# CSE 469 Computer and Network Forensics
# Class Project (Group Project) April 21st, 2016

## Overview

The purpose of this project is to write code that pertains to each of the three main tasks of the forensic process: acquisition, authentication, and analysis. Since the project will require many address conversions, you are also required to write a utility that can convert between multiple hard drive address types, the details of which are below. The code you write for this project should be in a Linux-compatible language (e.g. C/C++, Java, Python, Perl, etc.).

## Task 1. Conversion Utility (40pts)

This task is to build two conversion utilities: address conversion and MAC conversion.

**Tool a) Address Conversion**

In order to both simplify addressing mechanisms and to reduce the number of bits necessary to locate all areas within a logical space (like a partition) that hold data, multiple addressing techniques are used on IBM PC-compatible hard drives and in FAT file systems. You are to write a **Unix-like command line utility** that will convert between three different address types when an address of a different type is given. Use the following usage specifications for your utility:

```
address4forensics -L|-P|-C [-b offset] [-B [-s bytes]] [-l address] [-p address] [-c address -k sectors -r sectors -t tables -f sectors]
-L, --logical
      Calculate the logical address from either the cluster address or
      the physical address. Either –c or –p must be given.
-P, --physical
      Calculate the physical address from either the cluster address
      or the logical address. Either –c or –l must be given.
-C, --cluster
      Calculate the cluster address from either the logical address or
      the physical address. Either –l or –p must be given.
-b offset, --partition-start=offset
      This specifies the physical address (sector number) of the start
      of the partition, and defaults to 0 for ease in working with
      images of a single partition. The offset value will always
      translate into logical address 0.
-B, --byte-address
      Instead of returning sector values for the conversion, this
      returns the byte address of the calculated value, which is the
      number of sectors multiplied by the number of bytes per sector.
-s bytes, --sector-size=bytes
      When the –B option is used, this allows for a specification of
      bytes per sector other than the default 512. Has no affect on
      output without –B.
-l address, --logical-known=address
      This specifies the known logical address for calculating either
      a cluster address or a physical address. When used with the –L
      option, this simply returns the value given for address.
```

```
-p address, --physical-known=address
      This specifies the known physical address for calculating either
      a cluster address or a logical address. When used with the -P
      option, this simply returns the value given for address.
-c address, --cluster-known=address
      This specifies the known cluster address for calculating either
      a logical address or a physical address. When used with the -C
      option, this simply returns the value given for address. Note
      that options -k, -r, -t, and -f must be provided with this
      option.
-k sectors, --cluster-size=sectors
      This specifies the number of sectors per cluster.
-r sectors, --reserved=sectors
      This specifies the number of reserved sectors in the partition.
-t tables, --fat-tables=tables
      This specifies the number of FAT tables, which is usually 2.
-f sectors, --fat-length=sectors
      This specifies the length of each FAT table in sectors.
```

An example of this in use would be the following, where the desired number is the logical address of physical sector 12345678 in a partition that begins at physical sector 128:

```
$ address4forensics -L -b 128 --physical-known=12345678
12345550
```

Another example shows the utility getting the physical address of cluster 58, in a partition that begins at physical sector 128, has 2 FAT tables that are each 16 sectors long, 6 reserved sectors, and 4 sectors per cluster:

```
$ address4forensics -P --partition-start=128 -c 58 -k 4 -r 6 -t 2 -f
16
390
```

**Tool b) MAC Conversion**

The purpose of this task is to write code that performs the MAC conversion based on the following usage specification and input/output scheme. This conversion MUST follow the procedure that we discussed in the lecture note #6. We assume a little endian ordering is applied.

```
mac_conversion -T|-D [-f filename | -h hex value ]
-T Use time conversion module. Either -f or -h must be given.
-D Use date conversion module. Either -f or -h must be given.
-f filename
      This specifies the path to a filename that includes a hex value
      of time or date. Note that the hex value should follow this
      notation: 0x1234. For the multiple hex values in either a file
      or a command line input, we consider only one hex value so the
      recursive mode for MAC conversion is optional.
-h hex value
      This specifies the hex value for converting to either date or
      time value. Note that the hex value should follow this notation:
      Ox1234. For the multiple hex values in either a file or a
```

```
command line input, we consider only one hex value so the
recursive mode for MAC conversion is optional.
```

The converted time or date value should be based on the following scheme:

```
Time: hr:min:sec AM|PM
Date: Month day, Year
```

An example of this in use would be the following, where the time conversion is requested with a file:

```
$ mac_conversion -T -f test.txt
Time: 10:31:44 AM
```

Another example shows the date conversion with the hex value as an input:

```
$ mac_conversion -D -h 0x4f42
Date: Feb 15, 2013
```

## Task 2. Acquisition, Authentication, and Analysis (60 pts)

For this project, it is assumed that the work of acquiring a digital copy of a hard drive has already been performed. The following requirements should be accomplished in your code: a single package is preferred. A sample raw image and corresponding information are available at Blackboard.

**Requirement a)** First, you are to write a program that can take as input *the path to a RAW image* and open it as read-only for the requirements (b) and (c).

**Requirement b)** Before opening the RAW image, your program should first calculate MD5 and SHA1 checksums for the image. Both checksums should be stored as MD5-image-name.txt and SHA1-image-name.txt. For example, the name of RAW image is Sparky then your authentication module needs to generate MD5-Sparky.txt and SHA1-Sparky.txt before opening the RAW image.

**Requirement c)** The next tasks your program must be able to accomplish are:

1. Locate and extract the partition tables from the master boot record (MBR)
    a. Your program **MUST** generate the partition type including hex value and corresponding type, start sector address, and size of each partition in decimal as follows:
    `(07) NTFS, 0002056320, 0000208845`
2. For FAT16/32 partition, read each partition's volume boot record (VBR) and retrieve the geometric data of the file system. Your code **MUST** generate the following layout information:
    a. The layout should include:

```
Reserved area:  Start sector: 0  Ending sector: 6  Size:  7 sectors
Sectors per cluster:  32 sectors
FAT area:  Start sector: 7 Ending sector:  598
# of FATs: 2
The size of each FAT: 249 sectors
The first sector of cluster 2: 12234 sectors
```

Note. For the partition types, please refer to the lecture note #7 (Page 11, Slide 22)

## Deliverables

1. A report should include the following items and be submitted via Blackboard:
    a. Brief description of your design including the contribution of each member
    b. Compilation/usage instructions should include makefile, configuration details and screenshots
    c. The source code