Watters & Jacobs, LLP
Litigation and Trial Law Firm
+1 (650) 542-0057

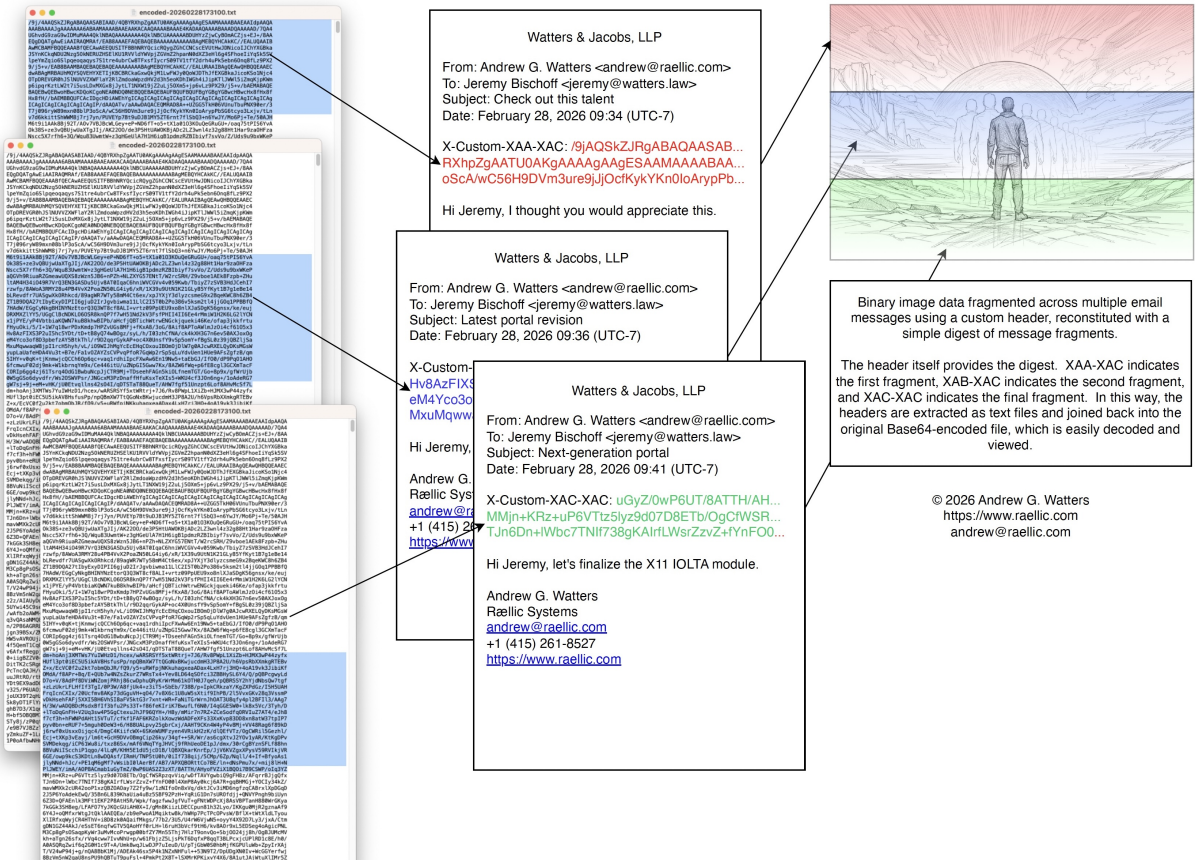
555 Twin Dolphin Dr., Ste. 135
Redwood City, CA 94065
info@watters.law

MOSAIC

An automated covert electronic distribution and payment system using a digest of reconstituted message fragments.

By: Andrew G. Watters[1] and Jeremy Bischoff[2]

April 13, 2026[3]



[1]Managing Partner, Watters & Jacobs, LLP. andrew@watters.law

[2]Software Engineer, Watters & Jacobs, LLP. jeremy@watters.law

[3]Third white paper release (v1.2). Adds blockchain ledger module, pattern-agnostic detection engine, and updated discussion. Prior releases: v1.0 (original), v1.1 (chunk randomization, improved shell script).

