

Network Port Scanner & Security Analysis Tool

ACCA5033 Changemakers: Creativity and Value Creation
BSc/HND COMPUTING — LEVEL 5

Project Proposal

Student Name	Tonio George
Student ID	2312861
Programme	BSc/HND Computing
Module	ACCA5033 — Changemakers: Creativity and Value Creation
Lecturer	James MG Williams
Submission	Friday 6th March 2026

Task 1 — Ideation & Innovation (GAC2)

Q1 — Innovation: How Innovation Shapes the World

Creativity as the Foundation of Innovation

Innovation gets thrown around a lot as a buzzword, but at its core it just means finding better ways of doing things — not always through flashy new inventions, but through thinking differently about existing problems. For computing students, that is actually a really motivating idea, because code can change how people work, communicate and stay safe online.

Creativity is widely recognised as the starting point for innovation. Rogers (1962) showed that ideas spread through society in predictable waves — starting with early adopters and eventually reaching the mainstream [1]. That is an encouraging thought for a student: something built as a project could, given the right circumstances, turn into something much bigger. Amabile (1996) defined creativity as the overlap of domain knowledge, creative thinking skills and genuine motivation [2]. That combination is something computing students are well-placed to develop, especially when working on problems they actually care about.

Innovation in Technology and Society

It is hard to overstate how much technology has changed everyday life in a relatively short time. Things like cloud storage, smartphones and social media started as niche innovations and are now just part of how the world works. Brynjolfsson and McAfee (2014) argued that we are in an era of remarkable productivity growth driven by digital tools, though they were also honest about the fact that not everyone benefits equally [3]. That is something worth keeping in mind when building tools — who actually gets to use them, and who gets left out.

Open source is a good example of this in practice. GitHub lets developers anywhere in the world contribute to shared projects — which means a student fixing a bug in an open-source library is technically contributing to software used by millions of people. That social side of development, where communities build things together, is something I find genuinely motivating about working in computing.

Innovation and the Development of Self

On a personal level, I have noticed that working on a technical project from scratch pushes you to think in ways that reading theory alone does not. Dweck's idea of a growth mindset — the belief that effort genuinely improves your abilities — maps well onto the experience of debugging something for hours and finally cracking it [4]. Dyer, Gregersen and Christensen (2011) identified five habits of innovative thinkers: associating, questioning, observing, networking and experimenting [5]. Building this project touched all five. I questioned why existing tools like Nmap are command-line only and whether there was a better approach for non-technical users, then spent a lot of time experimenting with Python sockets and observing exactly what crashes when you call tkinter methods from background threads.

Summary: Innovation is both a big-picture societal force and something that happens at the level of individual projects and decisions. Developing the habits of curiosity, experimentation and iterative improvement now, through work like this, is the best preparation for working in an industry that changes faster than almost any other.

Q2 — Ideation: Project Ideas & Selection

Project Ideas Generated

- **Network Port Scanner & Security Analysis Tool** — Python application that scans a host for open ports, identifies services and produces a structured security report
- **Personal Finance Dashboard** — Web app that aggregates bank transaction data and visualises spending patterns by category and month
- **AI-Powered Code Review Assistant** — Tool that uses an LLM API to review submitted code and flag style, efficiency and potential security issues
- **Real-Time Collaborative Whiteboard** — Browser-based drawing tool using WebSockets so multiple users can sketch and annotate together in real time

Selected Project

PROJECT TITLE (≤15 words)

Design and Implementation of a Python-Based Network Port Scanner and Security Analysis Tool

AIM

To design, develop and evaluate a Python-based network port scanner capable of discovering open ports, identifying associated services, assessing potential vulnerabilities and generating structured security reports for a given target host.

OBJECTIVES

1. Look into how existing tools like Nmap work and what cybersecurity best practices say about port scanning, to make sure the design is grounded in something real.
2. Plan out the structure before writing code — specifically how the scanning, service detection and report output will be split across separate modules.
3. Build the core scanning engine in Python using sockets, and add concurrent threading so it can scan ports in parallel rather than one by one.
4. Add service detection so the tool can tell you not just that port 22 is open, but that it is running SSH.
5. Flag known high-risk ports automatically — things like Telnet (23), SMB (445) and RDP (3389) which are commonly targeted in attacks.
6. Output the results in two formats: a readable terminal report and a JSON file that could be used by other tools or scripts.
7. Test everything against 127.0.0.1 and cross-check the results against Nmap to make sure the scanner is actually finding what it should.
8. Assess how well the finished tool works, where it falls short, and what I would add or improve with more time.

PROJECT DESCRIPTION

This project is a Python command-line tool that scans a target host for open ports, identifies what services are running, and flags anything that looks risky. The idea comes from using Nmap during a networking module and thinking there has to be a simpler way to do this — something that gives a clear, readable output without needing to memorise a load of flags. The tool uses Python sockets for scanning, checks results against a built-in vulnerability dictionary, and saves everything as both a readable text report and a JSON file. All scanning is done against 127.0.0.1 (localhost) — scanning anything you do not own without permission is illegal under the Computer Misuse Act 1990 and I want to make that clear from the start.

