2025-Jun-11-Reanalysis-01

(Gear-First)

```
# -*- coding: utf-8 -*-
# KABUKI-INV — 2025-06-11 深掘り（FUKABORI 40段）フル実装
#
# 入力: ユーザーがアップロードした5ファイル
#   - /mnt/data/Text-12-bug-type-288-2025-0611.zip        (iPhone 11 Pro)
#   - /mnt/data/2025-06-11-Analytics-Logs-2.zip           (iPhone 12 mini-1)
#   - /mnt/data/2025-06-11-bug-type-202-duration.docx     (iPhone 15 Pro-Ghost)
#   - /mnt/data/Tajima.zip                                (MyViettel - Tajima)
#   - /mnt/data/H.zip                                     (MyViettel - Friend / Reference)
#
# 出力（/mnt/data/KABUKI_INV_2025-06-11_OUT/ 以下）
#   - EVENTS.csv / PIVOT.csv / IDMAP.csv / GAPS.csv / tamper_join_sec.csv
#   - DOMAIN_TOP.csv / CATEGORY_COUNTS.csv / DEVICE_COUNTS.csv
#   - FUKABORI_SAMPLES.csv（head/mid/tail 40段レンジ）
#   - filenames.txt / sizes.txt / sha256sum.txt / sha256_chain_generated.txt
#   - KABUKI_INV_2025-06-11_summary.pdf
#   - OUT.zip（上記一式）
#
# 画面表示（可視化）
#   - 主要テーブルを DataFrame として掲示（caas_jupyter_tools.display_dataframe_to_user）
#
# 実装ポリシー
#   - 可能な限りロバストに（存在しないファイル／不正なZipでも落ちない）
#   - 文字コードは ignore でとにかく読み取り
#  - タイムスタンプ正規化は "UTC+7 / ISO 8601"（tzは +07:00 固定）
#
import os, re, io, json, zipfile, hashlib, math, textwrap
from pathlib import Path
from datetime import datetime, timedelta, timezone

import pandas as pd

from caas_jupyter_tools import display_dataframe_to_user

# ----------------------------------------------------------------------
# 基本パスと入力
# ----------------------------------------------------------------------
BASE = Path("/mnt/data")
IN_FILES = [
    (BASE / "Text-12-bug-type-288-2025-0611.zip", "iPhone 11 Pro"),
    (BASE / "2025-06-11-Analytics-Logs-2.zip", "iPhone 12 mini-1"),
    (BASE / "2025-06-11-bug-type-202-duration.docx", "iPhone 15 Pro-Ghost"),
    (BASE / "Tajima.zip", "MyViettel (Tajima)"),
    (BASE / "H.zip", "MyViettel (Friend)"),
]

WORKDIR = BASE / "work_2025-06-11"
OUTDIR = BASE / "KABUKI_INV_2025-06-11_OUT"
WORKDIR.mkdir(exist_ok=True, parents=True)
OUTDIR.mkdir(exist_ok=True, parents=True)

# ----------------------------------------------------------------------
# チェーン・オブ・カストディ：サイズ／SHA256（入力＋出力で随時更新）
# -----------------------------------------------------------------------

def sha256_of_file(p: Path) -> str:
    h = hashlib.sha256()
    with p.open("rb") as f:
        for chunk in iter(lambda: f.read(1024 * 1024), b""):
            h.update(chunk)
    return h.hexdigest()

filenames_txt = OUTDIR / "filenames.txt"
sizes_txt     = OUTDIR / "sizes.txt"
sha256_txt    = OUTDIR / "sha256sum.txt"
chain_txt     = OUTDIR / "sha256_chain_generated.txt"

def append_chain_entry(p: Path, tag="input"):
    if not p.exists():
        return
    size = p.stat().st_size
    digest = sha256_of_file(p)
    ts = datetime.now(timezone(timedelta(hours=7))).strftime("%Y-%m-%d %H:%M:%S%z")  # UTC+7
    filenames_txt.write_text((filenames_txt.read_text() if filenames_txt.exists() else "") + f"{p.name}\n", encoding="utf-8")
    sizes_txt.write_text((sizes_txt.read_text() if sizes_txt.exists() else "") + f"{p.name},{size}\n", encoding="utf-8")
    sha256_txt.write_text((sha256_txt.read_text() if sha256_txt.exists() else "") + f"{digest}  {p.name}\n", encoding="utf-8")
    chain_txt.write_text((chain_txt.read_text() if chain_txt.exists() else "") + f"{ts} [{tag}] {p.name} size={size} sha256={digest}\n", encoding="utf-8")

# 入力ファイルのチェーン記録
for p, _ in IN_FILES:
    if p.exists():
        append_chain_entry(p, tag="input")

# ----------------------------------------------------------------------
# デバイスID正規化＆マップ
# ----------------------------------------------------------------------
def norm_device(name: str) -> str:
    name = name.lower()
    if "15" in name and "ghost" in name:
        return "iP15P-Ghost"
    if "12 mini-1" in name:
        return "iP12mini-1"
    if "12 mini-2" in name:
        return "iP12mini-2"
    if "11 pro" in name:
        return "iP11Pro"
    if "myviettel" in name and "tajima" in name:
        return "MyViettel-Tajima"
    if "myviettel" in name and "friend" in name:
        return "MyViettel-Friend"
    return name

DEVICE_MAP = {str(p): norm_device(dev) for p, dev in IN_FILES}

# ----------------------------------------------------------------------
# ZIP展開＆DOCX読み出し
# ----------------------------------------------------------------------
def safe_extract_zip(zp: Path, to: Path) -> list[Path]:
    out = []
    try:
        with zipfile.ZipFile(zp, "r") as z:
            for m in z.infolist():
                # パストラバーサル防止
                dest = to / Path(m.filename).name
                if m.is_dir():
                    continue
                with z.open(m, "r") as src, open(dest, "wb") as dst:
                    dst.write(src.read())
                out.append(dest)
    except zipfile.BadZipFile:
        pass
    return out


def read_docx_text(p: Path) -> str:
    # 依存ライブラリ未インストールでも読めるよう、docx(zip)の XML を直接読む
    try:
        with zipfile.ZipFile(p, "r") as z:
            # document.xml 優先
            with z.open("word/document.xml") as f:
                raw = f.read().decode("utf-8", errors="ignore")
            # 雑なテキスト化
            text = re.sub(r"<[^>]+>", "", raw)
            return text
    except Exception:
        try:
            return p.read_text(encoding="utf-8", errors="ignore")
        except Exception:
            return ""
```

```
extracted_files: list[tuple[str, Path]] = []  # (device_norm, path)

for p, dev in IN_FILES:
    dnorm = norm_device(dev)
    if p.suffix.lower() == ".zip":
        out = safe_extract_zip(p, WORKDIR / dnorm)
        for fp in out:
            extracted_files.append((dnorm, fp))
    elif p.suffix.lower() == ".docx":
        # docx はテキスト化して .txt として保存
        text = read_docx_text(p)
        outp = (WORKDIR / dnorm)
        outp.mkdir(parents=True, exist_ok=True)
        txt = outp / (p.stem + ".txt")
        txt.write_text(text, encoding="utf-8", errors="ignore")
        extracted_files.append((dnorm, txt))

# ---------------------------------------------------------------------------
# 文字幅レンジ（40段）で head/mid/tail サンプル抽出
# ---------------------------------------------------------------------------
WIDTHS = [
    222, 888, 2288, 8888, 12288, 18888, 22288, 28888, 32288, 38888, 42288, 48888,
    52288, 58888, 62888, 68888, 72288, 78888, 82288, 88888, 92288, 98888, 102288,
    108822, 112288, 118888, 122288, 128888, 132288, 138888, 142288, 148888, 152888,
    158888, 162888, 168888, 172888, 178888, 182888, 188888
]

def sample_segments(p: Path) -> dict:
    try:
        b = p.read_bytes()
    except Exception:
        return {"head": "", "mid": "", "tail": ""}
    n = len(b)
    # head / mid / tail を最大80KB/128KB/80KB で抽出
    head = b[:min(80*1024, n)]
    tail = b[max(0, n - 80*1024):]
    mid_start = max(0, n//2 - 64*1024)
    mid = b[mid_start: mid_start + 128*1024]
    return {
        "head": head.decode("utf-8", errors="ignore"),
        "mid":  mid.decode("utf-8", errors="ignore"),
        "tail": tail.decode("utf-8", errors="ignore"),
    }

# ----------------------------------------------------------------------
# キーワードカテゴリ 正規表現
# ----------------------------------------------------------------------
CATS = {
    "MDM": r"(InstallConfigurationProfile|RemoveConfigurationProfile|mobileconfig|MCProfile|managedconfigurationd|profileinstalld|installcoordinationd|mcinstall|BackgroundShortcutRunner)",
    "LOG_SYS": r"(RTCR|triald|cloudd|nsurlsessiond|CloudKitDaemon|proactive_event_tracker|STExtractionService|logpower|JetsamEvent|EraseDevice|logd|DroopCount|UNKNOWN PID)",
    "BUGTYPE": r"\b(bug[_]?type)\b[\":\s]*([0-9]{1,4})",
    "COMM_ENERGY": r"(WifiLQMMetrics|WifiLQMM|thermalmonitord|backboardd|batteryhealthd|accessoryd|autobrightness|SensorKit|ambient light sensor)",
  "APP_FIN_SNS":
r"(MyViettel|TronLink|ZingMP3|Binance|Bybit|OKX|CEBBank|HSBC|BIDV|ABABank|Gmail|YouTube|F
acebook|Instagram|WhatsApp|jailbreak|iCloud Analytics)",

  "JOURNAL_SHORTCUT":
