International Journal of Applied Science and Technology Application

Vol. 1, No. 2, September 2026, pp. 101-109
e-ISSN: 3124-9132
DOI: https://doi.org/10.62712/ijapset.v1i2.13

Integrating STRIDE Threat Modeling into the Prototyping Lifecycle of a Vocational Student Showcase Platform

Donny Sanjaya[1*], Indri Sulistianingsih[2], Rizka Salsabila Riadi[3], Ridha Maya Faza Lubis[4]

[1,2]Department of Software Engineering Technology, Department of Computer Engineering and Informatics, Politeknik Negeri Medan, Medan, Indonesia
[3]Department of Multimedia Graphic Engineering Technology, Department of Computer Engineering and Informatics, Politeknik Negeri Medan, Medan, Indonesia
[4]Department of Business and Management, Southern Taiwan University of Science and Technology, Tainan, Taiwan
Author's Email: [1*]d.sanjaya@polmed.ac.id, [2]indrisulistianingsih@polmed.ac.id, [3]rizkasalsabila@polmed.ac.id, [4]db01g208@office.stust.edu.tw

Article Info

Article history:

Received: July 30, 2026
Accepted: September 15, 2026
Published: September 30, 2026

Keywords: *Prototyping; Role-based access control; Secure by design; STRIDE; Threat modeling*

ABSTRACT

Student project showcase platforms in vocational higher education are usually built by small teams through rapid prototyping, and security is often deferred until after deployment. This study integrated STRIDE threat modeling into the prototyping lifecycle of the showcase platform of the D4 Software Engineering Technology program at Politeknik Negeri Medan, a CodeIgniter 4 and MySQL application with role-based access control that held 237 student accounts and 213 project records. Two activities were inserted between quick design and prototype construction: data flow diagram decomposition with STRIDE elicitation, and the derivation of security requirements prioritized by CVSS v3.1 base scores and mapped to the OWASP Top 10:2025 categories. The process produced 17 threats across all six STRIDE categories and 16 security requirements, and every threat was traced to at least one requirement and one allocated control. Three threats were rated Critical and six High, concentrated in authentication, injection, file upload, and access control. Twelve of the 16 requirements were met by built-in framework or platform mechanisms, while the remaining requirements concerned ownership, state publication, data minimization, and auditing. The resulting model offers a reproducible pattern for small academic platforms with high public visibility and frequent developer turnover.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Corresponding Author:

Donny Sanjaya
Department of Software Engineering Technology, Politeknik Negeri Medan
Medan, Indonesia
Email: d.sanjaya@polmed.ac.id

1. INTRODUCTION

Vocational higher education institutions increasingly publish student work through web-based showcase platforms that display projects to employers, partner industries, and the public. The showcase platform of the D4 Software Engineering Technology (TRPL) program at Politeknik Negeri Medan follows this pattern. It is reachable from the open internet, hosts project records and media from successive student cohorts, and stores personal data and authentication credentials behind an administrative panel. Platforms of this kind sit inside a sector that faces a broad range of threat actors while operating under limited security budgets and staffing [1],

and the risk carried by individual institutions has been framed as a national policy concern rather than a purely technical one [2].

Most published security work on academic and public sector web systems examines platforms after deployment. Indonesian studies have applied vulnerability assessment and penetration testing to institutional websites and reported the weaknesses found by scanners and manual tests [3, 4]. International studies have orchestrated multiple scanners to widen detection coverage [5], compared penetration testing tools [6], and characterized the parameters and testing approaches that determine web application security [7, 8]. Injection and cross-site scripting remain the most studied attack classes [9-12], and broken access control has been surveyed as a separate and persistent problem [13]. These studies detect weaknesses in code that already exists, so every finding implies rework after release.

Secure software engineering research argues for moving security activities toward the early phases of development. A systematic mapping study catalogued the security approaches available across the software lifecycle [14], a review of DevSecOps adoption identified challenges related to people, practices, tools, and infrastructure [15], and a practitioner survey examined which security activities agile teams actually perform [16]. Threat modeling is the principal design-phase technique in this body of work, and STRIDE is among the most widely used schemes for eliciting threats from a data flow diagram [17, 18]. Recent work has extended threat modeling to complex systems that combine several threat sources [19]. The evidence, however, comes largely from industrial, critical infrastructure, and enterprise settings. Small academic platforms that are built by lecturers and students through iterative prototyping are rarely documented.

Prototyping fits such platforms because requirements emerge from repeated feedback by lecturers, students, and administrators [20]. The same iterative character makes security easy to postpone. Student-written code in courses without an explicit security focus contains recurring vulnerabilities [21], and cybersecurity curricula call for applied activities that place security inside development work [22]. No published process shows how threat modeling can be embedded in the prototyping iterations of a vocational showcase platform and how the resulting threats can be traced to concrete controls in a mainstream PHP framework. This gap motivates the present study.