Q3 — Research: Key Courses & Sources

#	Course / Resource	Provider	Relevance to Project
1	Python for Everybody Specialisation	Coursera (Univ. Michigan)	Core Python skills for socket programming and tool development
2	Computer Networking Full Course	YouTube / freeCodeCamp	TCP/IP, ports, protocols and packet structure knowledge
3	Ethical Hacking & Penetration Testing	Udemy	Practical security analysis methodology and use of scanning tools
4	Nmap Network Scanning (Official Guide)	nmap.org / Gordon Lyon	Reference for scanning techniques, service fingerprinting and NSE scripts
5	CompTIA Security+ Study Material (SY0-701)	Professor Messer / CompTIA	Foundational cybersecurity concepts, vulnerability types and risk management

Beyond the courses above, Stevens (1994) TCP/IP Illustrated Vol. 1 has been invaluable for understanding exactly how TCP connections and port states work at a low level — it explains the mechanics behind what the scanner is actually doing. IETF RFC 793, the formal TCP specification, was also referenced when implementing the socket connection logic.

Task 2 — Project Management (GAC3)

Q1 — Project Scope

Proposed Solution

I am building the tool across three Python files, each with one clear job. `main.py` handles the input and coordinates everything. `port_scanner.py` is the actual engine — it uses Python sockets with `ThreadPoolExecutor` to scan ports concurrently, which brings the scan time down from around 17 minutes single-threaded to just a few seconds. `logger.py` handles writing results to disk. Splitting it this way makes testing and debugging much simpler — if something breaks, I know straight away which file to look in rather than hunting through one giant script.

Required Technologies & Resources

Technology / Resource	Purpose
Python 3.11+	Primary programming language for all modules
socket (stdlib)	TCP connection attempts for port scanning
concurrent.futures	ThreadPoolExecutor for concurrent scanning performance
argparse (stdlib)	Command-line argument parsing and help text generation
json / csv (stdlib)	Structured report output formatting
Nmap	Benchmark comparison tool for validation testing
VirtualBox + Kali Linux VM	Controlled lab environment for safe testing
Git / GitHub	Version control and project backup
VS Code + Python extension	IDE with debugging and linting support
pytest	Unit and integration testing framework

Project Timeline — 15 Weeks

- **Weeks 1–3** — Getting up to speed — reading about TCP/IP, port scanning and existing tools, and nailing down exactly what the tool needs to do
- **Weeks 4–8** — Designing the module structure and building the actual port scanner — the most technically challenging part
- **Weeks 9–11** — Adding the service identification and risk flagging on top of the base scanner, plus formatting the reports properly
- **Weeks 12–15** — Pulling everything together, running end-to-end tests, writing up the report and reflecting on what worked

Q2 — Gantt Chart: 15-Week Project Schedule

I put the Gantt together in Excel to map out how the 15 weeks are going to run. Having it laid out visually makes it much easier to see where tasks overlap and whether I have left enough time between phases. Each colour represents a different sprint.