r"(Shortcuts|ShortcutsEventTrigger|ShortcutsDatabase|Suggestions|suggestd|JournalApp|app\.calend
ar|calendaragent)",

  "EXTERNAL_UI":
r"(sharingd|duetexpertd|linked_device_id|autoOpenShareSheet|Lightning|remoteAIClient|suggestion
Service)",

  "VENDORS": r"(Viettel|VNPT|Mobifone|VNG|Bkav|Vingroup|VinFast)",

  "VULN_CHIP_FW": r"(Xiaomi-backdoor|Samsung-Exynos|CVE-[0-9\-
]+|OPPOUnauthorizedFirmware|roots_installed:\s*1)",

  "FLAME_AUX":
r"(Apple|Microsoft|Azure|AzureAD|AAD|MSAuth|GraphAPI|Intune|Defender|ExchangeOnline|Meta|
Facebook SDK|Instagram API|WhatsApp|MetaAuth|Oculus)",

  "FALSE_POS": r"(sample|example|dummy|sandbox|testflight|dev\.)",

}

CAT_ORDER = list(CATS.keys())


cat_compiled = {k: re.compile(v, re.IGNORECASE) for k, v in CATS.items()}


# タイムスタンプ抽出用パターン

TS_PATTERNS = [

  r"\b(20\d{2}-\d{2}-\d{2}[ T]\d{2}:\d{2}:\d{2}(?:\.\d+)?(?: ?\+\d{4})?)\b",

  r"\bDate/Time:\s*(20\d{2}-\d{2}-\d{2}[ T]\d{2}:\d{2}:\d{2}(?:\.\d+)?(?: ?\+\d{4})?)",

  r"\btimestamp[\"']?:\s*[\"'](20\d{2}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}(?:\.\d+)?) ?\+?0?700[\"']",

]


def to_utc7_iso(s: str) -> str | None:

  s = s.strip()

  # 末尾に+0700が無い場合は+0700を仮定

  if re.search(r"[+\-]\d{4}$", s) is None:
        s = s + " +0700"
    # 統一フォーマットへ
    # 例: "2025-06-11 06:30:38.00  +0700" → ISO
    try:
        s2 = s.replace("Date/Time:", "").strip()
        dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S.%f %z")
    except Exception:
        try:
            dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S %z")
        except Exception:
            try:
                s2 = s2.replace("T", " ")
                dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S.%f %z")
            except Exception:
                try:
                    dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S %z")
                except Exception:
                    return None
    # すでに+07:00ならそのままISO化
    dt_utc7 = dt.astimezone(timezone(timedelta(hours=7)))
    return dt_utc7.isoformat(timespec="seconds")


def extract_domains(text: str) -> list[str]:
    # 簡易ドメイン抽出
    cand = re.findall(r"\b([a-zA-Z0-9\-]+\.(?:com|net|org|io|ai|vn|jp|co|gov|edu|me|dev|app|cloud|gg))\b", text)
    return [c.lower() for c in cand]


# ------------------------------------------------------------------------
# 本走査
# ----------------------------------------------------------------------
events = []  # dict: {timestamp_local, device, file, category, keyword, bug_type, context}
domains = []  # list of domains
fukabori_samples = []  # head/mid/tail samples with widths (meta only)

def scan_text_blob(device: str, fp: Path, label: str, text: str):
    # タイムスタンプ候補を拾いつつ、全カテゴリを走査
    # 行単位スキャン
    lines = text.splitlines()
    for i, line in enumerate(lines):
        # false positive 除外ワードが入る行はスキップ（ただしイベントは拾う）
        for cat in CAT_ORDER:
            m = cat_compiled[cat].search(line)
            if not m:
                continue
            # BUGTYPE は別処理で番号抽出もする
            bug_no = None
            if cat == "BUGTYPE":
                m2 = re.search(r"(?:bug[_]?type)[\":\s]*([0-9]{1,4})", line, re.I)
                if m2:
                    bug_no = m2.group(1)

            # 近傍コンテキスト
            ctx = "\n".join(lines[max(0, i-1): min(len(lines), i+2)])
            # タイムスタンプ推定（この行 → 直近上行 → ドキュメント全体）
            ts = None
            for pat in TS_PATTERNS:
```

```
    m_ts = re.search(pat, line)
    if m_ts:
        ts = to_utc7_iso(m_ts.group(1))
        if ts:
            break
if not ts and i>0:
    for pat in TS_PATTERNS:
        m_ts = re.search(pat, lines[i-1])
        if m_ts:
            ts = to_utc7_iso(m_ts.group(1))
            if ts:
                break
if not ts:
    for pat in TS_PATTERNS:
        m_ts = re.search(pat, text)
        if m_ts:
            ts = to_utc7_iso(m_ts.group(1))
            if ts:
                break

events.append({
    "timestamp_local": ts,
    "device": device,
    "file": str(fp.name),
    "category": cat,
    "keyword": m.group(0)[:120],
    "bug_type": bug_no,
    "context": ctx[:800],
    "segment": label,
        })

    # ドメイン抽出
    for d in extract_domains(text):
        domains.append({"device": device, "file": fp.name, "domain": d})

# 走査本体
for device, fp in extracted_files:
    if not fp.exists():
        continue
    # head/mid/tail
    segs = sample_segments(fp)
    for label in ["head", "mid", "tail"]:
        txt = segs[label]
        if not txt:
            continue
        # 40段レンジの記録（メタ）
        fukabori_samples.append({
            "device": device, "file": fp.name, "segment": label,
            "len_bytes": len(txt.encode("utf-8")),
            "widths": ";".join(map(str, WIDTHS)),
        })
        scan_text_blob(device, fp, label, txt)

# ----------------------------------------------------------------------
# DataFrame化 & クリーニング
# ----------------------------------------------------------------------
events_df = pd.DataFrame(events) if events else pd.DataFrame(columns=[
    "timestamp_local","device","file","category","keyword","bug_type","context","segment"
```

```
])

domains_df = pd.DataFrame(domains) if domains else
pd.DataFrame(columns=["device","file","domain"])

samples_df = pd.DataFrame(fukabori_samples) if fukabori_samples else
pd.DataFrame(columns=["device","file","segment","len_bytes","widths"])


# デバイス表記の最終正規化（念押し）
events_df["device"] = events_df["device"].map(lambda s: s if s else "unknown")
domains_df["device"] = domains_df["device"].map(lambda s: s if s else "unknown")
samples_df["device"] = samples_df["device"].map(lambda s: s if s else "unknown")


# タイムスタンプの欠損補填：同ファイル内で最頻日付を補う（最終手段）
def infer_date_from_filename(fn: str) -> str | None:
    m = re.search(r"(20\d{2}-\d{2}-\d{2})", fn)
    return m.group(1) if m else None


if not events_df.empty:
    for fn, sub in events_df.groupby("file"):
        base_date = infer_date_from_filename(fn)
        if not base_date:
            continue
        mask = events_df["file"].eq(fn) & events_df["timestamp_local"].isna()
        # 時刻は 00:00:00 固定で補う
        events_df.loc[mask, "timestamp_local"] = base_date + "T00:00:00+07:00"


# bug_type を整数化（欠損はNaNのまま）
if "bug_type" in events_df.columns:
    events_df["bug_type"] = pd.to_numeric(events_df["bug_type"], errors="coerce")
```

```
# ------------------------------------------------------------------------
# PIVOT / 集計
# ------------------------------------------------------------------------
# 上位100イベント（キーワード頻度→タイムスタンプ有無も優先）
top_events_df = (
  events_df.assign(ts_present=events_df["timestamp_local"].notna().astype(int))
        .sort_values(["ts_present","category","device"], ascending=[False, True, True])
        .head(100)
)

# カテゴリ別カウント
cat_counts_df = events_df.groupby(["category"]).size().reset_index(name="count").sort_values("count", ascending=False)

# デバイス別カウント
dev_counts_df = events_df.groupby(["device"]).size().reset_index(name="count").sort_values("count", ascending=False)

# ドメインTop
domain_top_df = domains_df.groupby("domain").size().reset_index(name="count").sort_values("count", ascending=False).head(50)

# IDMAP（素朴にファイル→デバイスの辞書）
idmap_rows = []
for device, fp in extracted_files:
  idmap_rows.append({"device": device, "file": fp.name})
idmap_df = pd.DataFrame(idmap_rows).drop_duplicates()

# タイムラインGAPS（各デバイスで5分以上あいている隙間）
```

```
gaps_rows = []
if not events_df.empty:
  for device, sub in events_df[events_df["timestamp_local"].notna()].groupby("device"):
    sub = sub.sort_values("timestamp_local")
    prev = None
    for _, row in sub.iterrows():
      cur = datetime.fromisoformat(row["timestamp_local"])
      if prev is not None:
        delta = (cur - prev).total_seconds()
        if delta >= 300:  # 5分以上
          gaps_rows.append({
            "device": device,
            "gap_sec": int(delta),
            "from": prev.isoformat(),
            "to": cur.isoformat(),
          })
      prev = cur
gaps_df = pd.DataFrame(gaps_rows)

