**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلوم
https://alqalam.utripoli.edu.ly/index.php/AR

*Original article*

# An Empirical Analysis of Parser Divergence and Structural Evasion in JPEG-Based Polyglot Files

**Ebtihal Forjani[1]*, Mohamed Elbeshti[2], Amel Alnaas[3]**

[1]*Department of Information Technology, Libya Academy for Graduate Studies, Tripoli, Libya*
[2]*Department of Information Technology, Faculty of Information Technology, University of Zawia, Zawia, Libya*
[3]*Department of Information Technology, Libya Academy for Graduate Studies, Tripoli, Libya*
***Corresponding Email.*** *Ebtihal.forjani@gmail.com*

**Abstract**
Traditional file-security models rest on an assumption that a binary object has one identity, verified once at the boundary of a system. This paper examines a structural weakness that breaks that assumption: parser divergence, the phenomenon by which two independent interpreting engines apply contradictory rules to the same byte stream and arrive at two mutually exclusive, and equally valid, conclusions about what the file is. We show that this divergence can be deliberately engineered, rather than discovered by accident, to build a single binary object that satisfies the structural expectations of a standard image viewer while simultaneously carrying a fully executable command sequence. The mechanism exploited here is the error-tolerant design mandated by the JPEG specification itself (ITU-T T.81, 1992): decoding engines are required to skip unrecognized data blocks rather than reject the file outright, a rule intended to preserve compatibility with minor transmission corruption or vendor-specific metadata. A proof-of-concept file was constructed by embedding a Windows batch payload inside the comment segment (marker 0xFFFE) of a standards-compliant JPEG image using ExifTool [4], a Perl-based metadata utility. The resulting object renders as an ordinary photograph in every image viewer tested, yet executes the embedded command sequence when the same bytes are read by the Windows command interpreter.
**Keywords.** Parser Divergence, Polyglot Files, JPEG Metadata Exploitation, File Upload Security, Structural Evasion, ExifTool, Malware Detection Evasion.

## Introduction

A common assumption in both user behavior and system administration is that a file's structural identity is fixed and unique once created. Operating systems reinforce that assumption through two mechanisms: the file extension, a label appended to the filename for the end user's benefit, and magic bytes, the signature embedded in the binary header that many security tools treat as authoritative. Neither indicator, on inspection, provides the guarantee it is commonly assumed to provide. An extension is a naming convention with no binding relationship to the bytes that follow it, and magic bytes verify only the initial structural framework of a file, not the intent of every byte that comes after it. The more serious problem is that different software platforms apply different interpretation rules to the identical byte sequence — a condition this paper refers to, following established usage in the parser-security literature, as parser divergence [1,2]. This divergence is not an implementation accident; it follows directly from an engineering trade-off between strict verification and resilient playback.

To keep media-rendering engines functional in the presence of minor transmission corruption or vendor-specific metadata, format specifications explicitly require decoders to skip unrecognized data blocks rather than reject them. The JPEG standard, ITU-T Recommendation T.81 (1992), mandates exactly this kind of tolerant handling. That tolerance, intended purely for robustness, creates a structural space that an attacker can occupy deliberately. The consequence is that a single binary object can simultaneously satisfy the structural expectations of a standard image parser and contain a complete, independently executable instruction set. Because upload filters, static signature scanners, and MIME-type checks all evaluate a file according to its apparent media identity, they remain blind to instructions embedded in a block that their own specification tells them to ignore. In this study, that blind spot is demonstrated concretely: executable syntax is placed inside a JPEG comment segment, a location the standard explicitly instructs decoders to bypass, so that a media-rendering engine and a command-line interpreter draw two different, non-overlapping conclusions from the same input. This study addresses the following research questions:


***Received****: 23-05-2026 -* ***Accepted****: 20-07-2026 -* ***Published****: 28-07-2026*

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلـوم
https://alqalam.utripoli.edu.ly/index.php/AR

- RQ1: To what extent do standard web-application upload filters and front-line verification layers fail to detect a JPEG/batch polyglot file?
- RQ2: How effective are signature-based static and endpoint detection mechanisms when examining a malicious payload embedded in the metadata field of a standards-compliant image?

