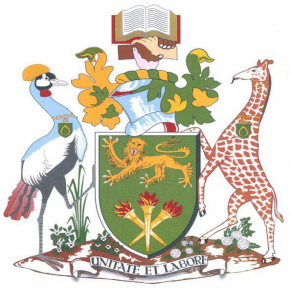
Incorporating Mobile Forensics into Nugget



Eric Chomba Ng'ang'a
P53/6504/2017

MSc Distributed Computing Technology

School of Computing and Informatics

University of Nairobi

Supervised by Dr. Elisha Abade

Submitted on 31st July 2019

Submitted in partial fulfilment for the award of MSc Distributed Computing Technology

Declaration

STUDENT

This project, as presented in this report, is my original work and has not been presented for any award in any other university.

Signed …… Date ……

Eric Chomba Ng'ang'a
P53/6504/2017

SUPERVISOR

This research project is submitted as a partial fulfillment for the award of Master of Science Degree in Distributed Computing Technology with my approval as the university supervisor.

Signed …… Date ……

Dr. Elisha Abade
School of Computing and Informatics

Acknowledgements

I sincerely thank everyone who contributed to the success of this project. I thank my supervisor for being available and resourceful as well as providing me with guidance during the undertaking of this project.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADB Android Debug Bridge

AFF Advanced Forensic Format

AFF4 Advanced Forensic Format 4

ANTLR ANother Tool for Language Recognition

AOSP Android Open Source Project

ARFF Attribute-Relation File Format

ASCII American Standard Code for Information Interchange

BGA Ball Grid Array

CFG Context Free Grammar

CLI Command Line Interface

CTI Cyber Threat Intelligence

CybOX Cyber Observable eXpression

DDL Data Definition Language

DEX Digital Evidence eXchange

DFRWS Digital Forensics Research Workshop

DFXML Digital Forensics XML

DSL Domain Specific Language

EBNF Extended Backus-Naur Form

eMMC embedded MultiMediaCard

EWF Expert Witness Format

FTK Forensic Tool Kit

GLONASS Global Navigation Satellite System

GPS Global Positioning System

GUI Graphical User Interface

HLR Home Location Register

IEEE Institute of Electrical and Electronics Engineers

JSON JavaScript Object Notation

JTAG Joint Test Action Group (JTAG)

MIME Multipurpose Internet Mail Extension

NAND Not-AND

NOR Not-OR

RPC Remote Procedural Call

SCARF SCAlable Realtime Forensics

STIX Structured Threat Information Expression

TAP Test Access Port

TSK The Sleuth Kit

TUI Text User Interface

VLR Visitor Location Register

XIRAF XML Information Retrieval Appproach to digital Forensics

XML eXtensible Markup Language

XSD XML Schema Definition

Abstract

Mobile devices are a significant component in our daily lives. Their ability to be ubiquitous gives digital forensic investigators the ability to gain a deeper insight into cases they are investigating. The use of mobile devices in digital forensic investigations is on the rise due to the wealth of information they contain. To address the increasing heterogeneity of digital forensic tools, Stelly and Roussev (2018) came up with Nugget. Nugget is a digital forensics Domain Specific Language (DSL) that provides a formal and useful way to describe digital forensic computations that abstracts out their implementation. Nugget gives investigators the ability to declare forensic operations to perform on digital evidence without specifying how they are to be done. However, Nugget does not currently support mobile digital forensics. The objective of this study is to extend Nugget by enabling it to declare and execute mobile digital forensic computations. This was done by integrating available forensic tools with Nugget and extending Nugget's DSL to accommodate mobile forensic computations.

Keywords *Nugget, mobile device forensics*

Chapter 1

INTRODUCTION

1.1 Background

Digital forensic investigators are charged with the responsibility of collection, extraction, analysis and reporting of data from digital devices. In order to accomplish these tasks, investigators use a myriad of tools. Each of these tools has specific characteristics that make it different from the rest. These differences are evident in a tool's properties such as (*a*) how the tool is installed or setup, (*b*) how the user interacts with the tool, for example, via Graphical User Interface (GUI) or Text User Interface (TUI) Command Line Interface (CLI), (*c*) what inputs the tool accepts, (*d*) how the tool outputs information or results and (*e*) how correct or accurate the results are. Due to these differences, the tools have different ways of specifying and implementing a specific digital forensic task. The increase in the number of available digital forensic tools available for use has made the differences to grow exponentially. The investigator requires to understand how to correctly and effectively use a tool and how to interpret the tool's results correctly.

To select the best tool for a specific forensic task, an investigator needs to know how to evaluate a tool to determine if it would work as intended. Evaluation of forensic tools has proved to be cumbersome. Since each tool has a specific way of specifying a task to be executed, an evaluator first requires to know how all the tools being evaluated are used. Secondly, the evaluator would write a specification for a task to be executed and its success criteria. Next, the evaluator runs the tool through the specification, either manually or through automation. Based on the results given by the tool, the evaluator can assess whether the tool works as desired and if it is accurate.

The diversity in the implementation and use of these tools has flourished because of the lack of, or minimal use of formal specifications for forensic tasks and representation of results. Raghavan (2013) notes that there is a diversity problem brought about by the existence of multiple forensic tools as well as a significant variation in the types and sources of digital evidence. An analysis that was done by Lillis, Becker, O'Sullivan and Scanlon (2016) shows that the complexity of sharing digital evidence between investigators has increased due to the existence of multiple digital forensic storage standards.

Formal specifications or standards allow tools to align together. The harmony brought about by standards creates a firm base on which information exchange and integrations can occur.

To address the increasing heterogeneity of digital forensic tools, Stelly and Roussev (2018) came up with Nugget. Nugget is a digital forensics DSL targeted at digital forensic investigators. It provides a formal and useful description of digital forensic computations (Roussev, 2015) that abstracts out their implementation. Nugget aims to provide a formal specification of the operations to be carried out. Nugget's primary target users are digital forensic investigators, who can either be technical or non-technical. Technical investigators have some understanding of computer systems or programming while non-technical investigators may not. The DSL is designed to meet the needs of both kinds of investigators by employing syntax that is easily recognisable by forensic domain experts.

Nugget is designed to work in conjunction with the various digital forensic tools available. It does not seek to implement any of the tasks done by existing forensic tools. Nugget integrates with the tools required to perform a task. Currently, Nugget has been integrated with The Sleuth Kit (TSK) (Carrier, 2018a) for disk forensics, Volatility (The Volatility Foundation, 2018) for memory forensics and *tshark* (Wireshark Foundation, 2018) for network forensics. A sample of the Nugget DSL is presented in Listing 1.1.

Listing 1.1: Sample Nugget snippet

```
1 files = "forensic-target.E01" | extract as ntfs
2 videos = files | filter filename=="/*.m4v"
3 // print to stdout
4 print videos
```

In this example, files are extracted from the specified forensic target image and files with the extension *m4v* are filtered from the list of extracted files. The files paths are then displayed to the user. An example of some of the output from the Nugget interpreter is presented in Listing 1.2.

Listing 1.2: Sample Nugget response snippet

```
nugget> files = "forensic-target.E01" | extract as ntfs
nugget> videos = files | filter filename=="/Videos/*.m4v"
nugget> print videos
[
  /Videos/MontereyKitty.m4v
  /Videos/MontereyKittyHQ.m4v
  /Videos/TiggerTheCat.m4v
  /Videos/Cat.m4v
]
```

1.2 Problem definition

Mobile devices are essential sources of digital information used in investigations. Tools have been developed to extract information from mobile devices such as communication records, images, videos and much more. However, these tools have not yet been integrated into Nugget.

Nugget separates the complexities of implementation of such a tool from the specification of how the tool is applied. Different tools have different ways of displaying output results of a forensic task. Without standardisation, tools being integrated would have too much influence on the DSL making it change frequently.

1.3 Research Objectives

1. To review and evaluate existing standards of digital forensic information containers
2. To review and evaluate existing mobile forensic tools
3. To design and implement integrations of forensic tools with Nugget
4. To test the implemented integrations using publicly available digital forensic corpora

1.4 Justification

This research project seeks to embark on adding mobile forensics to Nugget and introduce a standard representation of digital forensic evidence. Addition of mobile forensics would allow digital forensic investigators to use Nugget for digital forensics of mobile devices while standardisation of how to represent digital forensic evidence presents an opportunity for more integrations and allow Nugget's output to become input into other tools.

1.5 Scope of the study

There are various mobile platforms available each with its own set of tools to perform digital forensics. This research shall limit its scope to the incorporation of open source or publicly available forensic tools that can be invoked via the command line for the Android and iPhone platforms.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

Roussev, Bertino and Sandhu (2016) describe forensics as the systematic application of scientific methods to gather and analyse evidence for legal purpose. Digital forensics is a sub-field within forensics that focuses on the identification, extraction, analysis and reporting of digital evidence stored in digital devices within a legal context. Digital forensics encompasses all digital devices. Mobile forensics on the other hand, is a speciality within digital forensics that focuses on using the digital forensic techniques on mobile devices.

2.1.1 Importance of mobile forensics to investigations

Mobile devices have become increasingly prevalent in our day-to-day activities. These devices come in different forms including smart / feature phones, tablets, smart watches and many more. Users can accomplish more than just communication using the devices. With these devices being almost as powerful as personal computers, mobile devices can handle more and more tasks. This usage has lead to accumulation of users' of confidential and personal information on the devices. This treasure trove of personal data is precious to digital forensic investigators. Acquiring a suspect's mobile device could indicate whom the suspect was communicating with, where they were (e.g. using GPS or telecommunication network's Home Location Register (HLR) and Visitor Location Register (VLR)), multimedia files and much more data. Due to their connectivity, mobile devices offer evidence beyond the contents of the devices. Telecommunication service providers and internet service providers can be sources of data or metadata, e.g. communication logs, while cloud providers can be sources of data, e.g. backups.

Mobile devices have become instrumental in providing inculpatory or exculpatory evidence to investigators, law enforcement officers and subsequently prosecutors. For example, mobile devices were instrumental cases such as the 2010 NewYork car bomb investigation (Forensicon Inc., 2010; Mazzetti, Tavernise & Healy, 2010) and the Capital Market Authority's (CMA) investigation into suspected insider trading at Kenol-Kobil (Omondi, 2019).

2.1.2 Challenges faced in mobile device forensics

While working with mobile devices, digital forensic investigators, as well as researchers, have identified some challenges:

1. Hardware and software variance

   Hardware used by mobile devices is different. Manufacturers update their hardware and software often thus increasing the fragmentation of the available devices. The operating systems used on the devices further fragment the devices. The most common mobile device operating system, Android, has various releases, all of which are running in different devices. This variance requires digital forensic investigators to adopt different approaches to collecting evidence for seemingly similar devices. Forensic tool developers require to also keep up with changes introduced to make sure that their forensic tools work with the updated devices.

2. Security measures

   Since mobile devices store personal and confidential information, device manufacturers and software developers have enhanced the protection of this data (Skulkin, Tindall & Tamma, 2018). Measures to protect the data include full-disk encryption, password or PIN locks, remote data wiping or remote factory reset, application sandboxing. Some mobile devices enhance this protection by encrypting backups of the device data.

   These measures are essential to protect user information. However, they present a hindrance to investigators. Without the encryption key, encrypted data is not usable, even if it is extracted. Brute forcing passwords or PINs on some devices may result in the device wiping all the data it contains. Suspects may wipe the data from their device even after the device is in the investigator's possession.

3. Ease of altering evidence on the device

   Information stored on a mobile device can be casually manipulated, whether intended or not. For example, the simple act of waking a locked mobile device from standby mode could change data. In the process of picking up a confiscated device, an investigator may look at a device. A device that employs facial recognition technology to unlock may unsuccessfully attempt to recognise the investigator's face. This simple act can cause the device to harden its defences, e.g. requiring a password or PIN code, if it detects the wrong face a number of times.

4. Offline and Online data storage

   Mobile devices with access to the internet can store data both on the device and in the cloud. Data that's on the device could present a partial representation of the true picture. Thus, investigators may require to include data stored in the cloud in the investigation. Determining what data is stored in which cloud service may prove to be a challenging task.

5. Rise of the amount of devices to be analysed

   The rise in use of mobile devices has lead to a significant increase in their use in investigations.

6. Device connectivity

   Mobile devices offer user connectivity via multiple channels e.g. cellular, WiFi, Bluetooth, Infrared, NFC. Once a device has been taken into custody, communication via the available channels requires to be discontinued e.g. by using Faraday bags. This step is necessary so as to ensure that the device data isn't altered by any of the connectivity channels.

7. Power requirements Mobile devices get powered via portable batteries. If a device has been found, it requires to be powered lest its power supply runs out and it shuts down.

8. Anti-forensic techniques

   A number of users have developed techniques that seek to counter forensic acquisition and analysis of data in their devices. Such techniques include secure wiping a device's contents, data hiding and data obfuscation. Anti-forensic techniques may be as sophisticated as detection of Faraday bags and triggering a full device wipe.

Mobile forensics typically begins with the identification of a mobile device and its seizure, getting it under the control or custody of the investigator. The device should be handled and stored with care since it affects how data will be acquired.

2.1.3 Data acquisition from mobile devices

Data extraction methods can be categorised into two major categories: (*a*) Logical extraction and (*b*) Physical extraction. Logical extraction involves using software that mediates data access from the storage locations in the device. The user uses the software to access the stored data. Software that manages access to the device's stored data could be the

device's Operating System or any other program that provides access to the device. This method is limited since the data extracted is limited to what the software has access to or what the user has access to. In modern mobile devices, security measures employed on the devices ensure that installed applications and users access just what they require and thus access to raw memory or raw disk information is limited or non-existent. To overcome this challenge, physical extraction is used. It employs the use of hardware and software to extract information directly from the mobile device without the intervention of any mediating software. Physical extraction grants investigators access to raw memory or storage information. Access to raw memory or storage allows investigators to get more information from the media such as deleted files. Components from the mobile device, such as the embedded MultiMediaCard (eMMC), can be extracted from the mobile device's circuitry and read using specialised hardware. This particular approach demonstrates that physical extraction is more destructive to a mobile device than logical extraction.

Beyond these two categories, (Brothers, 2011) proposed a way to group techniques applied to retrieve information from mobile devices. The proposed hierarchy, as shown in Figure 2.1, has five groups.

