**Reflection on the Secure Software Development Module**

*GitHub e-Portfolio URL: [url]*

**WHAT**

When I first enrolled in the Secure Software Development module, I felt genuinely excited. The word “secure” instantly stood out. In my earlier software development studies, security was barely touched. This module felt like a gap-closing opportunity that could push my skills into a more professional direction.

I started the secure Command Line Interface (CLI) application challenge with confidence and ambition. I had planned several advanced features beyond the Unit 6 design document, including secure admin tools and deeper attack simulations. Just as development began, I transitioned into a new role at work as Artificial Intelligence (AI) Team Lead. I found myself managing meetings, researching models, and mentoring juniors while still trying to honor my academic goals.

I felt torn. On one hand, I was proud of the career growth. On the other, I was frustrated that I couldn’t implement all I had envisioned. There were moments I doubted whether I could manage both roles well. Despite the time constraints, I stayed focused on building a stable, secure CLI system. The most satisfying parts were testing, encryption, and clean code structure. Before this, I had used plain .env or .json files without much thought for security. After learning about bcrypt and the cryptography package (Romano and Krüger, 2021), I started rethinking my approach.

I implemented Fernet-based encryption for secure storage and bcrypt for password hashing (Python Package Index, no date-a). Excited by what I had learned, I went back to earlier work scripts and upgraded them with encryption and secure editing via a small Graphical User Interface (GUI). These changes gave me a sense of applying learning directly to real use.

Another breakthrough came with code quality. Inspired by module guidance and the use of flake8, I began running my files through linter checks (Python Package Index, no date-b). I improved naming, eliminated unused variables, organized code into smaller, maintainable modules, and aligned everything with Python Enhancement Proposal (PEP) 8 (van Rossum, Warsaw and Coghlan, 2001). This significantly improved readability and helped me build structured unit tests. I started to see code cleanliness not just as a stylistic detail but as a reflection of disciplined, secure practice.

Unit 9 introduced modular testing using pytest (Python Package Index, no date-c). Previously, I would run a script manually and visually confirm it worked. This project helped me develop pytest modules, including one for simulating brute-force attacks using hacker.py. Tests were isolated and structured, aided by the clean, modular layout I had adopted. For the first time, I saw how testing could serve as both a quality gate and a learning tool.

**SO WHAT**

Balancing this module with full-time work was not easy. I struggled most with attending live seminars, as many clashed with important client meetings or reviews. I often had to rely on recordings and missed the chance to ask questions live. This felt isolating at times, and I had to stay self-motivated.

Despite the time pressure, the Unit 6 team design phase was a great experience. Our team was collaborative and flexible. If someone couldn't attend a meeting, we would simply reschedule. I contributed not only by organizing meetings but also by shaping key parts of the design. I shared ideas on CLI structure, user roles, and Open Worldwide Application Security Project (OWASP) threat mitigation. Collaborating with others exposed me to tools like Mermaid for class diagrams (Mermaid, no date), which I hadn’t used before. It made documenting class relationships much clearer.

The design phase also deepened my understanding of security risks. I saw how OWASP threats like A03 (injection) and A10 (insufficient logging and monitoring) could be addressed by implementing login attempt logging and using Click decorators for input sanitization (OWASP Foundation, 2021; Python Package Index, no date-d). These insights were applied directly in the final build.

The biggest change for me came in how I viewed testing and code quality. I had always assumed clean code was mostly about aesthetics. Now, I understand it’s a foundation for reliability and maintainability. Linter checks helped reinforce good habits, and I’ve started using flake8 and pytest by default in all new scripts. I even introduced these practices to my team, where we now rely on individual test modules instead of testing the entire system at once.

I was particularly proud when a team member adopted my testing template and said, "I’ve never seen testing made this simple before." In that moment, I realized I wasn’t just ticking off an academic requirement; I was actively reshaping how we work as a team. It was the first time I felt like my MSc studies were feeding directly into my leadership.

**NOW WHAT**

This module has transformed how I approach development. I now treat testing, encryption, and code structure as defaults, not extras. Configuration files will always be encrypted. Scripts will follow a secure, modular structure with test coverage and formatting checks built in.

For future projects and modules, I plan to engage more with peers. I hesitated a bit this time when it came to messaging teammates. Everyone was kind and open, but I found myself holding back. Next time, I want to reach out earlier and be more collaborative from the start. Engaging more would not only improve project quality but also enhance the reflective learning process.

This module has also made me feel more secure about my MSc journey. Initially, I was nervous about the security modules. I worried whether I would meet expectations while adjusting to a demanding new job. But the application of these concepts in both academic and work settings has been energizing. I now look forward to future modules with genuine interest, especially those related to security.

Unit 10’s content on faceted data and secure information flows has also sparked new ideas. I want to explore how Python decorators could be used to enforce data access layers, ensuring users only see what they’re permitted to. This aligns directly with what I’m working on in my current role, and I see opportunities to build these ideas into real-world systems.

**Conclusion**

This module showed me that secure software development isn’t about ticking boxes. It’s a mindset. Through testing, encryption, clean code, and collaborative design, I’ve adopted new habits that are now embedded in both my academic work and professional practices.

This was more than just a learning unit. It was a turning point in how I write software and how I help others do the same. I started this module with curiosity. I leave it with clarity, conviction, and real confidence in what I’ve gained.

**References**

Click (no date) *Click – Python Package for Creating Command Line Interfaces*. Python Package Index. Available at: <https://pypi.org/project/click/> (Accessed: 18 July 2025).

Flake8 (no date) *Your Tool For Style Guide Enforcement*. Python Package Index. Available at: <https://pypi.org/project/flake8/> (Accessed: 18 July 2025).

Mermaid (no date) *Markdown-inspired diagramming and charting tool*. Available at: <https://mermaid.js.org/> (Accessed: 18 July 2025).

OWASP Foundation (2021) *OWASP Top Ten: The Ten Most Critical Web Application Security Risks*. Available at: <https://owasp.org/Top10> (Accessed: 18 July 2025).

Python Package Index (no date-a) *bcrypt*. Available at: <https://pypi.org/project/bcrypt/> (Accessed: 18 July 2025).

Python Package Index (no date-b) *flake8*. Available at: <https://pypi.org/project/flake8/> (Accessed: 18 July 2025).

Python Package Index (no date-c) *pytest*. Available at: <https://pypi.org/project/pytest/> (Accessed: 18 July 2025).

Python Package Index (no date-d) *Click*. Available at: <https://pypi.org/project/click/> (Accessed: 18 July 2025).

Romano, F. and Krüger, H. (2021) *Learn Python Programming*. 4th edn. Packt Publishing.

van Rossum, G., Warsaw, B. and Coghlan, N. (2001) *PEP 8 – Style Guide for Python Code*. Python Software Foundation. Available at: <https://peps.python.org/pep-0008/> (Accessed: 18 July 2025).