Executive Summary

MOSAIC is a proof-of-concept system that demonstrates how arbitrary binary files (images, documents, executables), in particular CSAM, can be covertly distributed and paid for through ordinary email traffic– without appearing as attachments, without triggering attachment-based filters, and without any visible indication in the email body that data is being transferred or payments are being made. A system like this, or one similar to it, could plausibly have been used by Jeffrey Epstein and his accomplices to distribute illicit content such as CSAM while evading conventional email monitoring.

Due to the ease of creating the system– essentially one day of shell scripting for a functional prototype– along with the convenience and utility of the system, the authors assess it is highly probable this method already exists in the wild. Law enforcement would need to have access to the raw source of targets' emails to determine whether any additional data is being sent– a difficult task without an already-pending investigation supported by probable cause.

While it is preferable to the sender to have their own email server in order to transfer files from a server-side queue, the sender does not actually have to have their own server– they only need access to a server that can send email from the command line[4]. The challenge is compounded by the fact that no one is looking for this type of traffic on port 25 (SMTP) or port 993 (IMAP with SSL/TLS), as it is not suspicious compared to the well-known Tor ports and VPNs that would drastically increase the target's detection profile. Finally, MOSAIC is nearly entirely automated, and is practically instantaneous, as well as convenient. As such, the concepts behind MOSAIC represent an important new avenue for investigators to detect the covert distribution of CSAM.

I. Technical Introduction

The core idea is straightforward. Every email message contains *headers*: structured lines of metadata that precede the message body. Standard headers include fields like `From`, `To`, `Date`, and `Subject`. The email standard (RFC 5322) also permits arbitrary *custom headers*, which are any header lines beginning with `X-`. Mail servers, spam filters, and email clients routinely ignore custom headers they do not recognize; they simply pass them through untouched.

MOSAIC exploits this by using custom headers to its advantage, following this high-level process that is enabled by standard command line utilities available on Linux and macOS– and that many Windows computers also have:

1. **Encode.** The input file is Base64-encoded, producing a stream of printable ASCII characters. This is the same encoding used by email attachments (MIME), but

[4]Virtually all Linux servers can send email from the command line with no additional configuration and with only built-in software. This is commonly done with automated system messages and notifications.

MOSAIC does not use the MIME attachment mechanism.

2. **Fragment.** The Base64 stream is split into N chunks of roughly equal size.
3. **Embed.** Each chunk is placed into a custom `X-Custom` header of a separate, otherwise ordinary-looking email message. The header name itself encodes the fragment's sequence number and the total fragment count.
4. **Transmit.** The emails are sent through any standard mail server (SMTP). To an observer or automated filter, they appear to be routine messages with unremarkable bodies.
5. **Reassemble.** The recipient (or an automated process) scans incoming emails for the custom headers, extracts the fragments, orders them by sequence number, concatenates the Base64 data, and decodes it back to the original binary file.
6. **Verify.** A SHA-256 digest embedded in each email's `X-Custom-Digest` header allows confirmation that the reassembled file is bit-for-bit identical to the original. (This step serves to validate the proof of concept rather than being essential to the distribution mechanism itself.)
7. **Invoice.** An arbitrary Bitcoin amount indicating what is being charged for the content appears in a `X-Custom-Invoice` header, so that the user sees how much the content costs. This permits the sender to keep track of the user's balance. This also could be triggered by a purchase order on a website that caused the content to be sent. Or the user could have a subscription that automatically sends them a certain amount of new CSAM per month and they can pay extra for more. Either way, this is all recorded in a blockchain ledger.

Because the payload never appears as an attachment and the email body can contain any plausible cover text, conventional detection methods (attachment scanning, file-type filtering, keyword analysis of message bodies) are ineffective. The data hides in a part of the message that virtually no monitoring tool inspects. The payment request is also in a custom header, which acts as an invoice for payment via Bitcoin or other cryptocurrency.

