TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Nikita Timokhin 214710IVCM

Structural Assessment and Automatic Aggregation of OS X Forensic Artifacts

Master's thesis

Supervisor: Pavel Tšikul
PhD Candidate

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikita Timokhin 214710IVCM

OS X KOHTUEKSPERTIISI ARTEFAKTIDE STRUKTUURI HINDAMINE JA AUTOMAATNE KOGUMINE

Magistritöö

Juhendaja: Pavel Tšikul
Doktorant

Tallinn 2023

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Nikta Timokhin

[15.05.2023]

Abstract

This paper examines the state of forensic artifacts in modern versions of Mac OS X for desktop computers. This operating system is gradually gaining more and more market share, which implies more OS X based devices are becoming part of forensic investigations. However, there is no up-to-date structured map of its forensic artifacts, such as files and system entries, as there is no abundant tooling for their automated excavation and aggregation.

Based off the existing research on forensic artifact locations in OS X and such, the presented study explores the OS and creates a comprehensive, structured map of thereof. This map is then used in conjunction with design science guidelines to produce a tool for automated discovery, excavation, aggregation, description and processing of the artifacts.

The presented artifact map is a valuable open-source document useful for any researcher, developer or forensic expert working with OS X, while the tool serves as a ready-made aggregation solution and a base for further development.

This thesis is written in English and is 59 pages long, including 8 chapters, 7 figures and 10 tables.

List of abbreviations and terms

AFP	Apple File Protocol
APFS	Apple-Proprietary File System
BASH, Bash	Bourne Again SHell
CLI	Command Line Interface
EXT4	Fourth Extended Filesystem
FS	File System
HFS+	Hierarchical File System Plus
IP	Internet Protocol
JSON, .json	JavaScript Object Notation
OS	Operating System
OS X, Mac OS X, Mac OS	Apple’s proprietary operating system for desktop computers
.plist	Preference List file
SQLite, .sqlite	Structured Query Language Lite database file
URL	Universal Resource Location
VS	Versus
XML	eXtended Markup Language

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

Apple's Macintosh desktop and laptop computer operating system (OS), Mac OS X introduced XML-based property list (.plist) files in 2001, with the release of version 10.0 for PowerPC, followed by the nowadays-prevalent binary preference lists since OS X 10.2 [21,6]. These files may contain "user and application preference information and application's session, user's information and many more artifacts" [13] and system properties [15] – they are generated mostly by the OS X applications themselves, and the information they contain varies greatly with the application itself.

Although on the surface it may seem that program preferences do not carry any specific forensic value, this is proven otherwise by most big forensic solutions such as FTK Imager supporting binary .plist decoding natively [15]. This is because sometimes these files would have sensitive data, ranging from usernames, language settings, and timestamps to Internet Protocol (IP) addresses, geographic coordinates, and base64-encoded datastores.

Apple's OS X market share has been steadily increasing ever since 2010 [16] and likewise did increase both the attacks on it and its usage by criminal individuals or individuals with malicious intent. Respectively, the attacked OS X systems need protection, and the attackers' systems captured as forensic evidence need examination.

OS X on a Hierarchical File System Plus (HFS+) or Apple-Proprietary File System (APFS) formatted drive has plenty of low-level capabilities that can be used forensically on its own, such as Journal and timestamps [1]. Very high-level forensic capabilities, such as simply opening the mail application (app) also exist.

However, the layer between disk-level and app-level examination lacks both academic research and practical tooling. This layer includes forensically useful artifacts like the file metadata[10], SQLite databases[15], and, among everything else, the .plist files[12].

1.2 Research Motivation

In the present reality where OS X is slowly and steadily climbing the market share ladder of the operating systems, and the overall index of cybercriminal activity is rising, it is important to keep on researching forensically interesting aspects of the operating system and develop tools for its examination. Conducting design science research for a particular narrow class of OS X forensic artifacts like .plist files that are on the “middle layer” – between disk-level and app-level examination – is appealing in many ways.

Firstly, it adds to the generally available experience on the matter, which is currently not at all numerous, lacks solid frameworks and, for a considerable part, is not accessible to the general public. Carrying out well-documented practical research would contribute to the overall pool of research on this and adjacent topic, and facilitate further research, potentially facilitating the eventual establishment of robust artifact research frameworks and techniques.

Secondly, automated tooling in this area is currently lacking. While solutions like EnCase and FTK Imager exist, and can perform disk recovery, and facilitate the formal part of the forensic examination and data acquisition, no solution with native .plist-centric functionality excavates, sorts, classifies, or parses these files from a volume that has OS X on it[15]; indeed, it is not even the stated mission of these tools. Designing a tool that specializes in this particular forensic task and adding it to the roster of generally available tooling facilitates the work of forensic experts and other researchers alike.

Finally, and most importantly, OS X file-level forensics appears to be a fairly narrow field of study, and for that reason is somewhat underrepresented compared to both disk-level and app-level studies. It is imperative to add to this narrow field of research to contribute to keeping all layers of OS X forensically inclined research well-represented and synergetic.

2 Literature Review

A pool of literature suiting the research topic was created. Advanced logic filtering techniques were used along with forward and reverse snowballing techniques to cover as much material as needed while keeping it as close to the topic as possible. The resulting selection covers questions related directly to the present research topic, as well as various topics adjacent to the present research and additional topics to supplement various research aspects.

2.1 OS X Preference Lists in the Forensic Context

Overall, the preference lists turned out to be somewhat revered inside the overall OS X forensics topic. One of the selected works, "Mac OS X Forensic artifact Locations" by Michael Cook et al.[12], is of particular interest, as it picks on the exact question the present research does as well. It has been conducted in 2015 – that is 8 years before the current study – and its goals are admittedly close to this study.

The work provides a table of locations of forensic artifacts inside OS X, many of which are of .plist type. This work, however, lacks a rigid description of searching methods; they likely were manual. Likely for this reason, the map appears incomplete. Moreover, the study has been conducted years ago, possibly rendering some of the artifact locations nonactual. This study is the one the present research aims to update and expand on by adding the tool design and research science aspects.

The remaining range of works directly related to the topic is existent, albeit quite limited. Dr. Digvijaysinh Rathod argues that the preference list files serve direct forensic use, and provide a .plist generated by the Safari web browser with a description of its forensically interesting content as an example[13]. Christian Hummert shares this point of view, adding that no modern forensic solutions fully support binary preference lists[15].

2.2 General OS X Preference List Information

Research on .plist files in their forensic context is hard, if not pointless, without proper backing information on the file format, its history, and its peculiar aspects. Janet Bass et al. report the OS X preference lists being quite a chaotic and disorganized format[4]. Indeed, it is apparent from their work how Apple's preference lists not only come in different base formats (NeXTSTEP, XML, Binary, JSON), but also may represent the same type of data as a different construct – for example, for sequential data points there is no clear pattern of when a dictionary is used versus when an array is, and each .plist's structure ends up depending on the particular programmer's choices at the end of all. This unruliness makes .plist files harder to carve and recover, among other artifacts like images, and to figure out what exactly its fields mean.

The prevalent format of .plist files in OS X is Binary .plists, and Christian Hummert provides an in-depth explanation of the format's insides and principles[15]. This information is not as valuable to the present research as its scope does not include file recovery, to which the information provided in [15] would make a substantial guide. Along with this information, a brief history of the .plist format in the OS X context is provided. No other relevant information source of such quality on OS X preference list structure and purpose was located. Hummert puts OS X in a forensic context later on in the book, expanding on the analysis of fields. This information intersects well with the presented study's aim to gather and assess .plist files common on modern OS X systems.

2.3 Other Types of OS X Forensic Artifacts

Naturally, forensic artifacts come in various formats and appear in various levels of Mac OS X. One of the selected base papers mentions .sqlite files and cache dumps as valuable higher-level artifacts [12]. However, a substantial amount of work has been put into studying the lower-level OS X systems.

Following Extended File System 4 (EXT4) of Linux receiving extended journalling and other forensically valuable capabilities[19] and having them studied by independent researchers in 2007, Apple's HFS got upgraded to HFS+ in 2009[5] and received numerous capabilities of very similar kind, such as Journal. Naturally, the lower-level OS X forensic capabilities depend on these new capabilities of the formatted drive OS X is running on (HFS+ or APFS), hence, they are investigated thoroughly. Jong-Hwa Song et al. investigated [1] the APFS timestamps, and Kurt K. Hansen et al. decoded[5] the APFS file system structure at byte levels. Such findings for HFS+ and APFS likewise are put into forensic context by Philip Craiger et al., providing proper explanations of using the disk- and memory-level techniques[11]. The lower-level OS X forensic artifacts are numerous, and at this point, they are relatively well documented; extracting and systematizing them is possible with some of the modern forensic examination solutions.

2.4 Existing Tooling

It is apparent that big, well-known forensic examination solutions like FTK Imager do not have built-in deep .plist operations[15], and that no substantial framework for file-level artifact discovery has been located. For this reason, it is necessary to retrieve research on more specialized solutions and their documented development processes to incorporate the upsides into the present research.

Admittedly, the topic of specialized OS X forensic tooling is quite sparsely populated. It appears that the forensic specialists who find themselves in need of very specific actions with OS X use a popular solution (FTK imager or such) to gather the data, and then process it manually or write proprietary scripts without publishing them.

For the most part, the present research benefits from materials on the topic of new forensic tooling for OS X targetting artifacts that reside above the lower levels of Journal, timestamp, and such. Any related experience, even with lower-level tools, is also feasible. Robert A. Joyce et al. documented their development of MEGA [14], a tool that uses OS X's own generated search databases and other mechanisms that facilitate search. MEGA uses these mechanisms in combination with general knowledge of OS X forensic artifacts, such as file metadata and in-app data like contacts in OS X's native "Mail" e-mail client. On the other side, Gyu-Sang Cho developed a tool named FACT (Forensic Analyzer based Cluster Information Tool) – a tool prototype for very low-level, cluster-based HFS+ volume assessment [8].

There are two issues with MEGA. Firstly, it was released in 2008 – fifteen years before the present study is carried out. It does not appear that the project is being supported, so it is highly likely out of date in both its implementation technology and its assessment methods. Secondly, while exposing forensically interesting information, it still leaves the search process up to the expert; the work does not outline any particular way of searching the interesting bits, and neither does it provide any automated means of doing so. Lastly, MEGA and FACT share a similar fate of development traces getting lost, leading to the conclusion of both programs essentially being abandonware at this point.

Aside from MEGA and FACT, there are not at all many well-documented pieces of research and even somewhat maintained tools. Apple themselves did release a Developer Tool called "Property List Editor" on their website, which allows viewing and editing of all OS X-native .plist files[17]. However, the development process of this tool is not part of any research, and the tool itself is but an archival remnant on the Internet Archive, which does not exactly contribute to the modern specialized tooling variety.