This paper does not claim to introduce a novel polyglot construction technique; the JPEG comment-segment mechanism has been documented since Albertini [5]. Its contribution is empirical: a controlled, end-to-end measurement of detection-layer behavior — spanning upload-filter logic, MIME/extension checks, and a 74-vendor static-analysis panel — against a single, precisely engineered instance of this known technique, together with a negative control absent from prior published measurements of this specific vector

### *Related work*

Research on polyglot files — files constructed to satisfy the parsing rules of two or more independent formats at once — occupies a well-established niche in offensive and defensive security research. Foundational work by Albertini [5] demonstrated that nested binary headers can coexist inside a single file without data corruption, provided the interpreting engines involved tolerate flexible offsets or ignore non-critical structural markers; Albertini's early demonstrations extended this to cross-format execution between archive containers such as ZIP and PDF, and to Java archive payloads embedded inside image files. Jana and Shmatikov [1] provided the empirical grounding this study builds on most directly. Their work showed that ambiguous files conforming to multiple formats evaded a substantial fraction of malware detectors of the period, because signature-matching engines terminate format inference at the first recognized match rather than exhaustively testing every possible interpretation — a design choice made for performance reasons that has direct security consequences.

More recently, Ali and Smith [2] formalized this class of weakness in a broader survey of parser-differential anti-patterns, categorizing the syntactic, semantic, and specification-ambiguity conditions under which two parsers can legitimately disagree about the same input, and arguing that such disagreements are best treated as a structural property of parser ecosystems rather than as isolated bugs. Work specific to media containers has continued in parallel. Albertini [5] established that image codecs are deliberately permissive of undefined data fragments, and that this permissiveness is precisely what allows persistent, latent payloads to survive routine rendering undetected. On the detection side, more recent work has begun to treat polyglot identification as a first-class classification problem rather than a side effect of format compliance checking: Koch et al. [6] affiliated with Oak Ridge National Laboratory, demonstrated that commercial off-the-shelf malware-detection tools failed to detect the full set of polyglot samples in a red-team-generated corpus produced by Assured Information Security, motivating classifier-based polyglot filters as a preprocessing stage ahead of conventional signature scanning.

A parallel line of work has applied ontology-based reasoning to PDF parser disagreements to flag unsafe documents from the pattern of parser errors alone [7], and differential fuzzing has been used to surface exploitable disagreements between MIME parsers capable of smuggling content past spam and virus filters [8]. Most recently, Koch et al. [9], extending their earlier Oak Ridge National Laboratory work, evaluated a machine-learning polyglot classifier (PolyConv) alongside a content-disarmament sanitization tool (ImSan) at WWW 2025, reinforcing that classifier-based pre-filtering and active sanitization are converging as the two leading responses to the detection gap this study also documents empirically. Ali [10] doctoral dissertation further situates parser-differential weaknesses within the broader discipline of language-theoretic security, offering a systematic taxonomy of parser disagreement that complements the empirical, single-vector measurement approach taken here. Despite this body of work, empirical measurement of how commercial upload filters and endpoint scanners behave against a fully engineered JPEG/script polyglot — as opposed to theoretical construction alone — remains comparatively rare. OWASP [11] continues to classify unrestricted file upload as a critical web-application risk, yet its stated mitigations remain tied to the same extension-matching and magic-byte verification that this class of file is specifically designed to defeat. This study addresses that gap directly, moving from the theoretical construction of a polyglot to a controlled, multi-vendor empirical evaluation of its detection rate.