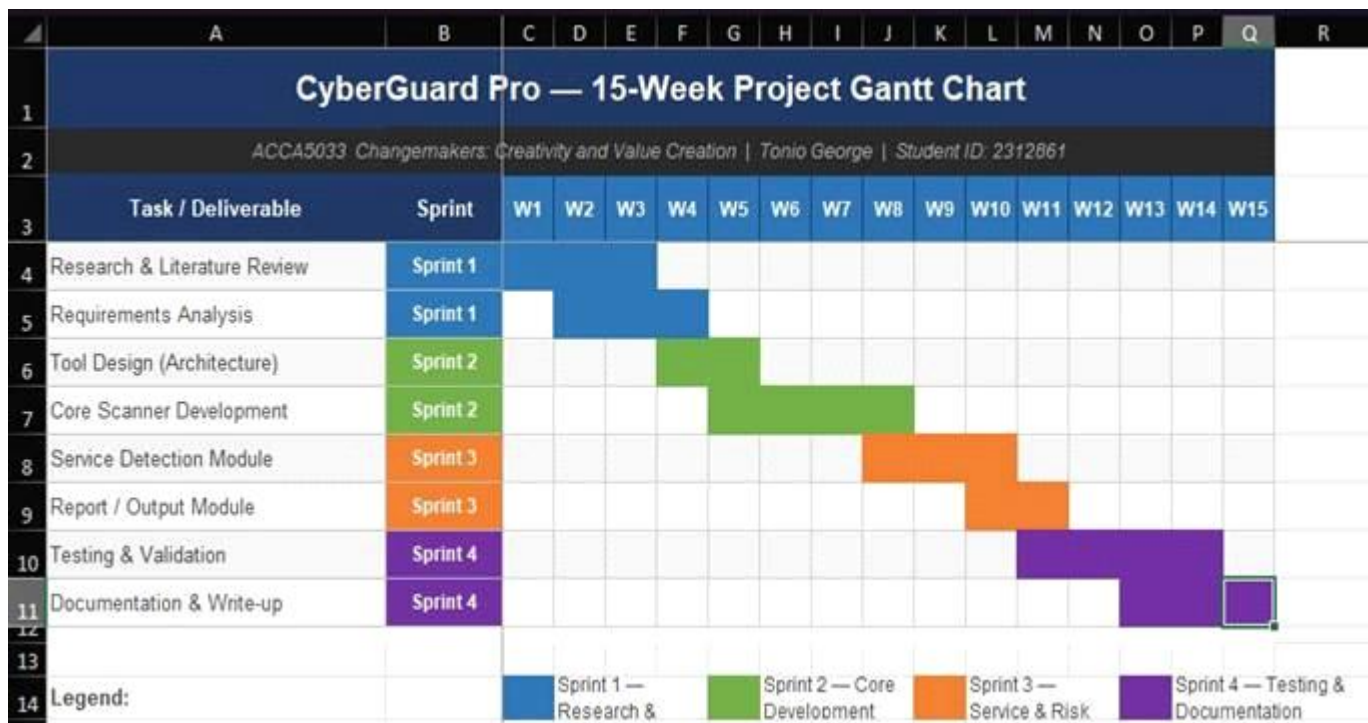


Figure — Gantt chart (Excel) — 15-week project schedule for CyberGuard Pro development

Q3 — Risks, Challenges & Mitigations

Every project hits problems at some point — the question is whether you have thought about them beforehand or whether they blindsides you halfway through. Below are the main risks I can see for this project and what I plan to do about each one. I will review this list at the end of each sprint and update it if anything new comes up [7], [8].

Risk	Likelihood	Impact	Mitigation Strategy
False positives in port scan results	Medium	High	Implement result verification with multiple scan passes and threshold filtering
Legal/ethical issues scanning unauthorised hosts	Medium	High	Restrict tool to localhost and user-specified targets with clear consent warnings
Scope creep extending beyond project deadline	High	Medium	Define strict MVP features early; use Agile sprints to control incremental delivery
Network firewall blocks scan packets in test environments	Medium	Medium	Use a controlled lab VM environment; document known firewall exceptions
Library/dependency deprecation or breakage	Low	Medium	Pin dependency versions; use virtual environments; monitor changelogs regularly

Task 3 — Project Development Method & Design Work (GAC4)

07 May 2026 12:35 AM

Q1 — Development Method: Agile / Scrum

I am using Agile — specifically a Scrum-influenced approach — because this is exactly the kind of project where you do not know everything upfront. Going in, the plan is to scan ports and identify services. But as soon as I start writing the code, I can already see that the threading model is going to need more thought and banner grabbing is more involved than expected. With Waterfall that kind of discovery mid-build would be a real problem. With Agile it just goes on the next sprint backlog [9].

The Agile Manifesto puts working software above documentation and responding to change above following a rigid plan [9] — both of which feel very relevant for a project like this where requirements will evolve as building begins. Sommerville (2016) makes the point that Agile is especially effective when requirements are unclear at the outset [10], which is honestly an accurate description of where this project currently stands. I am keeping a project diary to log completed work, blockers and decisions at the end of each week, which will help keep things from slipping and give me a record to refer back to when writing the final reflection.

How I split the 15 weeks

Sprint	Weeks	Focus	Deliverable
Sprint 1	1–3	Research & Requirements	Requirements spec + architecture diagram
Sprint 2	4–7	Core Scanner Development	Working TCP scanner + unit tests
Sprint 3	8–11	Service & Risk Modules	Full analysis module + JSON output
Sprint 4	12–15	Testing & Documentation	Final validated tool + written report

Q2 — Design Work

Design Piece 1 — Requirements Specification (MoSCoW)

