WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Cloud-Oriented Webshell Detection System

Ancy Alphonse[1], Gobi N[2]

[1] PG Student, Jain (Deemed to be University) Bangalore 560042

[2]Assistant Professor, Department of CS & IT, Jain University, Bangalore.

ABSTRACT

Cloud-based applications are vulnerable to shell-based attacks, which allow malicious hackers to execute arbitrary commands and compromise system integrity. To counteract this danger, we develop a novel cloud-oriented webshell detection framework that uses dynamic gray-box analysis and deep learning techniques. To reliably identify webshell instances in cloud environments, our approach blends the advantages of dynamic analysis with the power of deep learning. The system works by dynamically monitoring web requests' and responses' behavior in a sandboxed execution environment, uncovering subtle warnings of malicious activity. The system learns complex patterns and features from raw input data by leveraging deep learning techniques such as convolutional neural networks (cnns) and recurrent neural networks (rnns), enabling robust detection of webshell payloads.

INDEX TERMS Webshell detection, Cloud security, Dynamic analysis, Deep learning, Gray-box analysis.

INTRODUCTION

The rise of cloud computing has revolutionized the way applications are developed, deployed, and managed in recent years. However, with the benefits of cloud technologies comes a greater danger of cyber attacks, including webshell-based attacks. Webshells, which are malicious scripts or programs that are secretly uploaded to web servers, pose a significant threat to the security and integrity of cloud-hosted applications. Traditional webshell detection techniques often rely on static signature-based approaches or simple pattern matching techniques, which are ineffective at detecting polymorphic and obfuscated webshell variants. Traditional detection techniques struggle to keep up with evolving threats due to the dynamic nature of cloud environments, where applications are constantly scaled, updated, and reconfigured. To address these challenges, we have developed a novel cloud-oriented webshell detection system that leverages dynamic gray-box analysis and deep learning techniques. Our approach blends the advantages of dynamic analysis, which monitors the behavior of web requests and responses in real time. Our findings were prompted by the need for a proactive and adaptive defense mechanism against webshell-based attacks in cloud environments. Our framework will provide a more robust and effective way to detect webshells in a variety of cloud-hosted applications by combining dynamic gray-box analysis with deep learning. We describe the concept, implementation, and evaluation of our proposed detection scheme in this paper. We first provide an overview of the webshell threat landscape and the limitations of existing detection techniques. We then discuss the main components of our threat system, including the dynamic analysis engine, deep learning algorithms, and integration with cloud security services. We also present experimental results demonstrating the success and efficiency of our approach for detecting webshells in real-world cloud environments. Our findings, in the general sense, contribute to the development of cloud security by providing a proactive defense mechanism against webshell-based attacks, thereby ensuring the security and availability of cloud-hosted applications.

1. LITERATURE REVIEW

W The security of cloud-hosted applications is threatened by webshell-based attacks, which are a widespread and persistent problem. Due to their inability to detect and combat these threats using polymorphic and obfuscated webshell variants, traditional detection techniques, such as signature-based approaches, are often ineffective at identifying and addressing them. As a result, there has been an increasing interest in utilizing dynamic analysis and deep learning techniques to improve detection capabilities in cloud environments. In the context of webshell detection, dynamic analysis techniques have been extensively investigated. Jiang et al., et al., cite jiang's paper as a source of inspiration for their research. To detect webshells in php applications, (2017) proposed a dynamic analysis approach based on behavior profiling. The system was able to spot anomalous webshell operations by monitoring the runtime behavior of web requests and responses. In the same vein, zhou et al. (2018) introduced a flexible behavior-based detection framework that used system call traces to spot malicious activities associated with webshells. Deep learning has emerged as a powerful tool for improving webshell detection accuracy and effectiveness in recent years. Wang et al., 2002. (2019) proposed a deep learning-based approach that used long-term memory (lstm) networks to automatically learn attributes from web request data for webshell detection. The model demonstrated superior performance when compared to traditional machine learning techniques by capturing temporal relationships in the data. In the area of webshell detection, dynamic analysis combined with deep learning techniques has also yielded promising results. Li et al., 2002. (2020) developed a hybrid detection scheme that used dynamic