2.5 Design Science Approach

Since the research includes designing software and analyzing its behavior, it is crucial to select a scientific development framework that is widely approved and has been used in other academic studies that include software development.

Such frameworks are numerous, but the one that fits the practical tooling inclined type of the present research is "Design Science in Information Systems Research" by Alan Hevner et al. This work provides a multitude of practically applicable tool development and testing strategies that fit into the formal and academic research aspect and is nearing eight thousand citations as of April 2023.

Admittedly, this paper has some of its focuses rather visibly tilted toward the enterprise environment; some of the design and research strategies outlined are not exactly fit for research conducted by a single person. However, these aspects of the paper mostly cover the topics that are generated by the enterprise itself, such as business strategy and such. The parts that cover the research and development process itself are perfectly applicable in the context of the present research.

3 Research Gap Review

After assembling a pool of literature important for the presented study, studying and critically overviewing it, a research gap can be identified.

Firstly, most studies related to OS X forensics are rather theoretical. In itself, this is reasonable and acceptable. However, there appears to not be enough studies with more practical studies documenting not only the findings themselves, but the process and results of using these findings in the field. Such studies exist, but they are far outnumbered by more observational studies that state facts about the ways OS X works. Conducting a design science study that incorporates a working tool prototype as one of its results would contribute to the general experience and knowledge of practical OS X forensic artifact location.

Secondly, while some studies do state the locations of some of the forensically interesting files, such as preference lists, SQLite databases, and mail caches, no study clearly defines a repeatable process of searching for the artifacts. This sets further studies like the present research itself back, by having the publicly accessible research space more or less devoid of documented experience in artifact searching. The existing research does provide some base points in the OS X file hierarchy to start the searches from, which is useful, but the exact sequence and algorithm of searching that could be reused was not located. Hence, it makes sense to have the present research have a well-defined and sufficiently reasoned procedure of artifact location, which can then be modified and repeated as necessary.

Furthermore, the overall OS X forensic artifact and capability research are somewhat skewed towards favoring lower-level activities, such as the HFS+/APFS Journal parsing, timestamp discovery, file recovery, and so on – this topic appears furnished enough and partially implemented in practice in existing solutions. The other, smaller portion of the research, focuses on rather high-level concepts, such as using the default applications as forensic data sources; this research does not seem as relevant in 2023, when most people use cloud services by companies like Google to access e-mail services, store contacts, and such, rather than using the built-in apps like “Mail” for managing e-mails and “Contacts” for manually storing and retrieving useful contact information. The middle ground – a logic level above Journal and a logic level below “Mail” – seems rather sparsely populated in terms of relevant research. It makes sense to make .plist files the main target of the present research, as they fall exactly in that less researched middle area.

4 Research Design

In this section of the document, research design decisions, assumptions and strategies are explained. The present research has the two main objectives in its scope:

1. Create a solution capable of automated forensic artifact excavation and processing. The solution should take into account the usefulness, type and rank of the artifact, and work by the composed map to excavate as much of the forensically interesting files as possible. The solution should not require installations of any additional software on an OS X system (be portable) and allow saving, loading and processing its own data, aggregating files from the system onto external volumes, and such. Since the aggregated and processed data will be of similar type, it makes sense to implement the file path aggregator for the data processing step as part of the overall solution even before implementing other parts.

2. Build a cohesive, full, up-to-date and technically useful map of computer-generated OS X forensic artifacts, such as the .plist files. The map should consist of files present on most OS X systems and have high tolerance to difference of usage time, OS version and past user actions. The map should provide clear details as to where the file is usually found, group them by their useful traits and rank them by their forensic usefulness.

This implies the research itself falls into two simultaneous parts:

1. Tool design – the part at which the automated tool prototype is created. At this point, the data processing part has already yielded its results in form of a robust and useful artifact map. The tool shall contain selected parts of this map, and the tool's actions shall be based off the properties of the map entries. The tool design is conducted with accordance to the seven Design-Science Research Guidelines[7] in order to facilitate and systematize the design process.

2. Data processing – the part at which the artifact map is created. At this step, data on .plist files is automatically collected using a tool (program) designed specifically for the present research needs. The data is then structured (compared, condensed and de-noised) and assessed (grouped, ranked, overviewed) to form the map. The ethical aspects of data collection shall be addressed along with the technicalities.

4.1 Ethical Aspects of Research

Ethical usage of data subjects' rights is crucial for any research that processes personal or potentially identifying information, let alone a forensically inclined one. It is of utmost importance to outline rigid requirements and guidelines of data usage for the present research and make sure that all the research procedures and practices adhere to them. In order to achieve proper guidelines, it must be well understood what information the study would collect at which steps, and how sensitive the data is.

Information on file locations is gathered from numerous live systems. Such system may belong to any person who replies to a call for research participants, not necessarily part of the research group or otherwise related to the researcher. This data shall be minimized to only the most necessary parts – the present research only needs the file paths and nothing more, hence it must be assured that the tool's functionality responsible for gathering this data never accesses the file contents and only programmatically checks if files exist or not. The only directly identifying information in the collected data (lists of file paths) is the user name; it shall be uniformly pseudonymized. Finally, no original information shall be submitted as part of the final results; the reason for collecting the information is searching for common, ubiquitous file paths – so only this information may directly contribute to the further steps of the research and the final results. The volunteers shall be clearly notified of what information the tool searches for exactly, be able to review the code before running it and the data output before submitting it to the research.

Information on forensically interesting artifact file contents is gathered from the same kind of live OS X installations on personal machines. However, for research steps that require reading file contents, only machines of the research group members may be used. Such a narrowing restriction essentially leaves two individuals; however it is necessary to ensure that the highly sensitive data gathered in this step is gathered from the individuals who clearly know the research roadmap and are aware of how exactly the data is being treated. The file contents themselves shall be used for analysis of artifact importance, the type of information found in the file and such. Raw file contents may not be part of the research product or the analysis – the research is restricted to only submitting generalized properties, conclusions and such (e.g. the fact of the existence of a cache dump in a .plist file) without submitting the contents themselves (e.g. the cache dump data).

4.2 General Research Plan

As noted in the beginning of Chapter 4, the research is comprised of two simultaneous processes: tool design and data processing. While these two parts have separate requirements and working processes, they may not be conducted sequentially one after another, and instead rely on each other's output as the research proceeds.

For this reason, the research part in which the parts work in tandem is logically divided into three major stages:

1. Preference list path acquisition and processing – the tool is completed on a level sufficient to run the assembly of all .plist file paths in selected search places and search for same file paths across different export results. The tool and instructions are sent out to the individuals volunteering to be part of the research, the returned data exports are aggregated at the researcher machine, the common file paths are detected using the tool running on the researcher machine. A list of overlapping file paths is composed.

2. Ranking and grouping of the artifacts – the resulting list from step 1 is used in conjunction with the tool to manually assess, rank and assign groups to each artifact. Although this task would be possible without the tool, for better repeatability and more reliable process, the tool is complete with a utility that facilitates the process and outputs an automatically formatted artifact map JSON

file. Files are being accessed locally on the researcher machine; if the file is missing, it is discarded from the artifact map.

3. Evaluation and field testing. At this stage, the tool's main functionality of excavating and rank-sorting files guided by the artifact map once ran on any system is implemented. The tool is ran on a number of machines matching operational and ethical criteria. The resulted file collections are aggregated on the researcher machine and examined in order to draw a conclusion on the effectiveness of the tool, the approach and the methods.

4.3 Plan of Tool Design

4.3.1 Requirements

A tool is developed for the present study. This tool carries out all programmatic needs of the study: gathering, structuring, assessing and managing of data. Before programming the tool, a set of requirements is formed.

1. The tool shall not be heavy on machine resources, shall not require root level privileges, and not require a lot of external components.

2. The tool shall gather the information needed for study in minimized and pseudonymized way. It shall not gather more information than needed for the study. It shall export the data locally at the data subject's machine so that the data subject themselves could examine the data before manually submitting it to the present research. It shall not establish any internet connections while gathering the data.

3. The tool shall automatically structure the gathered data by reading existing data exports submitted by the data subjects. It shall find overlapping file paths and write them to another file for further usage. It shall allow for custom tolerance – on how many of the data exports can a file be missing – to be provided to it.

4. The tool shall provide at least a basic manual mean of assessing the files by accessing the file paths, displaying the file, and allowing the researcher to assign

groups and a rank. The tool shall notify upon failing to read from the file path; some Apple programs have links in their preferences subdirectory that would point to a non-existing file. The tool shall export the map of assessed files as a separate file.

5. The tool shall provide a mean to use the artifact map as a guide to excavate interesting files from an arbitrary OS X system with Python 3 installed. The tool shall copy files from across the system it is ran on to its own export subfolder and sort them by ranks.

6. The tool shall provide a command-line user interface subsystem. It shall prompt the user or the researcher for arguments to its commands, be the arguments not provided after the command itself. It shall provide help messages for each of its commands. It shall not be a script ran with shell arguments, but rather its own subsystem with multiple available commands.

7. Finally, the tool shall be open source and easy to read, assess and modify. It shall require the most basic development kit possible, as to make it feasible for further research, modification, and such.

4.3.2 Implementation

Requirements 1 and 6 of the list in 3.1.1 imply Python as the programming language choice. It is already de facto the most widely-used prototyping tool by many businesses and individuals alike[20]. It has an extensive library of pre-installed modules that implement functionality otherwise having to be programmed manually, which enhances readability, universality and development speed. Admittedly, Python lacks the robustness and optimality of languages such as C# and Rust – however, those require more than installing one universal package and writing code in any text editor, making them a worse choice for prototyping.

To satisfy Requirement 2, it is enough to simply use the built-in "os" Python module to scan the local file tree and write only the file paths and other attributes that are significant to the present research to a JavaScript Object Notation (.json) file for further processing. The data subject can view it using any online JSON viewer, or by simply changing the extension to .txt and opening it in a basic text editor, such as TextEdit for OS X or Notepad on Windows. The information in the file shall be pseudonymized; since the only information stored is the file paths, and they include one personal identifier – the machine username of the data subject – it shall be replaced with a randomly generated pseudonym.

Requirement 3 is satisfiable using the default Python tools and capabilities. Since the data exported by the structuring (finding overlapping files) step shall be used to then attempt accessing the needed files on any OS X machine, and most of the file paths may contain a username, the generated pseudonyms shall all be replaced with a single indicator of a username location in the file path string, later to be replaced with an actual username when needed. A list of such username-templated file path strings shall be exported as a simple .txt file.

Requirements 4, 5 and 6 are satisfied by creating a clear and user friendly command line interface (CLI). For the assessment part, it shall iteratively read and display the file contents to the user, then prompting them for a rank and the groups to add the file to.