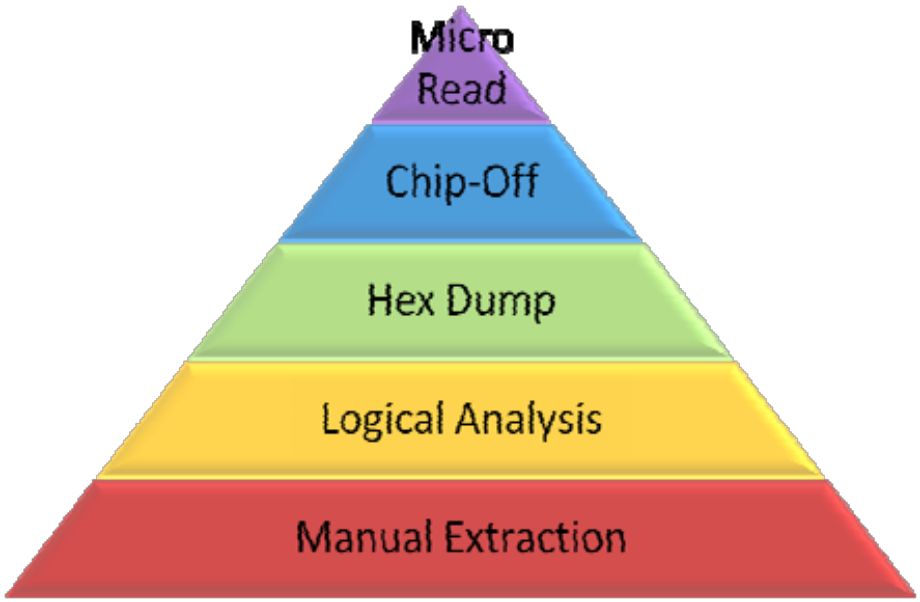


Figure 2.1: Tool hierarchy. Source: Brothers, 2011

Moving up the pyramid, the techniques used are highly specialised and technical, they take longer times to analyse, they require more sophisticated and expensive tools and are more invasive to the device (Murphy, 2011).

- Manual extraction

  It's the most straightforward technique involving reviewing the mobile device's documentation and manually browsing the device using its buttons or touch interface to look at and record the information presented on the screen. Data recorded is typically done using photographs. This method will not get all the data held in a device, is prone to human error, time-consuming and will not work on broken or damaged devices. Additionally, using the mobile device's buttons may interfere with or alter the evidence on the device, e.g. reading unread messages.

- Logical extraction

  The mobile device being examined is connected to a computer using a wired connection (e.g. USB data cable) or a wireless connection (e.g. Bluetooth, WiFi) and information is extracted from the device. The investigating computer sends commands to the mobile device, which responds with the requested data.

- Hex dump / Joint Test Action Group (JTAG) (JTAG)

  Hex dump is a way for raw data to be extracted from a mobile device's memory storage. A custom boot-loader is loaded onto the mobile device being investigator using a computer. The boot-loader dumps the phone's memory.

  An Institute of Electrical and Electronics Engineers (IEEE) standard entitled *Standard Test Access Port and Boundary-Scan Architecture*, developed by the JTAG association, is a way of accessing a device's raw data when connected to a standardised Test Access Port (TAP) on its circuit board.

- Chip-off

  In this approach, the digital forensic examiner removes a mobile device's chip from its circuit board and reads data from it (Mikhaylov & Skulkin, 2016). The device chip is an eMMC that combines a flash memory controller, flash memory and the multimedia card interface integrated on the same Ball Grid Array (BGA) package (JEDEC Solid State Technology Association, 2007). This technique is especially useful when dealing with devices that may have been damaged or devices that may have a locked bootloader.

- Micro read

  An investigator analyses the physical gates on a Not-AND (NAND) or a Not-OR (NOR) chip using an electron microscope (Tahiri, 2016) (Tamma, Skulkin, Mahalik

& Bommisetty, 2018). The analysis results into a stream of 0s and 1s which can be decoded into human-readable data, e.g. into ASCII characters. It is a long and painstaking process that is rarely done.

2.2 Nugget

Nugget is an external DSL that allows users to describe forensic computations. Fowler (2010) defines a DSL as a computer programming language of limited expressiveness focused on a particular domain. Fowler further categorises a DSL as either an external DSL, an internal DSL or a language workbench. A language workbench is a specialised development environment that allows users to define and build DSLs that results in scripts that tightly couple the editing environment with the language. An internal DSL is one that relies on the host general-purpose language. The application that uses the DSL provides the host language that supplies the superset of the language features. Unlike an internal DSL, an external DSL is separate from the primary language of the application it works with. An external DSL defines its own syntax, parsers, lexers, code generation and compilers. Examples include SQL and Awk.

In Nugget, forensic computations can be specified using any one of four types of operators: extractors, filters, transformers and serialisers. Listing 2.1 demonstrates these operators.

Listing 2.1: A sample Nugget script showing file extraction, filtering and hashing on an NTFS volume. Source: Stelly and Roussev, 2018

```
1 files  = ”file:harddrive.EO1” | extract as ntfs [63,512]
2 jpgs   = files | filter name==”*.jpg”
3 hashes = jpgs.content | sha1, md5
4 jpgs = jpgs | add hashes
5 print jpgs
```

Extractors are responsible for taking a data source and extracting information from them. In the example shown in listing 2.1, line 1 demonstrates the extraction of files from an NTFS image from block 63 using a block size of 512. Information extracted is passed on to the filters which manipulate the result by adding or removing data. Line 2 illustrates filtering of files whose file names end with the *jpg* extension. The filtered data is passed on to the transformers that generate some output based on the data given. An example of this is shown in line 3 where hashes are generated from each file in the results. Serialisers convert the data into a format that can be usable by the user e.g. a text representation or another tool e.g. digital evidence containers like Advanced Forensic Format (AFF).

2.2.1 Nugget architecture

Building on their previous work, SCAlable Realtime Forensics (SCARF), Stelly and Roussev (2017) designed the Nugget architecture as demonstrated in Figure 2.2. The architecture uses Linux containers to encapsulate Nugget as well as the forensic tools. Communication to integrated tools and sharing of resources between Nugget and the forensic tools is done via RPC. Nugget is designed as a scalable platform that is meant to scale up or down as required. Digital forensic tasks can be distributed across multiple networked containers.

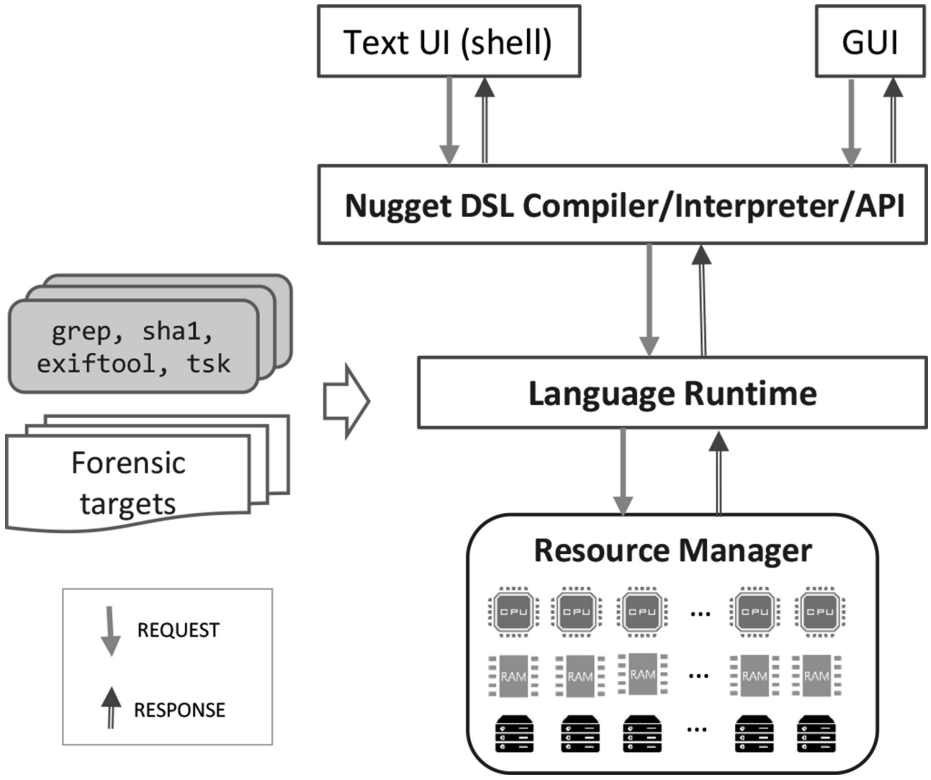


Figure 2.2: Nugget runtime architecture. Source: Stelly and Roussev, 2018

The user interacts with Nugget via a TUI or a GUI. The interface would accept Nugget DSL as input and parse it using the interpreter. The instructions would be executed by the DSL's runtime and work with forensic tools on forensic targets. The output of this operation is displayed to the investigator via the TUI or GUI. The resource manager is responsible for scheduling computations, logging all operations and returning the results of the computation.

2.2.2 Nugget's grammar

Nugget is a Context Free Grammar (CFG) that is defined in Extended Backus-Naur Form (EBNF) using ANother Tool for Language Recognition (ANTLR) (Parr, 2014). ANTLR provides the constructs to perform lexical analysis and syntax analysis. The constructs enable the Nugget runtime to parse and interpret the DSL.

Currently, tools integrated to Nugget dictate what will be part of the grammar. For example, language elements such as *sha1* and *listof-sha1* are based on the *sha1* hashing tool while *sha256* and *listof-sha256* are based on the *sha256* hashing tool. Both these tools provide hashing but they are part of the language. With the incorporation of more tools into Nugget, language elements will grow exponentially.

2.2.3 Extensibility of Nugget

There are numerous tools used in digital forensic investigations. A majority of the tools are not yet integrated with Nugget. This requires that Nugget accommodates extra tools that it does not integrate with by default. Currently, Nugget does this by allowing end users to extend the DSL and write some boiler-plate code. Users require to generate language constructs using ANTLR to extend the DSL thereby extending Nugget to use the desired tools. Since the target audience for Nugget is both technical and non-technical users, the latter may find this approach to extensibility too technical. Due to the loose coupling brought about by separation of forensic tools from Nugget, developers are able to extend Nugget. Some of the ways in which Nugget's extensibility can be accomplished include:

1. Extending the DSL

   Nugget is able to add new functionality by updating the DSL. The implementer of the new functionality would have to generate and compile the code necessary to accomplish the new functionality.

2. Extending the source code

   Nugget's source code can be modified to incorporate new features. The implementer would have to compile the new code into Nugget.

3. Extending capabilities via RPC

   Nugget communicates with forensic tools via RPC. Addition of a new tool means creating, building and running a new RPC target.

The process of extending Nugget would be accomplished by (*a*) identifying the data type to be consumed and produced, (*b*) incorporate the tool's functionality into a container, and (*c*) build Nugget to add the extension into the DSL grammar. For now, the extension of Nugget requires some level of technical or programming knowledge to be able to incorporate new features.

2.2.4 Standardisation of information representation and exchange

The increase in digital forensic tools has also increased the number of ways digital evidence can be represented and exchanged. Digital evidence takes various forms including raw disk images, files, memory dumps, network packets and many more. While the raw data is the actual evidence, meta information such as hashes and case details are equally as important. Attempts have been made to standardise digital forensic evidence e.g. Structured Threat Information Expression (STIX) (OASIS Open, 2018) (Casey, Back & Barnum, 2015) and DFXML (S. Garfinkel, 2012a).

Nugget aims to unify digital forensic computations into one standard. However, the output from such computations is not standardised. Thus, different tools implementing the same computation would result in different outputs. The use of standard information representation schemes would allow Nugget to integrate with standards compliant tools easily. Evaluation and comparison of different forensic tools would also be possible since both tools would give similarly structured output for a given input.

When tools conform to a standard, there shall be little to no need to have complicated extractors for each tool to be integrated to Nugget. Also, there shall be little or no influence on the DSL by the tools. Standardisation simplifies the DSL by removing the complexities of tool-specific data representation from the DSL and using a standard information exchange to handle data representation.

2.3 Digital forensic information representation and exchange

2.3.1 Problems caused by lack of standardisation

Challenges have arisen as a result of the lack of standardisation of information and its exchange. Some of them include:

1. Difficulty in information exchange between investigators

Investigators and researchers working in different organisations may find it difficult to exchange information between themselves since the information may be in different formats. This can be due to a number of factors e.g. using different forensic tools to accomplish the same task. This difficulty can hinder investigators in different legal jurisdictions from knowing that different crimes being investigated are actually executed by the same perpetrator (Casey et al., 2015).

2. Loss of provenance information

   Most existing evidence formats ignore information on how the evidence was obtained (Casey et al., 2015). This provenance information is very beneficial since it allows other investigators or researchers to repeat the same operations using the specified tools in the specified order to arrive at the presented results. Evidence that's retrieved from a repeatable operation has a higher degree of confidence.

3. Difficulty in combining forensic tool outputs

   Other than chaining, a forensic operation may utilise different outputs from different forensic tools by combining the results into one. For example, a forensic operation on a digital storage device may combine information on the different volumes with information on the different files in the volumes. Without a common standard, merging results by different forensic tools into one a common format is laborious and error prone (Casey et al., 2015).

4. Difficulty in chaining different forensic tools

   It is often desirable to use the output of one forensic tool as the input of another. For example, a forensic tool may specialise in extracting files while another specialises in analysing the files' contents. Without a format that is common between them, chaining these tools may not work correctly. Such digital forensic tools lack composability (S. Garfinkel, 2012a). This deficiency would make forensic tools that combine multiple specialised tools to become more complicated so as to support different tool formats.

5. Difficulty in automation

   Automation of forensic processes or operations would involve co-ordinating different forensic tools. Without the promise of composability of the tools in use, the work of automating these processes becomes troublesome (S. Garfinkel, 2012a).

6. Difficulty in tool and algorithm validation

To validate that a forensic tool works as designed, its output must be comparable with the expected output. If the output of the tool and the expected data are in different formats, the effort of verification increases (S. Garfinkel, 2012a; Roussev, 2015). This problem increases exponentially due to the number of different tools available that accomplish similar tasks and produce different formats.

7. Difficulty in scalability

   Scalability of forensic operations is highly dependent on the degree of composability of forensic tools. As noted previously, automation has become notoriously difficult due to the different formats. This difficulty cascades down to scalability, making it difficult to scale up forensic operations.

8. Poor documentation and proprietary formats

   Poorly documented data formats and proprietary formats significantly reduce the methods and techniques used to examine and analyse forensic targets (Common Digital Evidence Storage Format Working Group (Digital Forensics Research Workshop), 2016). As van den Bos and van der Storm note, that proprietary data formats are poorly documented and which leads to reverse engineering.

9. The Proliferation of more data formats

   Without one standard that is common to all, there is little incentive for forensic tool developers to conform to or enforce a standard way of representing information. Thus, they would create a data format that suits their tool's input or output.

2.3.2 Existing digital evidence representation formats

RAW Images

These are files representing an identical copy of the source media. While the image might be the focus of a digital forensic analyst's attention, critical meta information is missing from the raw image e.g. cryptographic hashes. Also, storage optimisations e.g. compression have not been done on the raw image thus requiring equal or more storage capacity than the raw image (Roussev et al., 2016).