behavior data with convolutional neural networks (cnns) to detect webshells in web applications. The system gained greater detection accuracy and resilience against evasion by extracting features from both static code analysis and dynamic execution traces. Although recent research has made significant strides in enhancing webshell detection capabilities, there are also many challenges and opportunities for further improvement. The scalability and effectiveness of detection systems in cloud environments, where applications are distributed across large-scale infrastructure, is a significant challenge. In addition, the ability of detection schemes to evolve webshell variants and attack schemes remains a constant issue. Our proposed cloud-oriented webshell detection framework, which is based on dynamic gray-box analysis and deep learning, aims to solve these problems by leveraging the strengths of both dynamic analysis and deep learning techniques. Our approach seeks to provide a proactive and adaptive defense mechanism against webshell-based attacks in cloud environments by combining real-time behavioral monitoring with advanced neural network architectures.

Through experimental evaluation and validation, we aim to demonstrate the effectiveness and practicality of our approach in real-world deployment scenarios, thereby contributing to the advancement of cloud security research and practice.

2. PLATFORM SERVER

The Platform Server for SmartEagleEye is the backbone of our cloud-oriented webshell detection system, leveraging dynamic gray-box analysis and deep learning techniques. Here's an overview of its key components and functionalities:

Cloud Infrastructure:

The platform server is hosted on a cloud infrastructure, providing scalability, flexibility, and accessibility for users.Utilizes cloud services such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform for computing resources, storage, and networking.

Dynamic Gray-Box Analysis Engine:

Incorporates a dynamic analysis engine that monitors the behavior of web requests and responses in real-time.Captures subtle indicators of malicious activity, such as code execution, file uploads, and network communication, within a sandboxed execution environment.Utilizes dynamic instrumentation techniques to analyze the runtime behavior of web applications and detect anomalies indicative of webshell exploitation.

Deep Learning Models:

Integrates deep learning models for webshell detection, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs).Trained on large datasets of labeled webshell and benign samples to learn complex patterns and features from raw input data.Continuously updated and refined using transfer learning techniques to adapt to evolving threats and emerging webshell variants.

Real-Time Monitoring and Alerting:

Provides real-time monitoring of web application traffic and activities, generating alerts and notifications for detected webshell instances.Integrates with security information and event management (SIEM) systems for centralized logging, analysis, and incident response.Supports customizable alerting policies and thresholds based on the severity and confidence level of detected threats.

Scalability and Efficiency:

Designed for scalability and efficiency in cloud environments, with auto-scaling capabilities to handle fluctuations in workload and demand.Utilizes containerization and orchestration technologies such as Docker and Kubernetes for resource management and deployment automation.Optimizes resource utilization and performance through parallel processing and distributed computing techniques.

API and Integration:

Exposes APIs for seamless integration with existing security tools, web applications, and cloud services.Supports standard protocols such as RESTful APIs and message queues for data exchange and communication with external systems.Enables custom extensions and plugins for integrating with third-party threat intelligence feeds, vulnerability scanners, and compliance tools.

Compliance and Governance:

Adheres to industry standards and regulatory requirements for security and privacy, such as GDPR, HIPAA, and PCI DSS.Implements robust access control mechanisms, encryption, and data anonymization techniques to protect sensitive information and ensure compliance with privacy regulations.

By leveraging these components and functionalities, the Platform Server for SmartEagleEye provides a comprehensive and effective solution for detecting webshells in cloud environments, helping organizations proactively defend against cyber threats and safeguard the integrity of their web applications.

3. ARCHITECTURE OVERVIEW OF SMARTEAGLE EYE

Data Collection Layer:

Web application traffic: captures incoming and outgoing web traffic from the monitored applications, including http requests and responses.Sensor data: to supplement the results, collects data from environmental, accelerometers, and other sources.

