# Notebook #3 — Independent WAF Validation

**Purpose:** Fully independent verification notebook that ingests a raw consolidated detections log and produces:

- Daily volume + 7-day baseline
- Spike day ranking vs baseline
- Scanner-IP table (suspicious %, active hours, first/last seen)
- Hostile Intensity Index (hourly)
- Optional enforcement recommendations (commands only; never executed)

**Safety:** HOME_IP is always excluded from any block candidate list.

In [1]:
```
from datetime import datetime, timezone
from IPython.display import Markdown, display

generated_utc = datetime.now(timezone.utc).strftime("%Y-%m-%d %H:%M:%SZ")

display(Markdown(f"""
**Generated:** {generated_utc}
"""))
```

**Generated:** 2026-03-04 20:45:02Z

## Cell 1 — Configuration (AUTHORITATIVE)

Edit paths and parameters here only. All downstream cells read from these globals.

In [2]:
```
# ==========================
# Cell 1 — Configuration (AUTHORITATIVE)
# ==========================

from pathlib import Path

# --- Required inputs ---
# Point this to your consolidated detections log (raw source of truth)
DETECTIONS_LOG = Path("/var/log/rust/detections.log")  # <-- change if neede

# Optional: restrict to a single vhost/source label found in log lines (e.g.
# Set to None to include all.
SOURCE_FILTER = None  # e.g. "astropema.ai_ssl_access"

# Time window controls (UTC)
WINDOW_HOURS = 24  # analyze last N hours (set to None to analyze entire fil
NOW_UTC_OVERRIDE = None  # e.g. "2026-01-11 16:54:52" (UTC). None uses curre
```

```
# --- Safety / policy ---
HOME_IP = "66.241.78.7"           # DO NOT BLOCK
TTL_SECONDS = 7 * 24 * 3600       # 7 days (recommendation only; notebook ne

# --- Thresholds ---
SCANNER_SUSP_PCT_THRESHOLD = 0.80 # "scanner-like" if >= 80% suspicious
MIN_EVENTS_PER_IP = 50            # include IPs with at least this many ever

# --- Output controls ---
PRINT_DEBUG = True

print("Config OK")
print("DETECTIONS_LOG:", DETECTIONS_LOG)
print("SOURCE_FILTER:", SOURCE_FILTER)
print("WINDOW_HOURS:", WINDOW_HOURS)
print("HOME_IP:", HOME_IP)
print("TTL_SECONDS:", TTL_SECONDS)
```

```
Config OK
DETECTIONS_LOG: /var/log/rust/detections.log
SOURCE_FILTER: None
WINDOW_HOURS: 24
HOME_IP: 66.241.78.7
TTL_SECONDS: 604800
```

# Cell 2 — Imports

Standard libraries only (pandas + matplotlib).

In [3]:
```
# ========================
# Cell 2 — Imports
# ========================

import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

pd.set_option("display.max_rows", 200)
pd.set_option("display.max_columns", 200)
pd.set_option("display.width", 220)

print("Imports OK")
```

```
Imports OK
```

# Cell 3 — Parse detections.log into df_events (CANONICAL)

This notebook is independent: it parses the raw consolidated detections log.

**Expected canonical columns:** `ts_utc`, `source`, `ip`, `method`, `path`, `status`, `verdict`, `reason`, `expected_action` (best-effort).

In [4]:

```
# ========================
# Cell 3 — Parse detections.log -> df_events (CANONICAL)
# ========================

from datetime import datetime, timezone, timedelta
from pathlib import Path

def _parse_ts_utc(s: str):
    try:
        return pd.to_datetime(s, utc=True)
    except Exception:
        return pd.NaT

# Primary pattern with Reason:
LINE_RE = re.compile(
    r'^\[(?P<source>[^\]]+)\]\s+\[(?P<ts>[^\]]+)\]\s+(?P<verdict>[A-Z_]+)\s+
    r'(?P<ip>[^|]+?)\s+\|\s+(?P<method>[A-Z]+)\s+(?P<path>[^|]+?)\s+\|\s+'
    r'(?P<status>\d+)\s+\|\s+Reason:\s*(?P<reason>[^|]+?)\s+\|',
    re.IGNORECASE
)

# Fallback if Reason isn't present
LINE_RE_FALLBACK = re.compile(
    r'^\[(?P<source>[^\]]+)\]\s+\[(?P<ts>[^\]]+)\]\s+(?P<verdict>[A-Z_]+)\s+
    r'(?P<ip>[^|]+?)\s+\|\s+(?P<method>[A-Z]+)\s+(?P<path>[^|]+?)\s+\|\s+'
    r'(?P<status>\d+)',
    re.IGNORECASE
)

def norm_ip(x):
    return str(x).strip().split("/")[0]

def load_events_from_detections_log(path: str | Path) -> pd.DataFrame:
    path = Path(path)
    if not path.exists():
        raise FileNotFoundError(f"detections log not found: {path}")
    rows = []
    with path.open("r", errors="replace") as f:
        for line in f:
            line = line.rstrip("\n")
            m = LINE_RE.search(line) or LINE_RE_FALLBACK.search(line)
            if not m:
                continue
            d = m.groupdict()
            ts = _parse_ts_utc(d.get("ts", ""))
            rows.append({
                "ts_utc": ts,
                "source": (d.get("source") or "").strip(),
                "ip": norm_ip(d.get("ip") or ""),
                "method": (d.get("method") or "").strip().upper(),
                "path": (d.get("path") or "").strip(),
                "status": int(d.get("status") or 0),
                "verdict": (d.get("verdict") or "").strip().upper(),
                "reason": (d.get("reason") or "").strip(),
                "raw": line,
```

```
            })
    df = pd.DataFrame(rows)
    if df.empty:
        raise RuntimeError("Parsed 0 events. Update LINE_RE patterns for you
    df = df.dropna(subset=["ts_utc"]).copy()
    df["ip_norm"] = df["ip"].map(norm_ip)
    return df

df_events = load_events_from_detections_log(DETECTIONS_LOG)

if SOURCE_FILTER:
    df_events = df_events[df_events["source"] == SOURCE_FILTER].copy()

now_utc = pd.Timestamp.utcnow() if NOW_UTC_OVERRIDE is None else pd.to_datet
if WINDOW_HOURS is not None:
    start_utc = now_utc - pd.Timedelta(hours=int(WINDOW_HOURS))
    df_events = df_events[(df_events["ts_utc"] >= start_utc) & (df_events["t
else:
    start_utc = df_events["ts_utc"].min()

df_events["day_utc"] = df_events["ts_utc"].dt.floor("D")
df_events["hour_utc"] = df_events["ts_utc"].dt.floor("h")

# Analysis-only: map suspicious to "BLOCK" expected action
df_events["expected_action"] = np.where(df_events["verdict"].eq("SUSPICIOUS"

print("[INFO] df_events ready:", df_events.shape)
print("[INFO] Audit window (UTC):", start_utc, "→", now_utc)
display(df_events.head(10))
```

```
[INFO] df_events ready: (1605, 13)
[INFO] Audit window (UTC): 2026-03-03 20:45:28.611861+00:00 → 2026-03-04 20:
45:28.611861+00:00
```

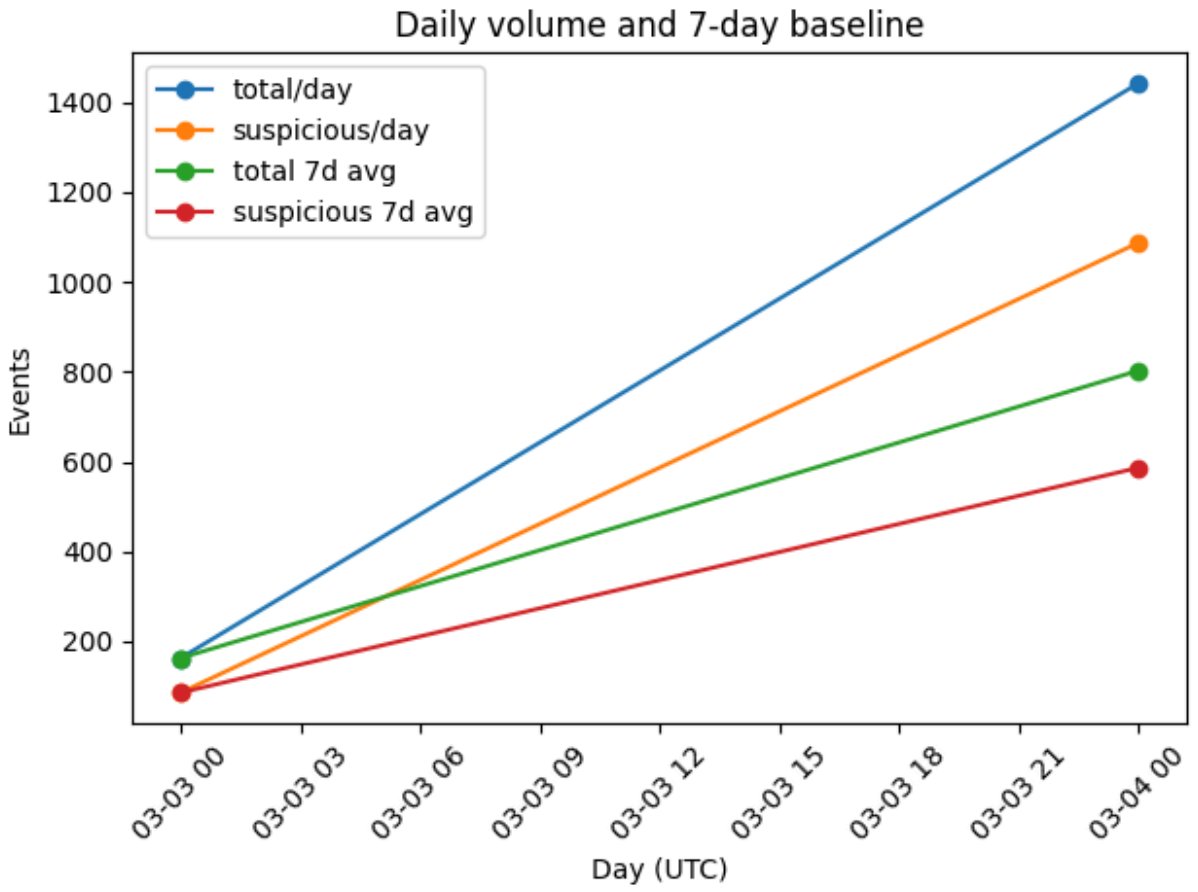
| | ts_utc | source | ip | method | |
|---|---|---|---|---|---|
| **174758** | 2026-03-03 20:50:37+00:00 | astropema_ssl_access | 127.0.0.1 | POST | |
| **174759** | 2026-03-03 20:50:37+00:00 | astropema_ssl_access | 127.0.0.1 | GET | |
| **174760** | 2026-03-03 20:51:01+00:00 | astromap-ssl-access | 127.0.0.1 | GET | |
| **174761** | 2026-03-03 20:51:03+00:00 | astropema_ssl_access | 127.0.0.1 | GET | |
| **174762** | 2026-03-03 20:53:38+00:00 | astromap-access | 175.6.217.4 | GET | |
| **174763** | 2026-03-03 20:53:41+00:00 | astromap-ssl-access | 127.0.0.1 | GET | |
| **174764** | 2026-03-03 20:56:29+00:00 | astromap-ssl-access | 127.0.0.1 | GET | |
| **174765** | 2026-03-03 21:11:07+00:00 | astromap-access | 43.157.181.189 | GET | |
| **174766** | 2026-03-03 21:11:09+00:00 | astromap-ssl-access | 127.0.0.1 | GET | |
| **174767** | 2026-03-03 21:11:40+00:00 | astromap-ssl-access | 127.0.0.1 | GET | dns=sgQBAAABAAA |