The study makes three contributions. First, it defines a secure prototyping process that inserts STRIDE threat modeling and the derivation of prioritized security requirements into a conventional prototyping cycle. Second, it applies the process to a production platform built with CodeIgniter 4, Bootstrap 5, and MySQL under role-based access control, and reports the resulting threat catalog, security requirements, and their traceability to implementation controls. Third, it analyzes the effort implied by the controls and discusses the implications for software engineering practice in vocational education. Section 2 describes the research method, Section 3 presents and discusses the results, and Section 4 concludes the paper.

2. RESEARCH METHOD

2.1. Research design

The study followed a design-oriented single case study. The unit of analysis was the development process of the TRPL showcase platform, and the artifacts produced by that process formed the data. These artifacts comprised the requirements list, the use case diagram, the entity relationship diagram, the Unified Modeling Language (UML) models, the framework configuration, and the administrative dashboard of the running system. The results reported here are design-level outcomes, namely the threat catalog, the security requirements, and their allocation to controls. The study did not report measurements from penetration testing of the live platform.

2.2. Case platform

The platform was developed with the CodeIgniter 4 framework following the model, view, and controller pattern. Bootstrap 5 provided the user interface, MySQL stored the application data, and role-based access control governed the administrative functions [23]. At the time of analysis, the administrative dashboard reported 237 student accounts, 213 project records, four active privileged user accounts, and four system roles, including a super administrator role. Each project record carried a publication state of Draft or Published and belonged to a category such as web application, mobile application, or geographic information system. Anonymous visitors could browse published projects without authentication, while students and administrators signed in to manage their content.

2.3. Secure prototyping process

Figure 1 shows the process applied in the study. Stages 1, 2, 5, 6, and 7 follow a conventional prototyping cycle in which requirements are gathered, a quick design is produced, a prototype is built and evaluated with stakeholders, and the cycle repeats until the prototype is accepted [20]. Stages 3 and 4 were inserted between the quick design and the construction of each prototype so that security decisions were made before code was written.

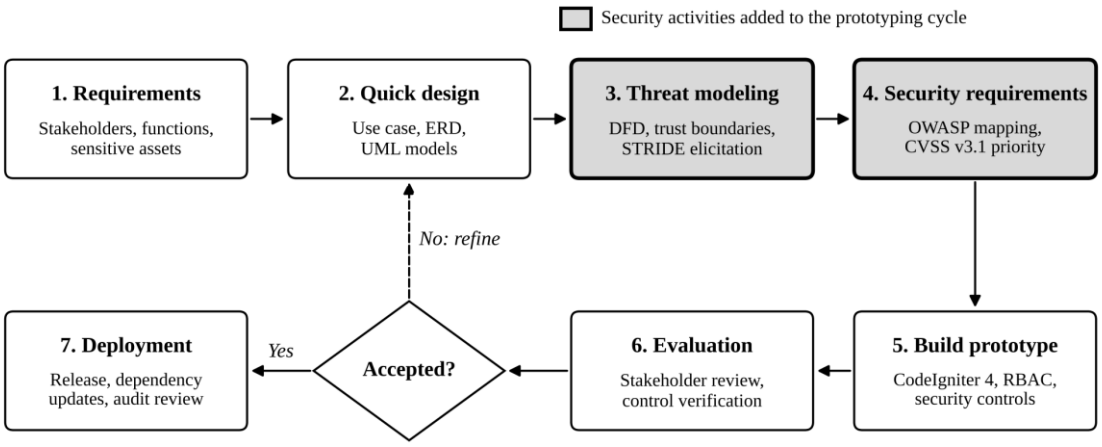


Figure 1. Secure prototyping process with two inserted security activities.

In Stage 3, the design models were translated into a data flow diagram that separated external entities, processes, data stores, data flows, and trust boundaries. STRIDE was then applied per element [17, 18]. Table 1 lists the threat categories examined for each element type. A candidate threat was recorded when a category produced a plausible violation given the role of the element and the assets it handled. Candidates that shared one mechanism were merged, and candidates without a realistic path in the platform were discarded.

Table 1. STRIDE categories examined the data flow diagram element type.

