symmetry

MDPI

Article

A Machine Learning-Based Detection for Parameter Tampering Vulnerabilities in Web Applications Using BERT Embeddings

Sun Young Yun [1] and Nam-Wook Cho [2,*]

[1] Graduate School of Public Policy and IT, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea; juna77@seoultech.ac.kr
[2] Department of Industrial and Information Systems Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea
* Correspondence: nwcho@seoultech.ac.kr; Tel.: +82-2970-6448

Abstract

The widespread adoption of web applications has led to a significant increase in the number of automated cyberattacks. Parameter tampering attacks pose a substantial security threat, enabling privilege escalation and unauthorized data exfiltration. Traditional pattern-based detection tools exhibit limited efficacy against such threats, as identical parameters may produce varying response patterns contingent on their processing context, including security filtering mechanisms. This study proposes a machine learning-based detection model to address these limitations by identifying parameter tampering vulnerabilities through a contextual analysis. The training dataset aggregates real-world vulnerability cases collected from web crawls, public vulnerability databases, and penetration testing reports. The Synthetic Minority Over-sampling Technique (SMOTE) was employed to address the data imbalance during training. Recall was adopted as the primary evaluation metric to prioritize the detection of true vulnerabilities. Comparative analysis showed that the XGBoost model demonstrated superior performance and was selected as the detection model. Validation was performed using web URLs with known parameter tampering vulnerabilities, achieving a detection rate of 73.3%, outperforming existing open-source automated tools. The proposed model enhances vulnerability detection by incorporating semantic representations of parameters and their values using BERT embeddings, enabling the system to learn contextual characteristics beyond the capabilities of pattern-based methods. These findings suggest the potential of the proposed method for scalable, efficient, and automated security diagnostics in large-scale web environments.

Keywords: web security; parameter tampering vulnerabilities; BERT; XGBoost; SMOTE; machine learning

Academic Editor: Zhixun Su

Received: 30 April 2025
Revised: 12 June 2025
Accepted: 16 June 2025
Published: 22 June 2025

Citation: Yun, S.Y.; Cho, N.-W. A Machine Learning-Based Detection for Parameter Tampering Vulnerabilities in Web Applications Using BERT Embeddings. *Symmetry* **2025**, *17*, 985. https://doi.org/10.3390/sym17070985

1. Introduction

Web applications serve as critical infrastructure for modern online services; however, their widespread adoption has led to increasingly sophisticated cyberattacks. Technological advances have enabled more sophisticated automated attacks, particularly through parameter tampering techniques. Parameter tampering attacks represent a nuanced cybersecurity threat characterized by the strategic manipulation of web request values, enabling malicious actors to circumvent user privilege controls and potentially execute unauthorized data extraction. The complex asymmetrical response patterns of these attacks render traditional rule-based detection methods inadequate.

Artificial intelligence (AI) technologies have emerged in the field of cybersecurity, fundamentally transforming the nature of cyberattacks. Contemporary threat actors can now leverage AI to automatically analyze expansive data volumes, systematically identify potential attack trajectories, and strategically bypass defensive infrastructure. Truong et al. [1] critically emphasized AI's dual-purpose potential in cyberspace, underscoring the imperative for sophisticated, real-time detection capabilities. Recent research has increasingly focused on covert-channel attack techniques. Chen et al. [2] proposed a high-speed, software-based covert channel attack (WRITE+SYNC) that exploits memory-disk synchronization time differences, achieving high success rates across various operating system environments. These attacks exploit the limitations of existing detection systems, enabling increasingly sophisticated threats.

Modern approaches to web application security assessment are classified into two categories: manual testing and automated diagnostic tools. Manual testing requires significant time and resources, involving meticulous inspection of web pages, with outcomes dependent on evaluator expertise and prone to human error. Conversely, automated tools provide rapid and reproducible assessments based on established attack patterns but exhibit significant limitations in processing unstructured and unexpected inputs. These tools rely on rigid input formats and struggle to detect attacks that exploit the structural asymmetries between requests and responses. Sun et al. [3] proposed a full mesh aggregation-based orchestration technique to configure service function chains in multi-domain network environments efficiently, providing a technical foundation for the design of dynamic and automated security detection and response systems.

This study aims to design a machine learning–based model capable of automatically detecting web application vulnerabilities caused by parameter tampering. Traditional automated tools typically rely on predefined input substitution techniques, which exhibit significant limitations in identifying non-standard or context-sensitive inputs. To address this, this study constructs a training dataset derived from real-world vulnerability cases and evaluates the model performance following supervised training. Furthermore, the detection capability of the model is validated using GET-based web URLs that include confirmed parameter tampering vulnerabilities, and its effectiveness in identifying compromised parameters within such request formats is assessed.

The main contributions of this paper are as follows:

(1) We propose a novel method for detecting parameter tampering vulnerabilities by analyzing parameter–value contexts and applying manipulated inputs, thereby overcoming the limitations of conventional tools that rely on substituting predefined input patterns.
(2) We construct and train a machine learning model using real-world vulnerability data, enabling the identification of similar attacks without relying on fixed rules, and we present a strategy for prioritizing high-risk parameters.
(3) Our proposed methodology enables automated end-to-end vulnerability detection across entire application environments to address the difficulty of manually inspecting large-scale web applications.

The remainder of this paper is structured as follows: Section 2 provides a comprehensive review of related research; Section 3 outlines the employed methodology; Section 4 presents the experimental results and their analysis; and Section 5 offers conclusions and directions for future research.

2. Related Work

This section reviews related research focusing on automated security vulnerability detection and machine learning-based approaches for identifying vulnerabilities.

2.1. Research on Automated Vulnerability Detection

Sarker et al. [4] introduced IntruDTree, an innovative cyber intrusion detection framework that leverages feature selection techniques from security data. The researchers successfully classified multiple cyber-attack categories by constructing a robust, tree-based detection model. Their experimental validation demonstrated excellent performance metrics across diverse attack typologies, including Denial of Service (DoS), Probe, Remote-to-Local (R2L), and User-to-Root (U2R) attacks. Alhogail et al. [5] developed Kashef, an advanced automated extension-based penetration testing model for comprehensive web application security assessment. The methodological framework of the model encompasses Information Gathering, Vulnerability Assessment, and Exploitation & Reporting. Empirical evaluations substantiate the model's superiority, demonstrating enhanced vulnerability analysis efficiency and superior detection capabilities for critical vulnerabilities, such as SQL Injection, Server-Side Request Forgery (SSRF), and diverse exploit mechanisms. Tóth et al. [6] conducted a comprehensive security assessment of web application code generated by large language models (LLMs). Their experimental protocol involved deploying 2500 PHP-based websites generated by GPT-4 within Docker containerized environments and systematically analyzing them using the Burp Suite. The study revealed a substantial vulnerability landscape, identifying 2440 vulnerable parameters across the generated web applications. Bisht et al. [7] proposed NoTamper, a black-box analysis tool engineered to detect parameter tampering vulnerabilities in web applications. The study identified critical security vulnerabilities through extensive experimental validation across open-source applications and real-world production websites, including potential unauthorized fund transfer mechanisms and privilege escalation exploits. Deepa et al. [8] introduced DetLogic, a black-box approach for identifying web service vulnerabilities, comprising three systematic phases: Learning, Attack Generation, and Discovery. The experimental results demonstrate the effectiveness of the proposed technique in identifying complex vulnerabilities, particularly those associated with parameter manipulation, access control circumvention, and workflow bypass strategies.

