A Novel Approach to the Detection and Isolation of Stegomalware in PDFs Using Kolmogorov Complexity

***Abstract*—PDF-based stegomalware is a growing cybersecurity threat that disguises malicious payloads within the tree structure of PDF documents, exploiting features like embedded JavaScript, metadata, and open actions while maintaining the outward appearance of a legitimate file. Traditional detection methods (both heuristic and signature-based), often fail to identify stegomalware due to the external similarities between clean and infected documents.**

This project presents a novel method for detecting PDF-based stegomalware using Kolmogorov complexity. Kolmogorov complexity measures the shortest possible program to describe an output, but it cannot be computed directly. However, this project indirectly calculated complexity by analyzing the compressed lengths of attributes using compression algorithms that group repetitive (non-complex) patterns. By calculating the complexity of components within a PDF's internal tree structure and comparing them to baselines derived from known clean documents, malicious modifications can be identified by detecting deviations from expected values.

For testing, a local mail server was developed to process incoming PDFs by calculating and comparing Kolmogorov complexity to stored baselines of clean PDFs tagged with matching metadata IDs. PDFs with significant complexity increases within suspicious attributes are quarantined using Firejail, while others are safely delivered to the user.

Initial testing achieved a 100% true positive rate but an 86.2% false positive rate when analyzing the entire PDF as a whole. Refining the approach to analyze specific PDF components, focusing on suspicious changes, and ignoring innocent variations like file size or text content improved results to a 97.8% true positive rate and a 3.7% false positive rate.

***Index Terms*—Steganography, Kolmogorov Complexity, Malware Detection, PDF Security**

I. Introduction

Steganography, the practice of concealing information within other information, has become increasingly prevalent as a method for cybercriminals to distribute malicious software without suspicion [1]. By embedding harmful payloads within files such as images, audio, or documents, attackers can evade traditional security mechanisms. PDFs, as a widely used format in business and academia, are particularly susceptible to stegomalware. Given the universal use of PDFs, the potential for malware to be concealed within them poses a serious risk to individuals and organizations alike.

The HP Wolf Security Threat Insights Report (Q4 2023) highlighted a concerning trend in which malicious actors increasingly employed PDF documents as vehicles for delivering malware [2], [3]. The report found that PDF threats rose by 7% compared to earlier quarters, with malware such as WikiLoader, Ursnif, and DarkGate being spread via PDF files [2]. This rise in PDF-based threats highlights the need for working, reliable detection techniques capable of identifying both known and novel forms of stegomalware embedded in these types of files [4].

The infection chain of PDF-based Stegomalware typically takes one of two forms—either exploiting vulnerabilities within the PDF reader software itself, or presenting information to the human reader that convinces them to install more direct forms of malware [5]. Exploiting the vulnerability of a PDF reader is a more direct and reliable way for an attacker to gain access to a system, especially if they know that a company is using an unpatched, vulnerable, version of the software (which is often the case). On the other hand, getting the reader to download the malware themselves can make entering a system much easier. This can be done in a multitude of ways - a great example being an alert to update a PDF reader [6].". When the user opens the stegomalware, a popup from, lets say, Adobe, will ask them to update Acrobat. This would seem fairly standard to the user - after all, it's not as though anything out of the ordinary is happening. However, the hyperlinked button in the alert to update really has the user download a fake version of Acrobat, which compromises the system. Either way, hidden data is being added to the document that has no legitimate reason to be in the document - and that is what this project aims to detect.

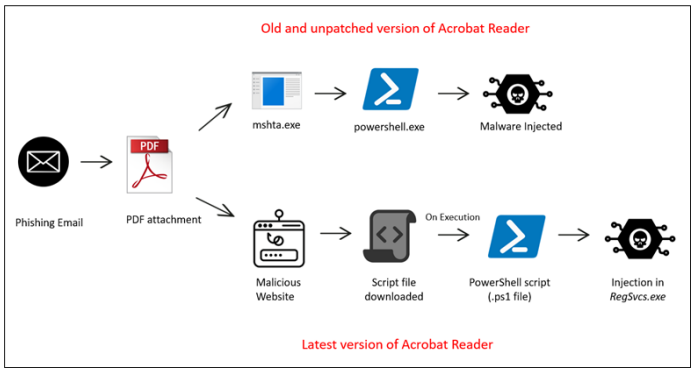

Fig. 1. Example infection chain [3].

