ECE566
Enterprise Storage Architecture

Spring 2025

Data recovery and forensics

Tyler Bletsch
Duke University

The problem

- **Recovery**: Restoring data without backups after loss
  - **Goal: get our data back**
  - Challenges:
    - Damaged media
    - Software fault caused corruption
    - Accidental deletion
  - *NOTE: This implies your backups failed, which implies that you're bad!*
- **Forensics**: Recovering data in legal scenario
  - **Goal: get <u>their</u> data out**
  - Challenges:
    - <u>Intentional</u> deletion
    - Finding information <u>leaked</u> to disk
  - Note: we'll just focus on persistent storage file recovery, not legal evidence rules, event reconstruction, compromised system analysis, authorship analysis, etc.

Formal definition

- Given: data storage device(s) in unknown state
  - Device may be damaged, filesystem may be corrupt, files may be deleted
- Goal: Want to recover either specific files or do general exploration
- Key insight: Can go <u>beneath</u> the filesystem abstraction

Preparation of disk image

1. Shut down device
2. Document hardware configuration
3. Put disk(s) into a separate system with an operating system that won't attempt to auto-mount or scan them (i.e., not Windows/Mac)
4. Image the disks to a file on a separate storage device
   - "Image" means reading from the raw disk into a file, also known as "ghosting" the drive
   - If the disks are damaged/failing, use an error-tolerant tool like `ddrescue` to ignore read errors efficiently
5. (Forensic only) Document the disk image hash, date, time, drive serial numbers, etc.
6. Analyze disk image to recover data;
   don't touch the original disk(s) again!

Adapted from "Computer Forensics" presentation, Bassel Kateeb and Tim Altimus, U. Pittsburgh.

Using loop devices

- We need to treat the disk image files like a block device

- There's a facility just for this in Linux: the **loop device**
  - The kernel makes a device `/dev/loopN` (where N is an integer) into a block device where each request is satisfied by a read of a regular file in a mounted filesystem
  - Configured manually by `losetup` or automatically when mounting an image with the `-o loop` option.
  - Can deal with partitioned block devices by turning on partition scan (`losetup` with `-P` option)

Dealing with RAID/LVM

- If there are multiple disks in a RAID or LVM, use the on-disk metadata to identify the configuration. Examples:
  - If it's a Linux 'md' software RAID, a superblock will describe the configuration, and you can use the 'md' driver on the loop devices
  - If it's a hardware RAID0/4/5/6, you can write a simple program to merge them together into a single image
  - If it's LVM that joins two block devices in sequence, just concatenate them into a new image or use the actual LVM support of the OS