2.2. ML-Based Vulnerability Detection

Aslam et al. [9] introduced AntiPhishStack, an LSTM-based stacked generalization model for malicious URL detection. The model employs a two-phase methodology that integrates machine learning-based prediction results (Phase I) with advanced LSTM-based predictions (Phase II) to generate comprehensive final outputs. They achieved notable performance with minimal parameter optimization by implementing character-level TF-IDF feature extraction. Bao et al. [10] developed a credit card fraud detection model using BERT. To address the critical challenge of dataset imbalance, they implemented a comprehensive preprocessing strategy involving correlation analysis and a nuanced combination of undersampling and oversampling techniques for normalizing the dataset. The proposed model demonstrated strong predictive performance, achieving a fraud detection accuracy of 99.95% in systematic experimental evaluation. Kshetri et al. [11] presented algoXSSF, an advanced machine learning-based framework for comprehensive detection and analysis of Cross-Site Request Forgery (XSRF) and Cross-Site Scripting (XSS) attacks. We conducted a comparative performance assessment of three models: AdaBoost, LSTM-Attention, and CNN-LSTM. The AdaBoost-based model demonstrated superior performance, achieving an accuracy of 99.92%. Castagnaro et al. [12] proposed an automated and optimized directory brute-forcing methodology for web application security assessment, leveraging the emerging paradigm of Offensive Artificial Intelligence. They compared probabilistic and AI-based methods and found that AI-driven approaches significantly outperformed probabilistic techniques. An et al. [13] developed V-CNN,

a sophisticated vulnerability detection model utilizing convolutional neural networks (CNNs), to systematically identify and classify Common Weakness Enumeration (CWE) and Common Vulnerabilities and Exposures (CVE) patterns. The proposed model achieved an accuracy of 98%, showing improved performance compared with the baseline Random Forest model. Chen et al. [2] proposed a software-based covert channel, WRITE+SYNC, which uses memory-disk synchronization delays. Fu et al. [14] proposed the H^2IDE model for fraud detection in multi-relational graphs. The H^2IDE model demonstrated superior detection performance and scalability compared with state-of-the-art techniques. Xu et al. [15] designed a 2D coupled-BFUs-based META tri-stage polynomial multiplication accelerator to address performance bottlenecks in polynomial multiplication operations in post-quantum cryptography (PQC) systems. The related research is summarized in Table 1.

Table 1. Summary of research on automated and ML-based vulnerability detection.

Ref. #	Title	Methodology	Results	Future Work
[2]	Write+Sync: Software Cache Write Covert Channels Exploiting Memory-Disk Synchronization	Software cache writes covert channels exploiting memory-disk synchronization timing (WRITE+SYNC)	Up to 253 Kb/s transmission rate, high success rates across diverse OSes/environments	Investigating dynamic protection techniques and detection feasibility analysis
[4]	IntruDTree: A machine learning-based cyber security intrusion detection model	Decision Tree-based IDS with feature ranking	98% accuracy, outperforming SVM/LogReg/KNN	Advanced feature selection techniques and additional dataset integration
[5]	Automated Extension-Based Penetration Testing for Web Vulnerabilities	Automated PenTest model based on browser extension	Effective detection of diverse web vulnerabilities faster than existing tools	Developing ML-based automated analysis/classification models and expanding support for network layer analysis
[6]	LLMs in Web Development: Evaluating LLM-Generated PHP Code Unveiling Vulnerabilities and Limitations	LLM (GPT-4)-based PHP code generation with static and dynamic analysis	11.16% sites with exploitable vulnerabilities; broader analysis than prior studies	Expanding the analysis to other programming languages and broader vulnerability categories
[7]	No Tamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications	Blackbox-based automated Parameter Tampering detection tool	Nine exploitable vulnerabilities were detected across nine applications	Extending support for AJAX and improving scalability for large-scale web applications
[8]	DetLogic: A black-box approach for detecting logic vulnerabilities in web applications	Blackbox FSM-based logic vulnerability detection (DetLogic)	Successful detection of Param Manipulation, access control, and Workflow vulnerabilities	Incorporating multi-user traces and dynamic analysis to enhance the detection of complex logic flaws
[9]	AntiPhishStack: LSTM-based Stacked Generalization Model for Optimized Phishing URL Detection	LSTM-based Stacked Generalization model for Phishing URL detection	96.04% accuracy, better than existing ML/DL models	Incorporating features from web page content/structure and evaluating larger and more diverse datasets
[10]	Application of Deep Learning in Financial Credit Card Fraud Detection	Transformer (BERT)-based Credit Card Fraud detection model	99.95% accuracy, outperforming previous approaches	Applying the model to detect other fraud types and enhancing adaptability to evolving fraud patterns

Table 1. *Cont.*

Ref. #	Title	Methodology	Results	Future Work
[11]	algoXSSF: Detection and analysis of cross-site request forgery (XSRF) and cross-site scripting (XSS) attacks via Machine learning algorithms	ML-based algoXSSF model for XSRF/XSS detection	99.92% XSS accuracy using AdaBoost and other ML algorithms	Using larger and more up-to-date datasets and exploring hybrid learning approaches
[12]	Offensive AI: Enhancing Directory Brute-forcing Attack with the Use of Language Models	LM-enhanced Directory Brute-forcing attack model	Significant performance improvement over traditional wordlist-based attacks	Evaluating in real-world environments and optimizing initial request efficiency
[13]	A CNN-based automatic vulnerability detection	CNN-based V-CNN model for CWE/CVE detection	98% accuracy, outperforming Random Forest	Improving detection accuracy and utilizing real-world source code datasets
[14]	Nowhere_to_H2IDE_Fraud_Detection_From_Multi-Relation_Graphs_via_Disentangled_Homophily_and_Heterophily_Identification	Disentangled representation learning-based H^2IDE model for fraud detection in multi-relation graphs	Superior benchmark results; demonstrated scalability	Extending to more complex heterogeneous graphs and diverse relation types
[15]	Meta: A Memory-Efficient Tri-Stage Polynomial Multiplication Accelerator Using 2D Coupled-BFUs	META tri-stage polynomial multiplication accelerator using 2D Coupled-BFUs	Up to 10× memory efficiency improvement, up to 13.8× ATP improvement over prior work	Optimizing performance via ASIC implementation and extending to secure application domains