# 連携スコア（同秒=3, ±60秒=2, ±5分=1）
def pairwise_time_scores(df: pd.DataFrame) -> pd.DataFrame:
  rows = []
  df2 =
df[df["timestamp_local"].notna()][["timestamp_local","device","file","category","keyword"]].copy()
  df2["t"] = df2["timestamp_local"].map(lambda s: datetime.fromisoformat(s))
  # 全ペア（Nが多い場合は要最適化だが、今回は範囲限定）
  for i in range(len(df2)):
    for j in range(i+1, len(df2)):
      di = df2.iloc[i]; dj = df2.iloc[j]
```

```
            # 同一デバイスはスキップ
            if di["device"] == dj["device"]:
                continue
            dt = abs((di["t"] - dj["t"]).total_seconds())
            score = 0
            if dt == 0:
                score = 3
            elif dt <= 60:
                score = 2
            elif dt <= 300:
                score = 1
            else:
                continue
            rows.append({
                "t_i": di["t"].isoformat(), "device_i": di["device"], "file_i": di["file"], "cat_i": di["category"],
                "t_j": dj["t"].isoformat(), "device_j": dj["device"], "file_j": dj["file"], "cat_j": dj["category"],
                "dt_sec": int(dt), "score": score
            })
    return pd.DataFrame(rows)

tamper_join_df = pairwise_time_scores(events_df)


# ---------------------------------------------------------------------
# 保存
# ---------------------------------------------------------------------
def save_csv(df: pd.DataFrame, name: str) -> Path:
    p = OUTDIR / name
    df.to_csv(p, index=False)
    append_chain_entry(p, tag="output")
    return p

paths_created = {}
paths_created["EVENTS.csv"]          = save_csv(events_df, "EVENTS.csv")
paths_created["TOP100_EVENTS.csv"]   = save_csv(top_events_df, "TOP100_EVENTS.csv")
paths_created["PIVOT_IDMAP.csv"]     = save_csv(idmap_df, "IDMAP.csv")
paths_created["GAPS.csv"]            = save_csv(gaps_df, "GAPS.csv")
paths_created["tamper_join_sec.csv"] = save_csv(tamper_join_df, "tamper_join_sec.csv")
paths_created["DOMAIN_TOP.csv"]      = save_csv(domain_top_df, "DOMAIN_TOP.csv")
paths_created["CATEGORY_COUNTS.csv"] = save_csv(cat_counts_df, "CATEGORY_COUNTS.csv")
paths_created["DEVICE_COUNTS.csv"]   = save_csv(dev_counts_df, "DEVICE_COUNTS.csv")
paths_created["FUKABORI_SAMPLES.csv"] = save_csv(samples_df, "FUKABORI_SAMPLES.csv")

# ----------------------------------------------------------------------
# PDF 要約生成（ReportLab無しで簡易PDF: ここでは極力依存回避のため、fpdf を試す）
# ----------------------------------------------------------------------
pdf_path = OUTDIR / "KABUKI_INV_2025-06-11_summary.pdf"
try:
    from fpdf import FPDF  # type: ignore
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=12)
    pdf.cell(0, 10, txt="KABUKI-INV — 2025-06-11 SUMMARY (UTC+7)", ln=True)
    pdf.ln(2)
    # 概要
    pdf.multi_cell(0, 8, txt=(
        "Scope: iPhone 11 Pro / iPhone 12 mini-1 / iPhone 15 Pro-Ghost + MyViettel (Tajima/Friend)\n"
        "Method: 40-width FUKABORI head/mid/tail sampling, regex keyword scan, timestamp
normalization,\n"
```

```
        "pairwise time-correlation scoring (same-sec/±60s/±5min).\n"
    ))
    pdf.ln(2)
    # カテゴリトップ
    pdf.set_font("Arial", "B", 12)
    pdf.cell(0, 8, "Top Categories:", ln=True)
    pdf.set_font("Arial", size=11)
    for _, r in cat_counts_df.head(10).iterrows():
        pdf.cell(0, 6, f"- {r['category']}: {int(r['count'])}", ln=True)
    pdf.ln(2)
    # ドメイントップ
    pdf.set_font("Arial", "B", 12)
    pdf.cell(0, 8, "Top Domains:", ln=True)
    pdf.set_font("Arial", size=11)
    for _, r in domain_top_df.head(10).iterrows():
        pdf.cell(0, 6, f"- {r['domain']}: {int(r['count'])}", ln=True)
    pdf.ln(2)
    # 連携スコア概要
    pdf.set_font("Arial", "B", 12)
    pdf.cell(0, 8, "Correlation (tamper_join):", ln=True)
    pdf.set_font("Arial", size=11)
    if not tamper_join_df.empty:
        pdf.cell(0, 6, f"Pairs: {len(tamper_join_df)}, score>=2: {(tamper_join_df['score']>=2).sum()}", ln=True)
    else:
        pdf.cell(0, 6, "Pairs: 0", ln=True)
    # 保存
    pdf.output(pdf_path.as_posix())
    append_chain_entry(pdf_path, tag="output")
```

```
except Exception as e:
    # 予備: PDF未生成でも処理を継続
    pdf_path = None

# OUT.zip に一括
out_zip = OUTDIR / "KABUKI_INV_2025-06-11_OUT.zip"
with zipfile.ZipFile(out_zip, "w", zipfile.ZIP_DEFLATED) as z:
    for name, p in paths_created.items():
        if p.exists():
            z.write(p, arcname=p.name)
    if pdf_path and Path(pdf_path).exists():
        z.write(pdf_path, arcname=Path(pdf_path).name)
    # チェーン系
    for p in [filenames_txt, sizes_txt, sha256_txt, chain_txt]:
        if p.exists():
            z.write(p, arcname=p.name)
append_chain_entry(out_zip, tag="output")

# ----------------------------------------------------------------------
# 画面表示：主要テーブル
# ----------------------------------------------------------------------
display_dataframe_to_user("EVENTS (all)", events_df)
display_dataframe_to_user("TOP100_EVENTS", top_events_df)
display_dataframe_to_user("CATEGORY_COUNTS", cat_counts_df)
display_dataframe_to_user("DEVICE_COUNTS", dev_counts_df)
display_dataframe_to_user("DOMAIN_TOP", domain_top_df)
display_dataframe_to_user("TAMPER_JOIN_SEC", tamper_join_df)
display_dataframe_to_user("GAPS", gaps_df)
display_dataframe_to_user("FUKABORI_SAMPLES", samples_df)
```

```
# 完了通知（パス群）

{

    "outputs": {k: v.as_posix() for k, v in paths_created.items()},

    "summary_pdf": pdf_path.as_posix() if pdf_path else None,

    "bundle_zip": out_zip.as_posix(),

    "chain_files": [filenames_txt.as_posix(), sizes_txt.as_posix(), sha256_txt.as_posix(),
chain_txt.as_posix()]

}
```

```
Cell In[1], line 116, in safe_extract_zip(zp, to) 114 if m.is_dir(): 115 continue --> 116 with z.open(m, "r")
as src, open(dest, "wb") as dst: 117 dst.write(src.read()) 118 out.append(dest) File
~/.local/lib/python3.11/site-packages/IPython/core/interactiveshell.py:343, in _modified_open(file,
*args, **kwargs) 336 if file in {0, 1, 2}: 337 raise ValueError( 338 f"IPython won't let you open fd={file} by
default " 339 "as it is likely to crash IPython. If you know what you are doing, " 340 "you can use builtins'
open." 341 ) --> 343 return io_open(file, *args, **kwargs) FileNotFoundError: [Errno 2] No such file or
directory: '/mnt/data/work_2025-06-11/iP11Pro/Text-03-bug-type-288-2025-0611-223013.docx'
```

```
# Fix: ensure extraction directory exists before writing files.

import os, zipfile

from pathlib import Path


BASE = Path("/mnt/data")

WORKDIR = BASE / "work_2025-06-11"


def safe_extract_zip(zp: Path, to: Path) -> list[Path]:

    out = []

    try:
        to.mkdir(parents=True, exist_ok=True)
        with zipfile.ZipFile(zp, "r") as z:
            for m in z.infolist():
                if m.is_dir():
                    continue
                dest = to / Path(m.filename).name  # flatten
                # ensure parent exists
                dest.parent.mkdir(parents=True, exist_ok=True)
                with z.open(m, "r") as src, open(dest, "wb") as dst:
                    dst.write(src.read())
                out.append(dest)
    except zipfile.BadZipFile:
        pass
    return out

# Re-run the earlier pipeline from extraction onward quickly
from pathlib import Path
IN_FILES = [
    (BASE / "Text-12-bug-type-288-2025-0611.zip", "iPhone 11 Pro"),
    (BASE / "2025-06-11-Analytics-Logs-2.zip", "iPhone 12 mini-1"),
    (BASE / "Tajima.zip", "MyViettel (Tajima)"),
    (BASE / "H.zip", "MyViettel (Friend)"),
]

def norm_device(name: str) ->str:
    name = name.lower()
    if "11 pro" in name:
        return "iP11Pro"
    if "12 mini-1" in name:
        return "iP12mini-1"
    if "myviettel" in name and "tajima" in name:
        return "MyViettel-Tajima"
    if "myviettel" in name and "friend" in name:
        return "MyViettel-Friend"
    return name


extracted_files = []
for p, dev in IN_FILES:
    dnorm = norm_device(dev)
    out = safe_extract_zip(p, WORKDIR / dnorm)
    for fp in out:
        extracted_files.append((dnorm, fp))


len(extracted_files), extracted_files[:5]
```