DERRIC

DERRIC (van den Bos & van der Storm, 2011) is a domain specific Data Definition Language (DDL) for specifying data structures, in digital forensics. Researchers aimed to

achieve scalability (handle volumes of data) and flexibility (easy to modify and customise) by separation of data analysis and data formatting. DERRIC contains three sections, a header, a sequence and a set of structures.

Attribute-Relation File Format (ARFF)

ARFF is a description of a list of instances sharing a set of attributes (Computing and Mathematical Sciences, University of Waikato, 2018). The description is written in an American Standard Code for Information Interchange (ASCII) text file. The file has a header section and a data section. The header section contains the name of the relation (data) and a list of the attributes (columns) of the data and their types. The data section contains the actual data. This format is mainly used with the Weka machine learning project.

Expert Witness Format (EWF)

It is a proprietary format used by EnCase from OpenText (formerly Guidance Software). The popularity of EnCase has resulted in EWF's widespread use in digital storage forensics. There are open source implementations that have been done, such as libefw, (Metz, 2018) enabling forensic tools, other than EnCase, such as TSK and Forensic Tool Kit (FTK) to work with the format.

Advanced Forensic Format (AFF) and Advanced Forensic Format 4 (AFF4)

AFF and its successor AFF4 are containers that store disk image content and associated metadata (Cohen, Garfinkel & Schatz, 2009).

XML Information Retrieval Appproach to digital Forensics (XIRAF)

XIRAF (Alink, Bhoedjang, Boncz & de Vries, 2006) is a framework that extracts features from a forensic target using forensic tools and stores them in a high-performance eXtensible Markup Language (XML) database system. The database is queried using XQuery. An analyst friendly web interface simplifies the composition of an XQuery and its execution. XIRAF has three components, (*a*) tool repository (*b*) feature extraction, and (*c*) storage subsystem. The tool repository contains feature extraction tools, used by the feature extraction component. Features extracted are stored on and queried from the XML database

Digital Evidence eXchange (DEX)

DEX (Levine & Liberatore, 2009) is an XML description of the transformations made to acquired raw data and the tools used. The tool agnostic format is aimed at documenting the products of a forensic investigation. A crucial part of being agnostic of the tool used is that different tools can be used to work on raw data but the result is documented in a similar fashion. Thus, digital forensic investigators are able to compare and exchange investigation results without being required to use the same tools. Investigation results can also be reproduced using the DEX format and the raw data. In order to generate DEX output, the digital forensic tools used in investigations are wrapped.

Digital Forensics XML (DFXML)

DFXML is a subset of the XML language that is designed to represent and exchange structured forensic information and forensic processing results (S. Garfinkel, 2012a). Despite its name, DFXML is not restricted to XML format only. It can be represented using other formats such as protocol buffers and JavaScript Object Notation (JSON). Some of the goals DFXML seeks to accomplish include:

- Human readable
- Extensible
- Easy to generate and ingest
- Complement existing forensic formats
- Adhere to existing practices and standards

These goals allow new and existing users to use DFXML with minimal prior experience with it. The goals also enable developers to incorporate DFXML into their forensic tools and extend DFXML to match industry practices and standards.

As of version 1.3.0, DFXML represents the following sections:

- Metadata describing the source input

  The *source* element describes the input given to the digital forensic tool.

- Metadata describing the forensic tool used

  The *creator* element describes the forensic tool that did the analysis and created the DFXML document. The *build_environment* element describes the environment in

which the forensic tool was built. The *execution_environment* element describes the environment in which the forensic tool was executed.

- Metadata describing the storage media

  The *diskimageobject* element describes the storage media being analysed.

- Metadata describing the storage contents The *partitionsystemobject* and *volume* elements represent the partitions and the volumes present on the storage media.

- Metadata describing files in a partition

  The *fileobject* element represents the files and folder found on the storage media being analysed.

- Resource usage

  The *rusage* element describes how much time and resources used by the forensic operation.

Cyber Observable eXpression (CybOX)

CybOX is used to represent cyber observables. A cyber observable is a set of properties that describe an entity in a cyber environment e.g. a UNIX file or a Windows Registry Key. CybOX has been incorporated into STIX 2.0.

Structured Threat Information Expression (STIX)

It's used to represent and exchange Cyber Threat Intelligence (CTI) (OASIS Open, 2018). STIX defines four major object types: domain objects, relationship objects, marking definition objects and bundle objects. Domain objects are used to represent unique concepts in CTI. The twelve domain objects that are defined in STIX are Attack Pattern, Campaign, Course of Action, Identity, Indicator, Intrusion Set, Malware, Observed Data, Report, Threat Actor, Tool and Vulnerability. Relationship objects declare relationships between domain objects. Marking definition objects contain the actual data markings representing handling or sharing requirements of the data. Bundle is a collection of objects grouped together.

2.4 Digital forensic corpora

A digital forensic corpus is a collection of items that are the focus of digital forensic operations. Such a collection can contain data from a variety of sources e.g. hard drive images, memory images, cell phone and camera images, network packet capture. A specially crafted digital forensic corpus containing realistic data has a number of benefits, including testing correctness of a forensic tool's results, comparing two or more forensic tools, training and educating digital forensic experts (S. Garfinkel, Farrell, Roussev & Dinolt, 2009). Digital forensic corpora enable researchers to test techniques they have developed and also validate techniques other researchers have developed (S. Garfinkel et al., 2009). Thus, the tests can be conducted and results validated by anyone. S. Garfinkel notes that maintaining an extensive digital forensic corpus has a number of challenges chiefly related to the diversity and size of the data stored.

2.5 Proposed concept overview

The overall conceptual is shown in Figure 2.3.

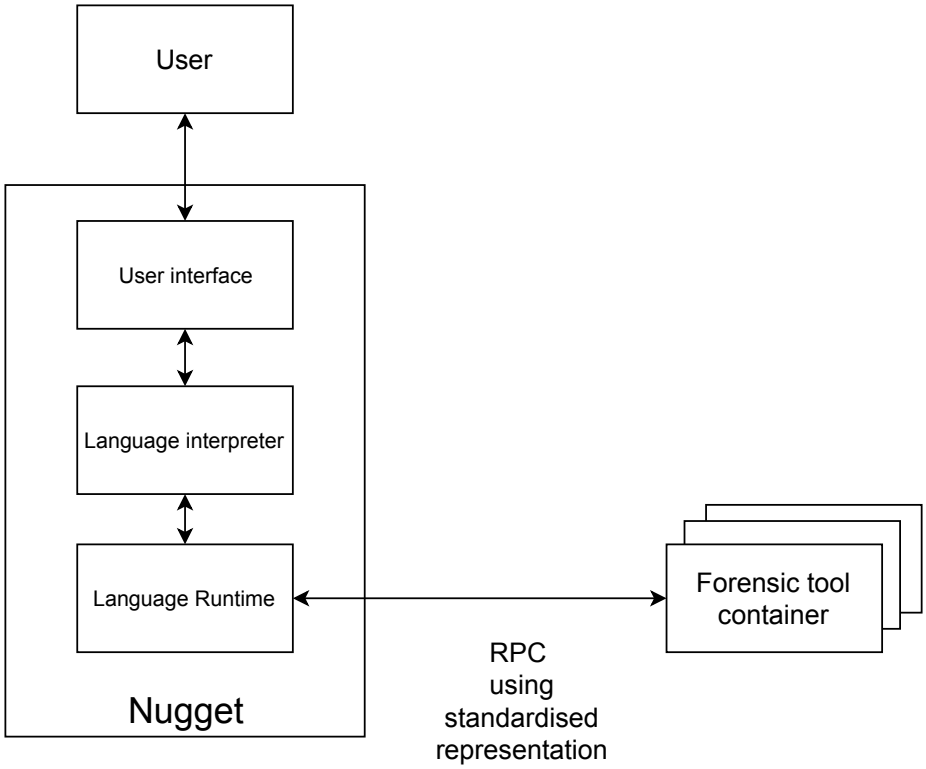


Figure 2.3: Proposed architecture overview.

The implementation would has the mobile forensic tool implemented as a containers. The mobile forensic tools would communicate via RPC with the Nugget runtime. The data representation and exchange format was made to use a standardised forensic information container.

Chapter 3

RESEARCH METHODOLOGY

3.1 Research design

The research methodology describes how the research will be done so as to accomplish the identified research objectives. The research was divided into three major phases: (*a*) Review and evaluate existing standards of digital forensic information containers, (*b*) Review and evaluation of existing mobile forensic tools, (*c*) Design, implementation and testing of integrations with mobile forensic tools.

Each phase was then broken down into specific research tasks. These tasks handled specific areas of the research objective.

3.1.1 Review and evaluate existing standards of digital forensic information containers

This phase of the research shall be an exploratory study, focusing on existing standards of digital forensic information containers.

1. Identify a list of open source or publicly available digital forensic information containers

   This research task was accomplished by conducting a survey of the concerning literature.

2. Identify the container's attributes

3. Determine the best container to use for Nugget's extractors and serialisers

The output of this phase shall inform the researcher what digital forensic container format to incorporate into Nugget, satisfying the second research objective.

3.1.2 Review and evaluation of existing mobile forensic tools

In this phase, the researcher implemented an exploratory study into mobile forensic tools that can be incorporated into Nugget. An exploratory study consists of collecting, analysing

and interpreting observations about known designs, systems or models (Edgar & Manz, 2017). The research tasks done to implement this research objective were:

1. Identify a list of open source or publicly available mobile forensic tools

   An exploratory study was done by doing a survey of the concerning literature (Kothari, 2004). Some of the tools identified were also a result of the exploratory study done in the first phase of the research.

2. Identify the tools' inputs and their resulting outputs

3. Determine how the tool can be incorporated into Nugget

The output of this phase informed the researcher which tools to integrate with Nugget, thus satisfying the first research objective.

3.1.3 Design, implementation and testing of integration with mobile forensic tools

In this phase, the researcher will design, implement and test the integration of mobile forensic tools with Nugget. The tools to be integrated would be the outcome of the first phase. Standardisation of information within Nugget will be informed by the second phase. To implement this phase, the researcher used applied experimentation (Edgar & Manz, 2017) techniques. These techniques are useful when comparing alternative solutions to a problem. Steps to be taken in this phase are:

1. Identify items in digital forensic corpora to be used in testing the integrations. Samples to be used for testing in the experiment will be sourced mainly from digital forensic corpora e.g. Digital Corpora (Digital Corpora, 2017) as well as digital forensic challenges, e.g. Digital Forensics Research Workshop (DFRWS) challenges.

2. Identify a mobile forensic tool to integrate with Nugget

3. Design and implement the integration of the tool with Nugget

4. Test the integration using digital forensic items identified in step 1

5. If step 4 succeeds, repeat steps 2, 3 and 4 using another mobile forensic tool.

The above steps integrate well with the system development methodology chosen, evolutionary prototype. The methodology creates robust prototypes in a structured way and is constantly refined according to the requirements of the final system. The output of this

phase shall be the integration of mobile forensic tools to Nugget, satisfying the third research objective.

3.1.4 Conceptual architecture

The conceptual architecture of the forensic tool wrapper is illustrated in Figure 3.1.

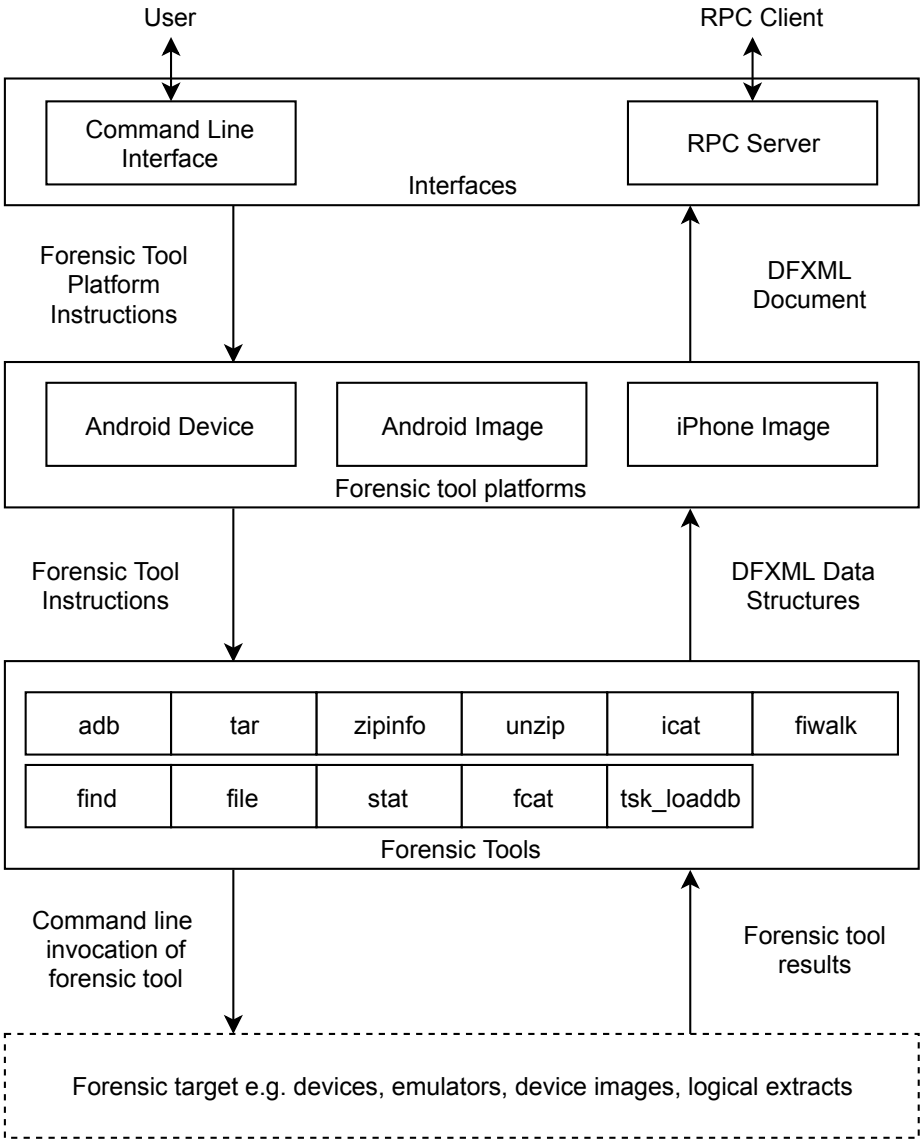


Figure 3.1: Conceptual forensic tool wrapper architecture.

The architecture consists of three primary layers:

- Interfaces

The *Interfaces* layer contains the ways in which the forensic tool platforms can be accessed by either users via the command line interface or the Nugget runtime via the RPC Server.

- Forensic tool platforms

  This layer contains the logic to work with specific mobile device platforms e.g. iPhone and Android. Android Debug Bridge (ADB) is a specialisation of the Android platform that works with physical devices or emulators. The components in this layer accept instructions from the interfaces and return DFXML documents. Forensic computation is delegated to the forensic tool layer.

