
xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
What
When
Contact
Why
Where
Who
How

Alexis Brignoni

Orlando, Florida


DEPARTMENT OF JUSTICE
FEDERAL BUREAU OF INVESTIGATION
FIDELITY
BRAVERY
INTEGRITY
For identification purposes only


DEPARTMENT OF JUSTICE
FEDERAL BUREAU OF INVESTIGATION
FIDELITY
BRAVERY
INTEGRITY
For identification purposes only
Community

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

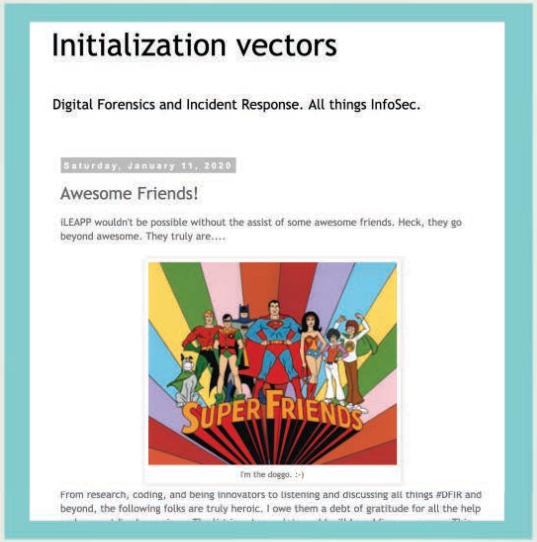
Contributors

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.

Saturday, January 11, 2020

Awesome Friends!

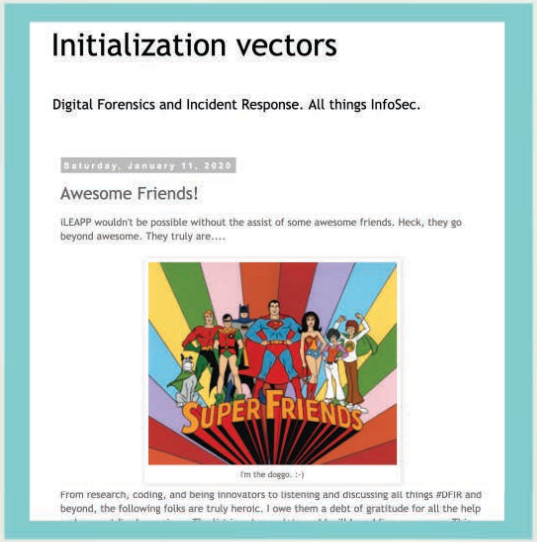
iLEAPP wouldn't be possible without the assist of some awesome friends. Heck, they go beyond awesome. They truly are....

I'm the doggo. :-)

From research, coding, and being innovators to listening and discussing all things #DFIR and beyond, the following folks are truly heroic. I owe them a debt of gratitude for all the help

Contributors

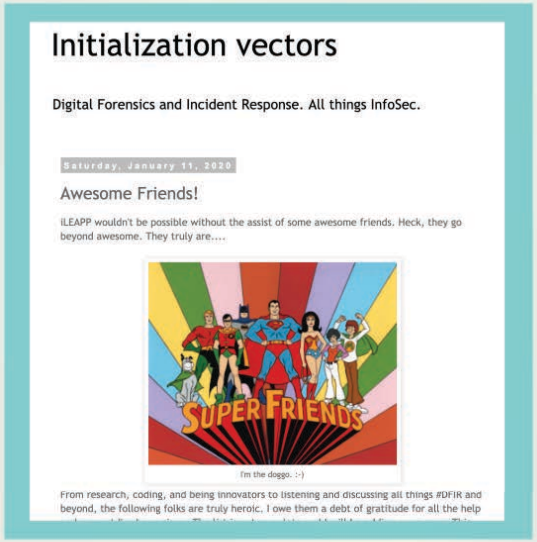
Super Friends:



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
DFIR300

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


DEPARTMENT OF JUSTICE
FEDERAL BUREAU OF INVESTIGATION
FIDELITY
BRAVERY
INTEGRITY
For identification purposes only
Community


xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
What
When
Contact
Why
Where
Who
How

xLEAPP

iOS & Android Artifact Parsers

iLEAPP - iOS Logs, Events, and Preferences Parser

ALEAPP - Android Logs, Events, and Protobuf Parser

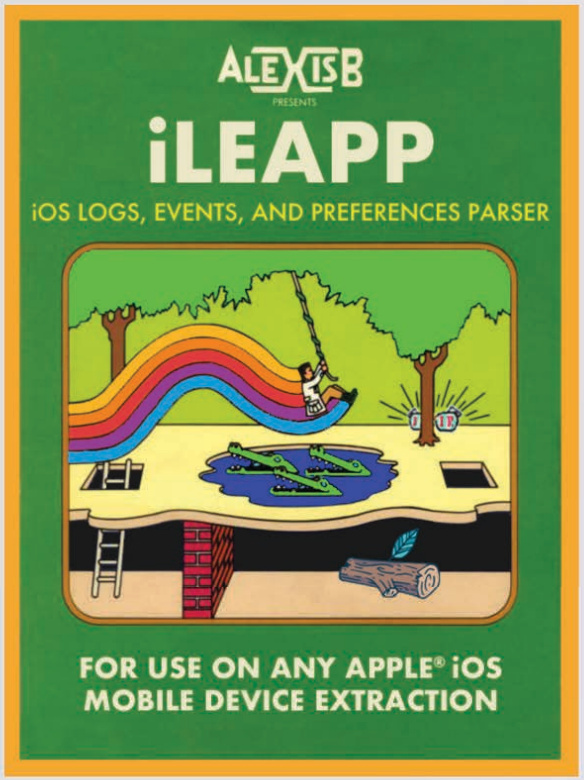
xLEAPP

iOS & Android Artifact Parsers

iLEAPP - iOS Logs, Events, and Preferences Parser

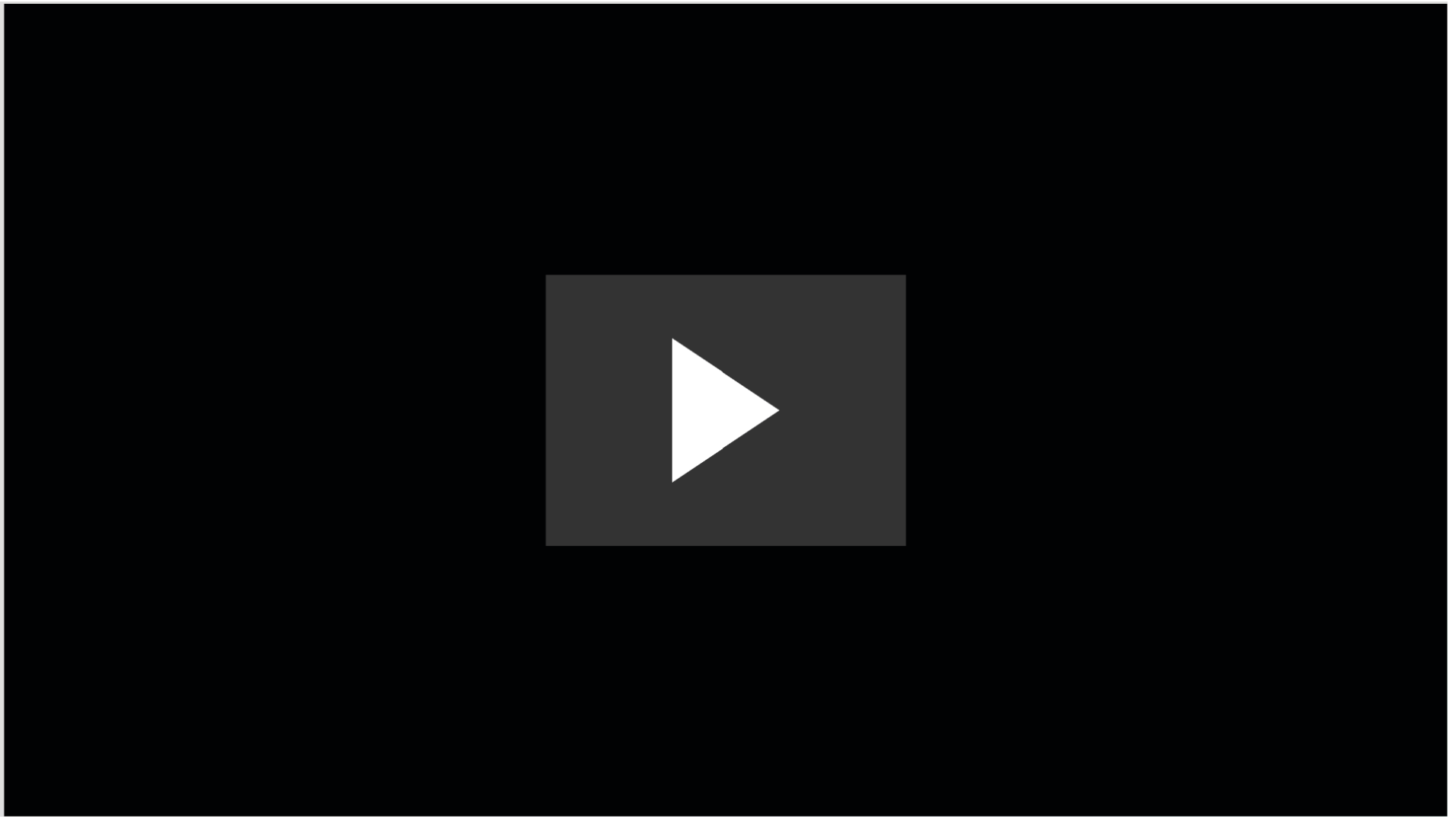
ALEAPP - Android Logs, Events, and Protobuf Parser