4.3.3 Evaluation

The tool has three main actions: gather - assemble information from the machine it is ran on, structurize – or find the overlapping file paths between data exports, and assess – give rank and groups to each file. They all rely on a clean, user-friendly CLI and robust underlying code system. Each of the three main actions rely on the previous one's results, which means even basic evaluation of structurization is not possible without the gathering part complete. Each action shall be iteratively evaluated using the generate-test cycle[12, Fig. 3] before applying it to existing live systems and taking the results into account for the research.

Evaluation of basic data gathering functionality is conducted against the researcher's machine in the mentioned iterative fashion. It is possible to create dummy data to test it on first, and then test it on the actual system and compare the outputs with the apparent state of affairs on the FS. Once this capability is deemed fully working, the program is sent to the data subjects to gather real life file path .json data.

The structurization capabilities are also iteratively evaluated. It is necessary to use the .json files received after the previous step in order to find tolerances and other aspects that are optimal to the research.

The assessing functionality is to be once again developed and tested fully on the researcher machine. This implies that the files that are ubiquitous to an extent, but not present on the machine, will not be included in the final map due to the live system inventory constraints. The assessing process shall prove to be easy and output a correct file before running the assessment routine to generate the present research's artifact map.

Finally, the file excavation and aggregation means are validated against one or more machines belonging to the research group. A metric of how many files were found of all files in the artifact map shall be collected along with the files themselves. Validation of this functionality is validation of the solution as a whole, as it essentially indicates the forensic usefulness of the tool via field testing.

4.4 Plan of Data Processing

4.4.1 Collection

The final goal of this step is collecting data on .plist files located on computers running OS X. This task is not as trivial, though, as there is an immense amount of such files on the systems, and a lot of them do not present any sort of usefulness. Even more complexity is added by the fact that the possibility of different modern versions of OS X having the targeted files differently named or structured cannot be completely ruled out, although locations appear to not have changed at all between Yosemite and El Capitan versions of OS X[12].

To make such a complex task more manageable and to reduce the needed output data to clutter ratio, which would in its turn make structuring and processing easier, the present study is basing itself off existing research, such as "Mac OS X Forensic artifact Locations" by Michael Cook et al.[12] – this type of research contains output somewhat similar to the target of the present research, although outdated and not necessarily structured in the most useful way.

Examining existing known locations of OS X forensic artifacts of needed type, one can build a list of folders to recursively search in. In the aforementioned work, most artifacts occur in subfolders of a small handful of core system folders, such as /User/<username>/Library/. However, the system itself contains numerous .plist files that do not seem to serve any forensic use.

Therefore, the scanned folders should be of low depth (not more than four layers deep relatively to the volume root) and created by the system installation, so that they are present even on very fresh OS X systems. This ensures that the list of scanned places of interest stays short and comprehensive, and that uninteresting files are effectively excluded.

After building the "interesting locations" for .plist files separately, a script shall be created that scans the locations recursively and generates a list of paths to the target artifacts that it found, along with the "interesting location" they were found in and the username scanning is conducted for. This script shall output a JavaScript Object Notation (JSON) text dump that can be further used for processing. It shall be ran on as many modern systems running OS X as possible in order to amass many combinations of existing artifacts, so that during later steps the common ones could be detected. Since one of the biggest practical papers ([12]) with a list the present research bases itself off investigates two systems running OS X Yosemite (2014) and OS X El Capitan (2015), it makes sense to set the system novelty bar to High Sierra (2017) and above, favouring Big Sur (2020) and above, as at the moment everything older than Big Sur is at its end of lifecycle.

Ethical aspects of data collection shall be addressed at this planning step. First, data minimization has to be enforced; this means the script has to collect as little data as possible. First and foremost, this means that as the artifacts themselves contain sensitive data, which the research is after, the files themselves shall not be exported from the user systems – only the file names and locations. Furthermore, since the data is being aggregated targeting a particular system user (usually the sole owner of a laptop running

OS X) – the username itself has to be protected before dumping the data to JSON. This shall be done by the use of pseudonymization.

One could argue that pseudonymization in this case is redundant, as people almost never use their full names as their computer usernames, and even if they do – they are definitely not the single person ever with such legal names. However, a solid argument is that “any information relating to an identified or identifiable natural person”[2] counts as personal data, legal name included. This implies that the aggregation script has to have built in pseudonymization by default. Moreover, “same pseudonym cannot be used for more than one subject is a fundamental rule, even if the person concerned chooses its form”[18]. These aspects have to be taken into account while designing the data aggregation script along with the present research’s technical requirements. After the research is complete, the amassed data shall be safely erased.

4.4.2 Structuring

After data has been successfully aggregated to the researcher laptop (MacBook Pro 13” 2012 running OS X High Sierra) it has to be structured. Structuring the data in this case means finding a uniformity among the provided data sets, which will then reveal locations of forensic artifacts that are more ubiquitous than others. This is an important research step, as the amount of forensically uninteresting .plist on a randomly chosen computer running OS X may be tremendous, and manually assessing them may turn into an impossibly long and, most importantly, pointless task.

For this reason, the received data shall be processed in a way that finds artifacts common among all the gathered datasets from individual machines. This action will reveal a list of files that are shared among the machines, i.e. are common artifacts. The number of these files will be significantly lower than the total number of detected .plist’s on any given system, and these can hence be assessed manually.

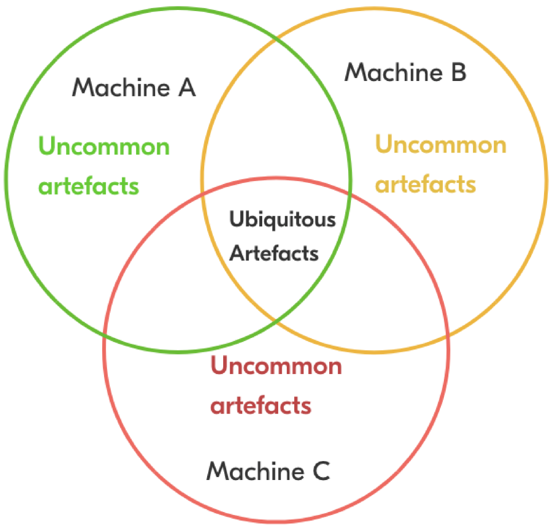


Figure 1. Graphic example of finding common artifacts

It is important to note that web browsers are a type of application that may generate .plist files, yet do not fit very well into this way of artefact searching. Browsers store a vast amount of potentially interesting information[15] on disk – in the file formats targeted by the present research (.plist) and otherwise. However, browsers are not necessarily ubiquitous across a small selection of machines due to the fact that browser choice is a matter of preference of the machine owner, and preferences differ. Although Chrome has been showing steady growth of market share over the past decade[9], other browsers like Firefox and, in case of OS X specifically, Safari, have rather wide user bases too. The described method of file filtering is most likely to filter out a lot of files of potential interest, as one machine may run Firefox, while other runs Chrome – and since they generate differently named files in different directories, they are not deemed common. The only counteraction to this is manual inclusion of these files to the final map, which falls out of the automatic aggregation scope. Moreover, browses like Chrome have more abundant forensically interesting information stored in the memory and low-level caches[6] than in the formats the present research targets. Hence, for the present research, such files shall be omitted.

4.4.3 Assessment

After gathering enough data from live systems running OS X and eliminating most of noise, the remaining ubiquitous forensic artifacts shall be manually assessed. The goal of this step is to contextualize and prioritize the gathered data. Separating more forensically valuable data from less valuable yields different tiers of artifacts, which then can be separately processed. Grouping and contextualizing the artifacts by their contents – the groups of forensically interesting information they have, such as language, date and time, filenames etc – may be profitable for building more advanced automated assessment tools that process a particular group of artifacts targeting its informational context.

The final artifact map contains a separate table per rank, yielding four sequential tables total. A single entry in the map is formatted the following way, demonstrated on some made up data:

Table 1. Example of a map entry row.

Path	Groups	Notes
/Applications/Firefox.app/contents/options.plist	datetime, language	Contains date of last program launch

“Scan Location” indicates which of the preprogrammed “interesting locations” the artifact was discovered in. “Path” is the path to the exact file relatively to the scan locations, ergo full path would be scan location followed by the file path. This separation is done because some scan locations will include username, which will make it harder to reuse the map if the absolute path is stored; every entry with username in it will have to be separately preprocessed to replace the username with a valid string.

The artifacts are manually examined and assigned one or more of the following groups:

Table 2. Artifact groups

Group	Meaning
geo	Any info related to geographic locations (country codes, coordinates...)
language	Records of language settings
datetime	Records of any date and time events, e.g. last program execution time
web	Any web-addresses and URLs

ip	Any IP addresses except local loopback/localhost
name	Any logins, names and other name-related info
data	Any potentially interesting data like hashes, b64-encoded strings, etc
path	Local file paths
other	Other potentially interesting fields

Ranking the artifacts will happen according to the following guide:

Table 3. Artifact ranks.

Rank	Denominator
Superior (S)	Has three or more unique interesting fields, falls into three or more different groups, is in at least two of groups: geo, web, ip, data
A-grade (A)	Has three or more unique interesting fields, falls into three or more different groups, is in at least one of groups: geo, web, ip, data
B-grade (B)	Has two or more unique interesting fields, falls into two or more different groups
C-grade (C)	Has one or more unique interesting field of any group

The ranking system will help the tool to prioritize its information sources, and is designed so that the better the rank, the more valuable information is presented, and in bigger quantities it is presented in. This implies that an artifact that has a lot of interesting bits of information, and a few of them contain more sensitive information (e.g. an IP address or a password hash) are valued above other, less data-abundant artifacts. This, however, by no means implies that all but S-ranked artifacts shall be disregarded; the files that hold no forensic meaning do not get any rank whatsoever and are subsequently discarded from the final map altogether, so, all ranked files may have important information in them; the ranking system only helps to facilitate choices like processing order and such.

5 Tool Design

This chapter explains the logic of tool development and explains the reasoning behind major design choices. Tool development is conducted simultaneously with data processing (Chapter 6), hence, each new layer of functionality is tested locally before contributing to the research. The practical applicability of the resulting final tool is further assessed by analyzing the results of field tests on real-life machines (Chapter 7).