- Forensic tools

  This layer provides utilities to interact with the actual tools that do the work. The output from components within this layer is appropriate DFXML objects e.g. *FileObject* is the output from file-based operations.

3.1.5 Forensic tool platform data flow

The forensic tool platforms are meant to consume the evidence presented, perform the forensic computation requested and provide the results in DFXML. Even though there are different wrappers for the different mobile device platforms, the phases the wrappers go through to process the data and present results are similar. The phases are summarised in Figure 3.2.

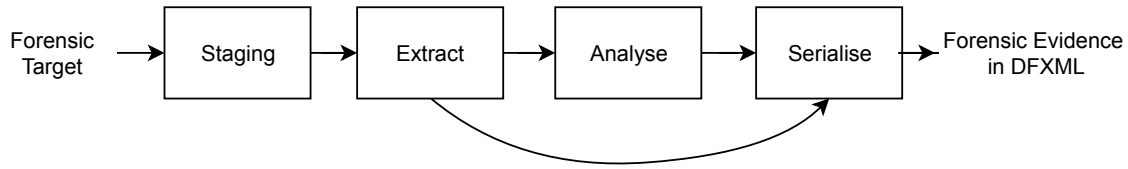


Figure 3.2: Forensic tool platform phases.

Each tool is designed to implement the following phases:

- Staging Phase

  In this phase the forensic tool platform prepares the provided artefact to be used by the forensic tool. The staging process typically means that the provided artefact is placed in a specific working directory, ready to be ingested by the forensic tool.

- Extraction Phase

The forensic tool wrapper executes the forensic tool targeting the staged artefact to extract one or more specific files or pieces of data. In the case of forensic images, the forensic tools used would be SleuthKit, archive files would use ZIP or TAR while Android devices or emulators would use the ADB.

- Analysis Phase (optional)

  This is an optional phase that performs post-extraction processing of the data. If the user's or client's request requires a pre-analysed response, this phase will be responsible for that. An example of this would be counting the number of files in a folder from the data source or evaluating the contents of an on-device data store e.g. the SMS database.

- Output Phase

  The data retrieved from the forensic tool is wrapped in the appropriate DFXML representation and sent to the user.

3.2 Implementation

3.2.1 DFXML Extension

The representation and exchange format chosen for the project was DFXML. Some of the advantages DFXML brings to the table include:

- Extensibility

  It can be extended to represent items that were not part of the initial specification.

- Provenance

  It can store information related to how and where the forensic computation was done, allowing other users to independently verify or duplicate the results.

- Actively used in forensic tools

  The format is used in forensic tools such as *fiwalk* (File Inode Walk) (S. L. Garfinkel, 2009). Therefore integrating such tools would be relatively easy.

- Influenced by an existing and popular forensic tool

  The format has been influenced by the SleuthKit, a popular forensic tool-kit. Due to this, forensic examiners that are familiar with the SleuthKit tool-kit would be familiar

with DFXML.

- Generic / General usage, not just disk forensics

  Unlike other representation standards, DFXML is not specific to one area of forensics e.g. disk forensics. Its extensibility enables the user to use it for more than what it was initially specified.

- Ease of validation of output

  Being a subset of XML, DFXML can be easily validated using XML Schema Definition (XSD) files. The XSD files define the data structures used in DFXML and validate their usage.

- Human readable

  Compared to other standards, DFXML is easier to use and understand since it's not a binary format.

The DFXML format offers ways to describe disk or image-based evidence e.g. from storage devices. As of this writing, the latest version of DFXML (version 1.2.0) is based on disk forensics, with a heavy influence from TSK. DFXML supports extensibility and thus evidence sources that were not handled by DFXML can be easily incorporated. The DFXML specification is written in XSD files (DFXML Working Group, 2017).

Support for mobile forensics evidence types

Mobile device forensics introduces new evidence types to deal with, other than on-disk files, such as SMS and MMS messages and call logs. To extend the specification, the researcher created an XSD file that specified the following elements in the *mobile* XML namespace:

- **SMS and MMS Message**

  This object specifies SMS and MMS messages extracted from the input device or file. Details of this object are highlighted in Table 3.1. The SMS and MMS evidence structure report was influenced by the structure used by the *SMS_Message_Object* schema (The MITRE Corporation, 2016) used by CybOX, the iPhone SMS/MMS message format and the Android SMS/MMS message format.

Table 3.1: SMS and MMS Message representation

Field	Description	Type
Kind	Kind of message i.e. SMS or MMS	string
Sender	The phone number of the message's sender	string
Recipient	The phone number of the message's recipient	string
Body	Contents of the message	string
DateSent	The date and time when the message was sent	date time
DateReceived	The date and time when the message was received	date time
Read	Whether the message was read or not	boolean
Headers	SMS/MMS message headers	binary
Subject	The subject of an MMS message	string
CountryCode	The ISO 3166-1 alpha-2 country code or the Mobile Country Code e.g. KE or 639 for Kenya	string
Type	Type of the message e.g. inbox, sent, outbox	string

A sample representation of an SMS is highlighted in Listing 3.1.

Listing 3.1: SMS message representation

```
<mobile:sms_mms>
  <mobile:kind>sms</mobile:kind>
  <mobile:sender>+17033409661</mobile:sender>
  <mobile:body>How&#39;s the flashmob going</mobile:body>
  <mobile:date_received>2012-07-11T20:07:59+03:00</mobile:date_received>
  <mobile:read>0</mobile:read>
  <mobile:type>inbox</mobile:type>
</mobile:sms_mms>
```

- **Phone call**

  This object represents the call history extracted from the device. Details of this object are highlighted in Table 3.2.

Table 3.2: Phone Call representation

Field	Description	Type
From	The phone number of the initiator of the phone call	string
To	The phone number of the call's recipient	string
DateCalled	The date and time when the call was made	date time
Duration	How long the phone call lasted, in seconds	positive integer
CountryCode	The ISO 3166-1 alpha-2 country code or the Mobile Country Code of the sender or recipient e.g. KE or 639 for Kenya	string
Type	Type of the phone call e.g. incoming, outgoing, blocked	string
Blocked	Whether the phone call was blocked or not	boolean

A sample representation of a phone call is highlighted in Listing 3.2.

Listing 3.2: Phone call representation

```
<mobile:call>
  <mobile:from>5713083236</mobile:from>
  <mobile:date_called>2012−07−06T18:18:50+03:00</mobile:date_called>
  <mobile:duration>244</mobile:duration>
  <mobile:country_code>310</mobile:country_code>
  <mobile:type>incoming</mobile:type>
</mobile:call>
```

- **Location information**

  This object represents an instance of coordinates found in the device. The coordinates may come from the device's cellular information, WiFi information or Global Positioning System (GPS) / Global Navigation Satellite System (GLONASS) chips. Details of this object are highlighted in Table 3.3.

Table 3.3: Location information representation

Field	Description	Type
Longitude	The longitude of the location	floating point number
Latitude	The latitude of the location	floating point number
Source	The source of the location e.g. GPS, cell, wifi	string
Confidence	The confidence level of the location information	positive integer
Timestamp	The date and time when the location information was recorded	date time
CellMCC	The cellular Mobile Country Code used (if source is *cell*) e.g. 639 for Kenya	string
CellMNC	The cellular Mobile Network Code used (if source is *cell*) e.g. 02 for Safaricom	string
CellLAC	The cellular Location Area Code used (if source is *cell*)	string
CellCI	The cellular Cell Identifier used (if source is *cell*)	string
WifiMAC	The MAC of the WiFi network adapter used (if source is *wifi*)	string

The representation combines the available location sources available on a mobile device and separates between the different sources using the *Source* attribute. Listing 3.3 shows a representation of a location from a WiFi source while listing 3.4 shows a representation of a location from a device's cellular information.

Listing 3.3: Location information from a WiFi source

```
<mobile:location>
 <mobile:long>−77.08572131</mobile:long>
 <mobile:lat>38.8291192</mobile:lat>
 <mobile:source></mobile:source>
 <mobile:confidence>50</mobile:confidence>
 <mobile:timestamp>2012−07−10T19:46:29+03:00</mobile:timestamp>
 <mobile:wifi_mac>0:a0:f8:bc:25:a7</mobile:wifi_mac>
</mobile:location>
```

Listing 3.4: Location information from cellular service

```
<mobile:location>
 <mobile:long>−77.11546951</mobile:long>
 <mobile:lat>38.87767624</mobile:lat>
 <mobile:source></mobile:source>
 <mobile:confidence>70</mobile:confidence>
 <mobile:timestamp>2012−06−13T22:01:21+03:00</mobile:timestamp>
 <mobile:cell_mcc>310</mobile:cell_mcc>
 <mobile:cell_mnc>410</mobile:cell_mnc>
 <mobile:cell_lac>7985</mobile:cell_lac>
 <mobile:cell_ci>160043533</mobile:cell_ci>
</mobile:location>
```

- **Bookmarks and web browser history**

  This was another new concept introduced to DFXML, borrowing from the *URL_History_Object* defined in CybOX (The MITRE Corporation, 2016). This object represents the web browsing history of the user and also any bookmarks found. Details of this object are highlighted in Table 3.4.

Table 3.4: Bookmark and web browser history representation

Field	Description	Type
WebBrowser	Name of the web browser	string
URL	URL of the bookmark or visited location	string
ReferrerURL	The referring page's URL	string
PageTitle	Title of the page or bookmark	string
VisitDate	When the bookmark was added or when the URL was visited	string

- **Device Information**

  Metadata about the device being investigated was supported by DFXML using the *dc* namespace (DFXML Working Group, 2017; Dublin Core Metadata Initiative, 2018). The namespace offers a definition of meta information such as the creator of the DFXML document, date of creation, description and format of the forensic target. The Device Information object extends this information by adding information about the mobile forensics target. Details of this object are highlighted in Table 3.5.

Table 3.5: Device information representation

Field	Description	Type
Name	Name of the device	string
Model	Model of the device	string
Release	Release number of the device	string
Manufacturer	Name of the device's manufacturer	string
SerialNumber	Serial number of the device	string
IMEI	IMEI of the device	string

Listing 3.5: Sample representation of device information

```
<mobile:device_info>
   <mobile:name>iPhone OS</mobile:name>
   <mobile:release>4.2.1</mobile:release>
</mobile:device_info>
```

Support for multiple commands

Some of the provenance information in DFXML is stored in the *ExecutionEnvironment* section. The section has a *CommandLine* property that shows how the tool was invoked. An assumption made was that only one command should be run. However, some extraction procedures require use of more than one forensic tool. Thus, the definition of tool invocation had to be expanded to allow representation of multiple invocations and preserving their order. To fulfil this requirement, the *mobile* namespace defines a new element, *CommandLine* that has the command being invoked and also its position in a sequence of commands. Details of this object is highlighted in Table 3.6.

Table 3.6: Process representation

Field	Description	Type
Command	The command line invocation of the forensic tool	string
Sequence	The position of the command in a sequence of commands	positive integer

3.2.2 Nugget Extension

Incorporation of DFXML

Nugget was already capable of handling disk, image and network information. However, this information was not standardised. This allowed the tools being integrated to influence the internal data structures used in Nugget. DFXML was used to standardise this information. In addition to standardising existing data structures, DFXML enables Nugget to work with the mobile extensions of DFXML.

The updated DFXML schema was incorporated into Nugget's runtime. This enabled Nugget to consume and manipulate DFXML documents. Parts of the Nugget runtime that were changed to accommodate DFXML were the extractors and the serialisers. As the external data ingestor of the runtime, extractors were rewritten to enable consumption of DFXML forensic data from forensic tool platforms. The serialisers are the output system of the runtime so they were changed to be able to output DFXML.

Provenance information

Addition of extra information to enable different investigators to reproduce the results of a forensic computation.

Case identification information

This information allows forensic investigators to have a way to associate a forensic computation expressed in Nugget with an investigation. It is set at the top of a Nugget script file by specifying the case number, the investigation date and the investigator.

Multiple extractors, same type

Existing a file extractor. Devices require different extractor but present similar information description

The visual presentation of information

Introduction of tabulated information enables users to consume more information about the results. For example, to display files found from a forensic computation, only the filename was shown.

```
[
  /Videos/Cat.m4v
]
```

With the extra information, the forensic investigator can be able to view the file's meta information such as the modification, access and create times.

```
  Filename       Created       Modified              Accessed     Size  Type
---------------- -------- ------------------- ------------------- ------ -----
 /Videos/Cat.m4v  <nil>   2019-03-27T22:42:44 2019-02-17T17:56:49  7688    d
```

3.2.3 RPC Server Interface

It provides a simple interface for the Nugget runtime, to extract information from forensic targets. It mediates communication between Nugget scripts and forensic tools. The RPC interface provides its output in DFXML format. To keep the interface simple, only one method was exposed via RPC, *Dfxml*. The *Dfxml* method required the client to supply specific arguments, as defined in Table 3.7.

Table 3.7: Arguments passed onto the *Dfxml* RPC method

Argument	Description	Required?
Target	The forensic target to be worked on	Yes
Platform	The platform that corresponds to the forensic target. The supported options are *android* to represent Android forensic images, *iphone* to represent iPhone forensic images and *adb* to represent Android devices or emulators.	Yes
Command	The forensic computation to be done	Yes
CommandArgs	Arguments to be passed on to the forensic computation	No

The result of the forensic computation was wrapped in a DFXML object and sent to the RPC client.

3.2.4 Command line interface

While the primary client of the forensic tool platforms is the Nugget runtime, a user can also use it as a stand-alone program. The command line interface provides the users with an interface similar to the RPC server. To provide the user with a friendly way to consume the data output, the command line interface provides users with a tabular format.

3.2.5 Android Image Platform

The Android Image platform focused on extracting information from images or logical extracts from devices running the Android Operating System. Images of devices could be raw disk images or specialised forensic containers, e.g. the Expert Witness Format. Logical extracts could be stored as file archives e.g. ZIP or TAR. The Android Open Source Project (AOSP) maintains and provides the source code to the open source parts of Android. Therefore, finding documentation and implementation of features such as SMS and phone calls was reasonably straight-forward.

SMS and MMS extraction

Until Android 6, SMS and MMS information was stored in an SQLite database located at */data/data/com.android.providers.telephony/databases/mmssms.db* The database contains a table named *sms* that stores SMS and MMS information using the columns described in Table 3.8.