Contemporary cybersecurity studies have substantiated various methodological approaches for automated vulnerability detection, with most investigations leveraging machine learning algorithms. Despite these contributions, existing methods primarily focus on detection using predefined input structures, limiting their effectiveness against sophisticated attacks like parameter tampering. This study introduces an innovative detection methodology that systematically trains a machine learning model using authentic, real-world vulnerability datasets to identify parameter tampering vulnerabilities. The main objective of this study is to overcome the limitations of traditional automated detection techniques, which rely heavily on static input patterns, and establish a more adaptive framework capable of addressing unstructured cybersecurity threats.

3. Methodology

3.1. Data Collection and Preprocessing

This study systematically collected and labeled web parameters and their corresponding values to facilitate the detection of vulnerabilities attributable to parameter manipulation attacks. Data acquisition was conducted using a multi-source approach encompassing web crawling operations, penetration testing documentation analysis, and systematic extraction from the Exploit-DB vulnerability repository.

From the Exploit-DB database, data samples were selectively curated based on parameter manipulation vulnerabilities categorized under the "Type (WebApps)" classification. The datasets extracted from penetration testing reports and Exploit-DB were methodically labeled as either normal or vulnerable, as these sources contained paired instances of legitimate parameter values alongside their corresponding manipulated counterparts for identical parameters. Furthermore, the dataset was augmented with vulnerability data from open-source software repositories, where parameter manipulation exploits had been previously documented and verified.

Web crawling operations targeted open-source web applications, with parameter selection criteria focusing on security-critical elements susceptible to tampering. Priority was given to parameters governing user identification (ID), authentication mechanisms, and pricing structures, which present significant attack vectors for privilege escalation, unauthorized price modification, and illicit data access. Since crawled data exclusively contained legitimate parameter values from operational systems, these instances were uniformly classified as benign samples.

The comprehensive composition and distributional characteristics of the compiled dataset are presented in Table 2 and illustrated in Figure 1, providing detailed insights into the data structure and class distribution employed for model training and validation.

Table 2. Training dataset composition and class distribution.

Category	Data	Percentage (%)
Benign Data(0)	4363	63.60%
Vulnerable Data(1)	2497	36.40%
Total	6860	100%

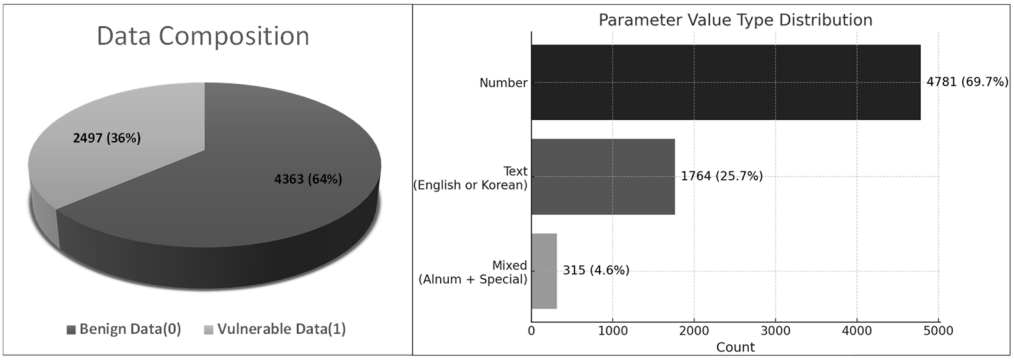

Figure 1. Overview of dataset composition and parameter value types.

The collected empirical data predominantly comprise structured inputs characterized by alphanumeric configurations. The semantic interpretation of these inputs demonstrates intricate complexity, wherein parameter-value combinations exhibit nuanced relational dynamics that transcend traditional linear mappings. These relationships manifest pronounced asymmetrical characteristics, with semantic meaning being dynamically contingent upon contextual deployment. To capture intricate semantic nuances, this study employed BERT embeddings, a natural language processing technique that is capable of context-aware semantic analysis. Unlike conventional methodological approaches, such as TF-IDF or Word2Vec, which rely predominantly on word frequency or static similarity metrics, BERT facilitates semantic vectorization through comprehensive bidirectional contextual analysis. The embedding methodology effectively captures relational complexity by semantically positioning the parameter values in the vector space.

BERT's algorithmic approach enables the generation of semantically proximal vector representations for contextually similar parameter values. While traditional string-based comparisons might treat inputs such as color = red and color = blue as discrete entities, BERT's advanced learning mechanism recognizes their contextual similarities, generating comparably structured embeddings. This nuanced approach significantly enhances the precision of anomalous value detection in parameter-tampering scenarios.

The vectorization process employed the BERT tokenizer to transform the input data, strategically extracting the [CLS] token vector as a comprehensive representative feature encapsulating the holistic semantics of each input configuration. To ensure methodological

rigor and dataset consistency, the researchers implemented specific preprocessing parameters: padding = True to standardize input lengths, truncation = True to handle inputs exceeding the maximum lengths, and return_tensors = 'pt' to convert data into PyTorch tensors (version 2.5.0) for optimized computational processing.

The implementation of BERT embeddings renders the dataset well-suited for detecting parameter tampering vulnerabilities. By enabling sophisticated machine learning models to discern the semantic relationships between normative and potentially malicious values, this approach provides a robust mechanism for vulnerability identification. The relationship-centric methodology effectively identifies both symmetrical and asymmetrical structures, significantly enhancing the overall detection performance.

3.2. Research Framework

As illustrated in Figure 2, the proposed framework encompasses four critical methodological stages: Data Collection, Data Processing, Model Selection, and Vulnerability Analysis. Each stage is designed to systematically process security vulnerability-related data and apply advanced machine learning techniques for comprehensive vulnerability detection.