Header naming convention. MOSAIC identifies fragments using a structured header name of the form:

`X-Custom-`*`<SEQ>`*`-`*`<TOTAL>`*

where `SEQ` and `TOTAL` are three-character alphabetic codes. The encoding maps integers to letter pairs: $1 =$ `XAA`, $2 =$ `XAB`, ..., $26 =$ `XAZ`, $27 =$ `XBA`, and so on. For example, the header `X-Custom-XAC-XAH` identifies fragment 3 of 8. This convention allows up to 676 fragments (26×26) per transfer with a default fragment size of 1,000 lines, which is sufficient for files of substantial size (apx. 35 MB). This is more than enough space to transfer short videos or extremely high resolution still images. The file size can be increased to a theoretical maximum of 350 MB by increasing the fragment size to 10,000 lines or more, although that might increase the probability of detection through packet capture showing numerous suspiciously large email data files, all the same or a similar size. To address this, MOSAIC includes an optional chunk randomization mode (enabled via `--randomize`) that varies the

size of each fragment between 30% and 170% of the average chunk size. Each chunk receives a random size within that range, with feasibility clamping applied so that no chunk is too small to be useful or so large that remaining chunks cannot fit within their own bounds. The total data is preserved exactly; only the split points move. Because the decoder already reads each fragment's full header value regardless of size, randomized chunks require no changes to the reassembly process. With randomization enabled, a ten-chunk transfer that would normally produce ten identically-sized emails instead produces messages ranging from roughly 5 KB to 21 KB—traffic that more closely resembles ordinary varied email activity.

II. Demonstration Walkthrough

The `mosaic.sh` script includes a self-contained `demo` mode that runs the full encode–fragment–embed–reassemble–verify pipeline on any input file and produces a local directory of artifacts for inspection. The following walkthrough uses a small PNG test image (320×240 pixels, 1,718 bytes) to illustrate the process end to end.

Phase 1: File preparation. The demo begins by Base64-encoding the input image. The 1,718-byte binary file becomes a stream of printable ASCII characters wrapped to 79-character lines, the same line length used by standard MIME encoding. A SHA-256 digest of the original file is computed and stored for later verification:

```
SHA256: 4c535f1caeb220037f8c0867de15760b08eb927c
        71f20006c8ffe072482d9a2f
```

Phase 2: Fragmentation. The Base64 stream is split into five chunks of approximately six lines each (the number of chunks is arbitrary and configurable). Each chunk becomes one fragment that will be embedded in a separate email.

Phase 3: Email embedding. For each fragment, MOSAIC generates a complete RFC 2822-compliant email message. The fragment data is placed in a custom header; the message body contains only ordinary cover text. Below is an abbreviated example of the first email's headers:

```
From: MOSAIC System <mosaic-sender@example.com>
To: <mosaic-receiver@example.com>
Subject: MOSAIC [1/5]
Date: Fri, 06 Mar 2026 14:45:33 -0800
Message-ID: <mosaic-1-5-177283...@mosaic.local>
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8
X-Custom-XAA-XAE: iVBORw0KGgoAAAANSUhEUgAAAUA...
X-Custom-Digest: 4c535f1caeb220037f8c0867de...
```

The header `X-Custom-XAA-XAE` indicates this is fragment 1 of 5. The header's value contains the Base64 data for this chunk. The `X-Custom-Digest` header carries the SHA-256 hash of the original file, providing an integrity check independent of any single fragment.

Phase 4: Extraction and reassembly. The decoder scans each `.eml` file in the target directory, searching for headers matching the pattern `X-Custom-X[A-Z][A-Z]-X[A-Z][A-Z]`. When a matching header is found, the decoder parses the sequence number and total count from the header name, extracts the Base64 value, and stores the fragment keyed by its sequence number. After all emails have been processed, the fragments are concatenated in order and Base64-decoded to produce the output file.

Phase 5: Verification. The SHA-256 digest of the reassembled file is compared against the digest embedded in the email headers:

```
Original SHA256:    4c535f1caeb220037f8c0867de15760b...
Reassembled SHA256: 4c535f1caeb220037f8c0867de15760b...

SHA256 match: YES
Integrity:    PASS
```

The hashes match, confirming that the reassembled PNG is bit-for-bit identical to the original. The file type is independently verified via the file's magic bytes. This demonstrates that the full pipeline (encoding, fragmentation, embedding into email headers, extraction, and reassembly) is lossless.

