xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
Alexis Brignoni

Orlando, Florida
For identification purposes only
Contributors

Research:

Sarah Edwards
Twitter: @iamevltwin
Blog: mac4n6.com
Contributors

Research:

Sarah Edwards
Twitter: @iamevltwin
Blog: mac4n6.com
Contributors

Co-Author:

Yogesh Khatri
Twitter: @SwiftForensics
Blog: swiftforensics.com
Contributors

Integration with Autopsy:

Mark McKinnon
Twitter: @markmckinnon
Blog: medium.com/
@markmckinnon_80619
Contributors

Super Friends:

Initialization vectors

Digital Forensics and Incident Response. All things InfoSec.

Tuesday, January 11, 2023

Awesome Friends!

SIEM wouldn't be possible without the efforts of some awesome friends. Heck, they go beyond awesome. They truly are...
Contributors

Super Friends:

Initialization vectors

Digital Forensics and Incident Response. All things InfoSec.

Tuesday, January 11, 2019

Awesome Friends!

It wouldn't be possible without the ached of some awesome friends. Heck, they go beyond awesome. They truly are...

From monitoring, charting, and beings modules on sharing and discussing all things digital and beyond, the following talks are truly heroic. I owe them a debt of gratitude for all they help.

abrignoni.blogspot.com/2020/01/awesome-friends.html
Contributors

Super Friends:

- Christopher Vance
- Shafik Punja
- Cheeky4N6monkey
- Jack Farley
- Douglas Klein
- Brooke Gottlieb
- Oleg4n6
- Edward Greybeard
- DFIR30c

- Agam Dua
- Sarah Edwards
- Heather Mahalik
- Jessica Hyde
- Brian Moran
- Geraldine Blay
- Phill Moore
- Mattia Epifani
- Mike Williamson

abrignoni.blogspot.com/2020/01/awesome-friends.html
For identification purposes only
xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
xLEAPP
iOS & Android Artifact Parsers

iLEAPP - iOS Logs, Events, and Preferences Parser
ALEAPP - Android Logs, Events, and Protobuf Parser
xLEAPP
iOS & Android Artifact Parsers

iLEAPP - iOS Logs, Events, and Preferences Parser
ALEAPP - Android Logs, Events, and Protobuf Parser
iLEAPP

iOS Logs, Events, and Preferences Parser

DEMO
ALEAPP

Android Logs, Events, and Protobuf Parser

DEMO
xLEAPP
iOS & Android Artifact Parsers

Raison d'être
xLEAPP
iOS & Android Artifact Parsers

Raison d'être
xLEAPP
iOS & Android Artifact Parsers

Raison d’être

- Assist investigators with the parsing of artifacts when vendor tools are out of reach.
xLEAPP
iOS & Android Artifact Parsers

Raison d’être

• Assist investigators with the parsing of artifacts when vendor tools are out of reach.
• Be part of an open source / free mobile forensics tool set.
xLEAPP
iOS & Android Artifact Parsers

Raison d’être

- Assist investigators with the parsing of artifacts when vendor tools are out of reach.
- Be part of an open source / free mobile forensics tool set.
- Serve as a triage tool for well sourced forensic labs and examiners.
xLEAPP
iOS & Android Artifact Parsers

Raison d’être

- Assist investigators with the parsing of artifacts when vendor tools are out of reach.
- Be part of an open source / free mobile forensics tool set.
- Serve as a triage tool for well sourced forensic labs and examiners.
- Provide an independent testing and validation tool framework.
xLEAPP
iOS & Android Artifact Parsers

Raison d’être

- Assist investigators with the parsing of artifacts when vendor tools are out of reach.
- Be part of an open source / free mobile forensics tool set.
- Serve as a triage tool for well sourced forensic labs and examiners.
- Provide an independent testing and validation tool framework.
- Parsing platform for newly discovered artifacts.
xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
CMD Line
CMD Line
CMD Line

Alexiss-MacBook-Pro:ALEAPP abrignoni$ python3 aleapp.py -h
usage: aleapp.py [-h] -t {fs,tar,zip} -o OUTPUT_PATH -i INPUT_PATH

ALEAPP: Android Logs, Events, and Protobuf Parser.

optional arguments:
  -h, --help  show this help message and exit
  -t {fs,tar,zip}  Input type (fs = extracted to file system folder)
  -o OUTPUT_PATH, --output_path OUTPUT_PATH  Output folder path
  -i INPUT_PATH, --input_path INPUT_PATH  Path to input file/folder
CMD Line
CMD Line

A Processing started. Please wait. This may take a few minutes...

--------------------
ALEAPP v1.2: Android Logs, Events, and Protobuf Parser
Objective: Triage Android Full System Extractions.
By: Alexis Brignoni | @AlexisBrignoni | abrignoni.com
By: Yogesh Khatri | @SwiftForensics | swiftforensics.com
Artifact categories to parse: 46
File/Directory selected: /Volumes/Black_Samsung_T5/_Android_Testing_Images/Android 10 Zip
        Non-Cellebrite Extraction/Pixel 3.zip

--------------------
Wellbeing artifact executing
Wellbeing artifact completed

Wellbeing artifact executing
Wellbeing artifact completed

Wellbeing artifact executing
Wellbeing artifact completed
CMD Line
Processes completed.
Processing time = 00:01:21

Report generation started.
Report generation Completed.