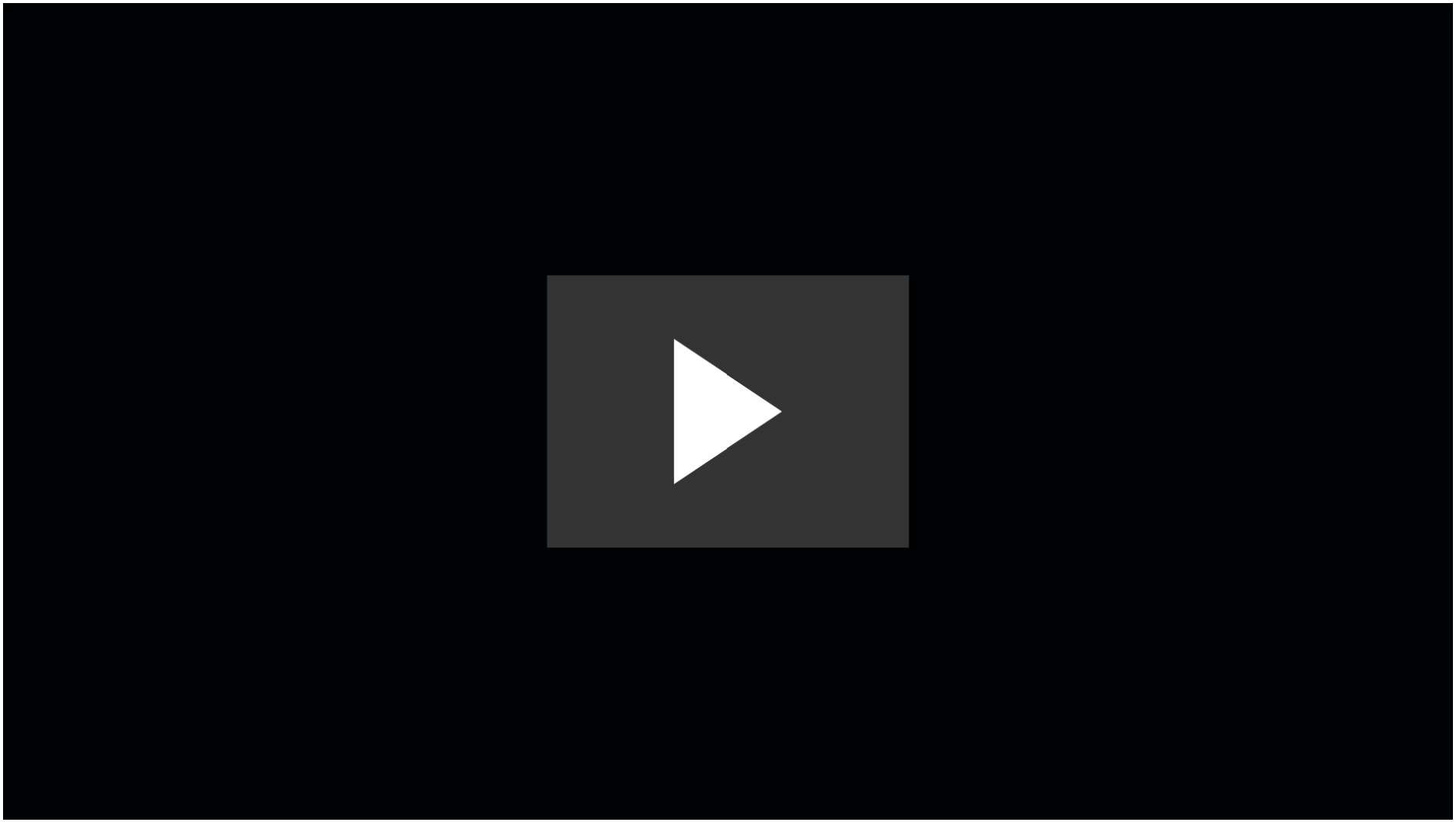
iLEAPP

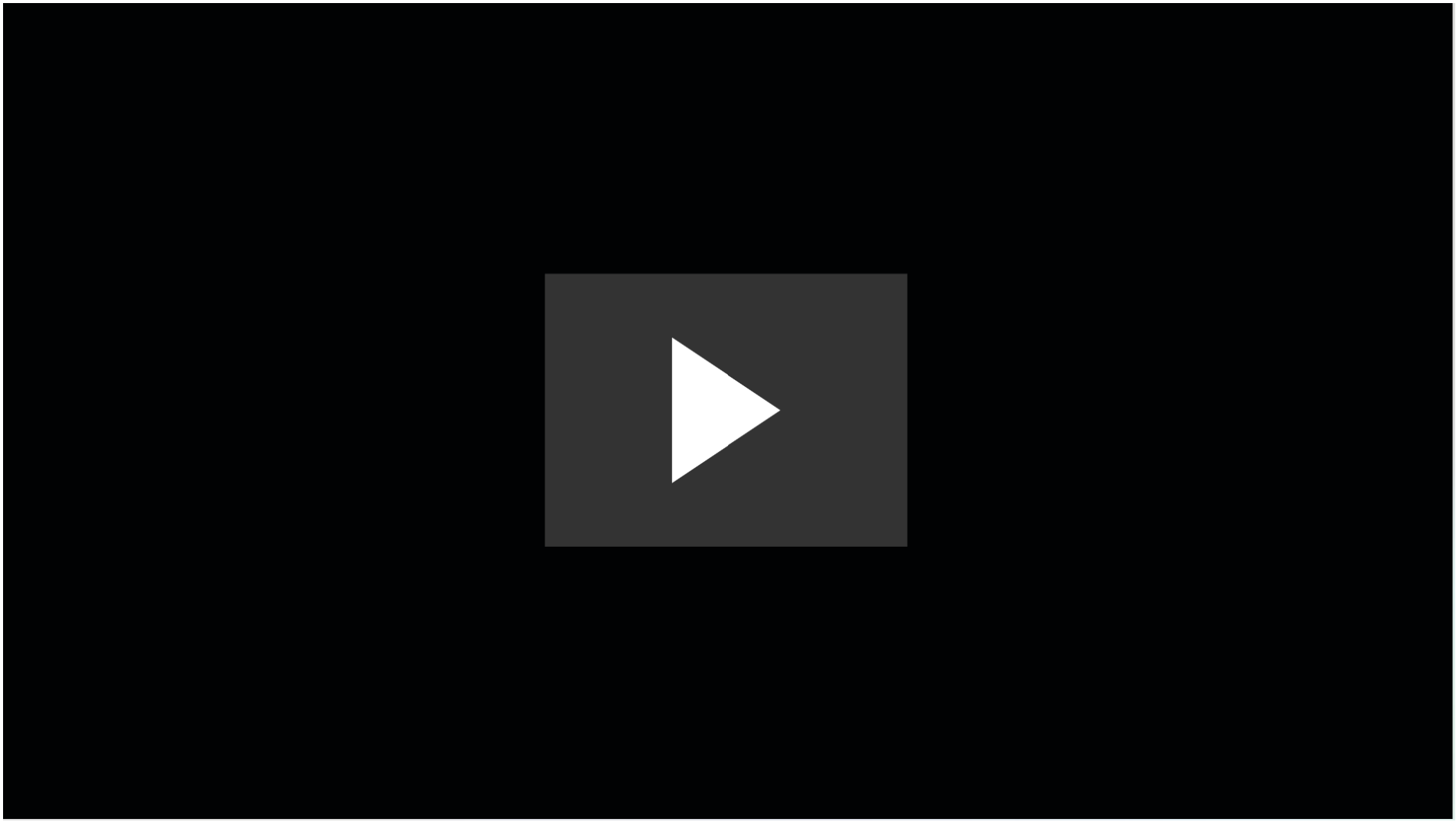


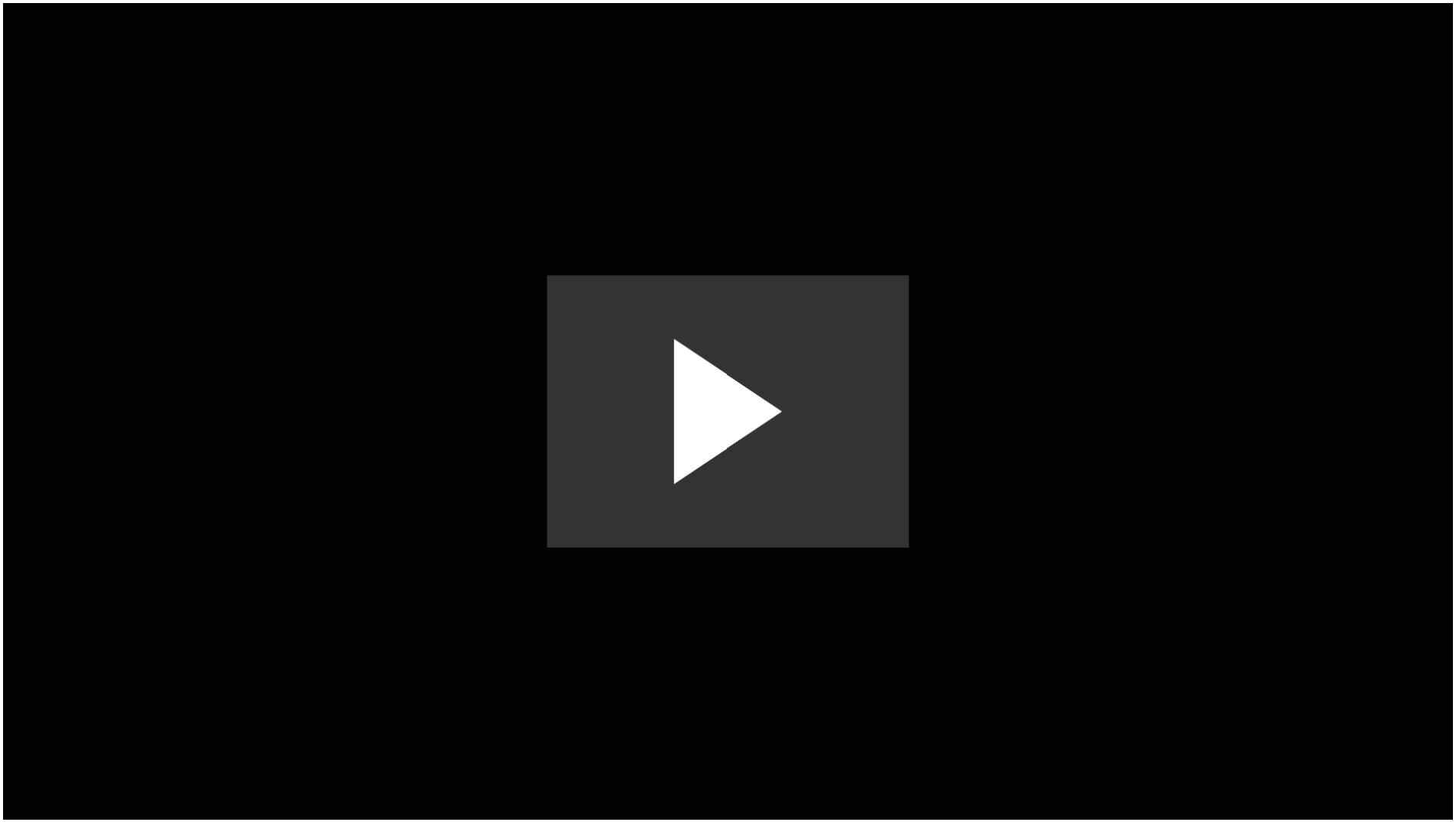
iOS Logs, Events, and Preferences Parser