## Cell 4 — Daily volume + 7-day baseline

In [5]:

```
# ==========================
# Cell 4 — Daily volume + 7-day baseline
# ==========================

df_daily = (
    df_events
    .groupby("day_utc", as_index=False)
    .agg(
        total_day=("day_utc", "size"),
        suspicious_day=("verdict", lambda x: (x == "SUSPICIOUS").sum()),
```

```
    )
    .sort_values("day_utc")
)

df_daily["total_7d_avg"] = df_daily["total_day"].rolling(7, min_periods=1).m
df_daily["suspicious_7d_avg"] = df_daily["suspicious_day"].rolling(7, min_pe

plt.figure()
plt.plot(df_daily["day_utc"], df_daily["total_day"], marker="o", label="tota
plt.plot(df_daily["day_utc"], df_daily["suspicious_day"], marker="o", label=
plt.plot(df_daily["day_utc"], df_daily["total_7d_avg"], marker="o", label="t
plt.plot(df_daily["day_utc"], df_daily["suspicious_7d_avg"], marker="o", lab
plt.title("Daily volume and 7-day baseline")
plt.xlabel("Day (UTC)")
plt.ylabel("Events")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

display(df_daily)
```



| | day_utc | total_day | suspicious_day | total_7d_avg | suspicious_7d_avg |
|---|---|---|---|---|---|
| **0** | 2026-03-03 00:00:00+00:00 | 163 | 86 | 163.0 | 86.0 |
| **1** | 2026-03-04 00:00:00+00:00 | 1442 | 1087 | 802.5 | 586.5 |

## Cell 5 — Top suspicious spike days vs baseline

In [6]:
```
# ========================
# Cell 5 — Top suspicious spike days vs baseline
# ========================

df_spikes = df_daily.copy()
df_spikes["susp_over_baseline"] = np.where(
    df_spikes["suspicious_7d_avg"] > 0,
    df_spikes["suspicious_day"] / df_spikes["suspicious_7d_avg"],
    np.nan
)
df_spikes["total_over_baseline"] = np.where(
    df_spikes["total_7d_avg"] > 0,
    df_spikes["total_day"] / df_spikes["total_7d_avg"],
    np.nan
)

df_spikes = df_spikes.sort_values("susp_over_baseline", ascending=False).res

print("Top suspicious spike days (vs 7-day rolling baseline):")
display(df_spikes[[
    "day_utc","suspicious_day","suspicious_7d_avg","susp_over_baseline",
    "total_day","total_7d_avg","total_over_baseline"
]].head(15))
```

```
Top suspicious spike days (vs 7-day rolling baseline):
```

| | day_utc | suspicious_day | suspicious_7d_avg | susp_over_baseline | total_day | total |
|---|---|---|---|---|---|---|
| **0** | 2026-03-04 00:00:00+00:00 | 1087 | 586.5 | 1.853367 | 1442 | |
| **1** | 2026-03-03 00:00:00+00:00 | 86 | 86.0 | 1.000000 | 163 | |

## Cell 6 — Per-IP aggregation (scanner-like classification)

In [7]:
```
# ========================
# Cell 6 — Per-IP aggregation (scanner-like classification)
# ========================
```

```
df_ip = (
    df_events
    .groupby("ip_norm", as_index=False)
    .agg(
        n_total=("ip_norm", "size"),
        n_susp=("verdict", lambda x: (x == "SUSPICIOUS").sum()),
        n_block_expected=("expected_action", lambda x: (x == "BLOCK").sum())
        first_seen_utc=("ts_utc", "min"),
        last_seen_utc=("ts_utc", "max"),
    )
)

df_ip["active_hours"] = (df_ip["last_seen_utc"] - df_ip["first_seen_utc"]).c
df_ip["susp_pct"] = df_ip["n_susp"] / df_ip["n_total"]
df_ip["is_home_ip"] = df_ip["ip_norm"].eq(HOME_IP)
df_ip["scanner_like"] = (df_ip["susp_pct"] >= SCANNER_SUSP_PCT_THRESHOLD) &

df_ip = df_ip.sort_values(["scanner_like", "n_susp"], ascending=[False, Fals

df_ip_view = df_ip[df_ip["n_total"] >= MIN_EVENTS_PER_IP].copy()
print(f"IPs analyzed (n_total >= {MIN_EVENTS_PER_IP}):", int(df_ip_view.shap
display(df_ip_view)
```

```
IPs analyzed (n_total >= 50): 5
```

| | ip_norm | n_total | n_susp | n_block_expected | first_seen_utc | last_seen_utc | ac |
|---|---|---|---|---|---|---|---|
| **0** | 172.190.142.176 | 338 | 338 | 338 | 2026-03-04 02:44:31+00:00 | 2026-03-04 11:44:31+00:00 | |
| **1** | 4.204.200.32 | 169 | 169 | 169 | 2026-03-04 10:54:42+00:00 | 2026-03-04 10:55:15+00:00 | |
| **2** | 52.169.206.229 | 159 | 159 | 159 | 2026-03-04 03:46:20+00:00 | 2026-03-04 03:47:57+00:00 | |
| **3** | 89.248.168.239 | 55 | 55 | 55 | 2026-03-04 05:08:04+00:00 | 2026-03-04 05:08:22+00:00 | |
| **22** | 127.0.0.1 | 708 | 376 | 376 | 2026-03-03 20:50:37+00:00 | 2026-03-04 12:41:23+00:00 | |

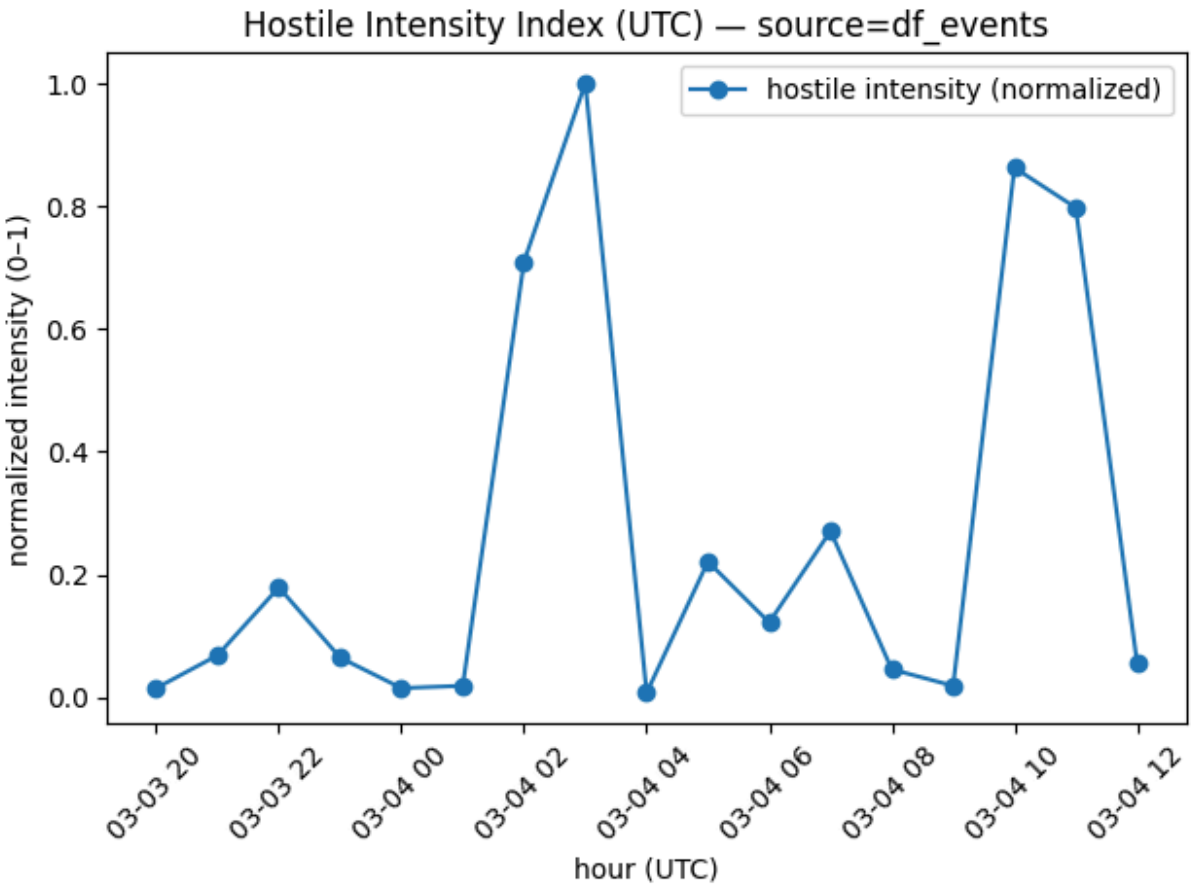
# Cell 7 — Hostile Intensity Index (hourly, normalized)

In [8]:

```
# ========================
# Cell 7 — Hostile Intensity Index (hourly, normalized)
# ========================

df_hourly = (
    df_events
    .groupby("hour_utc", as_index=False)
    .agg(
        total_events=("hour_utc", "size"),
```

```
        suspicious_events=("verdict", lambda x: (x == "SUSPICIOUS").sum()),
    )
    .sort_values("hour_utc")
)

peak = int(df_hourly["suspicious_events"].max()) if not df_hourly.empty else
df_hourly["hostile_intensity_norm"] = (df_hourly["suspicious_events"] / peak

print("[INFO] Peak suspicious/hour:", peak)
print("[INFO] Hours covered:", int(df_hourly.shape[0]))

plt.figure()
plt.plot(df_hourly["hour_utc"], df_hourly["hostile_intensity_norm"], marker=
plt.title("Hostile Intensity Index (UTC) — source=df_events")
plt.xlabel("hour (UTC)")
plt.ylabel("normalized intensity (0–1)")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

display(df_hourly.tail(72))
```

```
[INFO] Peak suspicious/hour: 262
[INFO] Hours covered: 17
```



| | hour_utc | total_events | suspicious_events | hostile_intensity_norm |
|---|---|---|---|---|
| **0** | 2026-03-03 20:00:00+00:00 | 7 | 4 | 0.015267 |
| **1** | 2026-03-03 21:00:00+00:00 | 38 | 18 | 0.068702 |
| **2** | 2026-03-03 22:00:00+00:00 | 84 | 47 | 0.179389 |
| **3** | 2026-03-03 23:00:00+00:00 | 34 | 17 | 0.064885 |
| **4** | 2026-03-04 00:00:00+00:00 | 18 | 4 | 0.015267 |
| **5** | 2026-03-04 01:00:00+00:00 | 17 | 5 | 0.019084 |
| **6** | 2026-03-04 02:00:00+00:00 | 212 | 186 | 0.709924 |
| **7** | 2026-03-04 03:00:00+00:00 | 279 | 262 | 1.000000 |
| **8** | 2026-03-04 04:00:00+00:00 | 19 | 2 | 0.007634 |
| **9** | 2026-03-04 05:00:00+00:00 | 75 | 58 | 0.221374 |
| **10** | 2026-03-04 06:00:00+00:00 | 93 | 32 | 0.122137 |
| **11** | 2026-03-04 07:00:00+00:00 | 95 | 71 | 0.270992 |
| **12** | 2026-03-04 08:00:00+00:00 | 43 | 12 | 0.045802 |
| **13** | 2026-03-04 09:00:00+00:00 | 46 | 5 | 0.019084 |
| **14** | 2026-03-04 10:00:00+00:00 | 294 | 226 | 0.862595 |
| **15** | 2026-03-04 11:00:00+00:00 | 216 | 209 | 0.797710 |
| **16** | 2026-03-04 12:00:00+00:00 | 35 | 15 | 0.057252 |

## Cell 8 — Impact estimate if scanner-like IPs were blocked (analysis only)

Estimates how much suspicious volume is attributable to scanner-like IPs in the current window.