Report location: /Volumes/Black_Samsung_T5/Output/ALEAPP_Reports_2020-07-10_Friday_095938
Alexiss-MacBook-Pro:ALEAPP_abrignoni$
CMD Line
Android Logs, Events, And Protobuf Parser

https://github.com/abigail/ALEAPP

- Select the file type or directory of the target Android full file system extraction for parsing:

Select Output Folder:

Available Modules

- wellbeing.py [Wellbeing]
- wellbeingURLs.py [Wellbeing]
- wellbeingaccount.py [Wellbeing]
- wellbeing.py [Wellbeing]
- usageapps.py [App Interaction]
- usagestats.py [Usage Stats]
- recentactivity.py [Recent Activity]
- installedapps/InstalledApps
- installedapps/InstalledApps
- installedapps_library.py [Installed Apps]
- calllog.py [Call Logs]
- accounts.db.py [Accounts.db]
- accounts.db.py [Accounts.db]
- accounts.db.py [Accounts.db]
- accounts.db.py [Accounts.db]
- accounts.db.py [Accounts.db]
Requirements

- Python 3
- Pip install -r requirements.txt
  - beautifulsoup4==4.8.2
  - protobuf==3.10.0
  - PySimpleGUI==4.16.0
  - PyCryptodome
  - packaging==20.1
  - pathlib2==2.3.5
  - PySimpleGUI==4.16.0
  - biplist
  - More...

https://github.com/abrignoni
Requirements

- Python 3
- Pip install -r requirements.txt
  - beautifulsoup4==4.8.2
  - protobuf==3.10.0
  - PySimpleGUI==4.16.0
  - PyCryptodome
  - packaging==20.1
  - pathlib2==2.3.5
  - PySimpleGUI==4.16.0
  -plist
  - More...

https://github.com/abrignoni
Autopsy 4.17

- Windows executable included
- Artifact support
- Configuration file
- Future release: ALEAPP & more artifacts

iOS Forensics With iLEAPP

There is a new "iOS Analyzer (iLEAPP)" ingest module that wraps the iLEAPP program from Alexis Brignoni, Yogesh Khatri, and others. This effort is a work in progress and this release parses a limited number of the wide variety of results that iLEAPP can produce. The next release will add the rest of the artifacts, support for disk images, and aLEAPP for Androids!

Inputs

For this release, the input to iLEAPP needs to be a tar file. If you add the tar file as a logical file to Autopsy and it is in the root folder, then the iOS Analyzer module will detect it and run iLEAPP on it.

For example:
- Your data source is at c:\images\case123\phone1.tar
- You add the c:\images\case123\phone1.tar file as a logical file into Autopsy.

Artifacts

This release focuses on artifacts that Autopsy already had support for. Such as web browsers, WiFi, installed programs, programs run, etc.

Autopsy gets them by running iLEAPP to produce TSV files. Autopsy has a configuration file that maps the files and columns to an Autopsy artifact. As iLEAPP adds new modules, all someone needs to do is update this XML file. No code is required!

Next Release

In addition to mapping more artifacts, the next release will also allow you to analyze disk images or logical images that are not in a TAR file. iLEAPP added a feature to output the list of files it needs. We’ve used that list to query Autopsy’s file manager for them, export them to disk, and then run iLEAPP on them.
Autopsy 4.17

- Windows executable included
- Artifact support
- Configuration file
- Future release: ALEAPP & more artifacts

iOS Forensics With iLEAPP

There is a new "iOS Analyzer (iLEAPP)" ingest module that wraps the iLEAPP program from Alexis Brignoni, Yogesh Khatri, and others. This effort is a work in progress and this release parses a limited number of the wide variety of results that iLEAPP can produce. The next release will add the rest of the artifacts, support for disk images, and aLEAPP for Androids!

Inputs

For this release, the input to iLEAPP needs to be a tar file. If you add the tar file as a logical file to Autopsy and it is in the root folder, then the iOS Analyzer module will detect it and run iLEAPP on it.

For example:

- Your data source is at c:\images\case123\phone1.tar
- You add the c:\images\case123\phone1.tar file as a logical file into Autopsy.

Artifacts

This release focuses on artifacts that Autopsy already had support for. Such as web browsers, WiFi, installed programs, programs run, etc.

Autopsy gets them by running iLEAPP to produce TSV files. Autopsy has a configuration file that maps the files and columns to an Autopsy artifact. As iLEAPP adds new modules, all someone needs to do is update this XML file. No code is required!