Table 3.8: Columns in the *sms* table of the *mmssms.db* database

Field	Description
_id	Unique identifier of the message
thread_id	Thread ID of the message
address	Address of the other party
person	Person ID of the sender
date	The date the message was received
date_sent	The date the message was sent. New in Android 2.1
protocol	Protocol identifier code e.g. MMS, SMS
read	Whether the message has been read
status	TP-status value for the message e.g. none, complete, pending, failed
type	Type of the message e.g. inbox, draft, sent
reply_path_present	Whether the TP-Reply-Path bit was set on this message
subject	Subject of the message if present
body	Body of the message
service_center	The service centre through which to send the message, if present
locked	If the message has been locked
error_code	Error code associated with sending or receiving the message
seen	Indicates whether the user has seen the message.
sub_id	Subscription ID to which the message belongs to. New in Android 6.0
creator	The name of the application that sends the message. New in Android 6.0

With the introduction of multiple users on one device, Android 5 and higher introduced separation of application and data based per user. Thus, each user has their own *mmssms.db* located at */data/user/0/com.android.providers.telephony/databases/mmssms.db*, where 0 is the user identifier. Starting from Android 5, a new messaging application was introduced to Android, called Messages. Its database is stored in */data/data/com.android.messaging/databases/bugle_db*. Similar to the previous Android version, the database was also separated per user, storing user-specific content in */data/user/0/com.android.messaging/databases/bugle_db*. The database had the columns described in Table 3.9:

Table 3.9: Columns in the *messages* table of the *bugle_db* database

Field	Description
_id	Unique identifier of the message
conversation_id	Foreign key relation to the related conversation
sender_id	Foreign key relation to the message's sender
sent_timestamp	When the message was sent
received_timestamp	When the message was received
message_protocol	Type of message e.g. SMS, MMS
message_status	TP-Status of the message e.g. none, complete, pending, failed
seen	Whether the user has seen the message or not
read	Whether the user has read the message or not
sms_message_uri	URI of the message
sms_priority	Message priority
sms_message_size	Size of the message in bytes
mms_subject	Subject of an MMS
mms_transaction_id	Transaction identifier of an MMS message
mms_content_location	Location of MMS content
mms_expiry	When MMS message will expire
raw_status	TP-Status of the message e.g. none,complete,pending,failed

Extraction of data from the *mmssms.db* and *bugle_db* databases was done in two steps. First the database file was extracted from the device or the image onto the workspace. Second, the extracted database file was opened in read-only mode to prevent writes to it and then SQL queries were issued to it. The resulting information was wrapped in an SMSMMSMessage DFXML object.

Call history extraction

Call history is stored in an SQLite database located at */data/data/com.android.providers.-contacts/databases/contacts2.db*. After the introduction of multiple users on a device, the location of the database file changed to */data/user/0/com.android.providers.contacts/databases/-contacts2.db* where 0 is the user identifier. The database contains a table named *calls* that stores call history information using the columns described in Table 3.10.

Table 3.10: Columns in the *calls* table of the *contacts2.db* database

Field	Description
_id	Unique identifier of the row
number	The number of the caller or callee
date	When the call was placed
duration	How long the call lasted
type	Type of call e.g. incoming, outgoing,
name	Name of the caller, if present
numbertype	Type of the number
numberlabel	Label of the number
countryiso	ISO if the country within which the call was made
voicemail_uri	URI of voicemail, if applicable
is_read	In case of a missed call, if it's read or not
geocoded_location	Location within which the call was made
matched_number	The matched number in the contacts database
normalized_number	The normalised number
photo_id	Link to a photo of the other party
formatted_number	The formatted number

Since Android 7.1, the location of call history has been moved from *contacts2.db* to *call-history.db*, both located in the same folder. The new database file contained the columns defined in Table 3.10 with some additional columns defined in Table 3.11

Table 3.11: Extra columns defined in the *calls* table of the *callhistory.db* database

Field	Description
via_number	If the call was redirected it's recorded here
subscription_id	Link to the call's subscription
phone_account_address	The address as stored on the phone
phone_account_hidden	Whether the account is hidden
last_modified	When the record was last modified
dirty	Whether the record has changed
deleted	Whether the record is deleted or not

File location and extraction

Location and extraction of files from Android forensic images was done using *fiwalk* (S. L. Garfinkel, 2009), *icat* (Venema, 2018) and *fcat* (Carrier, 2018b) from the SleuthKit tool collection. *fiwalk* was used to extract all the information about disk volumes, folders and files available on the forensic image. The output of *fiwalk* was formatted as a DFXML document. Using the results from *fiwalk*'s image analysis, the Android wrapper was able to

locate specific files of interest, e.g. *contacts2.db* for SMS or MMS messages. The located file contained its full path and name, inode number, integrity hash values and other attributes as highlighted in Table 3.12

Table 3.12: Attributes of a DFXML *FileObject*

Attribute	Description
Filename	Name of the file
Type	Type of the file e.g. folder, regular file, character file

The *FileObject* information and enabled the extraction of the file's content from the forensic image using either *icat* or *fcat*. *icat* extracted a file's content using the file's inode number while *fcat* extracted a file's content using the file's full path. The extracted files are then checked for their contents' integrity by using the hashing algorithm and value provided by the *FileObject* instance.

Device information
Information about a device is stored in a properties text file, located at */system/build.prop*. The file contains details such as the device's name, its manufacturer, its CPU architecture, its Android SDK release and much more.

3.2.6 Android Device Platform

The ADB is a command line tool that allows users to communicate with a connected device or emulator (Google, 2018). Devices connect to the forensic machine using USB, WiFi or Bluetooth. ADB is composed of three components:

- **client** that runs on the forensic machine and sends commands to the server
- **daemon** that runs on the Android device or emulator
- **server** that runs on the forensic machine and manages communication between the client and the daemon

With ADB, an investigator has access to files on the device, can manipulate applications and services on the device and much more. If the device is rooted, the entire device's contents are available to the investigator. Using ADB may not be forensically sound since it does not guarantee that it would not result in the modification of data on the device. Examples of this include:

- If the device is powered off, it has to be powered back on to be able to use the ADB server on the device
- To make the device accessible via ADB, the debugging setting needs to be enabled. Enabling the debugging setting on a device requires unlocking the device (if it was previously locked), enabling developer options (if it wasn't enabled) and finally enabling the debug setting.
- To allow access to system files, ADB requires to be run in root mode. Running in root mode gives the user root privileges only if the device has been rooted. Devices that have not been rooted require to be rooted first in order to gain root access. Rooting a device modifies data on the device.
- To run very long shell commands, shell script files require to be written onto the device's file system and then executed.

Even with the challenges highlighted above, ADB can be used carefully to ensure minimal or no modifications are done on the device. McKemmish proposes that forensically sound evidence should:

- not be affected by the digital forensic process
- document all errors identified
- be capable of independent verification
- be taken by a sufficiently experienced digital forensic analyst

By taking the above into consideration, evidence recovered using ADB can be forensically sound. The provenance information provided by the Android Device Platform's DFXML output can also help the investigator prove that the evidence is forensically sound.

The ADB provides a shell utility that allows users to execute arbitrary commands on the device via a command line interface.

File location and extraction

Locating files on the device's file system was done using *find*. *find* traverses the specified folder tree looking for files that match the specified search criteria. Once found, a file's meta information is extracted using *stat*. Execution of this technique differs between pre-Android 7 devices and post-Android 7 devices because of the differing versions of *find*. For devices running Android 7 and above, the *find* command was able to run the *stat* command for each file located using the *-exec* option. Devices that run less than Android 7 lack the

-exec command line argument and thus required the two commands to be run as separate invocations, with the result of *find* being the input of *stat*. The output of *find* could be larger than the allowed maximum length of commands. A shell script pushed to the device and executed on the device was used to overcome this.

Extracting files from the device's file system was done using *adb pull* (Google, 2018).

SMS and MMS extraction
This process is similar to the Android platform's SMS and MMS extraction process. However, on mobile devices without root access, this extraction would not work because of permission restriction.

Call history extraction
This process is similar to the Android platform's SMS and MMS extraction process. However, on mobile devices without root access, this extraction would not work because of permission restriction.

Device information
The techniques used to extract device information in the Android platform applied to the Android Device Platform as well. Additionally, ADB provides the *getprop* utility that simplifies accessing device information.

3.2.7 iPhone Image Platform

The iPhone Image platform dealt only with images and logical extracts from iPhone devices. The iPhone Operating System, iOS, is closed source. Thus, much of the knowledge about the internals of iOS have been acquired through observation and reverse engineering. Public projects such as *iphoneanalyzer* (Crawford & matproud, 2012), *iPhone Tracker* (Warden, 2011) and *The iPhone Wiki* (The iPhone Wiki, 2015a) contributed to the techniques used in the iPhone platform.

File location and extraction
It worked in similar fashion to the Android Platform's file location and extraction since they both deal with forensic images or file archives.

SMS and MMS extraction
SMS and MMS messages are managed by the Messages system application. The application's database is located at */private/var/mobile/Library/SMS/sms.db* (Tamma et al., 2018;

The iPhone Wiki, 2015b). The main table in the *sms.db* database is the *messages* table. Some of the known columns defined in the *sms.db* database are highlighted in Table 3.14.

Table 3.13: Some of the known columns defined *message* table of the *sms.db* database

Field	Description
ROWID	A unique identifier of the row
address	Name or phone number of the other party
date	Date message was sent / received
text	Contents of the message
flags	Status of the message e.g. sent, inbox, outbox, failed to send
svc_center	Service centre
group_id	Unique identifier of the message's conversation group
subject	Subject of an MMS message
country	ISO 3166-1 alpha-2 country code e.g. KE for Kenya
read	Whether the user has read the message or not

Call history extraction

Table 3.14: Columns defined *message* table of the *sms.db* database

Field	Description
ROWID	A unique identifier of the row
address	Name or phone number of the other party
date	Date message was sent / received
duration	Duration of the call in seconds
flags	Status of the call e.g. blocked, incoming, outgoing
id	Id of the contact being called, -1 for incoming calls
country_code	Mobile country code of the country the phone was in when the call was placed
network_code	The Mobile network code of the network the phone was in when the call was placed

Location history extraction

Location history was retrieved from the *consolidated.db* database. The database contains locations recorded by the device from cellular sources and WiFi sources.

Device information

Device information was retrieved from a property list file named *SystemVersion.plist*. The file is located at */private/var/System/Library/CoreServices/*.

3.2.8 Wrapping forensic tools

Table 3.15 summarises the tools used to execute forensic commands.

Table 3.15: Summary of the tools used to execute forensic computations

Name	Description
adb	The Android Debug Bridge, used to communicate with Android devices or emulators.
icat	Part of the SleuthKit tool collection that outputs the contents of a file from a forensic image-based on its inode number.
fcat	Part of the SleuthKit tool collection that outputs the contents of a file from a forensic image-based on its name.
file	A tool to determine the Multipurpose Internet Mail Extension (MIME) type of a file.
fiwalk	Part of the SleuthKit tool collection that finds and extracts file information from a forensic image into DFXML.
stat	A tool to display metadata about a file or a file system.
tar	A tool to examine and extract files from TAR archive files
tsk_loaddb	Part of the SleuthKit tool collection that finds and extracts file information from a forensic image into an SQLite database.
unzip	A tool to extract files from ZIP archive files.
zipinfo	A tool to examine ZIP archive files.

3.3 Testing

Testing the system was done to ensure the accomplishment of two main goals; the integration with the Nugget runtime and the evaluation of mobile forensic evidence. To test the accomplishment of these two goals, the testing was split into two phases. The first phase of testing focused on testing the system's ability to ingest and evaluate mobile forensic evidence. In this phase, the system was used via its two interfaces, the Command Line Interface and the RPC interface. Input was given to the system and the output evaluated against previously established expected output. The system was tested in isolation, without connecting to the Nugget runtime. The second phase of testing focused on testing the integration of the system with Nugget's runtime. Input during this phase of testing was Nugget scripts that declare forensic computations on the specified mobile forensic evidence. The Nugget runtime interpreted the scripts and sent the appropriate commands to the system. The output of the results was compared against the previously established expected output. A sample Nugget script used in testing is shown in Listing 3.6. The scripts declares that message, phone call and location information be extracted from the provided forensic targets and the

results presented in tabular form.

Listing 3.6: Sample listing of Nugget script used for testing

```
1  case_number "001"
2  investigator "Chomba Ng'ang'a"
3  investigation_date "08/04/2018"
4  workspace "/nugget-tools/tmp"
5
6  msgs="iphone:/it_forensics/tracy-phone-2012-07-15-final.tar" | extract as message
7  print msgs.SMSMMSMessage as table
8
9  calls="iphone:/it_forensics/tracy-phone-2012-07-15-final.E01" | extract as call
10 print calls.PhoneCall as table
11
12 locations="iphone:/it_forensics/tracy-phone-2012-07-15-final.tar" | extract as location
13 print locations.Location as table
```

3.3.1 Test samples

Test samples were obtained to evaluate the system. Each platform had its own specific set of test samples. Table 4.3 highlights the samples.

Table 3.16: Summary of the test samples used to evaluate the system

#	ID	Platform	Description
1	iPhone-1	iPhone Image	TAR file (Tracy's phone)
2	iPhone-2	iPhone Image	EO1 file (Tracy's phone)
3	iPhone-3	iPhone Image	L01 file (Tracy's phone)
4	Android-1	Android Image	TAR file (Carry's tablet)
5	Android-2	Android Image	E01 file (Carry's tablet)
6	Android-3	Android Image	BIN file (Carry's phone)
7	Android-4	Android Image	ZIP file (Carry's phone, FTK logical)
8	Android-5	Android Image	NANDDUMP file (Nexus One)
9	Android-6	Android Image	<no extension> binary file (Nexus S1)
10	Android-7	Android Image	BIN file (Samsung Galaxy S6 Edge)
11	ADB-1	Android Device	Blackberry Priv
12	ADB-2	Android Device	Sony Xperia SP
13	ADB-3	Android Device	Google Nexus 7 2013
14	ADB-4	Android Device	Android 9 Emulator
15	ADB-5	Android Device	Android 8 Emulator
16	ADB-6	Android Device	Android 7 Emulator
17	ADB-7	Android Device	Android 6 Emulator
18	ADB-8	Android Device	Android 5.0 Emulator
19	ADB-9	Android Device	Android 4.3 Emulator
20	ADB-10	Android Device	Android 2 Emulator

iPhone Image Platform samples

The iPhone platform was tested using multiple images of the same iPhone obtained from a simulated scenario about an attack in a city (Digital Corpora, 2018). The iPhone was used by a fictitious character in the scenario called Tracy. The images iPhone-2 and iPhone-3 were created using EnCase (for logical extraction) while iPhone-1's method of extraction was unspecified. The EnCase produced files were in *L01* and *ZIP* formats while the unspecified extraction tools were in *TAR* and *E01* formats.

Android Image Platform samples

The platform was tested using various images from different sources. The first image source was the attack in a city scenario (Digital Corpora, 2018). It provided a phone and tablet owned by Carry, a fictitious character in the scenario. Physical extraction by UFED version 1.2.0.0 was used to extract data from Carry's phone into BIN files while logical extraction was done using FTK Imager into a ZIP file. Extraction from the tablet was done using EnCase into E01 and TAR file formats. The second image source was from the Digital Corpora's mobile repository (Digital Corpora, 2015). This source provided two device images, Nexus One and Nexus S1. The method of extraction was not specified. The third image source was from the DFRWS annual forensic challenges. The 2018 challenge provided a physical image of a Samsung Galaxy Edge S6 (DFRWS, 2018). The method of extraction was not specified.