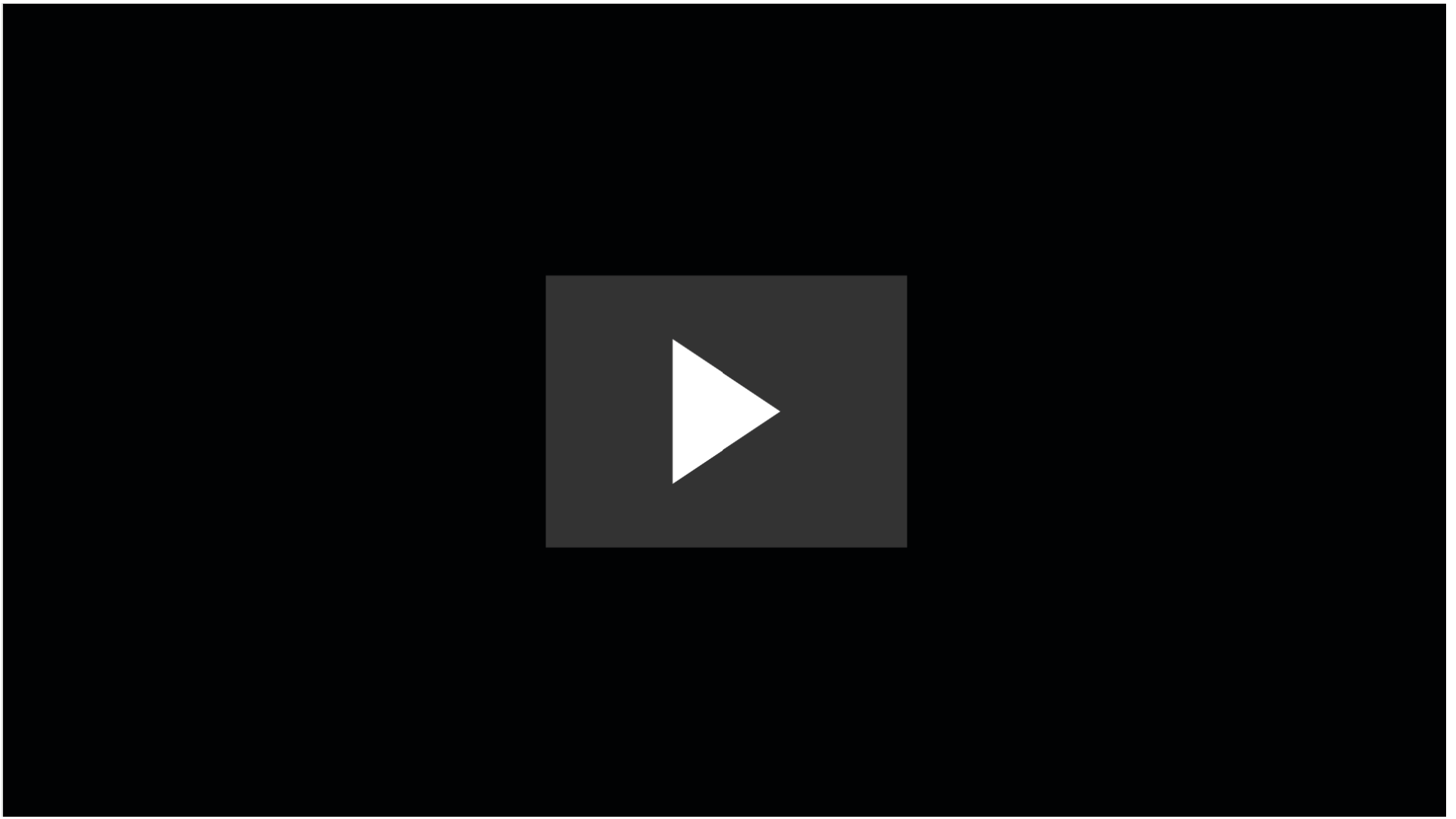
DEMO

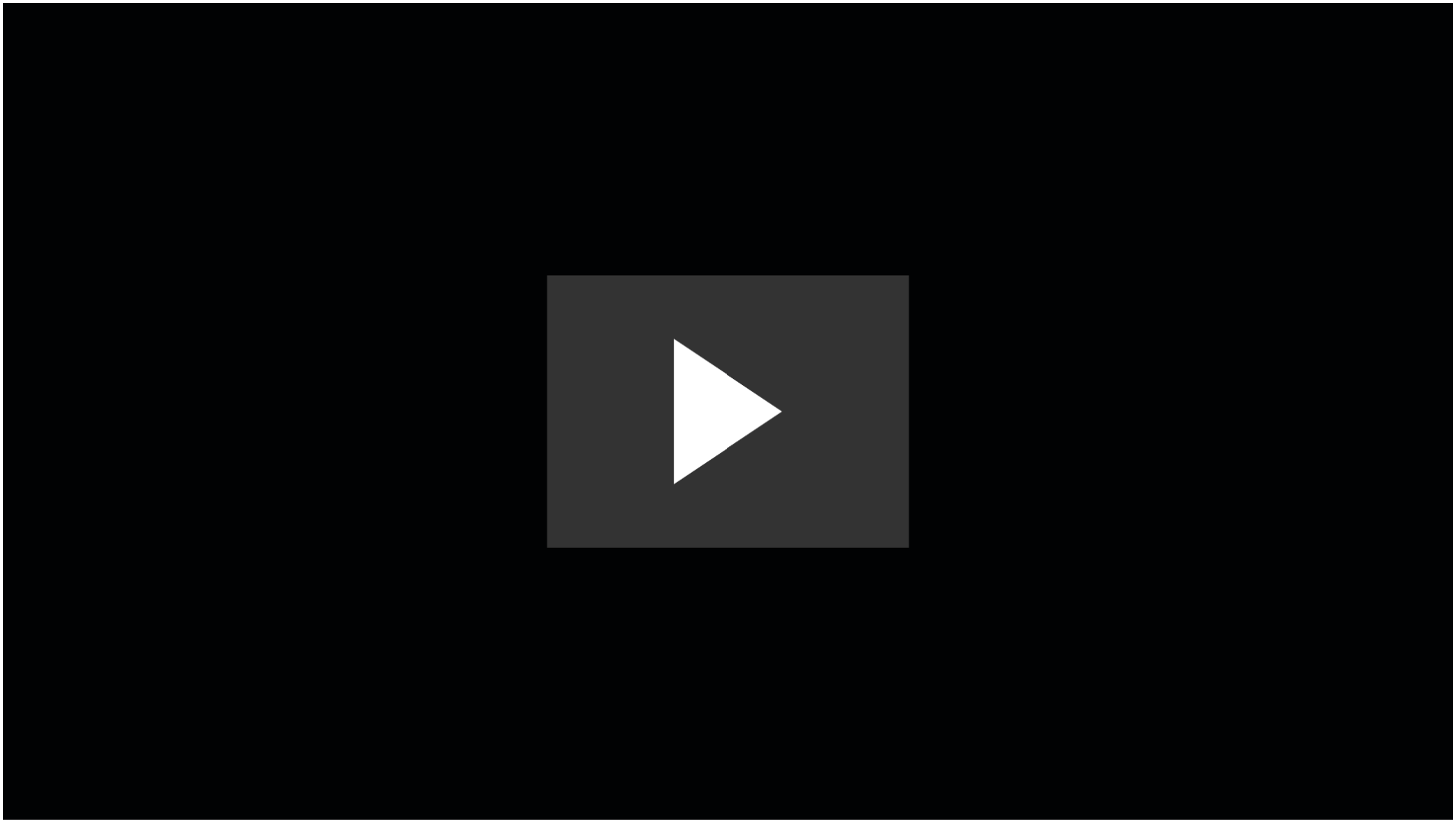


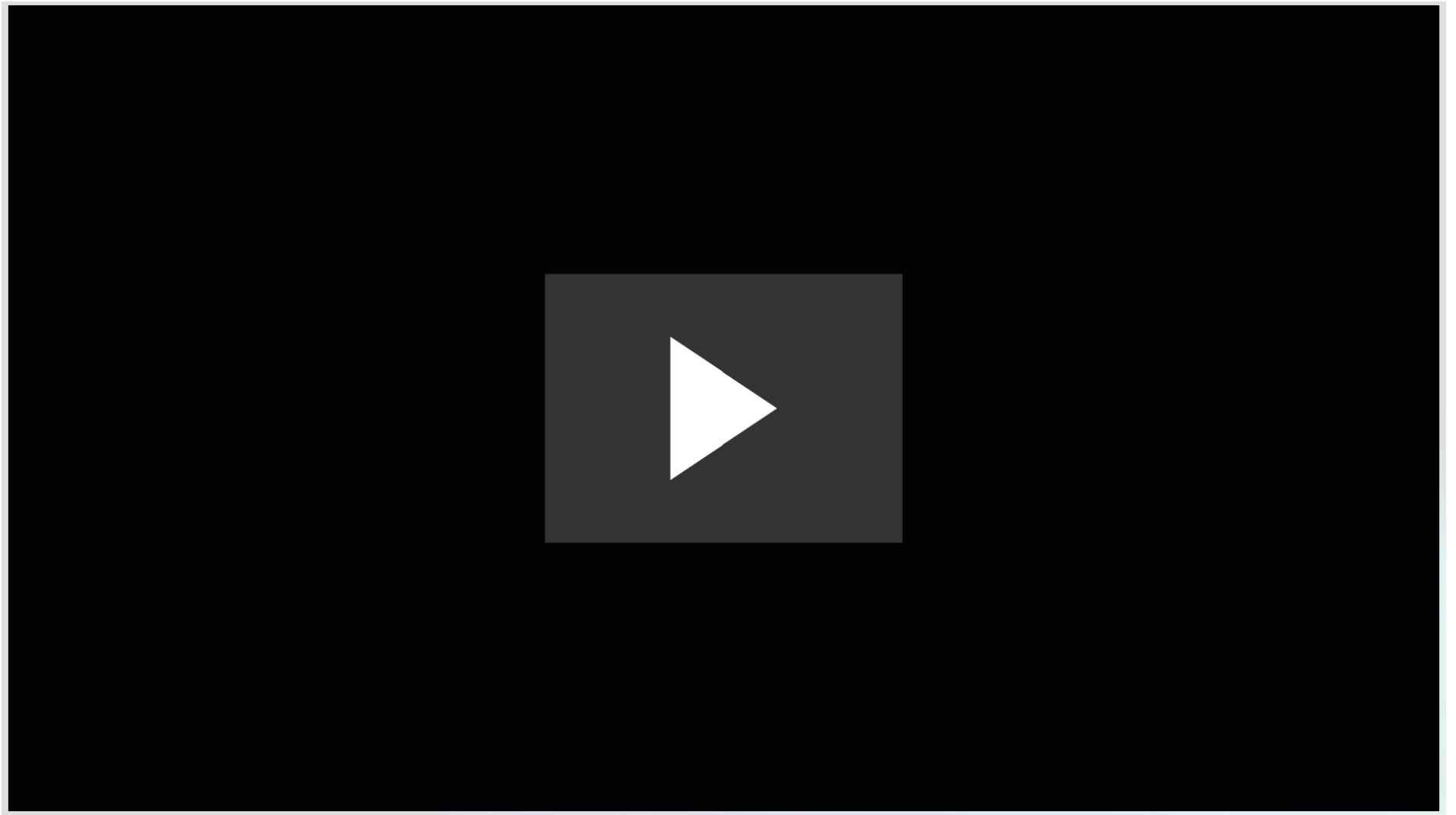


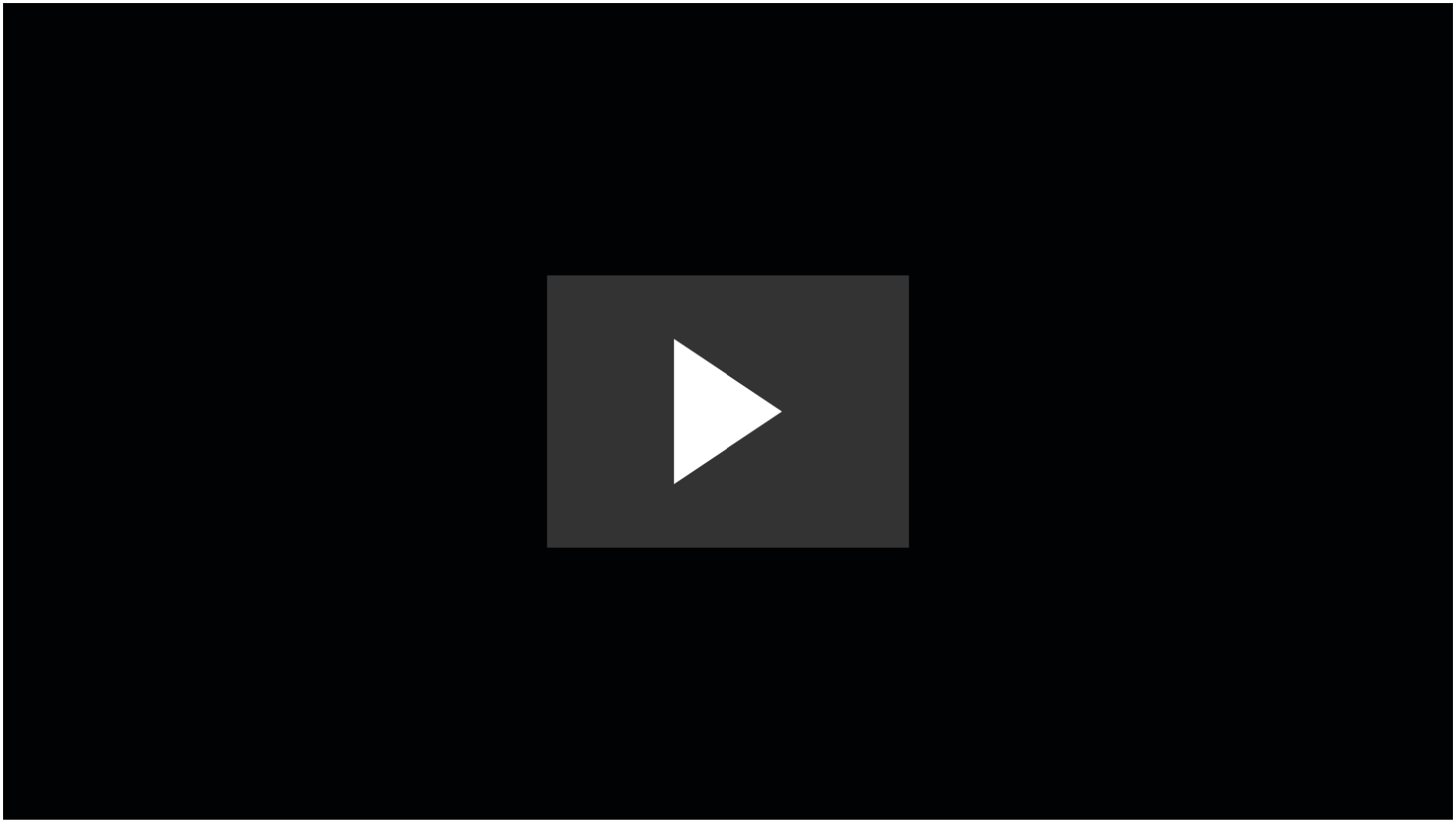








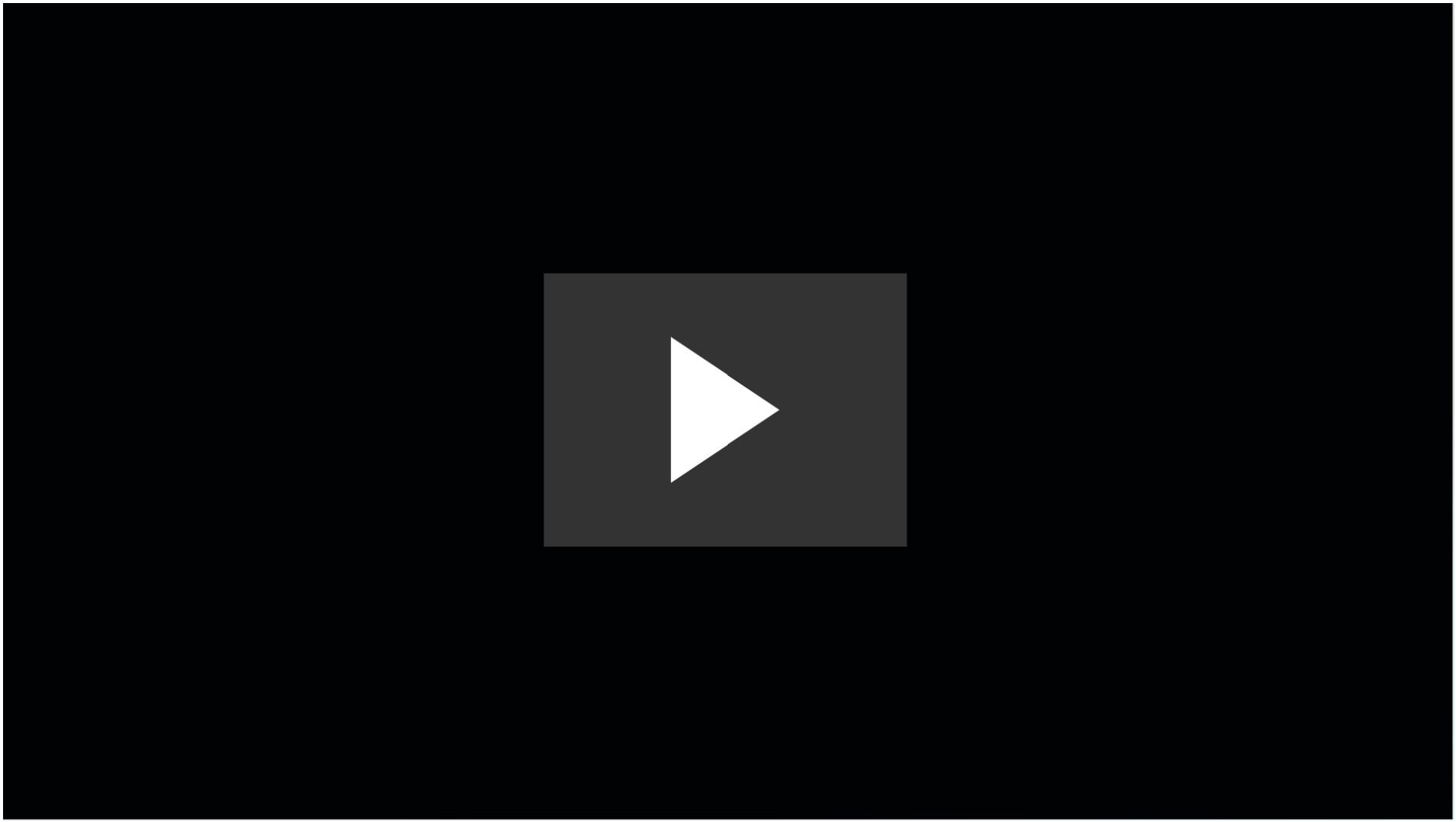


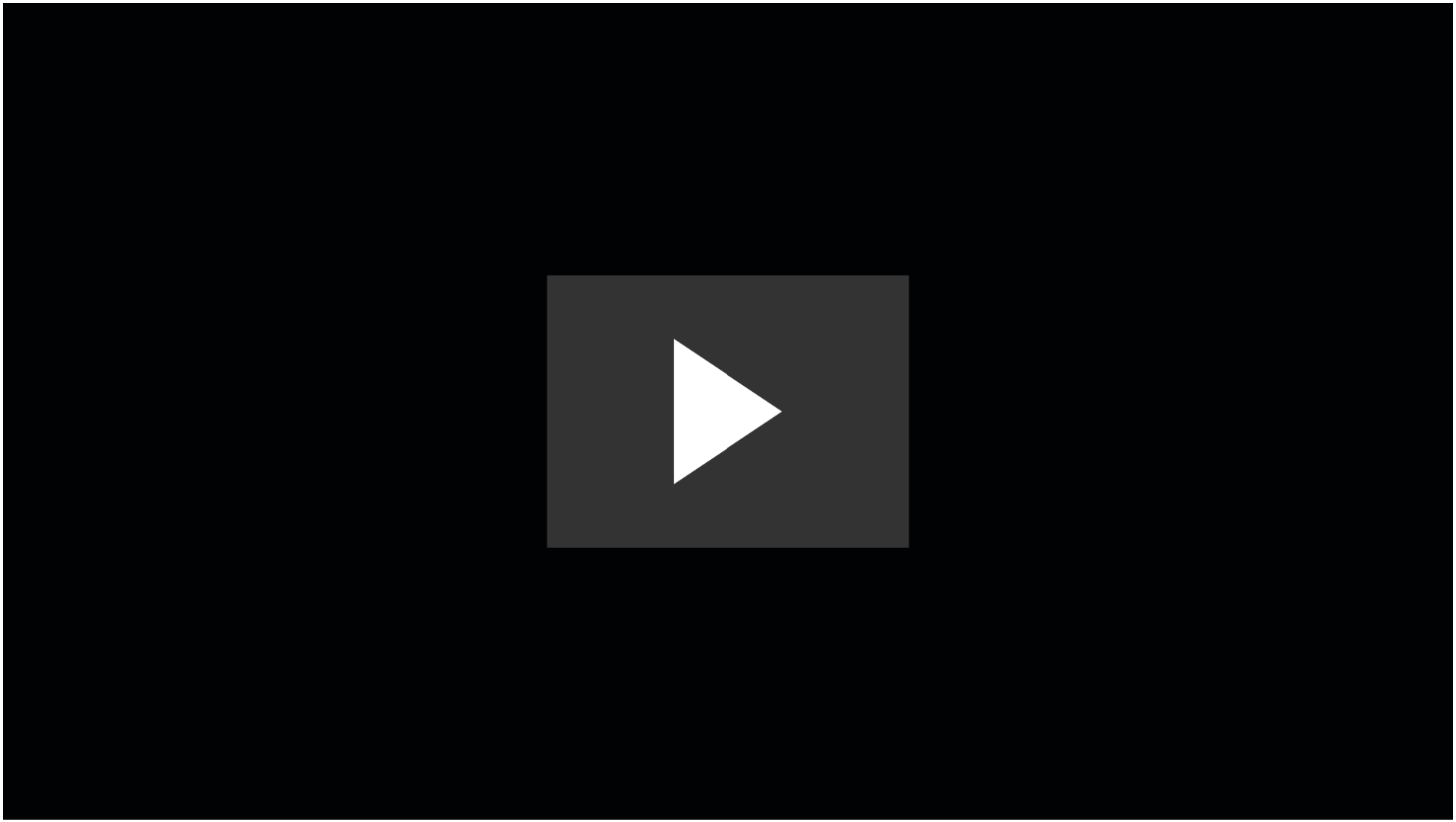


ALEAPP

Android Logs, Events, and Protobuf Parser

DEMO






xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
What
When
Contact
Why
Where
Who
How

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.