Next Release

In addition to mapping more artifacts, the next release will also allow you to analyze disk images or logical images that are not in a TAR file. iLEAPP added a feature to output the list of files it needs. We’ve used that list to query Autopsy’s file manager for them, export them to disk, and then run iLEAPP on them.
Extracted Content
- BlueTooth Pairings (61)
- Bluetooth Adapter (94)
- Calendar Entries (119)
- Device Info (10)
- GPS Last Known Location (917)
- Installed Programs (470)
- Operating System Information (9)
- Recent Documents (16)
- Run Programs (6952)
- Web History (69)
- Web Search (6)
- Wireless Networks (2156)
xLEAPP for iOS & Android
 Parse and validate mobile artifacts with Python
Parsing
Parsing

- SQLite
Parsing

- SQLite
- Protobuf
**Parsing**

- SQLite
- Protobuf
- Binary Files
```python
import glob
import os
import pathlib
import sqlite3

from scripts.api import ArtifactMetaReport
from scripts.api import logfunc, tsf, timestamp, timeline, is_platform_windows

def get_tileAppFiles(file_found, report_folder, seeker):
    for file in file_found:
        file_found = str(file_found)

        if file_found.endswith('TileNetworkDB.sqlite'):
            break

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute(""
SELECT
    @TIMESTAMP,’unixepoch’,'31 years’),
    NAME,
    @TIMESTAMP,’unixepoch’,'31 years’),
    DATETIME(UNIX_TIMESTAMP,’unixepoch’,’31 years’),
    DATETIME(UNIX_TIMESTAMP,’unixepoch’,’31 years’),
    ZONE_TYPE,
    ZONE,
    UPPER(ADDRESS),
    @TIMESTAMP,’unixepoch’,'31 years’),
    LAST_CONNECTION,
    ZONE,
    ""
FROM ZTILEENTITY_MODE INNER JOIN ZTILEENTITY_TILESTATE ON ZTILEENTITY_MODE.ZTILEENTITY TILESTATE = ZTILEENTITY_TILESTATE.Z_PK
""

all_rows = cursor.fetchall()

for row in all_rows:
    data_list = []
    if row[1] is not None:
        data_list.append(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10])
        data_list.append(row[11])

        description = ""
        report = ArtifactMetaReport("Tile App - Tile Information & Geolocation")
        report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
        report.add_script()
        report.write_artifact_data_table(data_headers, data_list, file_found)
        report.end_artifact_report()

        tsvname = 'Tile App DB Info Geolocation'
        tsv(report_folder, data_headers, data_list, tsvname)

        tsvactivity = 'Tile App DB Info Geolocation'
        timeline(report_folder, tsvactivity, data_list, data_headers)

        htmlactivity = 'Tile App DB Info Geolocation'
        htmlreport(report_folder, htmlactivity, data_list, data_headers)
    else:
        logfunc('No Tile App DB data available')

db.close()
```

```python
import glob
import os
import pathlib
import sqlite3

from scripts.artifact_report import ArtifactHtmlReport
from scripts.ilapfuncs import logfunc, tsv, timeline, is_platform_windows

def get_TileApp(files_found, report_folder, seeker, wrap_text):
    for file_found in files_found:
        file_found = str(file_found)

        if file_found.endswith('TileNetwork.db.sqlite3'):
            break

db = sqlite3.connect(file_found)
cursor = db.cursor()
cursor.execute(''

SELECT
  B.Intime(ZTIMEESTAMP, 'unixepoch', '31 years'),
  NAME,
  date_time(ZDATESTAMP, 'unixepoch', '31 years'),
  date_time(ZDATESTAMP, 'unixepoch', '31 years'),
  ZLATITUDE,
  ZLONGITUDE,
  ZID,
  ZNODE_TYPE,
  ZSTATUS,
  ZIS_LOST,
  SELECT ZLAST_LOST_TILE_COMMUNITY_CONNECTION, 'unixepoch', '31 years')
FROM ZENTITY_NODE INNER JOIN ZENTITY_TILESTATE ON ZENTITY_NODE.ZENTITY_ID = ZENTITY_TILESTATE.Z_PK

'''

all_rows = cursor.fetchall()
baseentitlen = len(all_rows)
data_list = []

for row in all_rows:
    data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

description = ''

report = ArtifactHtmlReport('Tile App - Tile Information & GeoLocation')
report.start_artifact_report(report_folder, 'Tile App DB Info & GeoLocation', description)
report.add_script('

for data in data_list:
    datadesc = data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7], data[8], data[9], data[10], data[11]

    tsvname = 'Tile App DB Info GeoLocation'
tsv(trash_folder, data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7], data[8], data[9], data[10], data[11], data[0])

else:
    logfunc('No Tile App DB data available')

db.close()

return
```

import glob
import os
import pathlib
import sqlite3

from scripts.uploads import ArtifactReport
from scripts.uploads import ArtifactReport

def get_tileApp(file_found, report_folder, searcher):
    for file_found in files_found:
        file_found = str(file_found)
        if file_found.endswith("tile-TileNetwork.B.db.sqlite"):
            break

db = sqlite3.connect(file_found)
cursor = db.cursor()
cursor.execute("SELECT
                   sqlite3.schema_name,
                   sqlite3.object_type,
                   sqlite3.name,
                   sqlite3.created,
                   sqlite3.updated,
                   sqlite3.size
FROM
        sqlite3.objects
WHERE
        type = 'table';")
rows = cursor.fetchall()
data_list = []
for row in all_rows:
    data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