In [9]:
```
# ========================
# Cell 8 — Impact estimate if scanner-like IPs were blocked (analysis only)
# ========================

df_ip_scanners = df_ip_view[df_ip_view["scanner_like"]].copy()
SCANNER_IPS = df_ip_scanners["ip_norm"].astype(str).unique().tolist()

scanner_events = df_events[df_events["ip_norm"].isin(SCANNER_IPS)].copy()

summary = {
    "total_events": int(df_events.shape[0]),
    "total_suspicious": int((df_events["verdict"] == "SUSPICIOUS").sum()),
    "scanner_events": int(scanner_events.shape[0]),
```

```
    "scanner_suspicious": int((scanner_events["verdict"] == "SUSPICIOUS").su
}
summary["scanner_event_share_pct"] = 100.0 * summary["scanner_events"] / max
summary["scanner_suspicious_share_pct"] = 100.0 * summary["scanner_suspiciou
summary["scanner_ip_count"] = int(len(SCANNER_IPS))

print(summary)

bins = [-np.inf, 0.5, 6, 24, 72, np.inf]  # hours
labels = ["<30 min", "30 min – 6 h", "6 – 24 h", "1 – 3 days", ">3 days"]
df_ip_scanners["duration_class"] = pd.cut(df_ip_scanners["active_hours"], bi

df_dur = (
    df_ip_scanners
    .groupby("duration_class", as_index=False)
    .agg(
        scanner_ips=("ip_norm", "nunique"),
        total_events=("n_total", "sum"),
        total_suspicious=("n_susp", "sum"),
    )
)

display(df_dur)

impact = {
    "current_suspicious_events": summary["total_suspicious"],
    "suspicious_removed_if_blocked": summary["scanner_suspicious"],
    "remaining_suspicious_events": summary["total_suspicious"] - summary["sc
    "suspicious_reduction_pct": 100.0 * summary["scanner_suspicious"] / max(
    "current_total_events": summary["total_events"],
    "events_removed_if_blocked": summary["scanner_events"],
    "remaining_events": summary["total_events"] - summary["scanner_events"],
}
print(impact)
```

```
{'total_events': 1605, 'total_suspicious': 1173, 'scanner_events': 721, 'sca
nner_suspicious': 721, 'scanner_event_share_pct': 44.92211838006231, 'scanne
r_suspicious_share_pct': 61.46632566069906, 'scanner_ip_count': 4}
```

```
/tmp/ipykernel_997185/4200406672.py:27: FutureWarning: The default of observ
ed=False is deprecated and will be changed to True in a future version of pa
ndas. Pass observed=False to retain current behavior or observed=True to ado
pt the future default and silence this warning.
  df_ip_scanners
```

| | duration_class | scanner_ips | total_events | total_suspicious |
|---|---|---|---|---|
| **0** | <30 min | 3 | 383 | 383 |
| **1** | 30 min – 6 h | 0 | 0 | 0 |
| **2** | 6 – 24 h | 1 | 338 | 338 |
| **3** | 1 – 3 days | 0 | 0 | 0 |
| **4** | >3 days | 0 | 0 | 0 |

```
{'current_suspicious_events': 1173, 'suspicious_removed_if_blocked': 721, 'r
emaining_suspicious_events': 452, 'suspicious_reduction_pct': 61.46632566069
906, 'current_total_events': 1605, 'events_removed_if_blocked': 721, 'remain
ing_events': 884}
```

# Cell 9 — Enforcement recommendations (commands only; NOT executed)

This notebook never enforces. It prints a shell snippet for a human admin to run.

In [10]:
```python
# ========================
# Cell 9 — Enforcement recommendations (commands only; NOT executed)
# ========================

block_ips = (
    df_ip_scanners[df_ip_scanners["n_total"] >= MIN_EVENTS_PER_IP]["ip_norm"
    .astype(str)
    .unique()
    .tolist()
)
block_ips = [ip for ip in block_ips if ip != HOME_IP]

print("ADMIN ACTION (manual, from shell) — NOT executed by this notebook.")
print(f"TTL policy: {TTL_SECONDS}s (recommended TTL blocking; not permanent)
print(f"Safety: HOME_IP is excluded: {HOME_IP}\n")

print("If you approve enforcement, run the following commands on the host:\r
print("sudo ipset list bad_ips >/dev/null 2>&1 || { echo 'bad_ips ipset not
for ip in block_ips:
    print(f"sudo ipset add -exist bad_ips {ip} timeout {TTL_SECONDS}")

print("\nCount:", len(block_ips))
if len(block_ips) > 0:
    print("First 20:", block_ips[:20])
```

```
ADMIN ACTION (manual, from shell) — NOT executed by this notebook.
TTL policy: 604800s (recommended TTL blocking; not permanent).
Safety: HOME_IP is excluded: 66.241.78.7

If you approve enforcement, run the following commands on the host:

sudo ipset list bad_ips >/dev/null 2>&1 || { echo 'bad_ips ipset not found';
exit 1; }
sudo ipset add -exist bad_ips 172.190.142.176 timeout 604800
sudo ipset add -exist bad_ips 4.204.200.32 timeout 604800
sudo ipset add -exist bad_ips 52.169.206.229 timeout 604800
sudo ipset add -exist bad_ips 89.248.168.239 timeout 604800

Count: 4
First 20: ['172.190.142.176', '4.204.200.32', '52.169.206.229', '89.248.168.
239']
```

In [11]:
```python
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt

def _col(df, *names):
    """Return first matching column name from names, else None."""
    cols = set(df.columns)
    for n in names:
        if n in cols:
            return n
    return None

def _ensure_ts_utc(df, ts_col="ts_utc"):
    if ts_col not in df.columns:
        raise RuntimeError(f"Missing timestamp col: {ts_col}. Available: {li
    d = df.copy()
    d[ts_col] = pd.to_datetime(d[ts_col], utc=True, errors="coerce")
    d = d.dropna(subset=[ts_col])
    return d

def _normalize_path(s):
    s = str(s) if s is not None else ""
    s = s.split("?", 1)[0]                # strip query
    s = s.split("#", 1)[0]                # strip fragment
    s = s.replace("//", "/")
    if not s.startswith("/"):
        s = "/" + s
    return s

def _status_bucket(code):
    try:
        c = int(code)
    except Exception:
        return "other"
    if 100 <= c < 200: return "1xx"
    if 200 <= c < 300: return "2xx"
    if 300 <= c < 400: return "3xx"
    if 400 <= c < 500: return "4xx"
    if 500 <= c < 600: return "5xx"
    return "other"
```

In [12]:

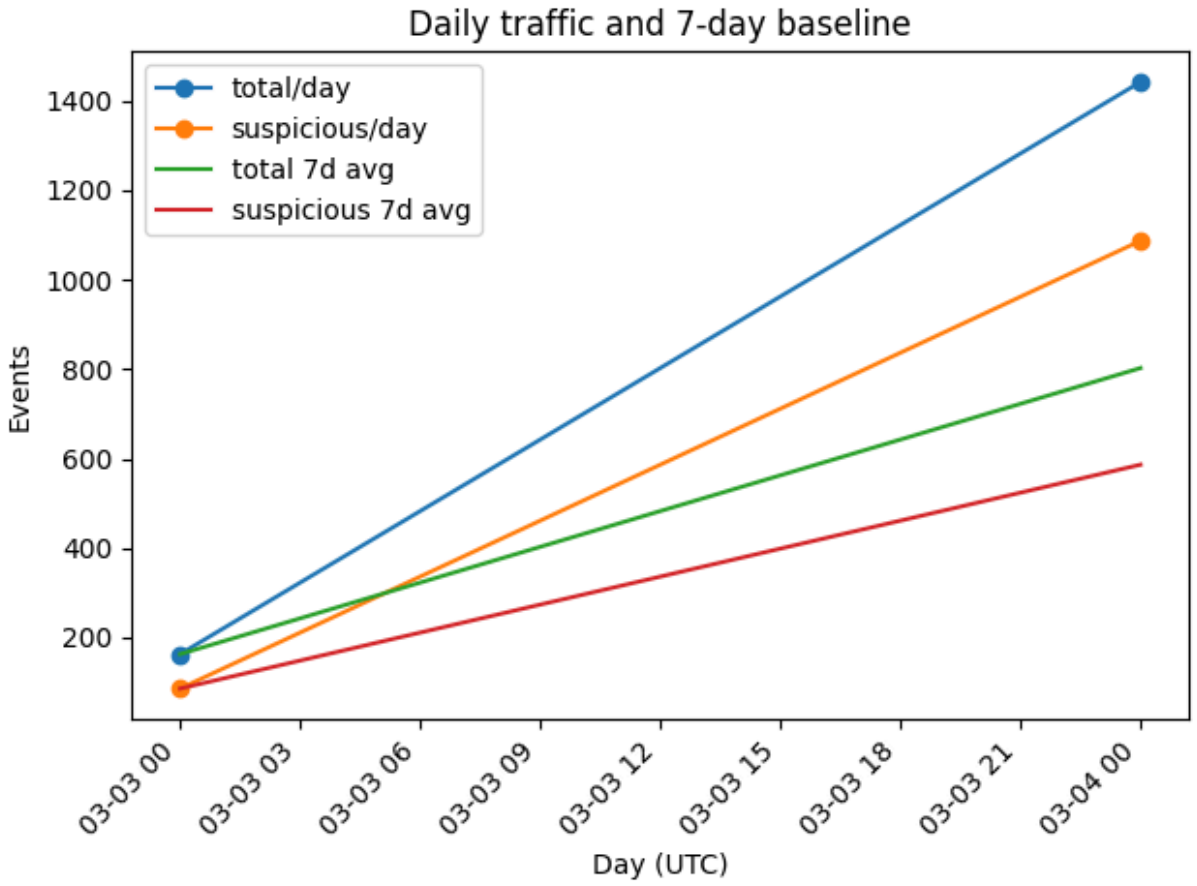
```
# Preconditions
if "df_events" not in globals():
    raise RuntimeError("Notebook #3: expected df_events to exist before dash

ts_col = _col(df_events, "ts_utc", "timestamp_utc", "ts", "time_utc")
path_col = _col(df_events, "path", "uri", "request_path", "url_path")
status_col = _col(df_events, "status", "status_code", "http_status")
ip_col = _col(df_events, "ip", "remote_ip", "client_ip")
verdict_col = _col(df_events, "verdict", "decision", "label")

df_dash = _ensure_ts_utc(df_events, ts_col=ts_col)

if path_col:
    df_dash["path_norm"] = df_dash[path_col].map(_normalize_path)
else:
    df_dash["path_norm"] = "/"
```

```
if status_col:
    df_dash["status_bucket"] = df_dash[status_col].map(_status_bucket)
else:
    df_dash["status_bucket"] = "other"

if verdict_col:
    df_dash["is_susp"] = (df_dash[verdict_col].astype(str).str.upper() == "S
else:
    # fallback: no verdict available
    df_dash["is_susp"] = False

# Time buckets
df_dash["day_utc"] = df_dash[ts_col].dt.floor("D")
df_dash["hour_utc"] = df_dash[ts_col].dt.floor("h")
df_dash["dow"] = df_dash[ts_col].dt.day_name()
df_dash["hour_of_day"] = df_dash[ts_col].dt.hour

# Basic counts
print("Dashboard rows:", len(df_dash))
print("Window UTC:", df_dash[ts_col].min(), "->", df_dash[ts_col].max())
```

```
Dashboard rows: 1605
Window UTC: 2026-03-03 20:50:37+00:00 -> 2026-03-04 12:41:29+00:00
```

In [13]:

