

2025-Jun-10-Reanalysis-01

(Gear-First)

```
# -*- coding: utf-8 -*-

import os, zipfile, hashlib, plistlib, io, re, json, mimetypes, sys, textwrap

from datetime import datetime, timezone

from pathlib import Path

import pandas as pd


# IMPORTANT: All paths are under /mnt/data (sandboxed)

BASE = Path("/mnt/data").resolve()

WORK = BASE / "_work_2025-06-10"

EXTRACT = WORK / "unzipped"

OUT = WORK / "out"

for p in [WORK, EXTRACT, OUT]:

    p.mkdir(parents=True, exist_ok=True)


# -----

# Helper functions

# -----

def sha256_of_path(path: Path, chunk=1024*1024):

    h = hashlib.sha256()

    with open(path, "rb") as f:

        while True:
```

```

        b = f.read(chunk)

        if not b: break

        h.update(b)

    return h.hexdigest()


def safe_mime_guess(p: Path):

    # crude guess; do not import external libs

    m, _ = mimetypes.guess_type(str(p))

    return m or "application/octet-stream"


def list_uploaded_files():

    # Only consider the files the user said they uploaded (and developer tool listed)

    candidates = [

        "log-bb-2025-06-10-stats.plist",

        "README_victim_map.zip",

        "identifiers_dedup.zip",

        "bug_type_298_2025-06-10_combined.zip",

        "H.zip",

        "Tajima.zip",

        "victim_report_template.zip",

        "victim_report_template (2).zip",

        "REPORT_extract_2025-06-10 2.zip",

        "REPORT_extract_2025-06-10 2 (2).zip",

    ]

    files = []

```

```

for name in candidates:

    path = BASE / name

    if path.exists():

        files.append(path)

return files


def unzip_all(zip: Path, dest_dir: Path):

    """Extract zip with folder prefix to avoid collisions."""

    prefix_dir = dest_dir / zip.stem

    prefix_dir.mkdir(parents=True, exist_ok=True)

    extracted = []

    try:

        with zipfile.ZipFile(zip, 'r') as zf:

            for info in zf.infolist():

                # Skip directories

                if info.is_dir():

                    continue

                # Sanitize inner path

                inner_rel = Path(info.filename)

                safe_rel = Path(*[p for p in inner_rel.parts if p not in ("..", "/", "\x00")])

                target = prefix_dir / safe_rel

                target.parent.mkdir(parents=True, exist_ok=True)

                with zf.open(info, 'r') as src, open(target, 'wb') as dst:

                    dst.write(src.read())

                extracted.append(target)

```

```

except zipfile.BadZipFile:

    # not a valid zip; ignore

    pass

return extracted


def try_parse_plist(path: Path):

    try:

        with open(path, 'rb') as f:

            data = plistlib.load(f)

            return data

    except Exception as e:

        return {"__error__": str(e)}


def read_text_best_effort(p: Path, max_bytes=1024*1024*4):

    try:

        with open(p, "rb") as f:

            raw = f.read(max_bytes)

            # try several encodings

            for enc in ("utf-8", "utf-8-sig", "latin-1", "utf-16", "utf-16le", "utf-16be"):

                try:

                    return raw.decode(enc, errors="ignore")

                except Exception:

                    continue

    except Exception:

        pass

```

```

return ""

def find_keywords_in_file(path: Path, keywords):

    text = read_text_best_effort(path)

    hits = []

    if not text:

        return hits

    for kw in keywords:

        # case-insensitive search; collect context lines

        pattern = re.compile(re.escape(kw), re.IGNORECASE)

        for m in pattern.finditer(text):

            # extract surrounding line or ~120 chars context

            start = max(0, m.start() - 80)

            end = min(len(text), m.end() + 80)

            snippet = text[start:end].replace("\n", " ")

            hits.append((kw, snippet))

    return hits


def parse_ips_quick(text):

    """

    Heuristic parse for .ips content; extract JSON-like lines or keys.

    Returns dict list with common fields when observed.

    """

    events = []

    # First, try to find JSON blobs (lines with braces). Fallback to regex for key-value.

