Introduction to BitCurator

BitCurator Users Forum
Yale University Libraries
New Haven, CT
24 October 2019

Amy Berish
Dianne Dietrich
[Matthew] Farrell
Martin Gengenbach

Introductions and setup

- How is everyone doing?
- Are you ready to rumble?
- Put `buf19.img` on your Desktop
- Make a new folder on the Desktop called `buf19_output`

This Morning:

- Try to mount the test disk image
- bulk extractor to find personally identifying information
- fiwalk to generate technical metadata (DFXML)
- methods of file extraction for disk images

Let's try to mount this image

- Right-click and select "Disk Image Mount"

Let's try to mount this image

- Right-click and select "Disk Image Mount"
- …does anything even happen?
- Oh no.
- Do not panic.

Obtaining content from a disk image

- Even if you can't mount a disk image, there are ways you can analyze its contents, and even extract files, without mounting it.
- Let's try to see what we can learn about this image, even though we can't mount it (right now).

What is PII and how do you look for it?

- Personally identifying information can be Social Security Numbers, bank account information, credit card information, etc
- Utilities can look for patterns to flag sensitive material
- Can't find everything, and may turn up false positives

What is bulk_extractor?

- Command-line tool with many PII-patterns built in
- Can act on disk images, single files, directories
- Benefit is that it is file-type agnostic -- it will search readable contents of whatever you point it to

Running bulk_extractor (nutshell)

- bulk_extractor requires two arguments
  - □ Path to an output directory
  - □ Path to input disk image

Base command

```
bulk_extractor -o [OUTPUT_DIR] [DISK_IMAGE_FILENAME]
```

Running bulk_extractor (step-by-step)

1. Open a Terminal Window
2. Enter `bulk_extractor -o ~/Desktop/buf19_output/pii_results ~/Desktop/buf19.img`
3. Press Enter

Reflection 1

- How do you know the tool has completed successfully?
- Where do you find the output?
- How is the output organized?
- Is there personally identifying in this disk image?
- Can you determine what files have personally identifying information with this output?

Reflection 2

- What other types of sensitive data might be missing from this output?
- What do you think you might do with this information -- in the form that it is in -- at this point?
- Is there anything unclear to you at this point about the tool, or the disk image itself?

How do you get technical metadata from disk images?

- Certain utilities can take disk images as input
  (as opposed to individual files)
- When interacting with tools that work on disk images, there are a few things to be aware of:
  - ☐ Default / auto-detect settings may be misleading
  - ☐ Some filesystems are incompatible with certain tools

What is fiwalk?

- fiwalk is a utility that produces technical metadata (e.g., hash values, time stamps, etc) for files found on disk images
- It can output technical metadata in a variety of ways, including DFXML output

Running fiwalk (nutshell)

- fiwalk has one requirement, the disk image filename
- Additional options include
  - Only reporting allocated files
  - Reporting filetypes of each file on the image
  - Output to XML

Running fiwalk (step-by-step)

- Open a Terminal Window
- Type
  ```
  fiwalk -O -f -X
  ~/Desktop/buf19_output/fiwalk_output.xml
  ~/Desktop/buf19.img
  ```
- Press Enter

Reflection 1

- What do you think about this output?
- What does it tell you about the disk image?
- What parts of it don't make sense to you?
- Where can you find information about the files?
- How might you use this information in a workflow?
- What additional questions do you have?

After Break:

- File extraction using tsk_recover and the BitCurator Disk Image Access tool

File extraction

- Also referred to as “carve/carving files”
- Command line tool: tsk_recover
- GUI tool: BitCurator Disk Image Access

What is tsk_recover?

- Command line tool
- Exports files from a disk image to a directory
- By default, recovers unallocated files (free space) only
- Use -a flag to recover allocated files only
- Use -e flag to recover all (unallocated + allocated) files

Running tsk_recover (nutshell)

- Must be run from the directory containing the disk image
- Tsk_recover requires two arguments:
  - □ Disk image filename
  - □ Output directory (where should the files be saved?)