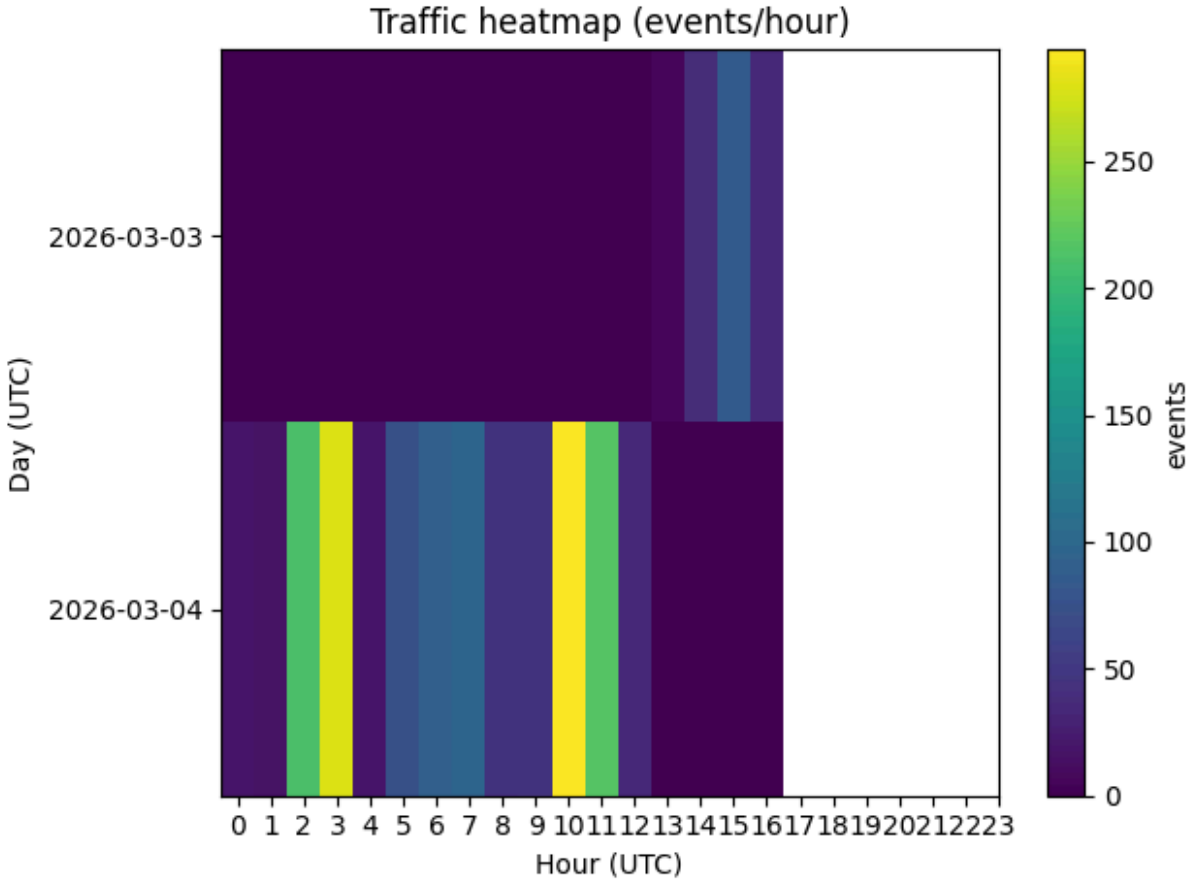
```
g = df_dash.groupby("day_utc", as_index=False).agg(
    total_events=("day_utc", "size"),
    suspicious=("is_susp", "sum")
).sort_values("day_utc")

g["total_roll7"] = g["total_events"].rolling(7, min_periods=1).mean()
g["susp_roll7"] = g["suspicious"].rolling(7, min_periods=1).mean()

plt.figure()
plt.plot(g["day_utc"], g["total_events"], marker="o", label="total/day")
plt.plot(g["day_utc"], g["suspicious"], marker="o", label="suspicious/day")
plt.plot(g["day_utc"], g["total_roll7"], label="total 7d avg")
plt.plot(g["day_utc"], g["susp_roll7"], label="suspicious 7d avg")
plt.xticks(rotation=45, ha="right")
plt.xlabel("Day (UTC)")
plt.ylabel("Events")
plt.title("Daily traffic and 7-day baseline")
plt.legend()
plt.tight_layout()
plt.show()
```

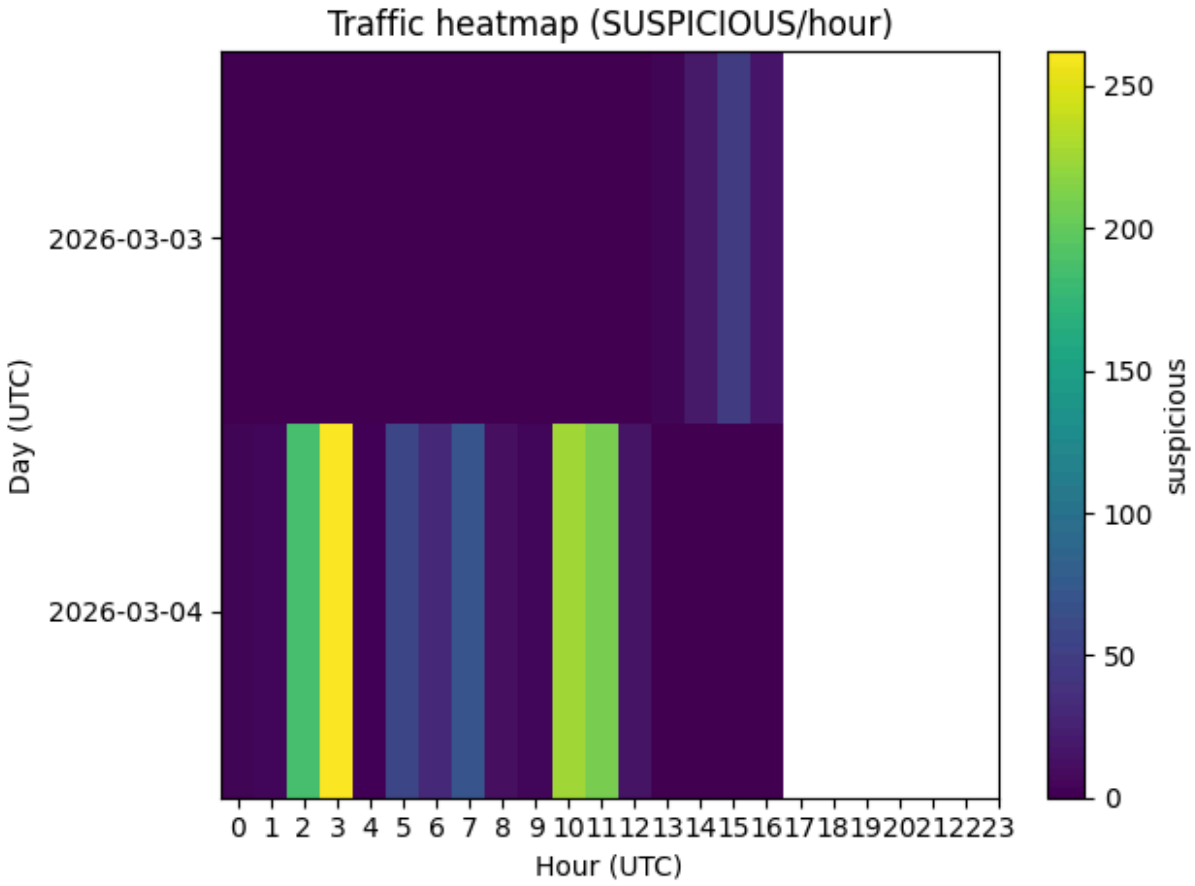



In [14]:

```
# Heatmap: day x hour, values = events (and suspicious events as separate)
pivot_total = (
    df_dash.pivot_table(index="day_utc", columns="hour_of_day", values="path
    .sort_index()
)

plt.figure()
plt.imshow(pivot_total.values, aspect="auto")
plt.yticks(range(len(pivot_total.index)), [d.strftime("%Y-%m-%d") for d in p
plt.xticks(range(24), list(range(24)))
plt.xlabel("Hour (UTC)")
plt.ylabel("Day (UTC)")
plt.title("Traffic heatmap (events/hour)")
plt.colorbar(label="events")
plt.tight_layout()
plt.show()

# Suspicious-only heatmap (if you have verdicts)
pivot_susp = (
    df_dash[df_dash["is_susp"]]
    .pivot_table(index="day_utc", columns="hour_of_day", values="path_norm",
    .sort_index()
)

if len(pivot_susp) > 0:
    plt.figure()
    plt.imshow(pivot_susp.values, aspect="auto")
    plt.yticks(range(len(pivot_susp.index)), [d.strftime("%Y-%m-%d") for d i
```

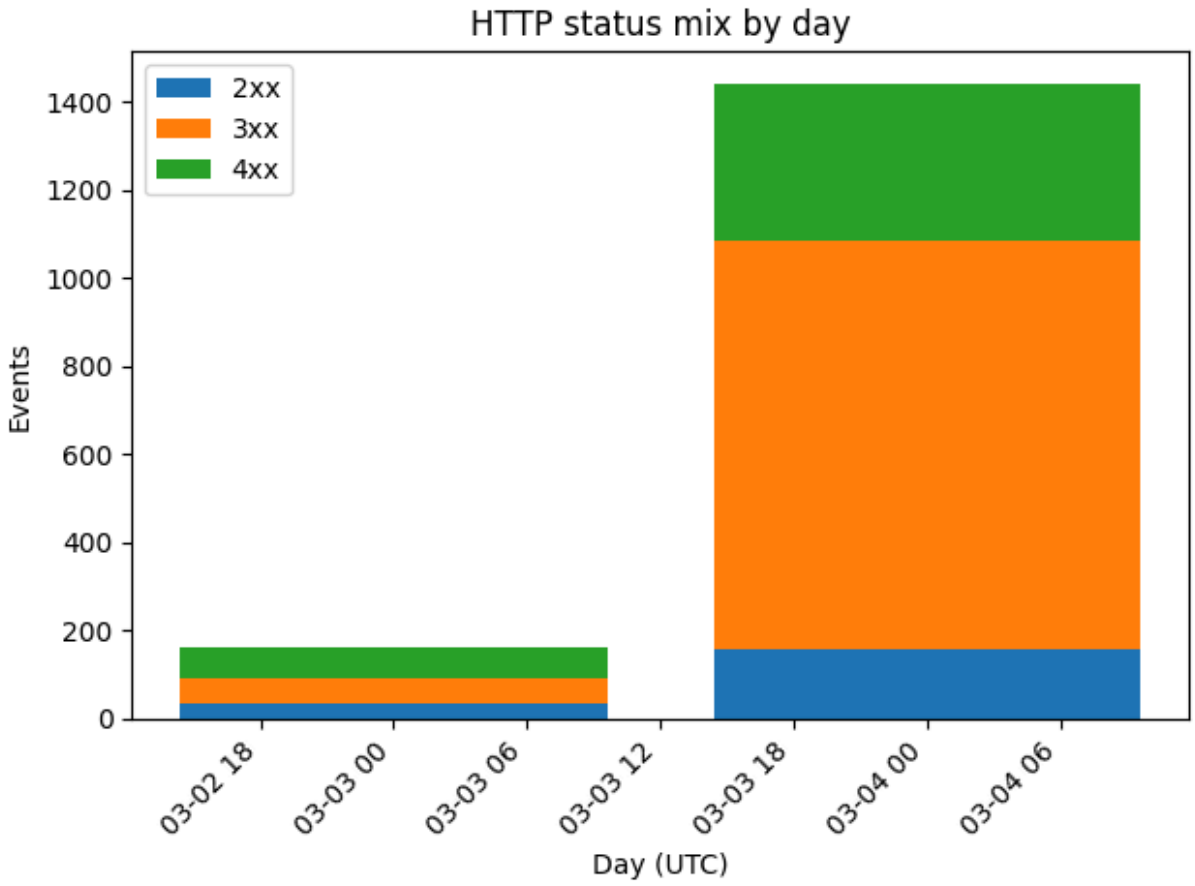
```
    plt.xticks(range(24), list(range(24)))
    plt.xlabel("Hour (UTC)")
    plt.ylabel("Day (UTC)")
    plt.title("Traffic heatmap (SUSPICIOUS/hour)")
    plt.colorbar(label="suspicious")
    plt.tight_layout()
    plt.show()
```