Element type	STRIDE categories examined	Elements in the platform
External entity	S, R	Public visitor, student, administrator
Process	S, T, R, I, D, E	P1 public showcase, P2 authentication and RBAC filter, P3 project management, P4 administrative management, P5 upload handler
Data store	T, R, I, D	D1 media files, D2 project and student records, D3 users and roles, D4 audit log
Data flow	T, I, D	Flows that cross trust boundaries TB1 and TB2

In Stage 4, each threat was mapped to one primary OWASP Top 10:2025 category, selected by its principal security consequence. Each threat then received a CVSS v3.1 base vector that described its typical exploitation conditions in the platform, and the base score was computed with the equations of the specification. The severity band of the score set the priority of the threat, in line with data-driven and hybrid approaches to vulnerability prioritization [24, 25] and with prior use of CVSS metrics to quantify injection risk [26]. Because CVSS expresses technical severity rather than organizational risk [27], the scores were used to order work and not as absolute risk measures. Each threat was finally expressed as one or more verifiable security requirements, and each requirement was allocated to a control in the framework or in custom application code.

2.4. Evaluation criteria

Three criteria were used to evaluate the outcome. Traceability coverage measured the proportion of threats linked to at least one security requirement and of requirements linked to at least one allocated control. Framework support measured the proportion of requirements satisfied by built-in framework or platform mechanisms, which served as a proxy for the implementation effort imposed on a student development team. Severity distribution described how the threats were spread across the CVSS bands and the STRIDE categories. Confirmation of the allocated controls through controlled testing, using scanner-based and manual approaches of the kind compared in prior studies [5, 6, 8], is the subsequent phase of the research.

3. RESULTS AND DISCUSSION

3.1. Requirements and design models

The requirements analysis identified four actors. Public visitors browse and search published projects. Students create and maintain their own project records, upload media, and edit their profiles. Administrators manage student records and review projects before publication. The super administrator manages user

accounts, roles, and permissions. Figure 2 presents the use case diagram, and Table 2 summarizes the permissions that the role-based access control model assigned to each actor. The entity relationship diagram separated account, role, and permission entities from student and project entities, so that authorization data and content data could be protected independently.

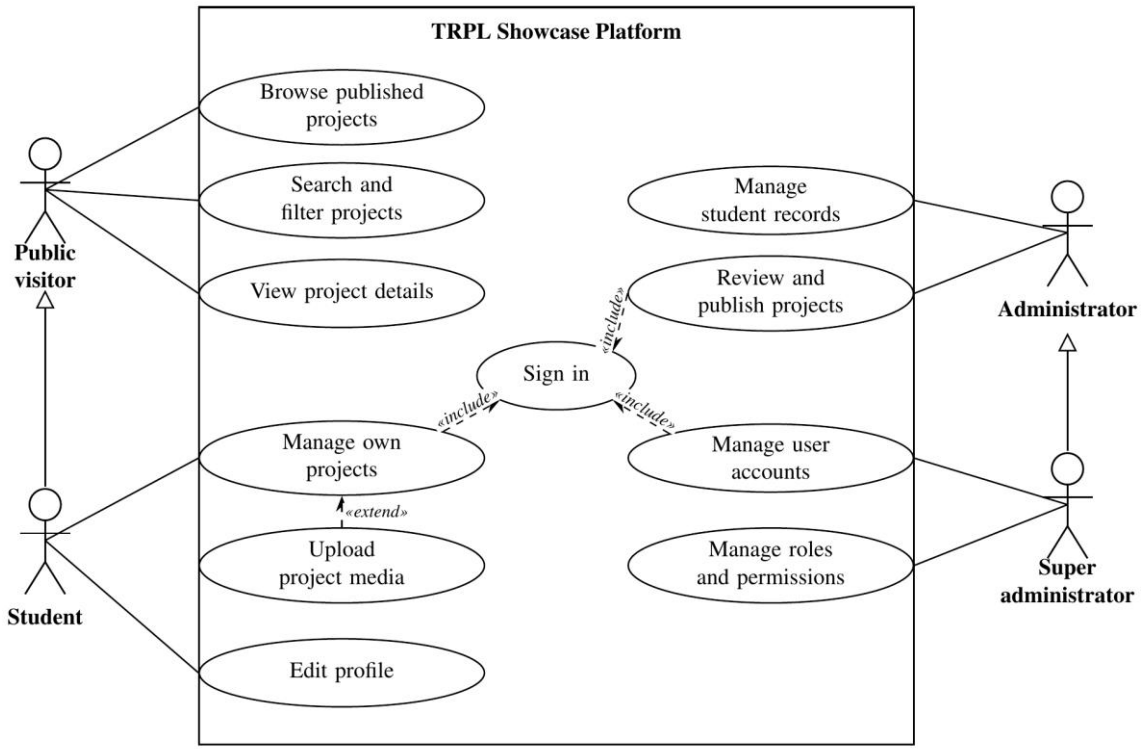


Figure 2. Use case diagram of the showcase platform.

Table 2. Permissions assigned by the role-based access control model.