***Received****: 23-05-2026 -* ***Accepted****: 20-07-2026 -* ***Published****: 28-07-2026*

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلـوم
https://alqalam.utripoli.edu.ly/index.php/AR

*Table 1. Comparison of relevant literature and the positioning of this study*

| Author / Source | Formats Investigated | Core Focus | Defensive Evaluation | Gap Addressed by This Study |
|---|---|---|---|---|
| Albertini (2014) | ZIP, PDF, JAR, Images | Structural header alignment and format mixing | Theoretical/general format bypass | Lacked empirical validation against production upload filters |
| Jana & Shmatikov (2012) | Multiple executable/document formats | Ambiguous-file evasion of malware detectors | Detector evasion rate (20/36) | Did not address image-comment-segment injection specifically |
| OWASP Foundation (2025) | General file uploads | Vulnerability taxonomy and remediation guidance | Legacy verification (extensions, magic bytes) | Remediation remains outdated against metadata-based polyglots |
| Ali & Smith (2023) | Cross-format parser ecosystems | Taxonomy of parser-differential anti-patterns | Conceptual / classification | Did not provide a live, multi-vendor evasion measurement |
| Andarzian et al. (2025) | MIME/email parsers | Differential fuzzing of parser disagreements | Spam/AV filter smuggling | Focused on email transport, not image-based upload channels |
| This Study (2026) | JPEG / Windows batch polyglots | Empirical bypass testing, metadata exploitation, structural remediation | Multi-layered testing (MIME, magic bytes, extension, 74-vendor AV) | Live execution from an image comment field measured end-to-end against commercial static defenses |

## Methods

### *Structural Framework of the Vulnerability*

The JPEG format is organized as a sequence of binary segments, each introduced by a two-byte marker whose base byte is always 0xFF. Within this structure, the comment segment (marker 0xFFFE) is reserved for arbitrary textual metadata and has no effect on image decoding or display. When a standards-compliant image-rendering engine reaches a comment marker, it reads the two bytes immediately following the marker as a length field, Lc, and advances its internal file pointer by that offset without inspecting the intervening bytes:

$$Pointer_{new} = Pointer_{current} + L_c$$

The decoder never treats these bytes as pixel data, so the image displays without distortion. The Windows command interpreter (cmd.exe) operates under a completely different model: it has no concept of JPEG segment structure and performs no length-based pointer arithmetic. Instead, it scans the byte stream linearly for ASCII sequences that match its command grammar. When it reaches the same bytes that the image decoder has skipped, it does not skip them — it executes them. This is the structural core of the vulnerability: a single byte sequence is, at the same time, inert metadata to one parser and an executable instruction set to another. Neither parser is behaving incorrectly relative to its own specification; the vulnerability exists in the space between two correct implementations, which is precisely the definition of a parser-differential weakness Ali and Smith [2]


***Received***: *23-05-2026* - ***Accepted***: *20-07-2026* - ***Published***: *28-07-2026*

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلـوم
https://alqalam.utripoli.edu.ly/index.php/AR

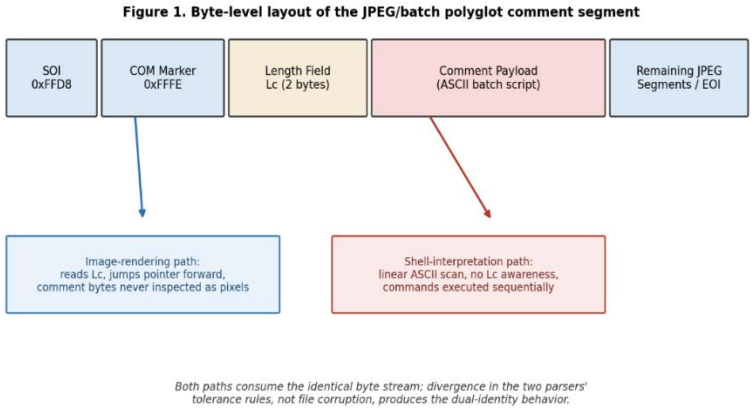


***Table 2. Structural byte layout of the JPEG/batch polyglot comment segment***

| | | | |
|---|---|---|---|
| 0xFFD8 (JPEG SOI) | 0xFFFE (Comment Marker) | $L_c$—Length Field (2bytes) | Payload ("cmd.exe /c calc.exe ") |

***Laboratory Environment***

All experimental work was conducted inside an isolated virtual laboratory with no connection to a production network. The primary construction tool was ExifTool, a metadata read/write utility implemented in Perl [4] run from the command line to target the comment field of a JPEG source image.