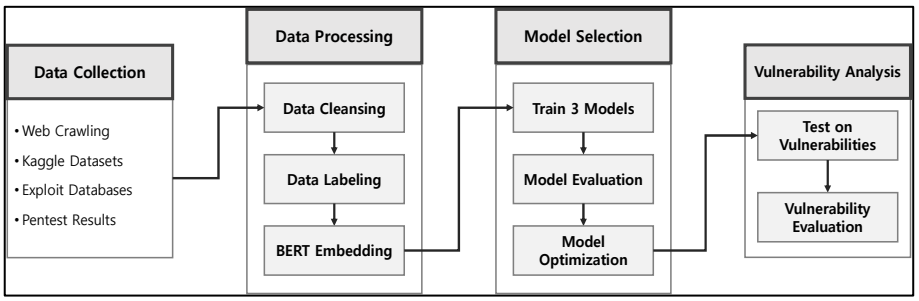

Figure 2. Research framework.

In the Data Collection stage, a comprehensive approach is employed to aggregate security-related data from diverse sources. Web application data are systematically collected through sophisticated web crawling techniques from publicly accessible online platforms. Furthermore, security vulnerability information is rigorously extracted by conducting in-depth analyses of specialized exploit databases, such as Exploit-DB, and by leveraging penetration testing results. The Data Processing stage involves sophisticated data refinement and transformation methodologies to prepare the collected data for analytical purposes. During the rigorous data cleansing process, multiple critical operations are performed: duplicate entries are systematically eliminated, identified errors are methodically corrected, and irrelevant information is meticulously filtered to ensure optimal data consistency. Each data sample is labeled as benign or vulnerable using a standardized protocol. We apply BERT embedding to convert the semantic relationships between the parameters and their values into vector forms for machine learning. In the Model Selection stage, a comprehensive comparative evaluation of multiple machine learning models is conducted to identify the most effective detection approach. Specifically, three advanced models (Random Forest, XGBoost, and LightGBM) are systematically trained and rigorously evaluated. The performance is assessed using evaluation metrics, including accuracy, precision, recall, and F1-score. The model with the highest overall performance is selected, followed by detailed optimization procedures to further enhance the detection capabilities. The Vulnerability Analysis stage is a critical validation mechanism used to assess the efficacy of the trained model in detecting security vulnerabilities. A carefully curated set of 30 authenticated security vulnerability samples is selected for comprehensive testing. Systematic evaluations are conducted to assess the detection performance of the

model across these samples. The overall detection rate of the automated diagnostic system is evaluated by applying the developed model to the entire sample set.

Algorithm 1 presents the procedure for determining the vulnerability of the parameter values in web request URLs. During the learning stage, BERT-based embeddings and XGBoost classifiers are employed to generate vectors and labels for each parameter-value pair. The algorithm extracts the parameters and values from the input URL during the detection stage. Then, it generates embeddings for each value and calculates the vulnerability probability using a classification model. A parameter is classified as vulnerable if the probability exceeds the specified threshold.

Algorithm 1: Web vulnerable parameter detection based on BERT embedding and XGBoost classifier

```
PROCEDURE: VULNERABLE PARAMETER DETECTOR
    Train BERT + XGBoost model using labeled parameter-value pairs;

PROCEDURE: PREDICT VULNERABLE PARAMETERS (URL, threshold)
    Extract parameter-value pairs from URL;
    Initialize empty list vulnerability_predictions;

    For each parameter-value pair, do
        Generate BERT embedding;
        Predict vulnerability probability using a trained model;
        If probability ≥ threshold, then
            Retrieve suggested keywords for attack testing;
            Add (parameter name, probability,
                suggested keywords) to vulnerability_predictions;
        end if
    end for

    For each entry in vulnerability_predictions, do
        Print parameter name, probability, suggested keywords;
    end for

    Return vulnerability_predictions;
end
```

3.3. Parameter Tampering Vulnerabilities

Parameter tampering is a method of attack that involves the manipulation of request parameters exchanged between clients and servers. This manipulation is intended to alter the behavior of web applications. Attackers endeavor to modify parameter values contained in URL query strings, form fields, cookies, HTTP headers, and other elements to gain elevated privileges, manipulate prices, access other users' data, and perform other malicious actions. Such attacks occur when an application lacks input or business logic validation, potentially leading to security issues such as SQL injection, cross-site scripting (XSS), and privilege escalation [16]. As illustrated in Figure 3, an attacker can manipulate the request parameters to alter the server's processing results.

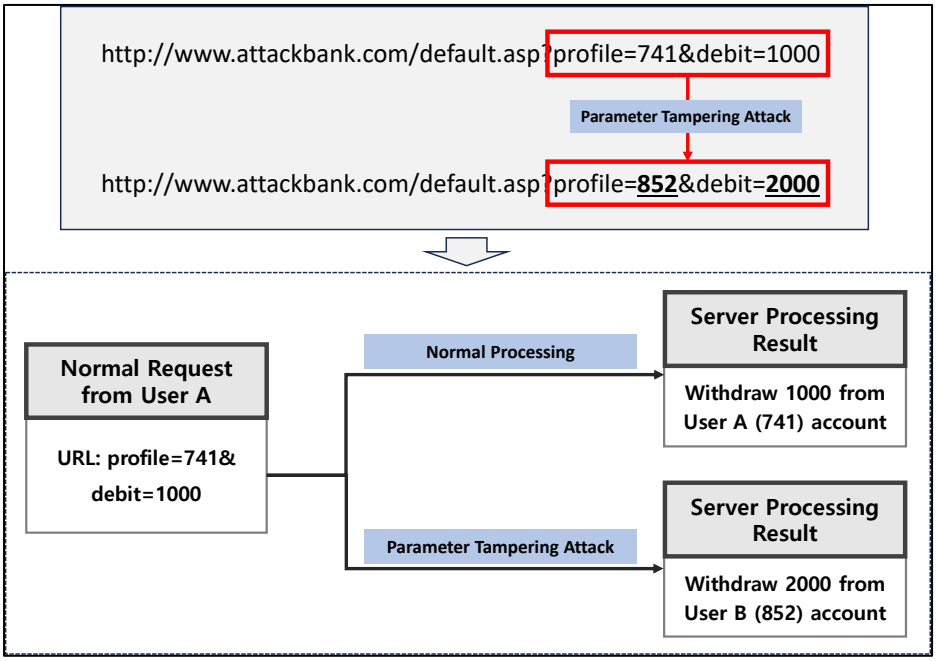

Figure 3. Conceptual diagram of parameter tampering attack [16].