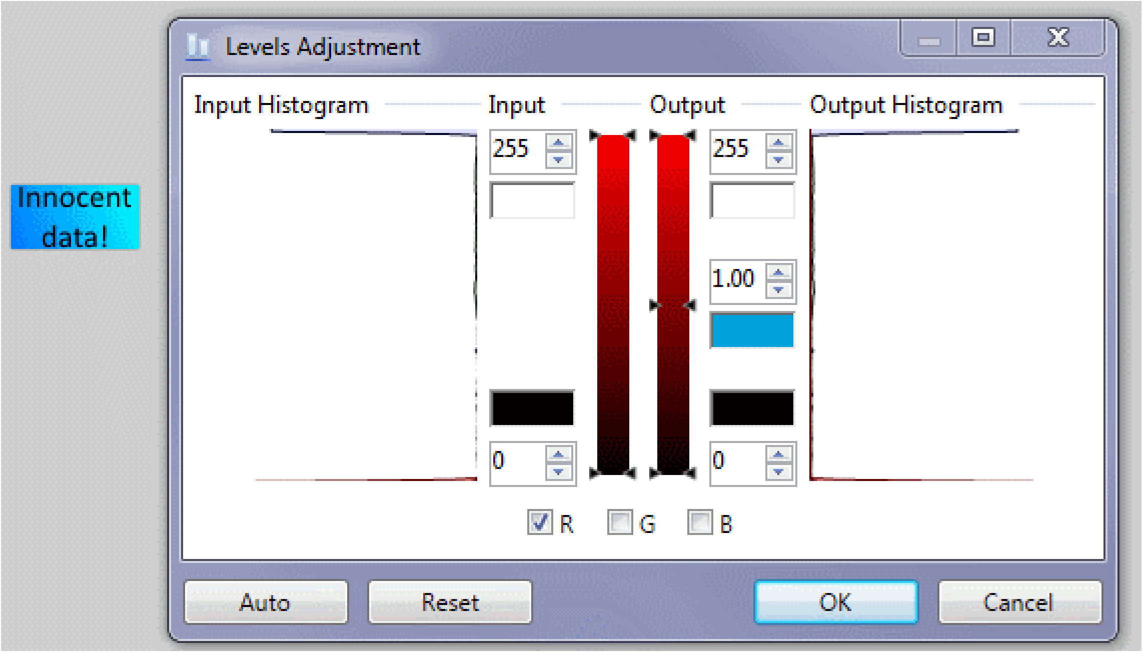
Where to find data

- **Existing Files**:
  - The normal files you expect
  - Logs
  - Temporary files and caches (including web and email caches and 'thumbs.db'!)
  - Special system files (registry, crontabs, printer spool, etc.)
  - Swap files (a bunch of random RAM! Could be useful!)
- **Deleted Files**:
  - Delete is implemented as a tiny metadata operation:
mark file deleted, mark space as available
  - Data stays around until a new allocation makes use of that region
  - Extreme case: FAT filesystem
    - Deleted files just have the first letter of filename changed to '?'
- **Filesystem journal**:
  - May contain just metadata or possibly also data
  - Totally unconnected to traditional filesystem structures,
invisible to user code

Adapted from "COEN 252 Computer Forensics: Data Analysis Techniques for Hard Drives" by Thomas Schwarz, Santa Clara Univ. and "Computer Forensics" presentation, Bassel Kateeb and Tim Altimus, U. Pittsburgh.

Where to hide data

- In forensics, what if they're hiding data intentionally?
- **Steganography**: The art of storing information in such a way that the existence of the information is hidden[1].
- Common technique: Low order bits of media data



[1] From "Computer Forensics" presentation, Bassel Kateeb and Tim Altimus, U. Pittsburgh.