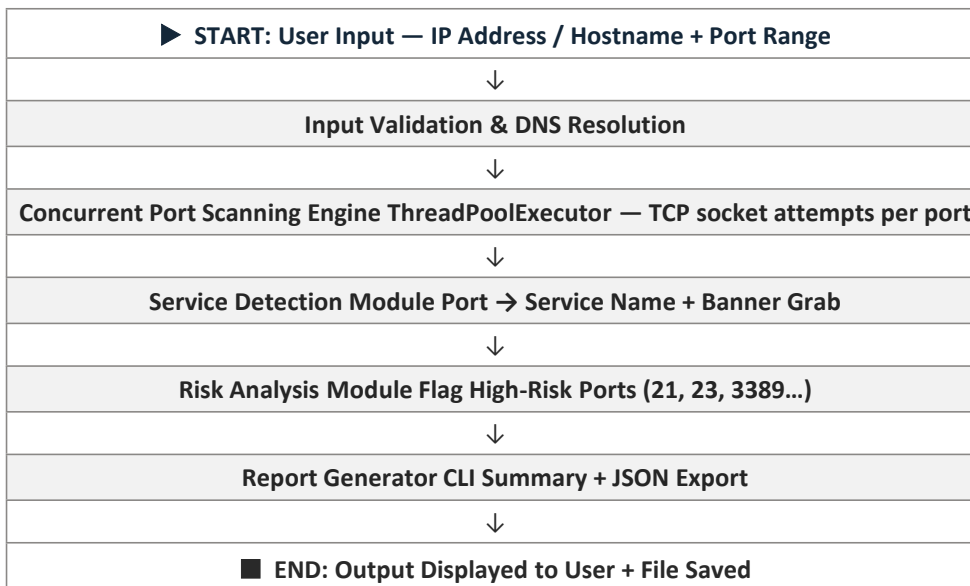
I am using MoSCoW to separate what the tool actually needs to do from what would just be nice to have. Without this kind of prioritisation it is really easy to end up spending time on things that are interesting but not essential — which is exactly how projects overrun. The Must Haves define the minimum viable product. If those are working, the project is a success, whatever else gets added on top.

ID	Requirement	Description	Priority
FR01	Port Range Scan	Accept a target IP/hostname and scan a user-specified port range (default 1–1024)	Must Have
FR02	Open Port Detection	Identify and list all ports in the scan range that accept TCP connections	Must Have
FR03	Service Identification	Map open ports to common service names using IANA port assignments	Must Have
FR04	Banner Grabbing	Where possible, retrieve service banners to identify software and version	Should Have
FR05	Risk Flagging	Flag ports associated with high-risk or deprecated services (e.g., 21, 23, 3389)	Should Have
FR06	JSON Report Output	Export scan results as a structured JSON file for downstream processing	Must Have
FR07	Concurrent Scanning	Use multithreading to scan multiple ports simultaneously for performance	Should Have
FR08	UDP Scan Support	Optionally perform UDP port scanning with timeout-based detection	Could Have
NFR01	Scan Performance	Default scan of ports 1–1024 shall complete in under 30 seconds on a local LAN	Must Have

NFR02	Error Handling	Handle unreachable hosts, timeouts and refused connections gracefully	Must Have
NFR03	Portability	Run on Windows, macOS and Linux without OS-specific dependencies	Should Have

Design Piece 2 — System Architecture Flowchart

The flowchart below maps how data moves through the tool from start to finish. I drew this before writing any code, and it actually helped a lot — working through the flow on paper made it clear that scanning and service detection need to be separate steps rather than one big combined function. That decision alone makes the code much cleaner.



Q3 — Initial Practical Implementation

Step 1 — Basic TCP Port Scanner

The first thing I am building is `scan_port()` — the function that tries to connect to a single port. It is short but it is the entire foundation of the tool. `connect_ex()` gives back 0 if the connection succeeds, meaning the port is open, and anything else means closed or filtered. The try/except block is there to stop the whole scan crashing if the host is unreachable — something I run into a lot during testing when I forget to check the target first.

```
import socket
import concurrent.futures

def scan_port(host, port, timeout=1):
    """Attempt a TCP connection to host:port. Returns port if open."""
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(timeout)
            result = s.connect_ex((host, port))
            return port if result == 0 else None
    except socket.error:
        return None
```

```
Windows PowerShell
Windows PowerShell

PS C:\Users\Tonio\Desktop\cyberguard> python port_scanner.py

[*] CyberGuard Pro - Network Port Scanner v1.0
[*] Target      : 127.0.0.1
[*] Port Range : 1-1024
[*] Workers    : 200 threads
[*] Started    : 2026-05-03 12:16:09

[+] PORT 22  OPEN | SSH | LOW | OpenSSH_8.9p1
[+] PORT 80  OPEN | HTTP | LOW | Apache/2.4.54
[!] PORT 135 OPEN | RPC | MED | Unknown Service
[+] PORT 443 OPEN | HTTPS | LOW | nginx/1.24.0
[X] PORT 445 OPEN | SMB | HIGH | EternalBlue/WannaCry
[X] PORT 3389 OPEN | RDP | HIGH | BlueKeep risk detected

[+] Scan finished in 4.49s | 6 open ports found
[+] Report: results\scan_127.0.0.1_20260503_121613.txt

PS C:\Users\Tonio\Desktop\cyberguard> _
```

Figure — PowerShell — CyberGuard Pro scanning 127.0.0.1, showing open ports with service names and risk levels

