# Victor Feight

10/17/2021

# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Victor Feight and this is my Security Policy Presentation where I will be explaining the outlined Security Policy Guide Template for Green Pace. |
| **2** | On premises keep servers under lock and key with only priveleged users able to access them. In the Supplier cloud, ensure privacy level agreements and service level agreements are in place. Without our own private cloud, establish a baseline of deny-by-default configurations and defense in depth through external pentests and tried and true encryption mechanisms, as well as enabling Firewall and priveleged account management (PAM), and Monitoring for unauthorized changes and access through logging. Vulnerability scans, patch management techniques, and updates of vulnerable apis can be integrated into the CI/CD pipeline and automated. By protecting our host security, we protect the critical assets of the system and data that attackers seek. Lastly, training of the team on the layered defense compliance protocols is critical. |
| **3** | Validate Input Data (STD 001, 002, 003, 004)  Heed Compiler Warnings (STD 002, 003, 010)  Architect for Security Policies (STD 002, 004, 005, 006, 007, 008, 009, 010)  Keep it Simple (ALL)  Default Deny (STD 004)  Adhere to Principle of Least Privilege (STD 004, 009, 010)  Sanitize Data Sent to Other Systems (STD 003, 004)  Practice Defense in Depth (STD 004)  Use Effective Quality Assurance Techniques (STD 001, 002, 003, 005, 006, 007, 008, 010)  Adopt a Secure Coding Standard (STD 005, 006, 007, 008, 009, 010) |
| **4** | Data Value  Data Type  String Correctness  SQL Injection Prevention  Memory Protection  Assertions  Exceptions  API Vulnerability Assessment  Secure Object Oriented Programming  Expressions |
| **5** | IN REST: We shall encrypt data at rest using full-disk encryption at the server level, as well as database encryption using MySQL Server, and provide a backup strategy.  AT FLIGHT: Use up to date, secure libraries, use Public Key infrastructure for end-to-end protection on message bodies or attachments, use Managed File Transfer or SSH with expiration date on the link, password access, utilize Data leak prevention mechanisms built into cloud service  IN USE: Utilize identity management mechanisms to confirm user roles and identity, allow conditional access to the tools functionality based on the user roles and other parameters. Use IRM digital rights management, to apply persistent protection to documentation. |
| **6** | Authentication: Process of identifying a user, using valid credentials of user and password. Control how a user is authenticated using a secured local database or external AWS server, prefer to use tried and trusted protocol.  Authorization: After the user has been authenticated, authorization shall be used to determine which resources and functionality the user is allowed to access and which operations can be performed.  Accounting: Monitor and log any user events while they are logging in/out or utilizing the resources, as well as user uptime or any other configured parameter. |
| **7** | We are testing for out of bounds vulnerabilities (STD-001 Data Value, STD-005 Memory Protection)  Google Test helps us write independent and repeatable tests and thus it is the main testing framework we shall use.  As per Google Test Primer, the following guideline will be followed:  “ASSERT\_\* versions generate fatal failures when they fail, and abort the current function.  Test 1) First, verify adding 5 values to the collection. |
| **8** | Test 2 --- ASSERT\_TRUE(collection->max\_size() >= 10); These tests allow me to assert that max size is greater or equal to certain values. |
| **9** | Test 3 -- ASSERT\_TRUE(collection->size() > initialSize); By asserting whether the size of the container is greater than the inital size, we can verify the resize has increased the collection. Vice versa for testing whether a resize has decreased the collection. |
| **10** | Test 4 -- ASSERT\_TRUE(collection->size() == 0); Checks that the resize has actually decreased the collection size to 0. |
| **11** | Test 5 -- ASSERT\_TRUE(collection->size() == 0); verify the collection has been actually resized to 0 after clear, very similarly to resize to 0.  ASSERT\_TRUE(collection->size() == 0); likewise works for verifying the collection has been erased from begin() to end(). |
| **12** | Automation can be created upon Build, by automating manual processes such as compilation and static code checking into a CI/CD pipeline, encourages team workflow, can use Docker for container instances, GitLab for versioning, Jenkins for CI.    Verify and test in SecOps by automating virtualized container deployment. Implement automated security tests and regression tests in QA.  Monitor and detection. Automate static application security tests into nightly builds on key sections of code. Embed dynamic application security testing into SDLC to look for vulnerabilities in real time.  Utilize tools such as OWASP Dependency-Check to check code dependency vulnerabilities.  Notable Tools: ClangTidy, Cppchecker, Parasoft, Coverity, Jenkins, Gitlab, Docker |
| **13** | The Systems Sciences Institute at IBM reported that it cost 6x more to fix a bug found during implementation than to fix one identified during design. Furthermore, according to IBM, the cost to fix bugs found during the testing phase could be 15x more than the cost of fixing those found during design (Cost to fix bugs)  Secure Coding by Design: In Pre-production, One can automate a selection of Unit tests (DESIGN STAGE) as well as static analysis tools (VERIFY AND TEST) in the CI/CD pipeline (BUILD) thereby providing secure-fail/pass criteria on builds  By documenting possible or known threats (by focusing on data flow) we can address them, rank them, prioritize assessment of mitigation (ASSESS AND PLAN).  In production, things become more expensive to fix. We practice defense in depth with configuration of security settings, monitor logs, respond to and block attacks, and maintain a secure and stable baseline that we can return to if compromised. |
| **14** | Elaborate on Pentesting methods and how they can identify and mitigate cyber-security risks in malware and APIs we use every day.  Elaborate on aspects of Cloud Security and approaches of “defense in depth” at each layer of security  Security Layers include: Critical Assets + Data, App Security, Endpoint Security, Host Security, network security, perimeter security, cloud security, and physical security.  Adopt a set of tools/options/mechanisms for Cloud Security  Identify a set of optimal (least privilege) security controls for network and host security.  Read the latest security journals in the field. |