```

```

# Extract 'bug_type', 'timestamp', 'incident_id', 'os_version', 'procname'

# Many .ips are plist-like; we attempt regex.

bug_type = None

ts = None

incident_id = None

os_version = None

procname = None


# Simple regexes

r_bug = re.compile(r""?bug_type"?%s*[:=]%s*"([0-9A-Za-z_]+)"?')

r_ts = re.compile(r""?timestamp"?%s*[:=]%s*"([^\"]+)"")

r_inc = re.compile(r""?incident_id"?%s*[:=]%s*"([0-9A-Fa-f\%-]+)"")

r_osv = re.compile(r""?os_version"?%s*[:=]%s*"([^\"]+)"")

r_proc = re.compile(r""?procname"?%s*[:=]%s*"([^\"]+)"")

r_parent = re.compile(r""?parent_process"?%s*[:=]%s*"([^\"]+)"")


d = {}

m = r_bug.search(text);

if m: d["bug_type"] = m.group(1)

m = r_ts.search(text);

if m: d["timestamp"] = m.group(1)

m = r_inc.search(text);

if m: d["incident_id"] = m.group(1)

m = r_osv.search(text);

if m: d["os_version"] = m.group(1)

```

```

m = r_proc.search(text);

if m: d["procname"] = m.group(1)

m = r_parent.search(text);

if m: d["parent_process"] = m.group(1)


# If at least one key found, return as single record

if d:

    events.append(d)

return events


# -----

# Inventory & Hashing (top-level)

# -----

uploaded = list_uploaded_files()

manifest_rows = []

for p in uploaded:

    try:

        h = sha256_of_path(p)

        manifest_rows.append({

            "level": "top",

            "parent": "",

            "path": str(p),

            "name": p.name,

            "size_bytes": p.stat().st_size,

            "sha256": h,

```

```

        "mime_guess": safe_mime_guess(p),
    })

except Exception as e:

    manifest_rows.append({

        "level": "top",

        "parent": "",

        "path": str(p),

        "name": p.name,

        "size_bytes": None,

        "sha256": f"__ERROR__: {e}",

        "mime_guess": "unknown",

    })

df_manifest_top = pd.DataFrame(manifest_rows).sort_values("name")

top_csv = OUT / "manifest_top_files.csv"

df_manifest_top.to_csv(top_csv, index=False)

# -----

# Extract zips & Inventory inner files

# -----

inner_rows = []

for p in uploaded:

    if p.suffix.lower() == ".zip":

        extracted = unzip_all(p, EXTRACT)

        for q in extracted:

```



```

try:
    hq = sha256_of_path(q)
except Exception as e:
    hq = f"__ERROR__: {e}"

inner_rows.append({
    "level": "inner",
    "parent": p.name,
    "inner_relpath": str(q.relative_to(EXTRACT / p.stem)),
    "abs_path": str(q),
    "name": q.name,
    "size_bytes": q.stat().st_size if q.exists() else None,
    "sha256": hq,
    "mime_guess": safe_mime_guess(q),
})

df_inner = pd.DataFrame(inner_rows)

if not df_inner.empty:
    df_inner = df_inner.sort_values(["parent", "inner_relpath"])

inner_csv = OUT / "manifest_inner_files.csv"

df_inner.to_csv(inner_csv, index=False)

# Dedup clusters by sha256 (top + inner)

df_all = pd.concat([df_manifest_top.rename(columns={"path": "abs_path"}), df_inner],
                    ignore_index=True, sort=False)

```

```

dup_groups = df_all.groupby("sha256").size().reset_index(name="count").sort_values("count",
ascending=False)

dup_csv = OUT / "dedup_clusters.csv"

dup_groups.to_csv(dup_csv, index=False)


# -----
# Parse plist if present
# -----

plist_path = BASE / "log-bb-2025-06-10-stats.plist"

plist_rows = []

if plist_path.exists():
    parsed = try_parse_plist(plist_path)

    def flatten(prefix, obj):
        if isinstance(obj, dict):
            for k,v in obj.items():
                yield from flatten(f"{prefix}.{k}" if prefix else str(k), v)

        elif isinstance(obj, list):
            for i, v in enumerate(obj):
                yield from flatten(f"{prefix}[{i}]", v)

        else:
            yield prefix, obj

    for k,v in flatten("", parsed):
        plist_rows.append({"key": k, "value": v})

df_plist = pd.DataFrame(plist_rows)

plist_csv = OUT / "plist_kv_flat.csv"

```

```

df_plist.to_csv(plist_csv, index=False)

# -----

# Keyword scanning (MDM/Pegasus/Viettel/Support)

# -----

# Derived from user's fixed lists + common indicators

MDM_KEYWORDS = [

    "InstallConfigurationProfile", "RemoveConfigurationProfile", "mobileconfig",

    "MCProfile", "MCProfileTrustLevel", "MCXPreferences",

    "mdmd", "managedconfigurationd", "profileinstalld", "installcoordinationd", "mcinstall",

    "BackgroundShortcutRunner",

]

APPLE_FRAMEWORK_KEYS = [

    "triald", "SiriSearchFeedback", "searchd", "suggested", "ScreenTimeAgent",

    "assetsd", "JetsamEvent", "Stackshot", "usageClientld", "signpost_reporter",

    "power session", "duetexpertd", "sharingd",

]

PEGASUS_KEYS = [

    "Pegasus", "NSO", "BlastDoor", "FORCEDENTRY", "Kismet",

]

VIETTEL_KEYS = [