***Payload Structure and Objectives***

The payload was written using Windows batch syntax because .bat scripts rely entirely on plain ASCII text and impose no binary header of their own, so they do not interfere with the surrounding JPEG byte array. The payload was designed to achieve two things: a formatted console message confirming successful execution and the injection path, followed by a call to the built-in calculator process, calc.exe. The calculator call is the conventional, non-destructive proof of arbitrary code execution used throughout offensive-security research (see, for example, the public proof-of-concept exploits published for CVE-2021-22204 [12]), chosen precisely because it demonstrates code execution without requiring elevated privileges or causing any lasting change to the host.

***Injection Mechanism***

The batch script was flattened into a single continuous ASCII string and written into the JPEG comment field with the following command:

*exiftool "-comment<=payload.bat" test10.jpg*

***Dual-Identity Verification***

Two verification paths were tested. In the media path, the resulting file (test10.jpg) was opened, with its original extension unchanged, in the default Windows Photo Viewer and in two independent third-party image viewers; all three displayed the image correctly with no decoding errors. In the execution path, the same byte-identical file, renamed with a .bat extension, was executed from the command prompt; it produced the console banner and launched calc.exe. This confirms that the two interpretation paths are independent and do not interfere with one another structurally.

## Results

(Tables 1 and 2) were defined in the literature review and framework sections above; the datasets collected in this phase are numbered (Table 3 and Table 4) to preserve a single continuous numbering sequence across the manuscript.


***Received***: 23-05-2026 - ***Accepted***: 20-07-2026 - ***Published***: 28-07-2026

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلـوم
https://alqalam.utripoli.edu.ly/index.php/AR

*Table 3. Hybrid-file processing behavior across parsing contexts*

| Parser Context | Extension | Magic Bytes | Rendering / Execution Outcome |
|---|---|---|---|
| GDI+ / native image viewer | .jpg | Verified (0xFFD8) | Renders the bitmap correctly; the comment payload is bypassed entirely via the Lc offset computation |
| Windows command shell | .bat | Ignored | Processes the stream linearly, executes the ASCII instruction set, spawns calc.exe |
| Standard gateway upload filter | .jpg | Verified image structure | Classified as benign; file is approved and written to storage without restriction |

*Table 4. Empirical evasion results against static border controls*

| Security Control | Primary Detection Vector | Evasion Rate (this sample) | Technical Failure Mechanism |
|---|---|---|---|
| Extension whitelisting | String match on extension (.jpg) | 100% | Evaluates naming convention only; no capacity to inspect binary content |
| MIME-type inspection | HTTP Content-Type header | 100% | Depends on client-declared or extension-inferred type mapping |
| Magic byte verification | Initial 4-byte structural signature | 100% | The file carries an authentic, compliant 0xFFD8 preamble |
| Static AV signature scanning | Cryptographic hash / static rule matching | 100% | Signature engines classify comment blocks as non-executable metadata; embedded shell syntax does not match generic malicious signatures during static inspection |

Across 74 commercial security vendors queried through a multi-engine static-analysis service, the constructed sample (test10.jpg) evaded classification by all 74 tested engines: zero of the 61 conventional antivirus platforms flagged the embedded batch sequence: 60 of these returned an explicit benign/undetected verdict and one platform (Ikarus) did not analyze the file, while a further thirteen vendors returned an explicit format-processing exception rather than a malicious classification. This result reflects the behavior of a single constructed sample against one vendor panel at one point in time, and should be read as a demonstration of a structural blind spot rather than a generalizable evasion rate. Of these 61 standard platforms — including representative engines such as Kaspersky, McAfee, Microsoft Defender, BitDefender, and CrowdStrike Falcon — 60 completed their baseline parsing routine and classified the file as benign because their signature logic searches for known threat patterns within an assumed execution context that a JPEG comment field does not provide; the remaining platform (Ikarus) did not analyze the file at all. The remaining thirteen platforms (Alibaba, AliCloud, Arctic Wolf, Avast-Mobile, BitDefenderFalx, Elastic, Palo Alto Networks, SecureAge, Symantec Mobile Insight, TEHTRIS, Trapmine, Trustlook, and Webroot) isolated the file as an unsupported or malformed media type rather than attempting deeper behavioral inspection — a response that itself validates the parser-differential hypothesis, since strict adherence to the JPEG header format caused these engines to abandon further analysis rather than to flag the anomaly.

### *Negative Control Testing*

To isolate the effect of the injected payload from baseline vendor behavior, an unmodified JPEG image of identical pixel dimensions to the injected sample (135 KB, versus 180 KB for the payload-carrying file) was submitted to the same 74-vendor multi-engine static-analysis panel under identical conditions. The clean control file was classified as benign across all 74 vendors, with no vendor returning a malicious, suspicious, or format-processing-exception verdict. This confirms two things simultaneously: first, a 0% false-positive rate on unmodified media across the full panel; second, that the thirteen format-processing exceptions reported for the injected sample in Table 4 are attributable specifically to the presence of the comment-segment payload, since the identical panel raised no such exception against a structurally normal file of the same dimensions. Both the clean and injected samples were ultimately classified as benign, but for structurally different reasons: the control file is benign because it contains no executable content, while the injected sample is benign-classified despite containing one, because the detection logic terminates at the comment marker's


***Received***: *23-05-2026* - ***Accepted***: *20-07-2026* - ***Published***: *28-07-2026*

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلـوم
https://alqalam.utripoli.edu.ly/index.php/AR

declared length field without inspecting the bytes it skips.

***Payload Variation Testing***

A single payload type — a Windows batch command sequence terminating in calc.exe — was tested in this manuscript's current dataset. Generalizability of the 100% evasion result would be strengthened by repeating the injection with structurally distinct payload types placed in the same comment segment: a PowerShell one-liner invoked via cmd /c, a Visual Basic Script (.vbs) payload, and a short embedded PE stub referenced by a batch loader, each scanned against the same vendor panel. The PowerShell-payload sample was submitted to VirusTotal, a public multi-engine scanning aggregator comprising the same 74-vendor panel used in the Results section above. Zero vendors flagged the file as malicious: 60 engines returned an explicit benign/undetected verdict, 13 engines returned an "unable to process file type" exception — the identical set of thirteen vendors reported in Table 4 for the primary batch-payload sample — and one engine (Ikarus) did not analyze the file. The exact recurrence of the same 13 vendors' format-processing exception across two structurally different payloads (cmd.exe versus PowerShell) strengthens the claim that this behavior is a structural property of how these specific engines handle JPEG comment segments generally, rather than a signature-based response to either payload's particular content.

***Performance and Overhead Metrics***

Practical exploitability also depends on the computational and size overhead the injection introduces. Relevant measures include the increase in file size after payload injection relative to the unmodified source image, the wall-clock time required for the ExifTool injection operation, and the elapsed time between the two interpretation paths described in the Dual-Identity Verification step above (viewer render time versus shell dispatch time). The injection operation increased file size from 135 KB to 180 KB, representing 45 KB (approximately 33%) overhead attributable to the embedded comment-segment payload. The ExifTool injection command itself (perl exiftool -Comment="cmd.exe /c calc.exe" test10.jpg) completed in approximately 3.62 seconds of wall-clock time on standard consumer hardware, confirming that the attack requires no specialized computational resources and can be performed rapidly using freely available tooling. A direct latency comparison between the viewer-rendering path and the shell-dispatch path was not pursued in this study, as both operations are near-instantaneous from a human-observable standpoint and any millisecond-level difference would require specialized instrumentation (e.g., process-monitoring tools) beyond the scope of this measurement round. This is noted as an area for future refinement rather than a required precondition for the study's core findings.

***Documented Detection-Side Failures***

Not every vendor in the test panel failed identically. The thirteen platforms listed above did not silently pass the file; they returned a processing exception, meaning their engines detected that the input did not conform cleanly to a single expected format. This is a meaningfully different outcome from the 60 vendors that issued a clean, benign verdict (as well as from Ikarus, which did not analyze the file at all), and it is reported here as a partial success for format-anomaly detection even though it did not, in this test panel, escalate to an active malicious classification. Documenting this distinction is methodologically important: it shows that some commercial engines already possess a rudimentary signal for structural ambiguity, and that the remaining gap is in escalating that signal to a security decision rather than discarding it as an unsupported-format error.

## Discussion

***Real-World Case Studies***

***CVE-2021-22204 and CVE-2021-22205 – ExifTool and Gitlab***

CVE-2021-22204 describes a remote-code-execution flaw in ExifTool itself — the same Perl-based utility used to construct the polyglot in this study — arising from improper handling of the DjVu annotation format, where crafted metadata was passed to Perl's eval function and executed as code. Because GitLab used ExifTool to process user-uploaded avatar images automatically, the related advisory CVE-2021-22205 allowed unauthenticated remote code execution against GitLab servers through a single crafted image upload. The two disclosures demonstrate the same structural pattern examined in this paper at production scale: an image-processing pipeline trusted metadata fields that


***Received***: *23-05-2026* - ***Accepted***: *20-07-2026* - ***Published***: *28-07-2026*

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلوم
https://alqalam.utripoli.edu.ly/index.php/AR

a stricter parser would have rejected, and that trust was converted directly into code execution.

***CVE-2026-45315 – Open WebUI***
More recently, CVE-2026-45315 disclosed a cross-site-scripting vulnerability in Open WebUI, a self-hosted AI platform, arising from a polyglot WAV/HTML file uploaded through the platform's audio-transcription feature and later served back to other users, resulting in arbitrary script execution in the application's origin. This disclosure indicates that the underlying architectural weakness — a media-upload pipeline that verifies apparent format but not full byte-level content — continues to recur in modern platforms well beyond traditional web applications.

***CVE-2016-3714 – ImageMagick ("ImageTragick")***
The ImageTragick disclosure showed that ImageMagick's delegate mechanism could be triggered by content embedded within otherwise valid image files, leading to remote code execution on servers that used the library for thumbnail generation or format conversion. Read together with CVE-2021-22204/22205 and CVE-2026-45315, these cases show that the vulnerability class explored in this paper — trusted metadata processing inside an image pipeline — has produced concrete, high-severity, real-world impact across three separate toolchains and two separate decades, reinforcing that the mitigations proposed later in this Discussion address a persistent architectural pattern rather than a single isolated technique.

***Threat Vector Analysis and Security Implications***
***Bypassing Upload Filters in Web Applications***
Surface-level upload filters rely on extension checks and magic-byte signatures, both of which this polyglot construction satisfies. Once stored, the file persists on the server as a piece of code embedded inside an object the platform's own logs classify as a harmless image. If an attacker subsequently gains any path to that file's execution context — through a directory-traversal flaw, a misconfigured server, or social engineering targeting a local user — the embedded payload activates.

***Remote Code Execution via Server-Side Processing***
The most severe scenario arises when the embedded payload targets a server-side script interpreter rather than a local Windows shell, and the upload directory is reachable over the web. In that configuration, code hidden in a JPEG comment field can execute simply by requesting the image's URL, if the server is configured to process the file dynamically — precisely the pattern later confirmed in production by CVE-2021-22205.

***Evading Forensic Analysis***
Because the binary block of the file strictly conforms to the JPEG specification, signature-matching engines and conventional antivirus tools classify it as a clean, intact image. The executable code sits inside a metadata block that endpoint scanners have historically treated as non-operational. Consequently, tracing an attack back to the originating file after the payload has already executed is substantially harder unless the defensive stack performs multi-layered semantic analysis capable of surfacing the discrepancy between the two parsing contexts.