II. Kolmogorov Complexity and PDF Structure

Kolmogorov complexity is the minimum amount of information required to represent an object, making it a useful

metric for finding complex, hidden code within a PDF. By calculating the complexity of different components of the PDF tree structure, focusing on suspicious fields like metadata, JavaScript, and embedded objects, baseline metrics can be established for clean, legitimate PDFs. These baseline metrics can then be used to compare against incoming documents to identify any unusual changes.

The tree structure of a PDF, as shown in Figure 3, represents the hierarchical organization of the components within a PDF. Each node in the tree corresponds to a different part of the document, such as the root, pages, or embedded JavaScript. By computing the Kolmogorov complexity for each subtree, anomalies can be detected that may indicate the presence of stegomalware. For instance, the inclusion of unexpected JavaScript or changes in specific nodes that do not typically vary (such as the /OpenAction node) can be flagged as suspicious.

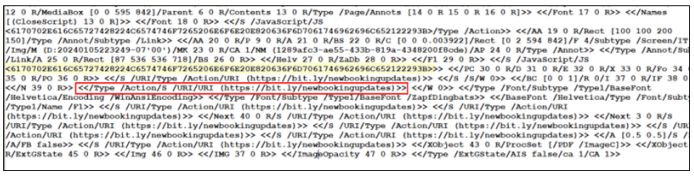

Fig. 2. Example embedded data within a PDF [3].

III. Applications in Business Environments

This approach is particularly suitable for businesses that frequently receive the same types of documents, such as invoices, contracts, or reports. By maintaining a repository of complexity metrics for clean documents, organizations can quickly compare new incoming PDFs against these baselines. While some components, like the number of pages, may naturally vary, other elements, such as JavaScript or specific metadata, should remain consistent. Any deviation in the complexity of these suspicious components could indicate an attempt to embed malicious content.

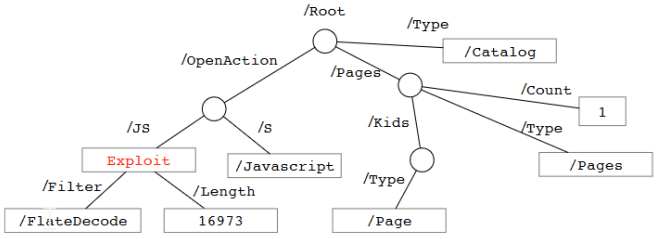

Fig. 3. The tree structure of a PDF malware [7].

IV. Solution/Methodology

By using Kolmogorov complexity approximation to detect stegomalware by analyzing deviation in the structural elements of PDFs, malicious pdfs can be accurately recognized.

A. Implementing Kolmogorov Complexity

Kolmogorov complexity is defined as the length of the shortest possible description of an object in a fixed universal description language. In simpler terms, it measures the minimal amount of information required to completely describe an object. However, when applied to computer science, calculating Kolmogorov complexity is provably impossible because it involves finding the shortest program that generates a given output [8]. The impossibility stems from the fact that there is no algorithm that can guarantee being able to identify the shortest program for all possible inputs.

However, by compressing an object and calculating its size, an upper bound of the complexity can be approximated. In this study, an individualized complexity calculation is used - approximating the complexity values of each part of the PDF tree. Compression algorithms, such as DEFLATE, work by identifying patterns and redundancies in data. The more patterns a data object contains, the more it can be compressed, leading to a shorter compressed length. By computing the length of compressed components of a PDF document, their Kolmogorov complexity is indirectly estimated. While not exact, this method works in that it is only necessary to find a difference between two complexity values - a blank clean sample document, and the incoming document.