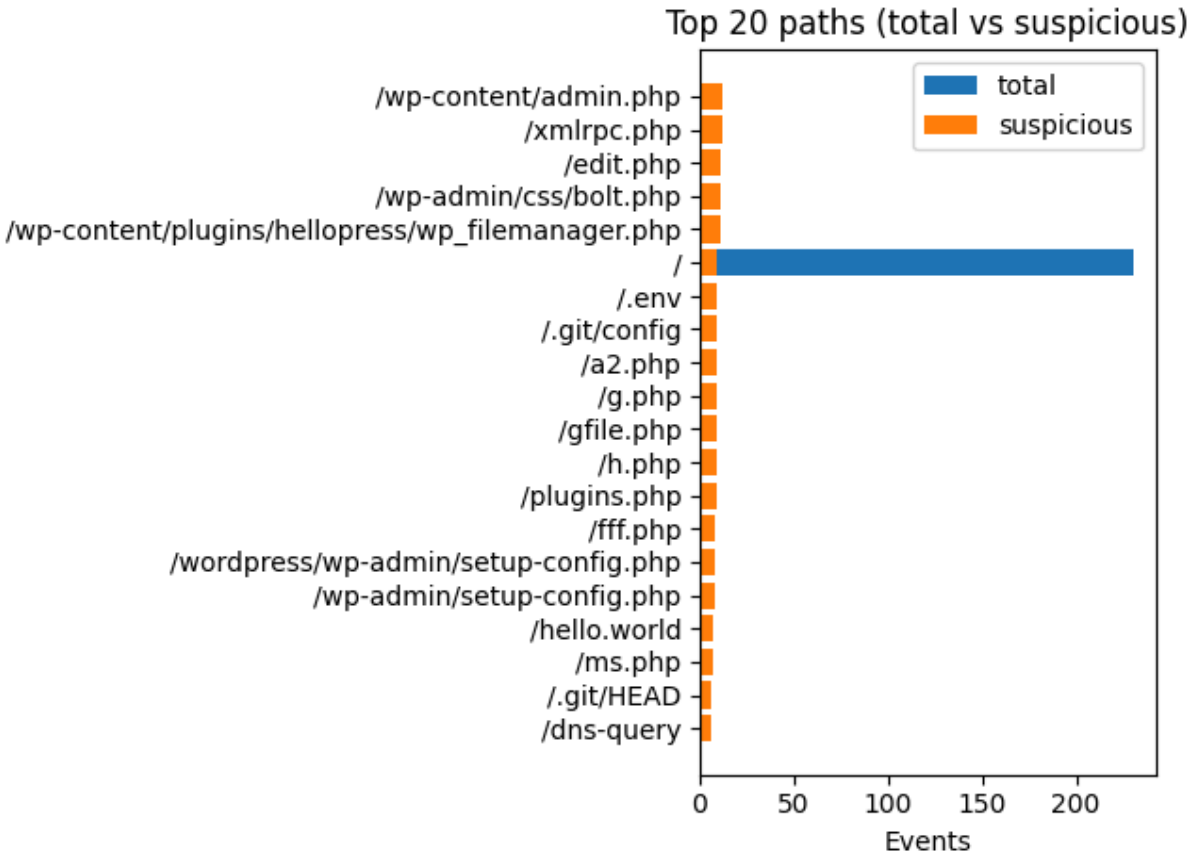

In [15]:

```
s = df_dash.groupby(["day_utc", "status_bucket"], as_index=False).size()
# pivot for stacked area/stacked bar feel
p = s.pivot(index="day_utc", columns="status_bucket", values="size").fillna(

plt.figure()
bottom = np.zeros(len(p))
for col in p.columns:
    plt.bar(p.index, p[col].values, bottom=bottom, label=col)
    bottom += p[col].values

plt.xticks(rotation=45, ha="right")
plt.xlabel("Day (UTC)")
plt.ylabel("Events")
plt.title("HTTP status mix by day")
plt.legend()
plt.tight_layout()
plt.show()
```




In [16]:

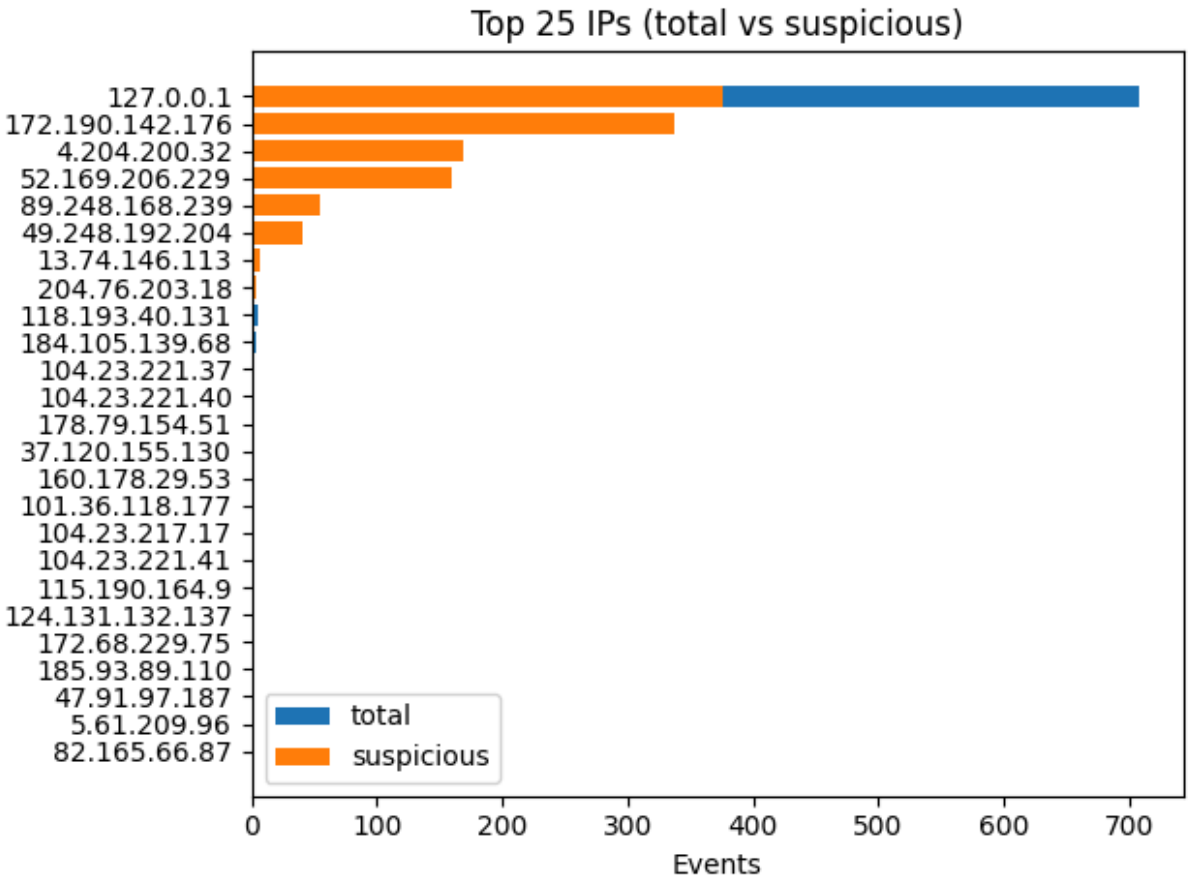
```
top_n = 20
by_path = df_dash.groupby("path_norm", as_index=False).agg(
    total=("path_norm", "size"),
    suspicious=("is_susp", "sum")
)
by_path["susp_pct"] = np.where(by_path["total"] > 0, by_path["suspicious"] /
by_path = by_path.sort_values(["suspicious", "total"], ascending=False).heac

# Horizontal bar “AWStats-like”
plt.figure()
plt.barh(by_path["path_norm"][::-1], by_path["total"][::-1], label="total")
plt.barh(by_path["path_norm"][::-1], by_path["suspicious"][::-1], label="sus
plt.xlabel("Events")
plt.title(f"Top {top_n} paths (total vs suspicious)")
plt.legend()
plt.tight_layout()
plt.show()

display(by_path.reset_index(drop=True))
```

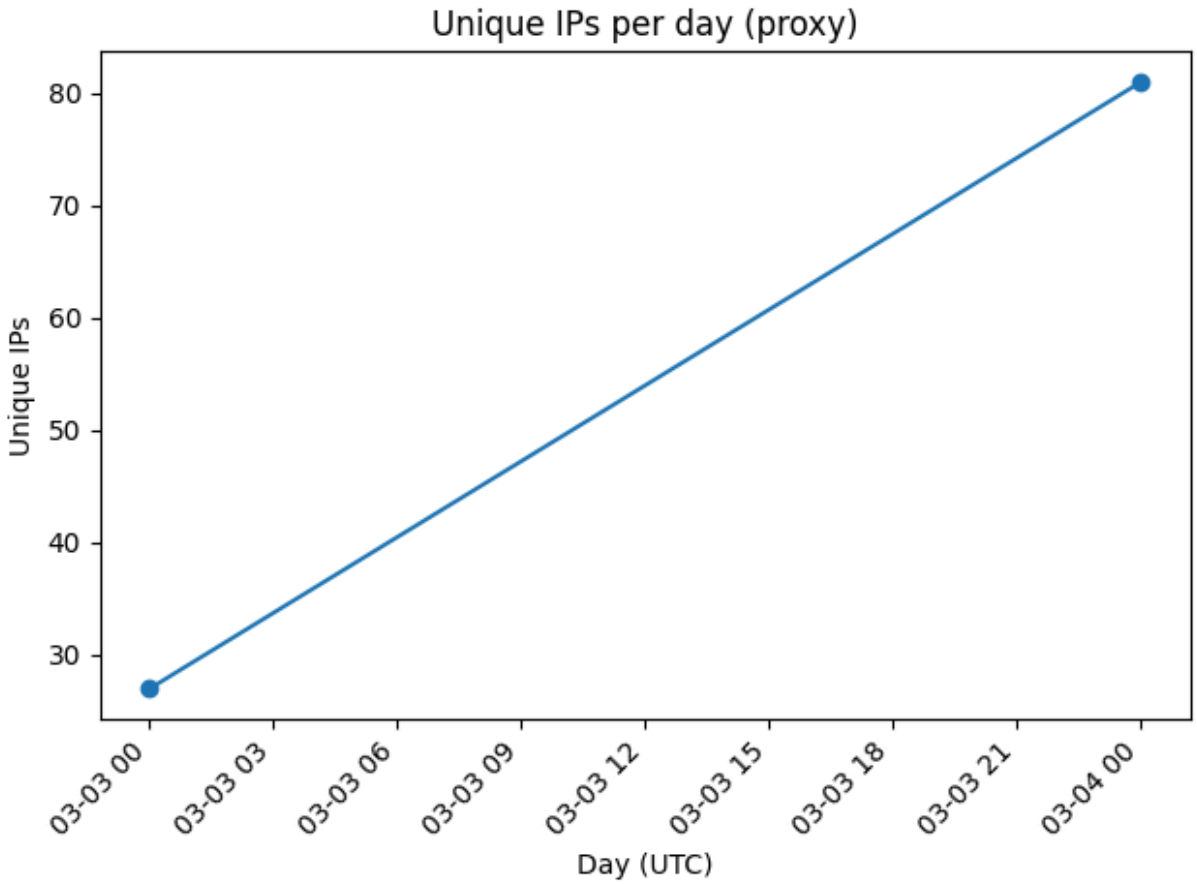

Top 20 paths (total vs suspicious)
total
suspicious
/wp-content/admin.php
/xmlrpc.php
/edit.php
/wp-admin/css/bolt.php
/wp-content/plugins/hellopress/wp_filemanager.php
/
/.env
/.git/config
/a2.php
/g.php
/gfile.php
/h.php
/plugins.php
/fff.php
/wordpress/wp-admin/setup-config.php
/wp-admin/setup-config.php
/hello.world
/ms.php
/.git/HEAD
/dns-query
0
50
100
150
200
Events

| | path_norm | total | suspicious | susp_pct |
|---|---|---|---|---|
| **0** | /wp-content/admin.php | 12 | 12 | 1.000000 |
| **1** | /xmlrpc.php | 12 | 12 | 1.000000 |
| **2** | /edit.php | 11 | 11 | 1.000000 |
| **3** | /wp-admin/css/bolt.php | 11 | 11 | 1.000000 |
| **4** | /wp-content/plugins/hellopress/wp_filemanager.php | 11 | 11 | 1.000000 |
| **5** | / | 231 | 9 | 0.038961 |
| **6** | /.env | 9 | 9 | 1.000000 |
| **7** | /.git/config | 9 | 9 | 1.000000 |
| **8** | /a2.php | 9 | 9 | 1.000000 |
| **9** | /g.php | 9 | 9 | 1.000000 |
| **10** | /gfile.php | 9 | 9 | 1.000000 |
| **11** | /h.php | 9 | 9 | 1.000000 |
| **12** | /plugins.php | 9 | 9 | 1.000000 |
| **13** | /fff.php | 8 | 8 | 1.000000 |
| **14** | /wordpress/wp-admin/setup-config.php | 8 | 8 | 1.000000 |
| **15** | /wp-admin/setup-config.php | 8 | 8 | 1.000000 |
| **16** | /hello.world | 7 | 7 | 1.000000 |
| **17** | /ms.php | 7 | 7 | 1.000000 |
| **18** | /.git/HEAD | 6 | 6 | 1.000000 |
| **19** | /dns-query | 6 | 6 | 1.000000 |