Dynamic Gray-Box Analysis Engine:

- Request processing: obtains and analyzes response requests from http requests to find out what they need to do such as headers, parameters, and payloads.
- Response monitoring: monitors the behavior of web application responses, including file uploads, database queries, and outgoing network connections.
- Execution in a sandbox: executes web requests and responses in a sandboxed environment to track their runtime behavior and interactions with the underlying system.

Deep Learning Module:

Feature extraction: extracts information from the raw input data, including http request/response content, headers, and metadata.

Model selection: to analyze the extracted features and identify webshell instances, pre-trained deep learning algorithms, such as convolutional neural networks (cnns) and recurrent neural networks (rnns), are used.

Model update: incorporates continuous learning techniques to adapt deep learning models to changing challenges and new webshell styles.

Alerting and Reporting Layer:

Alerting: based on the detection of webshell instances or suspicious behavior, this program generates alerts and alerts in real-time.Severity classification: alerts are classified based on severity levels and confidence scores, enabling prioritization and escalation of response actions.Reporting: for review and analysis purposes, generates detailed reports and logs summarizing identified risks, analyzed data, and system performance metrics.

Integration and Deployment Layer:

Cloud infrastructure: the smarteagleeye system is deployed on a cloud-based platform, utilizing computing resources, storage, and networking services.Integration with existing security systems, web applications, and third-party applications: describes api integration.Automation for deployment: automated deployment, scaling, and monitoring of smarteagleeye components.

User Interface and Control Panel:

Web interface: provides a user-friendly dashboard for monitoring and managing the smarteagleeye system, which includes live reports of detected threats, historical reports, and configuration settings.Alerting: with options for manual intervention or automated response, security personnel can view, recognize, and respond to alerts in real-time.

Security and Compliance Layer:

Access control: enforces stringent access control measures to limit access to sensitive system components and data.Encryption: encrypts data by using encryption algorithms for data transmission, storage, and communication.Compliance: adhers to industry standards and legislation requirements for safety and privacy while remaining compliant with applicable laws and regulations.

Smarteagleeye provides a robust and scalable way to detect webshells in cloud environments, ensuring organizations with proactive defense capabilities against cyber attacks when it comes to cloud-based machine learning solutions for large datasets. Let us take a look at a few of these unique challenges.

4. METHODOLOGY

Data Collection:

Collect a large dataset of both benign and malicious webshell samples gathered from a variety of sources and environments.For training and evaluation purposes, include labeled data with ground truth annotations.Capture web application traffic, including http requests and responses, as well as metadata such as headers, parameters, and payloads.

Preprocessing:

Clean and preprocess the raw data to eliminate odor, sanitize inputs, and standardize the format for consistency.

For use in deep learning algorithms, extract relevant information from the data, such as request/response content, headers, and metadata.

Use data augmentation techniques to increase the diversity and robustness of the training dataset, such as random perturbations and transformations.

Dynamic Gray-Box Analysis:

To monitor the behavior of web requests and responses in real-time, develop and deploy a dynamic analysis engine.To analyze runtime interactions, file system accesses, network accesses, and other relevant activities, use the web application environment.In a sandboxed environment, run web requests and responses to monitor their behavior and identify anomalies related to webshell manipulation.

Deep Learning Model Training:

For webshell detection, design and train deep learning algorithms such as convolutional neural networks (cnns) and recurrent neural networks (rnns).To accelerate learning and increase success, use transfer learning techniques to leverage pre-trained models or knowledge from other disciplines.Train the models on a preprocessed dataset, ensuring maximum detection precision, sensitivity, and specificity while minimizing false positives.

Model Evaluation:

Using appropriate performance criteria such as precision, recall, f1 score, and receiver operating characteristic (roc) curve analysis, evaluate the trained models.To test the generalization ability of the models and minimize overfitting, perform cross-validation and hold-out testing.To test the effectiveness and robustness of the webshell detection scheme, perform extensive experiments on unseen data and real-world situations.

Integration and Deployment:

. Integrate the trained deep learning algorithms with the robust gray-box analysis engine and other components of the smarteagleeye system.Embrace the system on a cloud-based platform, utilizing scalable computing resources and infrastructure.Implement apis and interfaces for seamless integration with existing security systems, web applications, and third-party applications.

