**Roadmap To Success**

**(Sections 8, 9, 12)**

**Section 8: Understand how to use version control for all elements of the software delivery lifecycle.**

* **Understand the principles of version control** 
  + The principles of version control create a positive workflow when it comes to making changes, team collaboration, and maintaining a high-quality codebase. Some of the principles are, atomic changes, commit often, branching and merging, backup and recovery to name a few. This ensures that changes to the code is documented and can be referenced back to if needed, any team member that makes changes to the source code doesn’t overwrite another member’s work. Following the principles can not only help you track changes, but to know when and where to deploy upgrades to the software when needed.
    - An example of this would be Tripwire Enterprise for me; It allows you to monitor any changes that has been made to files, tasks, nodes, policies or rules.
    - Within some of the files, tasks or rules are code or scripts that is required for the job to run successfully.
    - Any changes to these can be critical
* **Demonstrate proper usage of versioning, releases, issues, and merges in source code**
  + Versioning: Helps tracks different stages of your project.
  + Releases: Marks specific points in the project history
  + Issues: Documents and track feature requests or bugs
  + Merges: Integrate changes back into the main codebase
* **Demonstrate usage of dependencies and libraries**
  + Dependency: Any external source that your project relies on to function.
  + Libraries: Collection of reusable code that simplify common tasks**.**
    - **Within Tripwire Enterprise, you can find a library of tasks to the organization’s standards**
    - **Here straightforward tasks can be created and scheduled to run on a periodic timeframe**
    - **Allowing the reuse of code or scripts**

**Section 9: Understand and demonstrate how to build and test quality code, at scale.**

* **Understand how to build quality code at scale ensuring quality**
  + Using version control to ensure high quality code to test and build would be the ideal approach. Along with tools like SonarQube, ESLink, Jenkins to ensure code quality.
  + Some best practices can be to establish coding standards, code reviews, documentation, continuous deployment, or static code analysis**.**
* **Understand how to write test cases**
  + Test cases are meant to make sure your software works as expected and free of bugs.
  + The outline of a test case includes but not limited to
    - Understanding the requirements
    - Defining the scope
    - Being clear and concise
    - Defining expected results
    - Reviewing and Refinement
* **Understand the different types of testing, where to apply them, and the relative amounts**
  + Unit Testing – applied to the smallest testable parts of an application and the relative amount is high.
  + Integration Testing - applied to interactions between integrated components or systems and the relative amount is moderate
  + System Testing - applied to the entire system and the relative amount is moderate
  + Acceptance Testing - applied to the entire system involving end-users and stakeholders and the relative amount is low
  + Performance Testing - applied to the system to evaluate performance metrics and the relative amount is low to moderate
  + Security testing - applied to the system and the relative amount is low
  + Usability testing - applied to user interface and overall user experience and the relative amount is low
  + Regression Testing - applied to entire system after changes or updates and the relative amount is moderate to high
  + Exploratory testing - applied to any part of the system and the relative amount is low
* **Understand the successful use of test coverage tools**
  + In order to use a coverage tool you have to pick the right one for your project.
    - Some coverage tools are identified as JaCoCo: java projects
    - Coverage.py: Python projects
    - Istanbul: JavaScript projects
    - Cobertura: Java and Groovy projects
  + Then you would want to integrate with your CI/CD pipeline, set coverage goals, run test regularly, analyze coverage report, and automate and monitor.
* **Understand the successful use of performance, limits, and chaotic testing**
  + Performance testing is used to assess the speed, responsiveness and stability of a system under various conditions.
  + Limits testing ensures the system operates within specified limits and handles boundary conditions correctly.
  + Chaotic testing tests the system resilience by intentionally introducing failures and disruptions.

**Section 12: Support testing and debugging; participate in integration and deployment**

* For this section there were a few projects I was apart of that supported testing and debugging, also with integration and deployment.
  + One was configuring new servers for our CyberArk product, where I had to copy the appropriate files that housed the CyberArk application to the selected server.
  + Afterwards the next step was to connect to the primary CyberArk server after the files have been copied and the application has successfully been installed on the server.
  + However, on some of the servers because they required a different role within the scope of CyberArk server deployment couldn’t establish a connection.
  + After some troubleshooting I found that the selected servers that couldn’t establish a connection had firewalls rules blocking the connection and that the server wasn’t responding to the primary server over the specified port number.
  + Once that issue was resolved the servers were ready for production use.
* I also had to do prep work or create the necessary steps for my team members to implement during the deployment of applications onto the servers.
* Another project was the CI/CD of Splunk dashboards to monitor our applications. Adding them to the dashboard was done by inputting some code text on the platform to create a user interface to be able to see the changes within the applications.