**Reflection on the Secure Software Development Module**

*GitHub e-Portfolio URL: [*[*E-Portfolio Link*](https://github.com/mobeen-ali/MSc_SDD_PCOM7E_ePortfolio/tree/main)*]*

**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 approached the Command Line Interface (CLI) application challenge with confidence and ambition. My plans extended well beyond the Unit 6 design document. I intended to develop secure admin tools, integrate security-state tracking within the CLI (context-awareness), and simulate attacks. However, just as development began, I transitioned into a new role at work as Artificial Intelligence (AI) Team Lead. Managing meetings, researching models, and mentoring juniors alongside academic work became a real test of endurance.

Despite my enthusiasm, my initial Unit 11 submission received a grade of just 41%. This was a moment of deep disappointment, shame, and guilt. I realised I had underestimated the expectations of the module and submitted subpar work. It was a harsh but necessary wake-up call. I had taken the module too lightly and it showed in the results.

Rather than becoming discouraged, I chose to take the feedback as a learning opportunity. I carefully reviewed the tutor's comments and systematically rebuilt my implementation. The revised system included enhanced modularity, cleaner structure, and expanded security features aligned with real-world needs. I upgraded the implementation from partial to full compliance with the Open Worldwide Application Security Project (OWASP) Top Ten. Initially, only A03:2021 (Injection), A05:2021 (Security Misconfiguration), A07:2021 (Identification and Authentication Failures), and A10:2021 (Insufficient Logging and Monitoring) were implemented. The revised version addressed all categories from A01 through A10. Each was selected for its relevance to secure software design and its usefulness in strengthening system defences.

This module also reshaped how I approached encryption and authentication. Previously, I had used .env or Javascript Object Notation (JSON) files to store credentials, without much thought for security exposure. Learning about bcrypt and the cryptography package (Romano and Krüger, 2021) led me to redesign my approach. I implemented Fernet encryption for data at rest and used bcrypt for password hashing. I also integrated two-factor authentication (2FA) using the Python One-Time Password (PyOTP) library to defend against brute-force and replay attacks.

Linter tools such as flake8 helped me realise that code clarity is not just stylistic—it is a critical part of building reliable systems. I restructured the project into modular units and followed the Python Enhancement Proposal 8 (PEP 8) standard (van Rossum, Warsaw and Coghlan, 2001). This improved testability, readability, and long-term maintainability. I also began developing reusable templates to support test-driven development from the early stages of implementation.

Before this module, I had tested systems manually by running scripts and visually confirming outputs. Through the structured development process, I learned to create modular pytest suites, including one for simulating brute-force attacks via hacker.py. The clean layout and modular architecture made it easier to isolate components and enforce consistent behaviour. This helped me realise that testing is not just a final check, but a critical tool for learning, design validation, and long-term code reliability.

An important shift has been the way I now approach testing in my day-to-day work. Previously, I saw testing as something to complete at the end, often informally. This module introduced me to more structured tools and practices, including flake8 for code style, pytest for modular testing, and bandit for static security checks. I haven’t fully integrated these into all my work projects yet, but I’ve started using flake8 more regularly and experimenting with basic pytest setups when time allows. Running bandit on a few internal scripts also helped identify hardcoded secrets I had previously overlooked. These early steps are helping me bring more structure and security awareness into my own workflow and gradually into my team’s practices as well.

**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.

This experience of structured remediation and systematically reviewing tutor feedback, re-evaluating implementation decisions, and mapping behavior against expected outcomes, improved how I approach complex builds and increased my confidence in aligning design with delivery.

**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 increased my interest in pursuing the MSc, especially since my upcoming modules are also related to security. 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.

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.

Finally, organizing the portfolio taught me the value of structured documentation. I created submission checklists, artefact indexes, and navigation guides to ensure clarity. These habits now extend to my leadership work, where I must document project updates and communicate designs to a mixed team of developers and stakeholders.

**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**

OWASP Foundation (2021) *OWASP Top Ten: The Ten Most Critical Web Application Security Risks*. Available at: <https://owasp.org/Top10> (Accessed: 4 June 2025).

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

Schmitz, T., Probst, F., Ulbrich, S. and Holzmann, C. (2016) ‘Preventing data leakage with faceted data’, *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, pp. 65–75. Available at: <https://doi.org/10.1145/2933446.2933450> (Accessed: 14 July 2025).

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: 14 July 2025).