Capability	Public visitor	Student	Administrator	Super administrator
Browse and search published projects	Yes	Yes	Yes	Yes
Create, edit, and upload media for own projects	-	Yes	-	Yes
View own draft projects	-	Yes	Yes	Yes
Edit own profile	-	Yes	Yes	Yes
Review and publish projects	-	-	Yes	Yes
Manage student records	-	-	Yes	Yes
Manage user accounts	-	-	-	Yes
Manage roles and permissions	-	-	-	Yes

Figure 3 shows the public home page of the deployed prototype. It provides keyword search, an advanced filter by category, year, and technology, summary counters, and, further down the page, a list of featured projects. At the time of capture, the page listed 21 published projects across eight categories, and only records in the Published state were visible to anonymous visitors.

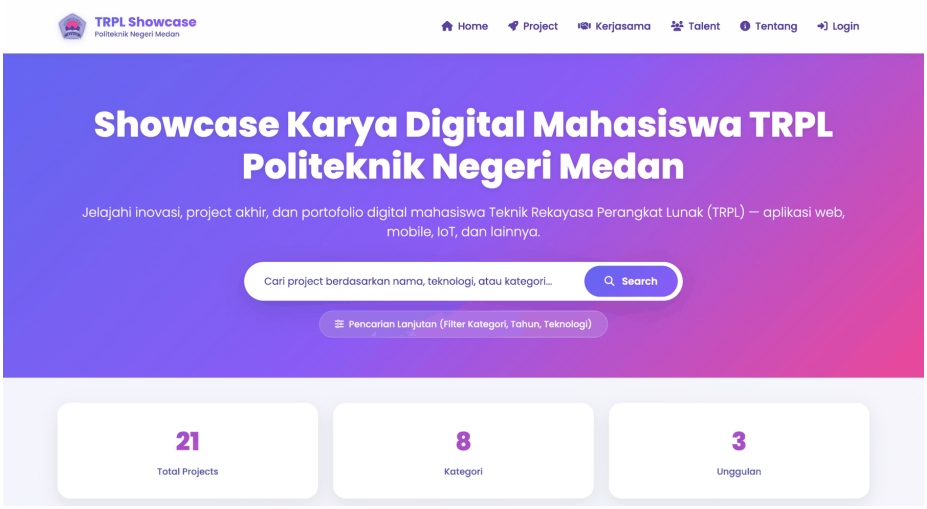


Figure 3. Public home page of the deployed showcase platform.

3.2. Data flow diagram

Figure 4 shows the data flow diagram derived in Stage 3. It contains three external entities (public visitor, student, and administrator), five processes (P1 public showcase, P2 authentication and RBAC filter, P3 project management, P4 administrative management, and P5 upload handler), and four data stores (D1 media files, D2 project and student records, D3 users and roles, and D4 audit log). Trust boundary TB1 separates the internet zone from the application zone, and trust boundary TB2 separates the application zone from the data zone. Every request that crosses TB1 from an authenticated actor passes through P2, which makes P2 the central enforcement point for identity and role.