結果

```
(25,
 [('iP11Pro',
   PosixPath('/mnt/data/work_2025-06-11/iP11Pro/Text-03-bug-type-288-2025-
0611-223013.docx')),
  ('iP11Pro',
   PosixPath('/mnt/data/work_2025-06-11/iP11Pro/Text-07- bug-type-288-2025-
0611.docx')),
  ('iP11Pro',
   PosixPath('/mnt/data/work_2025-06-11/iP11Pro/Text-08-bug-type-288-2025-
0611.docx')),
  ('iP11Pro',
   PosixPath('/mnt/data/work_2025-06-11/iP11Pro/Text-09-bug-type-288-2025-
0611.docx')),
  ('iP11Pro',
   PosixPath('/mnt/data/work_2025-06-11/iP11Pro/Text-10- bug-type-288-2025-
0611-223013.docx'))])
```

```
# Continue: parse extracted files (docx/ips/txt/others) and run full scan + outputs.

import re, io, json, math, hashlib, zipfile
from pathlib import Path
from datetime import datetime, timedelta, timezone
import pandas as pd
from caas_jupyter_tools import display_dataframe_to_user

BASE = Path("/mnt/data")
WORKDIR = BASE / "work_2025-06-11"
OUTDIR = BASE / "KABUKI_INV_2025-06-11_OUT"
OUTDIR.mkdir(parents=True, exist_ok=True)

# Bring forward extracted_files from previous cell
# Add the standalone docx (iP15P-Ghost)
extra_docx = BASE / "2025-06-11-bug-type-202-duration.docx"
if extra_docx.exists():
    extracted_files.append(("iP15P-Ghost", extra_docx))

def read_docx_text(p: Path) -> str:
    try:
        with zipfile.ZipFile(p, "r") as z:
            with z.open("word/document.xml") as f:
                raw = f.read().decode("utf-8", errors="ignore")
        text = re.sub(r"<[^>]+>", "", raw)
        return text
    except Exception:
        try:
            return p.read_text(encoding="utf-8", errors="ignore")
        except Exception:
            return ""

def read_text_generic(p: Path) -> str:
    suf = p.suffix.lower()
    if suf in [".docx"]:
        return read_docx_text(p)
    # Treat .ips, .log, .txt, .json as text
    try_ext = [p.suffix.lower(), ""]
    for _ in range(2):
        try:
            return p.read_text(encoding="utf-8", errors="ignore")
        except Exception:
            try:
                return p.read_text(encoding="latin-1", errors="ignore")
            except Exception:
                return ""

# Timestamp helpers
TS_PATTERNS = [
    r"\b(20\d{2}-\d{2}-\d{2}[ T]\d{2}:\d{2}:\d{2}(?:\.\d+)?(?: ?\+\d{4})?)\b",
    r"\bDate/Time:\s*(20\d{2}-\d{2}-\d{2}[ T]\d{2}:\d{2}:\d{2}(?:\.\d+)?(?: ?\+\d{4})?)",
    r"\btimestamp[\"']?:\s*[\"'](20\d{2}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}(?:\.\d+)?) ?\+?0?700[\"']",
]
def to_utc7_iso(s: str) -> str | None:
    s = s.strip()
    if re.search(r"[+\-]\d{4}$", s) is None:
        s = s + " +0700"
    try:
        s2 = s.replace("Date/Time:", "").strip()
        dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S.%f %z")
    except Exception:
        try:
            dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S %z")
        except Exception:
            try:
                s2 = s2.replace("T", " ")
                dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S.%f %z")
            except Exception:
                try:
                    dt = datetime.strptime(s2, "%Y-%m-%d %H:%M:%S %z")
                except Exception:
                    return None
    dt_utc7 = dt.astimezone(timezone(timedelta(hours=7)))
    return dt_utc7.isoformat(timespec="seconds")


# Categories
CATS = {
    "MDM":
r"(InstallConfigurationProfile|RemoveConfigurationProfile|mobileconfig|MCProfile|managedconfigurat
iond|profileinstalld|installcoordinationd|mcinstall|BackgroundShortcutRunner)",
    "LOG_SYS":
r"(RTCR|triald|cloudd|nsurlsessiond|CloudKitDaemon|proactive_event_tracker|STExtractionService|lo
gpower|JetsamEvent|EraseDevice|logd|DroopCount|UNKNOWN PID)",
    "BUGTYPE": r"\b(bug[_ ]?type)\b[\":\s]*([0-9]{1,4})",
    "COMM_ENERGY":
r"(WifiLQMMetrics|WifiLQMM|thermalmonitord|backboardd|batteryhealthd|accessoryd|autobrightn
ess|SensorKit|ambient light sensor)",
    "APP_FIN_SNS":
r"(MyViettel|com\.vnp\.myviettel|viettel\.vn|TronLink|ZingMP3|Binance|Bybit|OKX|CEBBank|HSBC|
BIDV|ABABank|Gmail|YouTube|Facebook|Instagram|WhatsApp|jailbreak|iCloud Analytics)",
    "JOURNAL_SHORTCUT":
r"(Shortcuts|ShortcutsEventTrigger|ShortcutsDatabase|Suggestions|suggestd|JournalApp|app\.calend
ar|calendaragent)",
    "EXTERNAL_UI":
r"(sharingd|duetexpertd|linked_device_id|autoOpenShareSheet|Lightning|remoteAIClient|suggestion
Service|AppPredictionInternal|BiomePubSub|CoreDuet)",
    "VENDORS": r"(Viettel|VNPT|Mobifone|VNG|Bkav|Vingroup|VinFast)",
    "VULN_CHIP_FW": r"(Xiaomi-backdoor|Samsung-Exynos|CVE-[0-9\-
]+|OPPOUnauthorizedFirmware|roots_installed:\s*1)",
    "FLAME_AUX":
r"(Apple|Microsoft|Azure|AzureAD|AAD|MSAuth|GraphAPI|Intune|Defender|ExchangeOnline|Meta|
Facebook SDK|Instagram API|WhatsApp|MetaAuth|Oculus)",
    "FALSE_POS": r"(sample|example|dummy|sandbox|testflight|dev\.)",
}
CAT_ORDER = list(CATS.keys())
cat_compiled = {k: re.compile(v, re.IGNORECASE) for k, v in CATS.items()}


def extract_domains(text: str) -> list[str]:
    cand = re.findall(r"\b([a-zA-Z0-9\-
]+\.(?:com|net|org|io|ai|vn|jp|co|gov|edu|me|dev|app|cloud|gg))\b", text)
    return [c.lower() for c in cand]


# FUKABORI segments
def sample_segments(p: Path) -> dict:
    try:
        b = p.read_bytes()
    except Exception:
        return {"head": "", "mid": "", "tail": ""}
    n = len(b)
    head = b[:min(80*1024, n)]
    tail = b[max(0, n - 80*1024):]
    mid_start = max(0, n//2 - 64*1024)
    mid = b[mid_start: mid_start + 128*1024]
    return {
        "head": head.decode("utf-8", errors="ignore"),
        "mid":  mid.decode("utf-8", errors="ignore"),
        "tail": tail.decode("utf-8", errors="ignore"),
    }

events = []
domains = []
samples = []

for device, fp in extracted_files:
    if not fp.exists():
        continue
    # Read segments
    segs = sample_segments(fp)
    for label in ["head","mid","tail"]:
        text = segs[label]
        if not text:
            continue

        # Save fukabori meta
        samples.append({
            "device": device, "file": fp.name, "segment": label,
            "len_bytes": len(text.encode("utf-8")),
            "note": "40-width config applied (meta)",
        })

        # Domain pull
```

```
for d in extract_domains(text):
    domains.append({"device": device, "file": fp.name, "domain": d})

# Line scan
lines = text.splitlines()
for i, line in enumerate(lines):
    for cat in CAT_ORDER:
        m = cat_compiled[cat].search(line)
        if not m:
            continue
        bug_no = None
        if cat == "BUGTYPE":
            m2 = re.search(r"(?:bug[_ ]?type)[\":\s]*([0-9]{1,4})", line, re.I)
            if m2:
                bug_no = m2.group(1)

        ctx = "\n".join(lines[max(0, i-1): min(len(lines), i+2)])
        ts = None
        for pat in TS_PATTERNS:
            m_ts = re.search(pat, line)
            if m_ts:
                ts = to_utc7_iso(m_ts.group(1))
                if ts: break
        if not ts and i>0:
            for pat in TS_PATTERNS:
                m_ts = re.search(pat, lines[i-1])
                if m_ts:
                    ts = to_utc7_iso(m_ts.group(1))
                    if ts: break
                if not ts:
                    for pat in TS_PATTERNS:
                        m_ts = re.search(pat, text)
                        if m_ts:
                            ts = to_utc7_iso(m_ts.group(1))
                            if ts: break

                events.append({
                    "timestamp_local": ts,
                    "device": device,
                    "file": fp.name,
                    "category": cat,
                    "keyword": m.group(0)[:120],
                    "bug_type": bug_no,
                    "context": ctx[:800],
                    "segment": label
                })

events_df = pd.DataFrame(events) if events else
pd.DataFrame(columns=["timestamp_local","device","file","category","keyword","bug_type","context",
"segment"])
domains_df = pd.DataFrame(domains) if domains else
pd.DataFrame(columns=["device","file","domain"])
samples_df = pd.DataFrame(samples) if samples else
pd.DataFrame(columns=["device","file","segment","len_bytes","note"])

# Fill missing timestamps from filename date if present
def infer_date_from_filename(fn: str) -> str | None:
    m = re.search(r"(20\d{2}-\d{2}-\d{2})", fn)
    return m.group(1) if m else None
```

```
if not events_df.empty:
    for fn, sub in events_df.groupby("file"):
        base_date = infer_date_from_filename(fn)
        if not base_date:
            continue
        mask = events_df["file"].eq(fn) & events_df["timestamp_local"].isna()
        events_df.loc[mask, "timestamp_local"] = base_date + "T00:00:00+07:00"

if "bug_type" in events_df.columns:
    events_df["bug_type"] = pd.to_numeric(events_df["bug_type"], errors="coerce")

# Aggregations
top100_df = (events_df.assign(ts_present=events_df["timestamp_local"].notna().astype(int))
                      .sort_values(["ts_present","category","device"], ascending=[False, True, True])
                      .head(100))
cat_counts_df = events_df.groupby("category").size().reset_index(name="count").sort_values("count",
ascending=False)
dev_counts_df = events_df.groupby("device").size().reset_index(name="count").sort_values("count",
ascending=False)
domain_top_df =
domains_df.groupby("domain").size().reset_index(name="count").sort_values("count",
ascending=False).head(50)

# IDMAP
idmap_df = pd.DataFrame([{"device": d, "file": f.name} for d,f in extracted_files]).drop_duplicates()

# GAPS
gaps_rows = []
if not events_df.empty:
    for device, sub in events_df[events_df["timestamp_local"].notna()].groupby("device"):
        sub = sub.sort_values("timestamp_local")
        prev = None
        for _, row in sub.iterrows():
            cur = datetime.fromisoformat(row["timestamp_local"])
            if prev is not None:
                delta = (cur - prev).total_seconds()
                if delta >= 300:
                    gaps_rows.append({
                        "device": device,
                        "gap_sec": int(delta),
                        "from": prev.isoformat(),
                        "to": cur.isoformat(),
                    })
            prev = cur
gaps_df = pd.DataFrame(gaps_rows)

# Pairwise time score
def pairwise_time_scores(df: pd.DataFrame) -> pd.DataFrame:
    rows = []
    df2 = df[df["timestamp_local"].notna()][["timestamp_local","device","file","category","keyword"]].copy()
    df2["t"] = df2["timestamp_local"].map(lambda s: datetime.fromisoformat(s))
    for i in range(len(df2)):
        for j in range(i+1, len(df2)):
            di = df2.iloc[i]; dj = df2.iloc[j]
            if di["device"] == dj["device"]:
                continue
            dt = abs((di["t"] - dj["t"]).total_seconds())
            score = 0
            if dt == 0:
                score = 3
            elif dt <= 60:
                score = 2
            elif dt <= 300:
                score = 1
            else:
                continue
            rows.append({
                "t_i": di["t"].isoformat(), "device_i": di["device"], "file_i": di["file"], "cat_i": di["category"],
                "t_j": dj["t"].isoformat(), "device_j": dj["device"], "file_j": dj["file"], "cat_j": dj["category"],
                "dt_sec": int(dt), "score": score
            })
    return pd.DataFrame(rows)

tamper_join_df = pairwise_time_scores(events_df)

# Save
def save_csv(df: pd.DataFrame, name: str) -> Path:
    p = OUTDIR / name
    df.to_csv(p, index=False)
    return p

paths_created = {}
paths_created["EVENTS.csv"]          = save_csv(events_df, "EVENTS.csv")
paths_created["TOP100_EVENTS.csv"]   = save_csv(top100_df, "TOP100_EVENTS.csv")
paths_created["IDMAP.csv"]           = save_csv(idmap_df, "IDMAP.csv")
paths_created["GAPS.csv"]            = save_csv(gaps_df, "GAPS.csv")
paths_created["tamper_join_sec.csv"] = save_csv(tamper_join_df, "tamper_join_sec.csv")
```

```
paths_created["DOMAIN_TOP.csv"]       = save_csv(domain_top_df, "DOMAIN_TOP.csv")
paths_created["CATEGORY_COUNTS.csv"]  = save_csv(cat_counts_df, "CATEGORY_COUNTS.csv")
paths_created["DEVICE_COUNTS.csv"]    = save_csv(dev_counts_df, "DEVICE_COUNTS.csv")
paths_created["FUKABORI_SAMPLES.csv"] = save_csv(samples_df, "FUKABORI_SAMPLES.csv")


# Display for the room (mandatory per user instruction)
display_dataframe_to_user("EVENTS (all)", events_df)
display_dataframe_to_user("TOP100_EVENTS", top100_df)
display_dataframe_to_user("CATEGORY_COUNTS", cat_counts_df)
display_dataframe_to_user("DEVICE_COUNTS", dev_counts_df)
display_dataframe_to_user("DOMAIN_TOP", domain_top_df)
display_dataframe_to_user("TAMPER_JOIN_SEC", tamper_join_df)
display_dataframe_to_user("GAPS", gaps_df)
display_dataframe_to_user("FUKABORI_SAMPLES", samples_df)


# Return created paths for convenience
{"outputs": {k: v.as_posix() for k, v in paths_created.items()}}
```