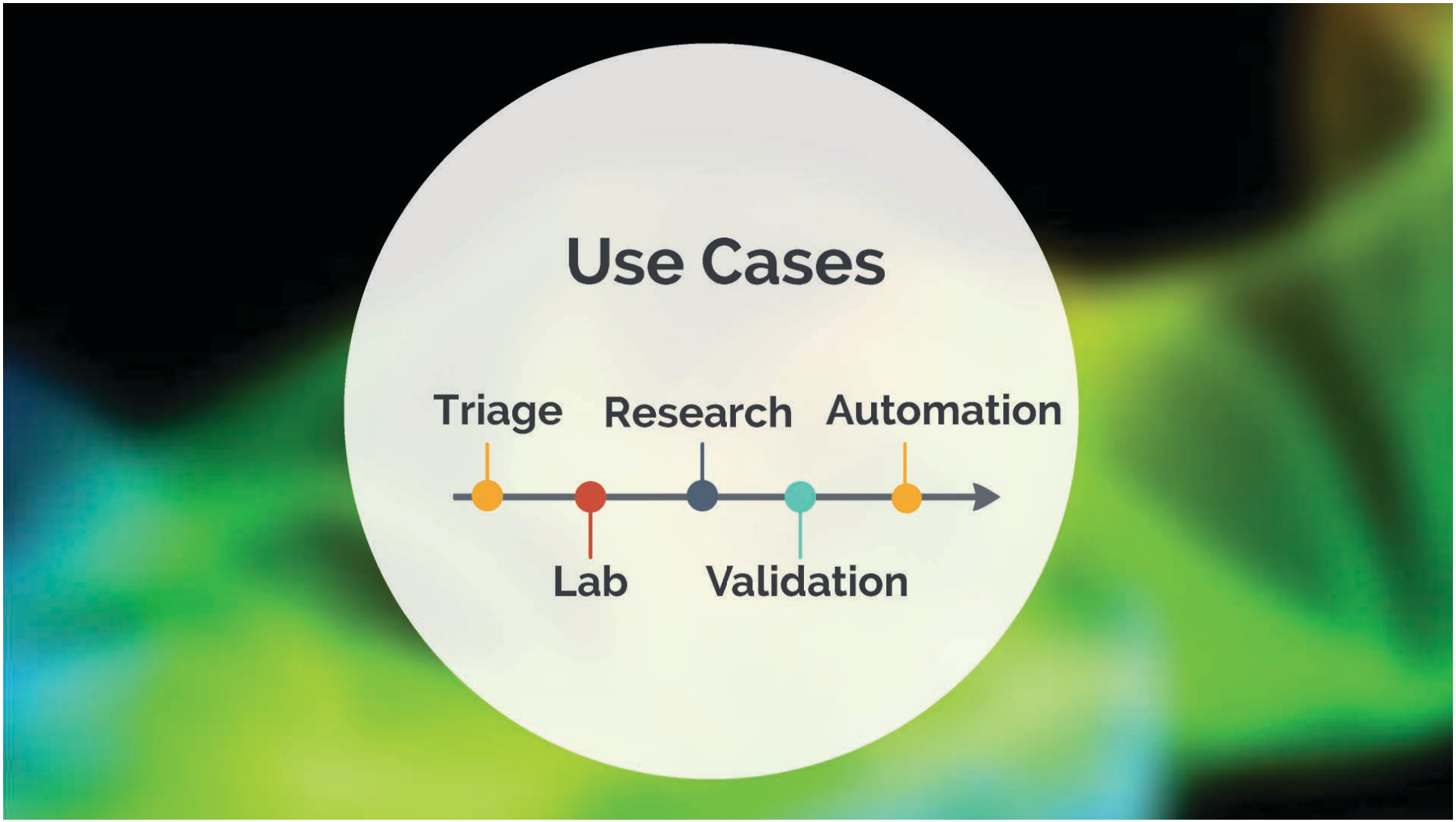
xLEAPP

iOS & Android Artifact Parsers

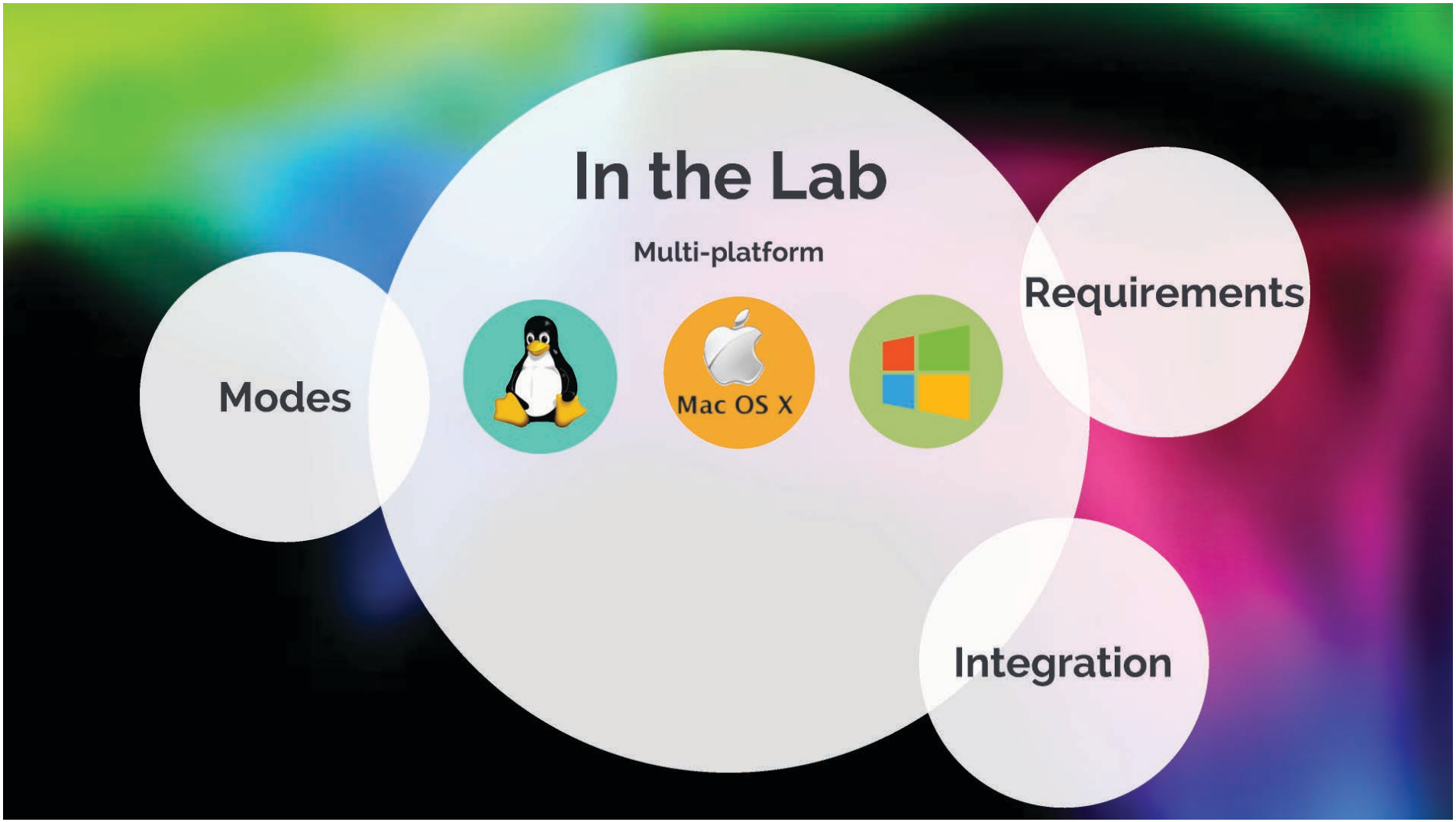
Raison d'être

- 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
What
When
Contact
Why
Where
Who
How


Use Cases
Triage
Research
Automation
Lab
Validation


xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
What
When
Contact
Why
Where
Who
How


In the Lab
Multi-platform
Mac OS X
Modes
Requirements
Integration

CMD Line

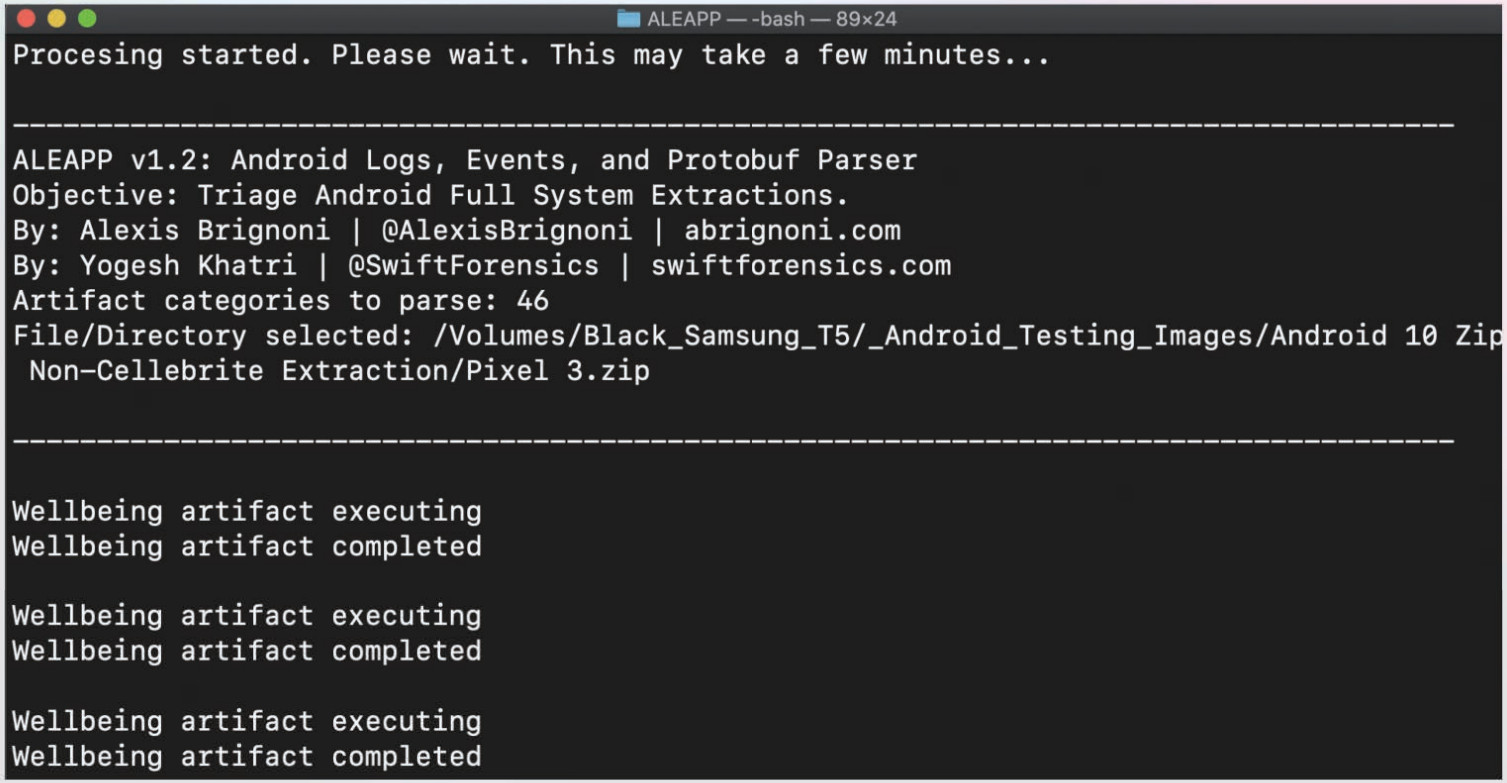
CMD Line

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


ALEAPP — -bash — 89×24
Procesing 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