3.4. Machine Learning Model Training and Data Processing Techniques

3.4.1. BERT Algorithm

BERT (Bidirectional Encoder Representations from Transformers) is a sophisticated pre-trained natural language processing (NLP) model developed by Google. It is designed to learn bidirectional contextual relationships between words and linguistic structures. Diverging from traditional approaches such as TF-IDF or Word2Vec, which predominantly rely on word frequency or static semantic representations, BERT effectively captures contextual sentence meanings, enabling dynamic semantic representations. The model's architectural foundation is based on the Transformer framework, which utilizes an innovative self-attention mechanism to learn the interdependencies between words in bidirectional contexts. This sophisticated approach enables BERT to generate dynamically contextualized vector representations for identical words contingent on their specific linguistic environment [17]. Figure 4 shows the pre-training and fine-tuning stages of the BERT model. This study used BERT embeddings to vectorize the intricate relationships between web parameters and their corresponding values, thereby enabling a more nuanced and precise differentiation between benign and vulnerable inputs.

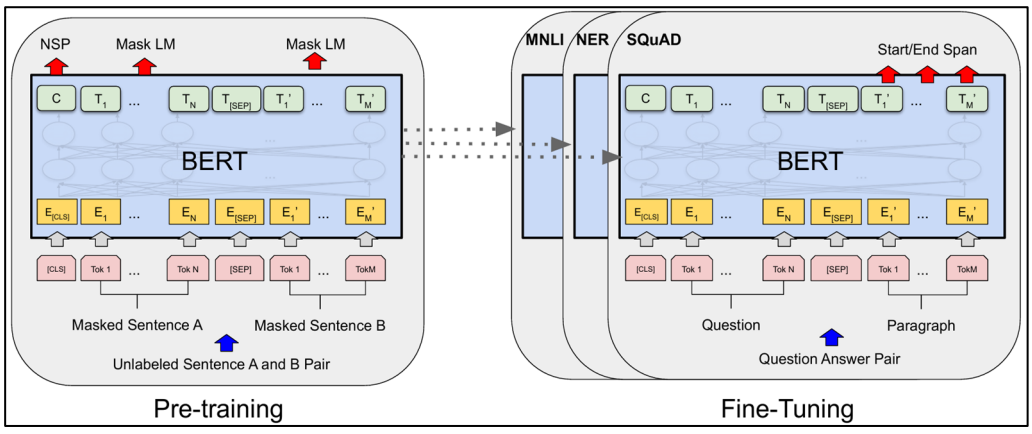

Figure 4. Pre-training and fine-tuning stages of the BERT model [17].

3.4.2. XGBoost Algorithm

XGBoost (eXtreme Gradient Boosting) is a prominent machine learning algorithm rooted in the Gradient Boosting framework, distinguished by its exceptional predictive performance and accelerated training capabilities. The underlying boosting methodology operates through the sequential training of multiple weak learners to systematically reduce prediction errors. XGBoost significantly enhances this process by implementing sophisticated optimization techniques that improve learning efficiency and model performance. The algorithm incorporates advanced regularization mechanisms to mitigate overfitting risks, and leverages parallelized tree construction strategies to expedite computational processes [18]. In this study, the vectorized data generated through BERT embeddings were strategically input into the XGBoost classifier to differentiate between the benign and vulnerable parameter values. Additionally, a comprehensive feature importance analysis was conducted using XGBoost to identify the critical parameters associated with security vulnerability detection.

3.4.3. SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) represents an innovative oversampling methodology designed to address class imbalance challenges in supervised learning environments. When datasets exhibit a substantially disproportionate sample distribution across classes, machine learning models may encounter significant limitations in adequately learning minority class characteristics, potentially resulting in biased predictive outcomes. SMOTE employs a sophisticated interpolation-based approach that utilizes the k-Nearest Neighbors (k-NN) algorithm to generate synthetic samples within the minority class. Unlike simple duplication methods, SMOTE preserves the inherent complexity of the underlying data distribution, consequently enhancing the model generalization performance [19]. In this study, SMOTE was systematically applied to mitigate class imbalance challenges when vulnerable samples constituted a significantly smaller proportion of the dataset than benign samples.

4. Results

4.1. Model Selection

In this study, the performances of three machine learning models, specifically Random Forest, XGBoost, and LightGBM, were systematically compared to determine the optimal approach for vulnerability detection. The critical issue of data imbalance was addressed by systematically applying the class weighting mechanisms inherent to each model throughout the comparative experiments.

For the Random Forest model, the class_weight = balanced option was applied to automatically adjust the class weights, while the LightGBM model utilized the is_unbalance = True setting to enable automatic correction for class imbalance. In contrast, XGBoost lacks a built-in mechanism for automatically handling class imbalances, necessitating a manual approach to parameter tuning. Consequently, the scale_pos_weight parameter was incrementally adjusted in steps of 0.1 to identify the optimal weighting strategy. The experimental results revealed a nuanced performance trend: as the scale_pos_weight value increased, the precision gradually decreased while the recall consistently improved. The comprehensive performance metrics are documented in Table 3, with a scale_pos_weight value of 2.9, ultimately selected as the optimal configuration due to the maximization of recall. This study used training data comprising three variables: parameters, values, and labels. The parameter-value pairs were vectorized using BERT embedding and used as the input features. As shown in Table 3, 5-fold cross-validation was applied to train and evaluate the model to improve its reliability.

Table 3. Cross-validated results of XGBoost classification performance.

Scale_Pos_Weight	Accuracy	Precision	Recall	F1-Score	AUC-ROC
1.9	0.7359 ± 0.0046	0.5996 ± 0.0061	0.8266 ± 0.0207	0.6949 ± 0.0067	0.7776 ± 0.0071
2.0	0.7348 ± 0.0074	0.5975 ± 0.0097	0.8338 ± 0.0142	0.6960 ± 0.0050	0.7771 ± 0.0071
2.1	0.7337 ± 0.0047	0.5960 ± 0.0072	0.8346 ± 0.0158	0.6952 ± 0.0020	0.7770 ± 0.0068
2.2	0.7335 ± 0.0053	0.5955 ± 0.0074	0.8366 ± 0.0118	0.6957 ± 0.0025	0.7769 ± 0.0056
2.3	0.7362 ± 0.0051	0.5974 ± 0.0074	0.8450 ± 0.0166	0.6998 ± 0.0041	0.7764 ± 0.0060
2.4	0.7354 ± 0.0053	0.5957 ± 0.0072	0.8510 ± 0.0121	0.7008 ± 0.0033	0.7778 ± 0.0068
2.5	0.7334 ± 0.0060	0.5931 ± 0.0074	0.8530 ± 0.0139	0.6996 ± 0.0050	0.7769 ± 0.0072
2.6	0.7345 ± 0.0076	0.5941 ± 0.0093	0.8562 ± 0.0092	0.7014 ± 0.0049	0.7759 ± 0.0066
2.7	0.7331 ± 0.0067	0.5925 ± 0.0082	0.8554 ± 0.0094	0.7000 ± 0.0040	0.7769 ± 0.0073
2.8	0.7329 ± 0.0055	0.5919 ± 0.0071	0.8590 ± 0.0099	0.7008 ± 0.0021	0.7772 ± 0.0066
2.9	0.7327 ± 0.0076	0.5911 ± 0.0088	0.8630 ± 0.0092	0.7015 ± 0.0056	0.7772 ± 0.0062
3.0	0.7321 ± 0.0054	0.5905 ± 0.0072	0.8626 ± 0.0135	0.7010 ± 0.0023	0.7763 ± 0.0053

Figure 5 illustrates the variation in recall as the scale_pos_weight value is systematically modified. The graph demonstrates how the recall progressively increases with incremental adjustments to the scale_pos_weight parameter, representing the evolving sensitivity of the model to the minority class.