In [17]:
```
if ip_col:
    df_dash["ip_norm"] = df_dash[ip_col].astype(str).map(lambda x: x.split("
else:
    df_dash["ip_norm"] = "unknown"

top_n = 25
by_ip = df_dash.groupby("ip_norm", as_index=False).agg(
    total=("ip_norm", "size"),
    suspicious=("is_susp", "sum"),
    first_seen=(ts_col, "min"),
    last_seen=(ts_col, "max")
).sort_values(["suspicious", "total"], ascending=False).head(top_n)

plt.figure()
plt.barh(by_ip["ip_norm"][::-1], by_ip["total"][::-1], label="total")
plt.barh(by_ip["ip_norm"][::-1], by_ip["suspicious"][::-1], label="suspiciou
```

```
plt.xlabel("Events")
plt.title(f"Top {top_n} IPs (total vs suspicious)")
plt.legend()
plt.tight_layout()
plt.show()

# “Unique visitors” proxy by day = count distinct IPs/day
uniq = df_dash.groupby("day_utc", as_index=False).agg(unique_ips=("ip_norm",
plt.figure()
plt.plot(uniq["day_utc"], uniq["unique_ips"], marker="o")
plt.xticks(rotation=45, ha="right")
plt.xlabel("Day (UTC)")
plt.ylabel("Unique IPs")
plt.title("Unique IPs per day (proxy)")
plt.tight_layout()
plt.show()

display(by_ip.reset_index(drop=True))
```




Unique IPs per day (proxy)
Unique IPs
80
70
60
50
40
30
03-03 00
03-03 03
03-03 06
03-03 09
03-03 12
03-03 15
03-03 18
03-03 21
03-04 00
Day (UTC)

| | ip_norm | total | suspicious | first_seen | last_seen |
|---|---|---|---|---|---|
| **0** | 127.0.0.1 | 708 | 376 | 2026-03-03 20:50:37+00:00 | 2026-03-04 12:41:23+00:00 |
| **1** | 172.190.142.176 | 338 | 338 | 2026-03-04 02:44:31+00:00 | 2026-03-04 11:44:31+00:00 |
| **2** | 4.204.200.32 | 169 | 169 | 2026-03-04 10:54:42+00:00 | 2026-03-04 10:55:15+00:00 |
| **3** | 52.169.206.229 | 159 | 159 | 2026-03-04 03:46:20+00:00 | 2026-03-04 03:47:57+00:00 |
| **4** | 89.248.168.239 | 55 | 55 | 2026-03-04 05:08:04+00:00 | 2026-03-04 05:08:22+00:00 |
| **5** | 49.248.192.204 | 41 | 41 | 2026-03-03 22:29:33+00:00 | 2026-03-04 10:04:21+00:00 |
| **6** | 13.74.146.113 | 7 | 7 | 2026-03-03 22:28:50+00:00 | 2026-03-03 22:28:52+00:00 |
| **7** | 204.76.203.18 | 4 | 3 | 2026-03-04 04:32:14+00:00 | 2026-03-04 12:24:44+00:00 |
| **8** | 118.193.40.131 | 6 | 2 | 2026-03-04 09:37:20+00:00 | 2026-03-04 09:37:24+00:00 |
| **9** | 184.105.139.68 | 4 | 2 | 2026-03-04 01:18:21+00:00 | 2026-03-04 01:21:57+00:00 |
| **10** | 104.23.221.37 | 2 | 2 | 2026-03-03 23:35:33+00:00 | 2026-03-03 23:37:48+00:00 |
| **11** | 104.23.221.40 | 2 | 2 | 2026-03-03 23:36:58+00:00 | 2026-03-03 23:37:58+00:00 |
| **12** | 178.79.154.51 | 2 | 2 | 2026-03-04 08:50:43+00:00 | 2026-03-04 08:50:43+00:00 |
| **13** | 37.120.155.130 | 2 | 2 | 2026-03-04 00:45:36+00:00 | 2026-03-04 00:45:36+00:00 |
| **14** | 160.178.29.53 | 2 | 1 | 2026-03-04 01:03:57+00:00 | 2026-03-04 01:03:58+00:00 |
| **15** | 101.36.118.177 | 1 | 1 | 2026-03-04 06:18:32+00:00 | 2026-03-04 06:18:32+00:00 |
| **16** | 104.23.217.17 | 1 | 1 | 2026-03-04 08:14:49+00:00 | 2026-03-04 08:14:49+00:00 |
| **17** | 104.23.221.41 | 1 | 1 | 2026-03-04 08:18:22+00:00 | 2026-03-04 08:18:22+00:00 |

| | ip_norm | total | suspicious | first_seen | last_seen |
|---|---|---|---|---|---|
| **18** | 115.190.164.9 | 1 | 1 | 2026-03-03 23:40:38+00:00 | 2026-03-03 23:40:38+00:00 |
| **19** | 124.131.132.137 | 1 | 1 | 2026-03-04 12:05:43+00:00 | 2026-03-04 12:05:43+00:00 |
| **20** | 172.68.229.75 | 1 | 1 | 2026-03-04 08:17:41+00:00 | 2026-03-04 08:17:41+00:00 |
| **21** | 185.93.89.110 | 1 | 1 | 2026-03-04 00:40:09+00:00 | 2026-03-04 00:40:09+00:00 |
| **22** | 47.91.97.187 | 1 | 1 | 2026-03-03 21:29:02+00:00 | 2026-03-03 21:29:02+00:00 |
| **23** | 5.61.209.96 | 1 | 1 | 2026-03-04 02:18:41+00:00 | 2026-03-04 02:18:41+00:00 |
| **24** | 82.165.66.87 | 1 | 1 | 2026-03-04 10:20:40+00:00 | 2026-03-04 10:20:40+00:00 |

In [18]:

```
# ==========================
# Cell G1 — Geolocation (RDAP via ipwhois) — OPTIONAL
# ==========================

import pandas as pd

# Controls
GEO_MAX_IPS = 200          # limit lookups for speed
GEO_TIMEOUT_SEC = 8        # best-effort; RDAP can be slow
GEO_USE_SCANNERS_ONLY = True  # True = only scanner-like IPs; False = top ta

try:
    from ipwhois import IPWhois
except Exception as e:
    raise RuntimeError(
        "ipwhois is not available in this kernel. "
        "Install it in the SAME env that runs Jupyter, e.g.: pip install ipw
    ) from e

# Preconditions
if "df_ip_view" not in globals():
    raise RuntimeError("Cell G1: expected df_ip_view from Cell 6.")
if "HOME_IP" not in globals():
    HOME_IP = "66.241.78.7"

# Pick IP set to geolocate
df_geo_src = df_ip_view.copy()
df_geo_src = df_geo_src[df_geo_src["ip_norm"].astype(str) != str(HOME_IP)].c

if GEO_USE_SCANNERS_ONLY and "scanner_like" in df_geo_src.columns:
    df_geo_src = df_geo_src[df_geo_src["scanner_like"]].copy()

df_geo_src = df_geo_src.sort_values(["n_susp", "n_total"], ascending=[False,
```

```
ips = df_geo_src["ip_norm"].astype(str).unique().tolist()
print(f"[GEO] Will attempt RDAP for {len(ips)} IPs (max={GEO_MAX_IPS}).")

# Cache across runs
if "GEO_CACHE" not in globals():
    GEO_CACHE = {}  # ip -> dict

def rdap_lookup(ip: str) -> dict:
    try:
        obj = IPWhois(ip)
        res = obj.lookup_rdap(depth=1)
        return {
            "ip_norm": ip,
            "asn": res.get("asn"),
            "asn_org": res.get("asn_description"),
            "asn_country": res.get("asn_country_code"),
            # best-effort extra fields
            "network_name": (res.get("network") or {}).get("name") if isinst
            "network_country": (res.get("network") or {}).get("country") if
        }
    except Exception as e:
        return {
            "ip_norm": ip,
            "asn": None,
            "asn_org": None,
            "asn_country": None,
            "network_name": None,
            "network_country": None,
            "geo_error": str(e),
        }

rows = []
new_lookups = 0
for ip in ips:
    if ip in GEO_CACHE:
        rows.append(GEO_CACHE[ip])
        continue
    rec = rdap_lookup(ip)
    GEO_CACHE[ip] = rec
    rows.append(rec)
    new_lookups += 1

df_geo = pd.DataFrame(rows)
print(f"[GEO] New lookups performed: {new_lookups}; cache size: {len(GEO_CAC
display(df_geo.head(30))
```

```
[GEO] Will attempt RDAP for 4 IPs (max=200).
[GEO] New lookups performed: 4; cache size: 4
```

| | ip_norm | asn | asn_org | asn_country | network_name | network_country |
|---|---|---|---|---|---|---|
| **0** | 172.190.142.176 | 8075 | MICROSOFT-CORP-MSN-AS-BLOCK - Microsoft Corpor... | GB | cloud | US |
| **1** | 4.204.200.32 | 8075 | MICROSOFT-CORP-MSN-AS-BLOCK - Microsoft Corpor... | US | MSFT | None |
| **2** | 52.169.206.229 | 8075 | MICROSOFT-CORP-MSN-AS-BLOCK - Microsoft Corpor... | US | MSFT | None |
| **3** | 89.248.168.239 | 202425 | INT-NETWORK, SC | NL | NET-2-168 | NL |

In [19]:

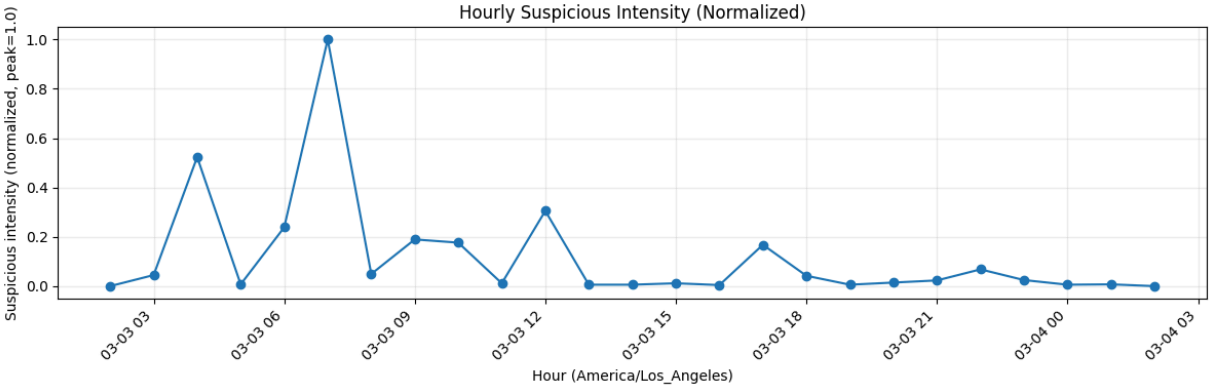
```
# ==========================
# Notebook #3 — Next Cell: "Traffic Graphs" (AWStats-style, data-only)
# Reads daily_waf_snapshot outputs and renders key charts.
# ==========================

import os
import glob
import pandas as pd
import matplotlib.pyplot as plt

SNAP_DIR = globals().get("SNAP_DIR", "/var/log/waf-snapshots")

# Prefer "today" stable filenames; fallback to most recent timestamped copie
today = pd.Timestamp.now().strftime("%Y%m%d")
ip_stable = os.path.join(SNAP_DIR, f"ip_summary_{today}.csv")
hr_stable = os.path.join(SNAP_DIR, f"hourly_intensity_{today}.csv")

def newest(pattern: str) -> str | None:
    files = sorted(glob.glob(os.path.join(SNAP_DIR, pattern)))
    return files[-1] if files else None

ip_path = ip_stable if os.path.exists(ip_stable) else newest("ip_summary_*.c
hr_path = hr_stable if os.path.exists(hr_stable) else newest("hourly_intensi

if not ip_path or not hr_path:
    raise RuntimeError(
        f"Snapshot CSVs not found in {SNAP_DIR}. "
        "Expected ip_summary_YYYYMMDD*.csv and hourly_intensity_YYYYMMDD*.cs
    )
```