5.1 User Interface

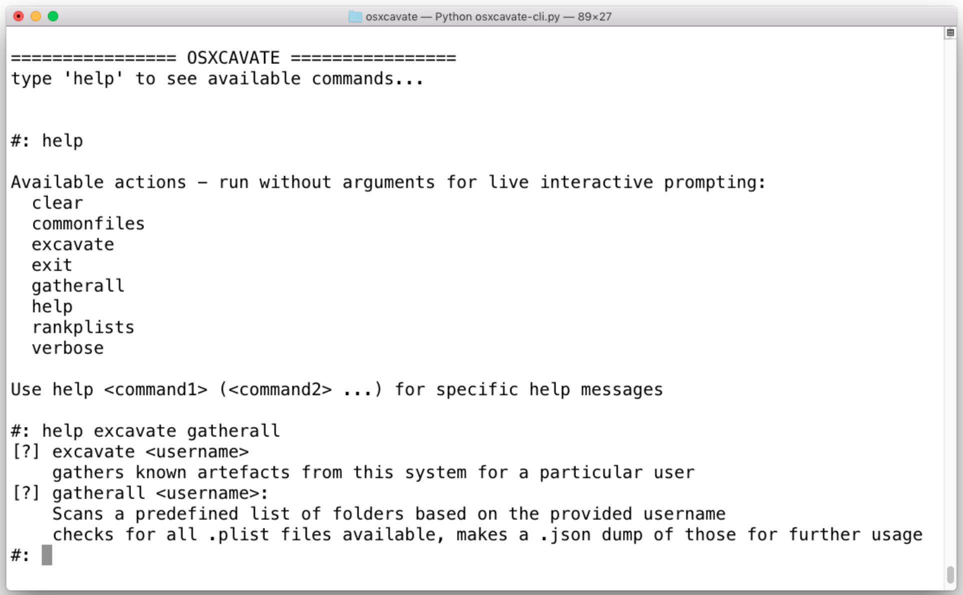
As per tool requirements and implementation guidelines set in Chapter 4, Python is the selected programming language for implementing the tool. Although lacking in execution speed and code optimality, its upsides of a free and lightweight development toolkit, vast built-in functionality useful for the tool, and Python being the most ubiquitous software prototyping option[20], far outweigh the downsides.

However, it is natural that a Python program can be written in many different ways. As per Design-Science Research Guideline 1, "Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation."[7]. The framework further states that "artifacts constructed in design-science research are rarely full-grown information systems that are used in practice. Instead, artifacts are innovations that define the ideas, practices, and technical capabilities"[7]. In practice, this implies that the development process shall focus on building innovation and capabilities while keeping the artifact (the tool) viable – hence, easily usable and practically useful.

These requirements define the choice of the user interface. As to not distract from the technical aspect of the tool development, making a graphical user interface (GUI) is out of the question. Making a proper GUI that keeps the tool viable requires massive amounts of additional research in the direction of both the user interface theory and the existing user interface implementations in widely-used solutions.

On the other hand, for the resulting software to be easily usable by any arbitrary individual without programming experience and deep knowledge of its source code. This eliminates the otherwise appealing option of making the tool adhere to the general Bourne Again SHell (BASH) utility program practices. A user who knows the program well may be able to successfully run it from the command line with all the correct arguments, a new user who is unfamiliar with the program easily gets lost. This is evident from observing the fact of "The Sleuth Kit" – a set of forensically useful scripts – getting way less usage from individuals than its fork with a well-made GUI, Autopsy.

This leaves one option viable: keeping the tool in the command line but giving it its command line interface which guides the user through required input arguments upon launching one of the commands, while keeping the BASH-style argument provision in place for advanced users as well. Such proprietary CLI is easily implemented in Python using its built-in high-level functionality. The interface code is structured so that adding new functions to the CLI code is intuitive and easy, and does not require writing excessive CLI-related code and manually validating provided arguments to commands. All the CLI functionality is stored in a "cli.py" module and is used by the main script to run almost all interface-related routines.



```
================ OSXCAVATE ================
type 'help' to see available commands...


#: help

Available actions - run without arguments for live interactive prompting:
  clear
  commonfiles
  excavate
  exit
  gatherall
  help
  rankplists
  verbose

Use help <command1> (<command2> ...) for specific help messages

#: help excavate gatherall
[?] excavate <username>
    gathers known artefacts from this system for a particular user
[?] gatherall <username>:
    Scans a predefined list of folders based on the provided username
    checks for all .plist files available, makes a .json dump of those for further usage
#: 
```

Figure 2. Tool's command line interface after executing the "help" command

5.2 Directory List Object

To make storing and retrieving information on artifact locations homogenous across the entire program and satisfy the research requirements, a directory list object (“Dirlist”) is implemented. A variable called “data”, stores the required data structure in the directory list object. It has to be a list of tuples. Each tuple has the following fields:

Table 4. Directory List object data structure tuple fields

Name	Type	Description
search_place	String, path-like	Global search place files were found in
directory	String, path-like	Subfolder of search_place files were found in
files	List of strings	List of located files

Hence, the list of such tuples contains a record of each folder that has any interesting files – .plist files in case of the present research – and each such record contains all the said files’ names in it. Concatenating “search_place”, “directory” and one of the filename strings from “files” yields an absolute path to a file of interest.

The “Dirlist” object is created on one of the two following occasions:

- The “gatherall” command is executed to gather information about locally existing .plist files in pre-programmed general search locations. “Dirlist” object holds information gathered from this local system.
- The “commonfiles” command is executed and the provided data dumps are imported. One “Dirlist” is spawned per imported data dump, and it holds information about that data system

The object also has two methods crucial for the whole tool’s operation - “import_json” and “export_json”. These methods govern the import and export of necessary “Dirlist” information so that states can be saved, loaded, and transported between machines as files.

5.3 Main commands

The tool has numerous functions realized as part of its command set. Some of them, such as “help” and “clear”, are for user convenience only, and although they majorly contribute to the usability of the tool, they hold no complicated technological solutions and are not the functions that directly contribute to the research process. Hence, this section focuses on the four main commands of the program: “gatherall”, “commonfiles”, “rankplists”, and “excavate”.

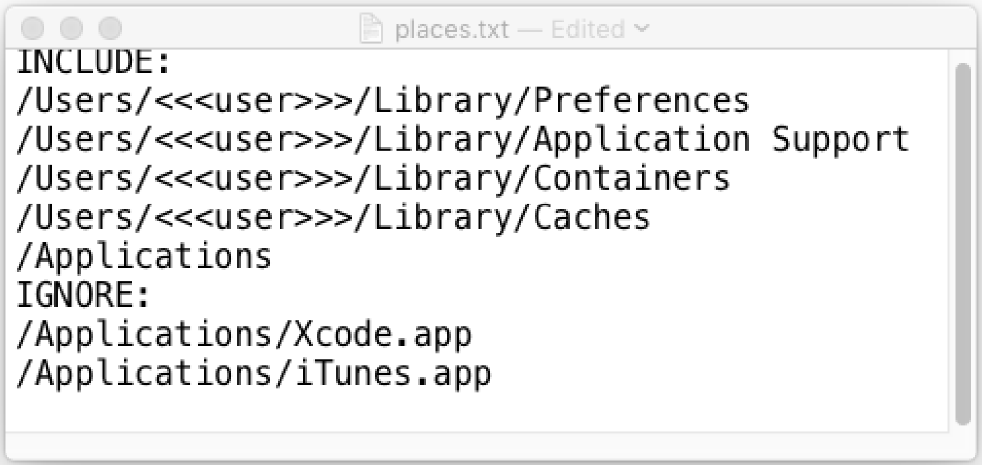
5.3.1 “gatherall”

This command governs the gathering of information from the local machine and is predominantly used for the first step of the research – gathering the interesting file paths from the research volunteers’ personal OS X systems.

The function requires a username argument. This username is then used in conjunction with a file at “lists/bundled/places.txt” to construct a list of global search places. The list has been constructed by the plan outlined in 4.4.1, however, it may be user-modified if needed.

The said file contains two sections with search locations listed on separate lines. The first section contains the paths that are included in the search process. It starts with the “INCLUDE:” directive on a separate line. The second section starts exactly after the first one with a line containing an “IGNORE:” directive.

Places noted in the first section get recursively searched. However, if encountered during the recursive searching, directories noted in the second section (and all their subdirectories) will get ignored. If a located directory contains at least one .plist file, information about its location and files is added as a formatted tuple to the “data” field of the “Dirlist” object that the function produces. After the search is concluded, the “Dirlist” object’s state is exported to a .json file using the object’s built-in method.



```
INCLUDE:
/Users/<<<user>>>/Library/Preferences
/Users/<<<user>>>/Library/Application Support
/Users/<<<user>>>/Library/Containers
/Users/<<<user>>>/Library/Caches
/Applications
IGNORE:
/Applications/Xcode.app
/Applications/iTunes.app
```

Figure 3. Contents of the "places.txt" file

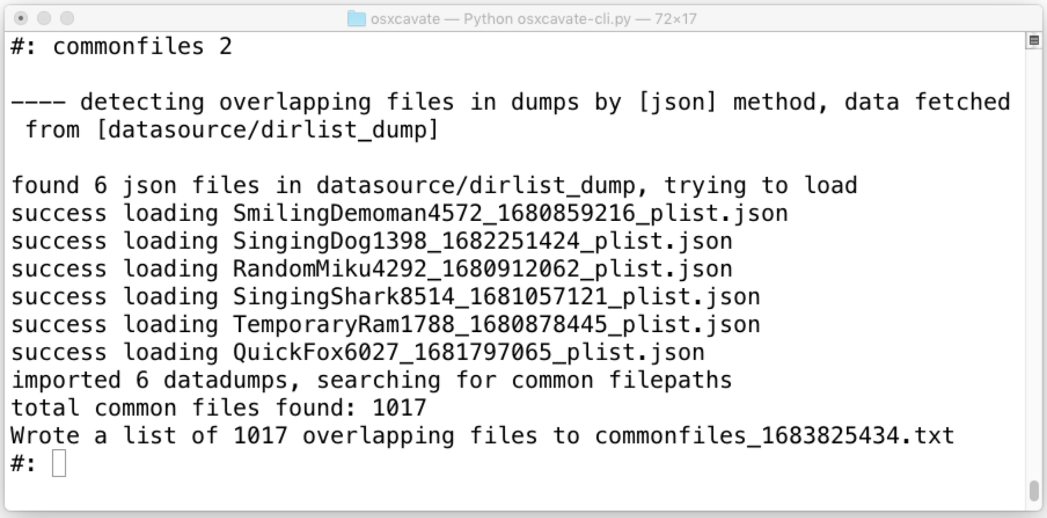
To construct locally existing search paths, the "<<<user>>>" field gets automatically replaced with the provided username using the templating functions implemented in the "common.py" module. However, as per the ethical requirements of the present research, the username may not be included in the final output. For this reason, a pseudonym generator is implemented in "common.py" – it randomly generates one of the four million possible easily human-readable pseudonyms that share no relation with execution time, provided username, or anything else potentially identifying, while being easily human-readable for easier transport. This pseudonym is used instead of the provided username in the exported .json file to keep the data subject pseudonymized.