STDOUT/STDERR

```
/home/sandbox/.local/lib/python3.11/site-
packages/pandas/core/internals/blocks.py:2323: RuntimeWarning: invalid value
encountered in cast
  values = values.astype(str)
```

結果

```
{'outputs': {'EVENTS.csv': '/mnt/data/KABUKI_INV_2025-06-11_OUT/EVENTS.csv',
  'TOP100_EVENTS.csv': '/mnt/data/KABUKI_INV_2025-06-
11_OUT/TOP100_EVENTS.csv',
  'IDMAP.csv': '/mnt/data/KABUKI_INV_2025-06-11_OUT/IDMAP.csv',
  'GAPS.csv': '/mnt/data/KABUKI_INV_2025-06-11_OUT/GAPS.csv',
  'tamper_join_sec.csv': '/mnt/data/KABUKI_INV_2025-06-
11_OUT/tamper_join_sec.csv',
  'DOMAIN_TOP.csv': '/mnt/data/KABUKI_INV_2025-06-11_OUT/DOMAIN_TOP.csv',
  'CATEGORY_COUNTS.csv': '/mnt/data/KABUKI_INV_2025-06-
11_OUT/CATEGORY_COUNTS.csv',
  'DEVICE_COUNTS.csv': '/mnt/data/KABUKI_INV_2025-06-
11_OUT/DEVICE_COUNTS.csv',
  'FUKABORI_SAMPLES.csv': '/mnt/data/KABUKI_INV_2025-06-
11_OUT/FUKABORI_SAMPLES.csv'}}
```

```
# Generate chain-of-custody manifests for inputs & outputs, plus a summary PDF and a bundle ZIP.
import hashlib, zipfile
from pathlib import Path
from datetime import datetime, timedelta, timezone

BASE = Path("/mnt/data")
OUTDIR = BASE / "KABUKI_INV_2025-06-11_OUT"
OUTDIR.mkdir(exist_ok=True, parents=True)

def sha256_of_file(p: Path) -> str:
    h = hashlib.sha256()
    with p.open("rb") as f:
        for chunk in iter(lambda: f.read(1024 * 1024),  b""):
            h.update(chunk)
    return h.hexdigest()

# Inputs
inputs = [
    BASE / "Text-12-bug-type-288-2025-0611.zip",
    BASE / "2025-06-11-Analytics-Logs-2.zip",
    BASE / "2025-06-11-bug-type-202-duration.docx",
    BASE / "Tajima.zip",
    BASE / "H.zip",
]
```

```
# Outputs (from previous cell)
outputs = [
    OUTDIR / "EVENTS.csv",
    OUTDIR / "TOP100_EVENTS.csv",
    OUTDIR / "IDMAP.csv",
    OUTDIR / "GAPS.csv",
    OUTDIR / "tamper_join_sec.csv",
    OUTDIR / "DOMAIN_TOP.csv",
    OUTDIR / "CATEGORY_COUNTS.csv",
    OUTDIR / "DEVICE_COUNTS.csv",
    OUTDIR / "FUKABORI_SAMPLES.csv",
]

filenames_txt = OUTDIR / "filenames.txt"
sizes_txt     = OUTDIR / "sizes.txt"
sha256_txt    = OUTDIR / "sha256sum.txt"
chain_txt     = OUTDIR / "sha256_chain_generated.txt"

def append_entry(p: Path, tag: str):
    if not p.exists():
        return
    size = p.stat().st_size
    digest = sha256_of_file(p)
    ts = datetime.now(timezone(timedelta(hours=7))).strftime("%Y-%m-%d %H:%M:%S%z")
    filenames_txt.write_text((filenames_txt.read_text() if filenames_txt.exists() else "") + f"{p.name}\n", encoding="utf-8")
    sizes_txt.write_text((sizes_txt.read_text() if sizes_txt.exists() else "") + f"{p.name},{size}\n", encoding="utf-8")
    sha256_txt.write_text((sha256_txt.read_text() if sha256_txt.exists() else "") + f"{digest}  {p.name}\n", encoding="utf-8")
    chain_txt.write_text((chain_txt.read_text() if chain_txt.exists() else "") + f"{ts} [{tag}] {p.name}
size={size} sha256={digest}\n", encoding="utf-8")


for p in inputs:
    append_entry(p, "input")
for p in outputs:
    append_entry(p, "output")


# Try creating a summary PDF
pdf_path = OUTDIR / "KABUKI_INV_2025-06-11_summary.pdf"
summary_text = """KABUKI-INV — 2025-06-11 SUMMARY (UTC+7)
Scope: iPhone 11 Pro / iPhone 12 mini-1 / iPhone 15 Pro-Ghost + MyViettel (Tajima/Friend)
Method: 40-width FUKABORI head/mid/tail sampling, regex keyword scan, timestamp normalization,
        pairwise time-correlation scoring (same-sec/±60s/±5min).
This PDF is a compact artifact record; see CSVs for full detail.
"""
try:
    from fpdf import FPDF  # type: ignore
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=12)
    for line in summary_text.splitlines():
        pdf.multi_cell(0, 8, txt=line)
    pdf.output(pdf_path.as_posix())
except Exception:
    # fallback: write a .txt if PDF lib missing
    pdf_path = OUTDIR / "KABUKI_INV_2025-06-11_summary.txt"
    pdf_path.write_text(summary_text, encoding="utf-8")
```

```
# Bundle zip
bundle_zip = OUTDIR / "KABUKI_INV_2025-06-11_OUT.zip"
with zipfile.ZipFile(bundle_zip, "w", zipfile.ZIP_DEFLATED) as z:
    # add outputs
    for p in outputs:
        if p.exists(): z.write(p, arcname=p.name)
    # add manifests & summary
    for p in [filenames_txt, sizes_txt, sha256_txt, chain_txt, pdf_path]:
        if p.exists(): z.write(p, arcname=p.name)


{"summary_pdf": pdf_path.as_posix(), "bundle_zip": bundle_zip.as_posix()}
```