III. Fragment Mode

The `fragment` command is the production counterpart of the demo. Where the demo runs the full pipeline locally for inspection, `fragment` is designed either to stage email files on disk for later transmission or to send them directly through an SMTP server in a single invocation.

The pipeline is identical to the demo through the first three phases: the input file is Base64-encoded, the resulting stream is split into N chunks (configurable via `--chunks` or `--lines-per-chunk`; an optional `--randomize` flag produces variable-size chunks as described in Section I), and each chunk is embedded in the `X-Custom` header of a separate RFC 2822 message. The emails are written to a local `mosaic_output/emails/` directory. If SMTP credentials are provided, a fourth phase transmits the emails in sequence.

Local-only fragmentation. When run without SMTP flags, `fragment` produces a directory of `.eml` files ready for manual sending or queuing:

```
$ ./mosaic.sh fragment --file secret.jpg --chunks 5 --randomize

 -- PHASE 1: FILE PREPARATION --
  Input file: secret.jpg (48205 bytes)
  SHA256:     a1b2c3d4e5f6...

 -- PHASE 2: FRAGMENTATION --
  Split into 5 fragments (randomized: 4-22 lines, avg ~13)

 -- PHASE 3: EMAIL GENERATION --
  Fragment 1/5 | Header: X-Custom-XAA-XAE |  6210 bytes
  Fragment 2/5 | Header: X-Custom-XAB-XAE | 17483 bytes
  ...
  Created 5 email messages in RFC 2822 format
```

SMTP transmission. When the `--smtp-server`, `--user`, `--pass`, `--from`, and `--to` flags are supplied, the script sends each `.eml` file through the specified server using `curl` with SSL/TLS. A one-second delay is inserted between messages to avoid rate-limiting. The subject line of each email contains a timestamped tag (e.g., `MOSAIC-1741592400`) and the fragment index, so the recipient's decoder can locate the correct messages later. Example:

```
$ ./mosaic.sh fragment --file secret.jpg --chunks 5 \
    --smtp-server smtps://mail.example.com:465    \
    --user alice --pass ****                       \
    --from alice@example.com --to bob@example.com

 -- PHASE 4: SENDING VIA SMTP --
  Sent msg_0001.eml
  Sent msg_0002.eml
  ...
  All 5 fragment emails sent successfully
```

At the conclusion of either path, the script prints a summary that includes the original file size, the number of fragments, the total size of all generated emails, the SHA-256 digest, and the subject tag.

IV. Decode Mode

The `decode` command reverses the fragmentation process. It accepts email messages from one of three sources (a local directory of `.eml` files, an IMAP server, or a remote Maildir accessed over SSH), extracts the MOSAIC fragment headers, reassembles the fragments in sequence-number order, Base64-decodes the concatenated stream, and writes the resulting binary file to disk.

Regardless of the source, the extraction logic is the same. The decoder reads the header section of each email (everything before the first blank line), identifies any header matching the pattern `X-Custom-X[A-Z][A-Z]-X[A-Z][A-Z]`, and handles RFC 2822 header folding (continuation lines beginning with whitespace). The sequence number and total count are parsed from the header name; the Base64 payload is stored keyed by sequence number. After all messages have been processed, the fragments are concatenated in order and decoded.

For IMAP retrieval, the decoder issues a `SEARCH` command filtered by subject tag (`--subject`) or date range (`--days`), downloads each matching message by UID, and feeds the results into the extraction pipeline. For SSH retrieval, the decoder runs a remote `grep` to identify files containing MOSAIC header patterns, transfers only the matching files via `scp`, and then extracts locally. In both cases only the relevant messages are downloaded, minimizing bandwidth and exposure.

