-redudent levels to the application can help protect the software, and prevent the unauthorized access. | | * Use Effective Quality Assurance Techniques | Good quality assurance techniques can be  effective in identifying and eliminating vulnerabilities. by elimnating vulnerabilities we can manage the general risk better of the software. | | * Adopt a Secure Coding Standard | Providing a standard helps larger applications and software in development. Implmementing a standard gives measurable aspects to the security of the development. This can help with minimizing excessive time in development as well as providing the more security universally to the software and applications in development. | |
| **2** | Do not read uninitialized memory​  Do not define a c style variadic function​  Do not attempt to create a std::string from a null pointer​  Prevent sql injection​  Properly deallocate dynamically allocated resources​  Use static assertions to teste the value of a constant expression​  Handle all exceptions​  Guarentee that storage for strings has sufficient space for character data and the null terminator​  Close files when they are no longer needed​  Do not use std::rand() for generating pseudorandom numbers​ |
| **3** | ranked in order of importance, prioritize more servre before lower |
| **4** | Encryption at rest: Data that is stored on devices or accessed remotely needs to be protected. That is why data at rest on servers/databases must always be encrypted if it can be accessed. Various tools exist as well as storage device encryption to ensure the integrity of the data in question.​  Encryption in flight: While in transit data can be vulnerable. Properly managing the requests and the inputs from users can help deter chances of corruption. Internal measures such as encrypting the data and properly handling can be used, as well as exterior measures such as firewall and authentication to protect remote accessing.​  Encryption in use: Whenever new data is created it is important to properly format and store this information. This involves hashing or encrypting the information and properly storing it. Managing user access to the program can limit ill-intent to manipulate new data within a system and properly storing can make it adequately protected along with the rest of the information ​  ​ |
| **5** | Authentication : is necessary to confirm identity of a user/admin within the system. There are usually keys or passwords required to authenticate within a system. MFA is another tool that can greatly improve the accendibility of those authorized within the system. Ensuring that users have been authenticated greatly improves the overall security of data that these authenticated individuals have access to.​  Authorization: give access rights to a system that otherwise average individuals would not be entitled to. By limiting access to crucial components within a system it allows for more vulnerable or sensitive information to be protected behind these right. This is one of the reasons why default denying privileges is a common practice within the industry and allows confidential data to have an additional level of security within the application protecting it. ​  Accounting: is necessary to implement to handle transactions within the system. Accurately tracking incoming and outgoing requests as well as system permissions can help confirm the accountability within a system. This also allows the system administrators to track user activity when necessary.​  ​ |
| **6** | Unit testing examples with Gunit testing |
| **7** | Automation is crucial in enforcing and defining these security standards within pre-production as well as production. ​  We can implement many of the practices in the entirety of coding practices throughout development. During designing and building, good coding practices and proper routing/handling can provide solid frameworks for activity within an application.​  We can verify and test results of these via assertion and health checking. From there, monitoring the systems and maintaining becomes a core responsibility of the team. ​ |
| **8** | Astree: Fast and precise analyses​   runtime error​   non-interference analysis​   automatically generated report files​  Parasoft C/C++ test: integrated tools for testing c and c++ code (VS studio/eclipse plugin)​   supports software development​   static/dynamic code analysis​   runtime error testing​  RuleChecker: Can be used to analyze handwritten or generated source code with complex memory usage​    supports MISRA, SEI CERT, and ISO/IEC standards​ |
| **9** | Pros ​   Early implementation allows for easier integration with minimized costs​   Proactive defense strategies are flexible and constantly improving​   Overall defense stronger​  ​  Cons​  Initial costs increase​  More expertise to properly implement​  Adds complexity​ |
| **10** | Follow up on discoveries with tools and testing​  Security as a foundation​  Proactively implementing proper unit testing​  Expectation of addressing to discrepancies |
| **11** | Validate Input Data ​  Heed Compiler Warnings ​  ⦁ Architect and Design for Security Policies ​  ⦁ Keep It Simple ​  ⦁ Default Deny ​  ⦁ Adhere to the Principle of Least Privilege ​  ⦁ Sanitize Data Sent to Other Systems ​  ⦁ Practice Defense in Depth ​  ⦁ Use Effective Quality Assurance Techniques ​  ⦁ Adopt a Secure Coding Standard |
| **12** | <https://www.absint.com/rulechecker/index.htm>​  ​  <https://www.parasoft.com/products/parasoft-c-ctest/>​  ​  <https://www.absint.com/astree/index.htm> |