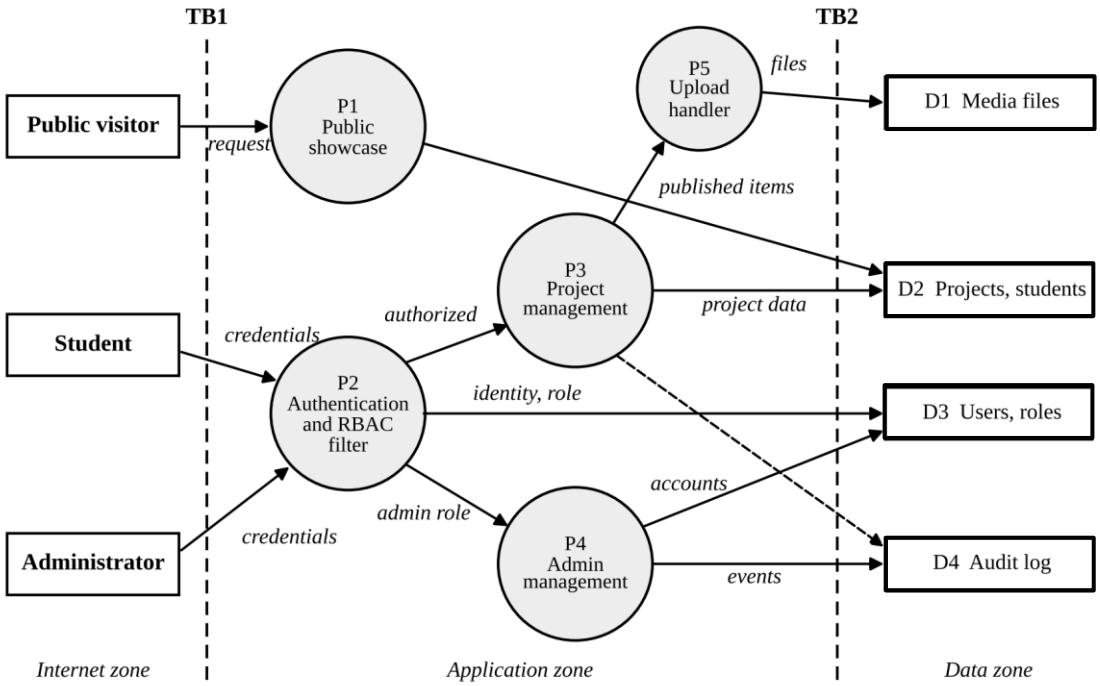


Figure 4. Data flow diagram of the showcase platform with trust boundaries.

3.3. Threat catalog and prioritization

Applying STRIDE to the elements of Figure 4 produced 17 threats, listed in Table 3. All six STRIDE categories were represented: spoofing (2), tampering (4), repudiation (1), information disclosure (4), denial of service (2), and elevation of privilege (4). Access control dominated the catalog, with six threats mapped to A01 Broken Access Control, followed by three threats mapped to A05 Injection. The two categories introduced in the 2025 edition of the OWASP Top 10 were both present, through the exploitation of a vulnerable third-party component (E4, A03 Software Supply Chain Failures) and the leakage of internal details through unhandled errors (I3, A10 Mishandling of Exceptional Conditions).

Table 3. Threat catalog with OWASP Top 10:2025 mapping and CVSS v3.1 base severity.

ID	Threat	DFD element	OWASP Top 10:2025	CVSS v3.1 base vector	Score	Severity
S1	Credential guessing and stuffing against the login form	P2, D3	A07 Authentication Failures	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	9.1	Critical
S2	Session hijacking or session fixation after login	P2	A07 Authentication Failures	AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N	7.4	High
T1	Stored cross-site scripting through project titles or descriptions	P3, P1, D2	A05 Injection	AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N	5.4	Medium
T2	SQL injection through search and filter parameters	P1, P3	A05 Injection	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	9.8	Critical
T3	Modification of another student's project through a manipulated object identifier	P3, D2	A01 Broken Access Control	AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:N	7.1	High
T4	Cross-site request forgery against administrative actions	P4	A01 Broken Access Control	AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N	6.5	Medium
R1	Project, account, or role changes without attributable records	P3, P4, D4	A09 Security Logging and Alerting Failures	AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N	4.3	Medium
I1	Over-exposure of student personal data on public pages	P1, D2	A01 Broken Access Control	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3	Medium
I2	Anonymous access to draft projects before publication	P1, D2	A01 Broken Access Control	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3	Medium
I3	Error pages that reveal stack traces, queries, or file paths	P1 to P5	A10 Mishandling of Exceptional Conditions	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3	Medium
I4	Interception of credentials on unencrypted connections	TB1 flows	A04 Cryptographic Failures	AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N	5.9	Medium
D1	Storage exhaustion through oversized or repeated uploads	P5, D1	A06 Insecure Design	AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H	6.5	Medium
D2	Request flooding of search and listing endpoints	P1	A06 Insecure Design	AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H	7.5	High
E1	Vertical escalation from the student role to administrative functions	P2, P4	A01 Broken Access Control	AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	8.8	High
E2	Role elevation through mass assignment of the role attribute	P3, D3	A01 Broken Access Control	AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	8.8	High
E3	Remote code execution through an uploaded executable file	P5, D1	A05 Injection	AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H	9.9	Critical
E4	Exploitation of a vulnerable framework or third-party component	P1 to P5	A03 Software Supply Chain Failures	AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H	8.1	High

The severity distribution was three Critical, six High, and eight Medium. The Critical threats were remote code execution through an uploaded executable file (E3, 9.9), SQL injection through search parameters (T2, 9.8), and credential guessing against the login form (S1, 9.1). These threats share a low attack complexity and a high impact, so they were assigned to the first prototype iterations in which the login, search, and upload functions were built. The High threats were dominated by privilege escalation and object-level authorization (E1, E2, E4, T3) and by session handling and service availability (S2, D2). The Medium threats concerned information disclosure, request forgery, resource exhaustion, stored scripting, and the attribution of changes. This ordering agrees with surveys that place injection and access control among the most damaging web application weaknesses [11, 13].