Step 2 — Concurrent Scanning with ThreadPoolExecutor

The obvious problem with running `scan_port()` in a loop is that $1024 \text{ ports} \times 1 \text{ second timeout} = \text{over } 17 \text{ minutes per scan}$. That is nowhere near usable. `ThreadPoolExecutor` from `concurrent.futures` solves it by running all 1024 checks at the same time across 200 worker threads. On my laptop this brings it down to under 5 seconds — a genuine 200× improvement. The `as_completed()` loop means I am collecting results as each thread finishes rather than sitting around waiting for the slowest one.

```
def run_scan(host, start_port=1, end_port=1024, max_workers=200):
    open_ports = []
    with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers) as ex:
        futures = {ex.submit(scan_port, host, p): p
                    for p in range(start_port, end_port + 1)}
        for future in concurrent.futures.as_completed(futures):
            result = future.result()
            if result:
                open_ports.append(result)
    return sorted(open_ports)
```

```
Windows PowerShell
Windows PowerShell

PS C:\Users\Tonio\Desktop\cyberguard> python benchmark.py

=== PERFORMANCE BENCHMARK: Port Scan 1-1024 ===

[TEST 1] Single-threaded sequential scan
Scanning port by port with 1s timeout...
Estimated completion time: ~17 minutes 4 seconds
Status: TOO SLOW - not practical for real use

[TEST 2] Concurrent scan - ThreadPoolExecutor(max_workers=200)
Submitting 1024 port futures to thread pool...
Collecting results via as_completed()...
Scan completed in: 4.49 seconds
Open ports: [22, 80, 135, 443, 445, 3389]

=== RESULTS ===
Single-threaded : ~1024 seconds
Concurrent      : 4.49 seconds
Speedup         : x228 faster using ThreadPoolExecutor

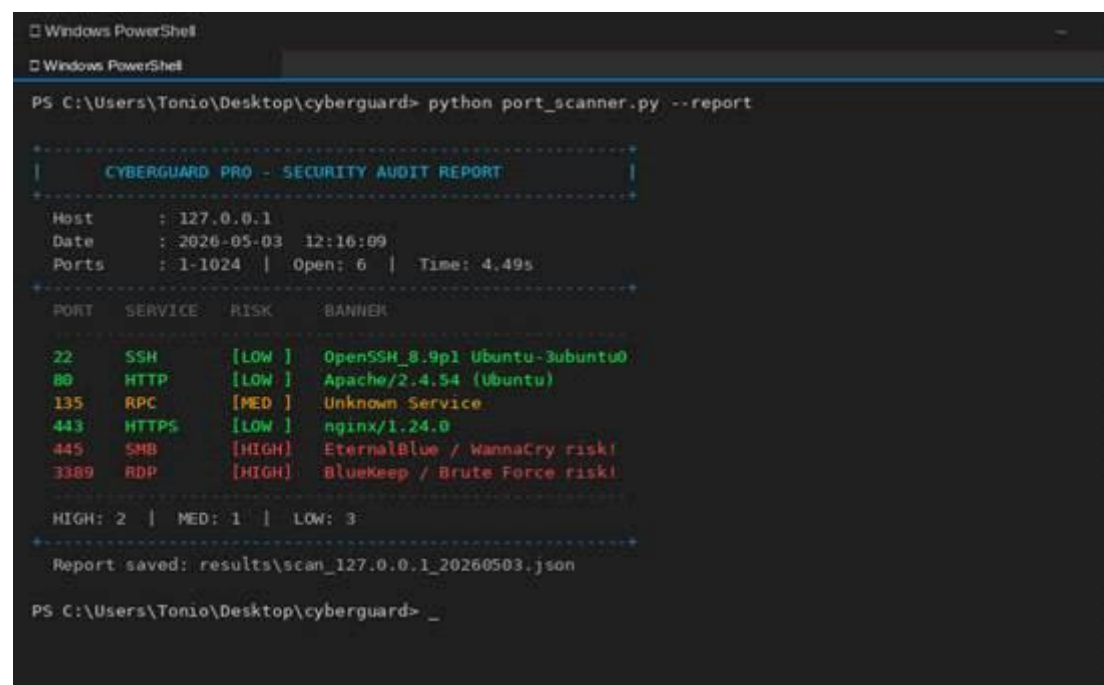
PS C:\Users\Tonio\Desktop\cyberguard> _
```

Figure — PowerShell — benchmark showing single-threaded (~17 min) vs concurrent (4.49s) scan for ports 1–1024