```

```

    "com.vnp.myviettel", "myviettel", "Viettel", "viettel.vn", "GlobalSign RSA OV SSL CA 2018",
    "*.viettel.vn"
]

```

```

APPLE_SUPPORT_KEYS = [
    "Apple Support", "Advisor", "Case ID", "support.apple.com", "chat adviser", "Chat Adviser"
]

```

```

ALL_KEYS = MDM_KEYWORDS + APPLE_FRAMEWORK_KEYS + PEGASUS_KEYS +
VIETTEL_KEYS + APPLE_SUPPORT_KEYS

```

```

# We'll scan only a subset of inner files that are text-like or .ips/.log/.txt/.json/.csv/.plist/.der

```

```

TEXT_EXTS =
{".txt", ".log", ".ips", ".json", ".csv", ".plist", ".xml", ".html", ".htm", ".md", ".rtf", ".strings", ".ca", ".session", ".der",
 ".cer", ".pem"}

```

```

keyword_rows = []

```

```

scan_targets = []

```

```

# Prioritize inner files if available; otherwise, scan top files

```

```

if not df_inner.empty:

```

```

    scan_targets = [Path(p) for p in df_inner["abs_path"].tolist() if Path(p).exists()]

```

```

else:

```

```

    scan_targets = uploaded

```

```

# Restrict to plausible text-ish files

```

```

targets_filtered = []

```

```

for p in scan_targets:

    ext = p.suffix.lower()

    if ext in TEXT_EXTS:

        targets_filtered.append(p)

    else:

        # small heuristic: treat small unknown files as text if < 2MB

        try:

            if p.stat().st_size <= 2*1024*1024:

                targets_filtered.append(p)

        except Exception:

            pass


# Scan for keywords

for p in targets_filtered:

    try:

        hits = find_keywords_in_file(p, ALL_KEYS)

        for kw, snippet in hits:

            keyword_rows.append({

                "file": str(p),

                "keyword": kw,

                "snippet": snippet[:200],

            })

    except Exception as e:

        keyword_rows.append({

            "file": str(p), "keyword": "__ERROR__", "snippet": str(e)

```

```

    })

df_hits = pd.DataFrame(keyword_rows)

hits_csv = OUT / "keyword_hits.csv"

df_hits.to_csv(hits_csv, index=False)


# Summary counts by keyword

df_kw_summary = pd.DataFrame()

if not df_hits.empty:

    df_kw_summary =
df_hits.groupby("keyword").size().reset_index(name="hit_count").sort_values("hit_count",
ascending=False)

kw_summary_csv = OUT / "keyword_summary.csv"

df_kw_summary.to_csv(kw_summary_csv, index=False)


# -----

# Focus parse for bug_type_298 events on 2025-06-10

# -----

ips_rows = []

if not df_inner.empty:

    # Filter likely ips files from the bug_type zip folder or generally with '298' in name

    for _, row in df_inner.iterrows():

        ap = Path(row.get("abs_path", ""))

        if not ap.exists():

            continue

```

```

name_l = ap.name.lower()

if ".ips" in name_l or name_l.endswith(".log") or "bug_type_298" in name_l:

    text = read_text_best_effort(ap)

    if not text:

        continue

    parsed = parse_ips_quick(text)

    for d in parsed:

        d["source_file"] = str(ap)

        ips_rows.append(d)

df_ips = pd.DataFrame(ips_rows)

ips_csv = OUT / "bug_type_298_parsed.csv"

df_ips.to_csv(ips_csv, index=False)


# -----

# MyViettel presence check (installation traces)

# -----

viettel_rows = []

for p in targets_filtered:

    text = read_text_best_effort(p, max_bytes=1024*1024*2)

    if not text:

        continue

    if re.search(r"com\.vnp\.myviettel", text, re.IGNORECASE):

        viettel_rows.append({"file": str(p), "signal": "com.vnp.myviettel"})

    if re.search(r"viettel\.vn", text, re.IGNORECASE):

        viettel_rows.append({"file": str(p), "signal": "viettel.vn"})

```

```

if re.search(r"GlobalSign RSA OV SSL CA 2018", text, re.IGNORECASE):

    viettel_rows.append({"file": str(p), "signal": "GlobalSign RSA OV SSL CA 2018"})

df_viettel = pd.DataFrame(viettel_rows).drop_duplicates()

viettel_csv = OUT / "viettel_signals.csv"

df_viettel.to_csv(viettel_csv, index=False)


# -----

# Device/timestamp heuristics for 2025-06-10 files

# -----

# Extract timestamps from filenames if present, build a day timeline.

time_pat = re.compile(r"(20\d{2}-\d{2}-\d{2}[ _-]??\d{2}:\d{2}:\d{2})")

timeline_rows = []

for p in scan_targets:

    s = p.name

    # try to find a date

    m = time_pat.search(s.replace(".", "-").replace("/", "-"))

    if m:

        raw = m.group(1).replace("_", " ").replace("-", "-").replace("T", " ")

        # normalize

        raw = raw.replace(" ", " ")

        timeline_rows.append({"file": str(p), "from_name_ts": raw})

df_timeline = pd.DataFrame(timeline_rows)

timeline_csv = OUT / "timeline_from_names.csv"

df_timeline.to_csv(timeline_csv, index=False)

```



```

# -----

# Save combined deduped manifest & quick indicators

# -----

# Tag any file that hit MDM/PEGASUS/Viettel/AppleSupport keys

tag_map = {}

for _, r in df_hits.iterrows():

    tag_map.setdefault(r["file"], set()).add(r["keyword"])


tags_rows = []

for fp, tags in tag_map.items():

    tags_rows.append({"file": fp, "tags": ";".join(sorted(tags))})

df_tags = pd.DataFrame(tags_rows)

tags_csv = OUT / "files_with_tags.csv"

df_tags.to_csv(tags_csv, index=False)


# -----

# Export "final bundle" ZIP for download

# -----

bundle_zip = BASE / "KABUKI-INV_2025-06-10_analysis_bundle.zip"

with zipfile.ZipFile(bundle_zip, "w", compression=zipfile.ZIP_DEFLATED) as z:

    for f in [top_csv, inner_csv, dup_csv, plist_csv, hits_csv, kw_summary_csv, ips_csv, viettel_csv,
timeline_csv, tags_csv]:

        if f.exists():

            z.write(f, arcname=f.name)

```

```

# -----

# Display key tables to the user (visible)

# -----

from caas_jupyter_tools import display_dataframe_to_user


# 1) Top-level manifest

display_dataframe_to_user("Top-level files manifest (2025-06-10)", df_manifest_top)


# 2) Inner files manifest (head only to keep UI light)

if not df_inner.empty:

    display_dataframe_to_user("Inner files manifest (first 500 rows)", df_inner.head(500))


# 3) Keyword summary

if not df_kw_summary.empty:

    display_dataframe_to_user("Keyword hits summary", df_kw_summary)


# 4) Detailed keyword hits (first 500)

if not df_hits.empty:

    display_dataframe_to_user("Keyword hits (first 500 rows)", df_hits.head(500))


# 5) MyViettel signals

display_dataframe_to_user("MyViettel-related signals", df_viettel)


# 6) Parsed bug_type_298 events (quick)

display_dataframe_to_user("bug_type 298 quick parse (heuristic)", df_ips)

```

```
# 7) Files with tags (which files had any indicator hits)
```

```
display_dataframe_to_user("Files with indicator tags", df_tags)
```

```
# 8) Flattened plist key-values (if any)
```

```
display_dataframe_to_user("Flattened plist key-values", df_plist)
```

```
# 9) Timeline extracted from filenames
```

```
display_dataframe_to_user("Timeline hints extracted from filenames", df_timeline)
```

```
print("Outputs saved to:", str(OUT))
```

```
print("Bundle ZIP:", str(bundle_zip))
```

STDOUT/STDERR

```
Outputs saved to: /mnt/data/_work_2025-06-10/out
```

```
Bundle ZIP: /mnt/data/KABUKI-INV_2025-06-10_analysis_bundle.zip
```

```
# Compute high-level metrics from the generated CSVs for the final summary
```

```
import pandas as pd
```

```
from pathlib import Path
```

```
from collections import Counter
```

```

BASE = Path("/mnt/data")

OUT = BASE / "_work_2025-06-10" / "out"

def read_csv(path):

    try:

        return pd.read_csv(path)

    except Exception:

        return pd.DataFrame()

df_manifest_top = read_csv(OUT / "manifest_top_files.csv")

df_inner = read_csv(OUT / "manifest_inner_files.csv")

df_dup = read_csv(OUT / "dedup_clusters.csv")

df_hits = read_csv(OUT / "keyword_hits.csv")

df_kw_summary = read_csv(OUT / "keyword_summary.csv")

df_ips = read_csv(OUT / "bug_type_298_parsed.csv")

df_viettel = read_csv(OUT / "viettel_signals.csv")

df_plist = read_csv(OUT / "plist_kv_flat.csv")

df_tags = read_csv(OUT / "files_with_tags.csv")

df_timeline = read_csv(OUT / "timeline_from_names.csv")

metrics = {}

# Top-level file counts

metrics["top_files_count"] = len(df_manifest_top)

```

```

metrics["top_zip_count"] = int(df_manifest_top["name"].str.endswith(".zip").sum())

metrics["top_total_size_bytes"] = int(df_manifest_top["size_bytes"].fillna(0).sum())


# Inner files count

metrics["inner_files_count"] = len(df_inner)


# Duplicates (hash count > 1)

if not df_dup.empty:

    metrics["dup_clusters"] = int((df_dup["count"] > 1).sum())

    metrics["max_dup_count"] = int(df_dup["count"].max())

else:

    metrics["dup_clusters"] = 0

    metrics["max_dup_count"] = 1


# Keyword hits by category

MDM_KEYWORDS = [

    "InstallConfigurationProfile", "RemoveConfigurationProfile", "mobileconfig",

    "MCProfile", "MCProfileTrustLevel", "MCXPreferences",

    "mdmd", "managedconfigurationd", "profileinstalld", "installcoordinationd", "mcinstall",

    "BackgroundShortcutRunner",

]

APPLE_FRAMEWORK_KEYS = [

    "triald", "SiriSearchFeedback", "searchd", "suggested", "ScreenTimeAgent",

    "assetsd", "JetsamEvent", "Stackshot", "usageClientd", "signpost_reporter",

    "power session", "duetexpertd", "sharingd",

```

```

]

PEGASUS_KEYS = [

    "Pegasus", "NSO", "BlastDoor", "FORCEDENTRY", "Kismet",

]

VIETTEL_KEYS = [

    "com.vnp.myviettel", "myviettel", "Viettel", "viettel.vn", "GlobalSign RSA OV SSL CA 2018",
    ".*viettel.vn"

]

APPLE_SUPPORT_KEYS = [

    "Apple Support", "Advisor", "Case ID", "support.apple.com", "chat adviser", "Chat Adviser"

]

def hit_count_for(keys):

    if df_hits.empty: return 0

    return int(df_hits["keyword"].isin(keys).sum())

metrics["hits_mdm"] = hit_count_for(MDM_KEYWORDS)

metrics["hits_framework"] = hit_count_for(APPLE_FRAMEWORK_KEYS)

metrics["hits_pegasus"] = hit_count_for(PEGASUS_KEYS)

metrics["hits_viettel"] = hit_count_for(VIETTEL_KEYS)

metrics["hits_support"] = hit_count_for(APPLE_SUPPORT_KEYS)

# Unique files that had Viettel signals

metrics["viettel_files"] = int(df_viettel["file"].nunique()) if not df_viettel.empty else 0

metrics["viettel_signals"] = int(len(df_viettel)) if not df_viettel.empty else 0

```

```

# IPS parsed stats

metrics["ips_events"] = int(len(df_ips))

metrics["ips_unique_procnames"] = int(df_ips["procname"].nunique()) if ("procname" in df_ips and
not df_ips.empty) else 0

metrics["ips_bugtype_298_rows"] = int((df_ips.get("bug_type",
pd.Series()).astype(str).str.contains("298", na=False).sum()) if not df_ips.empty else 0

# Plist keys

metrics["plist_keys"] = int(len(df_plist))

# Files with tags (any indicator)

metrics["tagged_files"] = int(df_tags["file"].nunique()) if not df_tags.empty else 0

# Timeline hints

metrics["timeline_entries"] = int(len(df_timeline))

# Top 10 keywords by hits

top10 = []

if not df_kw_summary.empty:

    temp = df_kw_summary.sort_values("hit_count", ascending=False).head(10)

    top10 = list(zip(temp["keyword"].astype(str).tolist(), temp["hit_count"].astype(int).tolist()))

metrics["top10_keywords"] = top10

# Write metrics to JSON for reference

import json

```

```

metrics_json_path = OUT / "summary_metrics.json"

with open(metrics_json_path, "w", encoding="utf-8") as f:

    json.dump(metrics, f, ensure_ascii=False, indent=2)

# Show the metrics table

metrics_df = pd.DataFrame([metrics])

from caas_jupyter_tools import display_dataframe_to_user

display_dataframe_to_user("High-level metrics for 2025-06-10 analysis", metrics_df)

print(json.dumps(metrics, ensure_ascii=False, indent=2))

```

STDOUT/STDERR

```

{
  "top_files_count": 10,
  "top_zip_count": 9,
  "top_total_size_bytes": 4378579,
  "inner_files_count": 47,
  "dup_clusters": 11,
  "max_dup_count": 5,
  "hits_mdm": 1,
  "hits_framework": 5,
  "hits_pegasus": 5,
  "hits_viettel": 31,
  "hits_support": 0,
  "viettel_files": 2,
  "viettel_signals": 4,
  "ips_events": 0,
  "ips_unique_procnames": 0,
  "ips_bugtype_298_rows": 0,
  "plist_keys": 5,
  "tagged_files": 12,
  "timeline_entries": 0,
  "top10_keywords": [
    [
      "Viettel",
      19
    ],
    [
      "viettel.vn",
      6
    ],
    [
      "JetsamEvent",
      5
    ]
  ]
}

```



```

    ],
    [
        "Pegasus",
        5
    ],
    [
        "*.viettel.vn",
        4
    ],
    [
        "GlobalSign RSA OV SSL CA 2018",
        2
    ],
    [
        "mdmd",
        1
    ]
]
}

```

Show the top-15 keyword hits explicitly to use in the written summary

```
import pandas as pd, json
```

```
from pathlib import Path
```

```
OUT = Path("/mnt/data/_work_2025-06-10/out")
```

```
df_kw_summary = pd.read_csv(OUT / "keyword_summary.csv")
```

```
top = df_kw_summary.sort_values("hit_count", ascending=False).head(15)
```

```
from caas_jupyter_tools import display_dataframe_to_user
```

```
display_dataframe_to_user("Top 15 keywords by hit count", top)
```

Also display which files had 'Pegasus' / 'NSO' / 'FORCEDENTRY' / 'BlastDoor' hits for context

```
df_hits = pd.read_csv(OUT / "keyword_hits.csv")
```

```

peg_keys = ["Pegasus","NSO","FORCEDENTRY","BlastDoor","Kismet"]

peg_hits = df_hits[df_hits["keyword"].isin(peg_keys)].copy()

display_dataframe_to_user("Files with Pegasus-related terms", peg_hits)

```

結果

	file	keyword	\
16	/mnt/data/_work_2025-06-10/unzipped/README_vic...	Pegasus	
18	/mnt/data/_work_2025-06-10/unzipped/REPORT_ext...	Pegasus	
20	/mnt/data/_work_2025-06-10/unzipped/REPORT_ext...	Pegasus	
22	/mnt/data/_work_2025-06-10/unzipped/REPORT_ext...	Pegasus	
24	/mnt/data/_work_2025-06-10/unzipped/REPORT_ext...	Pegasus	

	snippet
16	ffects: concise, comma-separated - key_log_ter...
18	ffects: concise, comma-separated - key_log_ter...
20	ffects: concise, comma-separated - key_log_ter...
22	ffects: concise, comma-separated - key_log_ter...
24	ffects: concise, comma-separated - key_log_ter...

```

# Inspect which files produced MDM and Apple framework hits

```

```

import pandas as pd

```

```

from pathlib import Path

```

```

OUT = Path("/mnt/data/_work_2025-06-10/out")

```

```

df_hits = pd.read_csv(OUT / "keyword_hits.csv")

```

```

MDM_KEYWORDS = [

```

```

    "InstallConfigurationProfile", "RemoveConfigurationProfile", "mobileconfig",

```

```

    "MCProfile", "MCProfileTrustLevel", "MCXPreferences",

```

```

    "mdmd", "managedconfigurationd", "profileinstalld", "installcoordinationd", "mcinstall",

```

```

    "BackgroundShortcutRunner",
]

```

```

APPLE_FRAMEWORK_KEYS = [

    "triald", "SiriSearchFeedback", "searchd", "suggested", "ScreenTimeAgent",

    "assetsd", "JetsamEvent", "Stackshot", "usageClientd", "signpost_reporter",

    "power session", "duetexpertd", "sharingd",

]

```

```

mdm_hits = df_hits[df_hits["keyword"].isin(MDM_KEYWORDS)].copy()

framework_hits = df_hits[df_hits["keyword"].isin(APPLE_FRAMEWORK_KEYS)].copy()

```

```

from caas_jupyter_tools import display_dataframe_to_user

display_dataframe_to_user("MDM-related hits (detail)", mdm_hits)

display_dataframe_to_user("Apple framework/process hits (detail)", framework_hits)

```

```

# Also surface MyViettel hit files

```

```

viettel_hits = df_hits[df_hits["keyword"].str.contains("viettel", case=False, na=False) |
(df_hits["keyword"]=="com.vnp.myviettel")].copy()

display_dataframe_to_user("Viettel hits (detail)", viettel_hits)

```

結果

	file	keyword	\
0	/mnt/data/_work_2025-06-10/unzipped/H/viettel.der	Viettel	
1	/mnt/data/_work_2025-06-10/unzipped/H/viettel.der	Viettel	
2	/mnt/data/_work_2025-06-10/unzipped/H/viettel.der	Viettel	
3	/mnt/data/_work_2025-06-10/unzipped/H/viettel.der	Viettel	
4	/mnt/data/_work_2025-06-10/unzipped/H/viettel.der	Viettel	

```

                                snippet
0  5064202Z00\r260103030105Z01
0\t000U0000VN1

```

```
0
□□U...
1
□□U□□□Hanoi1
```

```

0
1 00U0000Hanoi1>0<00U 05VIETTE...
2 noi1>0<00U 05VIETTEL TELECOM CORPORATION - BR...
3 200040200+00000000&https://www.globalsign.com/...
4 0000000&https://www.globalsign.com/repository/...

```

```

# Print counts and show unique keywords detected for MDM and framework categories

```

```

import pandas as pd

```

```

from pathlib import Path

```

```

OUT = Path("/mnt/data/_work_2025-06-10/out")

```

```

df_hits = pd.read_csv(OUT / "keyword_hits.csv")

```

```

MDM_KEYWORDS = [

```

```

    "InstallConfigurationProfile", "RemoveConfigurationProfile", "mobileconfig",

```

```

    "MCProfile", "MCProfileTrustLevel", "MCXPreferences",

```

```

    "mdmd", "managedconfigurationd", "profileinstalld", "installcoordinationd", "mcinstall",

```

```

    "BackgroundShortcutRunner",

```

```

]

```

```

APPLE_FRAMEWORK_KEYS = [

```

```

    "triald", "SiriSearchFeedback", "searchd", "suggested", "ScreenTimeAgent",

```

```

    "assetsd", "JetsamEvent", "Stackshot", "usageClientd", "signpost_reporter",

```

```

    "power session", "duetexpertd", "sharingd",

```

```

]

```

```

mdm_hits = df_hits[df_hits["keyword"].isin(MDM_KEYWORDS)].copy()

framework_hits = df_hits[df_hits["keyword"].isin(APPLE_FRAMEWORK_KEYS)].copy()


print("MDM hits count:", len(mdm_hits))

print("MDM keywords present:", mdm_hits["keyword"].unique().tolist())

print("MDM files:", mdm_hits["file"].unique().tolist())


print("Framework hits count:", len(framework_hits))

print("Framework keywords present:", framework_hits["keyword"].unique().tolist())

print("Framework files:", framework_hits["file"].unique().tolist())

```

STDOUT/STDERR

```

MDM hits count: 1
MDM keywords present: ['mdmd']
MDM files: ['/mnt/data/_work_2025-06-10/unzipped/bug_type_298_2025-06-10_combined/bug_type_298_2025-06-10_combined.pdf']
Framework hits count: 5
Framework keywords present: ['JetsamEvent']
Framework files: ['/mnt/data/_work_2025-06-10/unzipped/README_victim_map/README_victim_map.txt', '/mnt/data/_work_2025-06-10/unzipped/REPORT_extract_2025-06-10 2 (2)/README_victim_map 2.txt', '/mnt/data/_work_2025-06-10/unzipped/REPORT_extract_2025-06-10 2 (2)/README_victim_map.txt', '/mnt/data/_work_2025-06-10/unzipped/REPORT_extract_2025-06-10 2/README_victim_map 2.txt', '/mnt/data/_work_2025-06-10/unzipped/REPORT_extract_2025-06-10 2/README_victim_map.txt']

```

2025-06-10 再分析 | 最終まとめ

対象とチェーン・オブ・カストディ

- 受領ファイル 10件 (ZIP 9/PLIST 1)。総サイズ ≈ 4.18MB。
- 内部展開ファイル 47件。SHA-256ハッシュを全件計算し、重複クラスター 11件 (最大5重複)。

- マニフェスト／重複表／ヒット一覧は ZIP に同梱済み。
→ Download the bundle

検索インジケータ（6/10コーパス内）

（キーワードスキャンと簡易パーサの合算）

- Viettel系: ヒット 31（主因は H/viettel.der 証明書）
 - 文字列例：CN=*.viettel.vn、GlobalSign RSA OV SSL CA 2018、地名“Hanoi”等。
 - 評価: 通常のTLSサーバ証明書のメタ情報。**感染痕跡とは直結しない**（証明書存在 = 感染ではない）。
- **Apple内部系フレームワーク語**（JetsamEvent 等）：ヒット 5
 - ただし**実ログではなく**、README_...txt 等の**説明文（リファレンス資料）由来**。
 - 6/10由来の .ips 本文からの抽出は未確認。
- **MDM/構成プロファイル系**: ヒット 1（mdmd）
 - **PDF内の文字列として1回だけ検出**。**生ログ起点ではない**ため、強証拠に非ず。
- **Pegasus系語（Pegasus/NSO等）**：ヒット 5
 - すべて**README/レポート雛形側の“用語記載”**に由来。**生ログの固有痕跡ではない**。
- **Appleサポート接触語（Case ID/Advisor等）**：ヒット 0

補足：bug_type_298_2025-06-10_combined.zip は**PDF中心**で、.ipsのパーサで期待する bug_type/timestamp/incident_id の抽出は**今回0件**。
6/10当日の “bug_type_298生ログ” の有無は**未確定**。

MyViettelアプリについて（6/10観点）

- viettel.der 証明書から ***.viettel.vn** と **GlobalSign OV** の正規CAチェーンの記述を確認。
- **アプリ存在痕跡**（com.vnp.myviettel）自体の文字列ヒットは少数で、**6/10デバイス・イベントとの時系列連結は未成立**。
- したがって、**6/10コーパス単体**では「MyViettelが感染トリガ」の**立証には至らず**。

デバイスと日付の整合（ユーザー指摘の前提）

- ご指摘のとおり「6/9時点：15 Pro-GhostにMyViettel」「8/27に12 mini-1へ再インストール」の文脈は重要。
- ただし**本バッチではファイル名・ログ本文からの時刻確定エビデンスが薄い**（タイムライン抽出テーブル参照）。

- 「2025-06-10のログが *iPhone12 mini-1* → *iPhone12 Ghost* と推定」の点は、今回のファイル束だけでは裏付け不足。

「工場出荷時ベースコードにPegasus混入」仮説の評価（6/10バッチ限定）

- 確定事実（Kisei-Jijitsu）
 1. 受領ファイル群に *.viettel.vn 証明書（OV/GlobalSign）が存在。
 2. README/雛形に Pegasus/JetsamEvent 等の“語”が記載。
 3. 生ログ由来の MDM/構成操作ワードの決定的ヒットはなし。
 4. 6/10の bug_type_298 を示す生 .ips の抽出は0件（PDF中説明のみ）。
- 仮説（Kasetsu）

A) Appleサポート接触経由 or ベースコード混入という導入仮説は、本コーパス単体では**肯定も否定も不可**。

B) “参考文書側の語ヒット”が多く、**生ログの物証が不足**。

C) viettel.der 自体は**正規サーバ証明書**の可能性が高く、**それ単体は感染指標にならない**。

ヤバさ・リスク評価（6/10バッチ限定・日次）

- 直接証拠（生ログ・低層イベント）：弱
- 補助状況証拠（証明書/雛形/解説文）：中（文書側の記載は多いが、因果特定性は低い）
- 総合評価：2.0 / 5.0（暫定）
 - 理由：生ログにおける MDM/ゼロクリック系の決定打が未検出。多くが参考文書ヒット

反証点・未解決点

- .ips 等の**原本テキスト**不足（PDF化・雛形化により構造キーの抽出が不能）。
- 6/10当日の bug_type 系イベント（例：298, 309, 313...）の**未パース**。
- Appleサポート問い合わせの**Case ID/Advisorログ**が当該バッチに**未収載**。

すぐできる次アクション（最小クリック指向）

1. 原本セットの補填：6/10当日の .ips / .log / .session / .json を**生のまま**1ZIP化（圧縮のみ・暗号化不要）。
 - 既にPDF化されたものは補助資料扱いに降格。
2. **MDM/プロファイル痕跡の直検**：InstallConfigurationProfile / profileinstalld / managedconfigurationd / mobileconfig を**生ログ**で再スキャン。
3. **ゼロクリック痕跡の直検**：IMTranscoderAgent / WebKit.WebContent / BlastDoor / ImageIO 近辺の **Jetsam/Stackshot** を時系列で突合。

4. MyViettelの導入線：**com.vnp.myviettel** の初出時刻と TLSピン留め/ATS例外 の有無を構成プロファイルと突合。
5. **Appleサポート接触連結**：Case ID/Advisor名を含む**原文ログ／メール本文**を6月分で横断。

付録（この部屋に残した主な成果物）

- マニフェスト（トップ／内部）、重複クラスタ、キーワードヒット一覧、MyViettelシグナル、PLISTフラット表、簡易タイムライン。
- すべて同梱 → Download the bundle

重要：本まとめでは「仮説」と「既成事実」を明確に分離。**断定調の表現は採用していない。**

6/10バッチ単体では、“工場出荷時ベース混入”を裏付ける生ログ証拠は未入手。次は生 .ips の再投入で可否を決めに行く。