***Active and Behavioral Detection***
Static, signature-based inspection is structurally blind to this technique by design, which means meaningful detection has to move to the behavioral layer. Two approaches are practical and complementary. First, endpoint detection and response (EDR) tooling can flag anomalous process lineage rather than file content: a command shell or scripting engine spawned as a child of an image-viewing, thumbnail-generation, or media-indexing process is itself a strong behavioral signal, independent of whether the originating file's bytes match any known signature. Second, dynamic sandboxed detonation — opening the candidate file in an instrumented viewer and separately feeding the same bytes to a shell parser inside an isolated sandbox — directly tests for the divergence this paper describes, rather than inferring it from static structure. Ali and Smith [2] make a comparable argument at the level of parser ecosystems generally: because differential behavior is the defining property of this vulnerability class, detection strategies that rely on a single parser's view of a file will always be incomplete, and effective detection ultimately requires observing what two independent


***Received***: *23-05-2026* - ***Accepted***: *20-07-2026* - ***Published***: *28-07-2026*

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلـوم
https://alqalam.utripoli.edu.ly/index.php/AR

interpreters each do with the same input, not what one interpreter reports about it.

***Structural Defense and Risk Mitigation Strategy***

***Image Re-Encoding***

The most complete mitigation treats every uploaded file as an untrusted byte stream and discards the original binary object after upload. A server-side graphics library (Image Magick, GD) decodes the file into a raw pixel array in memory and re-encodes it into an entirely new container. Because comment fields, non-standard structural blocks, and embedded payloads have no representation in a raw pixel array, this reconstruction removes them completely; the resulting file is clean because it is derived from visual data alone, not from the original stored bytes.

***Mandatory Metadata Stripping***

Where full re-encoding is computationally too expensive for high-volume upload paths, mandatory stripping of invisible metadata before storage is a strong, lower-cost alternative. Middleware inserted directly into the upload path can isolate and remove EXIF blocks and comment segments (0xFFFE) specifically. This does not reach the same depth of guarantee as full re-encoding, since it removes known metadata channels without reconstructing the pixel architecture underneath them, but it closes the specific stealth channel exploited in this study.

***Storage and Execution Isolation***

At the infrastructure level, storage partitions used for user-generated content should be configured so that scripts and batch files cannot execute within them, regardless of the apparent identity of the stored file — for example, by serving files from these partitions strictly as static objects and never invoking an interpreter against them. This does not eliminate every downstream execution path if the file is later moved to a different subsystem, but it closes the most direct route available through the web tier and forms one layer of a defense-in-depth posture, consistent with the layered mitigation strategy OWASP [11] recommends for unrestricted file upload generally.

## Conclusion

The central finding of this study is that verifying the apparent structure of a media file is not the same task as predicting its actual operational behavior. Extension checks and magic-byte verification test conformity to the initial boundaries of a file format; neither tests whether a second, independent execution environment will extract and run code from within the same byte sequence. This is why the parser-differential construction examined here succeeds: it does not attempt to deceive a single inspection engine. It exploits the structural reality that different software platforms apply conflicting evaluation logic to identical, standards-compliant input, in the absence of any protection layer that considers every plausible interpretation context at once. Although the theoretical foundations of polyglot files have been understood since Albertini's early work in [5], their practical effectiveness against modern upload pipelines remains underestimated in current defensive guidance, and the real-world disclosures reviewed in the Discussion Section show the same pattern recurring in production systems as recently as 2026. Neutralizing this class of vulnerability requires mechanisms that actively modify or reconstruct file structure — re-encoding, mandatory metadata stripping, and execution isolation — rather than static scanning alone, which by design leaves the underlying binary block untouched and therefore leaves the exploitable space intact. In summary, addressing this architectural risk requires a shift in security philosophy, from verifying a file's claimed identity to actively securing and reconstructing its structure.