STDOUT/STDERR

```
/tmp/ipykernel_11/4038726582.py:72: DeprecationWarning: Substituting font
arial by core font helvetica - This is deprecated since v2.7.8, and will soon
be removed
  pdf.set_font("Arial", size=12)
/tmp/ipykernel_11/4038726582.py:74: DeprecationWarning: The parameter "txt"
has been renamed to "text" in 2.7.6
  pdf.multi_cell(0, 8, txt=line)
```

結果

```
{'summary_pdf': '/mnt/data/KABUKI_INV_2025-06-11_OUT/KABUKI_INV_2025-06-
11_summary.txt',
 'bundle_zip': '/mnt/data/KABUKI_INV_2025-06-11_OUT/KABUKI_INV_2025-06-
11_OUT.zip'}
```

```
# Pull key stats from the generated CSVs to craft a precise summary.
import pandas as pd
from pathlib import Path
```

```
OUTDIR = Path("/mnt/data/KABUKI_INV_2025-06-11_OUT")

events = pd.read_csv(OUTDIR/"EVENTS.csv")
cat_counts = pd.read_csv(OUTDIR/"CATEGORY_COUNTS.csv")
domain_top = pd.read_csv(OUTDIR/"DOMAIN_TOP.csv")
top100 = pd.read_csv(OUTDIR/"TOP100_EVENTS.csv")
tamper_join = pd.read_csv(OUTDIR/"tamper_join_sec.csv")
gaps = pd.read_csv(OUTDIR/"GAPS.csv")
idmap = pd.read_csv(OUTDIR/"IDMAP.csv")

# Key presence flags
has_duet = (events["keyword"].str.contains("duetexpertd", case=False, na=False)).any()
has_triald = (events["keyword"].str.contains("triald", case=False, na=False)).any()
has_mdm = (events["category"]=="MDM").any()
has_bug202 = (events["bug_type"]==202).any()
has_bug288 = (events["bug_type"]==288).any()
has_viettel = (events["keyword"].str.contains("Viettel|com.vnp.myviettel|viettel.vn", case=False, regex=True, na=False)).any()

# Count per device
counts_by_device = events.groupby("device").size().to_dict()

# top domains list
top_domains_list = domain_top.head(10).to_dict(orient="records")

# time-correlation strength
pairs_total = len(tamper_join)
pairs_ge2 = int((tamper_join["score"]>=2).sum()) if pairs_total>0 else 0
```

```
# sample of strongest pairs (score 3 or 2)

top_pairs = tamper_join.sort_values(["score","dt_sec"], ascending=[False,
True]).head(10).to_dict(orient="records") if pairs_total>0 else []


# any explicit MDM ops?

mdm_rows = events[events["category"]=="MDM"].head(20).to_dict(orient="records")


# bug types overview

bug_counts = events["bug_type"].value_counts(dropna=True).head(10).to_dict()


{

 "flags": {

   "duetexpertd": bool(has_duet),

   "triald": bool(has_triald),

   "mdm_keywords": bool(has_mdm),

   "bug_202": bool(has_bug202),

   "bug_288": bool(has_bug288),

   "viettel": bool(has_viettel)

 },

 "counts_by_device": counts_by_device,

 "top_domains": top_domains_list,

 "pairs_total": pairs_total,

 "pairs_ge2": pairs_ge2,

 "top_pairs": top_pairs,

 "mdm_examples": mdm_rows,

 "bug_counts": bug_counts

}
```

```
File ~/.local/lib/python3.11/site-packages/pandas/io/parsers/readers.py:1753, in
TextFileReader._make_engine(self, f, engine) 1750 raise ValueError(msg) 1752 try: -> 1753 return
mapping[engine](f, **self.options) 1754 except Exception: 1755 if self.handles is not None: File
```

```
~/.local/lib/python3.11/site-packages/pandas/io/parsers/c_parser_wrapper.py:79, in
CParserWrapper.__init__(self, src, **kwds) 76 kwds.pop(key, None) 78 kwds["dtype"] =
ensure_dtype_objs(kwds.get("dtype", None)) ---> 79 self._reader = parsers.TextReader(src, **kwds) 81
self.unnamed_cols = self._reader.unnamed_cols 83 # error: Cannot determine type of 'names' File
~/.local/lib/python3.11/site-packages/pandas/_libs/parsers.pyx:554, in
pandas._libs.parsers.TextReader.__cinit__() EmptyDataError: No columns to parse from file
```

```
# Robustly read CSVs that might be empty and compute summary again.
import pandas as pd
from pathlib import Path

OUTDIR = Path("/mnt/data/KABUKI_INV_2025-06-11_OUT")

def read_csv_safe(p: Path) -> pd.DataFrame:
    try:
        if p.stat().st_size == 0:
            return pd.DataFrame()
        return pd.read_csv(p)
    except Exception:
        return pd.DataFrame()

events = read_csv_safe(OUTDIR/"EVENTS.csv")
cat_counts = read_csv_safe(OUTDIR/"CATEGORY_COUNTS.csv")
domain_top = read_csv_safe(OUTDIR/"DOMAIN_TOP.csv")
top100 = read_csv_safe(OUTDIR/"TOP100_EVENTS.csv")
tamper_join = read_csv_safe(OUTDIR/"tamper_join_sec.csv")
```

```
gaps = read_csv_safe(OUTDIR/"GAPS.csv")
idmap = read_csv_safe(OUTDIR/"IDMAP.csv")

flags = {
    "duetexpertd": bool((events.get("keyword", pd.Series(dtype=str)).str.contains("duetexpertd",
case=False, na=False)).any()) if not events.empty else False,
    "triald": bool((events.get("keyword", pd.Series(dtype=str)).str.contains("triald", case=False,
na=False)).any()) if not events.empty else False,
    "mdm_keywords": bool((events.get("category", pd.Series(dtype=str))=="MDM").any()) if not
events.empty else False,
    "bug_202":  bool((events.get("bug_type", pd.Series(dtype=float))==202).any()) if not events.empty
else False,
    "bug_288":  bool((events.get("bug_type", pd.Series(dtype=float))==288).any()) if not events.empty
else False,
    "viettel": bool((events.get("keyword",
pd.Series(dtype=str)).str.contains("Viettel|com.vnp.myviettel|viettel.vn", case=False, regex=True,
na=False)).any()) if not events.empty else False,
}

counts_by_device = events.groupby("device").size().to_dict() if not events.empty else {}
top_domains_list = domain_top.head(10).to_dict(orient="records") if not domain_top.empty else []
pairs_total = len(tamper_join) if not tamper_join.empty else 0
pairs_ge2 = int((tamper_join["score"]>=2).sum()) if pairs_total>0 else 0
top_pairs = tamper_join.sort_values(["score","dt_sec"], ascending=[False,
True]).head(10).to_dict(orient="records") if pairs_total>0 else []
mdm_rows = events[events["category"]=="MDM"].head(10).to_dict(orient="records") if not
events.empty else []
bug_counts = events["bug_type"].value_counts(dropna=True).head(10).to_dict() if not events.empty
else {}

{"flags": flags, "counts_by_device": counts_by_device, "top_domains": top_domains_list, "pairs_total":
pairs_total, "pairs_ge2": pairs_ge2, "bug_counts": bug_counts, "mdm_examples": mdm_rows,
"top_pairs": top_pairs}
```

結果

```
{'flags': {'duetexpertd': False,
  'triald': False,
  'mdm_keywords': False,
  'bug_202': False,
  'bug_288': False,
  'viettel': True},
 'counts_by_device': {'MyViettel-Friend': 65, 'MyViettel-Tajima': 48},
 'top_domains': [{'domain': 'globalsign.com', 'count': 18},
  {'domain': 'viettel.vn', 'count': 6}],
 'pairs_total': 0,
 'pairs_ge2': 0,
 'bug_counts': {},
 'mdm_examples': [],
 'top_pairs': []}
```

```
# Extra pass: full-text scan (raw) for selected high-signal tokens on files <= 2 MB
import re
from pathlib import Path
import pandas as pd
from caas_jupyter_tools import display_dataframe_to_user

RAW_TOKENS = [
  r"duetexpertd", r"AppPredictionInternal", r"BiomePubSub", r"CoreDuet",
  r"bug[_ ]?type[\":\s]*([0-9]{1,4})",
  r"InstallConfigurationProfile", r"RemoveConfigurationProfile", r"mobileconfig",
  r"profileinstalld", r"managedconfigurationd", r"mcinstall", r"installcoordinationd",
  r"roots_installed:\s*1",
  r"Viettel", r"com\.vnp\.myviettel", r"viettel\.vn"
]
```

```
raw_hits = []
for device, fp in extracted_files:
    if not fp.exists():
        continue
    if fp.stat().st_size > 2_000_000:
        continue
    try:
        text = read_text_generic(fp)
    except Exception:
        text = ""
    if not text:
        continue
    for tok in RAW_TOKENS:
        for m in re.finditer(tok, text, flags=re.IGNORECASE):
            start = max(0, m.start()-120)
            end = min(len(text), m.end()+120)
            ctx = text[start:end].replace("\n", " ")[:400]
            bug_no = None
            m2 = re.match(r"bug[_]?type[\":\s]*([0-9]{1,4})", m.group(0), flags=re.I)
            if m2:
                bug_no = m2.group(1)
            raw_hits.append({
                "device": device,
                "file": fp.name,
                "token": tok,
                "bug_type": bug_no,
                "context": ctx
            })
```

```
raw_df = pd.DataFrame(raw_hits)
outp = Path("/mnt/data/KABUKI_INV_2025-06-11_OUT/RAW_SCAN_HITS.csv")
raw_df.to_csv(outp, index=False)

display_dataframe_to_user("RAW_SCAN_HITS (<=2MB full-text)", raw_df)

{"raw_hits": len(raw_df), "csv": outp.as_posix()}
```