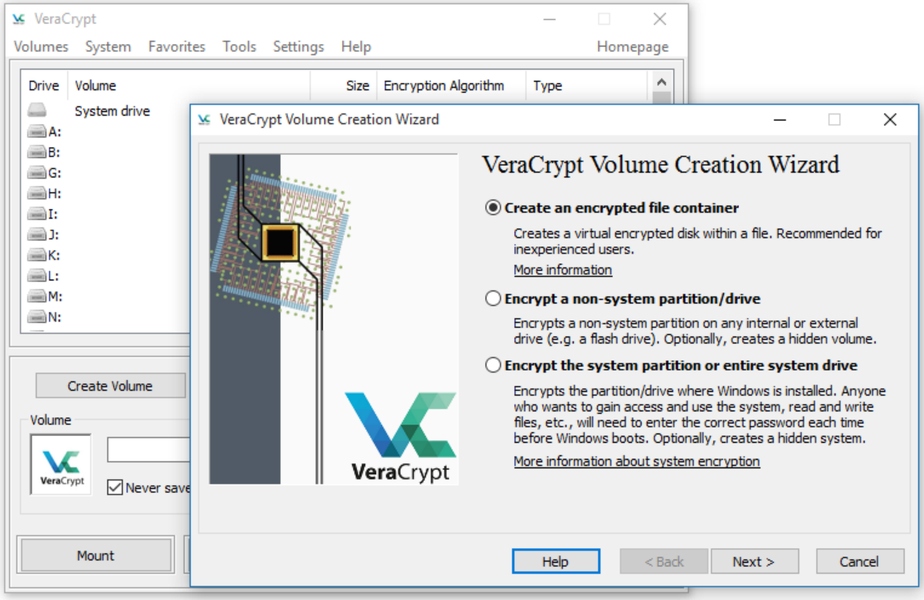
Places on a disk to hide stuff

- **Regular hiding**: Put stuff in system directories named "kernel.dat" instead of "secret.docx", etc.
- **Disk block slack**: Area between logical end of file and the end of the last disk block
  - May contain data from before block allocation (disk remnant) or unrelated data in RAM buffer used to flush data to disk (RAM remnant)
- **Partition wasted space**: A bunch of space is left unused after boot sector for historical reasons; may also have other wasted partition space (e.g. if partition isn't a multiple of the file system block size)
- **Inter-partition space**: Can have disk regions not in *any* disk partition (e.g. partition 1 is blocks 0-1000 and partition 2 is 1050-2050).
- **Bad sectors**: Can have file systems mark certain disk blocks as bad, then you can write to them yourself directly in secret.

Adapted from "Computer Forensics" presentation, Bassel Kateeb and Tim Altimus, U. Pittsburgh.

Hiding stuff with cryptography

- **Disk encryption software**: If it's encrypted right, you're not getting it.



- **Deniable volumes**: Some encryption software (e.g. VeraCrypt) allows you to have two logical containers, a "real" one presented when you give one key, and a "fake" one given when you provide a different key.
  - Rationale: If under duress, you can give up your fake key.
  - Encrypted content just looks like random data, can't prove that there is a "real" key or additional data
- **We can't do much about the above.**

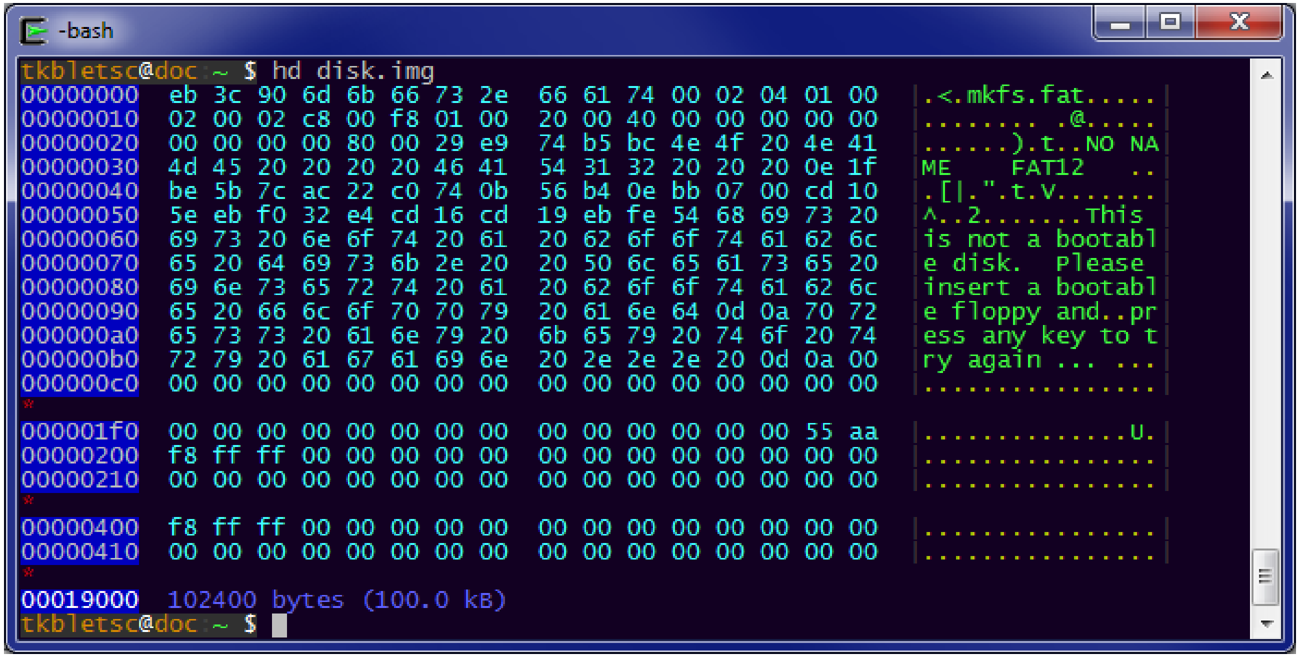
Basic recovery

- If there is little or no damage/corruption/deletion, you can just mount the image and retrieve the desired content

- Practices in dealing with data recovery (esp. in forensics):
  - Always mount images read-only: you never want to modify the source!
  - Never execute any code from suspect sources.
  - If you have to transport possible malware, put it in a passworded archive.
  - Preserve metadata where appropriate (e.g. `cp -a`).