Base command:

```
tsk_recover -a [DISK_IMAGE_FILENAME] [OUTPUT_DIR]
```

Running tsk_recover (step-by-step)

1. Open terminal window
2. Navigate to the directory containing the disk image:

```
cd Desktop
```

3. `tsk_recover -a ./buf19.img ./buf19_output/objects`
4. ENTER

Error!

Cannot determine file system type (Sector offset: 0)

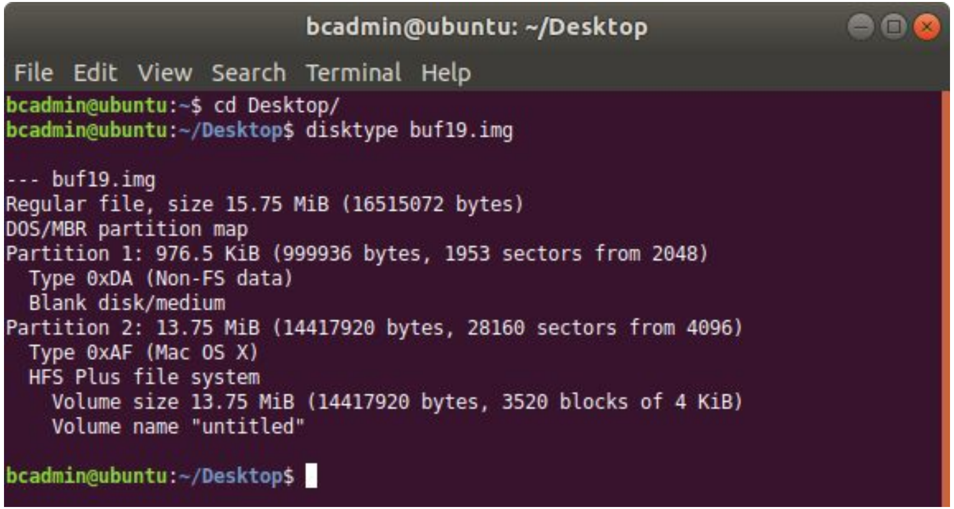
Files recovered: 0

disktype

- Command line disk format detector
- Outputs information about disk image contents, file systems, partition tables, etc.

Step by Step:

1. **`cd Desktop`**
2. **`disktype buf19.img`**
3. ENTER



bcadmin@ubuntu: ~/Desktop
File Edit View Search Terminal Help

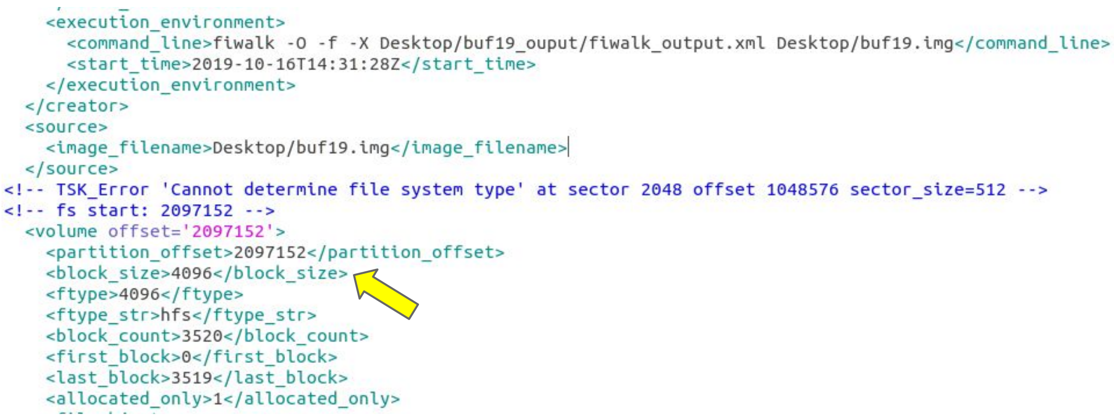
```
bcadmin@ubuntu:~$ cd Desktop/
bcadmin@ubuntu:~/Desktop$ disktype buf19.img

--- buf19.img
Regular file, size 15.75 MiB (16515072 bytes)
DOS/MBR partition map
Partition 1: 976.5 KiB (999936 bytes, 1953 sectors from 2048)
  Type 0xDA (Non-FS data)
  Blank disk/medium
Partition 2: 13.75 MiB (14417920 bytes, 28160 sectors from 4096)
  Type 0xAF (Mac OS X)
  HFS Plus file system
    Volume size 13.75 MiB (14417920 bytes, 3520 blocks of 4 KiB)
    Volume name "untitled"

bcadmin@ubuntu:~/Desktop$ 
```