5.3.2 "commonfiles"

This command implements the routine depicted in Figure 2 – finding common file paths between data exports from multiple machines. It reads "Dirlist" .json data dump files from a hard-coded subfolder inside the tool's own folder: "datasource/dirlist_export". Given there are two or more files manually loaded into the correct folder, the function will output a list of file paths that are common between the .json dumps to "export/commonfiles".

The “commonfiles” function requires a tolerance argument to be provided. Tolerance is expected to be an integer that is less than the number of .json in “datasource/dirlist_export” files being processed. If this argument is set to 0, then a file path only appears in the exported output if all the input .json files had it present, i.e. such a file was found on all the surveyed machines. However, a positive tolerance argument N allows to ignore the absence of a file path on N dumps. For instance, if 6 data dump files are input, a tolerance of “1” means a path will appear at the exported “commonfiles” result if it appeared on 6 – 1 = 5 files. This allows fine-tuning the width of the common file paths set to not be too restrictive, but also not include files that are too uncommon.

The main output of “commonfiles” is a .txt file at “export/commonfiles” marked with the execution time unix timestamp of 1-second resolution. In the file is a list of file paths that are common between the provided “Direntry” data dumps generated with “gatherall” on various machines. Paths are separated by a newline. Since all input .json dumps may have a pseudonymized username in them, “commonfiles” detects it and replaces it with “<<<user>>>”, fit for further templating when the exported file is reused.



```
#: commonfiles 2

---- detecting overlapping files in dumps by [json] method, data fetched
 from [datasource/dirlist_dump]

found 6 json files in datasource/dirlist_dump, trying to load
success loading SmilingDemoman4572_1680859216_plist.json
success loading SingingDog1398_1682251424_plist.json
success loading RandomMiku4292_1680912062_plist.json
success loading SingingShark8514_1681057121_plist.json
success loading TemporaryRam1788_1680878445_plist.json
success loading QuickFox6027_1681797065_plist.json
imported 6 datadumps, searching for common filepaths
total common files found: 1017
Wrote a list of 1017 overlapping files to commonfiles_1683825434.txt
#:
```

Figure 4. “commonfiles” command being executed

5.3.3 “rankplists”

The “rankplists” command allows for manual review, ranking, and group assignation of files, and exports a .json file describing the resulting forensic artifact map. It requires three arguments: file location, username, and mode. The provided .txt file is expected to contain a list of paths of forensically interesting files with the username location swapped for a “<<<user>>>” template marker. This file can be located anywhere in the system, however, it usually would be an exported result of “commonfiles” stored in “export/commonfiles”. The username is expected to be a real username of the local machine user – it will be used to construct actual file paths by templating it in where necessary. Finally, the “mode” argument has three options: “rank” to run the main ranking routine, “count” to display how many files are live on the local OS X system versus how many file paths are noted in the provided .txt list, and “filenames” to export the existing files’ real paths to a .txt file at “exports/rankplists”.

Naturally, the most useful mode of the three is “rank”. It iterates the noted files, attempting to open and parse each as a .plist file. If the program fails to open a file – i.e. the file path is noted in the source .txt list, but the actual file happens to be absent on this particular system – it is skipped. If the program successfully opens a file but fails to parse it as a .plist, it displays a corresponding message and the ranking continues.

After displaying each file, three prompts appear one after another, prompting the user to assign groups, rank, and an arbitrary textual comment respectively. For ease of operation, each group is assigned a digit, and the group prompt expects a sequence of selected digits representing groups, not the group names. If any of the prompts encounter an error, “rankplists” attempts prompting again, thus ensuring that progress is not lost upon providing unexpected data. Providing no groups and no rank (pressing “Enter” two times) discards the file from the final artifact map output.

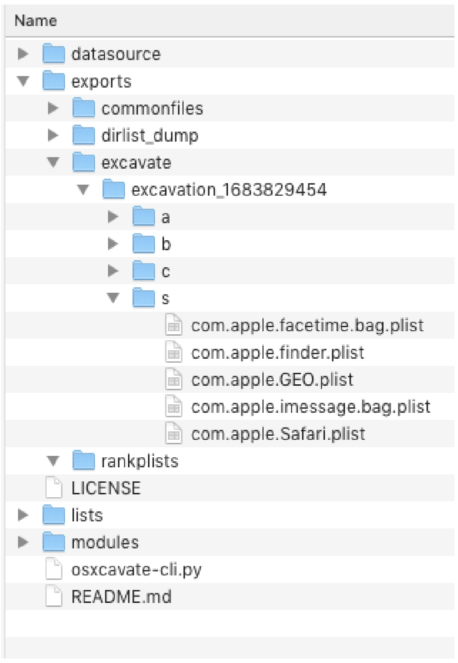
The resulting artifact map is stored at “exports/rankplists” once each of the files noted in the input list is either graded or discarded. Its structure is a list of dictionaries. Each dictionary contains an absolute path to the artifact with a “<<<user>>>” template marker instead of a real username, its rank, a list of its group names, and the textual comment.

5.3.4 “excavate”

The most practically useful command of the tool, “excavate” requires a valid local machine user name as its single argument, and automatically assembles the interesting files from across the local system.

Firstly, this command fetches information from “lists/bundled/files.json”. In the case of the final version of the tool, “files.json” is the output of “rankplists” command ran as part of the research – essentially, a copy of the artifact map presented in Appendix 2; this file can be edited or replaced but has to adhere to the “rankplists” export .json structure.

Finally, the fetched information is used to locate the files from the artifact map and copy them to the tool’s subfolder “export/excavate”. A subfolder structure is created; folders named after ranks from Table 3 are created in a common subfolder. The common subfolder includes an integer UNIX timestamp to universally indicate the time “excavate” has been executed. If located on the system, the files from the artifact map are copied to the folder corresponding to their respective rank. This output may be copied to a portable volume or compressed into an archive and sent over the Internet for further inspection.



```
Name
▶ datasource
▼ exports
    ▶ commonfiles
    ▶ dirlist_dump
    ▼ excavate
        ▼ excavation_1683829454
            ▶ a
            ▶ b
            ▶ c
            ▼ s
                com.apple.facetime.bag.plist
                com.apple.finder.plist
                com.apple.GEO.plist
                com.apple.imessage.bag.plist
                com.apple.Safari.plist
    ▼ rankplists
  LICENSE
▶ lists
▶ modules
  osxcavate-cli.py
  README.md
```

Figure 5. Files and folders generated by running “excavate”

6 Data Processing

6.1 File Path Information Collection

As per the research plan, paper [12] is checked against a live OS X system to find the folders containing numerous .plist files of interest in itself and its subfolders. The final selection yields a list of general search places – starting points from which the tool recursively searches for the required files. As planned, the folders in the selection are four layers deep at most and are generated by the system itself in any real-life case. These folders are compiled into a properly formatted list and saved as "places.txt" to "lists/bundled/" inside the tool folder for future use. The resulting selection is presented in Table 5.

Table 5. Locations for automatic searching

Location	Description
/Users/<<<user>>>/Library/Preferences	Contains a lot of .plist files with user-set preferences for apps etc
/Users/<<<user>>>/Library/Application Support	Contains sone assorted .plist files generated by apps
/Users/<<<user>>>/Library/Containers	Apps put some files here temporarily and should clear this out[15], but it may still contain residual information
/Users/<<<user>>>/Library/Caches	May contain residual info similarly to /Containers
/Applications	Folder with installed programs, each has a plethora of files inside its own package

The list formatting allows to specify directories that shall be ignored if encountered during the search. This functionality helps improve the usefulness of the resulting data and saves up some of the manual assessment time. The folders that were set to be ignored are the two applications that are common across many OS X systems and happen to generate immense amounts of .plist files and other data of no forensic use[3].

It has to be noted that despite OS X stores its applications in what seems to be a .app format file, those are, in fact, ordinary folders that can be examined and have rich substructure. The ignored locations are presented in Table 6.

Table 6. Ignored locations

Location	Reason
/Applications/Xcode.app	Generates thousands of .plists if the OS X user develops using Xcode, mostly with technical data on Xcode settings and such.
/Applications/iTunes.app	Contains hundreds of .plists related to license agreements and other uninteresting information

The list of searched and ignored locations is composed and the tool functionality is sufficient to gather information on .plist file paths and validated by local testing. The tool folder is then compressed into an archive file and sent out to the individuals volunteering to participate in the research as data subjects, accompanied by a text file containing detailed explanations of the actions needed to be conducted, as well as what data is the tool exactly collecting and how to verify correct pseudonymization and non-excessive information collection.

The data is acquired using the tool's "gatherall" method (Chapter 5) on six real-life personal systems running OS X High Sierra and above. The collaborators come from different professional backgrounds, predominantly arts & media or information technologies. The returned data dump .json files are stored on the researcher's machine for further usage.

6.2 Structuring and Assessment

After the period of data gathering is finished, the .json data dumps are rounded up and put to "datasource/dirlist_export" for the tool to read from. Each export file contains information about the locations of .plist files on the machine that the file comes from. The following structuring and assessment actions are aimed at creating a forensic .plist artifact map.

First of all, “commonfiles” and “rankplists” in “count” mode are used in conjunction to determine the most suitable “commonfiles” tolerance argument value. For each of the valid tolerance values from 0 until one less than the overall number of surveyed machines, which is 5 in the case of the present study, the following sequence of actions is taken:

- Run “commonfiles” with a tolerance, locate the generated path list .txt file
- Run “rankplists” in “count” mode, aim it at the freshly generated path list
- Note the number of noted file paths and the files located on the researcher machine

There are two reasons behind the last step. Firstly, a lot of applications have .plist file aliases in them which are treated as valid files by Python’s “os” module. This may lead to both duplicate file paths (multiple applications having aliases to the same real file) and dead file paths (alias pointing to a file that does not exist on the system). Secondly, the manual assessment of the files is conducted on the researcher's machine, so it is crucial to get an understanding of how many files may be accessed for review.

Table 7. “commonfiles” tolerance VS file path and located file counts

Tolerance	Common file paths	Located files
0	140	94
1	190	112
2	1017	149
3	4844	256
4	29817	2934
5	40538	4476

The file path and located file counts for each selected tolerance are provided in Table 7. Further research may only use one of the tolerance values. It is evident that when stepping the tolerance up from 3 to 4, the noted common file path count quadruples, and the located file count grows by more than 50%. This implies that the set includes a large portion of files not as common as required per research goals, and many of the located files may be of lesser interest. Hence, the tolerance selected for further research is “2”.

After running the tolerance tests and analyzing the outcomes, the list of common .plist file paths is generated by the “commonfiles” command executed with the selected tolerance. It is collected as the source list for manual assessment. The “rankplists” command is run in its ranking mode and is pointed at the selected source file. Each noted file is accessed and displayed on the screen for manual rank and group assignment. After the routine is finished, the resulting artifact map is exported to “exports/rankplists” as a .json file. Its full contents are presented in a human-readable form in Appendix 2. A short excerpt with only the artifacts ranked “S” is presented as an example in Table 8.