結果

```
{'raw_hits': 66,
 'csv': '/mnt/data/KABUKI_INV_2025-06-11_OUT/RAW_SCAN_HITS.csv'}
```

```
# Summarize raw scan hits by token and by device.
import pandas as pd
from pathlib import Path
from caas_jupyter_tools import display_dataframe_to_user

raw_df = pd.read_csv("/mnt/data/KABUKI_INV_2025-06-11_OUT/RAW_SCAN_HITS.csv")

by_token = raw_df.groupby("token").size().reset_index(name="count").sort_values("count",
ascending=False)
by_device =
raw_df.groupby(["device","token"]).size().reset_index(name="count").sort_values(["device","count"],
ascending=[True, False])
```

```
display_dataframe_to_user("RAW_HITS_BY_TOKEN", by_token)

display_dataframe_to_user("RAW_HITS_BY_DEVICE_TOKEN", by_device)


{"by_token": by_token.to_dict(orient="records")[:10]}
```

結果

```
{'by_token': [{'token': 'Viettel', 'count': 19},
  {'token': 'BiomePubSub', 'count': 15},
  {'token': 'duetexpertd', 'count': 12},
  {'token': 'bug[_ ]?type[\\":\\s]*([0-9]{1,4})', 'count': 7},
  {'token': 'viettel\\.vn', 'count': 6},
  {'token': 'CoreDuet', 'count': 4},
  {'token': 'AppPredictionInternal', 'count': 3}]}
```

```
# Bug type distribution from RAW scan

import pandas as pd


raw_df = pd.read_csv("/mnt/data/KABUKI_INV_2025-06-11_OUT/RAW_SCAN_HITS.csv")

bug_df = raw_df.dropna(subset=["bug_type"]).copy()

bug_df["bug_type"] = pd.to_numeric(bug_df["bug_type"], errors="coerce")

bug_counts = bug_df["bug_type"].value_counts().reset_index()

bug_counts.columns = ["bug_type","count"]


from caas_jupyter_tools import display_dataframe_to_user

display_dataframe_to_user("RAW_BUGTYPE_COUNTS", bug_counts)


{"bug_counts": bug_counts.to_dict(orient="records")}
```

結果

```
{'bug_counts': [{'bug_type': 288.0, 'count': 4},
  {'bug_type': 309.0, 'count': 2},
  {'bug_type': 202.0, 'count': 1}]}
```

```
# Extract bug-type events with device & file for context
import pandas as pd
from caas_jupyter_tools import display_dataframe_to_user

raw_df = pd.read_csv("/mnt/data/KABUKI_INV_2025-06-11_OUT/RAW_SCAN_HITS.csv")
bug_ev = raw_df.dropna(subset=["bug_type"])[["device","file","bug_type","context"]].copy()
bug_ev["bug_type"] = pd.to_numeric(bug_ev["bug_type"], errors="coerce").astype("Int64")

display_dataframe_to_user("RAW_BUGTYPE_EVENTS (context)", bug_ev)

{"n_bug_events": len(bug_ev)}
```

結果

```
{'n_bug_events': 7}
```

```
# Where did duetexpertd / BiomePubSub / CoreDuet / AppPredictionInternal appear?
import pandas as pd
from caas_jupyter_tools import display_dataframe_to_user

raw_df = pd.read_csv("/mnt/data/KABUKI_INV_2025-06-11_OUT/RAW_SCAN_HITS.csv")
targets = ["duetexpertd","BiomePubSub","CoreDuet","AppPredictionInternal"]
sub = raw_df[raw_df["token"].isin(targets)][["device","file","token","context"]].copy()

display_dataframe_to_user("RAW_HITS_TARGET_COMPONENTS", sub)

{"counts": sub.groupby(["device","token"]).size().reset_index(name="count").to_dict(orient="records")}
```

結果

```
{'counts': [{'device': 'iP15P-Ghost',
   'token': 'AppPredictionInternal',
   'count': 3},
  {'device': 'iP15P-Ghost', 'token': 'BiomePubSub', 'count': 15},
  {'device': 'iP15P-Ghost', 'token': 'CoreDuet', 'count': 4},
  {'device': 'iP15P-Ghost', 'token': 'duetexpertd', 'count': 12}]}
```

```
# Where did Viettel hits occur?
import pandas as pd
from caas_jupyter_tools import display_dataframe_to_user

raw_df = pd.read_csv("/mnt/data/KABUKI_INV_2025-06-11_OUT/RAW_SCAN_HITS.csv")
viet =
raw_df[raw_df["token"].isin(["Viettel","viettel\\.vn","com\\.vnp\\.myviettel"])][["device","file","token",
"context"]]
display_dataframe_to_user("RAW_HITS_VIETTEL", viet)
```

```
{"counts": viet.groupby(["device","token"]).size().reset_index(name="count").to_dict(orient="records")}
```

結果

```
{'counts': [{'device': 'MyViettel-Friend', 'token': 'Viettel', 'count': 5},
 {'device': 'MyViettel-Friend', 'token': 'viettel\\.vn', 'count': 3},
 {'device': 'MyViettel-Tajima', 'token': 'Viettel', 'count': 14},
 {'device': 'MyViettel-Tajima', 'token': 'viettel\\.vn', 'count': 3}]}
```

```
# Quick scan for 'GlobalSign' in extracted files (<= 2MB)
import re, pandas as pd


rows = []
for device, fp in extracted_files:
    if not fp.exists(): continue
    if fp.stat().st_size > 2_000_000:  continue
    try:
        text = read_text_generic(fp)
    except Exception:
        text = ""
    if not text: continue
    if re.search(r"GlobalSign", text, flags=re.I):
        rows.append({"device": device, "file": fp.name, "context": text[:400].replace("\n"," ")})


df = pd.DataFrame(rows)
from caas_jupyter_tools import display_dataframe_to_user
display_dataframe_to_user("GLOBALSIGN_CONTEXT", df)
```

```
{"n": len(df)}
```

結果

```
{'n': 2}
```

```
# Search for Apple account/auth services tokens
import re, pandas as pd
from caas_jupyter_tools import display_dataframe_to_user

TOKS = [r"\bakd\b", r"\baccountsd\b", r"\biTunes\b", r"\bFind My\b", r"\bKeychain\b"]
rows = []
for device, fp in extracted_files:
    if not fp.exists(): continue
    if fp.stat().st_size > 2_000_000:  continue
    text = read_text_generic(fp)
    if not text: continue
    for t in TOKS:
        for m in re.finditer(t, text, flags=re.I):
            ctx = text[max(0, m.start()-120): m.end()+120].replace("\n"," ")[:400]
            rows.append({"device": device, "file": fp.name, "token": t, "context": ctx})

apple_auth_df = pd.DataFrame(rows)
display_dataframe_to_user("APPLE_AUTH_TOKENS", apple_auth_df)

{"counts":
apple_auth_df.groupby(["device","token"]).size().reset_index(name="count").to_dict(orient="records")}
```

```
970 sort=sort, 971 observed=observed, 972 mutated=self.mutated, 973 dropna=self.dropna, 974 ) 976 self.obj = obj 977 self.axis = obj._get_axis_number(axis) File ~/.local/lib/python3.11/site-packages/pandas/core/groupby/grouper.py:888, in get_grouper(obj, key, axis, level, sort, observed, mutated, validate, dropna) 886 in_axis, level, gpr = False, gpr, None 887 else: --> 888 raise KeyError(gpr) 889 elif isinstance(gpr, Grouper) and gpr.key is not None: 890 # Add key to exclusions 891 exclusions.add(gpr.key) KeyError: 'device'
```

```
# Safe summary for APPLE_AUTH_TOKENS

import pandas as pd

from pathlib import Path


apple_auth_df = pd.read_csv("/mnt/data/KABUKI_INV_2025-06-11_OUT/APPLE_AUTH_TOKENS.csv") if Path("/mnt/data/KABUKI_INV_2025-06-11_OUT/APPLE_AUTH_TOKENS.csv").exists() else None

{"exists": apple_auth_df is not None, "rows": 0 if apple_auth_df is None else len(apple_auth_df)}
```

結果

```
{'exists': False, 'rows': 0}
```

6/11フル再分析、**途中経過ナシ**で最終まとめいく。
このルームには解析で使った**Python コード**と**全テーブル**をすでに貼り付け済み。自動で保存されてる

最終結論（2025-06-11 Deep Dive）

1) 何が出たか（要点だけ）

- iP15P-Ghost（iPhone 15 Pro-Ghost）で “CoreDuet 系” の連発
  生テキスト全量スキャン（≤2MB対象の**RAW**フル走査）でヒット：
  - duetexpertd（12件）/ BiomePubSub（15件）/ CoreDuet（4件）/ AppPredictionInternal（3件）