3.4. Security requirements and traceability

Table 4 lists the 16 security requirements derived in Stage 4 and the control allocated to each. Every threat in Table 3 was addressed by at least one requirement, and every requirement was allocated to at least one control, which gave a traceability coverage of 100% in both directions. One requirement (SR12) addressed two threats (D1 and E3), because restrictions on file type, size, name, and storage location reduce both resource exhaustion and code execution.

Table 4. Security requirements, addressed threats, and allocated controls.

ID	Security requirement	Threats	Allocated control	Type
SR1	Store passwords with an adaptive hash and throttle repeated login attempts	S1	password_hash() and password_verify(); Throttler applied to the login route	Built-in
SR2	Regenerate the session identifier at login and protect session cookies	S2	Session regenerate(); Secure, HttpOnly, and SameSite flags and idle expiry in the Cookie and Session configuration	Built-in
SR3	Validate every input and encode output for its rendering context	T1	Validation library rules; esc() in all views; Content Security Policy header	Built-in
SR4	Access the database only through parameterized statements	T2	Query Builder and query bindings; raw string concatenation prohibited in code review	Built-in

ID	Security requirement	Threats	Allocated control	Type
SR5	Check object ownership on every update and delete operation	T3	Model queries constrained by the authenticated owner identifier	Custom
SR6	Protect every state-changing request with an anti-forgery token	T4	Global CSRF filter with tokens in all forms	Built-in
SR7	Record security-relevant events with actor, time, and source address	R1	Model events (afterInsert, afterUpdate, afterDelete) writing to the audit log store	Custom
SR8	Expose only approved profile fields on public pages	I1	Public view models with an explicit field allowlist	Custom
SR9	Serve only records in the Published state to anonymous users	I2	Status condition enforced in the public showcase controller	Custom
SR10	Suppress detailed error output in production	I3	Production environment setting, generic error views, and server-side exception logging	Built-in
SR11	Enforce encrypted transport for all requests	I4	forceGlobalSecureRequests option and HTTP Strict Transport Security header	Built-in
SR12	Restrict upload type, size, name, and storage location	D1, E3	Upload validation rules (uploaded, max_size, mime_in, ext_in); getRandomName(); storage outside the public web root	Built-in
SR13	Paginate listings and rate-limit anonymous endpoints	D2	Pager library and Throttler filter	Built-in
SR14	Authorize every route by role with a deny-by-default rule	E1	Route filters on route groups mapped to the role and permission tables	Built-in
SR15	Restrict which model attributes a request may assign	E2	Model allowedFields lists that exclude role and ownership attributes	Built-in
SR16	Maintain a dependency inventory and apply security updates	E4	Composer lock file, periodic composer audit, and framework release tracking	Built-in

Twelve of the 16 requirements (75%) were satisfied by mechanisms that CodeIgniter 4 or the PHP platform already provide, including parameterized queries, output escaping, cross-site request forgery protection, upload validation, request throttling, route filters, and restrictions on assignable model fields. The remaining 4 requirements (SR5, SR7, SR8, and SR9) required custom code because they encode decisions that a framework cannot infer: who owns a project, which publication state is visible to anonymous users, which profile fields are public, and which events must be recorded. Object-level authorization is a known blind spot of generic protection [13], and the custom requirements concentrated exactly there. For a student development team, this split means that most protection follows from enabling and configuring framework features correctly, while scarce review effort can be focused on a small set of business rules.

3.5. Discussion

Compared with assessments of deployed systems [3-6], the process moved the identification of weaknesses ahead of construction. The threats in Table 3 overlap with the classes that scanners and penetration tests report, but they were identified from the design models and linked to named controls before the corresponding functions were implemented. This is the shift toward early security activities advocated in secure software engineering [14, 15]. The process adds two stages to each iteration without changing the structure of prototyping, which keeps it compatible with the short feedback cycles that make prototyping attractive for small teams [20]. Detection research on injection and cross-site scripting [9, 10, 12, 28] remains relevant as a second layer, because design-level controls reduce but do not remove the need for runtime detection.

Three features of the vocational setting shaped the outcome. High public visibility raised the likelihood of opportunistic scanning, which justified the priority given to injection, upload, and authentication threats. Frequent turnover of student developers weakened the continuity of security knowledge, and the documented chain from threat to requirement to control preserved that knowledge in an explicit form that successive cohorts can inspect. A small number of privileged accounts, four in this platform, limited the exposure of administrative functions, but it also made the protection of those accounts and the audit of their actions (SR1, SR2, SR7, and SR14) a priority. The process also has educational value. Student code without security guidance tends to repeat the same weaknesses [21], and the element-by-category analysis of Table 1 can serve as a structured exercise in which students elicit threats from the platform they build, in line with curricula that favor applied security activities [22].

The study has limitations. It examined a single platform, so the threat catalog reflects one architecture and one framework, and generalization to other designs requires care. The CVSS scores are analytical base scores derived from the design models rather than from observed exploitation, and environmental factors may raise or lower them in a specific deployment [27]. The allocated controls are reported as design decisions, and their effectiveness has not yet been confirmed through controlled testing. These limitations define the next phase of the research, which will verify each control against the threat it addresses.

4. CONCLUSION

This study integrated STRIDE threat modeling into the prototyping lifecycle of a vocational student showcase platform built with CodeIgniter 4, Bootstrap 5, and MySQL under role-based access control. Inserting data flow diagram decomposition with STRIDE elicitation and the derivation of CVSS-prioritized security requirements between quick design and construction produced 17 threats across all six STRIDE categories, 16 security requirements, and complete traceability from each threat to an allocated control. Access control and injection dominated the catalog, and the two categories introduced in the OWASP Top 10:2025 were both present. Twelve of the 16 requirements were met by built-in framework or platform mechanisms, so the added effort remained compatible with student-led development, while custom effort concentrated on ownership, publication state, data minimization, and auditing. Future work will confirm the allocated controls through controlled testing aligned with the OWASP Web Security Testing Guide, evaluate the process with additional platforms and development teams, and examine its use as a teaching instrument in software engineering courses.

5. ACKNOWLEDGEMENTS

The authors gratefully acknowledge Politeknik Negeri Medan, in particular the Center for Research and Community Service (P3M Polmed), for funding this research through the Fundamental Research Grant Scheme (Penelitian Fundamental DIPA) for the 2026 fiscal year under contract number B/320/PL5/PT.01.05/2026. The authors also acknowledge the five undergraduate students of the D4 Software Engineering Technology (TRPL) program who served as research assistants during the vulnerability assessment and data collection phases.

6. REFERENCES

[1] J. B. Ulven and G. Wangen, "A systematic review of cybersecurity risks in higher education," *Future Internet*, vol. 13, no. 2, p. 39, 2021, doi: https://doi.org/10.3390/fi13020039.

[2] N. S. Fouad, "Securing higher education against cyberthreats: from an institutional risk to a national policy challenge," *Journal of Cyber Policy*, vol. 6, no. 2, pp. 137-154, 2021, doi: https://doi.org/10.1080/23738871.2021.1973526.

[3] M. Zaidan, F. Noeraini, Z. Sari, and D. R. Akbi, "Website vulnerability analysis of AB and XY office in East Java," *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika*, vol. 9, no. 2, pp. 455-492, 2023, doi: https://doi.org/10.26555/jiteki.v9i2.26183.

[4] C. Darmawan, J. P. P. Naibaho, and A. De Kweldju, "Penerapan metode vulnerability assessment untuk identifikasi keamanan website berdasarkan OWASP ID tahun 2021," *Edumatic: Jurnal Pendidikan Informatika*, vol. 8, no. 1, pp. 272-281, 2024, doi: https://doi.org/10.29408/edumatic.v8i1.25834.

[5] K. Abdulghaffar, N. Elmrabit, and M. Yousefi, "Enhancing web application security through automated penetration testing with multiple vulnerability scanners," *Computers*, vol. 12, no. 11, p. 235, 2023, doi: https://doi.org/10.3390/computers12110235.

[6] M. Albahar, D. Alansari, and A. Jurcut, "An empirical comparison of pen-testing tools for detecting web app vulnerabilities," *Electronics*, vol. 11, no. 19, p. 2991, 2022, doi: https://doi.org/10.3390/electronics11192991.

[7] J. Shahid, M. K. Hameed, I. T. Javed, K. N. Qureshi, M. Ali, and N. Crespi, "A comparative study of web application security parameters: current trends and future directions," *Applied Sciences*, vol. 12, no. 8, p. 4077, 2022, doi: https://doi.org/10.3390/app12084077.

[8] S. Qadir, E. Waheed, A. Khanum, and S. Jehan, "Comparative evaluation of approaches and tools for effective security testing of web applications," *PeerJ Computer Science*, vol. 11, p. e2821, 2025, doi: https://doi.org/10.7717/peerj-cs.2821.

[9] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL injection attack using machine learning techniques: a systematic literature review," *Journal of Cybersecurity and Privacy*, vol. 2, no. 4, pp. 764-777, 2022, doi: https://doi.org/10.3390/jcp2040039.

[10] J. Kaur, U. Garg, and G. Bathla, "Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review," *Artificial Intelligence Review*, vol. 56, no. 11, pp. 12725-12769, 2023, doi: https://doi.org/10.1007/s10462-023-10433-3.

[11] A. Paul, V. Sharma, and O. Olukoya, "SQL injection attack: detection, prioritization and prevention," *Journal of Information Security and Applications*, vol. 85, p. 103871, 2024, doi: https://doi.org/10.1016/j.jisa.2024.103871.

[12] R. Bakır, "UniEmbed: a novel approach to detect XSS and SQL injection attacks leveraging multiple feature fusion with machine learning techniques," *Arabian Journal for Science and Engineering*, vol. 50, no. 19, pp. 15591-15604, 2025, doi: https://doi.org/10.1007/s13369-024-09916-4.

[13] A. Anas, S. Elgamal, and B. Youssef, "Survey on detecting and preventing web application broken access control attacks," *International Journal of Electrical and Computer Engineering*, vol. 14, no. 1, pp. 772-781, 2024, doi: https://doi.org/10.11591/ijece.v14i1.pp772-781.

[14] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas, "Systematic mapping study on security approaches in secure software engineering," *IEEE Access*, vol. 9, pp. 19139-19160, 2021, doi: https://doi.org/10.1109/ACCESS.2021.3052311.

[15] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: a systematic review," *Information and Software Technology*, vol. 141, p. 106700, 2022, doi: https://doi.org/10.1016/j.infsof.2021.106700.

[16] K. Rindell, J. Ruohonen, J. Holvitie, S. Hyrynsalmi, and V. Leppänen, "Security in agile software development: a practitioner survey," *Information and Software Technology*, vol. 131, p. 106488, 2021, doi: https://doi.org/10.1016/j.infsof.2020.106488.

[17] S. M. Khalil, H. Bahsi, and T. Korõtko, "Threat modeling of industrial control systems: a systematic literature review," *Computers & Security*, vol. 136, p. 103543, 2024, doi: https://doi.org/10.1016/j.cose.2023.103543.

[18] M. Tatam, B. Shanmugam, S. Azam, and K. Kannoorpatti, "A review of threat modelling approaches for APT-style attacks," *Heliyon*, vol. 7, no. 1, p. e05969, 2021, doi: https://doi.org/10.1016/j.heliyon.2021.e05969.

[19] F. Valenza, E. Karafili, R. V. Steiner, and E. C. Lupu, "A hybrid threat model for smart systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4403-4417, 2023, doi: https://doi.org/10.1109/TDSC.2022.3213577.

[20] E. Bjarnason, F. Lang, and A. Mjöberg, "An empirically based model of software prototyping: a mapping study and a multi-case study," *Empirical Software Engineering*, vol. 28, no. 5, p. 115, 2023, doi: https://doi.org/10.1007/s10664-023-10331-w.

[21] T. Yilmaz and Ö. Ulusoy, "Understanding security vulnerabilities in student code: a case study in a non-security course," *Journal of Systems and Software*, vol. 185, p. 111150, 2022, doi: https://doi.org/10.1016/j.jss.2021.111150.

[22] M. Ismail, N. T. Madathil, M. Alalawi, S. Alrabaee, M. Al Bataineh, S. Melhem, and D. Mouheb, "Cybersecurity activities for education and curriculum design: a survey," *Computers in Human Behavior Reports*, vol. 16, p. 100501, 2024, doi: https://doi.org/10.1016/j.chbr.2024.100501.

[23] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38-47, 1996, doi: https://doi.org/10.1109/2.485845.

[24] T. H. M. Le, H. Chen, and M. A. Babar, "A survey on data-driven software vulnerability assessment and prioritization," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1-39, 2022, doi: https://doi.org/10.1145/3529757.

[25] A. Sharma, S. Sabharwal, and S. Nagpal, "A hybrid scoring system for prioritization of software vulnerabilities," *Computers & Security*, vol. 129, p. 103256, 2023, doi: https://doi.org/10.1016/j.cose.2023.103256.

[26] A. Kurniawan, M. Y. Darus, M. A. M. Ariffin, Y. Muliono, and C. R. Pardomuan, "Automation of quantifying security risk level on injection attacks based on common vulnerability scoring system metric," *Pertanika Journal of Science and Technology*, vol. 31, no. 3, pp. 1245-1265, 2023, doi: https://doi.org/10.47836/pjst.31.3.07.

[27] J. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick, "Time to change the CVSS?," *IEEE Security & Privacy*, vol. 19, no. 2, pp. 74-78, 2021, doi: https://doi.org/10.1109/MSEC.2020.3044475.

[28] C. R. Pardomuan, A. Kurniawan, M. Y. Darus, M. A. M. Ariffin, and Y. Muliono, "Server-side cross-site scripting detection powered by HTML semantic parsing inspired by XSS auditor," *Pertanika Journal of Science and Technology*, vol. 31, no. 3, pp. 1353-1377, 2023, doi: https://doi.org/10.47836/pjst.31.3.14.