The DEFLATE algorithm combines LZ77 compression and Huffman coding for efficient compression. It captures repeated sequences of data and encodes them with shorter representations. Thus, highly structured or repetitive data compresses more effectively, indicating lower Kolmogorov complexity, while random or highly unique data compresses less effectively, indicating higher complexity.

Furthermore, a library was created to extract the PDF tree, as none exist at the time of writing. This was done by manually parsing the internal structure of PDF files, analyzing the hierarchical relationships between objects, and constructing a tree representation. This approach directly processes the raw PDF byte structure, identifying and extracting key elements such as /Catalog, /Pages, /OpenAction, /Kids, and embedded /JavaScript objects. The parser recursively traverses the document, mapping object references and decoding compressed streams where necessary. This method allows for the precise extraction of individual PDF components, creating an entirely localized complexity analysis.

By integrating the PDF tree extraction with a compression-based complexity estimator, this study provides a novel approach to detecting anomalies in PDFs. Each structural component's Kolmogorov complexity is estimated by compressing its data, allowing for direct comparisons between a clean baseline document and an incoming document. Anomalous sections—such as embedded JavaScript, unusual metadata, or obfuscated payloads—exhibit higher complexity values due to reduced compressibility. This enables targeted identification of potential stegomalware or other hidden manipulations within the PDF structure.

B. Baseline Establishment

1) **Document Collection**: Identify common document types like forms, contracts, and reports frequently exchanged in an organization. These should have a focus on documents that are publicly accessible, and sent to the company. These are the documents that most attackers will hide behind. A great example is the attack on LMG (Linus Media Group), where an attacker sent a PDF file disguised as a filled sponsorship application form, and successfully stole session cookies, allowing several company accounts to be compromised. While this PDF differed from the stegomalware this paper focuses on (it used the Acrobat PDF logo, and used the file extension, but was actually a screensaver file with malicious code - a common Windows exploit), the principle remains the same.
2) **Component Analysis**: Extract key PDF tree components using PyPDF2 and PyMuPDF. PyPDF2 is effective in handling general PDF structure, extracting metadata, and working with raw object data. However, the support for deep content inspection, such as analyzing embedded scripts and object streams is weak. PyMuPDF (also known as Fitz), has a more powerful interface for working with the internal structure of PDFs, allowing the tool to extract and analyze specific embedded components, such as JavaScript, rich media, and OpenAction triggers. Both libraries are using together for a more comprehensive analysis of the PDF. Branches of the PDF tree that contain objects such as embedded JavaScript, open actions, etc. are stored separately, as they are the main target to be compared.
3) **Complexity Metrics**: Use zlib compression to compute baseline complexity for clean PDFs, recording attributes like metadata size and embedded object lengths. Metrics are stored and labeled with the metadata ID of the external document - when the document is sent back, filled, the metadata ID is matched and compared. If no metadata ID is found, the email is quarantined to be reviewed. This tool should only be used on email addresses whose purpose is to receive client documents, as that is the point of entry for most attacks on businesses.

C. Deviation Analysis

1) **Incoming Document Parsing**: Parse incoming PDFs, separating the pdf by its tree structue: sections with JS, OpenActions, metadata, etc. are focused, but all branches are stored.
2) **Complexity Calculation**: Use DEFLATE to compress all components, then calculate and save length for each component to estimate Kolmogorov complexity.
3) **Risk Scoring**: Weights are assigned to each section - JS, OpenActions, etc. require smaller thresholds from base to incoming, while typically innocent changes like length and page count have much larger thresholds, if any.
4) **Analysis**:The tool uses the following mathematical framework as a means of categorizing a PDF:
   - **Baseline Complexity:** Given a clean set of PDFs, a baseline Kolmogorov complexity for each component in a document's tree structure is established. Each document D is represented as a hierarchical structure:
   $$T(D) = \{e_1, e_2, ..., e_n\} \tag{1}$$
   where each e_i represents an element in the PDF tree (e.g., metadata, JavaScript, embedded objects).
   - **Kolmogorov Complexity Estimation:** Since Kolmogorov complexity $K(e_i)$ is incomputable, it is approximated using compression techniques. The complexity of each element e_i is estimated as:
   $$K(e_i) \approx C(e_i) \tag{2}$$
   where $C(e_i)$ is the compressed length of e_i using a lossless compression algorithm such as DEFLATE.
   - **Deviation Analysis:** When a new PDF D' is received, deviation is calculated $\Delta K(e_i)$ for each component e_i compared to the clean baseline:
   $$\Delta K(e_i) = K(e_i) - K(e_i)_{\text{baseline}} \tag{3}$$
   If $\Delta K(e_i) > \tau$, where τ is a threshold for anomaly detection, the element is flagged as suspicious.
   - **Anomaly Score Computation:** The overall anomaly score $S(D')$ for a PDF D' is computed as a weighted sum of deviations:
   $$S(D') = \sum_i w_i \Delta K(e_i) \tag{4}$$
   where w_i are weights giving higher importance to high-risk components such as embedded JavaScript.
   - **Classification:** The document D' is classified as malicious if:
   $$S(D') > \gamma \tag{5}$$
   where γ is the containment threshold set based on empirical testing.