Android Device Platform samples

Samples to test this platform were physical devices and emulated devices. The physical devices were Blackberry Priv phone, Sony Xperia SP phone and Google Nexus 7 2013 tablet. Both the Sony and the Nexus were permanently rooted by installing LineageOS. The emulators were made using the Android SDK. Emulators were running Android 9, Android 8, Android 7, Android 6, Android 5, Android 4 and Android 2. The emulators were prepared for testing by making simulated messages, calls and location updates on the device.

3.3.2 Testing procedure

Before testing the system with the selected samples, expected results from the samples were obtained. The expected results provided a baseline against which the system's test results could be compared with and determined to be correct or not. A baseline expectation for

each of the forensic targets available was established, where applicable. The baseline for the Android Image platform was established by using another forensic tool, Autopsy. The baseline for the Android Device platform was established by pre-seeding the data in the emulators and devices. This was done by using the devices e.g. sending and receiving SMS, making and receiving phone calls.

After establishing the expected results, each forensic target was analysed using the system's command line interface and the RPC server interface. The output from the command line interface, in DFXML format, was validated against the modified DFXML's XSD specification as defined in Appendix A.1. To test the system's integration with Nugget, Nugget script files were written and executed. The output of the script files was validated against what was expected by the baseline previously set.

Chapter 4

RESULTS AND DISCUSSIONS

4.1 Android Image Platform

The Android Image platform used seven sample files for testing. The values listed in the *Expected* columns were obtained from using another forensic tool, Autopsy. Autopsy has an Android ingestion module that extracts information that is specific to Android-based device images.

Table 4.1: Summary of the Android platform test results

ID	Messages		Calls		Location		Device Information	
	Found	Expected	Found	Expected	Found	Expected	Found	Expected
Android-1	0	0	0	0	0	0	0	1
Android-2	0	0	0	0	0	0	1	1
Android-3	19	19	8	8	0	0	1	1
Android-4	0	0	0	0	0	0	0	0
Android-5	6	6	5	5	0	0	1	1
Android-6	0	0	0	0	0	0	0	1
Android-7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Sample Android-1 was a TAR archive. The archive had an error listing and extracting its contents. Sample Android 2 was an E01 container of the same device Android-1 was extracted from. The device did not have any calls or messages and its device information was able to be extracted successfully. Sample Android-3 was a BIN file containing the device's storage content. Extraction of messages, calls and device information was successful. Sample Android-4 was a logical extract of the same device Android-3 was extracted from. Being a logical extract, no system files were present thus no results expected. Sample Android-5 was a NANDDUMP file containing the device's storage content. Extracting calls and messages from the file succeeded while extracting the device's information did not succeed.

4.2 Android Device Platform

The Android Device platform used ten samples for testing. The values listed in the *Expected* columns were obtained by setting up the data on the devices and emulators by making and receiving calls and text messages.

Table 4.2: Summary of the Android Device Platform test results

ID	Messages		Calls		Location		Device Information	
	Found	Expected	Found	Expected	Found	Expected	Found	Expected
ADB-1	0	30	0	30	0	0	1	1
ADB-2	0	17	8	8	0	0	1	1
ADB-3	0	0	0	0	0	0	1	1
ADB-4	0	1	0	1	0	0	1	1
ADB-5	2	2	3	3	0	0	1	1
ADB-6	6	6	9	9	0	0	1	1
ADB-7	11	11	4	4	0	0	1	1
ADB-8	1	1	1	1	0	0	1	1
ADB-9	7	7	5	5	0	0	1	1
ADB-10	2	2	2	2	0	0	1	1

Retrieving device information from all samples was successful. Sample ADB-1 was the Blackberry Priv. It was not rooted thus access to protected files via ADB was not granted. This resulted in the failure to fetch any messages, calls or location information. Sample ADB-2 was the Sony Xperia SP. It had been permanently rooted by installing LineageOS (The LineageOS Project, 2019), which is based on the Android Operating System. Thus ADB was able to get root access to the device and pull the required information. However, the default messaging application on the device had been changed from the default one to Signal (Open Whisper Systems, 2019) thus SMS and MMS messages were not available to the system. Sample ADB-3 was a tablet that had been permanently rooted by installing LineageOS. The tablet did not have any cellular capabilities. This meant that it could not have been able to make/receive calls or send/receive SMS or MMS messages. Sample ADB-4 was an emulator running Android 9. An error that occurred during the testing prevented any analysis from being conducted. Samples ADB-5, ADB-6, ADB-7, ADB-8, ADB-9 and ADB-10 were emulators running various versions of the Android operating system. Test results from these samples all matched with their respective expected results.

4.3 iPhone Image Platform

The iPhone Image platform used three samples for testing. The values listed in the *Expected* columns are listed as not applicable because information about the contents of the messages or calls or location was not known before the testing started.

Table 4.3: Summary of the iPhone platform test results

ID	Messages		Calls		Location		Device Information	
	Found	Expected	Found	Expected	Found	Expected	Found	Expected
iPhone-1	25	N/A	5	N/A	2381	N/A	1	N/A
iPhone-2	25	N/A	5	N/A	2381	N/A	1	N/A
iPhone-3	0	N/A	0	N/A	0	N/A	0	N/A

The data obtained for the platform was from the same device. Samples iPhone-1 and iPhone-2 got identical results because they were both physical extracts of the same phone, though in different formats. Sample iPhone-3 did not get any results since it was a logical extract from the phone which did not contain the required system information.

4.3.1 SMS and MMS extraction

A section of a sample DFXML extract of SMS and MMS information is displayed in Listing 4.1. The complete output of the sample is available in appendix at Listing B.2.

Listing 4.1: SMS and MMS DFXML extracttion as DFXML

```
<mobile:sms_mms>
  <mobile:kind>sms</mobile:kind>
  <mobile:sender>+15713083236</mobile:sender>
  <mobile:body>What are you up to this weekend? </mobile:body>
  <mobile:date_received>2012−06−13T00:25:04+03:00</mobile:date_received>
  <mobile:read>1</mobile:read>
  <mobile:country_code>us</mobile:country_code>
  <mobile:type>inbox</mobile:type>
</mobile:sms_mms>
```

The above output shows the contents of one of the messages found in the forensic target that was analysed. In this case, the mobile device received an SMS message on 13^{th} June 2012 from a mobile subscriber with the phone number +15713083236 located in the United States of America. The user of the mobile device has already read the message.

4.3.2 Phone call extraction

A sample DFXML extract of phone call information is displayed in Listing 4.2. The complete output of the sample is available in appendix at Listing B.3.

Listing 4.2: Phone call extraction as DFXML

```
<mobile:call>
  <mobile:from>6508870260</mobile:from>
  <mobile:date_called>2012−06−12T23:04:50+03:00</mobile:date_called>
  <mobile:duration>20</mobile:duration>
  <mobile:country_code>310</mobile:country_code>
  <mobile:type>incoming</mobile:type>
</mobile:call>
```

The extracted information shows that a phone cal was received by the mobile phone user from a mobile subscriber with the number 6508870260 from a fictional country with the code 310. The phone call was made on 12^{th} June 2012 and lasted for 20 seconds.

4.3.3 Device information extraction

A sample DFXML extract of device information is displayed in Listing 4.3. The complete output of the sample is available in appendix at Listing B.4.

Listing 4.3: Device information extraction as DFXML

```
<mobile:device_info>
  <mobile:name>iPhone OS</mobile:name>
  <mobile:release>4.2.1</mobile:release>
</mobile:device_info>
```

The above output shows that the mobile device under analysis was an iPhone, whose operating system was iPhoneOS version 4.2.1.

4.3.4 Location extraction

A sample DFXML extract of location information is displayed in Listing 4.4. The complete output of the sample is available in appendix at Listing B.5.

Listing 4.4: Location information extraction as DFXML

```
<mobile:location>
  <mobile:long>−77.11546951</mobile:long>
  <mobile:lat>38.87767624</mobile:lat>
  <mobile:source></mobile:source>
  <mobile:confidence>70</mobile:confidence>
  <mobile:timestamp>2012−06−13T22:01:21+03:00</mobile:timestamp>
  <mobile:cell_mcc>310</mobile:cell_mcc>
```

```
  <mobile:cell_mnc>410</mobile:cell_mnc>
  <mobile:cell_lac>7985</mobile:cell_lac>
  <mobile:cell_ci>160043533</mobile:cell_ci>
</mobile:location>
<mobile:location>
  <mobile:long>-77.08571225</mobile:long>
  <mobile:lat>38.82908231</mobile:lat>
  <mobile:source></mobile:source>
  <mobile:confidence>50</mobile:confidence>
  <mobile:timestamp>2012-07-10T19:46:29+03:00</mobile:timestamp>
  <mobile:wifi_mac>0:a0:f8:bc:25:a5</mobile:wifi_mac>
</mobile:location>
```

The above output shows location information extraction from cellular information and from WiFi. The first location was retrieved from the device's cellular information. The information provides the coordinates in longitude and latitude and the cell in which the mobile subscriber was located. The cell information includes the mobile country code, the mobile network provider and the specific cell. The second location was retrieved from WiFi information. It provides the coordinates in longitude and latitude and the WiFi MAC address used. Both location information sections contain data on when the information was captured and also the confidence level of the information's accuracy.

Chapter 5

CONCLUSION AND RECOMMENDATIONS

5.1 Summary of the study

The overall objective of the study was to incorporate mobile forensics into Nugget. This objective was accomplished by designing, implementing and testing a system that integrated with Nugget and implemented forensic computations on mobile forensic evidence samples. The end product of the study was a system that integrated with Nugget and was able to process mobile forensic evidence targets.

The overall objective had specific research objectives that were to be achieved. The first objective was to review and evaluate existing standards of digital forensic information containers This objective was accomplished by first identifying a list of open source or publicly available digital forensic information containers. Out of the list, one container, DFXML was chosen to be used in the study.

The second objective was to review and evaluate existing mobile forensic tools. To accomplish this objective, a list of open source or publicly available mobile forensic tools was identified. For each tool, its inputs and outputs were evaluated and based on the evaluation, the method of incorporating the tool with Nugget was determined.

The third objective was to design and implement integrations of forensic tools with Nugget. This objective depended on the output of the first and second objectives. The forensic information container selected in the first objective was used to structure forensic information. The tools selected in the second objective were used to analyse forensic evidence targets. The implementation of the integration to Nugget was done. The result of this objective was the development of the system that integrated the forensic tools to Nugget via the chosen forensic information container.

The fourth objective was to test the implemented integrations using publicly available digital forensic corpora. Based on the output of the third objective, the objective was accomplished by using the device images and device extracts obtained from the digital forensic

corpora. In addition to the digital forensic corpora, testing was done on physical devices and emulated devices.

5.2 Limitations of the system

- **Access to devices or device images**

  To test the implemented system, test samples of mobile device forensic targets was required. In this case, the forensic targets included physical devices or emulated devices or device images or physical and logical extracts from devices. Locating physical devices for testing proved to be a challenge. Emulated devices for the Android platform were easy to access and set up. On the other hand, accessing iPhone platform device emulators proved to be difficult. Reliable device images or device extracts were obtained from open digital forensic corpora. Even if the corpora had the images and extracts, the number was limited. Device images and extracts are storage intensive and thus require large storage mediums and relatively fast connections to transfer them from one computing node to another.

- **Availability of comprehensive non-commercial forensic tools**

  Tools that deal with mobile device forensics are mainly commercial. They require buying or subscribing and their internal workings are not documented. The available open source or free tools are either outdated or not comprehensive enough to provide the required information. The forensic tool used in the study was Autopsy. It was able to work on the Android mobile platform only.

- **Documentation of proprietary technology**

  Much of the mobile device technology in circulation is the output of commercial companies that consider them to be trade secrets. This secrecy means locating documentation for the technology is very difficult. Without official documentation, the little information retrieved is subject to thorough testing and verification to prove its correctness. If no information is found, reverse engineering is the next best alternative to retrieving the required information. In this study, the iPhone and Android mobile platforms were used. Information on Android was more readily available and in a larger quantity than the iPhone.

5.3 Recommendations for future work

After pursuing this study, a number of items of work were identified that can be the basis of future work. The items are:

1. **Extend information collection capabilities**

   Other than phone call information, messaging information and location information, more information can be extracted from mobile devices. Sources of this information include deleted content, running processes, device memory, web browser bookmarks and history.

   With the ability to customise phone call and messaging applications, information regarding calls and messages may reside outside the default operating system's defined locations. For example, use of communication applications like WhatsApp or FaceBook Messenger for messaging or making phone calls is currently not visible to the system. Extending it to incorporate such information sources would be required.

2. **Graphical user interface for the tool**

   The information extracted from the mobile device forensic targets is currently in DFXML format. This mode of information presentation is not friendly to use on a daily basis and does not exploit the depth of the information available. For example, a timeline of events can be created based on the extracted information showing the location of the mobile device, when phone calls or messages were made / received. A Graphical User Interface would enable users to view the extracted information in a richer way than the DFXML format.

3. **Support other mobile platforms**

   The system currently supports iPhone and Android mobile platforms. However, more platforms exist, for example, BlackBerry OS, Windows Phone and Nokia's S40. Support for these platform would be an addition to the existing extraction platforms present in the system.

4. **Modify Nugget to fully utilise DFXML content**

   DFXML provides more information than Nugget can currently process. A good example of this is the provenance information. This information is crucial in showing how a forensic computation was performed and in what environment. This allows the independent replication of the forensic computation to validate the result of the

computation, if required.

5.4 Conclusion

Nugget allows digital forensic investigators to specify digital forensic computations and not worry about their implementation. The study aimed at improving Nugget by introducing the ability to specify and execute mobile device forensic computations. To accomplish this, an integration of mobile forensic tools to Nugget was designed, implemented and tested. This goal was achieved and in the course of doing so contributing to a new mobile forensic tool being developed and also the extension of Nugget to accommodate mobile device forensic computations.

REFERENCES

Alink, W., Bhoedjang, R., Boncz, P. & de Vries, A. (2006). XIRAF – XML-based indexing and querying for digital forensics. *Digital Investigation*, *3*, 50–58. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06). doi:https://doi.org/10.1016/j.diin.2006.06.016

Brothers, S. (2011). How cell phone "forensic" tools actually work–cell phone tool leveling system. In *DoD Cybercrime Conference*.