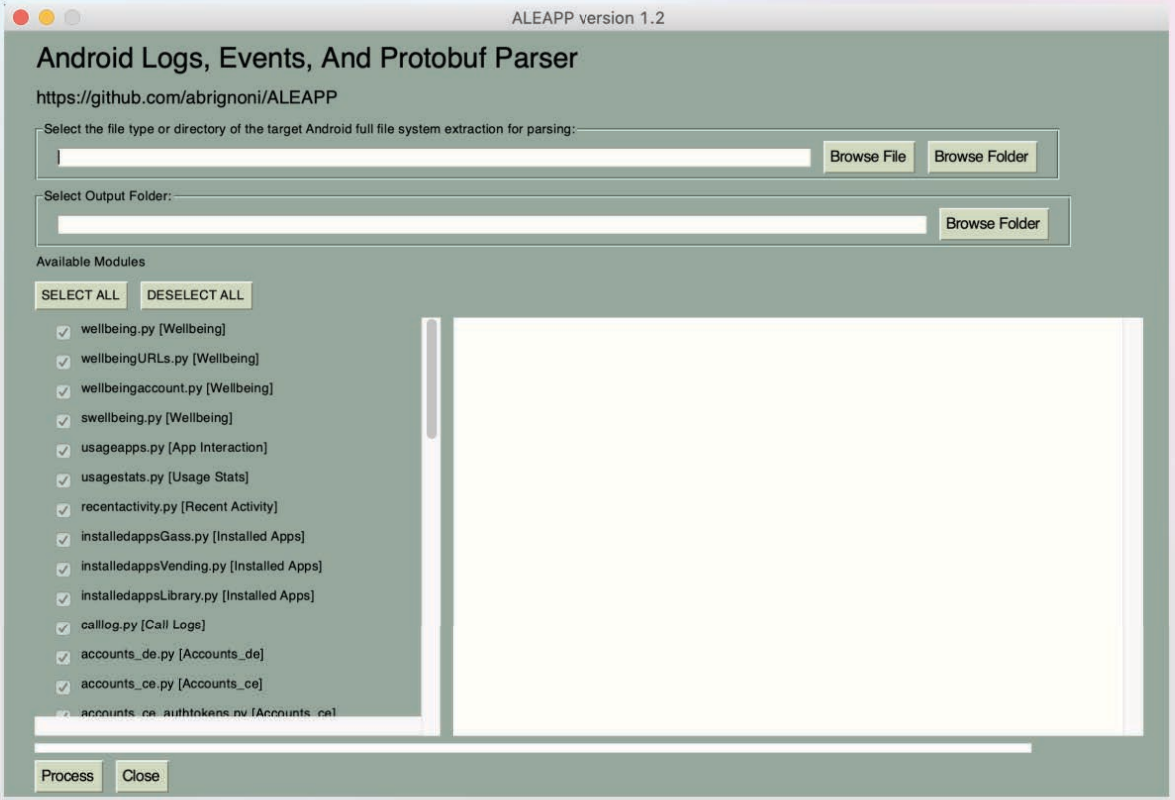
CMD Line

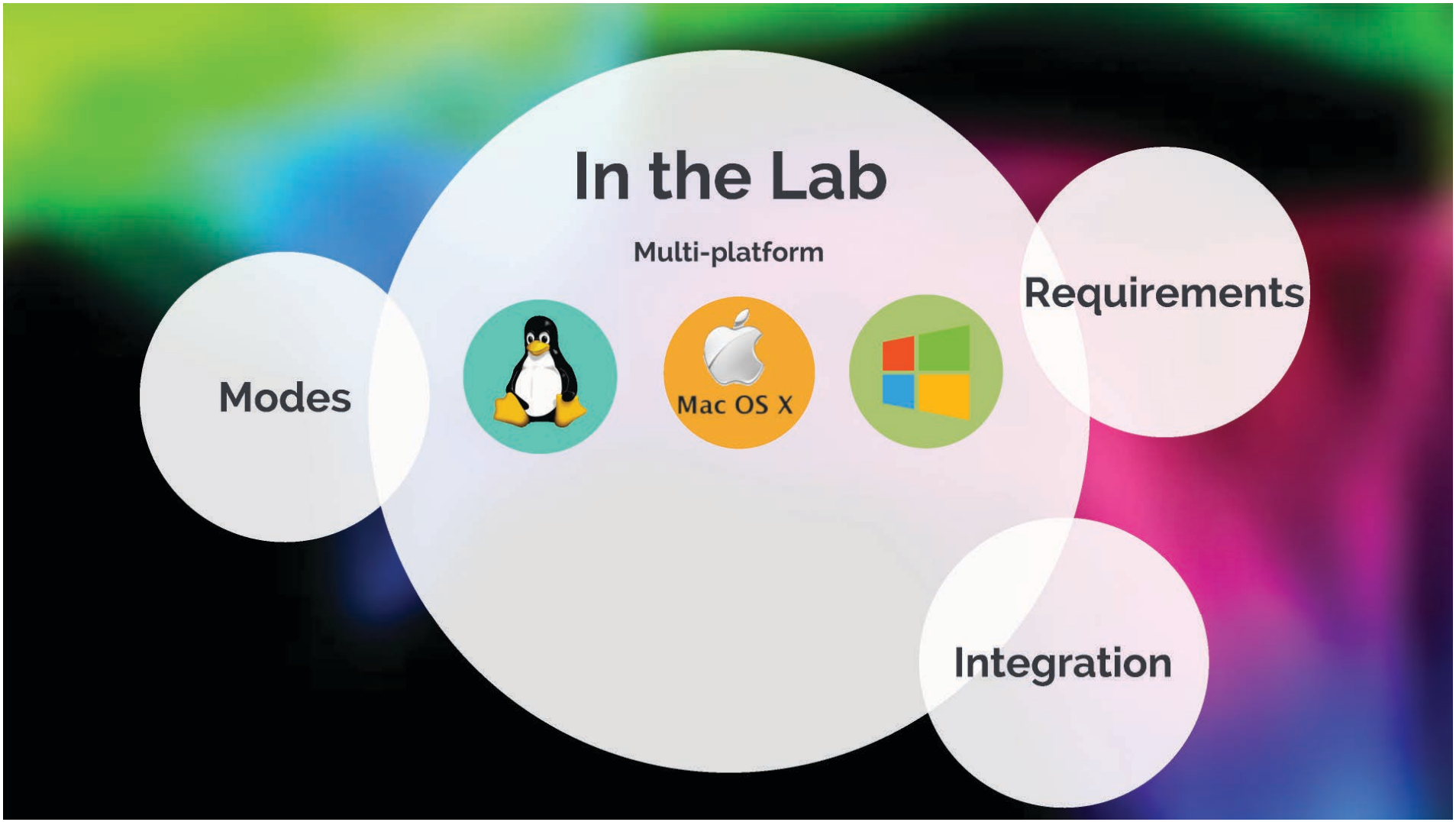
CMD Line

```
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

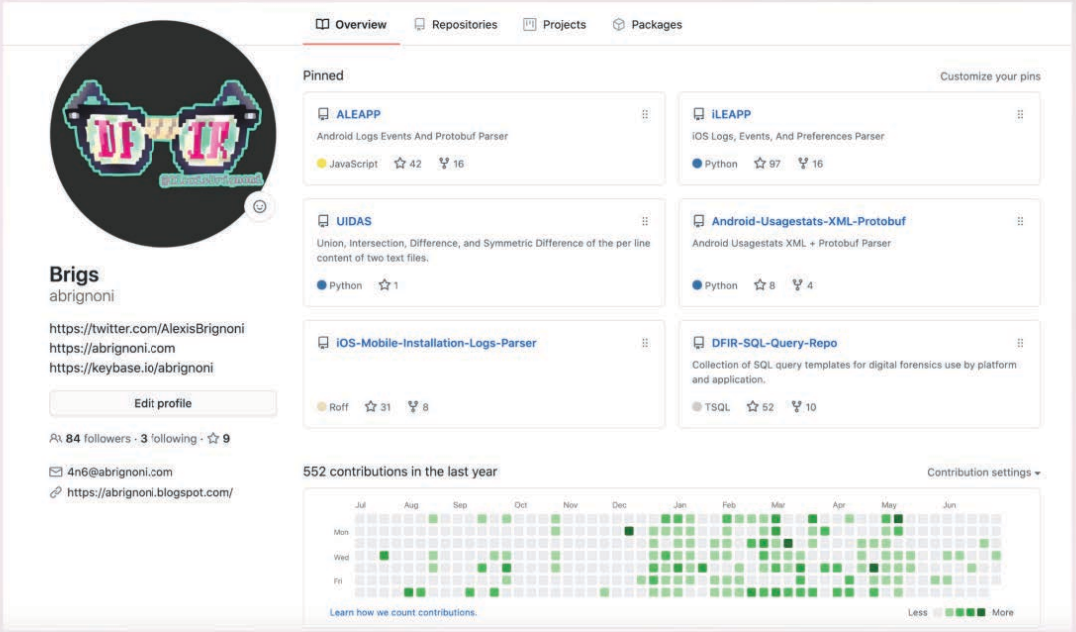
GUI


ALEAPP version 1.2
Android Logs, Events, And Protobuf Parser
https://github.com/abrignoni/ALEAPP
Select the file type or directory of the target Android full file system extraction for parsing:
Browse File
Browse Folder
Select Output Folder:
Browse Folder
Available Modules
SELECT ALL
DESELECT ALL
wellbeing.py [Wellbeing]
wellbeingURLs.py [Wellbeing]
wellbeingaccount.py [Wellbeing]
swellbeing.py [Wellbeing]
usageapps.py [App Interaction]
usagestats.py [Usage Stats]
recentactivity.py [Recent Activity]
installedappsGass.py [Installed Apps]
installedappsVending.py [Installed Apps]
installedappsLibrary.py [Installed Apps]
calllog.py [Call Logs]
accounts_de.py [Accounts_de]
accounts_ce.py [Accounts_ce]
Process
Close


In the Lab
Multi-platform
Mac OS X
Modes
Requirements
Integration

Requirements

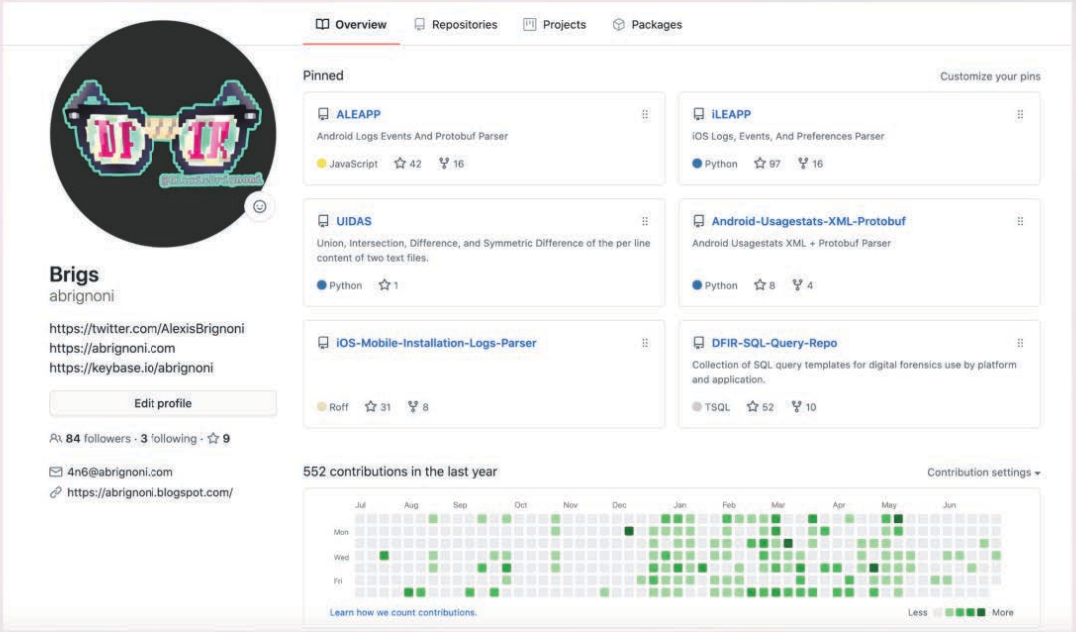
- Python 3
- Pip install -r requirements.txt
  - beautifulsoup4==4.8.2
  - protobuf==3.10.0
  - PySimpleGUI==4.16.0
  - PyCryptodome
  - packaging==20.1
  - pathlib2==2.3.5
  - PySimpleGUI==4.16.0
  - biplist
  - More...



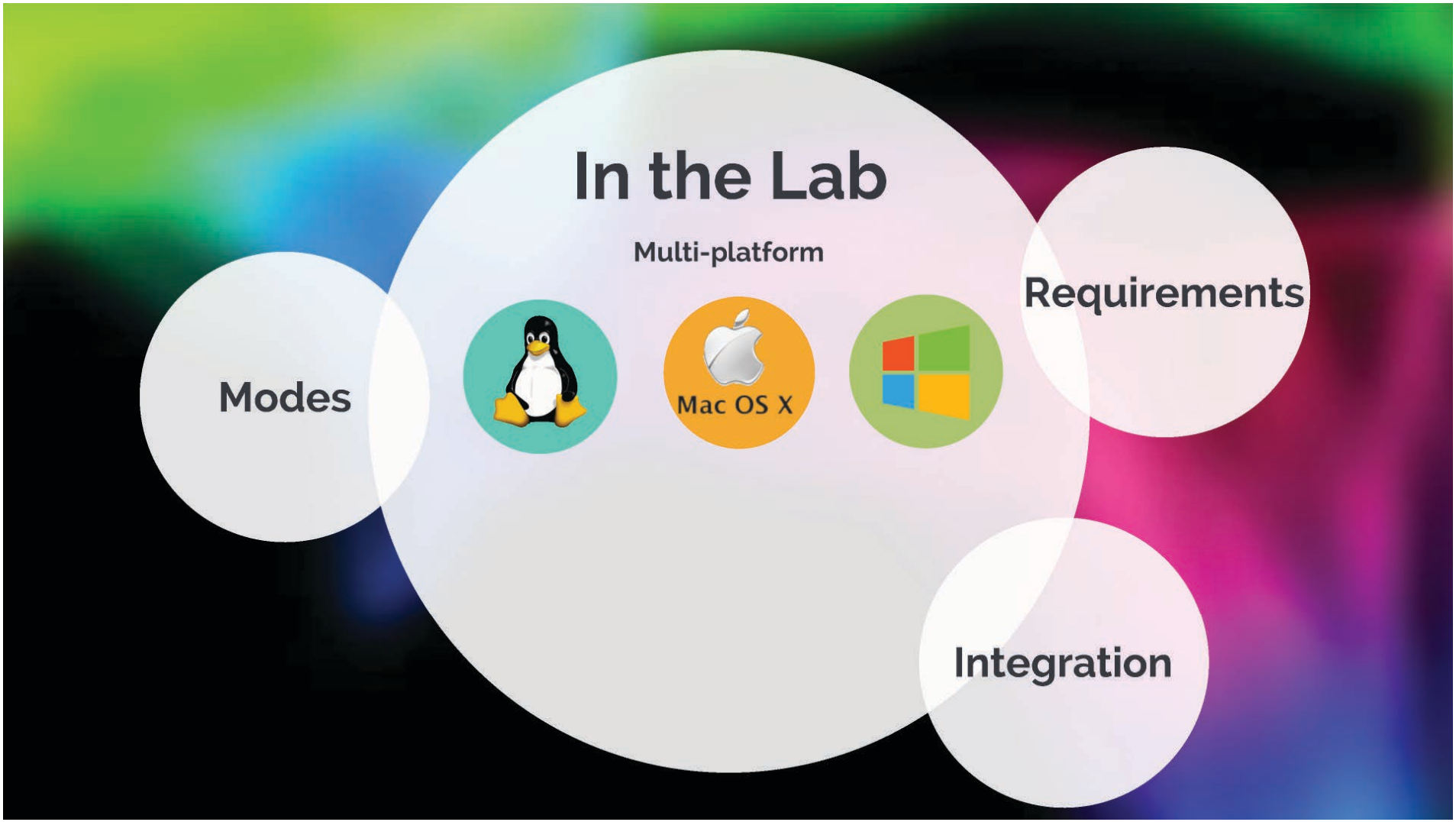
https://github.com/abrignoni

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
  - biplist
  - More...

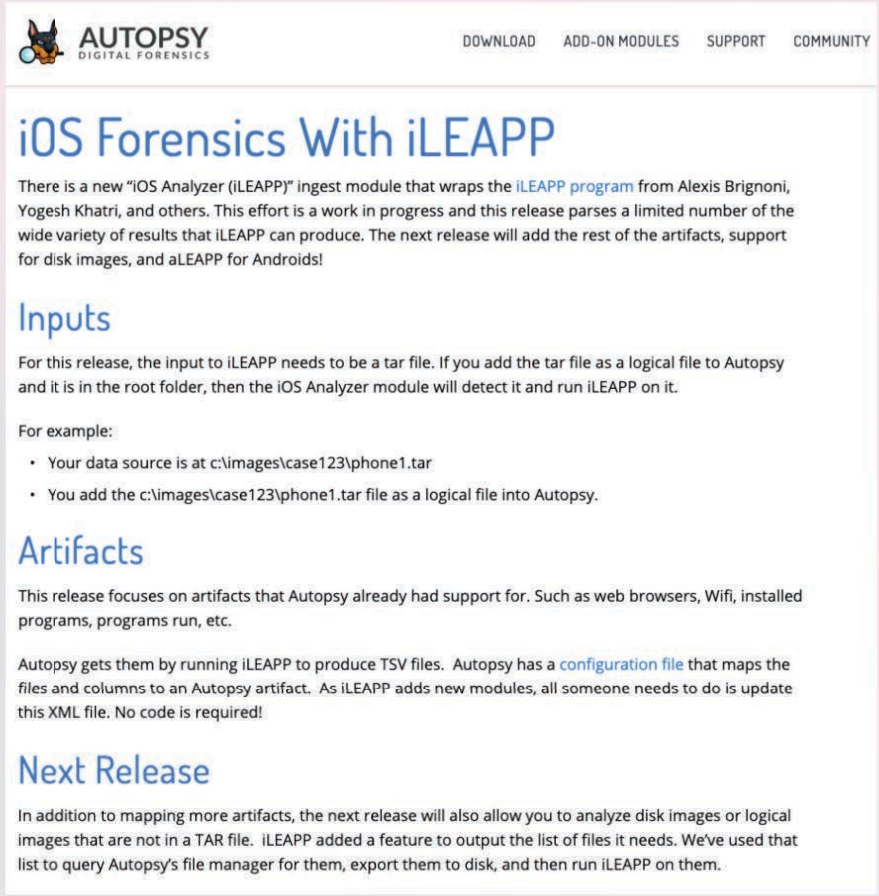


https://github.com/abrignoni


In the Lab
Multi-platform
Mac OS X
Modes
Requirements
Integration

Autopsy 4.17

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



AUTOPSY
DIGITAL FORENSICS

DOWNLOAD ADD-ON MODULES SUPPORT COMMUNITY

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.

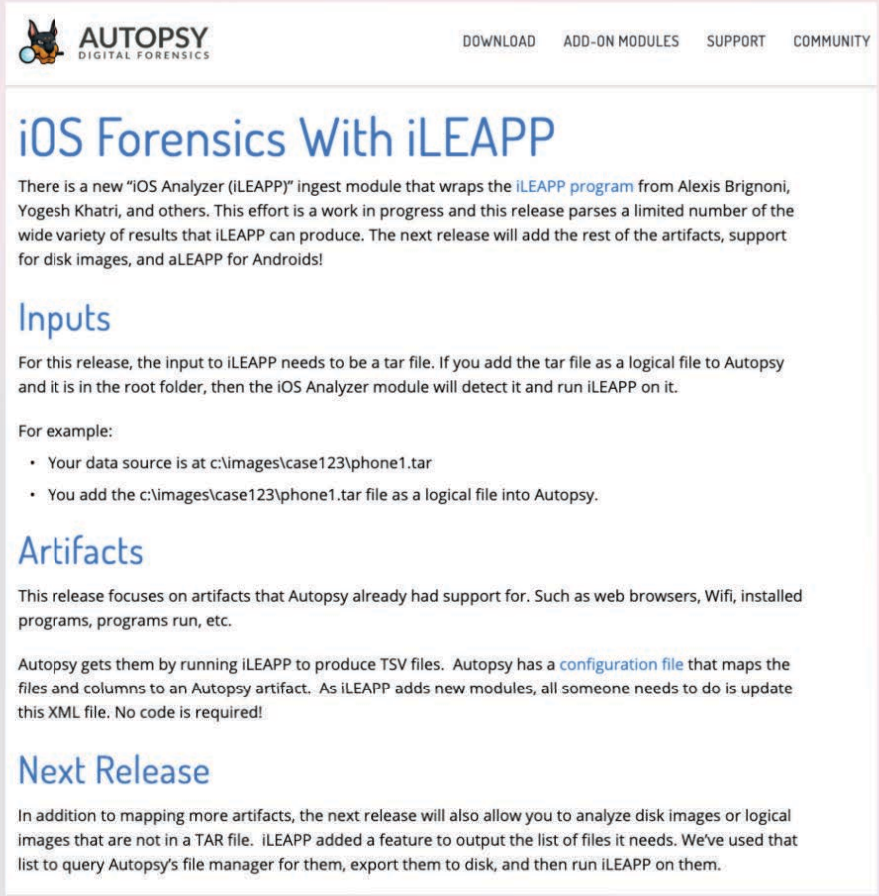
Autopsy gets them by running iLEAPP to produce TSV files. Autopsy has a configuration file that maps the files and columns to an Autopsy artifact. As iLEAPP adds new modules, all someone needs to do is update this XML file. No code is required!

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



AUTOPSY
DIGITAL FORENSICS

DOWNLOAD ADD-ON MODULES SUPPORT COMMUNITY

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.

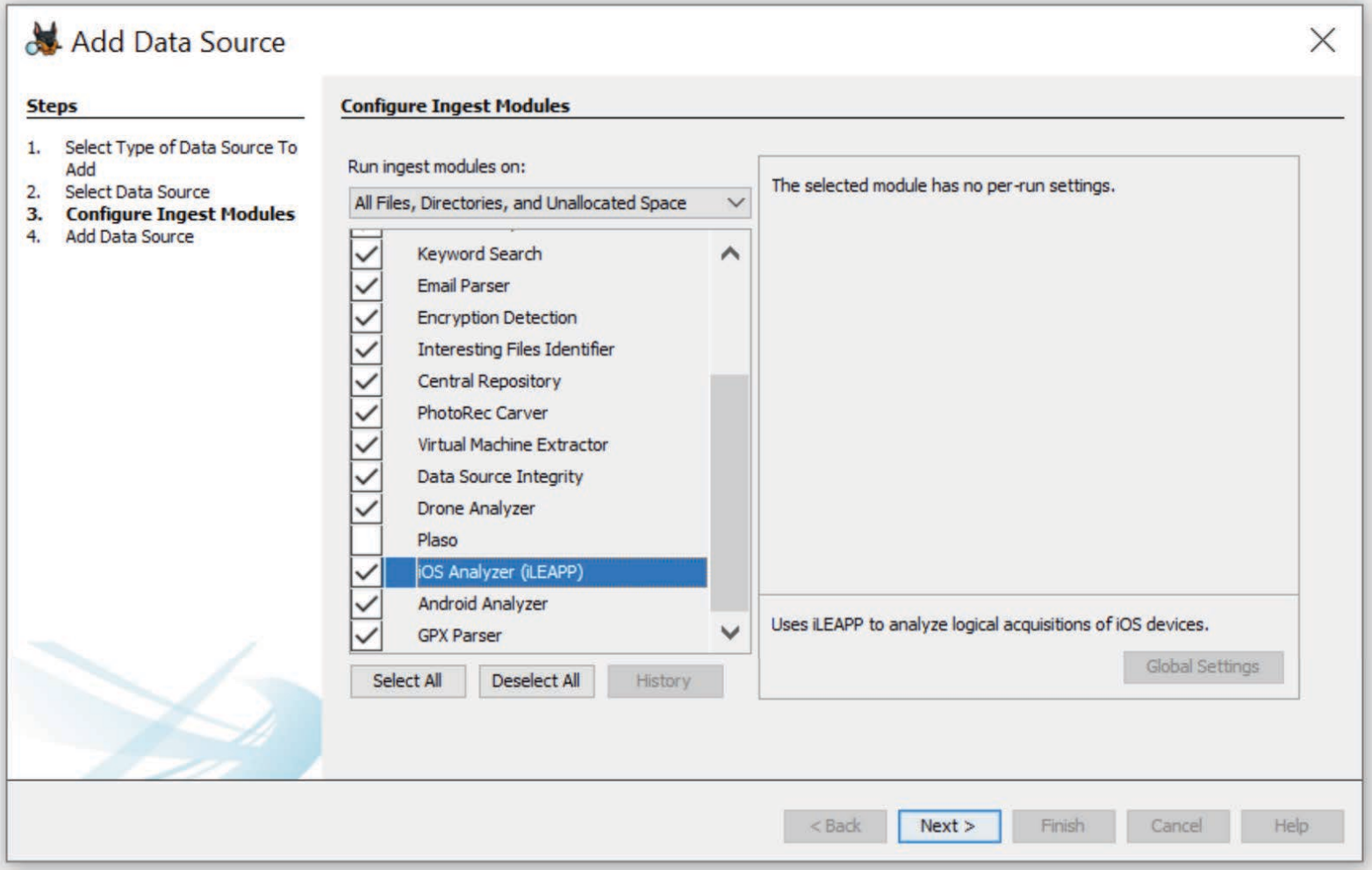
Artifacts

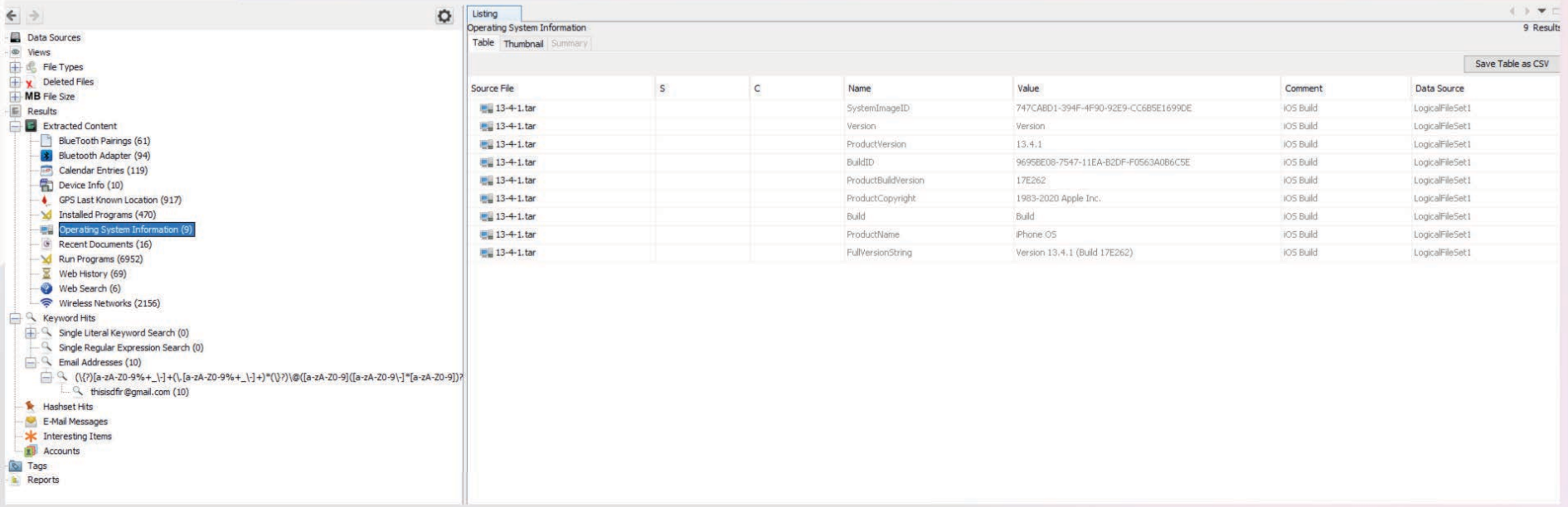
This release focuses on artifacts that Autopsy already had support for. Such as web browsers, Wifi, installed programs, programs run, etc.

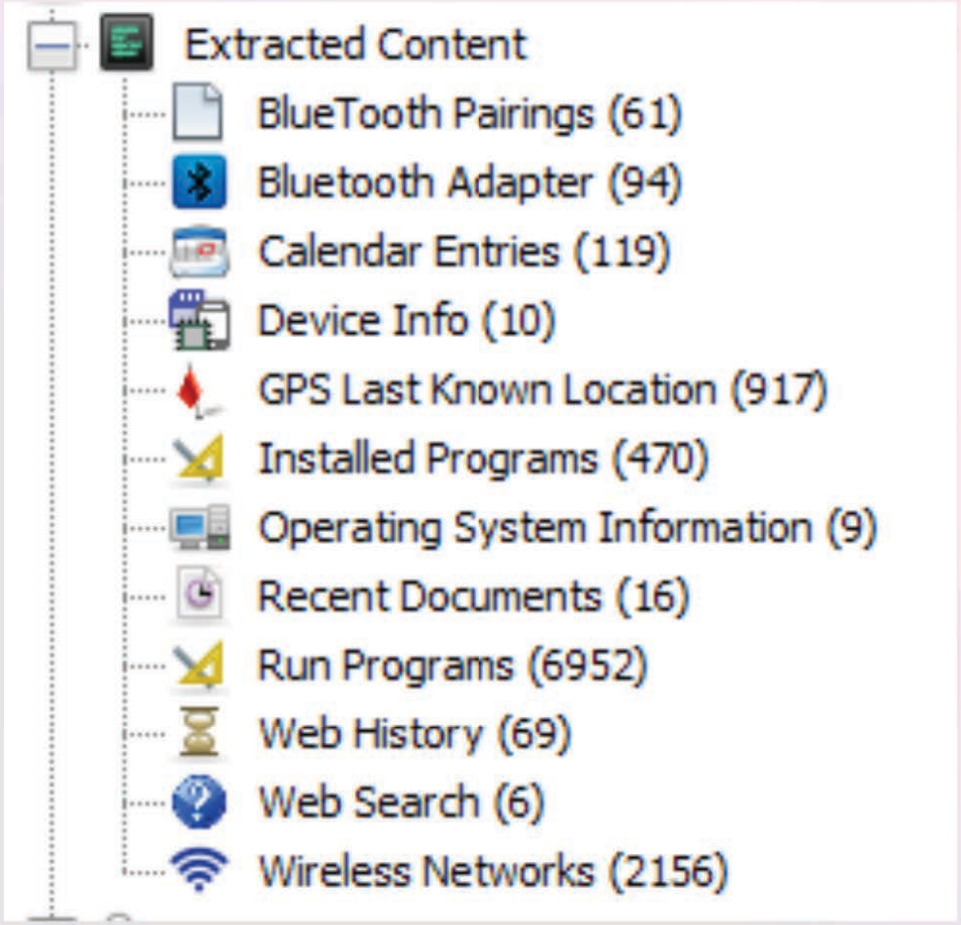
Autopsy gets them by running iLEAPP to produce TSV files. Autopsy has a configuration file that maps the files and columns to an Autopsy artifact. As iLEAPP adds new modules, all someone needs to do is update this XML file. No code is required!

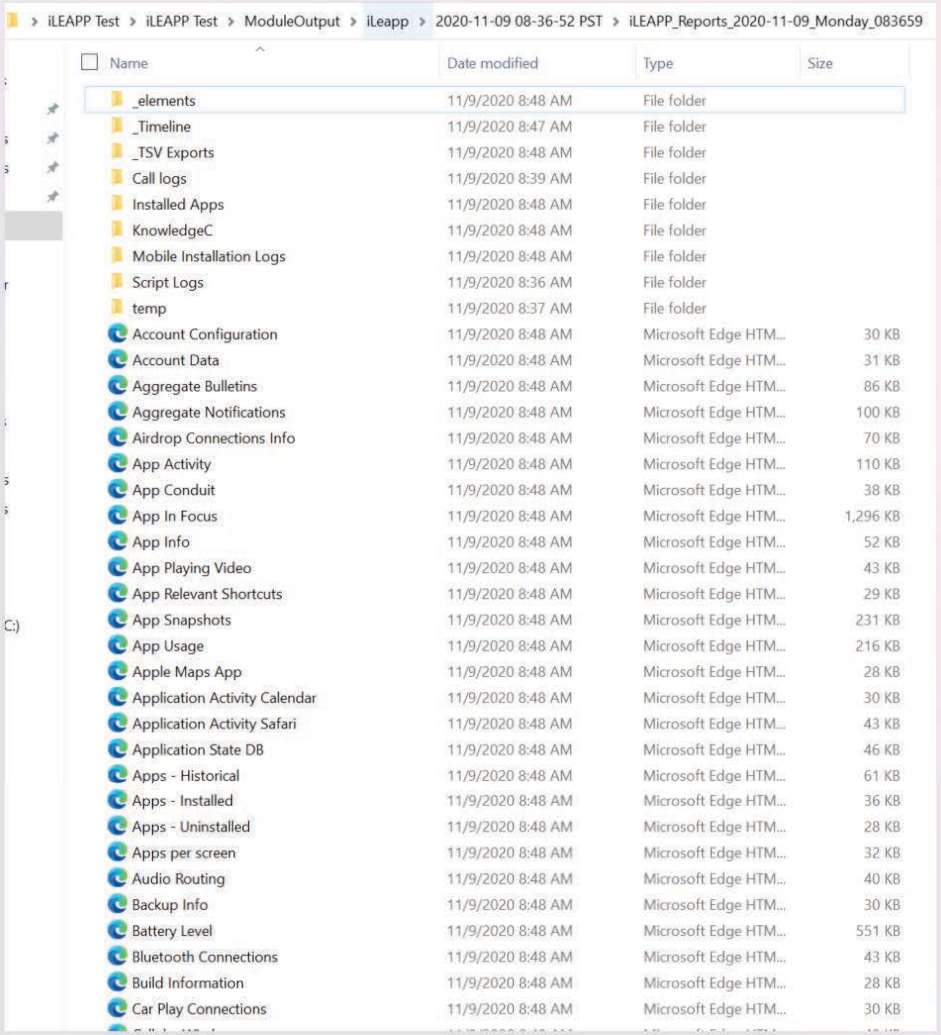
Next Release

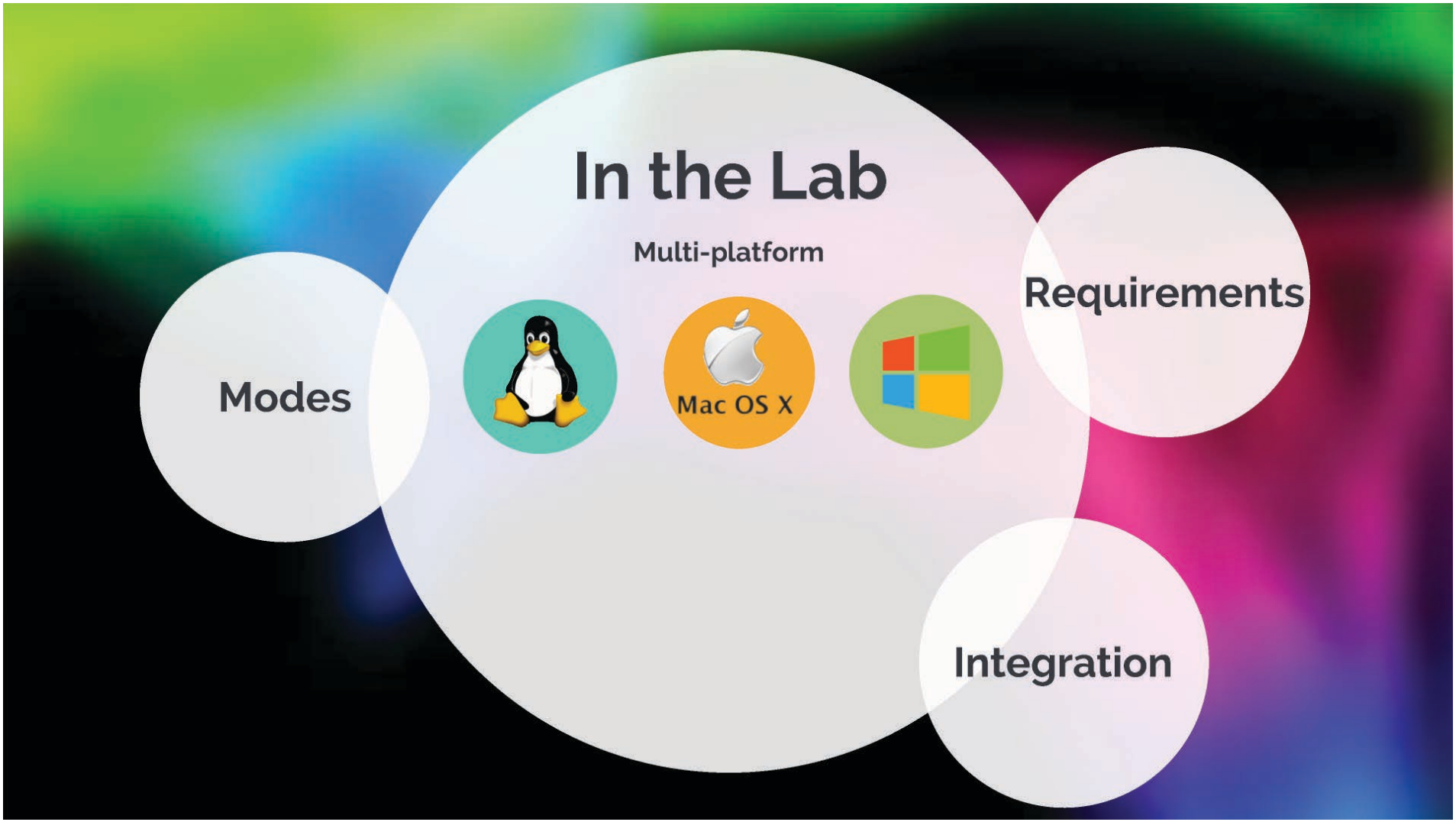
In addition to mapping more artifacts, the next release will also allow you to analyze disk images or logical images that are not in a TAR file. iLEAPP added a feature to output the list of files it needs. We've used that list to query Autopsy's file manager for them, export them to disk, and then run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


Add Data Source
Steps
1. Select Type of Data Source To Add
2. Select Data Source
3. Configure Ingest Modules
4. Add Data Source
Configure Ingest Modules
Run ingest modules on:
All Files, Directories, and Unallocated Space
Keyword Search
Email Parser
Encryption Detection
Interesting Files Identifier
Central Repository
PhotoRec Carver
Virtual Machine Extractor
Data Source Integrity
Drone Analyzer
Plaso
iOS Analyzer (iLEAPP)
Android Analyzer
GPX Parser
Select All
Deselect All
History
The selected module has no per-run settings.