description = """report = ArtifactReport('Tile App - Tile Information & Geolocation')
report.add_report_info('Tile App DB Info & Geolocation', description)
report.add_column('Tile ID', 'Tile Type', 'Status', 'Tile Last', 'Last Community Connection')
report.add_data_table(data_headers, data_list, file_found)
report.end_artifact_report()
tsvname = "Tile App DB Info Geolocation"
tsv(report_folder, data_headers, data_list, tsvname)
"
else:
    logfunc('No Tile App DB data available')

db.close()
import os
import shutil
from scripts.artifact_report import ArtifactReport
from scripts.ilapfuncs import tsv, display, parse_json

if os.path.exists('LoginDB.db.sqlite3'):
    db = sqlite3.connect('LoginDB.db.sqlite3')
    cursor = db.cursor()
    cursor.execute(""
SELECT
    * FROM UserTable
WHERE
    username = 'admin'
"")
    rows = cursor.fetchall()
import glob
import os
import pathlib
import sqlite3

from scripts.api.functions import logfunc, tsv, xmlgen, timeline, is_platform_windows

def get_titleApp(files_found, report_folder, seeker):
    for file_found in files_found:
        file_found = str(file_found)

        if file_found.endswith('TileNetworxNOdb.sqlite'):
            break

db = sqlite3.connect(file_found)
cursor = db.cursor()
cursor.execute("SELECT BIRTH_TIME,ZONAL_TIMESTAMP, unixepoch", "31 years");
NAME;
DATETIMEX(DATETIME_TIMESTAMP, unixepoch`, "31 years");
DATETIMEX(DATETIME_TIMESTAMP, unixepoch`, "31 years");
LATITUDE;
LONGITUDE;
ZID;
NUMERO_TYPE;
CLASS;
ZIS_LOST;
SELECT (TAPE_LOST_TILE_COMMUNITY_CONNECTION, unixepoch`, "31 years")
FROM ZTILEENTITY_NODE INNER JOIN ZTILEENTITY_TILESTATE ON ZTILEENTITY_TILESTATE.ZTILEENTITY_NODE = ZTILEENTITY_TILESTATE.Z_PK

all_rows = cursor.fetchall()

postgresql = |set| all_rows

data_list = []

if postgresql > 0:
    for row in all_rows:
        data_list.append([row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]])

        description = ""
        report = ArtifactHistReport("Tile App - Tile Information & Geolocation")
        report.start_artifact_report(report_folder, "Tile App DB Info & Geolocation", description)
        report.add_script(""
        data_headers = ["TileName", "TileNumber", "ActivationTimestamp", "RegistrationTimestamp", "Altitude", "Longitude", "Tile ID", "Tile Type", "Status", "Is Lost", "Last Community Connection"]
        report.write_artifact_data_table(data_headers, data_list, file_found)
        report.end_artifact_report()

        tsvname = "Tile App DB Info Geolocation"
        tsv(report_folder, data_headers, data_list, tsvname)

        lactivity = 'Tile App DB Info Geolocation'
timeline(report_folder, lactivity, data_list, data_headers)

        kmlactivity = 'Tile App DB Info Geolocation'
kmlgen(report_folder, kmlactivity, data_list, data_headers)

else:
    logfunc("No Tile App DB data available")

db.close()

