**Bruce Gaudet**

**CS-405**

**8/18/25**

**8-2 Journal**

Looking back on this course, I now understand that secure coding cannot be an afterthought. One of the most important lessons I have learned is that security must be integrated from the very beginning of the development cycle. If a team waits until the end of a project to apply security measures, it creates risk and often leads to expensive fixes. Instead, adopting a secure coding standard from the start ensures that software is built with protection already in place. This practice reduces vulnerabilities and reinforces the importance of secure development as part of the foundation, not as a last minute patch. Throughout the course, we studied how early use of standards such as input validation, proper access control, and memory safe functions can stop many common threats. These proactive techniques align with what professionals refer to as defense in depth, and they help reduce technical debt over time.

When considering the balance between risk and mitigation, one must evaluate both the cost and the benefit of the security being applied. Adding protection to every feature is not always efficient or realistic. This is where a structured risk assessment is helpful. We reviewed tools and frameworks that help developers prioritize their efforts based on the likelihood and severity of threats. For example, high-risk areas such as login systems or data storage should receive strong protection like encryption, two factor authentication, and auditing. On the other hand, a lower risk feature, such as a basic content viewer, may only need minimal safeguards. These decisions come down to understanding the attack surface and allocating resources where they matter most. Good developers must be able to explain these choices and document them so teams can work together with clarity and confidence.

The zero trust model also made a strong impression on me. It is based on the idea that no user or system should be trusted by default, even if they are already inside the network. This concept challenges the old belief that once someone has passed the firewall, they are safe. With zero trust, every action and request must be verified. This mindset pushes developers to write code that expects malicious input at all times, not just from the outside world. It also encourages features like session expiration, reauthentication prompts, and limited scope for user roles. While some may see this as extra work or user friction, it actually strengthens the integrity of applications and limits the damage that can occur if one part of the system is compromised.

Lastly, policy plays a central role in guiding how secure systems are designed and enforced. Throughout the course, I have learned that clear, written security policies give teams a shared understanding of what is acceptable and expected. These policies cover everything from password complexity to secure data handling and user access levels. Without them, development teams may follow different practices or overlook important details. A policy also serves as a point of reference during audits and reviews. By applying these standards in my future projects, I will be better prepared to contribute to secure environments that protect both users and the business.

This course has reshaped how I think about software development. Security is not just a checklist but a mindset that must be present in every decision and line of code. I plan to carry these lessons forward in my work and advocate for stronger, earlier, and more thoughtful security in every project I touch.