Working with Computer Forensics Data
July 20, 2011 (SOUPS 2011 Tutorial)

Simson L. Garfinkel, Ph.D
Associate Professor, Naval Postgraduate School
http://www.simson.net/

NPS is the Navy's Research University.

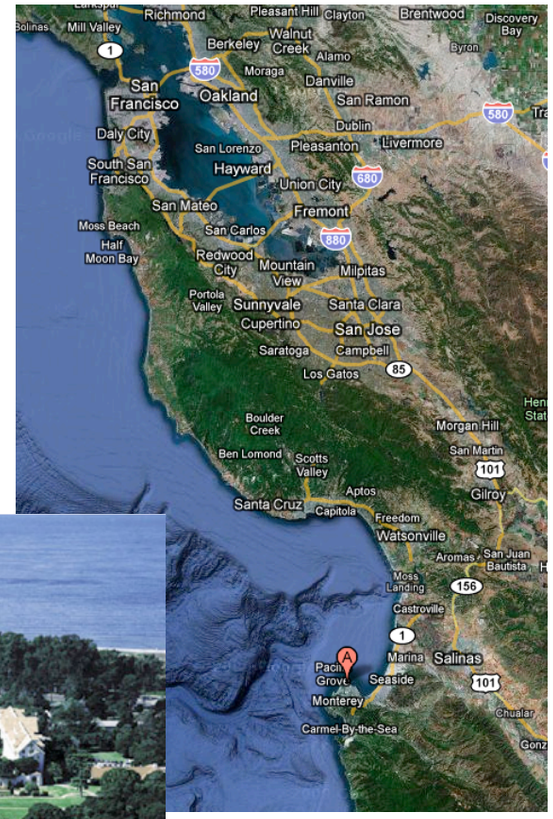
Location: Monterey, CA

Students: 1500

- US Military (All 5 services)
- US Civilian (Scholarship for Service & SMART)
- Foreign Military (30 countries)
- All students are fully funded

Schools:

- Business & Public Policy
- Engineering & Applied Sciences
- Operational & Information Sciences
- International Graduate Studies



NCR Initiative:

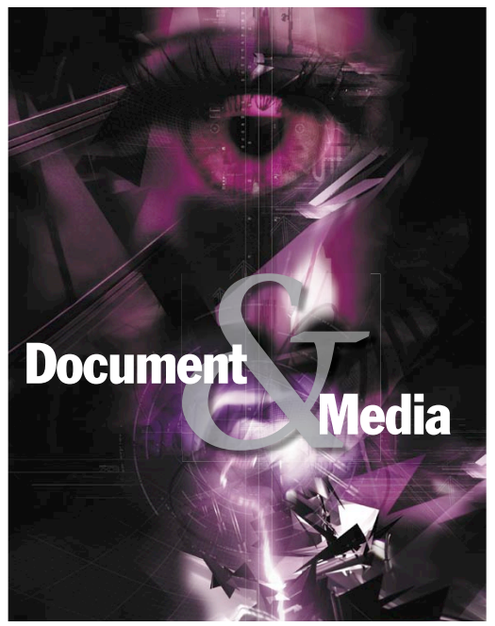
- 8 offices on 5th floor, 900N Glebe Road, Arlington
- FY12 plans: 4 professors, 2 postdocs, 2 researchers
- **Immediate slots for .gov/.mil PhDs!**

My current research: Automated Document & Media Exploitation

The DOMEX challenge is to turn digital bits into actionable intelligence.

Recent publications:

- DoD Risks from Facebook -
  —*http://simson.net/clips/academic/2011.CrossTalk.Facebook.pdf*
- Forensic Carving of Network Packets and Associated Data Structures
  —*DFRWS 2011 (August 2011)*
- Digital Forensics Research: The next 10 years
  —*http://simson.net/clips/academic/2010.DFRWS.Next10Years.pdf*



A computer used by Al Qaeda ends up in the hands of a *Wall Street Journal* reporter. A laptop from Iran is discovered that contains details of that country's nuclear weapons program. Photographs and videos are downloaded from terrorist Web sites.

As evidenced by these and countless other cases, digital documents and storage devices hold the key to many ongoing military and criminal investigations. The most straightforward approach to using these media and documents is to explore them with ordinary tools—open the word files with Microsoft Word, view the Web pages with Internet Explorer, and so on.