fiwalk_ouput.xml

Volume offset tag



```
    <execution_environment>
      <command_line>fiwalk -O -f -X Desktop/buf19_ouput/fiwalk_output.xml Desktop/buf19.img</command_line>
      <start_time>2019-10-16T14:31:28Z</start_time>
    </execution_environment>
  </creator>
  <source>
    <image_filename>Desktop/buf19.img</image_filename>
  </source>
<!-- TSK_Error 'Cannot determine file system type' at sector 2048 offset 1048576 sector_size=512 -->
<!-- fs start: 2097152 -->
  <volume offset='2097152'>
    <partition_offset>2097152</partition_offset>
    <block_size>4096</block_size>
    <ftype>4096</ftype>
    <ftype_str>hfs</ftype_str>
    <block_count>3520</block_count>
    <first_block>0</first_block>
    <last_block>3519</last_block>
    <allocated_only>1</allocated_only>
```

Running tsk_recover (step-by-step)

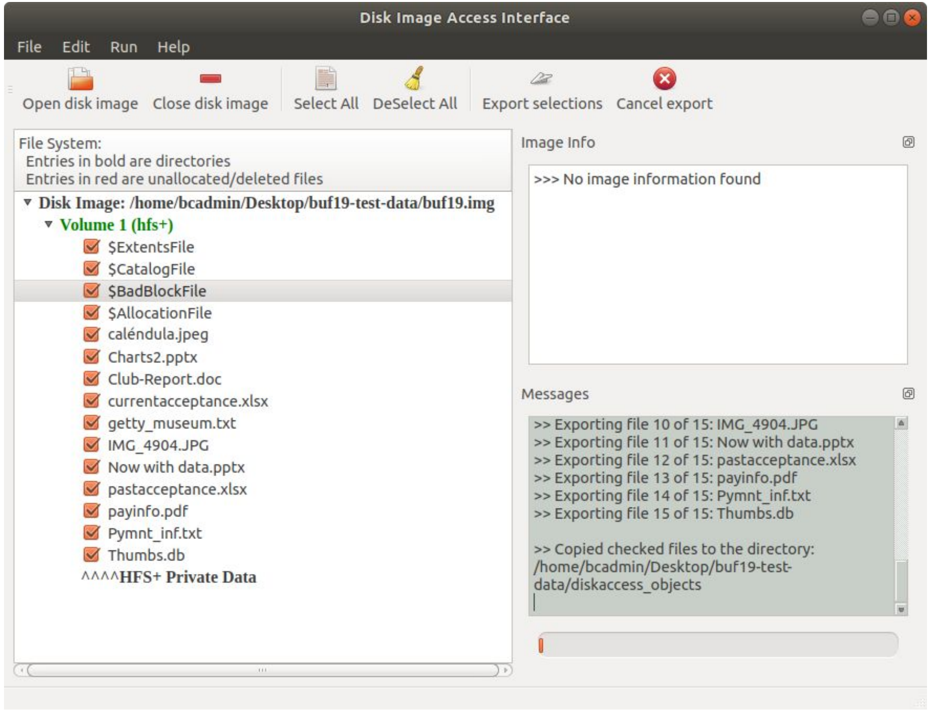
1. Open terminal window
2. Navigate to the directory containing the disk image:

   ```
   cd Desktop
   ```

3. Run `tsk_recover -a -o 4096 ./buf19.img ./buf19_output/objects`
4. ENTER

BitCurator Disk Image Access Tool

- Standalone GUI
- Used to access the contents of a disk image
- Select and export files



File System:
Entries in bold are directories
Entries in red are unallocated/deleted files

```
▼ Disk Image: /home/bcadmin/Desktop/buf19-test-data/buf19.img
  ▼ Volume 1 (hfs+)
      ☑ $ExtentsFile
      ☑ $CatalogFile
      ☑ $BadBlockFile
      ☑ $AllocationFile
      ☑ caléndula.jpeg
      ☑ Charts2.pptx
      ☑ Club-Report.doc
      ☑ currentacceptance.xlsx
      ☑ getty_museum.txt
      ☑ IMG_4904.JPG
      ☑ Now with data.pptx
      ☑ pastacceptance.xlsx
      ☑ payinfo.pdf
      ☑ Pymnt_inf.txt
      ☑ Thumbs.db
      ^^^^HFS+ Private Data
```

```
>>> No image information found
```

```
>> Exporting file 10 of 15: IMG_4904.JPG
>> Exporting file 11 of 15: Now with data.pptx
>> Exporting file 12 of 15: pastacceptance.xlsx
>> Exporting file 13 of 15: payinfo.pdf
>> Exporting file 14 of 15: Pymnt_inf.txt
>> Exporting file 15 of 15: Thumbs.db

>> Copied checked files to the directory:
/home/bcadmin/Desktop/buf19-test-
data/diskaccess_objects
```

Mounting that disk image

- Anyone have any thoughts now about what wasn't working when we first tried to mount the disk image?
- Do you think mounting a disk image is necessary to understand the contents of that image?

After Lunch:

- File characterization and reporting with Brunnhilde
- Packaging materials for transfer and package validation

Why perform file characterization?

Recordkeeping

- NDSA Levels of Preservation
  - "know your data"
- Reports available in multiple formats

Triage

- Identify formats that need more attention
  - unknown formats
  - formats at risk

End User Access

- Format migration for access copies
- Emulation for access

Why Brunnhilde?

- Runs a signature-based identification tool (Siegfried)
- Creates multiple reports based on Siegfried's output
- Optionally runs additional tools of use in digital preservation
  - virus scanner (ClamAV)
  - file export from disk images (tsk_recover)
  - sensitive data identification (bulk_extractor)
  - file system analysis (fiwalk)

Running Brunnhilde (nutshell)

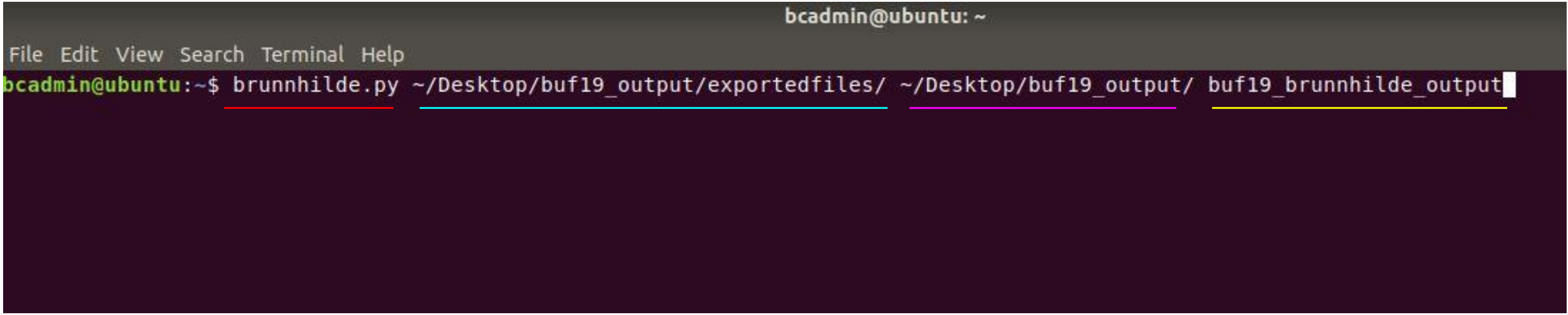
- Brunnhilde requires 3 arguments
  - SOURCE (directory holding the files to characterize)
  - DESTINATION (directory where Brunnhilde should create reports)
  - BASENAME (what should Brunnhilde call the report)