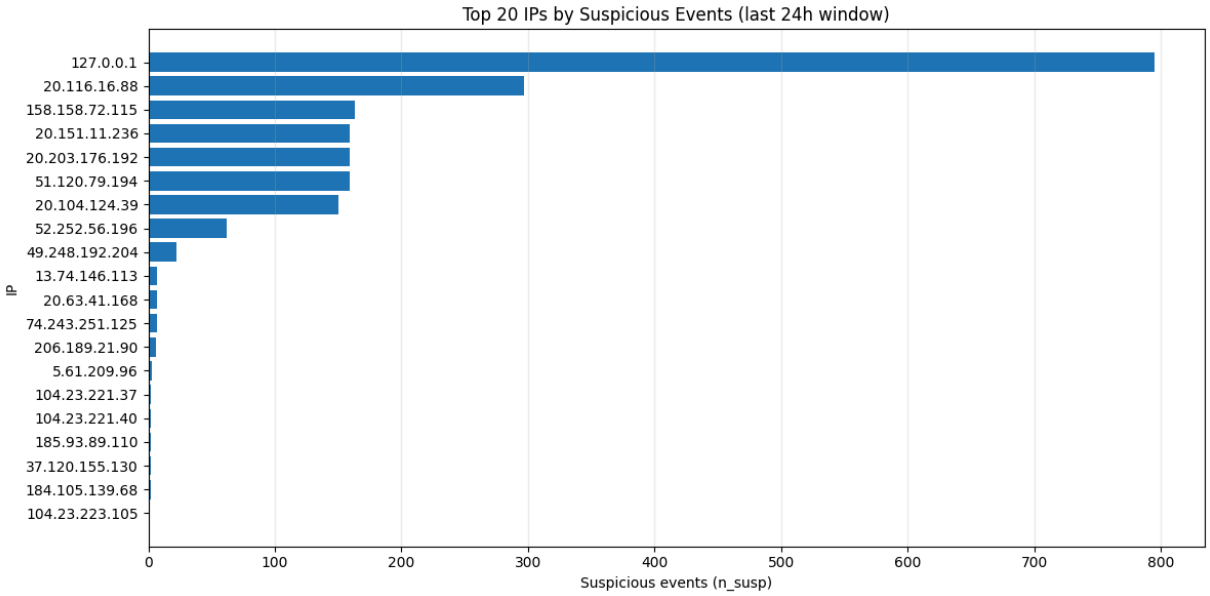
```
df_ip = pd.read_csv(ip_path)
df_hr = pd.read_csv(hr_path)

# Basic normalization / safety
for c in ["n_total", "n_susp", "susp_pct"]:
    if c in df_ip.columns:
        df_ip[c] = pd.to_numeric(df_ip[c], errors="coerce")

if "hour_utc" in df_hr.columns:
    df_hr["hour_utc"] = pd.to_datetime(df_hr["hour_utc"], errors="coerce", u

if "suspicious" in df_hr.columns:
    df_hr["suspicious"] = pd.to_numeric(df_hr["suspicious"], errors="coerce"

if "intensity_norm" in df_hr.columns:
    df_hr["intensity_norm"] = pd.to_numeric(df_hr["intensity_norm"], errors=

HOME_IP = globals().get("HOME_IP", "66.241.78.7")
if "ip_norm" in df_ip.columns:
    df_ip = df_ip[df_ip["ip_norm"].astype(str) != str(HOME_IP)].copy()

print(f"[Charts] Using snapshots:")
print(f"  IP summary : {ip_path}")
print(f"  Hourly     : {hr_path}")
print(f"[Charts] Rows: ip_summary={len(df_ip)} hourly={len(df_hr)}")

# -------------------------
# Chart 1: Hourly suspicious intensity (normalized)
# -------------------------
if df_hr.empty or "hour_utc" not in df_hr.columns:
    print("[Charts][WARN] hourly_intensity CSV is empty or missing hour_utc.
else:
    df_hr2 = df_hr.dropna(subset=["hour_utc"]).sort_values("hour_utc").copy(

    # Plot normalized intensity if available; otherwise plot raw suspicious.
    if "intensity_norm" in df_hr2.columns and df_hr2["intensity_norm"].notna
        y = df_hr2["intensity_norm"].fillna(0.0)
        y_label = "Suspicious intensity (normalized, peak=1.0)"
        title = "Hourly Suspicious Intensity (Normalized)"
    else:
        y = df_hr2["suspicious"].fillna(0).astype(int)
        y_label = "Suspicious events per hour"
        title = "Hourly Suspicious Events"

    plt.figure(figsize=(12, 4))
    plt.plot(df_hr2["hour_utc"].dt.tz_convert("America/Los_Angeles"), y, mar
    plt.title(title)
    plt.xlabel("Hour (America/Los_Angeles)")
    plt.ylabel(y_label)
    plt.xticks(rotation=45, ha="right")
    plt.grid(True, alpha=0.25)
    plt.tight_layout()
    plt.show()

# -------------------------
# Chart 2: Top N suspicious IPs (by n_susp)
```

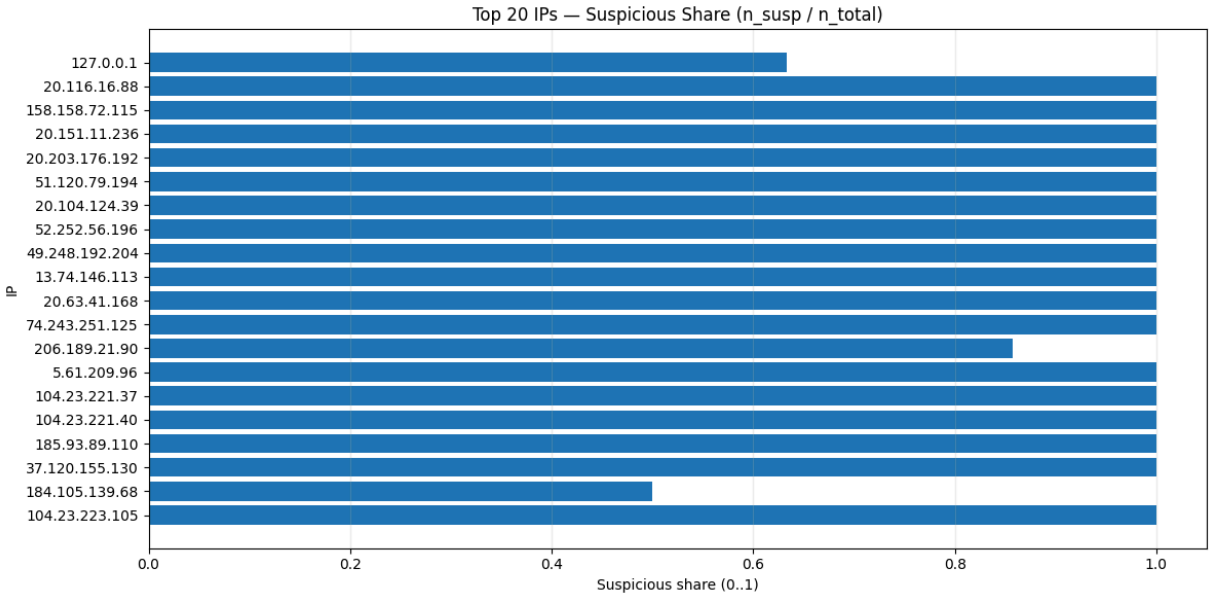
```python
# -------------------------
TOP_N = int(globals().get("TOP_N", 20))

needed = {"ip_norm", "n_susp", "n_total", "susp_pct"}
if not needed.issubset(df_ip.columns):
    print(f"[Charts][WARN] ip_summary missing columns: {sorted(needed - set(
else:
    df_top = (
        df_ip.dropna(subset=["ip_norm"])
            .assign(
                n_susp=lambda d: pd.to_numeric(d["n_susp"], errors="coerce")
                n_total=lambda d: pd.to_numeric(d["n_total"], errors="coerce
                susp_pct=lambda d: pd.to_numeric(d["susp_pct"], errors="coer
            )
            .sort_values(["n_susp", "susp_pct", "n_total"], ascending=[False
            .head(TOP_N)
            .copy()
    )

    if df_top.empty:
        print("[Charts][WARN] No IP rows available for Top-IP chart.")
    else:
        plt.figure(figsize=(12, 6))
        plt.barh(df_top["ip_norm"].astype(str), df_top["n_susp"])
        plt.title(f"Top {min(TOP_N, len(df_top))} IPs by Suspicious Events (
        plt.xlabel("Suspicious events (n_susp)")
        plt.ylabel("IP")
        plt.gca().invert_yaxis()
        plt.grid(True, axis="x", alpha=0.25)
        plt.tight_layout()
        plt.show()

        # Optional: suspicious percentage chart (shows “persistent hostile”
        plt.figure(figsize=(12, 6))
        plt.barh(df_top["ip_norm"].astype(str), df_top["susp_pct"])
        plt.title(f"Top {min(TOP_N, len(df_top))} IPs — Suspicious Share (n_
        plt.xlabel("Suspicious share (0..1)")
        plt.ylabel("IP")
        plt.gca().invert_yaxis()
        plt.grid(True, axis="x", alpha=0.25)
        plt.tight_layout()
        plt.show()

# -------------------------
# OPTIONAL: Geo/ASN summary (if you already built df_geo in earlier cells)
# Expects df_geo columns like:
#   ip_norm, asn, asn_org, asn_country, network_name, network_country
# -------------------------
if "df_geo" in globals() and isinstance(globals()["df_geo"], pd.DataFrame):
    df_geo = globals()["df_geo"].copy()
    if "ip_norm" in df_geo.columns and "ip_norm" in df_ip.columns:
        # Join geo to IP stats
        df_join = df_ip.merge(df_geo, on="ip_norm", how="left")

        # ASN summary by suspicious volume
        if "asn" in df_join.columns and "n_susp" in df_join.columns:
```

```
            df_asn = (
                df_join.dropna(subset=["asn"])
                       .groupby(["asn", "asn_org", "asn_country"], as_index=F
                       .agg(
                           suspicious=("n_susp", "sum"),
                           total=("n_total", "sum"),
                           unique_ips=("ip_norm", "nunique"),
                       )
                       .sort_values(["suspicious", "unique_ips"], ascending=[
                       .head(15)
            )
            if not df_asn.empty:
                plt.figure(figsize=(12, 6))
                labels = df_asn.apply(lambda r: f"AS{r['asn']} {str(r['asn_c
                plt.barh(labels, df_asn["suspicious"])
                plt.title("Top ASNs by Suspicious Volume (joined with Geo/AS
                plt.xlabel("Suspicious events (sum n_susp)")
                plt.ylabel("ASN")
                plt.gca().invert_yaxis()
                plt.grid(True, axis="x", alpha=0.25)
                plt.tight_layout()
                plt.show()
    else:
        print("[Charts][WARN] df_geo present but missing ip_norm; skipping G
else:
    print("[Charts] df_geo not present (OK). Geo/ASN charts will appear once
```

```
[Charts] Using snapshots:
  IP summary : /var/log/waf-snapshots/ip_summary_20260304.csv
  Hourly     : /var/log/waf-snapshots/hourly_intensity_20260304.csv
[Charts] Rows: ip_summary=152 hourly=25
```