Fundamental binary analysis tool: Hex dump/edit

- Otherwise, need to use forensic recovery tools/techniques
  - Most fundamental tool: **hex editor** (or just **hex dumper**)



  - All other tools just automate what could be done by you manually using a hex dump!

Forensic recovery approaches

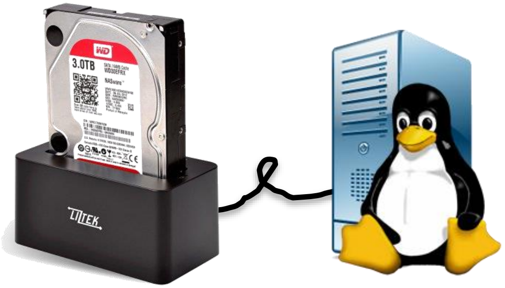
Forensic data recovery has two main approaches:

- **Top-down**: Use filesystem to guide search
  - Looking for a deleted file? Find its surviving metadata, use that to get to the surviving data
  - Example: SleuthKit, a set of command-line tools which implement filesystem logic in usermode programs. Includes:
    - `fsstat`: Identify filesystem type and configuration
    - `fls`: List files in an image (including ones deleted but still present!)
    - `icat`: Dump a file from an image with just its inode number (or similar numeric identifier, depending on filesystem)
    - `jcat`: Dump data from the filesystem journal
- **Bottom-up**: Mostly ignore filesystem, scan for keywords and/or well-known file types (jpeg, png, docx, etc.).
  - `strings`: Print out all human-readable data from binary file
  - `photorec`, `testdisk`, `foremost`: Seek out common file types (jpeg, png, docx, etc.).

Practical data recovery example #1

Scenario: Dying hard drive, no backups available (shame on you!)

1. Pull drive, slap into USB drive dock, hook to Linux box



2. Read drive using ddrescue (ignores errors quickly)

```
ddrescue /dev/sda disk.img
```

3. Attach disk image to loop device (including partition support)

```
losetup -P /dev/loop0 disk.img
```

4. Mount main partition to subdirectory Q

```
mount -o ro /dev/loop0p1 Q
```

5. Get data out of Q

Practical data recovery example #2

Scenario: You have some disk images to retrieve various secrets from

You're gonna figure this one out in the lab!

Brief intro: Forensic event reconstruction

- Lots of places store time information:
  - Application and kernel logs
  - File system metadata
  - File system journal
  - Communication systems (email, chat, etc.)
  - Web browser history
  - Document files (.docx, .pdf, etc.) and versioning systems (e.g. `git`)
  - File deletion managers (e.g. ‘Recycle bin’, ‘Trash can’, etc.)
- Each stores time in its own format
  (possibly with timezone issues)

- **Forensic event reconstruction:**
  - Create a unified timeline from all these sources

Brief intro: System security assessment

- Looking for evidence of actions undertaken by attacker during a breach
- First, do all of the previously discussed stuff
- Then:
  - Compare OS file hashes against known good hashes
  - Look for known malware by signature
  - Content from around the time and/or user account of known breach activity (can use event reconstruction here!)
  - Look for back doors. Examples:
    - UNIX: Setuid files (executables can confer owner's permission)
    - Windows: Daemons started via a variety of Windows on-boot hooks
    - Either: Kernel rootkits (hide stuff from userspace)
    - Either: User accounts added/enabled
- No matter what, don't return this system image to production!
  - Can never *prove* you found *all* the backdoors/malware
  - Instead, restore from sources or known good backup
  - *Separate issue: finding and fixing root cause of breach!*