Uses iLEAPP to analyze logical acquisitions of iOS devices.
Global Settings
< Back
Next >
Finish
Cancel
Help


Data Sources
Views
File Types
Deleted Files
MB File Size
Results
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)
Keyword Hits
Single Literal Keyword Search (0)
Single Regular Expression Search (0)
Email Addresses (10)
thisisdfir@gmail.com (10)
Hashset Hits
E-Mail Messages
Interesting Items
Accounts
Tags
Reports
Listing
Operating System Information
9 Results
Table
Thumbnail
Summary
Save Table as CSV
Source File
S
C
Name
Value
Comment
Data Source
13-4-1.tar
SystemImageID
747CABD1-394F-4F90-92E9-CC6B5E1699DE
iOS Build
LogicalFileSet1
13-4-1.tar
Version
Version
iOS Build
LogicalFileSet1
13-4-1.tar
ProductVersion
13.4.1
iOS Build
LogicalFileSet1
13-4-1.tar
BuildID
9695BE08-7547-11EA-B2DF-F0563A0B6C5E
iOS Build
LogicalFileSet1
13-4-1.tar
ProductBuildVersion
17E262
iOS Build
LogicalFileSet1
13-4-1.tar
ProductCopyright
1983-2020 Apple Inc.
iOS Build
LogicalFileSet1
13-4-1.tar
Build
Build
iOS Build
LogicalFileSet1
13-4-1.tar
ProductName
iPhone OS
iOS Build
LogicalFileSet1
13-4-1.tar
FullVersionString
Version 13.4.1 (Build 17E262)
iOS Build
LogicalFileSet1


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)


iLEAPP Test › iLEAPP Test › ModuleOutput › iLeapp › 2020-11-09 08-36-52 PST › iLEAPP_Reports_2020-11-09_Monday_083659
Name
Date modified
Type
Size
_elements 11/9/2020 8:48 AM File folder
_Timeline 11/9/2020 8:47 AM File folder
_TSV Exports 11/9/2020 8:48 AM File folder
Call logs 11/9/2020 8:39 AM File folder
Installed Apps 11/9/2020 8:48 AM File folder
KnowledgeC 11/9/2020 8:48 AM File folder
Mobile Installation Logs 11/9/2020 8:48 AM File folder
Script Logs 11/9/2020 8:36 AM File folder
temp 11/9/2020 8:37 AM File folder
Account Configuration 11/9/2020 8:48 AM Microsoft Edge HTM... 30 KB
Account Data 11/9/2020 8:48 AM Microsoft Edge HTM... 31 KB
Aggregate Bulletins 11/9/2020 8:48 AM Microsoft Edge HTM... 86 KB
Aggregate Notifications 11/9/2020 8:48 AM Microsoft Edge HTM... 100 KB
Airdrop Connections Info 11/9/2020 8:48 AM Microsoft Edge HTM... 70 KB
App Activity 11/9/2020 8:48 AM Microsoft Edge HTM... 110 KB
App Conduit 11/9/2020 8:48 AM Microsoft Edge HTM... 38 KB
App In Focus 11/9/2020 8:48 AM Microsoft Edge HTM... 1,296 KB
App Info 11/9/2020 8:48 AM Microsoft Edge HTM... 52 KB
App Playing Video 11/9/2020 8:48 AM Microsoft Edge HTM... 43 KB
App Relevant Shortcuts 11/9/2020 8:48 AM Microsoft Edge HTM... 29 KB
App Snapshots 11/9/2020 8:48 AM Microsoft Edge HTM... 231 KB
App Usage 11/9/2020 8:48 AM Microsoft Edge HTM... 216 KB
Apple Maps App 11/9/2020 8:48 AM Microsoft Edge HTM... 28 KB
Application Activity Calendar 11/9/2020 8:48 AM Microsoft Edge HTM... 30 KB
Application Activity Safari 11/9/2020 8:48 AM Microsoft Edge HTM... 43 KB
Application State DB 11/9/2020 8:48 AM Microsoft Edge HTM... 46 KB
Apps - Historical 11/9/2020 8:48 AM Microsoft Edge HTM... 61 KB
Apps - Installed 11/9/2020 8:48 AM Microsoft Edge HTM... 36 KB
Apps - Uninstalled 11/9/2020 8:48 AM Microsoft Edge HTM... 28 KB
Apps per screen 11/9/2020 8:48 AM Microsoft Edge HTM... 32 KB
Audio Routing 11/9/2020 8:48 AM Microsoft Edge HTM... 40 KB
Backup Info 11/9/2020 8:48 AM Microsoft Edge HTM... 30 KB
Battery Level 11/9/2020 8:48 AM Microsoft Edge HTM... 551 KB
Bluetooth Connections 11/9/2020 8:48 AM Microsoft Edge HTM... 43 KB
Build Information 11/9/2020 8:48 AM Microsoft Edge HTM... 28 KB
Car Play Connections 11/9/2020 8:48 AM Microsoft Edge HTM... 30 KB


In the Lab
Multi-platform
Mac OS X
Modes
Requirements
Integration


xLEAPP for iOS & Android
Parse and validate mobile artifacts with Python
What
When
Contact
Why
Where
Who
How

Automation - Find