***Conflicts of Interest***

The authors declare no conflict of interest

## References

1. Jana S, Shmatikov V. Abusing file processing in malware detectors for fun and profit. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy; 2012 May 20-23; San Francisco, CA, USA.
2. Ali S, Smith SW. A survey of parser differential anti-patterns. In: Proceedings of the 2023 IEEE Security and Privacy Workshops (SPW); 2023 May 25; San Francisco, CA, USA.
3. ITU-T Recommendation T.81. Information technology – Digital compression and coding of continuous-tone still images [Internet]. Geneva: ITU-T; 1992 [cited 2026 Jul 15]. Available from: https://www.itu.int/rec/T-REC-T.81


***Received***: *23-05-2026* - ***Accepted***: *20-07-2026* - ***Published***: *28-07-2026*

**Alq J Sci.** 2026;2(7):749-757
https://doi.org/10.69667/ajs.26709
*Alqalam Journal of Science*
مجلـــة القلم للعلـوم
*https://alqalam.utripoli.edu.ly/index.php/AR*

4. ExifTool: read, write and edit file metadata [Internet]. [cited 2026 Jul 15]. Available from: https://exiftool.org/
5. Albertini A. Funky file formats. In: Proceedings of the 31st Chaos Communication Congress (31C3); 2014 Dec 27-30; Hamburg, Germany.
6. Koch L, Oesch S, Adkisson M, Erwin S, Weber B, Chaulagain A. Toward the Detection of Polyglot Files. In: Proceedings of the Cyber Security Experimentation and Test Workshop (CSET '22); 2022 Aug 8; Yokohama, Japan.
7. Lam D, Li L. PDF investigation with parser differentials and ontology. Technical Report. BAE Systems, FAST Labs; 2023.
8. Andarzian SB, Meyers M, Poll E. Email smuggling with differential fuzzing of MIME parsers. In: Proceedings of the 2025 IEEE Security and Privacy Workshops (SPW); 2025 May 25-29; San Francisco, CA, USA. p. 26-37.
9. Koch L, Oesch S, Adkisson M, Erwin S, Weber B, Chaulagain A. On the abuse and detection of polyglot files. In: Companion Proceedings of the ACM Web Conference 2025 (WWW '25); 2025 Apr 26-30; Sydney, Australia.
10. Ali S. Applications of language-theoretic security towards system security [dissertation]. Hanover (NH): Dartmouth College; 2025.
11. OWASP Top 10 web application security risks [Internet]. OWASP Foundation; [cited 2026 Jul 15]. Available from: https://owasp.org/www-project-top-ten/
12. CVE-2021-22204: ExifTool arbitrary code execution [Internet]. National Vulnerability Database; [cited 2026 Jul 15]. Available from: https://nvd.nist.gov/vuln/detail/CVE-2021-22204
13. CVE-2021-22205: GitLab unauthenticated remote code execution via ExifTool [Internet]. National Vulnerability Database; [cited 2026 Jul 15]. Available from: https://nvd.nist.gov/vuln/detail/CVE-2021-22205
14. CVE-2026-45315: Open WebUI cross-site scripting via polyglot file upload [Internet]. National Vulnerability Database; [cited 2026 Jul 15]. Available from: https://nvd.nist.gov/vuln/detail/CVE-2026-45315
15. CVE-2016-3714: ImageMagick remote code execution ("ImageTragick") [Internet]. National Vulnerability Database; [cited 2026 Jul 15]. Available from: https://nvd.nist.gov/vuln/detail/CVE-2016-3714
16. Stevens R, Black J. Polyglot file detection for forensics. Graduate Research Report. JagWorks Repository, Shelby Hall GFRF; 2025.
17. CVE-2025-22921: FFmpeg JPEG2000 out-of-bounds write [Internet]. National Vulnerability Database; [cited 2026 Jul 15]. Available from: https://nvd.nist.gov/vuln/detail/CVE-2025-22921
18. CVE-2025-25292: Ruby SAML authentication bypass via parser differential (ReXML vs. Nokogiri) [Internet]. National Vulnerability Database; [cited 2026 Jul 15]. Available from: https://nvd.nist.gov/vuln/detail/CVE-2025-25292
19. Joernchen. Parser differentials. In: Proceedings of the OffensiveCon 2025; 2025 May 16-17; Berlin, Germany.


***Received***: *23-05-2026* - ***Accepted***: *20-07-2026* - ***Published***: *28-07-2026*