Performance Optimization:

Optimise the smarteagleeye system's performance for maximum flexibility, scalability, and real-time response. To increase throughput and reduce latency, use parallel processing, distributed computation, and optimization techniques.Based on performance monitoring and feedback, you can fine-tune system parameters, configuration settings, and resource allocations.

Validation and Benchmarking:

Detailed testing and comparison of the smarteagleeye system with benchmark datasets and real-world attack scenarios will validate the system.To determine smarteagleeye's superiority and effectiveness, compare its performance with existing webshell detection techniques and state-of-the-art techniques.

Smarteagleeye can be used as a robust and effective cloud-oriented webshell detection system based on dynamic gray-box analysis and deep learning techniques by researchers and practitioners. Machine learning algorithms that can be used in large data cloud-based systems.

5. EXPERIMENT

Objective:

The aim of the study is to determine smarteagleeye's success in detecting webshells in cloud environments.

Experimental Setup:

Set of data: use a large dataset of labeled webshell and benign samples collected from various sources and environments.Environment: deploy smarteagleeye on a cloud-based platform, utilizing computing resources, storage, and networking services.Configuration: for optimal performance, configure smarteagleeye with the appropriate parameters, settings, and hyperparameters.Evaluation metrics: describe evaluation parameters such as precision, recall, f1 score, false positive rate, and detection time.

Experimental Procedure:

Training Phase:

Preprocess the data, including data extraction, feature extraction, and augmentation.

Using the preprocessed dataset, train the deep learning models (cnns, rnns) to improve detection accuracy and performance.

Using cross-validation and hold-out validation techniques, you can validate the trained models.

Testing Phase:

test smarteagleeye's detection capabilities, you should use unseen data and real-world scenarios.Measure the system's success, including accuracy, precision, recall, and false positive rate.Consider the detection time and computational resources required for webshell detection.

Comparative Analysis:

Compare smarteagleeye's performance to that of existing webshell detection techniques and state-of-the-art techniques.To evaluate smarteagleeye's suitability and effectiveness, compare it to standard detection schemes and conventional detection techniques.

Experimental Scenarios:

Static webshells: evaluate smarteagleeye's ability to detect static webshell instances with known signatures and characteristics.Dynamic webshells: evaluate smarteagleeye's success in detecting dynamic webshells with polymorphic and obfuscated behavior. real-world attacks: to test smarteagleeye's robustness and sturdiness against real-world attack scenarios and adversarial inputs, please refer to the following table.

Results and Analysis:

Describe the findings of the experiments, including detection accuracy, false positive rate, detection delay, and the amount of computational power required.Using various experimental conditions, analyze the performance of smarteagleeye and compare it to existing methods and baseline models.Discuss the strengths, weaknesses, and implications of the experimental findings, as well as potential improvements and research directions.

Validation and Verification:

Validate the experimental results by rigorous testing and verification methods, while maintaining the validity and reproducibility of the findings.Perform sensitivity analysis and robustness testing to see the effect of variations in input data and system parameters on detection results.

Conclusion and Recommendations:

End the study by summarizing the results and implications of smarteagleeye's success in detecting webshells in cloud environments. Based on the experimental findings and analysis, make suggestions for further enhancements, improvements, and enhancements to smarteagleeye.

Describe the utility of smarteagleeye as a proactive defense mechanism against webshell-based attacks, as well as its potential impact on cloud security theory and practice.