Carrier, B. (2018a). The SleuthKit. Retrieved November 3, 2018, from https://www.sleuthkit.org

Carrier, B. (2018b). FCAT(1) manual page. Retrieved December 20, 2018, from https://www.sleuthkit.org/sleuthkit/man/fcat.html

Casey, E., Back, G. & Barnum, S. (2015). Leveraging CybOX™ to standardize representation and exchange of digital forensic information. *Digital Investigation*, *12*, S102–S110. DFRWS 2015 Europe. doi:https://doi.org/10.1016/j.diin.2015.01.014

Cohen, M., Garfinkel, S. & Schatz, B. (2009). Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. *Digital Investigation*, *6*, S57–S68. The Proceedings of the Ninth Annual DFRWS Conference. doi:https://doi.org/10.1016/j.diin.2009.06.010

Common Digital Evidence Storage Format Working Group (Digital Forensics Research Workshop). (2016). Survey of disk image storage formats. Retrieved December 29, 2018, from http://old.dfrws.org/CDESF/survey-dfrws-cdesf-diskimg-01.pdf

Computing and Mathematical Sciences, University of Waikato. (2018). ARFF. Retrieved December 29, 2018, from https://github.com/Waikato/weka-wiki/blob/master/docs/arff_stable.md

Crawford, L. & matproud. (2012). iPhone Analyzer. Retrieved January 5, 2019, from https://sourceforge.net/p/iphoneanalyzer

DFRWS. (2018, August 14). Challenge details. Retrieved January 5, 2019, from https://github.com/dfrws/dfrws2018-challenge/blob/master/challenge-details

DFXML Working Group. (2017). XML Schema for Digital Forensics XML. Retrieved December 30, 2018, from https://github.com/dfxml-working-group/dfxml_schema

Digital Corpora. (2017, April 29). Digital Corpora. Retrieved November 5, 2018, from https://digitalcorpora.org/

Digital Corpora. (2018, July 27). Natioal Gallery DC 2012 Attack. Retrieved November 5, 2018, from https://digitalcorpora.org/corpora/scenarios/national-gallery-dc-2012-attack

Digital Corpora. (2015, May 17). Retrieved November 5, 2018, from http://downloads.digitalcorpora.org/corpora/mobile/2011-android/

Dublin Core Metadata Initiative. (2018). DCMI Metadata Terms. Retrieved December 30, 2018, from http://www.dublincore.org/specifications/dublin-core/dcmi-terms/#section-3

Edgar, T. W. & Manz, D. O. (2017). *Research Methods For Cyber Security*. Elsevier Inc.

Forensicon Inc. (2010, May 4). Cell phone & email forensics investigation cracks nyc times square car bombing case. Retrieved November 30, 2018, from https://www.forensicon.com/forensics-blotter/cell-phone-email-forensics-investigation-cracks-nyc-times-square-car-bombing-case/

Fowler, M. (2010). *Domain Specific Languages*. Addison-Wesley Professional.

Garfinkel, S. (2012a). Digital forensics XML and the DFXML toolset. *Digital Investigation*, *8*(3), 161–174. doi:https://doi.org/10.1016/j.diin.2011.11.002

Garfinkel, S. (2012b). Lessons learned writing digital forensics tools and managing a 30TB digital evidence corpus. *Digital Investigation*, *9*, S80–S89. The Proceedings of the Twelfth Annual DFRWS Conference. doi:https://doi.org/10.1016/j.diin.2012.05.002

Garfinkel, S. L. (2009). Automating Disk Forensic Processing with SleuthKit, XML and Python. In *Systematic Approaches to Digital Forensic Engineering, IEEE International Workshop on(SADFE)* (Vol. 00, pp. 73–84). doi:10.1109/SADFE.2009.12

Garfinkel, S., Farrell, P., Roussev, V. & Dinolt, G. (2009). Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, *6*, S2–S11. The Proceedings of the Ninth Annual DFRWS Conference. doi:https://doi.org/10.1016/j.diin.2009.06.016

Google. (2018). Android Debug Bridge (adb) | Android Developers. Retrieved December 20, 2018, from https://developer.android.com/studio/command-line/adb

JEDEC Solid State Technology Association. (2007). *Embedded MultiMediaCard (eMMC) Product Standard, Standard Capacity*. Retrieved December 20, 2018, from https://www.jedec.org/system/files/docs/JESD84-A41.pdf

Kothari, C. R. (2004). *Research methodology methods and techniques* (2nd ed.). New Age International Publishers.

Levine, B. N. & Liberatore, M. (2009). DEX: Digital evidence provenance supporting reproducibility and comparison. *Digital Investigation*, *6*, S48–S56. The Proceedings of the Ninth Annual DFRWS Conference. doi:https://doi.org/10.1016/j.diin.2009.06.011

Lillis, D., Becker, B. A., O'Sullivan, T. & Scanlon, M. (2016). Current Challenges and Future Research Areas for Digital Forensic Investigation. In *Annual ADFSL Conference*

on Digital Forensics, Security and Law (Vol. 6). Retrieved from https://commons.erau.edu/adfsl/2016/tuesday/6

Mazzetti, M., Tavernise, S. & Healy, J. (2010, May 4). Suspect, charged, said to admit to role in plot. Retrieved November 9, 2018, from https://www.nytimes.com/2010/05/05/nyregion/05bomb.html

McKemmish, R. (2008). When is Digital Evidence Forensically Sound? In I. Ray & S. Shenoi (Eds.), *Advances in digital forensics iv* (pp. 3–15). Boston, MA: Springer US.

Metz, J. (2018). libewf. Retrieved December 29, 2018, from https://github.com/libyal/libewf

Mikhaylov, I. & Skulkin, O. (2016). Chip-off technique in mobile forensics. Retrieved December 19, 2018, from https://www.digitalforensics.com/blog/chip-off-technique-in-mobile-forensics/

Murphy, C. A. (2011). Developing Process for Mobile Device Forensics. Retrieved December 21, 2018, from https://digital-forensics.sans.org/media/mobile-device-forensic-process-v3.pdf

OASIS Open. (2018). Introduction to STIX. Retrieved December 30, 2018, from https://oasis-open.github.io/cti-documentation/stix/intro

Omondi, M. (2019, January 29). Kenol ceo's phone seized in insider trading probe. Retrieved February 3, 2019, from https://www.businessdailyafrica.com/news/phone-seized-in-insider-trading-probe/539546-4956286-ppn2q6/index.html

Open Whisper Systems. (2019). signalapp/Signal-Android. Retrieved July 3, 2019, from https://github.com/signalapp/Signal-Android

Parr, T. (2014). *The definitive ANTLR 4 reference*. The Pragmatic Programmers, LLC.

Raghavan, S. (2013). Digital forensic research: Current state of the art. *CSI Transactions on ICT*, *1*(1), 91–114. doi:10.1007/s40012-012-0008-7

Roussev, V. (2015). Building a forensic computing language. In *2015 48th Hawaii International Conference on System Sciences* (pp. 5228–5233). doi:10.1109/HICSS.2015.617

Roussev, V., Bertino, E. & Sandhu, R. (2016). *Digital forensic science: Issues, methods, and challenges*. Morgan & Claypool. Retrieved from https://ieeexplore.ieee.org/document/7809443

Skulkin, O., Tindall, D. & Tamma, R. (2018). *Learning Android Forensics* (2nd ed.). Packt Publishing Ltd.

Stelly, C. & Roussev, V. (2018). Nugget: A digital forensics language. In *DFRWS 2108 Europe - Proceedings of the Fifth Annual DFRWS Europe*, Elsevier Ltd.

Stelly, C. & Roussev, V. (2017). SCARF: A container-based approach to cloud-scale digital forensic processing. *Digital Investigation*, *22*, S39–S47. doi:https://doi.org/10.1016/j.diin.2017.06.008

Tahiri, S. (2016). *Mastering Mobile Forensics*. Packt Publishing Ltd.

Tamma, R., Skulkin, O., Mahalik, H. & Bommisetty, S. (2018). *Practical Mobile Forensics* (3rd ed.). Packt Publishing Ltd.

The iPhone Wiki. (2015a). The iPhone Wiki. Retrieved January 5, 2019, from https://www.theiphonewiki.com

The iPhone Wiki. (2015b). Messages - The iPhone Wiki. Retrieved January 5, 2019, from https://www.theiphonewiki.com/wiki/Messages

The LineageOS Project. (2019). LineageOS Android Distribution. Retrieved January 8, 2019, from https://www.lineageos.org/

The MITRE Corporation. (2016). CybOX Schemas and Schema Development. Retrieved December 30, 2018, from https://github.com/CybOXProject/schemas

The Volatility Foundation. (2018). Volatility. Retrieved November 3, 2018, from https://www.volatilityfoundation.org/

van den Bos, J. & van der Storm, T. (2011). Bringing Domain-Specific Languages to Digital Forensics. In *International conference on software engineering*. doi:10.1145/1985793.1985887

Venema, W. (2018). ICAT(1) manual page. Retrieved December 20, 2018, from https://www.sleuthkit.org/sleuthkit/man/icat.html

Warden, P. (2011). iPhone Tracker. Retrieved January 5, 2019, from https://github.com/petewarden/iPhoneTracker

Wireshark Foundation. (2018). Tshark. Retrieved November 3, 2018, from https://www.wireshack.org

Appendix A

Language specifications

A.1 Mobile extension to the DFXML version 1.2.0 specification

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:dfxml="http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML"
 xmlns:mobile="https://github.com/ngash/dfxml/mobile"
 targetNamespace="https://github.com/ngash/dfxml/mobile"
  elementFormDefault="qualified">
  <xs:annotation>
    <xs:documentation>
      This is the schema file for mobile extensions for DFXML v1.2.0
    </xs:documentation>
  </xs:annotation>
  <xs:import namespace="http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML" schemaLocation="dfxml.xsd" />
  <xs:import namespace="http://purl.org/dc/elements/1.1/" schemaLocation="ref/dc.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="ref/xml.xsd"/>

  <xs:element name="device_info">
    <xs:annotation>
      <xs:documentation>A representation of a mobile device information</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element minOccurs="0" maxOccurs="1" type="dfxml:string" name="name"></xs:element>
        <xs:element minOccurs="0" maxOccurs="1" type="dfxml:string" name="model"></xs:element>
        <xs:element minOccurs="0" maxOccurs="1" type="dfxml:string" name="release"></xs:element>
        <xs:element minOccurs="0" maxOccurs="1" type="dfxml:string" name="manufacturer"></xs:element>
        <xs:element minOccurs="0" maxOccurs="1" type="dfxml:string" name="serial_number"></xs:element>
        <xs:element minOccurs="0" maxOccurs="1" type="dfxml:string" name="imei"></xs:element>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="sms_mms_kind_type">
    <xs:simpleContent>
      <xs:restriction base="dfxml:string">
        <xs:enumeration value="sms" />
        <xs:enumeration value="mms" />
```

```
      </xs:restriction>
    </xs:simpleContent>
</xs:complexType>

<xs:element name="sms_mms">
    <xs:annotation>
      <xs:documentation>A representation of an SMS or MMS message</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="kind" minOccurs="0" maxOccurs="1" type="mobile:sms_mms_kind_type" />
        <xs:element name="sender" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="recipient" minOccurs="0" maxOccurs="unbounded" type="dfxml:string" />
        <xs:element name="body" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="date_sent" minOccurs="0" maxOccurs="1" type="dfxml:dftime" />
        <xs:element name="date_received" minOccurs="0" maxOccurs="1" type="dfxml:dftime" />
        <xs:element name="read" minOccurs="0" maxOccurs="1" type="dfxml:bool01" />
        <xs:element name="headers" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="subject" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="country_code" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="type" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
</xs:element>

<xs:element name="call">
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="from" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="to" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="date_called" minOccurs="0" maxOccurs="1" type="dfxml:dftime" />
        <xs:element name="duration" minOccurs="0" maxOccurs="1" type="dfxml:nonNegativeInteger" />
        <xs:element name="country_code" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="type" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:element name="blocked" minOccurs="0" maxOccurs="1" type="dfxml:bool01" />
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
</xs:element>

<xs:element name="location">
    <xs:annotation>
      <xs:documentation>A representation of a (latitude, longitude) location</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="lat" minOccurs="1" maxOccurs="1" type="dfxml:float" />
        <xs:element name="long" minOccurs="1" maxOccurs="1" type="dfxml:float" />
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
```

```
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
</xs:element>

<xs:element name="app">
   <xs:annotation>
      <xs:documentation>A representation of a mobile device application</xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="name" minOccurs="1" maxOccurs="1" type="dfxml:string" />
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
</xs:element>

<xs:element name="contact">
   <xs:annotation>
      <xs:documentation>A representation of a contact</xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="name" minOccurs="1" maxOccurs="1" type="dfxml:string" />
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
</xs:element>

<xs:element name="process">
   <xs:annotation>
      <xs:documentation>A representation of a  process</xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="name" minOccurs="1" maxOccurs="1" type="dfxml:string" />
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
</xs:element>

<xs:element name="browsing">
   <xs:annotation>
      <xs:documentation>A representation of a web location</xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="url" minOccurs="1" maxOccurs="1" type="dfxml:string" />
```

```
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="command_line">
    <xs:annotation>
      <xs:documentation>A representation of command line execution</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="dfxml:string">
          <xs:attribute name="sequence" type="xs:nonNegativeInteger">
            <xs:annotation>
              <xs:documentation>Number in the sequence of commands </xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="error">
    <xs:annotation>
      <xs:documentation>Documentation of an error occurence</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="code" minOccurs="1" maxOccurs="1" type="dfxml:nonNegativeInteger" />
        <xs:element name="message" minOccurs="1" maxOccurs="1" type="dfxml:string" />
        <xs:element name="description" minOccurs="0" maxOccurs="1" type="dfxml:string" />
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A.2 Nugget language specification

```
grammar Nugget;

@header {
   // import "../NTypes"
}

prog: (            preamble |
           define_assign |
      operation_on_singleton |
           singleton_var )*
      EOF
```

```
;

preamble: (preamble_stmt)+ ;
preamble_stmt: preamble_action STRING;
preamble_action: ('case_number'|'investigator'|'investigation_date'|'workspace');

define_assign:   define |
                 define_tuple |
                 assign
;

define: ID nugget_type LISTOP? ;

define_tuple: ID 'tuple[' (','? nugget_type)+ ']' LISTOP?;


assign: ID '=' STRING ('|' nugget_action)* |
       ID '=' ID     ('|' nugget_action)*
;

operation_on_singleton: singleton_op  ID (',' ID)* output_as?;

output_as: 'as' output_type;
output_type: 'json'|'table';

singleton_op: ('type' | 'print' | 'size' | 'typex' | 'printx' | 'raw') ;

singleton_var: ID;

nugget_type:
     'string'       |
     'sha1'         |
     'md5'          |
     'ntfs'         |
     'file'         |
     'packet'       |
     'pcap'         |
     'exifinfo'     |
     'datetime'     |
     'memory'       |
     'http'         |
     'listof-md5'   |
     'listof-sha1'  |
     'listof-sha256' |
     'adb'          |
     'message'      |
     'call'         |
     'location'     |
     'contact'      |
     'files'        |
     'devinfo'
     ;

nugget_action: action_word ;
```

```
action_word:
    filter          |
    'extract' asType  |
    'sort'    byField |
    'sha1'          |
    'md5'           |
    'sha256'        |
    'getGetRequests'  |
    'diskinfo'      |
    'union'    ID    |
    'pslist'        |
    '%%%'
;

asType: 'as' nugget_type (byteOffsetSize)?;
byField:'by' ID;

byteOffsetSize : '['INT ',' INT ']';

filter : 'filter' filter_term (',' filter_term)*;
filter_term: ID COMPOP STRING;

COMPOP: ('>' | '<' | '>=' | '<=' | '==');
LISTOP: '[]';

INT : [0-9]+;
ID : [a-zA-Z]+[a-zA-Z0-9_]* ('.' [a-zA-Z]+[a-zA-Z0-9_]*)?;
STRING: '"' ('""'|~'"')* '"';

WS : [ \t\r\n]+ -> skip;
NL : '\r'? '\n';

LINE_COMMENT: '//' ~[\r\n]* -> skip;
```