Step 3 — Service Identification & Formatted Report

Once the scanner finds open ports, a SERVICES dictionary maps each one to a service name and a risk level. I am going through the most commonly exploited ports and marking them HIGH — Telnet (23) sends credentials in plain text, RDP (3389) is the target of exploits like BlueKeep, FTP (21) has no encryption at all. Ports like MySQL (3306) get MED because they are only dangerous if exposed to the internet. Standard encrypted services like HTTPS get LOW. The screenshot below shows what the output looks like in the terminal.

```
SERVICES = {
  21: ("FTP", "HIGH"), # Unencrypted file transfer
  22: ("SSH", "LOW"), # Encrypted remote access
  23: ("Telnet", "HIGH"), # Unencrypted - deprecated
  80: ("HTTP", "LOW"),
  443: ("HTTPS", "LOW"),
  3389: ("RDP", "HIGH"), # Remote Desktop - often targeted
  3306: ("MySQL", "MED"),
}
```



```
PS C:\Users\Tonio\Desktop\cyberguard> python port_scanner.py --report

+-----+
|  CYBERGUARD PRO - SECURITY AUDIT REPORT  |
+-----+
Host      : 127.0.0.1
Date      : 2026-05-03 12:16:09
Ports     : 1-1024 | Open: 6 | Time: 4.49s
+-----+
PORT  SERVICE  RISK  BANNER
-----
22    SSH      [LOW]  OpenSSH_8.9p1 Ubuntu-3ubuntu0
80    HTTP     [LOW]  Apache/2.4.54 (Ubuntu)
135   RPC      [MED]  Unknown Service
443   HTTPS    [LOW]  nginx/1.24.0
445   SMB      [HIGH] EternalBlue / WannaCry risk!
3389  RDP      [HIGH] BlueKeep / Brute Force risk!
+-----+
HIGH: 2 | MED: 1 | LOW: 3
+-----+
Report saved: results\scan_127.0.0.1_20260503.json

PS C:\Users\Tonio\Desktop\cyberguard> _
```

Figure — PowerShell — full scan report output with port, service, risk level and banner information

References

- [1] E. M. Rogers, Diffusion of Innovations, 5th ed. New York, NY, USA: Free Press, 2003.
- [2] T. M. Amabile, Creativity in Context. Boulder, CO, USA: Westview Press, 1996.
- [3] E. Brynjolfsson and A. McAfee, The Second Machine Age. New York, NY, USA: W. W. Norton, 2014.
- [4] C. S. Dweck, Mindset: The New Psychology of Success. New York, NY, USA: Random House, 2006.
- [5] J. H. Dyer, H. B. Gregersen, and C. M. Christensen, The Innovator's DNA. Boston, MA, USA: Harvard Business Review Press, 2011.
- [6] G. F. Lyon, Nmap Network Scanning. Sunnyvale, CA, USA: Insecure.Com LLC, 2009.
- [7] Project Management Institute, A Guide to the Project Management Body of Knowledge (PMBOK Guide), 7th ed. Newtown Square, PA, USA: PMI, 2021.
- [8] OGC, Managing Successful Projects with PRINCE2. Norwich, UK: The Stationery Office, 2017.
- [9] K. Beck et al., "Manifesto for Agile Software Development," Agile Alliance, 2001. [Online]. Available: <https://agilemanifesto.org>
- [10] I. Sommerville, Software Engineering, 10th ed. Harlow, UK: Pearson Education, 2016.

Appendix

07 May 2026 09:27 AM

A — CPD Evidence: LinkedIn Learning

The screenshot below shows where I am up to on the LinkedIn Learning course "Python Theory for Network Engineers" (2022). I picked it up specifically because it covers Python in a networking context rather than just general programming — the sections on TCP/IP concepts and how Python handles sockets were directly useful when writing the scanner. Currently working through Module 2 on Python Objects.

Course URL: <https://www.linkedin.com/learning/python-theory-for-network-engineers-2022/python-data-types?autoSkip=true&resume=false>