return
```python
import glob
import os
import sqlite3
from scripts.api_report import ArtifactHashReport
from scripts.api import logfunc, tsv, kmlgen, timeline, is_platform_windows

def get_titleApps(file_found, report_folder, seeker):
    for file_found in file_found:
        file_found = str(file_found)
        if file_found.endswith('TileTileNetworkDB.sqlite'):
            break

db = sqlite3.connect(file_found)
cursor = db.cursor()
cursor.execute(
    "SELECT
        B.buildtime(ZONE_TIMESTAMP, 'unixepoch', '31 years'),
        NAME,
        datatime(ZONE_TIMESTAMP, 'unixepoch', '31 years'),
        datatime(ZONE_TIMESTAMP, 'unixepoch', '31 years'),
        VALUE,
        LATITUDE,
        LONGITUDE,
        ID,
        SENDER_TYPE,
        COUNTRY,
        ZIP_CODE,
        ZIP_POSTAL,
        LAST_LOST_TILE_COMMUNITY_CONNECTION, 'unixepoch', '31 years')
FROM ZILENTITY_NODE INNER JOIN ZILENTITY_TILESTATE ON ZITLENTITY_NODENODE_ID = ZITLENTITY_TILESTATE.Z_PK
"
)
all_rows = cursor.fetchall()
all_rows = [row[0] for row in all_rows]
data_list = []
if len(all_rows) > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))
        description = ""
        report = ArtifactHashReport("Tile App - Tile Information & Geolocation")
        report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
        report.add_script(
            data_header = ["TitleApp", "Last Lost Tile", "Last Community Connection"],
            timestamp = "registration_timestamp",
            latitude = "latitude", longitude = "longitude", tileID = "tileID", tileType = "tileType", status = "status", isLast = "isLast", lastCommunityConnection = "lastCommunityConnection"
        )
        report.write_artifact_data_table(data_headers, data_list, file_found)
        report.end_artifact_report()
        tsvname = "Tile App DB Info Geolocation"
tsv(report_folder, data_headers, data_list, tsvname)
        tactivity = 'Tile App DB Info Geolocation'
timeline(report_folder, tactivity, data_list, data_headers)
        kmlactivity = 'Tile App DB Info Geolocation'
kmlgen(report_folder, kmlactivity, data_list, data_headers)
else:
    logfunc("No Tile App DB data available")

db.close()
```

```python
import glob
import os
import pathlib
import sqlite3

from scripts.api import ArtifactReport
import logfunc, tsv, xmlgen, timeline, is_platform_windows

def get_tileAppDB(file_found, report_folder, seeker):
    for file in os.listdir(file_found):
        if file.endswith('tile-networkDB.sqlite3'):
            break

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute("SELECT
                    BRTIMESTAMP, 'unixepoch', '31 years',
                    NAME,
                    DATETIMESTAMP_TIMESTAMP, 'unixepoch', '31 years',
                    DATETIMESTAMP_TIMESTAMP_TIMESTAMP, 'unixepoch', '31 years',
                    ZDATA,
                    LATITUDE,
                    LONGITUDE,
                    Z
                    FROM ZTILEENTITY_NODE INNER JOIN ZTILEENTITY_TILESTATE ON ZTILEENTITY_NODE.ZTILEENTITY_TILESTATE = ZTILEENTITY_TILESTATE.Z_PK"
                    ...
                )
    all_rows = cursor.fetchall()
    for row in all_rows:
        data_list = {}
        if len(row) > 0:
            for col in row:
                data_list[col] = row[col]

    description = ""
    report = ArtifactReport("Tile App - Tile Information & Geolocation")
    report_add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report_add_script()
    report_write_artifact_data_table(data_headers, data_list, file_found)
    report_end_artifact_report()

    tsvname = "Tile App DB Info Geolocation"
    tsv(report_folder, data_headers, data_list, tsvname)
    if activity == "Tile App DB Info Geolocation"
        timeline(report_folder, activity, data_list, data_headers)
        xmlgen(report_folder, activity, data_list, data_headers)
    else:
        logfunc("No Tile App DB data available")

    db.close()
```
import glob
import os
import pathlib
import sqlite3

from scripts.scripts import importArtifactReport
from scripts.sql import 


def get_tileAppFiles(file_found, report_folder, sekk): 
    for file_found in files_found: 
        file_found = str(file_found) 
        if file_found.endswith('tileNetworkDatabase.sqlite'): 
            from 
            db = sqlite3.connect(file_found) 
            cursor = db.cursor() 
            cursor.execute("SELECT 
                
                   好きなタイムスタンプ, 'unixepoch', '31 years', 
                    NAME, 
                    datetime('unixepoch', '31 years'), 
                    DATATIME, 
                    LATITUDE, 
                    LONGITUDE, 
                    J1D, 
                    ID, 
                    SMOKE, 
                    ZIG_Lost, 
                    DATETIME, 'tileCommunityConnection', 'unixepoch', '31 years') 
            FROM Z TILEENTITY_NODE INNER JOIN Z TILEENTITY_TILESTATE ON Z TILEENTITY_NODE.Z TILESTATE = Z TILEENTITY_TILESTATE.Z_PK 
            ...
            
            all_rows = cursor.fetchall() 
            data_list = [] 
            for row in all_rows: 
                if row[0]: 
                    data_list.append(row)

                    description = "" 
                    report = ArtifactReport('Tile App - Tile Information & Geolocation') 
                    report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description) 
                    report.add_script() 
                    data_headers = ['tileNames', 'tileNames', 'tileNames', 'tileNames', 'tileNames', 'tileNames', 'tileNames', 'tileNames', 'tileNames'] 
                    report.write_artifact_data_table(data_headers, data_list, file_found) 
                    report.end_artifact_report() 

                    tsvname = 'Tile App DB Info Geolocation' 
                    tsv(report_folder, data_headers, data_list, tsvname) 
                    tsvactivity = 'Tile App DB Info Geolocation' 
                    tsv(report_folder, tsvactivity, data_list, data_headers) 
                    kmlactivity = 'Tile App DB Info Geolocation' 
                    kmlreport(report_folder, kmlactivity, data_list, data_headers) 
                    else: 
                        logfunc('No Tile App DB data available') 
            
            db.close() 
            return

    return

Results
import glob
import os
import pathlib
import sqlite3
from scripts.api.functions import logfunc, ts, xmlgen, timeline, is_platform_windows

def sqlite(sql_file, report_folder, seeker):
    db = sqlite3.connect(sql_file)
    cursor = db.cursor()
    cursor.execute(sql)
    for row in cursor.fetchall():
        data_list = []
        data_list.append(row[0])
        data_list.append(row[1])
        data_list.append(row[2])
        data_list.append(row[3])
        data_list.append(row[4])
        data_list.append(row[5])
        data_list.append(row[6])
        data_list.append(row[7])
        data_list.append(row[8])
        data_list.append(row[9])
        data_list.append(row[10])
        data_list.append(row[11])

        description = """"""report = ArtifactReport('Tile App - Tile Information & Geolocation')
        report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
        report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', data_list)
        report.write_artifact_data_table(data_list, data_list, file_found)
        report.end_artifact_report()