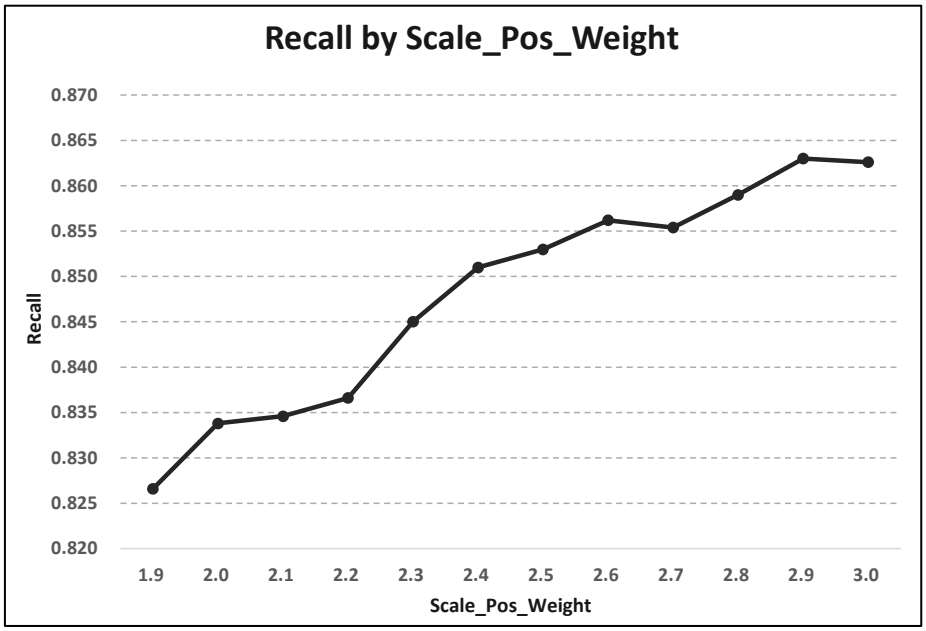

Figure 5. Recall by scale_pos_weight (XGBoost).

Following a comprehensive comparative evaluation that incorporated both class weighting and SMOTE techniques, XGBoost demonstrated a superior performance and was selected as the final detection model. The performance metrics for all three models are comprehensively summarized in Table 4 and are illustrated in Figure 6.

Table 4. Comparison of cross-validated classification performance results by model (with SMOTE and class weight adjustment).

Model	Dataset	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Random Forest	Original (No Weighting)	0.6990 ± 0.0131	0.6158 ± 0.0232	0.4598 ± 0.0336	0.5260 ± 0.0274	0.7840 ± 0.0055
LightGBM		0.7082 ± 0.0118	0.6006 ± 0.0162	0.5923 ± 0.0278	0.5961 ± 0.0194	0.7813 ± 0.0065
XGBoost		0.7020 ± 0.0082	0.5940 ± 0.0127	0.5743 ± 0.0224	0.5837 ± 0.0139	0.7748 ± 0.0088
Random Forest	Original (With Weighting)	0.7169 ± 0.0059	0.5840 ± 0.0076	0.7737 ± 0.0214	0.6654 ± 0.0080	0.7844 ± 0.0043
LightGBM		0.7350 ± 0.0062	0.6093 ± 0.0078	0.7585 ± 0.0163	0.6757 ± 0.0081	0.7814 ± 0.0075
XGBoost		0.7426 ± 0.0036	0.6092 ± 0.0057	0.8174 ± 0.0163	0.6980 ± 0.0046	0.7775 ± 0.0077
Random Forest	SMOTE (No Weighting)	0.7147 ± 0.0039	0.5814 ± 0.0062	0.7741 ± 0.0158	0.6639 ± 0.0033	0.7855 ± 0.0039
LightGBM		0.7287 ± 0.0094	0.6054 ± 0.0117	0.7321 ± 0.0246	0.6626 ± 0.0132	0.7818 ± 0.0076
XGBoost		0.7273 ± 0.0101	0.6047 ± 0.0117	0.7245 ± 0.0254	0.6590 ± 0.0146	0.7781 ± 0.0064
Random Forest	SMOTE (With Weighting)	0.7147 ± 0.0039	0.5814 ± 0.0062	0.7741 ± 0.0158	0.6639 ± 0.0033	0.7855 ± 0.0039
LightGBM		0.7287 ± 0.0094	0.6054 ± 0.0117	0.7321 ± 0.0246	0.6626 ± 0.0132	0.7818 ± 0.0076
XGBoost		0.7327 ± 0.0076	0.5911 ± 0.0088	0.8630 ± 0.0092	0.7015 ± 0.0056	0.7772 ± 0.0062

We computed the mean recall with 95% confidence intervals for each setting to assess model reliability. All configurations showed stable performance, with standard deviations ≤ 0.038 (Table 5). XGBoost demonstrated superior consistency across all settings, with the SMOTE and class weighting combination achieving optimal performance: mean recall of 0.863 (Std Dev = 0.010, 95% CI: 0.850–0.876).