Appendix B

Sample source code and output listing

B.1 Sample source code listing

Listing B.1: Sample source code listing showing the archive extraction logic

```
package nugget

import (
   ”fmt”
   ”github.com/ngash/nugget−tools/dfxml”
   ”github.com/ngash/nugget−tools/log”
   ”github.com/ngash/nugget−tools/mobile”
   ”github.com/ngash/nugget−tools/mobile/android”
   ”github.com/ngash/nugget−tools/mobile/iphone”
   ”strings”
   ”time”
)

const (
   CMD_MESSAGE    = ”message”
   CMD_CALL       = ”call”
   CMD_FILE       = ”file”
   CMD_LOCATION   = ”location”
   CMD_CONTACT    = ”contact”
   CMD_PROCESS    = ”process”
   CMD_APPLICATION = ”application”
   CMD_DEVICEINFO  = ”devinfo”
)
const (
   PLATFORM_IPHONE  = ”iphone”
   PLATFORM_ANDROID = ”android”
   PLATFORM_ADB     = ”adb”
)

type Mobile struct{ Workspace string }
type MobileArgs struct {
   Target     string
   Platform   string
   Command    string
   CommandArgs map[string]string
}

func (m Mobile) Dfxml(args MobileArgs, reply ∗dfxml.DFXML) error {
   log.Debug(”%+v”, args)
   var w mobile.Wrapper
   switch args.Platform {
```

```
    case PLATFORM_IPHONE:
        w = iphone.ImageWrapper{Workspace: m.Workspace}
    case PLATFORM_ANDROID:
        w = android.ImageWrapper{Workspace: m.Workspace}
    case PLATFORM_ADB:
        w = android.DeviceWrapper{Workspace: m.Workspace}
        w.(android.DeviceWrapper).SetRoot(args.Target, true)
    default:
        return fmt.Errorf("Unknown mobile platform '%s'", args.Platform)
    }

    switch args.Command {
    case CMD_FILE, "files":
        return extractFiles(w, args, reply)
    case CMD_CALL:
        return extractCalls(w, args, reply)
    case CMD_MESSAGE:
        return extractMessages(w, args, reply)
    case CMD_LOCATION:
        return extractLocations(w, args, reply)
    case CMD_CONTACT:
        return extractContacts(w, args, reply)
    case CMD_APPLICATION:
        return extractApps(w, args, reply)
    case CMD_PROCESS:
        return extractProcesses(w, args, reply)
    case CMD_DEVICEINFO:
        return extractDeviceInfo(w, args, reply)
    default:
        return fmt.Errorf("Unknown command '%s'", args.Command)
    }
}

func extractDeviceInfo(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    start_time := time.Now()
    resp, cmdstack, err := w.DeviceInfo(args.Target)
    stop_time := time.Now()
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.DeviceInfo = resp
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
func extractProcesses(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    start_time := time.Now()
    resp, cmdstack, err := w.ListProcesses(args.Target)
    stop_time := time.Now()
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.Process = resp
```

```
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
func extractApps(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    start_time := time.Now()
    resp, cmdstack, err := w.ListApplications(args.Target)
    stop_time := time.Now()
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.Application = resp
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
func extractContacts(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    start_time := time.Now()
    resp, cmdstack, err := w.ListContacts(args.Target)
    stop_time := time.Now()
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.Contact = resp
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
func extractLocations(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    start_time := time.Now()
    resp, cmdstack, err := w.ListLocations(args.Target)
    stop_time := time.Now()
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.Location = resp
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
func extractFiles(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    fname := args.CommandArgs["filename"]
    if fname == "" {
        fname = "/"
    }
    fxn := w.ListFiles
    if strings.Contains(fname, "*") {
        fxn = w.FindFiles
    }
    start_time := time.Now()
    resp, cmdstack, err := fxn(args.Target, fname)
    stop_time := time.Now()
```

```
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.FileObject = resp
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
func extractMessages(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    start_time := time.Now()
    resp, cmdstack, err := w.ListMessages(args.Target)
    stop_time := time.Now()
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.SMSMMSMessage = resp
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
func extractCalls(w mobile.Wrapper, args MobileArgs, reply *dfxml.DFXML) error {
    start_time := time.Now()
    resp, cmdstack, err := w.ListCalls(args.Target)
    stop_time := time.Now()
    if err != nil {
        log.Error("%+v", err)
    }
    d := dfxml.NewDFXML()
    d.PhoneCall = resp
    updateDFXML(d, w, cmdstack, start_time, stop_time, args.Target, err)
    *reply = *d
    return nil
}
```

B.2 Sample output listing

B.2.1 SMS and MMS extraction

Listing B.2: SMS and MMS DFXML extraction as DFXML

```
<?xml version="1.0" encoding="UTF-8"?>
<dfxml xmlns="http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML" xmlns:xsi="http://www.w3.org/2001/
    ↪ XMLSchema-instance" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:mobile="https://github.com/ngash/dfxml/
    ↪ mobile" version="1.2.0">
  <metadata></metadata>
  <creator>
    <program>nugget-tools</program>
    <version>0.0.1</version>
    <execution_environment>
      <os_sysname>Linux</os_sysname>
```

```
    <os_release>5.0.5-arch1-1-ARCH</os_release>
    <os_version>#1 SMP PREEMPT Wed Mar 27 17:53:10 UTC 2019</os_version>
    <host>blackpearl</host>
    <arch>x86_64</arch>
    <uid>1000</uid>
    <username>barbossa</username>
    <start_time>2019-04-08T03:39:08.386512884+03:00</start_time>
    <mobile:command_line mobile:sequence="1">/usr/bin/tar -t -f evidence/tracy-phone-2012-07-15-final.tar</
      ↪ mobile:command_line>
    <mobile:command_line mobile:sequence="2">/usr/bin/tar -x -C /home/barbossa/Desktop/work/msc/units/y2-project/
      ↪ nugget-tools/stuff/workspace -f evidence/tracy-phone-2012-07-15-final.tar ./private/var/mobile/Library/SMS/
      ↪ sms.db</mobile:command_line>
  </execution_environment>
 </creator>
 <rusage>
  <utime>0.000213956</utime>
  <stime>0.000138481</stime>
  <maxrss>199640</maxrss>
  <minflt>131813</minflt>
  <majflt>0</majflt>
  <nswap>0</nswap>
  <inblock>0</inblock>
  <oublock>0</oublock>
  <clocktime>0.347418644</clocktime>
 </rusage>
 <mobile:sms_mms>
  <mobile:kind>sms</mobile:kind>
  <mobile:sender>+15713083236</mobile:sender>
  <mobile:body>What are you up to this weekend? </mobile:body>
  <mobile:date_received>2012-06-13T00:25:04+03:00</mobile:date_received>
  <mobile:read>1</mobile:read>
  <mobile:country_code>us</mobile:country_code>
  <mobile:type>inbox</mobile:type>
 </mobile:sms_mms>
</dfxml>
```

B.2.2 Phone call extraction

Listing B.3: Phone call extraction as DFXML

```
<?xml version="1.0" encoding="UTF-8"?>
<dfxml xmlns="http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML" xmlns:xsi="http://www.w3.org/2001/
      ↪ XMLSchema-instance" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:mobile="https://github.com/ngash/dfxml/
      ↪ mobile" version="1.2.0">
 <metadata></metadata>
 <creator>
  <program>nugget-tools</program>
  <version>0.0.1</version>
  <execution_environment>
   <os_sysname>Linux</os_sysname>
   <os_release>5.0.5-arch1-1-ARCH</os_release>
   <os_version>#1 SMP PREEMPT Wed Mar 27 17:53:10 UTC 2019</os_version>
   <host>blackpearl</host>
```

```
    <arch>x86_64</arch>
    <uid>1000</uid>
    <username>barbossa</username>
    <start_time>2019-04-08T03:39:20.457623379+03:00</start_time>
    <mobile:command_line mobile:sequence="1">/usr/bin/tar -t -f evidence/tracy-phone-2012-07-15-final.tar</
      ↪ mobile:command_line>
    <mobile:command_line mobile:sequence="2">/usr/bin/tar -x -C /home/barbossa/Desktop/work/msc/units/y2-project/
      ↪ nugget-tools/stuff/workspace -f evidence/tracy-phone-2012-07-15-final.tar ./private/var/wireless/Library/
      ↪ CallHistory/call_history.db</mobile:command_line>
  </execution_environment>
 </creator>
 <rusage>
  <utime>0.000173707</utime>
  <stime>0.000178225</stime>
  <maxrss>252172</maxrss>
  <minflt>135095</minflt>
  <majflt>0</majflt>
  <nswap>0</nswap>
  <inblock>0</inblock>
  <oublock>0</oublock>
  <clocktime>0.339693871</clocktime>
 </rusage>
 <mobile:call>
  <mobile:from>6508870260</mobile:from>
  <mobile:date_called>2012-06-12T23:04:50+03:00</mobile:date_called>
  <mobile:duration>20</mobile:duration>
  <mobile:country_code>310</mobile:country_code>
  <mobile:type>incoming</mobile:type>
 </mobile:call>
</dfxml>
```

B.2.3 Device information extraction

Listing B.4: Device information extraction as DFXML

```
<?xml version="1.0" encoding="UTF-8"?>
<dfxml xmlns="http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:mobile="https://github.com/ngash/dfxml/mobile" version="1.2.0">
 <metadata></metadata>
 <creator>
  <program>nugget-tools</program>
  <version>0.0.1</version>
  <execution_environment>
   <os_sysname>Linux</os_sysname>
   <os_release>5.0.5-arch1-1-ARCH</os_release>
   <os_version>#1 SMP PREEMPT Wed Mar 27 17:53:10 UTC 2019</os_version>
   <host>blackpearl</host>
   <arch>x86_64</arch>
   <uid>1000</uid>
   <username>barbossa</username>
   <start_time>2019-03-17T13:21:15.491100149+03:00</start_time>
```

```
    <mobile:command_line mobile:sequence="1">/usr/bin/tar -t -f evidence/tracy-phone-2012-07-15-final.tar</
      ↪ mobile:command_line>
    <mobile:command_line mobile:sequence="2">/usr/bin/tar -x -C /home/barbossa/Desktop/work/msc/units/y2-project/nugget-
      ↪ tools/stuff/workspace -f evidence/tracy-phone-2012-07-15-final.tar ./System/Library/CoreServices/SystemVersion.plist
      ↪ </mobile:command_line>
  </execution_environment>
 </creator>
 <rusage>
  <utime>0.000190765</utime>
  <stime>0.000161015</stime>
  <maxrss>176716</maxrss>
  <minflt>130250</minflt>
  <majflt>0</majflt>
  <nswap>0</nswap>
  <inblock>0</inblock>
  <oublock>0</oublock>
  <clocktime>0.475734025</clocktime>
 </rusage>
 <mobile:device_info>
  <mobile:name>iPhone OS</mobile:name>
  <mobile:release>4.2.1</mobile:release>
 </mobile:device_info>
</dfxml>
```

B.2.4 Location extraction

Listing B.5: Location information extraction as DFXML

```
<?xml version="1.0" encoding="UTF-8"?>
<dfxml xmlns="http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML" xmlns:xsi="http://www.w3.org/2001/
      ↪ XMLSchema-instance" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:mobile="https://github.com/ngash/dfxml/mobile
      ↪ " version="1.2.0">
 <metadata></metadata>
 <creator>
  <program>nugget-tools</program>
  <version>0.0.1</version>
  <execution_environment>
   <os_sysname>Linux</os_sysname>
   <os_release>5.0.5-arch1-1-ARCH</os_release>
   <os_version>#1 SMP PREEMPT Wed Mar 27 17:53:10 UTC 2019</os_version>
   <host>blackpearl</host>
   <arch>x86_64</arch>
   <uid>1000</uid>
   <username>barbossa</username>
   <start_time>2019-04-08T03:38:48.538677359+03:00</start_time>
   <mobile:command_line mobile:sequence="1">/usr/bin/tar -t -f evidence/tracy-phone-2012-07-15-final.tar</
     ↪ mobile:command_line>
   <mobile:command_line mobile:sequence="2">/usr/bin/tar -x -C /home/barbossa/Desktop/work/msc/units/y2-project/nugget-
     ↪ tools/stuff/workspace -f evidence/tracy-phone-2012-07-15-final.tar ./private/var/root/Library/Caches/locationd/
     ↪ consolidated.db</mobile:command_line>
  </execution_environment>
 </creator>
 <rusage>
```

```
   <utime>0.000245826</utime>
   <stime>0.000180958</stime>
   <maxrss>231728</maxrss>
   <minflt>137633</minflt>
   <majflt>0</majflt>
   <nswap>0</nswap>
   <inblock>0</inblock>
   <oublock>0</oublock>
   <clocktime>0.539221063</clocktime>
  </rusage>
  <mobile:location>
   <mobile:long>−77.11546951</mobile:long>
   <mobile:lat>38.87767624</mobile:lat>
   <mobile:source></mobile:source>
   <mobile:confidence>70</mobile:confidence>
   <mobile:timestamp>2012−06−13T22:01:21+03:00</mobile:timestamp>
   <mobile:cell_mcc>310</mobile:cell_mcc>
   <mobile:cell_mnc>410</mobile:cell_mnc>
   <mobile:cell_lac>7985</mobile:cell_lac>
   <mobile:cell_ci>160043533</mobile:cell_ci>
  </mobile:location>
  <mobile:location>
   <mobile:long>−77.08571225</mobile:long>
   <mobile:lat>38.82908231</mobile:lat>
   <mobile:source></mobile:source>
   <mobile:confidence>50</mobile:confidence>
   <mobile:timestamp>2012−07−10T19:46:29+03:00</mobile:timestamp>
   <mobile:wifi_mac>0:a0:f8:bc:25:a5</mobile:wifi_mac>
  </mobile:location>
</dfxml>
```