        tsvname = 'Tile App DB Info Geolocation'
        tsv(report_folder, data_list, data_list, tsvname)
        tsvactivity = 'Tile App DB Info Geolocation'
        timeline(report_folder, tsvactivity, data_list, data_list)

        xmlgen(report_folder, data_list, data_list)

        else:
            logfunc('No Tile App DB data available')

    db.close()
import glob
import os
import pathlib
import sqlite3
from scripts.api.deployment import logfunc, tsv, xmlgen, timeline, is_platform_windows

def get_tileApp(files_found, report_folder, seeker):
    for file_found in files_found:
        file_found = str(file_found)
        if file_found.endswith('TileApp.db.sqlite'):
            break

db = sqlite3.connect(file_found)
cursor = db.cursor()
cursor.execute(''
SELECT
   _begintimestamp(ZTIMESTAMP, 'unepoch', '31 years'),
    NAME,
    datetimetz(activation_timestamp, 'unepoch', '31 years'),
    ZALTITUDE,
    LATITUDE,
    LONGITUDE,
    ZDM,
    NOE_TYPE,
    CRISIS,
    ZLS_LAST,
    datetimetz(last_lost_TILE_COMMUNITY_CONNECTION, 'unepoch', '31 years')
FROM ZTILEENTITY_MODE INNER JOIN ZTILEENTITY_TILESTATE ON ZTILEENTITY_MODE.ZTILEENTITY_TILESTATE.Z_PK
    ...'
)
all_rows = cursor.fetchall()
segmentsInfo = [row[0] for row in all_rows]
data_list = []
for row in all_rows:
data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

    description = ''
    report = ArtifactReport('Tile App - Tile Information & Geolocation')
    report.add_artifact(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.write_artifact_data_file(data_list, file_found, file_found)
    report.end_artifact_report()

tsvname = 'Tile App DB Info Geolocation'
tsv(report_folder, data_list, tsvname)
lactivity = 'Tile App DB Info Geolocation'
timeline(report_folder, lactivity, data_list, data_headers)
mlactivity = 'Tile App DB Info Geolocation'
xmlgen(report_folder, mlactivity, data_list, data_headers)
else:
    logfunc('No Tile App DB data available')