Base command

```
brunnhilde.py /path/to/source/files /path/to/output basename
```

Running Brunnhilde (step-by-step)

1. Open a Terminal window
2. Enter `brunnhilde.py ~/Desktop/buf19_output/objects ~/Desktop/buf19_output buf19_brunnhilde_output`
3. Press Enter



```
bcadmin@ubuntu:~$ brunnhilde.py ~/Desktop/buf19_output/exportedfiles/ ~/Desktop/buf19_output/ buf19_brunnhilde_output
```

Reflection 1

Navigate to `buf19_brunnhilde_output` and open `report.html`

1. What are the sections found in this report?
2. What parts of this report would be useful for description?
3. What parts of this report would be useful for administrative purposes?
4. Navigate to the *Unidentified section*. Why do you think were these files unidentified?
5. Open **`siegfried.csv`** and navigate to the line for **`getty_museum.txt`**. What did Siegfried identify this file as? What does the column for "warning" (Column L) tell you?

Reflection 2

File Identification Tool Collisions (Siegfried vs. fiwalk)

1. In `siegfried.csv` locate the line for `Thumbs.db`
2. Note the format identification and basis for `Thumbs.db`
3. Navigate to the DFXML file we created this morning.
4. Locate the `<fileobject>` element for `Thumbs.db`
5. Note the file format in the `<libmagic>` element

1. Did Siegfried and Libmagic agree in their file identification?
2. Why might each tool have identified the file differently?
3. Which would you trust? Why?
4. If time allows, repeat the steps on the left for `Charts2.pptx`.

Packaging Objects using Bagit

- Consistent method to log, track, and verify materials between platforms
- Wide adoption, originally developed by Library of Congress and California Digital Library
- Content agnostic

Running Bagit (nutshell)

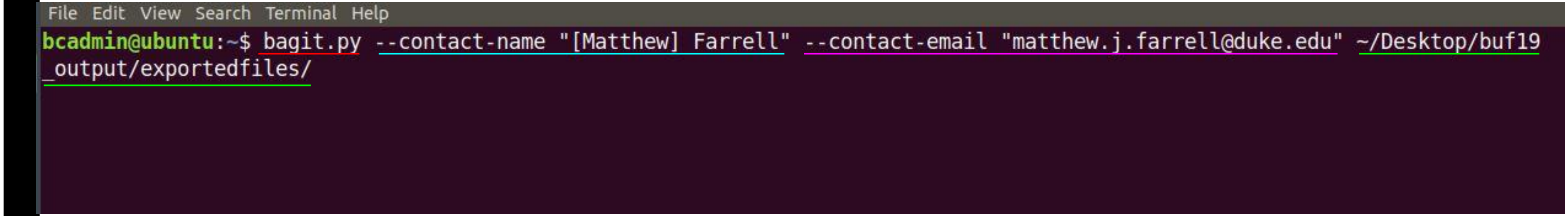
- Bagit requires 1 argument, and accepts optional metadata
    - required: the files to Bag
    - optional: contact name, contact email, checksum algorithm, etc.