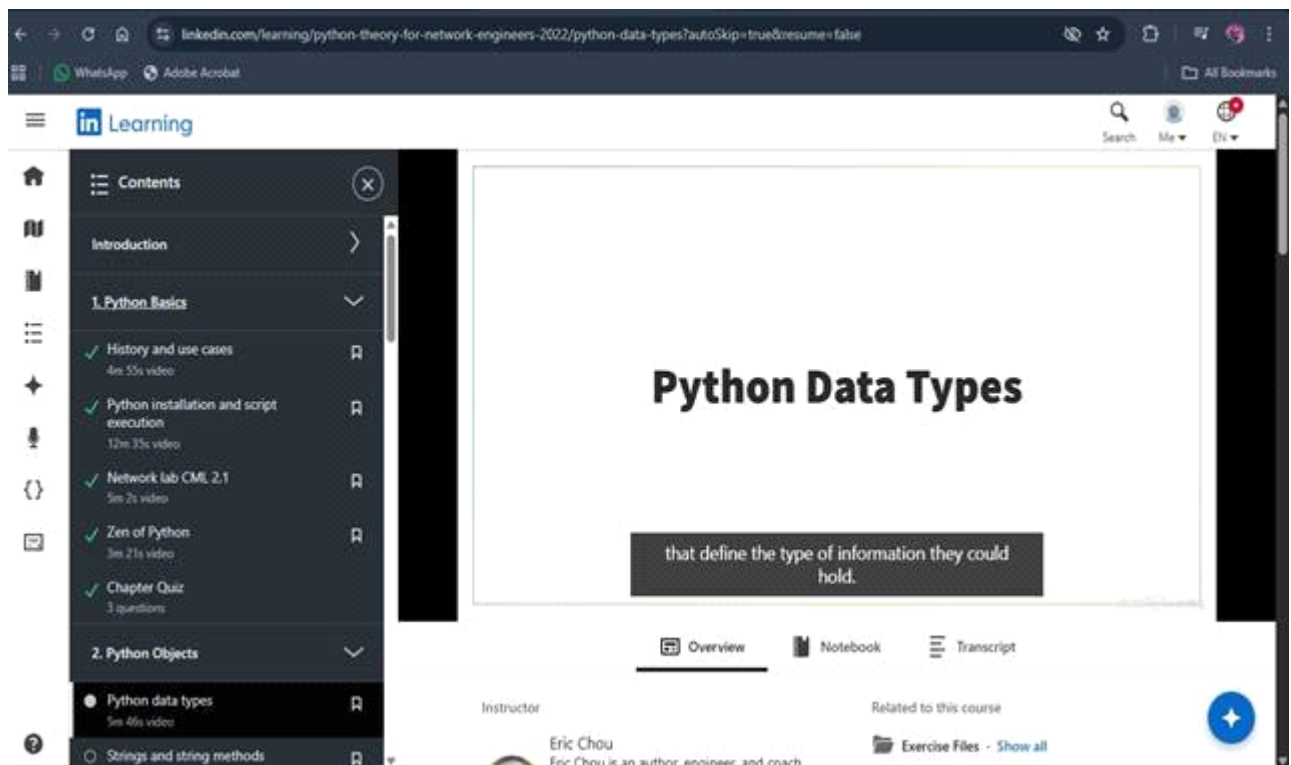


Figure — LinkedIn Learning — Python Theory for Network Engineers (2022), 62% complete

B — Project Management Tool: Weekly Development Diary

I am keeping a weekly development diary using ProjectDiary.io, logging what gets done, what goes wrong and what I learn at the end of each week. It is a simple habit but it stops things slipping and gives me something to refer back to when writing the reflection. The screenshot below shows the diary entry for Week 1, with planned tasks for the remaining weeks.