Table 8. S-ranked OS X .plist forensic artifacts

Path	G.	Comment
/Users/<<<user>>>/Library/Preferences/com.apple.GEO.plist	geo, date-time, web, data	last network etag
/Users/<<<user>>>/Library/Preferences/com.apple.imes-sage.bag.plist	datetime, web, data	potential imessage caches
/Users/<<<user>>>/Library/Preferences/com.apple.Safari.plist	geo, date-time, data	some of the safari set-tings/stats
/Users/<<<user>>>/Library/Preferences/com.apple.face-time.bag.plist	datetime, web, data, other	may contain facetime cache data etc
/Users/<<<user>>>/Library/Preferences/.GlobalPreferences.plist	geo, lan-guage, datetime, web, data, path, other	user's global prefer-ences. has a non-ex-haustive list of devices ever connected, rich lan-guage data, etc
/Users/<<<user>>>/Library/Preferences/com.apple.finder.plist	ip, data, path, other	finder datastore - has goto fields' history, nu-merous local and remote paths, user settings, etc

Upon finishing the manual assessment of the artifacts, “rankplists” generates a .json output file containing the entered data in a machine-readable format. This file is renamed to “files.json” and is put to “lists/bundled” along with “places.txt’. This copy of the final forensic artifact map is used by the tool’s “excavate” command as a guide to artifact locations and ranks. As per the research design, the tool’s “excavate” is executed on several machines to gather actual files and analyze their contents, from which the overall tool effectiveness may be evaluated.

7 Analysis

As the study procedures outlined in the research design come to a conclusion, the results are overviewed and analyzed. As per the design, the research produces two related results at its outcome: the OS X .plist forensic artifact map and the tool for artifact discovery and excavation. These two outcomes are analyzed separately, as both their natures and the categories they may be evaluated in evidently differ.

7.1 Forensic Artifact Map Analysis

This section reviews and evaluates aspects of the artifact map composed over the course of the study. The full map is displayed in Appendix 2.

The original map file is exported by the tool's "rankplists" command as a .json file. Although the choice was somewhat debatable at the moment of functionality implementation, it is evident that JSON is the most convenient of all due to its malleability. It can both be easily processed in its raw form by a machine using one of the many programming languages, as well as quickly converted to a human-readable format, like a table version presented in Appendix 2, using the widely-used non-default "pandas" module in case of Python.

The map contains 70 entries total, out of the 149 ranked located files. This means that 46.9% of the programmatically extracted and locally existing files were manually deemed being of forensic value. This percentage is evidently high, especially once it is taken into account how many .plist files are empty or hold strictly operational data of the program. Such high percentage of valuable files at the programmatically aggregated file locations shows that the method for locating and filtering common .plist files is working successfully.

Ranking of .plist files is conducted according to the strict and objective guidelines noted in Table 3. The guidelines were selected so that better ranked artifacts come in smaller quantities, e.g. there should be more “C” ranked ones than “B” ranked, and so on. This ensures the typical pyramid-like distribution, since one higher-ranked artifact should hold more value than a lower-ranked one. The final count of artifacts in each rank are as follows:

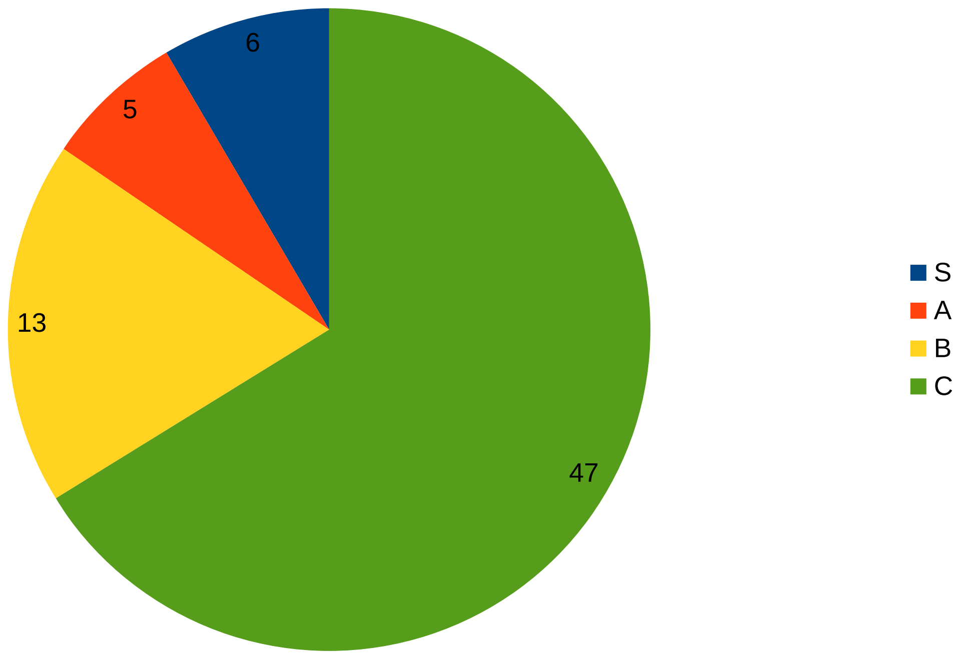


Figure 6. Artifact count per rank

As evident from Figure 6, there is approximately 3.5 times more C-ranked artifacts than the B-ranked ones. Likewise, B-ranked ones outnumber the A-ranked ones about threefold. However, the number of A- and S-ranked artifacts difffer by one. This indicates that generally the guidelines produce the desired result, however, the classifiers for the “S” rank could be made stricter, so that more files get ranked “A” instead and make the distribution closer to ideal.

The artifacts are assigned to groups. Eight groups are available by what presumably are the main classes of information possible to encounter in a file-level forensic artifact. The distribution of files is provided in Figure 7.

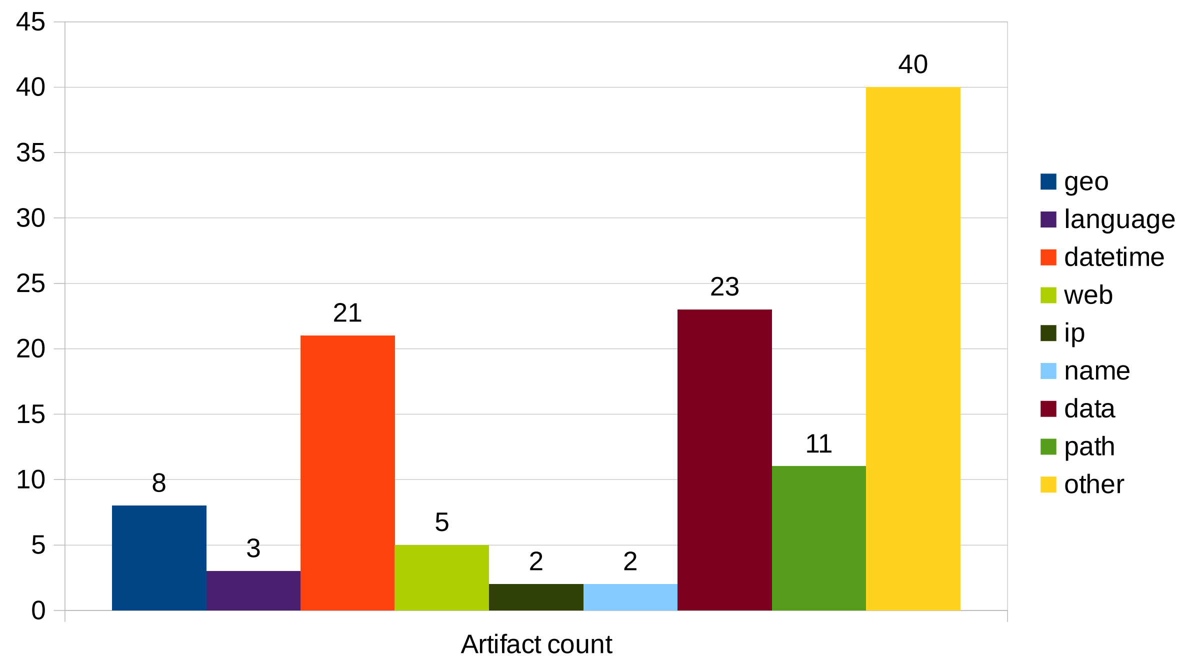


Figure 7. Artifact count per group

The three leading groups are datetime, data and other, followed by geo. It is of high value that many artefacts contain datetime field, as establishing dates and times of events is a big part of digital forensic expertise; the timestamps often indicate events like last app launch, last update, and so on. Many files contain data type fields: these are mostly cache dumps of some of the applications stored as hex byte arrays. These are possible to use given there exists a way to interpret the data generated by a given application.

The dominant category is “other”. Firstly it has to be noted that one artifact may be in many groups – and coincidentially, “other” is a group that mostly appears along with other, more particular group. Secondly, despite being named “other”, this group mostly contains personal user settings, such as mouse sensitivity, GUI settings, and such; these are of high value, as they may be of help for profiling a person. This implies that although the group separation generally produces the desired effect and effectively detects specific information like timestamps and geologic locations, it could be improved by breaking the “other” tier down into more specific groups.

Finally, every artifact in the map has a textual commentary field. Unlike rank and groups, this field may be left empty, and is merely to add context to a given artifact when possible. Out of 70 entries total, 57 entries are complete with a comment field entry. This implies that approximately 84% of the artifacts were successfully contextualized by searching Apple's official documentation and Apple-hosted user forums. The smaller fraction of the artefacts were either not possible to identify within the scope of the present research (e.g. a data dump within a tag of no humanly understandable meaning), or simply did not need additional commentary (e.g. a timestamp within a field clearly labelling which event's timestamp it is).

7.2 Tool Analysis

Efficiency of the tool's main functionality is evaluated by field testing it and critically assessing the test results. Two OS X in direct possession of the research group systems are picked to transfer the tool to and run the "excavate" command on. The output of this command – the folder with the excavated .plist files – is compressed into an archive file and stored on the researcher machine for manual examination and assessment.

Three machines are used as testing subjects. These machines are all personal, however, have been used for different amount of time and with differing intensity, and belong to people of different professional and personal backgrounds. Information about the machines and the global tool runtime statistics are presented in Table 9, followed by a detailed analysis. Since the hardware and Apple product model do not matter nearly as much as the OS X version and time of continuous usage without clean reinstallations, hardware details are omitted.