Base command

```
bagit.py --contact-name "Your Name" --contact-email
"email@email.com" /path/to/files
```

Running Bagit (step-by-step)

1. Open a Terminal window
2. Enter `bagit.py --contact-name "Your Name" --contact-email "email address" ~/Desktop/buf19_output/objects/`
3. Press Enter



```
bcadmin@ubuntu:~$ bagit.py --contact-name "[Matthew] Farrell" --contact-email "matthew.j.farrell@duke.edu" ~/Desktop/buf19
_output/exportedfiles/
```

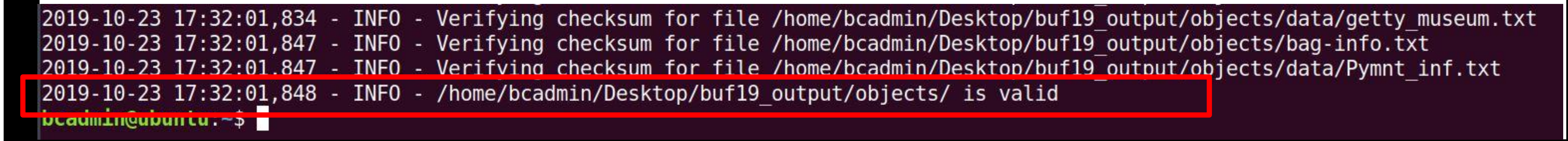
Reflection 1

Navigate to the directory `objects`

1. What did running bagit.py do?
2. Open `bag-info.txt`. What does it tell you?
3. What information do you find in the manifest files?
4. Why are there multiple manifests?
5. How might you use this tool in your own workflows?

Validating bags

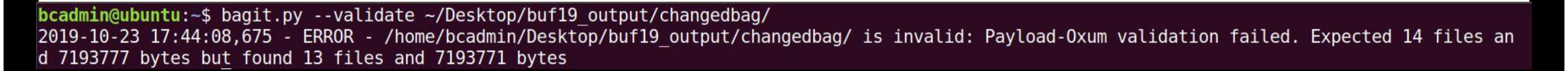
1. Open a Terminal window
2. Enter **`bagit.py --validate ~/Desktop/buf19_output/objects`**
3. Press Enter
4. Did the bag validate?



```
2019-10-23 17:32:01,834 - INFO - Verifying checksum for file /home/bcadmin/Desktop/buf19_output/objects/data/getty_museum.txt
2019-10-23 17:32:01,847 - INFO - Verifying checksum for file /home/bcadmin/Desktop/buf19_output/objects/bag-info.txt
2019-10-23 17:32:01,847 - INFO - Verifying checksum for file /home/bcadmin/Desktop/buf19_output/objects/data/Pymnt_inf.txt
2019-10-23 17:32:01,848 - INFO - /home/bcadmin/Desktop/buf19_output/objects/ is valid
bcadmin@ubuntu:~$ 
```

Invalidating bags

1. Make a copy of `objects` and rename it to `changedbag`
2. Open `changedbag` and delete the `Thumbs.db` file
3. In a Terminal Window, validate the bag `bagit.py --validate ~/Desktop/buf19_output/changedbag`
4. What does Bagit.py report?



```
bcadmin@ubuntu:~$ bagit.py --validate ~/Desktop/buf19_output/changedbag/
2019-10-23 17:44:08,675 - ERROR - /home/bcadmin/Desktop/buf19_output/changedbag/ is invalid: Payload-Oxum validation failed. Expected 14 files an
d 7193777 bytes but found 13 files and 7193771 bytes
```

Reflection 1

Bags, checksums and workflows

1. Why is the bag invalid?
2. How would you manage a hidden or unintentionally transferred file in your own institutional workflows?
3. In what cases would you want to use Bagit? Can you think of cases where you may want to use a different validation method?

Resources

- BitCurator Consortium
  - Links to the distribution git and quick start guide
  - Consortium member sample documentation and workflows

- BitCurator User Google Group
  - Helpful, friendly community who is here to help troubleshooting BitCurator, disk imaging, and other things digital forensics!

Thank you!

- Amy Berish
  - @amy_berish
- Dianne Dietrich
  - @smallandmath
- Laura Alagna
  - @Digitized_Laura
- Martin Gengenbach
  - @mjgengenbach
- [Matthew] Farrell
  - @laissezfarrell