db.close()
```python
import glob
import os
import pathlib
import sqlite3
from scripts import ArtifactDbReport
from scripts.laplfuncs import logfunc, tsv, xmlgen, timeline, is_platform_windows

def getTileAppDB(file_found, report_folder, seeker):
    for file_found in file_found:
        file_found = str(file_found)
        if file_found.endswith('TilesNetwork.db.sqlite'):
            db = sqlite3.connect(file_found)
            cursor = db.cursor()
            cursor.execute(
                'SELECT
                    HotelName, UniqID, uniopech, "15 years",
                    Name, referendum3, referendum4, referendum5, referendum6, referendum7, referendum8, referendum9, referendum10, referendum11, referendum12, referendum13, referendum14, referendum15, referendum16, referendum17, referendum18, referendum19, referendum20, referendum21, referendum22, referendum23, referendum24, referendum25, referendum26, referendum27, referendum28, referendum29, referendum30, referendum31, referendum32, referendum33, referendum34, referendum35, referendum36, referendum37, referendum38, referendum39, referendum40, referendum41, referendum42, referendum43, referendum44, referendum45, referendum46, referendum47, referendum48, referendum49, referendum50, referendum51, referendum52, referendum53, referendum54, referendum55, referendum56, referendum57, referendum58, referendum59, referendum60, referendum61, referendum62, referendum63, referendum64, referendum65, referendum66, referendum67, referendum68, referendum69, referendum70, referendum71, referendum72, referendum73, referendum74, referendum75, referendum76, referendum77, referendum78, referendum79, referendum80, referendum81, referendum82, referendum83, referendum84, referendum85, referendum86, referendum87, referendum88, referendum89, referendum90, referendum91, referendum92, referendum93, referendum94, referendum95, referendum96, referendum97, referendum98, referendum99, referendum100, referendum101, referendum102, referendum103, referendum104, referendum105, referendum106, referendum107, referendum108, referendum109, referendum110, referendum111, referendum112, referendum113, referendum114, referendum115, referendum116, referendum117, referendum118, referendum119, referendum120, referendum121, referendum122, referendum123, referendum124, referendum125, referendum126, referendum127, referendum128, referendum129, referendum130, referendum131, referendum132, referendum133, referendum134, referendum135, referendum136, referendum137, referendum138, referendum139, referendum140, referendum141, referendum142, referendum143, referendum144, referendum145, referendum146, referendum147, referendum148, referendum149, referendum150, referendum151, referendum152, referendum153, referendum154, referendum155, referendum156, referendum157, referendum158, referendum159
                FROM ZITLESTATE
                INNER JOIN ZITLESTATE ON ZITLESTATE.ZITLESTATE = ZITLESTATE.ZITLESTATE.Z_PK
                ...')
            all_rows = cursor.fetchall()
            segmentation = list(len_rows)
            data_list = []
            if segmentation > 0:
                for row in all_rows:
                    data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))
            description = ''
            report = ArtifactDbReport('Tile App - Tile Information & Geolocation')
            report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
            report.add_script(
                'sql
            tsv_report = tsv(report_folder, data_list, all_headers)
            tsv_report(tsv_folder, data_list, all_headers)
            else:
                logfunc('No Tile App DB data available')
            db.close()
``
import qglb
import os
import pathlib
import sqlite3

from scripts.flatten import flatten
import logfunc, tsv, xmlgen, timeline, is_platform_windows

def get_tileAppDB(file_found, report_folder, folder):
    if file_found in file_found:
        file_found = str(file_found)
        if file_found.endswith('.sqlite3'):
            print(file_found)

        db = sqlite3.connect(file_found)
        cursor = db.cursor()
        cursor.execute("SELECT
                        DATETIME(ZONETIMESTAMP, 'unixepoch', '31 years'),
                        NAME,
                        DATETIME(ZAKTIVIERUNG_TIMESTAMP, 'unixepoch', '31 years'),
                        ZAKTIVIERUNG_TIMESTAMP,
                        LATITUDE,
                        LONGITUDE,
                        Z
                        FROM ZTILESTATE WHERE ZTILESTATE = ZTILESTATE_Z_PK
                        ..."
                     )
        all_rows = cursor.fetchall()
        data_list = []
        if len(all_rows) > 0:
            for row in all_rows:
                data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9]))

        description = "",
        report = ArtifactReport('Tile App - Title Information & Geolocation')
        report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
        report.add_script() report.write_artifact_data_table(data_headers, filename, file_found)
        report.end_artifact_report()

        tsvname = 'Tile App DB Info Geolocation'
        tsv_report_folder, data_headers, filename, tsvname

        if activity == 'Tile App DB Info Geolocation'
        timeline(report_folder, tsvname, data_list, data_headers)
        else:
            logfunc('No Tile App DB data available')

        db.close()
import glob
import os
import os.path
import sqlite3

from scripts.lib import logfunc, tsv, xlsxgen, timeline, is_platform_windows

def get_tileApps(file_found, report_folder, seemsOk):
    for file_found in file_found:
        file_found = str(file_found)

        if file_found.endswith('tile-TileNetworkDB.sqlite'):
            return

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute(""
    SELECT
        BREETIMESTAMP, 'unixepoch', '31 years',
        NAME,
        datetimel('ZACCESSION_TIMESTAMP', 'unixepoch', '31 years'),
        datetimel('ZACCESSION_TIMESTAMP', 'unixepoch', '31 years'),
        ZLATITUDE,
        ZLONGITUDE,
        ZID,
        ZMODE_TYPE,
        ZSTATUS,
        ZIS_LOCAL,
        ZLAST_SENT_TILE_COMMUNITY_CONNECTION, 'unixepoch', '31 years')
    FROM ZTILEENTITY_NODE INNER JOIN ZTILEENTITY_TILESTATE ON ZTILEENTITY_NODE.ZTILE_STATE = ZTILEENTITY_TILESTATE.Z_PK
    ...
    ""
    all_rows = cursor.fetchall()
    segmentslist = [str(all_rows)"
    data_list = []
    if segmentslist > 0:
        for row in all_rows:
            data_list.append([row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]])
            description = '"
            report = ArtifactRemoval('Tile App - Tile Information & Geolocation')
            report.add_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()
    tsvname = 'Tile App DB Info Geolocation'
    tsv(report_folder, data_headers, data_list, tsvname)
    tactivity = 'Tile App DB Info Geolocation'
    timeline(report_folder, tactivity, data_list, data_headers)
    kmlactivity = 'Tile App DB Info Geolocation'
    kmrlogon(report_folder, kmlactivity, data_list, data_headers)
    else:
        logfunc('No Tile App DB data available')

    db.close()

return
```python
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

    description = ''
    report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
    report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_script()
    data_headers = ('Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration
    Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community
    Connection')
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()

    tsvname = 'Tile App DB Info Geolocation'
    tsv(report_folder, data_headers, data_list, tsvname)

    tactivity = 'Tile App DB Info Geolocation'
    timeline(report_folder, tactivity, data_list, data_headers)

    kmlactivity = 'Tile App DB Info Geolocation'
    kmlogen(report_folder, kmlactivity, data_list, data_headers)
else:
    logfunc('No Tile App DB data available')