Table 9. Test subject machines' descriptions

Machine	A	B	C
OS X Version	Big Sur	Catalina	High Sierra
Time w/o fresh install	4 years	1 year	6 years
User professional field	IT	Arts	Engineering, Music
Use intensity	Once every 1-2 days	1-2 times a week	Daily
Located artifacts count	65/70	67/70	69/70

User settings for the main OS X utilities such as Dock, Finder, Spotlight and Launchpad were captured for all three machines along with other miscellaneous forensic information, such as last login time, some of the automatic update and manual setting alteration timestamps, and such. Likewise, caches of FaceApp and Messages are located on all three machines, however, the files are evidently similar – likely due to the fact that neither of the machine owners ever use these platforms.

In addition, for machine A and C, it is possible to recover a list of installed applications, some of the recently accessed files, country code and language settings, recently connected storage devices. On top of that, the machine C export has an IP address of a remote AFP file sharing server frequented by the owner, as well as information on previously installed applications that are no longer present.

Manually interpreting the export results using the artifact map is evidently simple; since most artifacts have textual commentary on their contents in the table. The tool command line interface itself also appears to be simple and efficient, as no volunteering individual had trouble operating it, while deploying and running it on an unprepared system is a matter of minutes.

8 Summary

Modern Mac OS X versions use special files of the “Preference List” (.plist) type to store some of the user settings, options, program configuration, and other data. Collectively, these files have forensic significance, as they may contain fields of interest, such as timestamps, cache dumps, geolocation, language settings, file paths, URLs and IP addresses. The present study explores the forensic capability of these files using a practically inclined design-science approach.

The study yields two results: a forensic artifact map – a table with locations, descriptions and classifications of .plist files of interest, and a practical software tool that features the needed functionality for location, structuring, classification, and excavation of the said files. In conjunction, they produce a working solution prototype.

The map responds to the initial criteria and requirements within a small error margin. The selected methods are evidently useful for creation of such map. Judging by the fact of most files in the map being found on field test machines, the method of filtering out all files except for the ones universally common in OS X is sufficient. In future works, the skeleton of the search process may be reused, but the particular aspects, like the rank definitions and the set of available groups, may be fine-tuned to bring the resulting map even closer to the ideal expectations. An additional automatic method not based on file path ubiquity may be added to include artifacts generated by web browsers.

The tool implements all the planned functionality. It is capable of assembling the file paths while fulfilling the established ethical guidelines. The automatic structuring and manual assessing routines produce viable results in malleable format. Equipped with the resulting artifact map, it consistently excavates more above 90% of the noted files. The tool uses no third party modules and runs on user interface is clear and provides instant guidance to new users. The code allows for easy addition of new commands and functions by making numerous core functions reusable and modular.

While the tool furnishes the present research needs fully, it has its own gaps as a state-of-the art forensic solution. While manual assessment is feasible for a small number of common system-generated .plist files, automation is a crucial need when the artifact count is in the order of thousands or more. Regular expression and artificial intelligence could be employed to facilitate separating the forensically valuable files from the many files of no forensic interest. Finally, as per the research scope, the resulting tool works on live systems only. Built-in disk image reading support would be a valuable functionality expansion for it as a state-of-the-art forensic toolkit part.

Studying the under-explored area file-level forensic capabilities of default OS X proved to be viable. Even from a narrow, intentionally ubiquitous file selection of the same type, it is possible to infer a lot of valuable information, including that of previously connected media and devices, applications deleted at the moment of examination, and other info that would otherwise require tedious disk-level digging to discover. To further explore other artifact formats and discovery methods, the procedures used throughout the present research may be used in their original form, or modified to fit the new research. File-level automatic artefact aggregation compliments other, better furnished layers of OS X forensics, allowing for fast and efficient system examination covering a multitude of its facets.

References

[1] "A study on the APFS timestamps in MACOS", Jong-Hwa Song, Se Ho Kim, Song Yi Hwang, Seung Gyu Kim, Sung-Jin Lee, Div. of information and communication Baekseok University

[2] Article 4(1)(a), GDPR, data.consilium.europa.eu, last accessed 6 April 2023

[3] "Can I delete Containers?", Jerry Dammers, Léonie, discussions.apple.com, last accessed 6 April 2023

[4] "Configuration Management for Mac OS X, It's just Unix Right?", Janet Bass, David Pullman, NIST

[5] "Decoding the APFS file system", Kurt H. Hansen, Fergus Toolan

[6] "Digital forensic analysis of discord on google chrome", Khushi Gupta, Cihan Varol, Bing Zhou

[7] “Design Science in Information Systems Research”, Alan R. Hevner, Salvatore T. March, Jinsoo Park, Sudha Ram et al.

[8] "Development of a Forensic Analyzing Tool based on Cluster Information of HFS+ filesystem", Cho, Gyu-Sang

[9] "Global market share held by leading internet browsers from January 2012 to January 2023", Lionel Sujay Vailshery, statista.com, last accessed 6 April 2023

[10] "Hfs Plus File System Exposition And Forensics", Scott Ware

[11] "Mac Forensics: Mac OS X and the HFS+ File System", Philip Craiger, Paul K. Burke

[12] "Mac OS X Forensic Artifact Locations", Michael Cook, Jake Nicastro TJ Dalzell, Michael Geyer, Austin Truax

[13] "Mac OS X Forensics", Dr. Digvijaysinh Rathod

[14] "MEGA: A tool for Mac OS X operating system and application forensics", Robert A. Joyce, Judson Powers, Frank Adelstein

[15] "Mobile Forensics – The File Format Handbook", Christian Hummert, Dirk Pawlaszczyk

[16] "Operating System Market Share Worldwide", StatCounter Global Stats, statcounter.com, last accessed 6 April 2023

[17] "PLIST Files", Ryan R. Kubasiak, The Internet Archive, last accessed 16 April 2023

[18] "Pseudonymization and impacts of Big (personal/anonymous) Data processing in the transition from the Directive 95/46/EC to the new EU General Data Protection Regulation", Luca Bolognini, Camilla Bistolfi

[19] "The new ext4 filesystem: current status and future plans, Proceedings of the Linux symposium. Vol. 2. 2007.", Mathur, Avantika, et al.

[20] "Why Businesses Choose Python for Prototyping and Production", Will Serene, Adam Gao, for RevGen, last accessed 2 May 2023

Appendix 1 – Non-exclusive Licence for Reproduction and Publication of a Graduation Thesis[1]

I, Nikita Timokhin

1 Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Structural Assessment and Automated Aggregation of OS X Forensic Artifacts”, supervised by Pavel Tšikul

  1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

  1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2 I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3 I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

15.05.2023

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Full OS X .plist Forensic Artifact Map

Table 10. Full OS X .plist forensic artifact map

Path	Rank	Group	Comment
/Users/<<<user>>>/Library/Preferences/com.apple.-FontRegistry.user.plist	b	path, dat-etime	
/Users/<<<user>>>/Library/Preferences/com.apple.GEO.plist	s	geo, date-time, web, data	last network etag
/Users/<<<user>>>/Library/Preferences/com.apple.Fol-derActionsDispatcher.plist	c	data	
/Users/<<<user>>>/Library/Preferences/com.apple.uni-versalaccessAuthWarning.plist	b	path, other	lists applications author-ized for full disk access
/Users/<<<user>>>/Library/Preferences/com.apple.Ap-pleMultitouchMouse.plist	c	other	mouse preferences
/Users/<<<user>>>/Library/Preferences/com.apple.Ap-pleMultitouchTrackpad.plist	c	other	user trackpad settings
/Users/<<<user>>>/Library/Preferences/com.apple.iApps.plist	c	path	iphoto and other native apps' default photo lib-raries
/Users/<<<user>>>/Library/Preferences/com.apple.Face-Time.plist	c	other	may contain data on ex-ternally connected cam-eras/microphones
/Users/<<<user>>>/Library/Preferences/com.apple.calcu-lateframework.plist	c	datetime, other	last currency converter refresh date and curren-cies' rates for the day
/Users/<<<user>>>/Library/Preferences/com.apple.imes-sage.bag.plist	s	datetime, web, data	potential imessage caches
/Users/<<<user>>>/Library/Preferences/com.apple.mad-rid.plist	c	datetime	
/Users/<<<user>>>/Library/Preferences/com.apple.User-AccountUpdater.plist	c	other	shows if the library folder is hidden from user
/Users/<<<user>>>/Library/Preferences/com.apple.Core-Graphics.plist	c	other	screen grayscale/invert settings
/Users/<<<user>>>/Library/Preferences/com.apple.speech.voice.prefs.plist	c	datetime, other	text-to-speech stats
/Users/<<<user>>>/Library/Preferences/com.apple.ac-countsd.plist	b	data, other	has to do with the local account system
/Users/<<<user>>>/Library/Preferences/com.apple.Safar-i.plist	s	geo, date-time, data	some of the safari set-tings/stats

/Users/<<<user>>>/Library/Preferences/com.apple.Console.plist	c	other	console settings
/Users/<<<user>>>/Library/Preferences/com.apple.suggestions.plist	c	data	spotlight suggestions related
/Users/<<<user>>>/Library/Preferences/com.apple.coreauthd.plist	c	data	coreauthd daemon related
/Users/<<<user>>>/Library/Preferences/com.apple.imservice.ids.iMessage.plist	b	name, other	potential imessage account info
/Users/<<<user>>>/Library/Preferences/com.apple.driver.AppleBluetoothMultitouch.mouse.plist	c	other	bluetooth mouse settings
/Users/<<<user>>>/Library/Preferences/com.apple.systempreferences.plist	b	data, path	general user system settings
/Users/<<<user>>>/Library/Preferences/com.apple.dock.plist	a	datetime, data, path	dock preferences
/Users/<<<user>>>/Library/Preferences/com.apple.SystemProfiler.plist	a	geo, data, other	may contain previously used regions
/Users/<<<user>>>/Library/Preferences/loginwindow.plist	c	other	
/Users/<<<user>>>/Library/Preferences/com.apple.talagent.plist	c	datetime	datetime related to the transparent app lifecycle agent (autosave + versions thing)
/Users/<<<user>>>/Library/Preferences/com.apple.commerce.plist	a	datetime, web, other	has apple id email address
/Users/<<<user>>>/Library/Preferences/com.apple.AddressBook.plist	b	geo, datetime	may have previously used country codes
/Users/<<<user>>>/Library/Preferences/com.apple.icloud.fmfd.plist	c	data	contains some aps key, sensitivity unknown
/Users/<<<user>>>/Library/Preferences/com.apple.audio.AudioMIDISetup.plist	b	datetime, other	audio and midi settings
/Users/<<<user>>>/Library/Preferences/com.apple.TelephonyUtilities.plist	b	data, other	telephony settings
/Users/<<<user>>>/Library/Preferences/com.apple.facetime.bag.plist	s	datetime, web, data, other	may contain facetime cache data etc
/Users/<<<user>>>/Library/Preferences/com.apple.passd.plist	c	datetime	related to the apple pay and wallet daemon (passd)
/Users/<<<user>>>/Library/Preferences/com.apple.assistant.plist	b	geo, datetime	related to apple's assistant (enhanced voiceover service)
/Users/<<<user>>>/Library/Preferences/com.apple.iCal.plist	a	geo, datetime, other	calendar app settings memory; has last view's timezone
/Users/<<<user>>>/Library/Preferences/com.apple.com-	c	data	app store related, sensit-