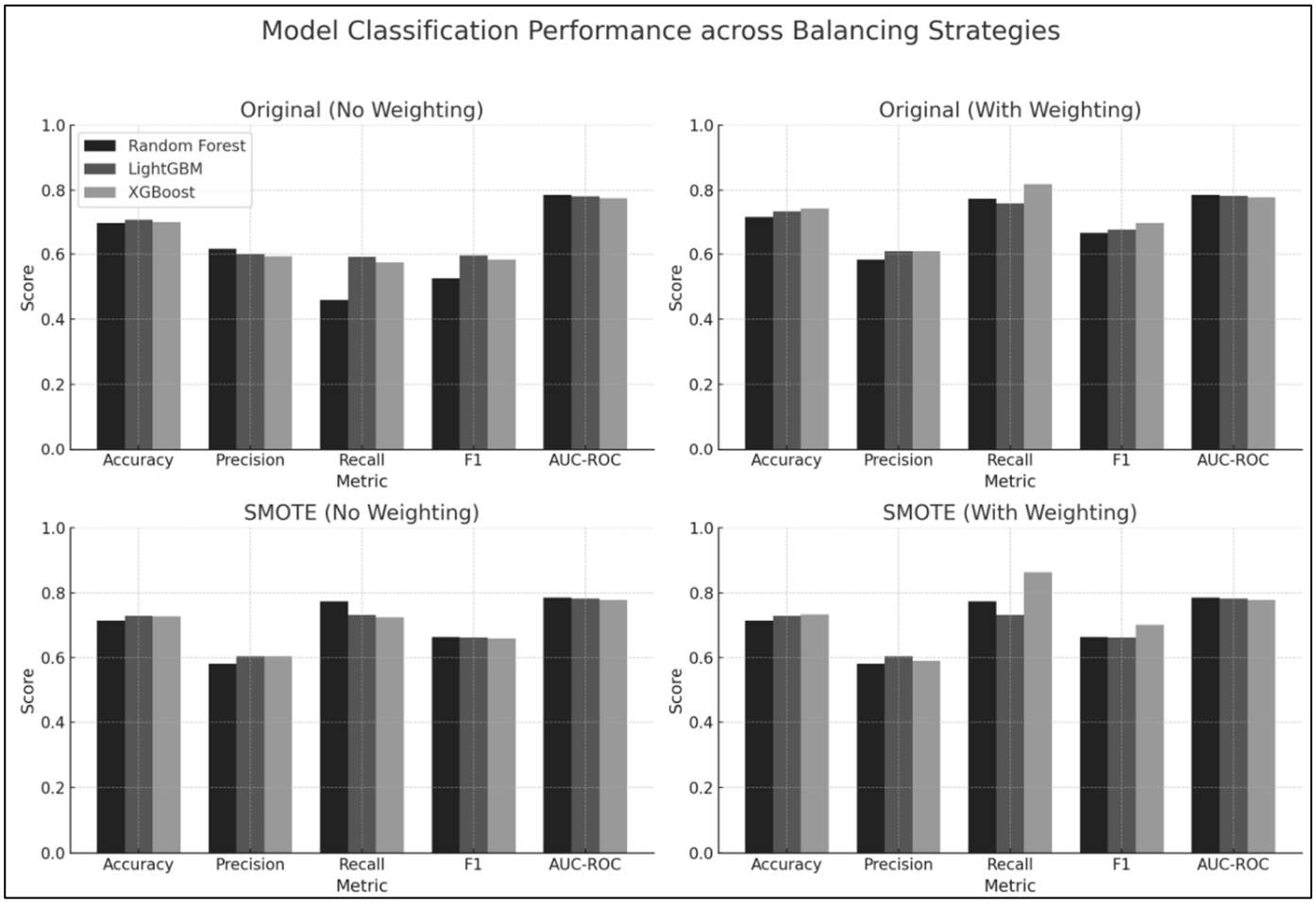

Figure 6. Cross-validated model classification performance across balancing strategies.

Table 5. Evaluation of model reliability across balancing strategies.

Model	Dataset	Mean Recall	Std Dev	95% CI Low	95% CI High	Interpretation
Random Forest	Original (No Weighting)	0.460	0.038	0.413	0.506	Stable
LightGBM		0.592	0.031	0.554	0.631	Stable
XGBoost		0.574	0.025	0.543	0.605	Highly stable
Random Forest	Original (With Weighting)	0.774	0.024	0.744	0.804	Highly stable
LightGBM		0.758	0.018	0.736	0.781	Highly stable
XGBoost		0.817	0.018	0.795	0.840	Highly stable
Random Forest	SMOTE (No Weighting)	0.774	0.018	0.752	0.796	Highly stable
LightGBM		0.732	0.028	0.698	0.766	Highly stable
XGBoost		0.724	0.028	0.689	0.760	Highly stable
Random Forest	SMOTE (With Weighting)	0.774	0.018	0.752	0.796	Highly stable
LightGBM		0.732	0.028	0.698	0.766	Highly stable
XGBoost		0.863	0.010	0.850	0.876	Highly stable

The primary hyperparameters of the XGBoost model were meticulously tuned, with the number of boosting iterations set to 300, the learning rate adjusted to 0.02, and tree depth limited to 6. Additional configurations included subsampling ratios of 0.8 for both subsample and colsample_bytree to mitigate the overfitting risk. To ensure experimental reproducibility, the model was trained using the logloss evaluation metric with a random seed fixed at 42. To address the class imbalance in the dataset, the scale_pos_weight parameter was set to 2.9 for the final model training. The training dataset was constructed from previously documented cases of vulnerability. Because confirming vulnerabilities requires response analysis, this study used a risk-oriented approach to maximize potential threat detection.

Because undetected vulnerabilities pose greater security risks than false positives in cybersecurity applications, the model training strategy prioritizes recall over precision. Recall was prioritized to enhance the detection sensitivity and ensure robust protection against parameter tampering.

4.2. Detection Results of Parameter Tampering Vulnerabilities

In this study, the vulnerability detection capabilities of the trained machine learning model were evaluated using 30 parameter tampering cases systematically selected from penetration testing reports, open-source forums, and e-commerce platforms. The detection methodology involved inputting GET-based web URLs containing these vulnerabilities into the model, with a methodical approach to calculating the vulnerability probability for each parameter in the URL. The detection mechanism was configured to estimate the vulnerability likelihood associated with each parameter using a carefully selected detection threshold of 0.3 to balance between identifying potential risks and minimizing false-positive results. A comparative analysis was conducted using an open-source automated tool to evaluate its vulnerability identification capabilities in open-source bulletin boards and e-commerce systems.

The experimental results revealed notable differences in the detection performance. The open-source automated tool showed significant constraints, with limited success in identifying parameter-tampering vulnerabilities. In comparison, the developed machine learning model demonstrated more promising results, successfully identifying 22 out of the 30 parameter tampering vulnerabilities, corresponding to a detection rate of 73.3 percent. These findings suggest the model's potential as a valuable approach for improving automated security vulnerability detection, although further refinement and validation would be beneficial.

5. Conclusions

This study presents a novel machine learning-based approach for detecting parameter tampering vulnerabilities in web applications, effectively addressing the limitations of existing pattern-based automated diagnostic tools. The proposed detection model was designed to identify parameter manipulation attacks that often evade conventional systems.