db.close()
return
```
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append([row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]])

    description = ''
    report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
    report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_script()
    data_headers = ['Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration
                   Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community
                   Connection']
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()

    tsvname = 'Tile App DB Info Geolocation'
    tsv(report_folder, data_headers, data_list, tsvname)

    tactivity = 'Tile App DB Info Geolocation'
    timeline(report_folder, tactivity, data_list, data_headers)

    kmlactivity = 'Tile App DB Info Geolocation'
    kmlgen(report_folder, kmlactivity, data_list, data_headers)
else:
    logfunc('No Tile App DB data available')

db.close()
return
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

    description = ''
    report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
    report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_script()
    data_headers = ('Timestamp', 'Tile Name', 'Registration',
                    'Activation Timestamp', 'Tile Type',
                    'Altitude', 'Latitude', 'Longitude',
                    'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community Connection')
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()

tsvname = 'Tile App DB Info Geolocation'
tsv(report_folder, data_headers, data_list, tsvname)

tactivity = 'Tile App DB Info Geolocation'
timline(report_folder, tactivity, data_list, data_headers)

else:
    logfunc('No Tile App DB data available')

db.close()
return
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))
    description = ':
    report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
    report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_script()
    data_headers = ('Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration
        Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community
        Connection')
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()

tsvname = 'Tile App DB Info Geolocation'
tsv(report_folder, data_headers, data_list, tsvname)

tactivity = 'Tile App DB Info Geolocation'
timeline(report_folder, tactivity, data_list, data_headers)

kmlactivity = 'Tile App DB Info Geolocation'
kmlgen(report_folder, kmlactivity, data_list, data_headers)

else:
    logfunc('No Tile App DB data available')

db.close()
return
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append([row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]])

    description = ''
    report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
    report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_script()
    data_headers = ('Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community Connection')
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()

tsvname = 'Tile App DB Info Geolocation'
sv(report_folder, data_headers, data_list, tsvname)

tactivity = 'Tile App DB Info Geolocation'
timeline(report_folder, tactivity, data_list, data_headers)

kmlactivity = 'Tile App DB Info Geolocation'
kmlgen(report_folder, kmlactivity, data_list, data_headers)

else:
    logfunc('No Tile App DB data available')

db.close()
return
```python
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

        description = ''
        report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
        report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
        report.add_script()
        data_headers = ('Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration
                        Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community
                        Connection')
        report.write_artifact_data_table(data_headers, data_list, file_found)
        report.end_artifact_report()

    tsvname = 'Tile App DB Info Geolocation'
    tsv(report_folder, data_headers, data_list, tsvname)

    tactivity = 'Tile App DB Info Geolocation'
    timeline(report_folder, tactivity, data_list, data_headers)

    kmlactivity = 'Tile App DB Info Geolocation'
    kmllen(report_folder, kmlactivity, data_list, data_headers)

else:
    logfunc('No Tile App DB data available')

db.close()
return
```
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

        description = ''
        report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
        report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
        report.add_script()
        data_headers = ['Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration
                        Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community Connection']
        report.write_artifact_data_table(data_headers, data_list, file_found)
        report.end_artifact_report()

        tsvname = 'Tile App DB Info Geolocation'
        tsv(report_folder, data_headers, data_list, tsvname)

        tlactivity = 'Tile App DB Info Geolocation'
        timeline(report_folder, tlactivity, data_list, data_headers)

        kmlactivity = 'Tile App DB Info Geolocation'
        kmlgen(report_folder, kmlactivity, data_list, data_headers)

    else:
        logfunc('No Tile App DB data available')

db.close()
return
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

        description = ''
        report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
        report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
        report.add_script()
        data_headers = ('Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration
            Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Communication Connection')
        report.write_artifact_data_table(data_headers, data_list, file_found)
        report.end_artifact_report()

        tsvname = 'Tile App DB Info Geolocation'
        tsv(report_folder, data_headers, data_list, tsvname)

        tactivity = 'Tile App DB Info Geolocation'
        timeline(report_folder, tactivity, data_list, data_headers)

        kmlactivity = 'Tile App DB Info Geolocation'
        kmlgen(report_folder, kmlactivity, data_list, data_headers)
else:
    logfunc('No Tile App DB data available')

db.close()
return
```
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

    description = ''
    report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
    report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_script()
    data_headers = ('Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration
        Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community Connection')
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()

    tsvname = 'Tile App DB Info Geolocation'
    tsv(report_folder, data_headers, data_list, tsvname)

    tactivity = 'Tile App DB Info Geolocation'
    timeline(report_folder, tactivity, data_list, data_headers)

    kmlactivity = 'Tile App DB Info Geolocation'
    kmlgen(report_folder, kmlactivity, data_list, data_headers)
else:
    logfunc('No Tile App DB data available')

db.close()
return
```
data_list = []
if usageentries > 0:
    for row in all_rows:
        data_list.append((row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11]))

    description = ''
    report = ArtifactHtmlReport('Tile App - Tile Information & Geolocation')
    report.start_artifact_report(report_folder, 'Tile App DB Info & Geolocation', description)
    report.add_script()
    data_headers = ('Timestamp', 'Tile Name', 'Activation Timestamp', 'Registration Timestamp', 'Altitude', 'Latitude', 'Longitude', 'Tile ID', 'Tile Type', 'Status', 'Is Lost?', 'Last Community Connection')
    report.write_artifact_data_table(data_headers, data_list, file_found)
    report.end_artifact_report()

tsvname = 'Tile App DB Info Geolocation'
tsv(report_folder, data_headers, data_list, tsvname)

tlactivity = 'Tile App DB Info Geolocation'
timeline(report_folder, tlactivity, data_list, data_headers)

kmlactivity = 'Tile App DB Info Geolocation'
kmlgen(report_folder, kmlactivity, data_list, data_headers)

else:
    logfunc('No Tile App DB data available')

db.close()
return