```
tosearch = {
    'wellbeing': ('Wellbeing', '**/com.google.android.apps.wellbeing/databases/app_usage*'),
    'wellbeingURLs': ('Wellbeing', '**/com.google.android.apps.wellbeing/databases/app_usage*'), # Get app_usage & app_usage-wal
    'wellbeingaccount': ('Wellbeing', '**/com.google.android.apps.wellbeing/files/AccountData.pb'),
    'swellbeing': ('Wellbeing', '**/com.samsung.android.forest/databases/dwbCommon.db*'),
    'usageapps': ('App Interaction', '**/com.google.android.as/databases/reflection_gel_events.db*'),
    'usagestats':('Usage Stats', '**/system/usagestats/*'), # fs: matches only 1st level folders under usagestats/, tar/zip matches every single
        file recursively under usagestats/
    'recentactivity':('Recent Activity', '**/system_ce/*'),
    'installedappsGass':('Installed Apps', '**/com.google.android.gms/databases/gass.db'),
    'installedappsVending': ('Installed Apps', '**/com.android.vending/databases/localappstate.db'),
    'installedappsLibrary': ('Installed Apps', '**/com.android.vending/databases/library.db'),
    'calllog': ('Call Logs', '**/com.android.providers.contacts/databases/calllog.db'),
    'accounts_de': ('Accounts_de', '**/system_de/*/accounts_de.db'),
    'accounts_ce': ('Accounts_ce', '**/system_ce/*/accounts_ce.db'),
    'accounts_ce_authtokens':('Accounts_ce', '**/accounts_ce.db'),
    'cmh':('Samsung_CMH', '**/cmh.db'),
    'sms_mms':('SMS & MMS', '**/com.android.providers.telephony/databases/mmssms*'), # Get mmssms.db, mms-wal.db
    'chrome':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeSearchTerms':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeDownloads':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeLoginData':('Chrome', ('**/app_chrome/Default/Login Data*', '**/app_sbrowser/Default/Login Data*')),
    'chromeBookmarks':('Chrome', ('**/app_chrome/Default/Bookmarks*', '**/app_sbrowser/Default/Bookmarks*')),
    'chromeCookies':('Chrome', ('**/app_chrome/Default/Cookies*', '**/app_sbrowser/Default/Cookies*')),
    'chromeTopSites':('Chrome', ('**/app_chrome/Default/Top Sites*', '**/app_sbrowser/Default/Top Sites*')),
    'chromeWebsearch':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeOfflinePages':('Chrome', ('**/app_chrome/Default/Offline Pages/metadata/OfflinePages.db*', '**/app_sbrowser/Default/Offline Pages/
        metadata/OfflinePages.db*')),
    'quicksearch_recent':('Google Now & QuickSearch', '**/com.google.android.googlequicksearchbox/files/recently/*'),
    'quicksearch':('Google Now & QuickSearch', '**/com.google.android.googlequicksearchbox/app_session/*.binarypb'),
    'googleNowPlaying':('Now Playing', '**/com.google.intelligence.sense/db/history_db*'),
    'googlePlaySearches':('Google Play', '**/com.android.vending/databases/suggestions.db*'),
    'siminfo':('Device Info', '**/user_de/*/com.android.providers.telephony/databases/telephony.db'),
    'build':('Device Info', '**/vendor/build.prop'),
    'userDict':('User Dictionary', '**/com.android.providers.userdictionary/databases/user_dict.db*'),
    'pSettings':('Device Info', '**/com.google.android.gsf/databases/googlesettings.db*'),
```

Automation - Find

```
tosearch = {
    'wellbeing': ('Wellbeing', '**/com.google.android.apps.wellbeing/databases/app_usage*'),
    'wellbeingURLs': ('Wellbeing', '**/com.google.android.apps.wellbeing/databases/app_usage*'), # Get app_usage & app_usage-wal
    'wellbeingaccount': ('Wellbeing', '**/com.google.android.apps.wellbeing/files/AccountData.pb'),
    'swellbeing': ('Wellbeing', '**/com.samsung.android.forest/databases/dwbCommon.db*'),
    'usageapps': ('App Interaction', '**/com.google.android.as/databases/reflection_gel_events.db*'),
    'usagestats':('Usage Stats', '**/system/usagestats/*'), # fs: matches only 1st level folders under usagestats/, tar/zip matches every single
        file recursively under usagestats/
    'recentactivity':('Recent Activity', '**/system_ce/*'),
    'installedappsGass':('Installed Apps', '**/com.google.android.gms/databases/gass.db'),
    'installedappsVending': ('Installed Apps', '**/com.android.vending/databases/localappstate.db'),
    'installedappsLibrary': ('Installed Apps', '**/com.android.vending/databases/library.db'),
    'calllog': ('Call Logs', '**/com.android.providers.contacts/databases/calllog.db'),
    'accounts_de': ('Accounts_de', '**/system_de/*/accounts_de.db'),
    'accounts_ce': ('Accounts_ce', '**/system_ce/*/accounts_ce.db'),
    'accounts_ce_authtokens':('Accounts_ce', '**/accounts_ce.db'),
    'cmh':('Samsung_CMH', '**/cmh.db'),
    'sms_mms':('SMS & MMS', '**/com.android.providers.telephony/databases/mmssms*'), # Get mmssms.db, mms-wal.db
    'chrome':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeSearchTerms':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeDownloads':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeLoginData':('Chrome', ('**/app_chrome/Default/Login Data*', '**/app_sbrowser/Default/Login Data*')),
    'chromeBookmarks':('Chrome', ('**/app_chrome/Default/Bookmarks*', '**/app_sbrowser/Default/Bookmarks*')),
    'chromeCookies':('Chrome', ('**/app_chrome/Default/Cookies*', '**/app_sbrowser/Default/Cookies*')),
    'chromeTopSites':('Chrome', ('**/app_chrome/Default/Top Sites*', '**/app_sbrowser/Default/Top Sites*')),
    'chromeWebsearch':('Chrome', ('**/app_chrome/Default/History*', '**/app_sbrowser/Default/History*')),
    'chromeOfflinePages':('Chrome', ('**/app_chrome/Default/Offline Pages/metadata/OfflinePages.db*', '**/app_sbrowser/Default/Offline Pages/
        metadata/OfflinePages.db*')),
    'quicksearch_recent':('Google Now & QuickSearch', '**/com.google.android.googlequicksearchbox/files/recently/*'),
    'quicksearch':('Google Now & QuickSearch', '**/com.google.android.googlequicksearchbox/app_session/*.binarypb'),
    'googleNowPlaying':('Now Playing', '**/com.google.intelligence.sense/db/history_db*'),
    'googlePlaySearches':('Google Play', '**/com.android.vending/databases/suggestions.db*'),
    'siminfo':('Device Info', '**/user_de/*/com.android.providers.telephony/databases/telephony.db'),
    'build':('Device Info', '**/vendor/build.prop'),
    'userDict':('User Dictionary', '**/com.android.providers.userdictionary/databases/user_dict.db*'),
    'pSettings':('Device Info', '**/com.google.android.gsf/databases/googlesettings.db*'),
```

Parsing

Parsing