D. Containment and Quarantine

1) **Quarantine**: Flagged documents are isolated using Firejail to prevent potential harm.
2) **Logging**: Logs are kept of each flagged anomaly for analysis.

E. Experimental Setup

To validate the system, a local network environment was configured with three machines:

1) A Raspberry Pi running a custom email server with the stegomalware detection software installed.
2) Two other machines were used to send and receive emails with attached PDF documents.

The setup allowed the simulation of real-world email flows, so the tool could be tested under realistic conditions. Clean and malicious PDFs were sent between the machines, and the Raspberry Pi scanned attachments in transit.

F. Creating and Testing Stegomalware

Stegomalware was generated in various ways to simulate realistic attack scenarios:

1) Embedding malicious JavaScript into PDF documents.
2) Modifying metadata and embedded objects to conceal payloads.
3) Using actions and triggers within the PDF tree to execute malicious code upon opening.

The system logged and quarantined flagged documents, allowing for detailed analysis of detected anomalies and false positives.

V. Results

A. Experimental Setup

Testing was conducted using 600 PDFs, with 100 clean documents and 500 modified to include stegomalware in five different ways. Stegomalware injection was done semi-randomly, with all methods being used at least once for each clean document, and at least one document with multiple peices of embedded stegomawalre. The local mail server intercepted and analyzed PDFs.

B. Detection Accuracy

- Initial whole-PDF analysis achieved a 100% true positive rate but an unacceptably high false positive rate of 86.2%. - Component-based analysis improved performance, achieving: - True Positive Rate: 97.8% - False Positive Rate: 3.7%

This difference highlights the need for component-based analysis. By isolating each branch of the PDF tree and analyzing each separately, real malicious changes are made visible, innocent ones are ignored. Without component-based analysis, this project is simply not viable.

C. Kolmogorov Complexity Metrics

- Clean PDFs: - JavaScript components showed consistent baseline complexity. - Suspicious items that still change, such as metadata, stayed within their thresholds, as more data was rarely being added - this is one reason that employing a method based on Kolmogorov Complexity works better than simply checking for change. - Malicious PDFs: - JavaScript and embedded objects consistently exhibited 35–50% higher complexity than clean baselines. With real malware, a higher complexity difference is expected - the sample malware was rather innocent, showing similar pop-ups and alerts, but lacked any other malicious activity.

D. Performance Metrics

- System ran on a Raspberry Pi with minimal performance impact. - Quarantine and logging processes added a minor delay but preserved real-time processing capabilities.

Note that the quarantining process must be streamlined for proper commercial use - intercepting hundreds of emails simultaneously, calculating complexity, and quarantining multiple at a time would surpass resources available.

E. Limitations and Observations

- High sensitivity to baseline thresholds for different document types. - False positives primarily stemmed from legitimate metadata changes in some external emails. - Detection struggled with stegomalware exploiting vulnerabilities beyond JavaScript. This means vulnerabilities with the document reader itself. Because of this, exploits could be inserted into branches of the PDF tree deemed safe. However, this type of exploit is rarer and can generally be prevented by keeping an updated pdf reader.

VI. Discussion

A. Key Insights

Improved Accuracy: Shifting from whole-PDF analysis to tree-based component isolation significantly reduced false positives (from 86.2% to 3.7%) while maintaining a high true positive rate (97.8%). Focused analysis on suspicious components like JavaScript and actions minimized the impact of benign changes (e.g., text length or page count).

Kolmogorov Complexity Application: Leveraging Kolmogorov complexity as a metric enabled precise detection of anomalies, effectively distinguishing malicious modifications from normal document variations. Compression-based approximations proved practical for real-time processing.

B. Challenges

Threshold Tuning: Balancing sensitivity with specificity remains necessary to minimize false positives without missing threats. Sensitivity is based on new document formats or attack patterns.

Stegomalware Variability: Stegomalware that exploits vulnerabilities beyond JavaScript (PDF reader-specific flaws) required broader detection capabilities, although switching to tree-based detection helped.

Metadata Dependence: Baseline IDs in metadata can be an annoyance for a company if several non-repetitive external emails are received.

C. Conclusion

This study introduces a novel approach to detecting stegomalware within PDF documents by leveraging Kolmogorov complexity as an indicator of hidden anomalies. Traditional malware detection methods struggle with the inherent structural similarities between clean and infected PDFs, making it difficult to identify embedded threats. By manually extracting the hierarchical tree structure of a PDF and applying compression-based complexity approximations to its individual components, this method enables targeted analysis of suspicious attributes such as embedded JavaScript, metadata, and `OpenAction` triggers.

Experimental results demonstrate that shifting from whole-PDF analysis to a component-based approach significantly enhances detection accuracy, reducing false positives from 86.2% to 3.7% while maintaining a high true positive rate of 97.8%. The proposed methodology allows for the isolation of anomalies without being misled by benign changes, such

as variations in page count or document length. Furthermore, the development of a custom PDF parsing library, which extracts structural elements at a more granular level provides a foundation for future research.

While this approach proves effective, challenges remain, particularly in optimizing threshold sensitivity for different document types and improving adaptability against evolving stegomalware techniques. Future work will focus on refining adaptive baseline models, and expanding real-time deployment capabilities for enterprise-level security applications. This research presents a step forward for combating PDF-based cyber threats, offering a scalable solution for detecting hidden malicious payloads within PDF documents.

REFERENCES

[1] MrAnon Stealer Spreads via Email with Fake Hotel Booking PDF. *FortiGuard Labs*, 2024. Available: https://www.fortinet.com/blog/threat-research/mranon-stealer-spreads-via-email-with-fake-hotel-booking-pdf

[2] HP Wolf Security Threat Insights Report, Q4 2023. *HP Development Company, L.P.*, 2024. Available: https://hp.com/wolf

[3] Mcafee Ž024 Rise in deceptive PDFs - The Gateway to Malicious Payloads *Mcafee*, 2024. Available: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/rise-in-deceptive-pdf-the-gateway-to-malicious-payloads/

[4] Infostealer Disguised as Adobe Reader Installer. *ASEC*, 2024. Available: https://asec.ahnlab.com/en/62853/

[5] From Document to Script: Insides of Darkgate's Campaign. *Force-Point X-Labs*, 2024. Available: https://www.forcepoint.com/blog/x-labs/phishing-script-inside-darkgate-campaign

[6] Byakugan Ž013 The Malware Behind a Phishing Attack. *FortiGuard Labs*, 2024. Available: https://www.fortinet.com/blog/threat-research/byakugan-malware-behind-a-phishing-attack

[7] Yizheng Chen, Shiqi Wang, Dongdong She, and Suman Jana. "On Training Robust PDF Malware Classifiers." *Proceedings of the 29th USENIX Security Symposium*, 2020. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/chen-yizheng

[8] Vitányi PMB. How Incomputable Is Kolmogorov Complexity? *Entropy*, 2020; 22(4):408. Available: https://doi.org/10.3390/e22040408