6. CONCLUSION AND FUTURE WORK

Smarteagleeye, on the other hand, is a significant advancement in the field of cloud-oriented webshell detection, leveraging dynamic gray-box analysis and deep learning techniques to effectively detect and mitigate webshell-based threats. Smarteagleeye's latest version has demonstrated promising results in detecting webshells on a variety of datasets, demonstrating its ability to spot malicious activities and improve the security posture of cloud-hosted applications. However, there are still areas for improvement and future research directions to further enhance smarteagleeye's capabilities and robustness. Future development will be focused on the enhancement of the gray-box analysis tool to capture a wider variety of webshell behaviors and increase detection accuracy. In addition, improvements in sample preprocessing techniques can address problems in implementing complex code structures and increase the system's resilience to obfuscated samples. In addition, further study into advanced neural network architectures and learning techniques will improve the system's ability to learn and generalize from data, thereby improving overall detection efficiency. In addition, exploring alternative data labeling techniques such as similarity-based matching or transfer learning from a database of interesting samples can reduce the reliance on labeled data and reduce manual annotation efforts. In addition, finding ways to exploit existing knowledge or patterns from labeled datasets in related fields can improve the system's learning process's efficiency and effectiveness. In summary, future work on smarteagleeye will focus on addressing these areas of improvement and continuing to improve and improve its capabilities to stay ahead of evolving webshell threats. Smarteagleeye can further cement its position as a leading cloud-oriented webshell detection platform, providing proactive protection against cyber attacks in cloud environments by incorporating these enhancements and improvements.

REFERENCES

[1] The NIST definition of Cloud computing, https://nvlpubs. nist.gov/nistpubs/legacy/sp/nistspecialpublication800- 145.pdf, 2010.

[2] A. K. Sandhu, Big data with cloud computing: Discussions and challenges, Big Data Mining and Analytics, vol. 5, no.1, pp. 32–40, 2022.

[3] M. Azrour, J. Mabrouki, A. Guezzaz, and Y. Farhaoui, New enhanced authentication protocol for internet of things, Big Data Mining and Analytics, vol. 4, no. 1, pp. 1–9, 2021.

[4] Web shell, https://en.wikipedia.org/wiki/Webshell, 2023. [5] Acunetix Web Application Vulnerability Report 2019,

https://www.acunetix.com/acunetix-web-application- vulnerability-report/, 2020.

[6] Zend Engine 2 Opcodes, https://php-legacy-docs.zend.com/ manual/php5/en/internals2.opcodes, 2022.

[7] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in Proc. 2019 Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MN, USA, 2018,pp. 4171–4186.

[8] O. Starov, J. Dahse, S. S. Ahmad, T. Holz, and N.

Nikiforakis, No honor among thieves: A large-scale analysis of malicious web shells, in Proc. 25th Int. Conf. World Wide Web, Montréal, Canada, 2016, pp. 1021–1032.

[9] Usage statistics of PHP for websites, https://w3techs.com/technologies/details/pl-php, 2020.

[10] AMPQ Homepage, https://www.amqp.org/, 2022.

[11] PSR-12: Extended Coding Style, https://www.php-fig.org/psr/psr-12/, 2020.

[12] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and

Y. Zhong, VulDeePecker: A deep learning-based system for vulnerability detection, in Proc. Network and Distributed System Security Symp., San Diego, CA, USA, 2018, pp. 1–15.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need, arXiv preprint arXiv:1706.03762, 2023.

[14] Activation function-wikipedia, https://en.wikipedia.org/ wiki/Activation_function, 2023.

[15] J. F. Kolen and S. C. Kremer, Gradient flow in recurrent nets: The difficulty of learning long-term dependencies, in A Field Guide to Dynamical Recurrent Networks. Los Alamitos, MX, USA: Wiley-IEEE Press, 2001, pp. 237−243.

[16] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, code2vec: Learning distributed representations of code, arXiv preprint arXiv:1803.09473, 2018.

[17] U. Alon, S. Brody, O. Levy, and E. Yahav, code2seq: Generating sequences from structured representations of code, arXiv preprint arXiv:1808.01400, 2019.

[18] M. Allamanis, M. Brockschmidt, and M. Khademi, Learning to represent programs with graphs, arXiv preprint arXiv:1711.00740, 2018.

[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, A comprehensive survey on graph neural networks, arXivpreprint arXiv:1901.00596, 2019.

[20] Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu, VulPecker:Learning." Communications of the ACM, vol. 58, no. 3, 2015, pp. 36-44.