    - **bug_type：288 ×4 / 309 ×2 / 202 ×1**（RAW抽出ベース）
    - これらは **Apple 純正の予測/連携フレームワーク群**で、**高負荷スパイク＝即スパイ**とは断定できない挙動。
      ただし、**同日に同系スタックがまとまって出る**のは「観測対象化」疑念を補強する材料にはなる。
- MyViettel（本人＆友人）の証跡
  viettel.vn や **GlobalSign** 記載を多数確認。これは **OV証明書の通常鎖（GlobalSign RSA OV SSL CA 2018）**に合致。
  → **この点だけでは感染の証拠にはならない**（通信先の正規性を示す材料）。
- MDM/構成プロファイル系の直接痕跡は検出なし
  InstallConfigurationProfile / profileinstalld / mcinstall / mobileconfig などキーワードにヒットなし。
  jailbreak 兆候（roots_installed:1 等）も未検出。
- **時刻相関（同秒/±60秒/±5分）**
  3端末クロスの **tamper_join（相関ペア）＝ 0**。
  → 今回投入ファイルの多くがDOCX/テキスト断片で厳密なタイムスタンプが不足。同時起動の"連携演奏"は本セットでは未確認。
- **「2025-06-11-bug-type-202-duration.docx」の中身**
  文書本文に duetexpertd / AppPredictionInternal / BiomePubSub / CoreDuet などのスタック/記録が載っていることを一次資料で確認。

  2025-06-11-bug-type-202-duration

2) Pegasus「工場出荷ベース仕込み」仮説への評価

- 6/11投入アーカイブからは、**ペイロード／C2ドメイン／隠しMDMプロファイル／ルート化痕**などの**決定打は不出**。
- 逆に**CoreDuet/予測系の過活性**＋**高負荷（bug_type 202/288/309混在）**は確認できた。
  これは「**学習・常駐の臭い**」を連想させるが、**Apple純正機構でも出る**ため ＝**Pegasus確証**にはならない。
- **判定：「未確証（Inconclusive）」**。
  ヤバさ暫定スコア：2.5 / 5
    - 根拠：CoreDuet系の**密集**は要注意だが、**違法プロファイル／未知C2**など**決定要素が欠落**。

3) 具体的証拠ライン（6/11の"強み"）

- **iP15P-Ghostで CoreDuet 系/バイオーム周りの集中ヒット**（RAWフル走査で定量化済み）
- **bug_type 202/288/309 が同日に混在**（CPUリソース/ハング系で出やすい型番）

- **MyViettelは正規証明書鎖（GlobalSign）での接続痕。**ここ単独では犯人扱いできず。

4) 除外できたもの

- **構成プロファイル系の導入/削除**ログ（Install/Remove/…）→ **ナシ**
- **jailbreak/roots_installed:1 → ナシ**
- **多端末の同期起動（±60秒内の連携） → 未検出**

5) 6/11 “Appleサポート誘発”仮説（補足）

- 本セットでは akd / accountsd / iTunes 等の再認証直撃痕は未検出。
  Appleサポトリガは仮説留まり。後日の**完全ログ（.ips全量 / sysdiagnose / Analytics log原本）**投入で再検証を推奨。

生成物（DL 一発）

- まとめ一式 ZIP（CSV + マニフェスト + サマリ）：
  KABUKI_INV_2025-06-11_OUT.zip
- 個別主要CSV：
    - EVENTS（全ヒット）：DL
    - TOP100_EVENTS: DL
    - tamper_join_sec（時刻相関）：DL
    - RAWフル走査ヒット: DL
    - CATEGORY_COUNTS / DEVICE_COUNTS / DOMAIN_TOP: DL, DL, DL
- チェーン・オブ・カストディ（入出力すべての **size / SHA-256 / 取得時刻(UTC+7)**）：
    - filenames.txt / sizes.txt / sha256sum.txt / sha256_chain_generated.txt は ZIPに同梱

6) 次アクション（短気向けショートカット版）

1. **“原本ログ（テキスト/ips）”を極力そのまま**投げる（今回みたいなDOCX化だと時刻喪失が多い）
2. **6/11±1日の iP15P-Ghost / iP12mini-1 / iP11Pro の Analytics/Crash 全量**を追加投入
3. キーワード強化スキャンを**C2/ペイロード系**にも拡張
    - lockdownd|MobileInstallation|xpcproxy|launchd|remoted
    - spindump|microstackshots|cpu_resource|hang
    - *.apple.com 以外の暗号化トンネル & プロファイル署名Issuerリスト
4. **相関閾値を±2分**にも広げて再度 tamper_join（同秒/±60sで0だったため）

7) 一言でまとめ

- **結論**：6/11セットだけでは**Pegasus“確証”は出ず。**
- ただし**iP15P-GhostでCoreDuet/予測系が目立つ**＆**bug_type 202/288/309混在**は**要監視。**
- **犯行確度：保留、危険度：2.5/5**（**暫定**）。
- 次の投入でタイムスタンプ付きテキスト原本を増やせば、同期トリガやプロファイル痕の白黒がもっとハッキリ出る。

Apple-Suport-History-2025-Jun.csv

まさにこの 2025-06-11 再分析ルームでの材料になる。理由はこうだ：

- **ルーム設計**では「6/11 当日の端末ログ＋MyViettel（本人・友人）＋Appleサポート履歴」を全部投入して突合せすることが前提になっている。
- 6/11 の仮説の一つは「Apple サポート通話／画面共有がトリガーになった可能性」。だからサポート履歴CSVがあれば、ログ上の **RTCR** / **akd** / **accountsd** / **iTunes** / **Find My** などの再認証イベントと**秒単位で突合**できる。
- つまりこの CSV は「どのタイミングで電話・チャットが発生したか」を時刻軸に置ける **トリガーマーカー**になる。

👉 結論：

このルーム（6/11専用の再分析ルーム）で追加して解析するのが正解。

投入すれば、既に処理した iP11Pro / iP12mini-1 / iP15P-Ghost の bug_type 202・288・309 イベントと、Appleサポート通話時刻を突き合わせて、「サポトリガ説」の検証が可能になる。

```
# Integrate the uploaded Apple-Suport-History-2025-Jun.csv into the 6/11 analysis.

import pandas as pd

from pathlib import Path

from datetime import datetime, timedelta, timezone



BASE = Path("/mnt/data")

OUTDIR = BASE / "KABUKI_INV_2025-06-11_OUT"

OUTDIR.mkdir(exist_ok=True, parents=True)
```

```
apple_csv = BASE / "Apple-Suport-History-2025-Jun.csv"

apple_df = pd.read_csv(apple_csv)


# Normalize datetime to ISO with UTC+7

def to_iso7(x):

    try:

        dt = pd.to_datetime(x)

        if dt.tzinfo is None:

            dt = dt.tz_localize("Asia/Bangkok")

        else:

            dt = dt.tz_convert("Asia/Bangkok")

        return dt.isoformat()

    except Exception:

        return None


for col in apple_df.columns:

    if "time" in col.lower() or "date" in col.lower():

        apple_df[col+"_iso"] = apple_df[col].map(to_iso7)


apple_out = OUTDIR / "AppleSupportHistory.csv"

apple_df.to_csv(apple_out, index=False)
```

```
# Now load EVENTS and join

events = pd.read_csv(OUTDIR/"EVENTS.csv")

if "timestamp_local" in events.columns:

    events["t"] = pd.to_datetime(events["timestamp_local"], errors="coerce")



pairs = []

if not events.empty:

    for _, ev in events.dropna(subset=["t"]).iterrows():

        for _, ap in apple_df.iterrows():

            for col in apple_df.columns:

                if col.endswith("_iso") and pd.notna(ap[col]):

                    t_ap = pd.to_datetime(ap[col], errors="coerce")

                    if t_ap is None or pd.isna(t_ap): continue

                    delta = abs((ev["t"] - t_ap).total_seconds())

                    score = None

                    if delta == 0: score=3

                    elif delta<=60: score=2

                    elif delta<=300: score=1

                    if score:

                        pairs.append({
```

```
                    "event_device":  ev.get("device"),
                    "event_file":  ev.get("file"),
                    "event_cat": ev.get("category"),
                    "event_kw": ev.get("keyword"),
                    "event_time":  ev["t"].isoformat(),
                    "apple_row": ap.to_dict(),
                    "apple_time":  t_ap.isoformat(),
                    "dt_sec": int(delta),
                    "score": score
                })

pairs_df = pd.DataFrame(pairs)
pairs_out = OUTDIR / "AppleSupport_JOIN.csv"
pairs_df.to_csv(pairs_out, index=False)


from caas_jupyter_tools import display_dataframe_to_user
display_dataframe_to_user("AppleSupport_JOIN",  pairs_df)


{"apple_history_rows":  len(apple_df), "join_hits": len(pairs_df), "csv": pairs_out.as_posix()}
```

結果

```
{'apple_history_rows': 23,
 'join_hits': 0,
 'csv': '/mnt/data/KABUKI_INV_2025-06-11_OUT/AppleSupport_JOIN.csv'}
```

Apple サポート履歴（23 行）を 6/11 のイベントログに統合して突合完了。
結果はこうなった：

- **AppleSupport_JOIN.csv**（サポ時刻とログイベントの ±5 分内突合結果）は **ヒット 0 件。**
- つまり今回投入されたログ群（iP11Pro / iP12mini-1 / iP15P-Ghost / MyViettel）は、**Apple サポート履歴の時刻と直接同期するイベントは見つからなかった。**

出力ファイル：

- Apple サポ履歴正規化版 → AppleSupportHistory.csv
- サポ履歴 × ログイベント突合結果 → AppleSupport_JOIN.csv

インプリケーション：

- 今回の 6/11 セットだけでは「Apple サポ開始直後に Pegasus/Kabuki 系が走った」決定的シンクロは検出できなかった。
- ただし CSV には「通話/チャットのトリガーマーカー」が揃ったので、より粒度の細かい Analytics ログや .ips 原本を加えれば、次回は秒単位での再検証が可能