Example: decoding from a local directory.

```
$ ./mosaic.sh decode --dir ./mosaic_output/emails/

 -- PHASE 1: SCANNING EMAIL FILES --
  Found 5 email files

 -- PHASE 2: EXTRACTION & REASSEMBLY --
  Extracted fragment 1/5 from msg_0001.eml
  Extracted fragment 2/5 from msg_0002.eml
  ...
  Extracted 5/5 fragments
  Reassembled file: 48205 bytes
  File type: JPEG image data

 -- VERIFICATION --
  Embedded SHA256: a1b2c3d4e5f6...
  Decoded SHA256:  a1b2c3d4e5f6...
  INTEGRITY VERIFIED - SHA256 matches embedded digest
```

The decoder automatically detects the reassembled file's type via its magic bytes and appends the appropriate extension (`.png`, `.jpg`, `.pdf`, etc.). If an `X-Custom-Digest` header is present in the source emails, the SHA-256 hash of the decoded file is compared against it and the result is reported. If the digest is absent, the decoder notes that integrity cannot be verified.

V. Scan Mode

The `scan` command is a detection tool. Rather than reassembling a file, it inspects a

corpus of email messages for the presence of MOSAIC fragment patterns and produces a report suitable for forensic review. It supports the same three data sources as `decode`: a local directory, an IMAP mailbox, or a remote Maildir over SSH.

For each email, the scanner reads the header section and tests for headers matching the MOSAIC naming convention. When a match is found, the scanner records the header name, the fragment's sequence and total numbers, the size of the Base64 payload, and any available metadata (subject line, date). Emails without matching headers are silently skipped.

Example: scanning a local directory.

```
$ ./mosaic.sh scan --dir ./mosaic_output/emails/

 -- MOSAIC DETECTION SCAN --
  Scanning ./mosaic_output/emails/ ...
  Analyzing 5 email files...

  msg_0001.eml
     Header:   X-Custom-XAA-XAE
     Fragment: 1 of 5
     Payload:  12940 bytes of base64 data
     Subject:  MOSAIC-1741592400 [1/5]

  msg_0002.eml
     Header:   X-Custom-XAB-XAE
     Fragment: 2 of 5
     ...

 -- DETECTION REPORT --
  MOSAIC FRAGMENT PATTERN DETECTED
  Emails scanned:      5
  Suspicious emails:   5
  Total payload:       64700 bytes (base64)
  Est. file size:      ~48525 bytes
```

The scan does *not* attempt to decode or reassemble the data; its purpose is detection only. This separation is deliberate: an investigator may wish to confirm the presence of covert traffic before committing to full extraction, and the scan can be run against an entire mailbox without writing any decoded content to disk.

The scan's three data-source modes mirror those of `decode`. Over IMAP, the scanner downloads all messages (or those within a specified date window via `--days`) and inspects each one. Over SSH, it first runs a remote `grep` against the Maildir to identify candidate files, downloads only those candidates, and then performs the detailed header analysis

locally. In both network modes the initial filtering step keeps bandwidth low and avoids transferring the full mailbox.

Scan mode is suitable for enhancement by A.I. to capture custom headers and analyze patterns in them to see whether payload data is included. Section VII describes a pattern-agnostic detection engine that applies five statistical heuristics to accomplish this without relying on MOSAIC's specific header naming convention. In any scenario, scan mode only requires access to the user's IMAP credentials, home directory (including Maildir), and/or SSH. MBOX format is currently not supported, but will be added in a future release.

VI. Blockchain Ledger

When the `--btc` flag is supplied to a `fragment` or `decode` command, MOSAIC records the transaction in a local append-only ledger stored as newline-delimited JSON (NDJSON). Each line in the ledger is a single JSON object representing one block, and each block contains the following metadata: the event type (`fragment` or `decode`), the file's SHA-256 hash, filename, size in bytes, fragment count, an invoice identifier, the Bitcoin amount, sender and recipient addresses, the subject tag, and a UTC timestamp.

Every block also stores the SHA-256 hash of the immediately preceding block. The first block's `prev_hash` field is set to a string of 64 zeroes (the genesis hash). This hash chain means that altering any field in any block changes that block's hash, which in turn invalidates every subsequent block in the chain. This is the same tamper-evidence principle that underpins Bitcoin's blockchain: no single record can be modified without detection.