merce.knownclients.plist			ivity undefined
/Users/<<<user>>>/Library/Preferences/com.apple.stock-holm.plist	c	other	
/Users/<<<user>>>/Library/Preferences/.GlobalPrefer-ences.plist	s	geo, lan-guage, datetime, web, data, path, other	user's global prefer-ences. has a non-ex-haustive list of devices ever connected, rich lan-guage data, etc
/ Users/<<<user>>>/Library/Preferences/com.apple.driver-.AppleHIDMouse.plist	c	other	hid mouse settings
/Users/<<<user>>>/Library/Preferences/com.apple.nc-prefs.plist	b	data, path	appears to list the cur-rently installed apps in /applications
/ Users/<<<user>>>/Library/Preferences/com.apple.Activ-ityMonitor.plist	c	other	activity monitor related
/Users/<<<user>>>/Library/Preferences/com.apple.Safar-i.SafeBrowsing.plist	c	datetime	
/Users/<<<user>>>/Library/Preferences/com.apple.tex-tInput.keyboardServices.textReplacement.plist	c	name	contains username
/Users/<<<user>>>/Library/Preferences/com.apple.i-Tunes.plist	c	other	may have recent itunes searches
/Users/<<<user>>>/Library/Preferences/com.apple.-java.util.prefs.plist	c	data	has names of ever con-figured java applications
/Users/<<<user>>>/Library/Preferences/com.apple.secur-ity.cloudkeychainproxy3.keysToRegister.plist	c	other	
/Users/<<<user>>>/Library/Preferences/com.apple.core-services.uiagent.plist	c	other	has names of some of the apps that have been ran
/ Users/<<<user>>>/Library/Preferences/com.apple.HIToo lbox.plist	b	language, other	has installed input meth-ods and the 2 last active methods
/Users/<<<user>>>/Library/Preferences/com.apple.Ser-vicesMenu.Services.plist	c	other	registered translations of the general (right click) context menu entries
/Users/<<<user>>>/Library/Preferences/com.apple.Setu-pAssistant.plist	c	other	has some flags related to events such as did user see the privacy state-ment
/ Users/<<<user>>>/Library/Preferences/com.apple.driver-.AppleBluetoothMultitouch.trackpad.plist	c	other	bluetooth multitouch mouse settings
/Users/<<<user>>>/Library/Preferences/com.apple.xp-c.activity2.plist	b	datetime, other	
/Users/<<<user>>>/Library/Preferences/com.apple.sys-temuiserver.plist	c	other	topmost bar left side se-lected widgets list
/Users/<<<user>>>/Library/Preferences/com.apple.Ter-	b	data,	terminal settings

minal.plist		other	
/Users/<<<user>>>/Library/Preferences/com.apple.universalaccess.plist	c	other	universal access settings
/Users/<<<user>>>/Library/Preferences/com.apple.DiskUtility.plist	a	data, path, other	has a root directory, presumably last viewed in disk utility
/Users/<<<user>>>/Library/Preferences/org.cups.PrintingPrefs.plist	c	ip	has ips of last used printers
/Users/<<<user>>>/Library/Preferences/com.apple.Spotlight.plist	c	datetime	timestamp likely the last time of spotlight launch
/Users/<<<user>>>/Library/Preferences/com.apple.finder.plist	s	ip, data, path, other	finder datastore - has goto fields' history, numerous local and remote paths, user settings, etc
/Users/<<<user>>>/Library/Preferences/com.apple.LaunchServices/com.apple.launchservices.secure.plist	c	other	has bindings of formats to programs, may contain uninstalled programs' names
/Users/<<<user>>>/Library/Application Support/icdd/deviceInfoCache.plist	c	data	may contain a device info cache dump
/Users/<<<user>>>/Library/Containers/com.apple.languageassetd/Data/Library/Caches/com.apple.DictionaryServices/DictionaryCache.plist	c	language, other	lists dictionary sources, directly related to selected input sources
/Users/<<<user>>>/Library/Containers/com.apple.QuickTimePlayerX/Data/Library/Preferences/com.apple.QuickTimePlayerX.plist	c	path	may have path of (possibly unexistent) files accessed with quicktime
/Users/<<<user>>>/Library/Containers/com.apple.siri.media-indexer/Data/Library/Preferences/com.apple.siri.media-indexer.plist	c	datetime	timestamps of siri's media indexing routine execution
/Users/<<<user>>>/Library/Containers/com.apple.TextEdit/Data/Library/Saved Application State/com.apple.TextEdit.savedState/windows.plist	c	path	has a path to a .txt file accessed with textedit
/Users/<<<user>>>/Library/Containers/com.apple.Preview/Data/Library/Preferences/com.apple.Preview.plist	c	data	
/Users/<<<user>>>/Library/Containers/com.apple.Preview/Data/Library/Preferences/com.apple.Preview.ViewState.plist	c	data	
/Users/<<<user>>>/Library/Caches/GeoServices/networkDefaults.plist	c	other	
/Users/<<<user>>>/Library/Caches/GeoServices/Resources/supportedCountriesDirections-20.plist	c	geo	
/Users/<<<user>>>/Library/Preferences/com.apple.AppStore.plist	c	other	

Appendix 3 – Code Excerpt and Tool Location

The following code is an excerpt from the main command line interface loop of the resulting tool. It showcases the four main functions and the general CLI logic, while omitting the more utilitarian code. The full code is available from the code repository located at https://github.com/kouyouelysian/osxcavate, the version resulting from the present research is commit auto-labelled "e372e1f", dating May 15, 2023.

```
# [... code omitted ...]

def gather_artefacts_all(args=None,h=False):
    """ exports dumps of all .sqlite and .plist filepaths in preprogrammed interesting locations """
    params_description = [\
    ("Please, input a valid username to gather artefacts on","user",None)\
    ]
    if (h):
        print("[?] gatherall <username>:\n    Scans a predefined list of folders based on the
provided username")
        print("    checks for all .plist files available, makes a .json dump of those for further
usage")
        return
    params = cli.get_params(params_description, args)
    if params == None:
        return
    user = params[0]
    global v
    p = common.pseudonym_generate()
    plist  = actions.scanplaces_plist(user, pseudonym=p, verbose=v)
    plist.export_json(p+"_"+str(common.timestmap())+"_plist")

# [... code omitted ...]

def rank_plists(args=None, h=False):
    """ a loop for manually ranking plist files from common_files-generated list """
    mode_options = ["rank", "count", "filenames"]
    params_description = [\
    ("Select input file - has to be a list of filenames generated by common_files","file"),\
    ("Please, input a valid username to access artefacts of","user"),\
    ("Select one of operation modes: rank, count, filenames","inlist",mode_options)\
    ]
    if (h):
        print("[?] rankplists <file> <username> <mode>\n    manually assign rank/groups to plist
files from a common_files-generated list")
        return
    params = cli.get_params(params_description, args)
    if params == None:
        return
    filename, user, mode = params[0], params[12], params[2']
    files = []
    fh = open(filename, 'r')
    for line in fh:
        files.append(common.template_fill_field(line, 'user', user).replace("\n", "").strip())
    fh.close()
    count = 0
    livefiles = []
    for f in files:
        fh = None
        try:
            fh = open(os.path.realpath(f), 'rb')
        except Exception as e:
            if (v == "vvv"):
                print("[x] failed to open "+f+"; file likely missing")
        if not fh == None:
            livefiles.append(f)
            count += 1
            fh.close()
    if mode == "count":
print("listed filepath count:",len(files))
        print("located file count:   ",count)
    elif mode == "filenames":
        fname = "rankplists_files_"+str(common.timestmap())+".txt"
        fpath = os.path.join(os.path.join(os.getcwd(), "exports/rankplists/"), fname)
        common.strlist_to_file(fpath, livefiles, v)
    elif mode == "rank":
        groups = ["geo","language","datetime","web","ip","name","data","path","other"]
        ranks = ["s", "a", "b", "c", ""]
        counter = 1
        out_data = []
        for f in livefiles:
            clear()
            data_row = {"path":None,"groups":None,"rank":None,"comment":None}
            # [... code omitted ...]
            out_data.append(data_row)
        name = "rankings_"+str(common.timestmap())+".json"
        fname = os.path.join(os.path.join(os.getcwd(), "exports/rankplists"), name)
        out = common.str_to_file(fname, json.dumps(out_data, indent=4), v)
        clear()
        print("ranking finished successfully")
    return


def excavate(args=None,h=False):
    """ excavates interesting files by the built-in .json list, exports to /exports/excavate """
    global v
    if (h):
        print("[?] excavate <username>\n    gathers known artefacts from this system for a particu-
lar user")
```

```
        return
    params_description = [\
    ("Please, input a valid username to export artefacts of","user",None)\
    ]
    params = cli.get_params(params_description, args)
    if params == None:
        return
    user = params[0]
    export_path = os.path.join(os.getcwd(), "exports/excavate",
"excavation_"+str(common.timestmap()))
    os.makedirs(export_path)
    for subf in ["s","a","b","c"]:
        os.makedirs(os.path.join(export_path, subf))
    artefacts_store = os.path.join(os.getcwd(), "lists/bundled/files.json")
    fh = open(artefacts_store, 'r')
    contents = fh.read()
    fh.close()
    artefacts = json.loads(contents)
    actions.export_artefacts(user, artefacts, export_path, v)


def common_files(args=None,h=False):
    """ finds overlapping files between json dumps from /datasource/json """
    global v
    if (h):
        print("[?] commonfiles <tolerance>\n    compares existing data dumps and finds out which
files exist across all of them")
        return
    opts = ["json"]
    params_description = [\
    ("Input elimination tolerance - positive integer less than number of data dumps
processed","int_pos")\
    ]
    params = cli.get_params(params_description, args)
    if params == None:
        return
    tolerance = params[0]
    method = "json" # this is here so that maybe later on some other data storing method gets added
    if (method == "json"):
        source = "datasource/json"
    out = actions.overlapping_files(source, method, tolerance, v)
    if out == False:
        return
    fname = "commonfiles_"+str(common.timestmap())+".txt"
    fpath = os.path.join(os.path.join(os.getcwd(), "exports/commonfiles/"), fname)
    common.strlist_to_file(fpath, out, v)
    print("Wrote a list of", len(out) ,"overlapping files to", fname)


# [... code omitted ...]
```