The methodology involved systematic data collection from diverse sources, including web crawlers, Exploit-DB repositories, and penetration testing documentation, and addressed class imbalance using the SMOTE technique. Model training and evaluation employed 5-fold cross-validation protocols to enhance the generalization performance and ensure statistical reliability. The performance stability and consistency were validated using statistical analysis. Among the tested models, XGBoost demonstrated superior performance and was selected for deployment. The trained model analyzes GET-based web URLs and generates probabilistic vulnerability assessments for each parameter by applying a fixed threshold for the final classification. The experimental results showed a detection rate of 73.3%, outperforming existing open-source tools.

Unlike traditional methods that rely on static input substitution, this study introduces a context-aware strategy that examines parameter–value relationships and applies semantic modifications for detecting vulnerabilities. By learning from real-world vulnerability data, the model can detect analogous attack patterns without depending on rule-based heuristics and prioritize high-risk parameters through semantic analysis. The proposed framework enables rapid, automated, and scalable vulnerability assessments in large-scale web environments where manual inspection is infeasible. The system's adaptability to diverse input formats and implementation-specific behaviors further enhances its applicability to real-world enterprise security scenarios.

This study focused on HTTP GET requests, which constrained the dataset diversity and represented a limitation. Additionally, as parameter security behavior may vary depending on context-specific processing mechanisms (e.g., security filters), the model emphasizes response-based vulnerability confirmation, with recall optimization prioritized during training and evaluation.

Future research will explore the application of advanced transformer-based models, such as RoBERTa, ALBERT, and DeBERTa, alongside the BERT embedding technique used in this work. Planned extensions include support for additional request formats (POST, JSON, XML), broader vulnerability categories, response-based verification mechanisms, and enhanced data augmentation strategies to improve detection robustness.

Author Contributions: Conceptualization, S.Y.Y. and N.-W.C.; methodology, S.Y.Y.; validation, S.Y.Y. and N.-W.C.; formal analysis, S.Y.Y.; investigation, N.-W.C.; resources, S.Y.Y.; data curation, S.Y.Y.; writing—original draft preparation, S.Y.Y.; writing—review and editing, N.-W.C.; visualization, S.Y.Y.; supervision, N.-W.C.; project administration, N.-W.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data used in this study contain real-world security vulnerability information and are, therefore, not publicly available due to confidentiality and ethical restrictions.

Acknowledgments: This research was supported by the Seoul National University of Science and Technology.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Truong, T.C.; Diep, Q.B.; Zelinka, I. Artificial Intelligence in the Cyber Domain: Offense and Defense. *Symmetry* **2020**, *12*, 410. [CrossRef]
2. Chen, C.; Cui, J.; Qu, G.; Zhang, J. Write+Sync: Software Cache Write Covert Channels Exploiting Memory-Disk Synchronization. *IEEE Trans. Inf. Forensics Secur.* **2024**, *19*, 8066–8078. [CrossRef]
3. Sun, G.; Li, Y.; Liao, D.; Chang, V. Service Function Chain Orchestration Across Multiple Domains: A Full Mesh Aggregation Approach. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1175–1191. [CrossRef]
4. Sarker, I.H.; Abushark, Y.B.; Alsolami, F.; Khan, A.I. IntruDTree: A Machine Learning Based Cyber Security Intrusion Detection Model. *Symmetry* **2020**, *12*, 754. [CrossRef]
5. Alhogail, A.; Alkahtani, M. Automated Extension-Based Penetration Testing for Web Vulnerabilities. *Procedia Comput. Sci.* **2024**, *238*, 15–23. [CrossRef]
6. Tóth, R.; Bisztray, T.; Erdődi, L. LLMs in Web Development: Evaluating LLM-Generated PHP Code Unveiling Vulnerabilities and Limitations. In *Computer Safety, Reliability, and Security. SAFECOMP 2024 Workshops*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2024; Volume 13942, pp. 425–437. [CrossRef]
7. Bisht, P.; Hinrichs, T.; Skrupsky, N.; Bobrowicz, R.; Venkatakrishnan, V.N. NoTamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications. In Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 2010), Chicago, IL, USA, 4–8 October 2010; pp. 607–618. [CrossRef]
8. Deepa, G.; Thilagam, P.S.; Praseed, A.; Pais, A.R. DetLogic: A Black-Box Approach for Detecting Logic Vulnerabilities in Web Applications. *J. Netw. Comput. Appl.* **2018**, *109*, 89–109. [CrossRef]
9. Aslam, S.; Aslam, H.; Manzoor, A.; Chen, H.; Rasool, A. AntiPhishStack: LSTM-Based Stacked Generalization Model for Optimized Phishing URL Detection. *Symmetry* **2024**, *16*, 248. [CrossRef]
10. Bao, Q.; Wei, K.; Xu, J.; Jiang, W. Application of Deep Learning in Financial Credit Card Fraud Detection. *J. Econ. Theory Bus. Manag.* **2024**, *1*, 51–57. [CrossRef]
11. Kshetri, N.; Kaur, N.; Kumar, D.; Osama, O.F.; Hutson, J. AlgoXSSF: Detection and Analysis of Cross-Site Request Forgery (XSRF) and Cross-Site Scripting (XSS) Attacks via Machine Learning Algorithms. In Proceedings of the 12th International Symposium on Digital Forensics and Security (ISDFS 2024), Istanbul, Türkiye, 22–23 April 2024; IEEE: Piscataway, NJ, USA, 2024. [CrossRef]
12. Castagnaro, A.; Conti, M.; Pajola, L. Offensive AI: Enhancing Directory Brute-Forcing Attack with the Use of Language Models. In Proceedings of the 2024 Workshop on Artificial Intelligence and Security (AISec'24), Salt Lake City, UT, USA, 14–18 October 2024; ACM: New York, NY, USA, 2024. [CrossRef]
13. An, J.H.; Wang, Z.; Joe, I. A CNN-Based Automatic Vulnerability Detection. *EURASIP J. Wirel. Commun. Netw.* **2023**, *2023*, 41. [CrossRef]
14. Fu, C.; Liu, G.; Yuan, K.; Wu, J. Nowhere to H^2IDE: Fraud Detection From Multi-Relation Graphs via Disentangled Homophily and Heterophily Identification. *IEEE Trans. Knowl. Data Eng.* **2024**, *36*, 1081–1094. [CrossRef]
15. Xu, Y.; Ding, L.; He, P.; Lu, Z.; Zhang, J. Meta: A Memory-Efficient Tri-Stage Polynomial Multiplication Accelerator Using 2D Coupled-BFUs. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2024**, *71*, 541–554. [CrossRef]
16. OWASP Foundation. Web Parameter Tampering. OWASP. Available online: https://owasp.org/www-community/attacks/Web_Parameter_Tampering (accessed on 6 June 2025).
17. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186. [CrossRef]
18. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794. [CrossRef]
19. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic Minority Over-Sampling Technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]