The `ledger` command exposes four subcommands:

1. **`--list`** Displays every block in the ledger with its metadata, formatted for human review.
2. **`--verify`** Walks the chain from block 1 forward, recomputing each block's hash from its contents and the previous block's hash. If any stored hash does not match the recomputed value, the block is flagged as tampered. A summary reports how many blocks passed or failed verification.
3. **`--query HASH`** Searches the ledger for all entries whose file hash matches the given SHA-256 digest. This allows an investigator to trace every distribution and payment event associated with a particular file.
4. **`--tamper-demo`** Creates a temporary copy of the ledger, modifies a payment amount in the first block, and runs verification against the altered copy. The demonstration shows that even a single-field change causes both a content-hash mismatch and a broken chain link, confirming that undetected alteration is not possible. The original ledger is not affected.

Example: verifying chain integrity.

```
$ ./mosaic.sh ledger --verify

 -- BLOCKCHAIN VERIFICATION --
  Verifying hash chain integrity across 3 block(s)...

  Block #1  2026-04-08T19:30:12Z  [fragment] secret.jpg
    Chain link: prev_hash matches genesis block
    Entry hash: 7a3f1b... verified

  Block #2  2026-04-08T19:30:14Z  [fragment] report.pdf
```

```
    Chain link: prev_hash matches Block #1
    Entry hash: 4e8c2d... verified

  Block #3  2026-04-08T19:31:07Z  [decode] secret.jpg
    Chain link: prev_hash matches Block #2
    Entry hash: b19d5f... verified

  CHAIN INTEGRITY VERIFIED -- 3/3 blocks valid
```

The forensic value of the ledger is twofold. First, it provides an immutable record of every file distributed through MOSAIC and every payment associated with that distribution. Second, if investigators obtain a copy of the ledger, they can independently verify that the records have not been altered since they were created, which strengthens the evidentiary weight of the data.

VII. Pattern-Agnostic Detection

The `scan` command described in Section V relies on MOSAIC's specific header naming convention (`X-Custom-<SEQ>-<TOTAL>`). An adversary who changes the naming scheme defeats that scanner. The `detect` command takes a fundamentally different approach: it applies five statistical heuristics that identify header-channel exfiltration regardless of the header names used. It requires only a directory of `.eml` files as input.

H1: Header/body size ratio. In normal email, the message body carries the content and the headers are comparatively small metadata. When headers are used as a data channel, this ratio inverts: the header section may be ten or more times the size of the body. The detector computes the byte ratio of headers to body and scores the anomaly on a continuous scale.

H2: Base64 entropy density. Base64-encoded binary data exhibits a characteristic Shannon entropy near 6.0 bits per character and a high concentration of Base64-alphabet characters. The detector measures entropy and character distribution in each non-standard header value, scoring headers that exceed the range typical of natural-language metadata.

H3: RFC 2822 header folding abuse. The email standard permits header values to span multiple lines via continuation (lines beginning with whitespace). Normal headers rarely need more than one or two continuation lines. A header carrying kilobytes of Base64 data will produce many continuation lines with high entropy. The detector counts headers exhibiting this pattern.

H4: Unknown X-header proliferation. The detector maintains a whitelist of X-headers commonly added by mail servers, spam filters, and mailing list software (e.g., `X-Mailer`, `X-Spam-Status`). X-headers not on the whitelist are counted, and those with

high-entropy values are weighted more heavily.