In [20]:

```
# ==========================
# Notebook #3 — Next Cell: Snapshot-to-snapshot comparison (effect validatio
# ==========================

import os
import glob
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

SNAP_DIR = "/var/log/waf-snapshots"

# Optional: list IPs you blocked manually (leave empty if you want auto-only
MANUAL_BLOCK_IPS = [
    # "159.203.73.141",
    # "165.227.68.103",
    # "159.203.137.145",
    # "185.177.72.10",
]
```

```
def _latest_two(pattern: str):
    paths = sorted(glob.glob(os.path.join(SNAP_DIR, pattern)))
    if len(paths) < 2:
        raise RuntimeError(f"Need at least 2 files matching {pattern} in {SN
    return paths[-2], paths[-1]

ip_prev_path, ip_curr_path = _latest_two("ip_summary_*.csv")
hr_prev_path, hr_curr_path = _latest_two("hourly_intensity_*.csv")

df_ip_prev = pd.read_csv(ip_prev_path)
df_ip_curr = pd.read_csv(ip_curr_path)
df_hr_prev = pd.read_csv(hr_prev_path)
df_hr_curr = pd.read_csv(hr_curr_path)

print("[Compare] Using snapshots:")
print("  IP summary prev:", ip_prev_path)
print("  IP summary curr:", ip_curr_path)
print("  Hourly    prev :", hr_prev_path)
print("  Hourly    curr :", hr_curr_path)

# Defensive column checks
need_ip = {"ip_norm", "n_total", "n_susp", "susp_pct"}
need_hr = {"hour_utc", "suspicious", "intensity_norm"}

missing_ip_prev = need_ip - set(df_ip_prev.columns)
missing_ip_curr = need_ip - set(df_ip_curr.columns)
missing_hr_prev = need_hr - set(df_hr_prev.columns)
missing_hr_curr = need_hr - set(df_hr_curr.columns)

if missing_ip_prev or missing_ip_curr or missing_hr_prev or missing_hr_curr:
    raise RuntimeError(
        "Snapshot schema mismatch:\n"
        f"  ip_prev missing: {missing_ip_prev}\n"
        f"  ip_curr missing: {missing_ip_curr}\n"
        f"  hr_prev missing: {missing_hr_prev}\n"
        f"  hr_curr missing: {missing_hr_curr}\n"
    )

# Summaries
def summarize_ip(df):
    total_events = int(df["n_total"].sum())
    total_susp = int(df["n_susp"].sum())
    peak_ip_susp = int(df["n_susp"].max()) if len(df) else 0
    uniq_ips = int(df["ip_norm"].nunique())
    return {"total_events": total_events, "total_susp": total_susp, "uniq_ip

def summarize_hr(df):
    peak_susp_hr = int(df["suspicious"].max()) if len(df) else 0
    total_susp = int(df["suspicious"].sum())
    hours = int(len(df))
    return {"peak_susp_per_hour": peak_susp_hr, "total_susp_hourly": total_s

s_ip_prev = summarize_ip(df_ip_prev)
s_ip_curr = summarize_ip(df_ip_curr)
s_hr_prev = summarize_hr(df_hr_prev)
```

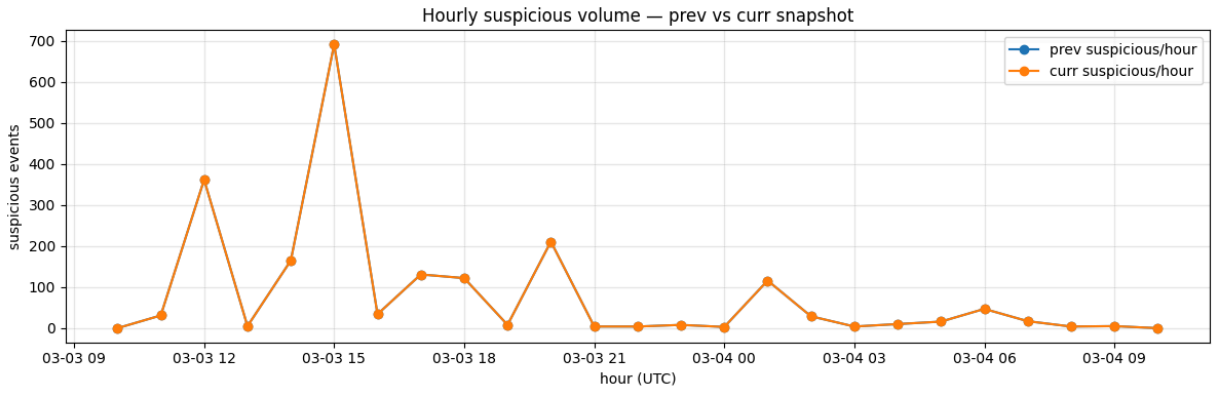
```
s_hr_curr = summarize_hr(df_hr_curr)

print("\n[Compare] Aggregate deltas (prev -> curr):")
print(f"  Unique IPs:            {s_ip_prev['uniq_ips']} -> {s_ip_curr['unic
print(f"  Total events:          {s_ip_prev['total_events']} -> {s_ip_curr['
print(f"  Total suspicious:      {s_ip_prev['total_susp']} -> {s_ip_curr['to
print(f"  Peak suspicious/hour:  {s_hr_prev['peak_susp_per_hour']} -> {s_hr_
print(f"  Peak per-IP suspicious:{s_ip_prev['peak_ip_susp']} -> {s_ip_curr['

# Top IPs comparison (by suspicious count)
TOPN = 20
top_prev = df_ip_prev.sort_values("n_susp", ascending=False).head(TOPN)[["ip
top_curr = df_ip_curr.sort_values("n_susp", ascending=False).head(TOPN)[["ip

print(f"\n[Compare] Top {TOPN} IPs by suspicious (prev):")
display(top_prev.reset_index(drop=True))

print(f"\n[Compare] Top {TOPN} IPs by suspicious (curr):")
display(top_curr.reset_index(drop=True))

# Check manual-blocked IPs presence (if provided)
if MANUAL_BLOCK_IPS:
    prev_hit = df_ip_prev[df_ip_prev["ip_norm"].isin(MANUAL_BLOCK_IPS)][["ip
    curr_hit = df_ip_curr[df_ip_curr["ip_norm"].isin(MANUAL_BLOCK_IPS)][["ip

    print("\n[Compare] Manual-block IPs — previous snapshot:")
    display(prev_hit.sort_values("n_susp", ascending=False).reset_index(drop

    print("\n[Compare] Manual-block IPs — current snapshot:")
    display(curr_hit.sort_values("n_susp", ascending=False).reset_index(drop

    # A simple verdict
    still_present = set(curr_hit["ip_norm"].tolist())
    if still_present:
        print(f"[Compare][WARN] Some manually-blocked IPs still appear in cu
        print("              This can be normal if you blocked mid-window, c
    else:
        print("[Compare][OK] None of the manually-blocked IPs appear in the

# Plot overlay: hourly suspicious counts (prev vs curr)
# Note: hour_utc is stored as string in CSV; parse and align.
df_hr_prev2 = df_hr_prev.copy()
df_hr_curr2 = df_hr_curr.copy()
df_hr_prev2["hour_utc"] = pd.to_datetime(df_hr_prev2["hour_utc"], utc=True,
df_hr_curr2["hour_utc"] = pd.to_datetime(df_hr_curr2["hour_utc"], utc=True,

plt.figure(figsize=(12, 4))
plt.plot(df_hr_prev2["hour_utc"], df_hr_prev2["suspicious"], marker="o", lab
plt.plot(df_hr_curr2["hour_utc"], df_hr_curr2["suspicious"], marker="o", lab
plt.title("Hourly suspicious volume — prev vs curr snapshot")
plt.xlabel("hour (UTC)")
plt.ylabel("suspicious events")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
[Compare] Using snapshots:
  IP summary prev: /var/log/waf-snapshots/ip_summary_20260304.csv
  IP summary curr: /var/log/waf-snapshots/ip_summary_20260304_101501.csv
  Hourly    prev : /var/log/waf-snapshots/hourly_intensity_20260304.csv
  Hourly    curr : /var/log/waf-snapshots/hourly_intensity_20260304_101501.c
sv

[Compare] Aggregate deltas (prev -> curr):
  Unique IPs:            152 -> 152   (Δ +0)
  Total events:          2631 -> 2631 (Δ +0)
  Total suspicious:      2028 -> 2028 (Δ +0)
  Peak suspicious/hour:  692 -> 692 (Δ +0)
  Peak per-IP suspicious:795 -> 795 (Δ +0)

[Compare] Top 20 IPs by suspicious (prev):
```

| | ip_norm | n_susp | n_total | susp_pct |
|---|---|---|---|---|
| **0** | 127.0.0.1 | 795 | 1256 | 0.632962 |
| **1** | 20.116.16.88 | 297 | 297 | 1.000000 |
| **2** | 158.158.72.115 | 163 | 163 | 1.000000 |
| **3** | 20.203.176.192 | 159 | 159 | 1.000000 |
| **4** | 20.151.11.236 | 159 | 159 | 1.000000 |
| **5** | 51.120.79.194 | 159 | 159 | 1.000000 |
| **6** | 20.104.124.39 | 150 | 150 | 1.000000 |
| **7** | 52.252.56.196 | 62 | 62 | 1.000000 |
| **8** | 49.248.192.204 | 22 | 22 | 1.000000 |
| **9** | 20.63.41.168 | 7 | 7 | 1.000000 |
| **10** | 13.74.146.113 | 7 | 7 | 1.000000 |
| **11** | 74.243.251.125 | 7 | 7 | 1.000000 |
| **12** | 206.189.21.90 | 6 | 7 | 0.857143 |
| **13** | 5.61.209.96 | 3 | 3 | 1.000000 |
| **14** | 184.105.139.68 | 2 | 4 | 0.500000 |
| **15** | 185.93.89.110 | 2 | 2 | 1.000000 |
| **16** | 104.23.221.40 | 2 | 2 | 1.000000 |
| **17** | 37.120.155.130 | 2 | 2 | 1.000000 |
| **18** | 104.23.221.37 | 2 | 2 | 1.000000 |
| **19** | 172.94.9.253 | 1 | 1 | 1.000000 |

```
[Compare] Top 20 IPs by suspicious (curr):
```

| | ip_norm | n_susp | n_total | susp_pct |
|---|---|---|---|---|
| **0** | 127.0.0.1 | 795 | 1256 | 0.632962 |
| **1** | 20.116.16.88 | 297 | 297 | 1.000000 |
| **2** | 158.158.72.115 | 163 | 163 | 1.000000 |
| **3** | 20.203.176.192 | 159 | 159 | 1.000000 |
| **4** | 20.151.11.236 | 159 | 159 | 1.000000 |
| **5** | 51.120.79.194 | 159 | 159 | 1.000000 |
| **6** | 20.104.124.39 | 150 | 150 | 1.000000 |
| **7** | 52.252.56.196 | 62 | 62 | 1.000000 |
| **8** | 49.248.192.204 | 22 | 22 | 1.000000 |
| **9** | 20.63.41.168 | 7 | 7 | 1.000000 |
| **10** | 13.74.146.113 | 7 | 7 | 1.000000 |
| **11** | 74.243.251.125 | 7 | 7 | 1.000000 |
| **12** | 206.189.21.90 | 6 | 7 | 0.857143 |
| **13** | 5.61.209.96 | 3 | 3 | 1.000000 |
| **14** | 184.105.139.68 | 2 | 4 | 0.500000 |
| **15** | 185.93.89.110 | 2 | 2 | 1.000000 |
| **16** | 104.23.221.40 | 2 | 2 | 1.000000 |
| **17** | 37.120.155.130 | 2 | 2 | 1.000000 |
| **18** | 104.23.221.37 | 2 | 2 | 1.000000 |
| **19** | 172.94.9.253 | 1 | 1 | 1.000000 |




In [ ]: