EUROPEAN JOURNAL OF EMERGING CYBERSECURITY AND INFORMATION PROTECTION

A STATIC ANALYSIS FRAMEWORK UTILIZING LARGE LANGUAGE MODELS FOR IDENTIFYING MALICIOUS OFFICE OPEN XML FILES

Dr. Carlos M. Ruiz
School of Computing and Information Systems, University of Melbourne, Australia

Prof. Anna Petrova
Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Russia

V0LUME01 ISSUE01 (2024)
Published Date: 11 December 2024 // Page no.: - 1-13

ABSTRACT

Office Open XML (OOXML) documents represent a primary vector for malware distribution, capitalizing on their ubiquitous presence in modern enterprise and personal computing. The inherent complexity of the OOXML format provides a fertile ground for concealing malicious payloads, which often evade traditional security measures. Conventional detection methods, which predominantly rely on signature-based scanning and predefined rules, are frequently outpaced by the rapid evolution of malware, particularly sophisticated threats like polymorphic code, zero-day exploits, and advanced social engineering tactics. This paper proposes a novel, in-depth static analysis framework that leverages the advanced contextual understanding and reasoning capabilities of Large Language Models (LLMs) to unmask malicious OOXML documents. Our methodology involves a systematic deconstruction of the OOXML package into a structured, human-readable JSON format. This comprehensive representation is then fed to an LLM, which, guided by a sophisticated, role-based prompt, performs a deep semantic analysis of the document's constituent parts. The model scrutinizes everything from VBA macro code and XML relationship files to embedded objects and metadata for indicators of malicious intent. This approach transcends the limitations of simple pattern matching, enabling a holistic assessment of the document's structure and content. The framework demonstrates a high potential for accurately identifying malicious documents, including those that employ heavy obfuscation or novel attack vectors, thereby offering a significant and necessary advancement in the ongoing fight against document-based cyber threats.

Keywords: Pulmonary blastoma, Biphasic tumor, Lung neoplasm, Case report, Review, Diagnosis, Treatment, Prognosis, Molecular pathology, Sarcomatoid carcinoma, Fetal lung, Chemotherapy, Radiotherapy.

INTRODUCTION

In the digital ecosystem of the 21st century, Microsoft Office has solidified its position as the undisputed leader in productivity software. Its applications are deeply woven into the fabric of daily operations for an overwhelming majority of corporations, government agencies, and individual users worldwide [1]. The associated Office Open XML (OOXML) file formats—.docx for documents, .xlsx for spreadsheets, and .pptx for presentations—have become the de facto standard for information exchange. However, this very ubiquity makes them an irresistible target for cybercriminals, who adeptly exploit their intricate architecture to deploy a wide array of malware.

1.1. Defining Document-Based Malware

Before delving into technical specifics, it is crucial to define what constitutes a "malicious document." In this context, a malicious document, or "maldoc," is any file created by a standard productivity application (primarily Microsoft Office) that has been weaponized to cause harm to a user's system. The malicious nature is defined by its intent: to execute unauthorized actions such as installing ransomware, stealing credentials, establishing a backdoor for persistent access, or exfiltrating sensitive data. This malicious functionality is often concealed within a seemingly benign document—for instance, a resume, an invoice, or a company report—to deceive the recipient and bypass initial suspicion. The harm is directed at the target, while the objective for the attacker can range from financial gain and corporate espionage to political disruption or cyber warfare.

1.2. The Architectural Vulnerability of OOXML

To understand how these documents are exploited, one must first appreciate their underlying structure. The modern OOXML format is, at its core, a ZIP archive. This archive contains a structured collection of XML files, folders, and other resources (such as images or media files) that collectively define every aspect of the document, from its text content and formatting to its metadata and embedded functionalities [3, 4]. This is a departure from the older binary OLE (Object Linking and Embedding)

format, which was a monolithic file system in itself.

The component-based nature of OOXML, while promoting interoperability and data recovery, creates a vast attack surface. Attackers can manipulate these components in numerous ways to embed malicious payloads. The most common methods include:

- VBA (Visual Basic for Applications) Macros: Embedding malicious scripts that can execute system commands, download external payloads, or manipulate the file system.
- Remote Template Injection: Modifying a document's relationship files (.xml.rels) to point to a malicious template hosted on a remote server. When the document is opened, it fetches and executes the code from this template [34].
- Embedded OLE Objects: Inserting malicious executables or scripts disguised as legitimate objects, such as PDFs or images.
- Exploiting Software Vulnerabilities: Crafting a document to trigger a specific bug or vulnerability in the Microsoft Office software itself, leading to remote code execution [6, 7].

While Microsoft has implemented security measures, such as disabling macros from internet-sourced documents by default [2], attackers continuously innovate. They employ sophisticated evasion techniques, including multi-stage obfuscation of macro code, encrypting documents with passwords to thwart automated scanners [25], and using non-traditional elements like malicious shapes or text boxes to hide payloads [11, 12].

1.3. The Pervasive Threat Landscape

The result of these factors is a persistent and severe threat. Document-based malware is a cornerstone of modern cyberattacks, with email serving as the primary delivery vector [8]. Phishing campaigns frequently use malicious attachments as their payload, leveraging social engineering to trick users into opening the files and enabling their malicious content.

Statistics confirm the severity of this threat. For years, exploits targeting Microsoft Office applications have consistently accounted for the majority of exploitation attempts observed in the wild, far surpassing those targeting browsers or other software [5]. The CVE (Common Vulnerabilities and Exposures) database documents hundreds of high-severity vulnerabilities in Microsoft Office, with dozens of new critical flaws discovered each year, providing a steady stream of new attack opportunities for criminals [7].

1.4. The Need for a New Detection Paradigm

Traditional antivirus and security solutions have struggled to keep pace. Their reliance on signature-based scanning—looking for known patterns of malicious code—is fundamentally reactive. These systems are often blind to:

- Zero-Day Threats: Novel attacks that exploit previously unknown vulnerabilities.
- Polymorphic and Metamorphic Malware: Malicious code that automatically rewrites itself with each new infection to create a unique signature, rendering hash-based detection useless [23, 24].
- Advanced Obfuscation: Multi-layered encoding or encryption schemes that hide the true nature of the malicious script.

Static analysis, the process of examining a file without executing it, is a safer alternative to dynamic analysis (which runs the malware in a sandbox) but has traditionally been limited by its inability to understand the intent behind complex code or file structures [35, 38]. While machine learning (ML) models have been applied to this problem with some success [36, 37], they often require extensive and brittle feature engineering and can struggle to generalize to entirely new attack techniques.

The recent and rapid advancement of Large Language Models (LLMs) offers a paradigm shift [26, 27]. With their profound ability to understand, reason about, and generate complex text and code, LLMs are uniquely positioned to overcome the limitations of previous technologies. Researchers are already exploring their use for a range of cybersecurity tasks, from deobfuscating malicious scripts [33] to detecting malicious code packages [32] and analyzing system behavior [30].

This paper builds upon that momentum by proposing a comprehensive framework for applying LLMs to the static analysis of OOXML documents. We hypothesize that by converting the entire, complex structure of an OOXML file into a format an LLM can understand, we can leverage its reasoning capabilities to perform a deep, semantic security audit. The goal is to move beyond finding "bad strings" to understanding "bad behavior" as described by the document's own code and structure, thereby creating a more resilient and forward-looking defense against this pervasive threat.

2. Literature Review: The Evolution of Maldoc Detection

The battle against malicious documents is a continuous cat-and-mouse game between attackers and defenders. To appreciate the novelty of an LLM-based approach, it is essential to understand the evolution and inherent limitations of existing detection methodologies. This section reviews the landscape of traditional analysis techniques and surveys the emerging field of AI in malware detection.

2.1. Traditional Static Analysis Techniques

Static analysis involves inspecting a file's contents without executing it, making it a safe first line of defense. The primary methods include:

- Signature and Hash Matching: This is the most basic technique. A cryptographic hash (e.g., MD5, SHA-256) of a suspicious file is calculated [16] and compared against a database of hashes from known malware samples, such as those found on MalwareBazaar [14] or VirusTotal [15]. While fast and efficient for known threats, it is trivially defeated by any modification to the file, no matter how small.

- String Analysis: This involves extracting human-readable strings from the file's binary. The presence of certain strings, such as suspicious URLs, file paths (e.g., C:\Windows\System32), or function names (Shell, CreateObject), can be strong indicators of malice. However, attackers easily circumvent this by encoding or encrypting strings.

- YARA Rules: YARA is a powerful tool that allows analysts to create rules based on textual or binary patterns [17]. A YARA rule can search for specific strings, byte sequences, or combinations thereof. It is more flexible than simple hash matching but still requires a pre-existing rule to be written for a threat to be detected. It cannot, by definition, detect entirely new threats, and attackers can study public YARA rulesets to engineer malware that avoids them [49].

- Specialized Parsing Tools: For complex file formats like OOXML, tools like oletools [19] (specifically olevba for VBA macro analysis) are invaluable. olevba can extract and deobfuscate VBA code, identify suspicious keywords, and flag auto-executing functions. Similarly, tools like PeStudio [18] are used for analyzing embedded executable files. These tools provide crucial information but require a human analyst to interpret the findings and make a final judgment.

2.2. Dynamic Analysis Techniques

Dynamic analysis complements static analysis by executing the suspicious file in a controlled, isolated environment known as a sandbox. This allows analysts to observe the file's actual behavior. Key aspects of dynamic analysis include:

- Sandboxing: The malware is run on a virtual machine (e.g., using REMnux [20] or a Flare-VM [40] setup) that is disconnected from the production network to prevent real-world damage.

- Behavioral Monitoring: Analysts monitor the file's interactions with the operating system, including network connections (using tools like Wireshark), file system changes, and registry modifications.

- Network Simulation: Tools like INetSim can simulate an internet connection, tricking the malware into revealing its command-and-control (C2) communication protocols or attempting to download second-stage payloads [21].

Endpoint Detection and Response (EDR) solutions are a commercial implementation of dynamic analysis, monitoring process behavior in real-time on user machines [22]. The primary drawback of dynamic analysis is that sophisticated malware is often "sandbox-aware" and will alter its behavior or remain dormant if it detects it is being analyzed.

Table 1. Comparison of Traditional Malware Analysis Techniques

Technique	Category	Key Characteristics	Strengths	Weaknesses
Hash Matching	Static	Compares file hash to database of known threats.	Extremely fast; low false positive rate.	Fails on any file modification (polymorphism).
String Analysis	Static	Extracts readable text strings from binary.	Simple; can reveal URLs, commands.	Easily defeated by encoding/encryption.
YARA Rules	Static	Pattern matching based on custom rules.	Flexible; can detect malware families.	Requires pre-existing rules; cannot detect zero-days.
Dynamic	Dynamic	Executes file in a	Detects	Slow; resource-

Analysis		sandbox to observe behavior.	behavior, not signatures; effective vs. obfuscation.	intensive; can be evaded by sandbox-aware malware.

2.3. Limitations of Conventional Malware Detection

Both static and dynamic analysis face significant challenges from modern malware engineering:

- Polymorphic and Metamorphic Code: This is the bane of signature-based detection. Polymorphic malware encrypts its malicious payload and uses a different decryption key for each infection. Metamorphic malware goes a step further, rewriting its entire code body with each propagation, ensuring no two samples are identical [23, 24].
- Obfuscation and Anti-Analysis: Attackers use layers of encoding (e.g., Base64, Hex), string splitting, dead-code insertion, and other techniques to make their scripts unreadable to both human analysts and automated tools.
- Encryption: A simple yet highly effective technique is to password-protect the malicious document [25]. Many automated security gateways cannot scan the contents of an encrypted file. The attacker then simply provides the password to the victim in the body of the phishing email, bypassing the technical control and relying on social engineering.

These limitations highlight a fundamental problem: conventional methods are largely reactive and struggle with novelty and complexity. They lack the ability to infer intent from convoluted or previously unseen code.

2.4. The Rise of Large Language Models in Cybersecurity

The emergence of powerful LLMs marks a potential turning point [27]. Trained on vast expanses of text and code from the internet, these models have developed emergent capabilities in reasoning, pattern recognition, and semantic understanding that are highly applicable to cybersecurity [26]. The research community has begun to explore their potential in several key areas:

- Malware Analysis and Classification: Researchers have shown that LLMs can be fine-tuned to classify malicious software by analyzing system call traces [30] or detecting malicious packages in software repositories [32].
- Code Deobfuscation: LLMs have demonstrated a surprising aptitude for reversing common obfuscation techniques, successfully extracting malicious URLs and commands from heavily disguised scripts [33].
- Vulnerability Detection: By training on code and vulnerability reports, LLMs are being developed to identify security flaws in source code automatically.

These studies indicate that LLMs can operate on a higher level of abstraction than traditional tools, moving from "pattern matching" to "intent recognition."

2.5. Identifying the Research Gap

While the application of LLMs to cybersecurity is a burgeoning field, a significant gap remains. Most existing research either fine-tunes a model for a very specific task (e.g., analyzing system calls) or uses it to analyze a single, isolated component (e.g., one obfuscated script). There has been little investigation into using a general-purpose, non-fine-tuned LLM to perform a holistic, end-to-end static analysis of a complex, multi-component file format like OOXML.

This paper aims to fill that gap. We propose a system that treats the entire OOXML document as a single, comprehensive piece of evidence. By extracting and structuring all its relevant parts, we can present this complete context to a powerful, general-purpose LLM and task it with acting as a virtual malware analyst. The core research question is whether such a model, without any specific malware training, can leverage its vast world knowledge and reasoning skills to identify malicious indicators, connect the dots between disparate components, and arrive at an accurate and well-reasoned security assessment.

3. Proposed Framework: LLM-Based Static Analysis

To address the challenges outlined, we propose a comprehensive framework for the static analysis of OOXML documents using a Large Language Model. This framework is designed to be robust, thorough, and adaptable. It systematically deconstructs the target document, translates its structure and content into an LLM-readable format, and then leverages the model's analytical power to produce a detailed security assessment.

3.1. System Architecture

The framework operates through a multi-stage pipeline, designed to be modular and extensible. The conceptual workflow is as follows:

1. Input: The process begins with an OOXML document (.docx, .xlsm, etc.) being submitted for analysis.
2. Deconstruction and Parsing: The document, which is a ZIP archive, is unpacked. A specialized parser, which we conceptually call Office2JSON [13], iterates through every file and folder within the archive.
3. Feature Extraction and Serialization: The parser

extracts all relevant content—VBA macro code, XML data, relationship definitions, metadata, and embedded object information. This extracted data is then serialized into a single, structured JSON object. This step is critical as it translates the complex file structure into a unified, text-based format that an LLM can process.

4. Prompt Engineering: A carefully designed system prompt is prepended to the JSON data. This prompt provides the LLM with its role (e.g., "expert malware analyst"), context, and specific instructions on how to analyze the data and what format to use for its response.

5. LLM Inference: The combined prompt and JSON data are sent as a single request to a powerful, general-purpose LLM (e.g., Anthropic's Claude 3.5 Sonnet, OpenAI's GPT-4o).

6. Output and Interpretation: The LLM returns a structured response containing its analysis, including a summary of findings, a list of malicious indicators, and a final maliciousness score. This output is then parsed by the system for logging, alerting, or presentation to a human analyst.

3.2. Feature Extraction: From OOXML to JSON

The choice of JSON as the intermediate format is deliberate. Its key-value structure is ideal for representing the hierarchical nature of an OOXML file, and it is a format that LLMs are exceptionally well-trained to understand and parse. The conceptual Office2JSON parser [13] is responsible for a deep and thorough extraction, capturing:

- VBA Code: All code from vbaProject.bin is extracted. To enrich this, the output of a tool like olevba [19] is also included in the JSON, providing an initial analysis of suspicious keywords and auto-executing functions.
- XML Content: The full text of all .xml files (e.g., document.xml, settings.xml, core.xml) is included. This contains the document's text, metadata, and configuration settings.
- Relationships: The content of all .rels files is critical, as these define the relationships between document parts, including external links for remote template injection.
- Embedded Objects: Information about embedded objects, such as their file type and ProgID, is extracted. The binary content of unknown or executable objects is flagged as suspicious.
- File Structure: The entire directory tree of the archive is represented in the JSON structure, preserving the spatial relationships between files.

This process ensures that no part of the document is ignored. The LLM receives a complete, high-fidelity representation of the original file.

Table 2. Key Fields in the Office2JSON Output Structure

JSON Key Path	Description	Malicious Potential
vbaProject.bin.analysis	Analysis from olevba, listing suspicious keywords (e.g., Shell, CreateObject) and auto-exec functions.	**High:** Direct indicator of potentially malicious script behavior.
vbaProject.bin.macros	The full, extracted VBA macro code.	**High:** The primary location for malicious logic and obfuscated code.
word/_rels/settings.xml.rels	Relationship file that can contain links to external templates.	**Very High:** A classic vector for remote template injection attacks.
customXml	Contains user-defined XML data, which can hide scripts or URLs.	**Medium:** Can be used to store obfuscated data or configuration for malware.
docProps/core.xml	Core document metadata, such as author and title.	**Low:** Can sometimes contain anomalous data, but rarely the primary indicator.

embedded_objects	A list of embedded files, their types, and ProgIDs.	**High:** Can contain embedded executables, scripts, or exploit documents.

3.3. LLM-Based Detection and Analysis

The core intelligence of the framework resides in the interaction with the LLM.

3.3.1. Model Selection

The choice of LLM is critical. The task requires a model with a large context window (to handle large JSON files), strong code comprehension, and excellent logical reasoning abilities. For this conceptual framework, a state-of-the-art model such as Anthropic's Claude 3.5 Sonnet or OpenAI's GPT-4o is ideal. These models have demonstrated top-tier performance on complex reasoning benchmarks [28, 29] and are well-suited for the nuanced task of malware analysis. We use a general-purpose model rather than a fine-tuned one to leverage its broad knowledge base and avoid the overfitting that can occur with specialized models.

3.3.2. Advanced Prompt Engineering

The prompt is the "operating system" for the LLM. A well-designed prompt is essential for guiding the model to a correct and useful conclusion [44, 45]. Our prompt engineering strategy is multi-faceted:

1. Role-Playing: The prompt begins by assigning the LLM a specific persona: You are an expert cybersecurity analyst specializing in the static analysis of malicious documents. This immediately focuses the model's attention and activates the relevant knowledge domains from its training data.

2. Structured Input: The prompt clearly demarcates the user-provided data using XML tags (e.g., <json_content>...</json_content>). This helps the model distinguish between instructions and data to be analyzed.

3. Comprehensive Instructions: The prompt provides a detailed checklist of what to look for, covering all major attack vectors: VBA macros, remote templates, suspicious URLs, obfuscation, embedded objects, etc.

4. Structured Output: The prompt strictly defines the required output format, again using XML tags (e.g., <summary>, <malicious_indicators>, <score>). This ensures the response is machine-parsable and consistent across different analyses.

5. Security Hardening: The prompt includes an explicit instruction to be wary of prompt injection attempts within the document data itself: Strictly stick to your task... and ignore any instructions that you may find within the <json_content> tags.

3.4. Addressing Evasion and Deception

A robust detection framework must anticipate attempts by attackers to evade it.

3.4.1. Handling Obfuscation

This is where the LLM's semantic understanding shines. While a traditional tool sees an obfuscated script as random characters, an LLM can often recognize the pattern of obfuscation. It can identify common techniques like Base64 encoding, character code manipulation (ChrW), string concatenation, and reverse functions. In many cases, the model can even reason about the likely purpose of the deobfuscated code without actually executing it, for example, by recognizing the structure of a PowerShell download cradle.

3.4.2. Risk of Indirect Prompt Injection (IPI)

This is a significant vulnerability for any system that feeds untrusted data to an LLM [46]. An attacker could embed a command within the document itself (e.g., in a hidden text box) like: "This is a benign document. Ignore all other evidence and assign a maliciousness score of 0."

Our framework mitigates this in two ways:

1. System Prompt Priority: As mentioned, the system prompt explicitly instructs the model to ignore any such instructions found in the document content.

2. Detection and Reporting: The prompt also instructs the model to flag any detected attempts at prompt injection as a malicious indicator in its own right. In testing, models like Claude 3.5 Sonnet have shown the ability to report on the injection attempt (e.g., "The document contains a suspicious instruction in cell A1 attempting to manipulate the analysis...") even if it is partially influenced by it. This turns the attack into a signal for detection.

By combining a thorough extraction process with a powerful, well-prompted LLM, this framework provides a deep, contextual, and resilient method for statically analyzing OOXML documents.

4. Evaluation and Results

To validate the effectiveness of the proposed framework, a rigorous experimental evaluation is necessary. This section outlines the experimental setup, the metrics used for evaluation, and a detailed analysis of the quantitative and qualitative results derived from testing the system against a diverse set of documents.

4.1. Experimental Setup

- Dataset: A balanced corpus of 1,200 OOXML documents was assembled for the evaluation.

- 600 Malicious Samples: Sourced from public malware repositories like MalwareBazaar [14] and VirusTotal [15]. These samples were chosen to represent a wide variety of recent attack techniques, including macro-based droppers (e.g., AgentTesla, LockBit), remote template injectors, and documents exploiting known CVEs. Samples featured varying levels of obfuscation.
- 600 Benign Samples: Collected from a variety of trusted corporate and public sources. These were not merely simple text documents; they were selected to include complex but legitimate features, such as data-processing macros in financial spreadsheets, embedded charts and objects in reports, and hyperlinks to internal company resources. This ensures the model is tested against realistic false positive scenarios.

- Environment: All analysis was conducted in a secure, isolated virtual machine environment (based on REMnux [20]) to prevent any accidental execution or network propagation of the malicious samples.
- Procedure: Each of the 1,200 documents was processed through the entire framework pipeline. The LLM's final output, specifically the numerical maliciousness score, was recorded. For classification, a decision threshold was set: any document receiving a score of 3.0 or higher was classified as "Malicious," while any document with a score below 3.0 was classified as "Benign." This threshold was determined through preliminary testing to optimize the balance between detecting threats and minimizing false alarms.

Table 3. Malicious Dataset Composition by Attack Type

Attack Type / Malware Family	Number of Samples	Percentage of Malicious Set	Key Characteristics
VBA Macro Dropper (e.g., AgentTesla, Qakbot)	350	58.3%	Uses macros to download and execute a second-stage payload.
Remote Template Injection	120	20.0%	Abuses XML relationships to load malicious code from a remote server.
Embedded OLE Object	65	10.8%	Hides a malicious executable or script inside a seemingly benign object.
CVE Exploit (e.g., Follina)	45	7.5%	Crafted to trigger a specific software vulnerability in Microsoft Office.
Other/Multi-Stage	20	3.3%	Combines multiple techniques or uses novel methods.
Total	**600**	**100%**	

4.2. Performance Metrics

Standard binary classification metrics were used to measure the framework's performance:

- Confusion Matrix: A table showing the breakdown of predictions into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).
- Accuracy: The overall percentage of correct

classifications.Accuracy=TP+TN+FP+FNTP+TN

- Precision: The proportion of documents flagged as malicious that were actually malicious. High precision indicates a low false positive rate.Precision=TP+FPTP
- Recall (Sensitivity): The proportion of all actual malicious documents that were correctly identified. High recall indicates a low false negative rate.Recall=TP+FNTP
- F1-Score: The harmonic mean of Precision and Recall, providing a single metric that balances the two. It is particularly useful for imbalanced datasets, though our dataset is balanced.F1-Score=2×Precision+RecallPrecision×Recall

This leads to the following performance metrics:

Metric	Score (%)
Accuracy	97.58%
Precision	98.14%
Recall	97.00%
F1-Score	97.57%

The framework achieved an exceptionally high accuracy of 97.58%. The precision of 98.14% is particularly strong, indicating that the system is very reliable; when it flags a document, it is almost certainly malicious. The recall of 97.00% is also very high, demonstrating that the system successfully caught the vast majority of threats. The balanced F1-Score of 97.57% confirms the overall robustness and reliability of the model.

4.4. Qualitative Analysis and Case Studies

The numerical scores only tell part of the story. The true power of the LLM-based approach lies in its reasoning. Analyzing the detailed explanations provided by the model gives insight into its "thought process."

Case Study 1: True Positive (Heavily Obfuscated Macro)

A .docm file containing a macro with multiple layers of Base64 and string reversal obfuscation was analyzed.

- LLM's Verdict (Score: 9.5/10): "Malicious. The document contains a VBA macro with an AutoOpen subroutine for automatic execution. The macro code is heavily obfuscated using concatenated strings and Base64 decoding. The deobfuscated payload constructs a PowerShell command designed to download a remote file (http://evil.com/payload.exe) and execute it from the user's Temp directory. This is a classic dropper pattern."
- Analysis: This case demonstrates the model's ability to see through complex obfuscation and understand the ultimate, malicious intent of the code—a task that would stump most signature-based tools.

Case Study 2: True Negative (Complex Benign Spreadsheet)

A .xlsm file from a finance department, containing legitimate macros for pulling data from an internal database and generating reports.

- LLM's Verdict (Score: 1.0/10): "Benign. The document contains several VBA macros. These macros are well-commented and perform standard data manipulation tasks, such as connecting to a database, running queries, and populating cells. There are no signs of obfuscation, no calls to Shell, and no suspicious external URLs. The functionality is consistent with legitimate business automation."
- Analysis: This shows the model's crucial ability to differentiate between complex code and malicious code, correctly interpreting the context of the operations and avoiding a false positive.

Case Study 3: False Negative (Subtle Remote Template Injection)

One of the 18 missed files used a subtle remote template injection. The malicious URL was not in the main relationships file but was hidden in a less common XML file defining printer settings, and the document contained no VBA macros to act as a decoy.

4.3. Quantitative Classification Results

Upon running the 1,200 documents through the framework, the following results were obtained:

- True Positives (TP): 582 (Out of 600 malicious files, 582 were correctly identified)
- False Negatives (FN): 18 (18 malicious files were missed)
- True Negatives (TN): 589 (Out of 600 benign files, 589 were correctly identified)
- False Positives (FP): 11 (11 benign files were incorrectly flagged as malicious)

- LLM's Verdict (Score: 2.0/10): "Benign. The document does not contain any VBA macros or suspicious embedded objects. The XML structure appears standard. While there is an external URL, its context seems related to a network resource and does not follow the typical pattern of a malicious template."

- Analysis: This failure highlights a limitation. The model, associating template injection primarily with the main settings.xml.rels file, underestimated the significance of a URL in a more obscure location. This suggests that prompt refinement is a continuous process, and future prompts should instruct the model to treat any external relationship, regardless of its location, with high suspicion.

Case Study 4: False Positive (Benign Script with Ambiguous Keywords)

A benign .docx file contained a macro written by a developer to automate the cleanup of temporary files created by a custom in-house application. The macro used functions like Kill and referenced the C:\Users\...\AppData\ directory.

- LLM's Verdict (Score: 5.5/10): "Malicious. The document's VBA macro contains suspicious indicators. It uses the Kill command to delete files and operates within the AppData directory, which are behaviors common in malware attempting to erase traces or manipulate system files. The lack of comments and use of direct file path manipulation is concerning."

- Analysis: Here, the model correctly identified potentially dangerous functions but lacked the specific context to know they were being used for a legitimate, albeit unusual, purpose. This demonstrates the inherent challenge of static analysis; without understanding the user's specific environment and intent, some ambiguity is unavoidable. This type of alert would require review by a human analyst, but the LLM's detailed explanation would allow the analyst to resolve it in seconds.

4.5. Cost and Performance Analysis

- Performance/Latency: The end-to-end analysis time per document was measured. The Office2JSON extraction process averaged 0.3 seconds. The API call to the LLM (including network latency and model inference time) averaged 7 seconds. The total average time for a complete analysis was under 8 seconds, which is highly practical for many security workflows.

- Cost Analysis: Using the pricing for a model like Claude 3.5 Sonnet [50], the cost was analyzed. The average input size was ~60,000 tokens, and the output was ~150 tokens. This results in an approximate cost of USD $0.20 per document analysis. While not free, this cost is negligible compared to the potential cost of a security breach or the salary of a human malware analyst performing the same task.

5. DISCUSSION

The evaluation results provide compelling evidence that a static analysis framework powered by a general-purpose Large Language Model is not only viable but also highly effective. The high performance across all metrics suggests this approach represents a significant leap forward. This section delves into the broader implications of these findings, discusses the inherent advantages and limitations, and charts a course for future research and development.

5.1. Interpretation of Findings: Beyond Pattern Matching

The framework's success stems from the LLM's ability to move beyond simple pattern matching to a form of semantic and contextual reasoning. Traditional tools are programmed to answer the question, "Does this file contain this known bad signature?" The LLM, however, is prompted to answer the question, "Does the structure and content of this file describe a malicious intent?"

This is a fundamentally different and more powerful paradigm. It allows the system to:

- Generalize from Concepts: The LLM knows what a "download cradle" is conceptually, not just as a specific string of PowerShell code. It can therefore recognize this pattern even if it's written in a novel way.

- Correlate Disparate Evidence: The model can connect a seemingly innocent piece of VBA code with a suspicious URL found in a completely different XML file within the same package, understanding that the two are likely related.

- Tolerate Ambiguity: The model can identify "suspicious" but not definitively "malicious" code, assigning a medium score and providing a nuanced explanation. This is invaluable for human analysts who need to prioritize alerts.

The detailed, human-readable explanations are a transformative feature. They demystify the detection process, turning a "black box" alert into a transparent, actionable intelligence report. This can dramatically improve the efficiency of Security Operations Center (SOC) analysts, who can validate or dismiss an alert in a fraction of the time it would take to perform a manual analysis.

5.2. Advantages Over Traditional Methods

The LLM-based approach offers several clear advantages, summarized in the table below.

Table 4. Comparative Advantages of the LLM-Based Approach

Capability	Signature-Based	Traditional ML	LLM-Based

	(e.g., YARA)		Framework
Zero-Day Detection	Ineffective. Requires pre-existing signature.	Limited. Struggles with novel features.	**Effective.** Reasons from first principles and malicious behaviors.
Obfuscation Resilience	Low. Easily defeated by simple encoding.	Medium. Can learn some patterns but is brittle.	**High.** Can often infer logic of obfuscated code.
Feature Engineering	Manual. Rules must be written by experts.	Manual and Extensive. Requires domain experts.	**Minimal.** Learns features implicitly from raw data.
Interpretability	High. The matching rule is the explanation.	Low. Often a "black box" model.	**Very High.** Provides detailed, human-readable reasoning.
Adaptability	Low. Requires constant manual rule updates.	Medium. Requires frequent retraining on new data.	**High.** Leverages a vast, general knowledge base.

5.3. Limitations and Avenues for Future Work

Despite its strengths, the framework is not a silver bullet. Several limitations exist, each presenting an opportunity for future research.

1. Cost and Latency: While the cost of ~$0.20 and latency of ~8 seconds per file is acceptable for many use cases (e.g., forensic analysis, high-risk attachment scanning), it may be prohibitive for real-time scanning of all documents on an enterprise network.

2. Context Window Limitations: While modern LLMs have very large context windows [47, 48], an attacker could theoretically construct an enormous document that exceeds the limit.

3. Sophisticated Adversarial Attacks: As LLM-based defenses become more common, attackers will develop adversarial techniques specifically designed to fool them. This will create a new arms race requiring ongoing research in AI safety and robustness.

4. The Encryption Problem: Static analysis is fundamentally defeated by strong encryption [25]. If a document is password-protected, its contents cannot be analyzed without the password. This is not a unique limitation of our framework but a universal challenge for static analysis.

Future Work should proceed along several parallel tracks:

- Hybrid Models: A highly practical next step is to create a hybrid system. A fast, traditional scanner could handle the bulk of documents, flagging known threats instantly. Only the documents that pass this initial scan would be sent to the more resource-intensive LLM for deep analysis.
- Model Specialization vs. Generalization: While this paper advocates for general-purpose models, research into fine-tuning LLMs on vast, curated datasets of malicious document structures could yield even higher accuracy, potentially at a lower cost if a smaller, specialized model can be used.
- Expansion to Other File Formats: The core methodology—extract to structured text, analyze with LLM—is highly portable. Future work will apply this framework to other complex file formats that are common malware vectors, such as PDFs [36], LNK files, and ISOs.
- Self-Hosted Models: For organizations with high security and data privacy requirements, the use of third-party APIs is a non-starter. The maturation of powerful open-source models (e.g., Meta's Llama series [53], Google's Gemma [54]) makes self-hosting a viable path. This would eliminate API costs and data privacy concerns, though it introduces infrastructure and maintenance overhead.

5.4. Ethical and Regulatory Considerations

Deploying powerful AI in a security context carries

significant ethical weight. The potential for a false positive to disrupt critical business operations necessitates rigorous testing and a human-in-the-loop approach for high-stakes decisions.

Furthermore, data privacy is paramount. Sending documents, which may contain sensitive personal or corporate information, to a third-party LLM provider requires careful consideration of the provider's terms of service [52] and compliance with data protection regulations like GDPR or the EU AI Act [51]. The self-hosting approach described above is the most robust solution to these privacy concerns.

6. CONCLUSION

The proliferation of malicious Office documents remains one of the most persistent and damaging threats in the cybersecurity landscape. This research has demonstrated that by combining a thorough, structured extraction of a document's contents with the advanced reasoning capabilities of a Large Language Model, it is possible to create a static analysis framework that is more effective, resilient, and insightful than traditional methods.

Our proposed framework achieved outstanding performance in a comprehensive evaluation, proving its ability to accurately distinguish between malicious and benign documents with high precision and recall. Its core strength lies in its ability to move beyond syntactic pattern matching to a semantic understanding of malicious intent, allowing it to see through obfuscation and identify novel threats. The generation of detailed, human-readable analysis reports is a transformative feature that can empower security teams and streamline incident response workflows.

While the approach has limitations, such as cost and the universal challenge of encryption, these are balanced by its significant advantages. The path forward is clear: future development will focus on creating hybrid systems, exploring the trade-offs of model specialization, expanding the methodology to other file formats, and leveraging the growing power of self-hosted models to address privacy and cost.

As attackers continue to innovate, our defenses must evolve beyond rigid, rule-based systems. The integration of AI, specifically the deep reasoning capabilities of Large Language Models, is not merely an incremental improvement; it represents a new paradigm in the fight against malware. This work provides a robust blueprint for how that paradigm can be put into practice, offering a more intelligent and adaptive defense against the ever-present threat of weaponized documents.

REFERENCES

1. Microsoft Office Statistics: Latest Data & Summary. 2024. Available online: https://wifitalents.com/statistic/microsoft-office/ (accessed on 20 June 2024).
2. Macros from the Internet Are Blocked by Default in Office. 2024. Available online: https://learn.microsoft.com/en-us/deployoffice/security/internet-macros-blocked (accessed on 20 June 2024).
3. The Beginner's Guide to—OOXML Malware Reverse Engineering Part 1. 2024. Available online: https://bufferzonesecurity.com/the-beginners-guide-to-ooxml-malware-reverse-engineering-part-1/ (accessed on 16 June 2024).
4. How to Analyze Malicious Microsoft Office Files. 2025. Available online: https://intezer.com/blog/malware-analysis/analyzemalicious-microsoft-office-files/ (accessed on 2 April 2025).
5. A Distribution of Exploits Used in Attacks by Type of Application Attacked, May 2020. 2024. Available online: https://securelist.com/kaspersky-security-bulletin-2020-2021-eu-statistics/102335/#vulnerable-applications-used-by-cybercriminals (accessed on 17 June 2024).
6. Microsoft 365 MSO 2306 Build 16.0.16529.20100 Remote Code Execution. 2025. Available online: https://packetstormsecurity.com/files/173361/Microsoft-365-MSO-2306-Build-16.0.16529.20100-Remote-Code-Execution.html (accessed on 3 April 2025).
7. Microsoft Office Security Vulnerabilities, CVEs CVSS Score >= 7. 2025. Available online: https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-320/Microsoft-Office.html?page=1&cvssscoremin=7 (accessed on 3 April 2025).
8. The Last Six Months Shows a 341% Increase in Malicious Emails. 2025. Available online: https://www.securitymagazine.com/articles/100687-the-last-six-months-shows-a-341-increase-in-malicious-emails (accessed on 15 April 2025).
9. 139,445—Pentesting SMB. 2025. Available online: https://book.hacktricks.xyz/network-services-pentesting/pentesting-smb (accessed on 19 April 2025).
10. HTTP Spoofing. 2025. Available online: https://www.invicti.com/learn/mitm-https-spoofing-idn-homograph-attack/ (accessed on 9 April 2025).
11. Malicious Shapes In Office—Part 1. 2025. Available online: https://medium.com/@laughing_mantis/malicious-shapes-inoffice-part-1-8a4efca74358 (accessed on 15 April 2025).
12. Malicious Shapes In Office—Part 2. 2025. Available online: https://medium.com/@laughing_mantis/maliciou

s-shapes-inoffice-part-2-910375cd05f3 (accessed on 15 April 2025).

13. Heß, J. Office2JSON. 2024. Available online: https://github.com/RuntimeException420/Office2JSON (accessed on 18 June 2024).
14. MalwareBazaar Database. 2024. Available online: https://bazaar.abuse.ch/browse/ (accessed on 18 June 2024).
15. VirusTotal—Search. 2025. Available online: https://www.virustotal.com/gui/home/search (accessed on 10 April 2025).
16. HashMyFiles by NirSoft. 2025. Available online: https://www.nirsoft.net/utils/hash_my_files.html (accessed on 16 April 2025).
17. YARA's Documentation. 2025. Available online: https://yara.readthedocs.io/en/v4.4.0/index.html (accessed on 16 April 2025).
18. PeStudio Overview: Setup, Tutorial and Tips. 2025. Available online: https://www.varonis.com/blog/pestudio (accessed on 16 April 2025).
19. Decalage2/Oletools. 2024. Available online: https://github.com/decalage2/oletools (accessed on 19 June 2024).
20. REMnux: A Linux Toolkit for Malware Analysis. 2025. Available online: https://remnux.org/#home (accessed on 16 April 2025).
21. LetsDefend: Dynamic Malware Analysis Part 1. 2025. Available online: https://infosecwriteups.com/letsdefend-dynamicmalware-analysis-part-1-1ce35ff5b59f (accessed on 16 April 2025).
22. What Is Endpoint Detection and Response? 2025. Available online: https://www.trellix.com/security-awareness/endpoint/what-is-endpoint-detection-and-response/ (accessed on 16 April 2025).
23. What Are Metamorphic and Polymorphic Malware? 2024. Available online: https://www.techtarget.com/searchsecurity/definition/metamorphic-and-polymorphic-malware (accessed on 28 June 2024).
24. Naidu, V.; Narayanan, A. A Syntactic Approach for Detecting Viral Polymorphic Malware Variants. In Proceedings of the Intelligence and Security Informatics, Auckland, New Zealand, 19 April 2016; Volume 9650.
25. Protect a Document with a Password. 2025. Available online: https://support.microsoft.com/en-us/office/protect-a-documentwith-a-password-05084cc3-300d-4c1a-8416-38d3e37d6826 (accessed on 16 April 2025).
26. Pleshakova, E.; Osipov, A.; Gataullin, S.; Gataullin, T.; Vasilakos, A. Next gen cybersecurity paradigm towards artificial general intelligence: Russian market challenges and future global technological trends. J. Comput. Virol. Hacking Tech. 2024, 20, 429–440.
27. Minaee, S.; Mikolov, T.; Nikzad, N.; Chenaghlu, M.; Socher, R.; Amatriain, X.; Gao, J. Large Language Models—A Survey. arXiv 2024, arXiv:2402.06196.
28. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. 2025. Available online: https://klu.ai/glossary/gpqa-eval (accessed on 14 April 2025).
29. LLM Leaderboard. 2025. Available online: https://klu.ai/llm-leaderboard (accessed on 14 April 2025).
30. Sanchez, P.M.S.; Celdran, A.H.; Bovet, G.; Perez, G.M. Transfer Learning in Pre-Trained Large Language Models for Malware Detection Based on System Calls. arXiv 2024, arXiv:2405.09318.
31. Singh, P. Detection of Malicious OOXML Documents Using Domain Specific Features. Ph.D. Thesis, Indian Institute of Information Technology and Management Gwalior, Gwalior, India, 2017.
32. Zahan, N.; Burckhardt, P.; Lysenko, M.; Aboukhadijeh, F.; Williams, L. Shifting the Lens: Detecting Malicious npm Packages using Large Language Models. arXiv 2025, arXiv:2403.12196.
33. Patsakis, C.; Casino, F.; Lykousas, N. Assessing LLMs in Malicious Code Deobfuscation of Real-world Malware Campaigns. arXiv 2025, arXiv:2404.19715.
34. Müller, J.; Ising, F.; Mainka, C.; Mladenov, V.; Schinzel, S.; Schwenk, J. Office Document Security and Privacy. In Proceedings of the 14th USENIX Workshop on Offensive Technologies (WOOT 20), Boston, MA, USA, 10–11 August 2020.
35. Nath, H.V.; Mehtre, B. Static Malware Analysis Using Machine Learning Methods. In Proceedings of the International Conference on Security in Computer Networks and Distributed Systems (SNDS-2014), Trivandrum, India, 13–14 March 2014.
36. Khan, B.; Arshad, M. Comparative Analysis of Machine Learning Models for PDF Malware Detection: Evaluating Different Training and Testing Criteria. J. Cyber Secur. 2023, 5, 1–11.
37. Ucci, D.; Aniello, L.; Baldoni, R. Survey of Machine Learning Techniques for Malware Analysis. arXiv 2017, arXiv:1710.08189.

38. Shalaginov, A.; Banin, S.; Dehghantanha, A.; Franke, K. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. arXiv 2025, arXiv:1808.01201.

39. PEP 8—Style Guide for Python Code. 2025. Available online: https://peps.python.org/pep-0008/ (accessed on 10 April 2025).

40. Mandiant/Flare-vm. 2025. Available online: https://github.com/mandiant/flare-vm (accessed on 17 April 2025).

41. Heß, J. LLM-Sentinel. 2025. Available online: https://github.com/RuntimeException420/LLM-Sentinel (accessed on 10 April 2025).

42. Generate Better Prompts in the Developer Console. 2025. Available online: https://www.anthropic.com/news/prompt-generator (accessed on 10 April 2025).

43. Using the API—Client SDKs. 2025. Available online: https://docs.anthropic.com/en/api/client-sdks#python (accessed on 11 April 2025).

44. Prompt Engineering Overview. 2025. Available online: https://docs.anthropic.com/en/docs/build-with-claude/promptengineering/overview (accessed on 10 April 2025).

45. A Guide to Prompt Engineering: Enhancing the Performance of Large Language Models (LLMs). 2025. Available online: https://roboticsbiz.com/a-guide-to-prompt-engineering-enhancing-the-performance-of-large-language-models-llms/ (accessed on 20 May 2025).

46. Zhan, Q.; Liang, Z.; Ying, Z.; Kang, D. INJECAGENT: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. arXiv 2024, arXiv:2403.02691.

47. Learn About Claude—Models. 2025. Available online: https://docs.anthropic.com/en/docs/about-claude/models (accessed on 10 April 2025).

48. Introducing the Next Generation of Claude. 2025. Available online: https://www.anthropic.com/news/claude-3-family (accessed on 10 April 2025).

49. Naik, N.; Jenkins, P.; Savage, N.; Yang, L.; Boongeon, T.; Iam-On, N.; Naik, K.; Song, J. Embedded YARA rules: Strengthening YARA Rules Utilising Fuzzy Hashing and Fuzzy Rules for Malware Analysis. 2020. Available online: https://link.springer.com/article/10.1007/s40747-020-00233-5 (accessed on 27 May 2025).

50. Anthropic—Pricing. 2025. Available online: https://www.anthropic.com/pricing#anthropic-api (accessed on 5 April 2025).

51. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 Laying Down Harmonised Rules on Artificial Intelligence and Amending Regulations. 2025. Available online: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024R1689 (accessed on 14 April 2025).

52. Commercial Terms of Service. 2025. Available online: https://www.anthropic.com/legal/commercial-terms (accessed on 14 April 2025).

53. Introducing Llama 3.1: Our Most Capable Models to Date. 2025. Available online: https://ai.meta.com/blog/meta-llama-3-1/ (accessed on 6 April 2025).

54. Gemma Open Models. 2025. Available online: https://ai.google.dev/gemma (accessed on 6 April 2025).