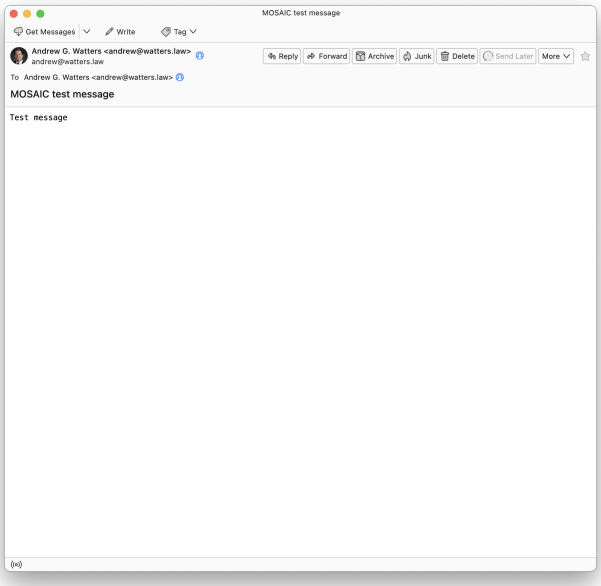
H5: Cross-message correlation. Heuristics H1 through H4 analyze individual emails. H5 operates across a corpus, clustering messages by sender/recipient pair and scoring each cluster on four signals: uniformity of header sizes (low variance is suspicious), overlap of non-standard header names across messages (high similarity is suspicious), similarity of message bodies (near-identical or trivially short bodies suggest automated generation), and temporal density (many messages sent within a narrow time window). H5 acts as a multiplier on the per-email scores within each cluster.

Each heuristic produces a score between 0.0 and 1.0. The per-email heuristics (H1 through H4) are combined as a weighted sum, and the H5 correlation score adjusts the composite upward when cross-message patterns are present. The final composite score maps to a risk level: LOW, MEDIUM, HIGH, or CRITICAL.

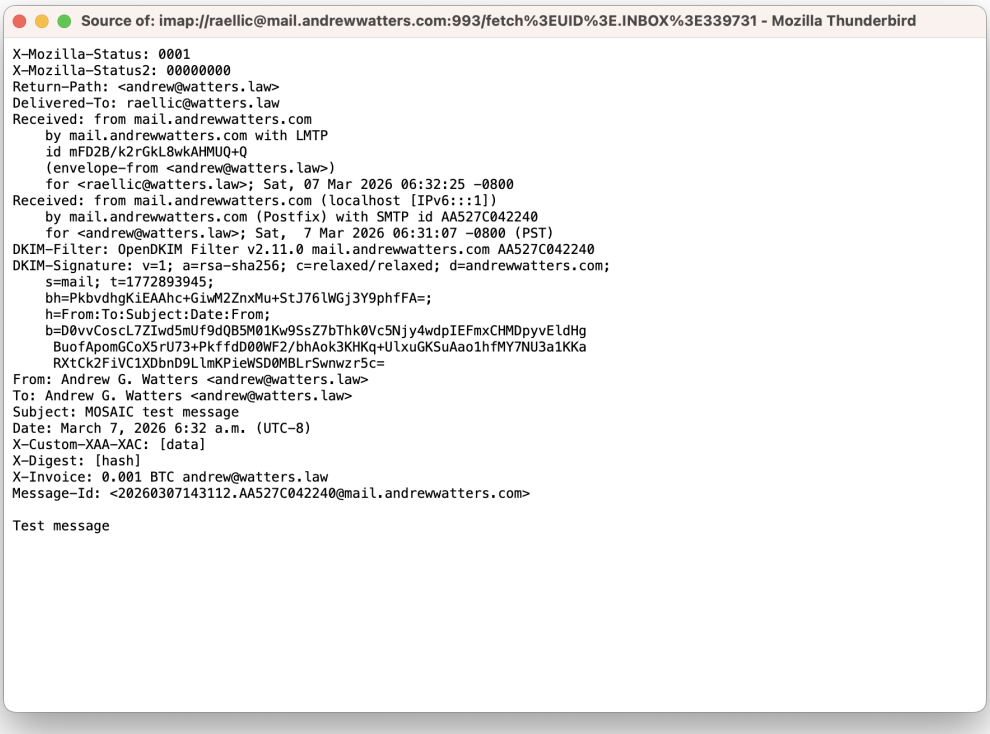
Because the heuristics are statistical rather than signature-based, false positives are possible. Emails with legitimately large custom headers (e.g., some enterprise tracking systems) may score in the MEDIUM range. The composite scoring is designed so that a single anomalous signal rarely pushes an email above MEDIUM; convergence across multiple heuristics is generally required to reach HIGH or CRITICAL.