```python
import os
import sqlite3
from scripts.artifact_report import ArtifactHtmlReport
from scripts.ilapfuncs import logfunc, tsv, is_platform_windows

def get_wellbeing(files_found, report_folder, seeker):

    for file_found in files_found:
        file_found = str(file_found)
        if not file_found.endswith('app_usage'):
            continue # Skip all other files

        db = sqlite3.connect(file_found)
        cursor = db.cursor()
        cursor.execute('''
        SELECT
                events._id,
                datetime(events.timestamp /1000, 'UNIXEPOCH') as timestamps,
                packages.package_name,
                events.type,
                case
                    when events.type = 1 THEN 'ACTIVITY_RESUMED'
                    when events.type = 2 THEN 'ACTIVITY_PAUSED'
                    when events.type = 12 THEN 'NOTIFICATION'
                    when events.type = 18 THEN 'KEYGUARD_HIDDEN & || Device Unlock'
                    when events.type = 19 THEN 'FOREGROUND_SERVICE_START'
                    when events.type = 20 THEN 'FOREGROUND_SERVICE_STOP'
                    when events.type = 23 THEN 'ACTIVITY_STOPPED'
                    when events.type = 26 THEN 'DEVICE_SHUTDOWN'
                    when events.type = 27 THEN 'DEVICE_STARTUP'
                    else events.type
                    END as eventtype
                FROM
                events INNER JOIN packages ON events.package_id=packages._id
        ''')
```

- SQLite

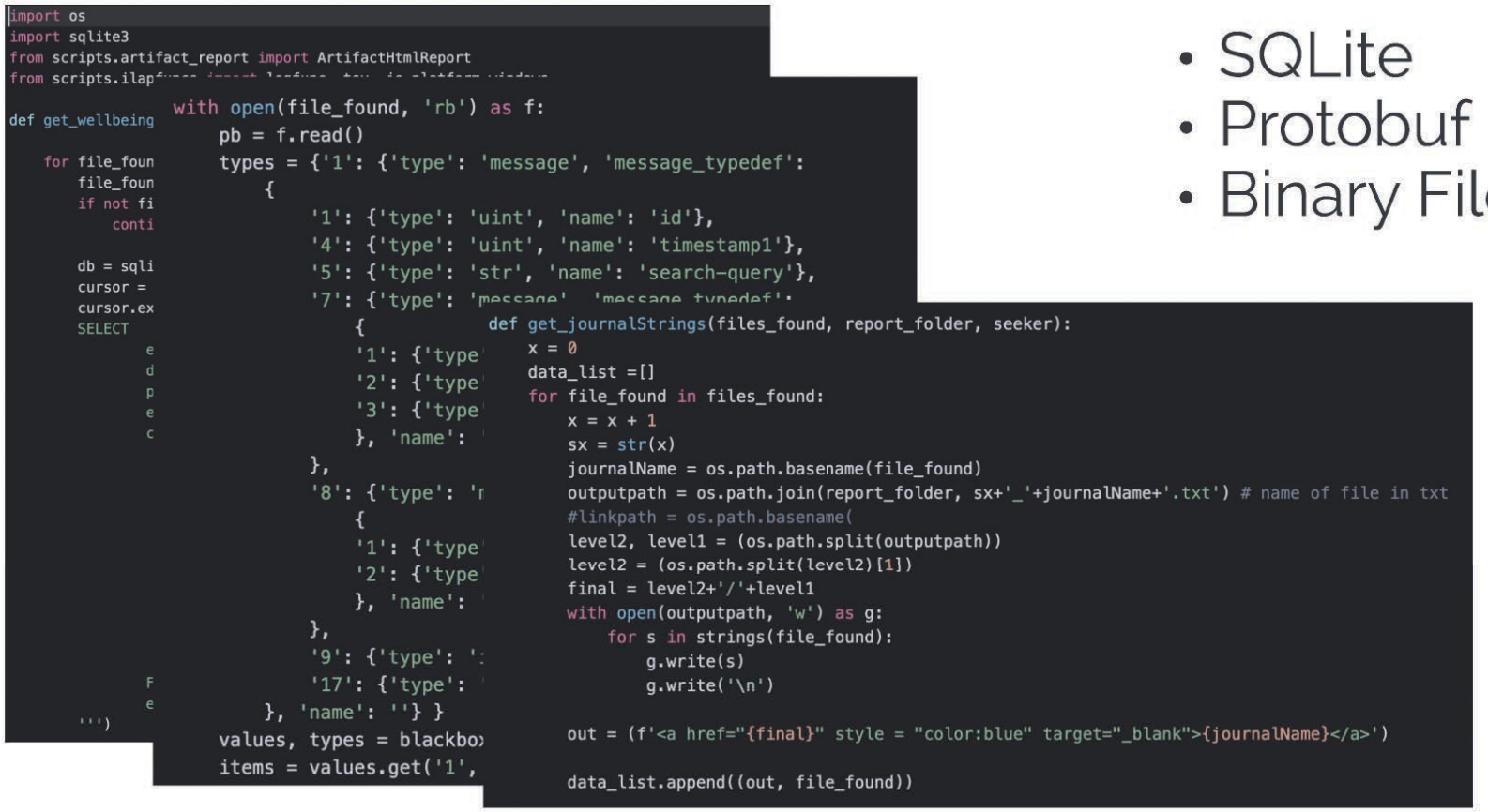
Parsing

```
import os
import sqlite3
from scripts.artifact_report import ArtifactHtmlReport
from scripts.ilap

def get_wellbeing

    for file_foun
        file_foun
        if not fi
            conti

        db = sqli
        cursor =
        cursor.ex
        SELECT
                e
                d
                p
                e
                c

                F
                e
        ''')
```

```
with open(file_found, 'rb') as f:
    pb = f.read()
    types = {'1': {'type': 'message', 'message_typedef':
        {
            '1': {'type': 'uint', 'name': 'id'},
            '4': {'type': 'uint', 'name': 'timestamp1'},
            '5': {'type': 'str', 'name': 'search-query'},
            '7': {'type': 'message', 'message_typedef':
                {
                '1': {'type': 'str', 'name': 'url'},
                '2': {'type': 'str', 'name': 'url-domain'},
                '3': {'type': 'str', 'name': 'title'}
                }, 'name': 'page'
            },
            '8': {'type': 'message', 'message_typedef':
                {
                '1': {'type': 'str', 'name': 'category'},
                '2': {'type': 'str', 'name': 'engine'}
                }, 'name': 'search'
            },
            '9': {'type': 'int', 'name': 'screenshot-id'},
            '17': {'type': 'uint', 'name': 'timestamp2'},
        }, 'name': ''} }
    values, types = blackboxprotobuf.decode_message(pb, types)
    items = values.get('1', None)
```

- SQLite
- Protobuf

Parsing



```
import os
import sqlite3
from scripts.artifact_report import ArtifactHtmlReport
from scripts.ilap

def get_wellbeing

    for file_foun
        file_foun
        if not fi
            conti

        db = sqli
        cursor =
        cursor.ex
        SELECT
                e
                d
                p
                e
                c

                F
                e
        ''')
```

```
    with open(file_found, 'rb') as f:
        pb = f.read()
        types = {'1': {'type': 'message', 'message_typedef':
            {
                '1': {'type': 'uint', 'name': 'id'},
                '4': {'type': 'uint', 'name': 'timestamp1'},
                '5': {'type': 'str', 'name': 'search-query'},
                '7': {'type': 'message', 'message_typedef':
                    {
                    '1': {'type
                    '2': {'type
                    '3': {'type
                    }, 'name':
                },
                '8': {'type': 'r
                    {
                    '1': {'type
                    '2': {'type
                    }, 'name':
                },
                '9': {'type': '
                '17': {'type':
            }, 'name': ''} }
        values, types = blackbo
        items = values.get('1',
```

```
def get_journalStrings(files_found, report_folder, seeker):
    x = 0
    data_list =[]
    for file_found in files_found:
        x = x + 1
        sx = str(x)
        journalName = os.path.basename(file_found)
        outputpath = os.path.join(report_folder, sx+'_'+journalName+'.txt') # name of file in txt
        #linkpath = os.path.basename(
        level2, level1 = (os.path.split(outputpath))
        level2 = (os.path.split(level2)[1])
        final = level2+'/'+level1
        with open(outputpath, 'w') as g:
            for s in strings(file_found):
                g.write(s)
                g.write('\n')

        out = (f'<a href="{final}" style = "color:blue" target="_blank">{journalName}</a>')

        data_list.append((out, file_found))
```

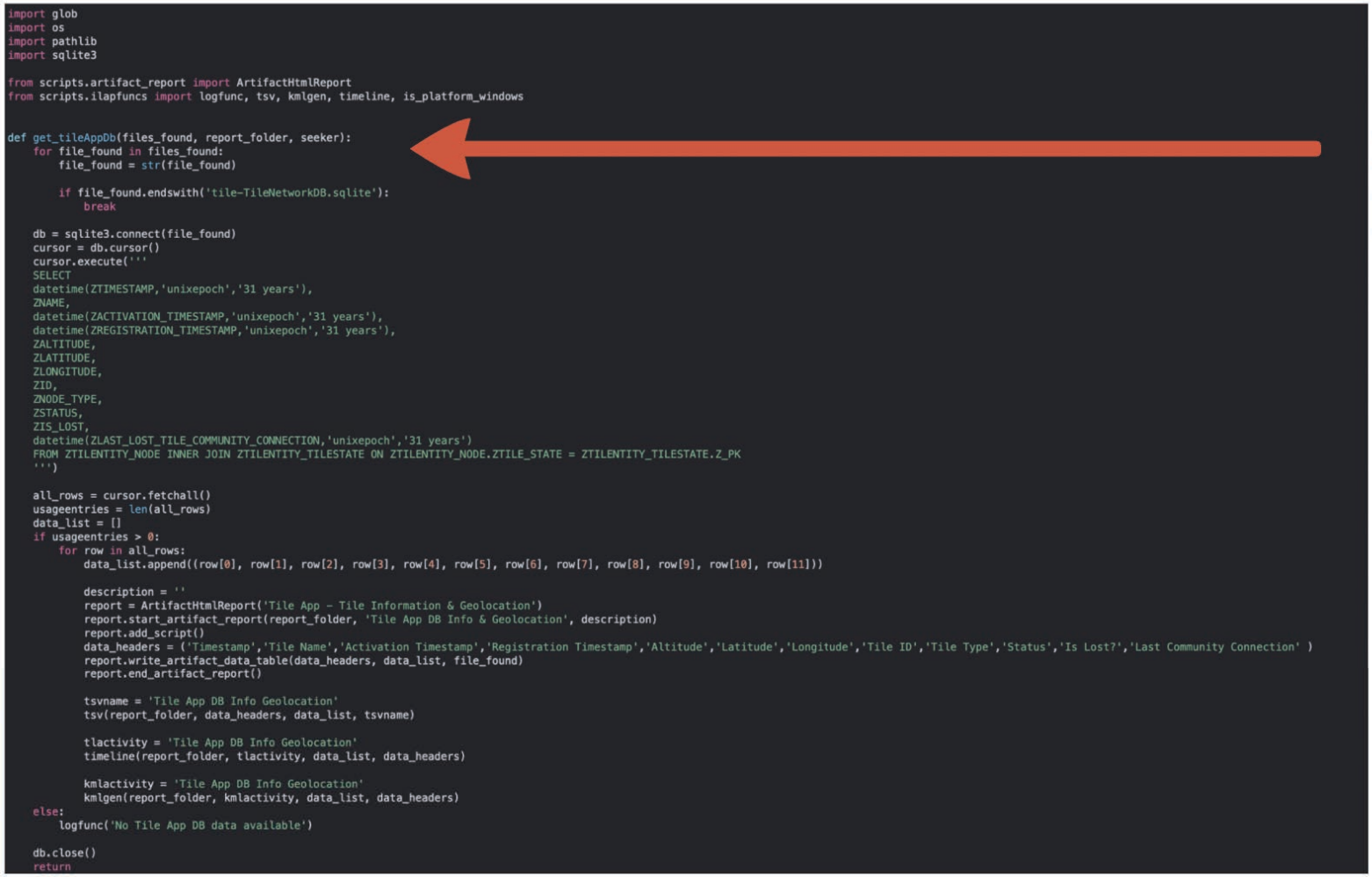
- SQLite
- Protobuf
- Binary Files

```
import glob
import os
import pathlib
import sqlite3

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

Imports

```
import glob
import os
import pathlib
import sqlite3

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```



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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

Search Results

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

Your magic

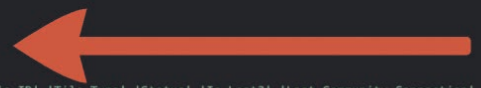
```python
import glob
import os
import pathlib
import sqlite3

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

Results

```
import glob
import os
import pathlib
import sqlite3

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```



Start reporting

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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
```

End reporting

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

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


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

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

    db = sqlite3.connect(file_found)
    cursor = db.cursor()
    cursor.execute('''
    SELECT
    datetime(ZTIMESTAMP,'unixepoch','31 years'),
    ZNAME,
    datetime(ZACTIVATION_TIMESTAMP,'unixepoch','31 years'),
    datetime(ZREGISTRATION_TIMESTAMP,'unixepoch','31 years'),
    ZALTITUDE,
    ZLATITUDE,
    ZLONGITUDE,
    ZID,
    ZNODE_TYPE,
    ZSTATUS,
    ZIS_LOST,
    datetime(ZLAST_LOST_TILE_COMMUNITY_CONNECTION,'unixepoch','31 years')
    FROM ZTILENTITY_NODE INNER JOIN ZTILENTITY_TILESTATE ON ZTILENTITY_NODE.ZTILE_STATE = ZTILENTITY_TILESTATE.Z_PK
    ''')

    all_rows = cursor.fetchall()
    usageentries = len(all_rows)
    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 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 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
```

Results

```
    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 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
```

Report title

```
    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 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
```

Category

```
    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 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
```

Headers

```
    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 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
```

TSV