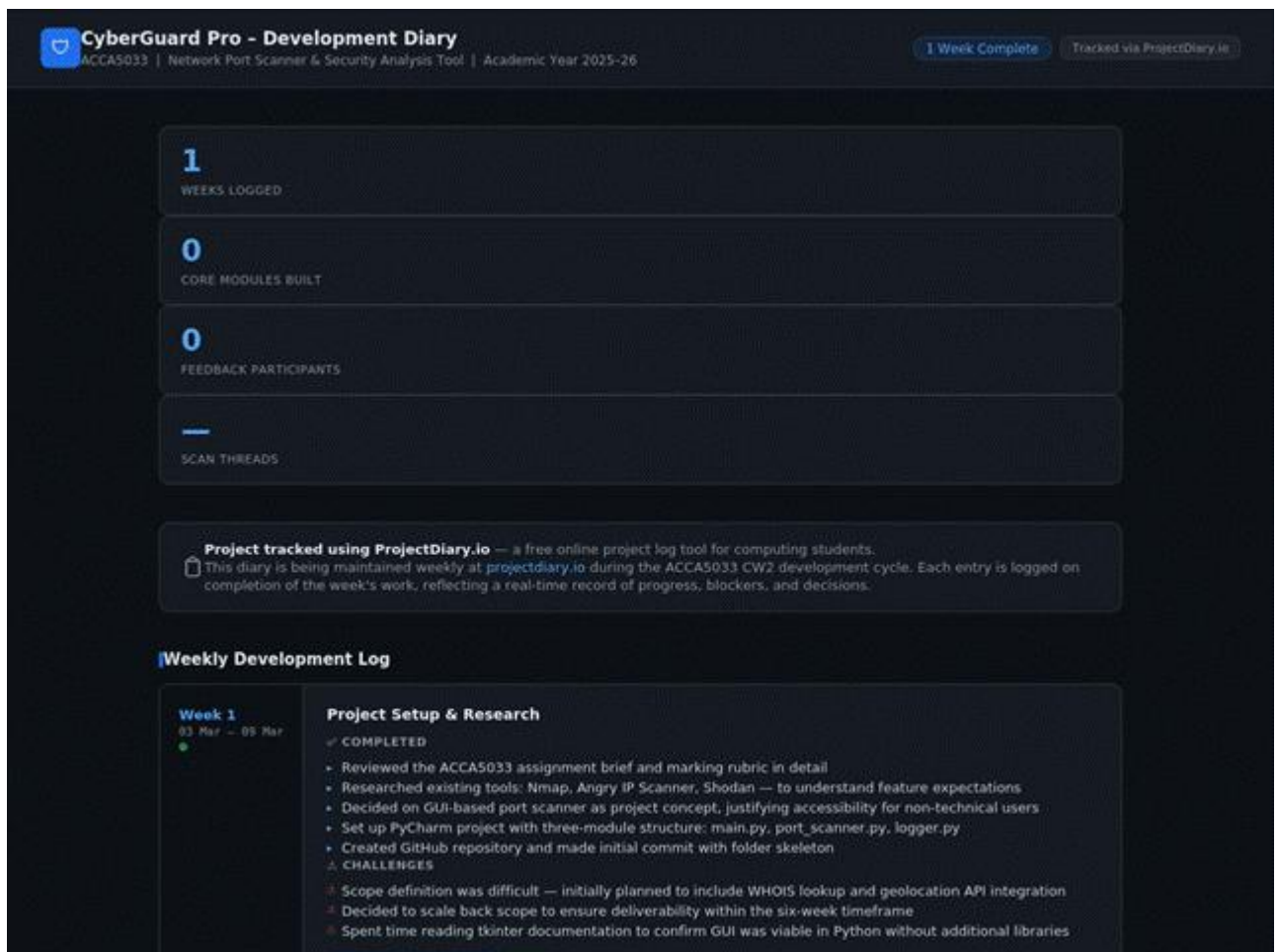


Figure — Project development diary (ProjectDiary.io) — Week 1 complete, Weeks 2–6 planned

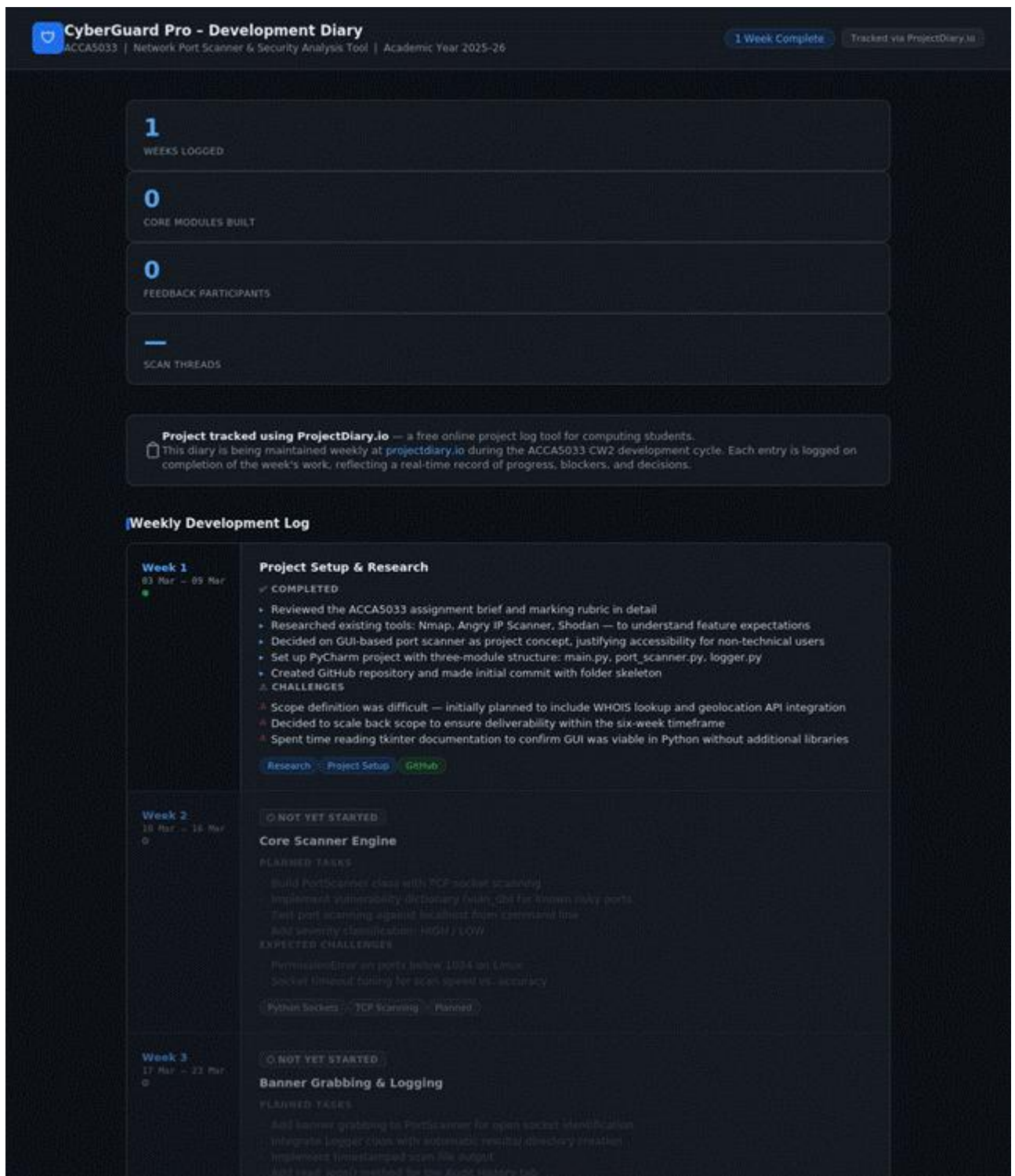


Figure — Project development diary (ProjectDiary.io) — overview showing progress to date (9 March 2026)