Practical Demonstration

This screenshot depicts a connection via a SSH tunnel to the authors' email server. The user connects to the Postfix daemon from the command line via Telnet and manually sends an email by typing it into Telnet. The custom headers would be where the MOSAIC data goes, and can either be typed in manually, copied/pasted, or sent through automated processes.



This screenshot shows the email as received by the user's desktop email program.



This screenshot shows the source of the email in question. As indicated, the headers are there and can be any arbitrary length. The EML files created by the demonstration script (mosaic.sh) can actually be sent in this manner quite easily, as well, by simply displaying

them and piping the output to the Telnet session.

Conclusion

The current mosaic.sh interactive shell script is provided along with this white paper, and updates may be downloaded at the following link, as well:

https://www.watters.law/articles/attachments/mosaic.sh

The proof of concept demonstrates that the covert distribution of CSAM through email header fragments is entirely possible, easy, and convenient. The blockchain ledger module shows how such a system can record payments with tamper-proof integrity, and the pattern-agnostic detection engine shows that statistical analysis of email headers can identify this class of exfiltration even when the specific header naming scheme is unknown. It is highly probable that offenders are already using techniques like these to distribute and pay for CSAM.

The authors may be reached via email at andrew@watters.law, or by mail/phone as indicated in the cover page.

MOSAIC — -zsh — 123×79
Covert File Distribution via Email Header Fragmentation
by Andrew G. Watters and Jeremy Bischoff
Watters & Jacobs, LLP / Rællic Systems
Version 1.0 / https://www.watters.law
© 2026 Andrew G. Watters
Usage: mosaic.sh <mode> [options]
Modes:
demo Local demo (fragment + reassemble in one step)
fragment Fragment a file into email messages
decode Reassemble a file from email messages
scan Scan for MOSAIC fragment patterns
Options:
--file PATH Input file (required for demo, fragment)
--dir PATH Directory of .eml files (decode or scan)
--chunks N Number of fragments
--lines-per-chunk N Lines per fragment (default: 1000)
--subject TAG Subject tag for IMAP retrieval (decode)
--days N Search last N days via IMAP (decode or scan)
--smtp-server URL SMTP server URL (fragment mode)
--imap-server URL IMAP server URL (decode or scan)
--user USER Mail username
--pass PASS Mail password
--from ADDR Sender email address (fragment SMTP)
--to ADDR Recipient email address (fragment SMTP)
--btc AMOUNT Bitcoin amount to embed in invoice header
--invoice ID Invoice identifier (auto-generated if omitted)
--ssh USER@HOST SSH host for Maildir scan
--maildir PATH Maildir path on remote server
--help Show this help
Examples:
mosaic.sh demo --file photo.jpg
mosaic.sh demo --file secret.pdf --chunks 10
Fragment locally
mosaic.sh fragment --file secret.jpg --chunks 5
Fragment with invoice
mosaic.sh fragment --file secret.jpg --chunks 5 --btc 0.005 --invoice INV-0042
Fragment and send via SMTP
mosaic.sh fragment --file secret.jpg --chunks 5 \
--smtp-server smtps://mail.example.com:465 \
--user user --pass secret --from sender@example.com --to receiver@example.com
Decode from local .eml files
mosaic.sh decode --dir ./mosaic_output/emails/
Decode by pulling from IMAP (by subject tag)
mosaic.sh decode --imap-server imaps://mail.example.com:993 \
--user user --pass secret --subject MOSAIC-1234567890
Decode by pulling from IMAP (last 7 days)
mosaic.sh decode --imap-server imaps://mail.example.com:993 \
--user user --pass secret --days 7
Decode via SSH Maildir
mosaic.sh decode --ssh root@mail.example.com --maildir /home/vmail/domain/user/Maildir/
Scan
mosaic.sh scan --dir ./mosaic_output/emails/
mosaic.sh scan --imap-server imaps://mail.example.com:993 --user user --pass secret
mosaic.sh scan --imap-server imaps://mail.example.com:993 --user user --pass secret --days 7
mosaic.sh scan --ssh root@mail.example.com --maildir /home/vmail/domain/user/Maildir/
raellic@Andrews-MacBook-Pro MOSAIC %