Although this straightforward approach is easy to understand, it can miss a lot. Deleted and invisible files can be made visible using basic forensic tools. Programs called *carvers* can locate information that isn't even a complete file and turn it into a form that can be readily processed. Detailed examination of e-mail headers and log files can reveal where a computer was used and other computers with which it came into contact. Linguistic tools can discover multiple documents that refer to the same individuals, even though names in the different documents have different spellings and are in different human languages. Data-mining techniques such as cross-drive analysis can reconstruct social networks—automatically determining, for example, if the computer's previous user was in contact with known terrorists. This sort of advanced analysis is the stuff of DOMEX, the little-known intelligence practice of document and media exploitation.

The U.S. intelligence community defines DOMEX as "the processing, translation, analysis, and dissemination

Exploitation

SIMSON L. GARFINKEL, PH.D.

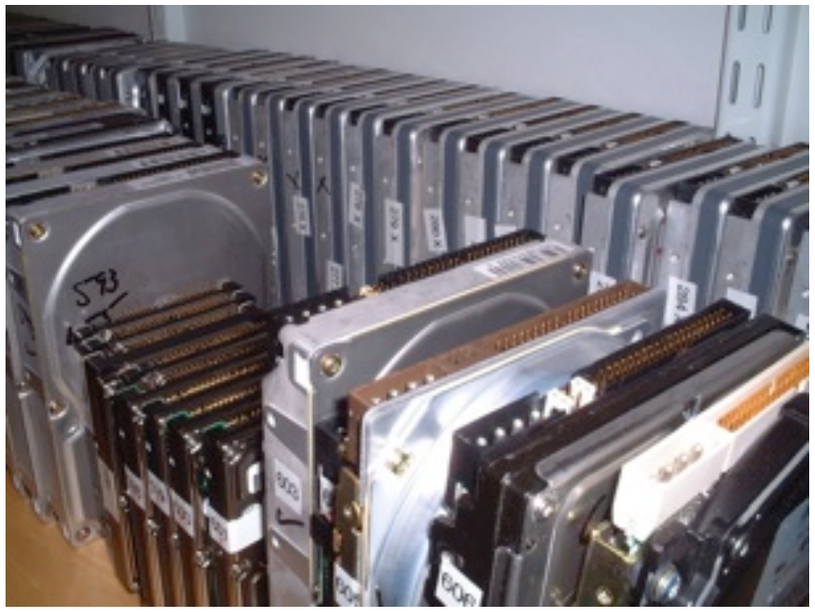
The DOMEX challenge is to turn digital bits into actionable intelligence.

http://www.simson.net/clips/academic/2007.ACM.Domex.pdf

Current NPS research thrusts in digital forensics

Area #1: End-to-end automation of forensic processing

- Digital Forensics XML Toolkit
- Disk Image -> Power Point



Area #2: Bulk data analysis

- Statistical techniques (sub-linear algorithms)
- Similarity metrics
- Sector hashing

Area #3: Data mining for digital forensics

- Automated social network analysis
- Cross-drive analysis

Area #4: Creating standardized forensic corpora

- Freely redistributable disk and memory images, packet dumps, file collections.

You will find additional information on the Forenscs Wiki.

http://www.forensicswiki.org/

Log in / create account

page | discussion | view source | history

Main Page

This is the **Forensics Wiki**, a Creative Commons -licensed wiki devoted to information about digital forensics (also known as computer forensics). We currently list a total of 498 pages.

Much of computer forensics is focused on the tools and techniques used by investigators, but there are also a number of important papers, people, and organizations involved. Many of those organizations sponsor conferences throughout the year and around the world. You may also wish to examine the popular journals and some special reports.

navigation:
- Main Page
- Categories

about forensicswiki.org:
- Recent changes
- Random page
- Donations

search
Go Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Selected Forensics Research

2008-Aug-13

Lest We Remember: Cold Boot Attacks on Encryption Keys

J. Alex Halderman, Princeton University; Seth D. Schoen, Electronic Frontier Foundation; Nadia Heninger and William Clarkson, Princeton University; William Paul, Wind River Systems; Joseph A. Calandrino and Ariel J. Feldman, Princeton University; Jacob Appelbaum; Edward W. Felten, Princeton University

USENIX Security '08 Refereed Paper

Awarded Best Student Paper

Increasingly memory analysis is of interest in forensic research---both because new malware only resides in memory, and because memory analysis is frequently the only way for analysts to get the keys that are used to protect cryptographic file systems. In this paper the authors show that cryptographic keys in memory are vulnerable to exploitation *after the computer is turned off.* The authors show that the contents of dynamic RAM are retained seconds, and sometimes

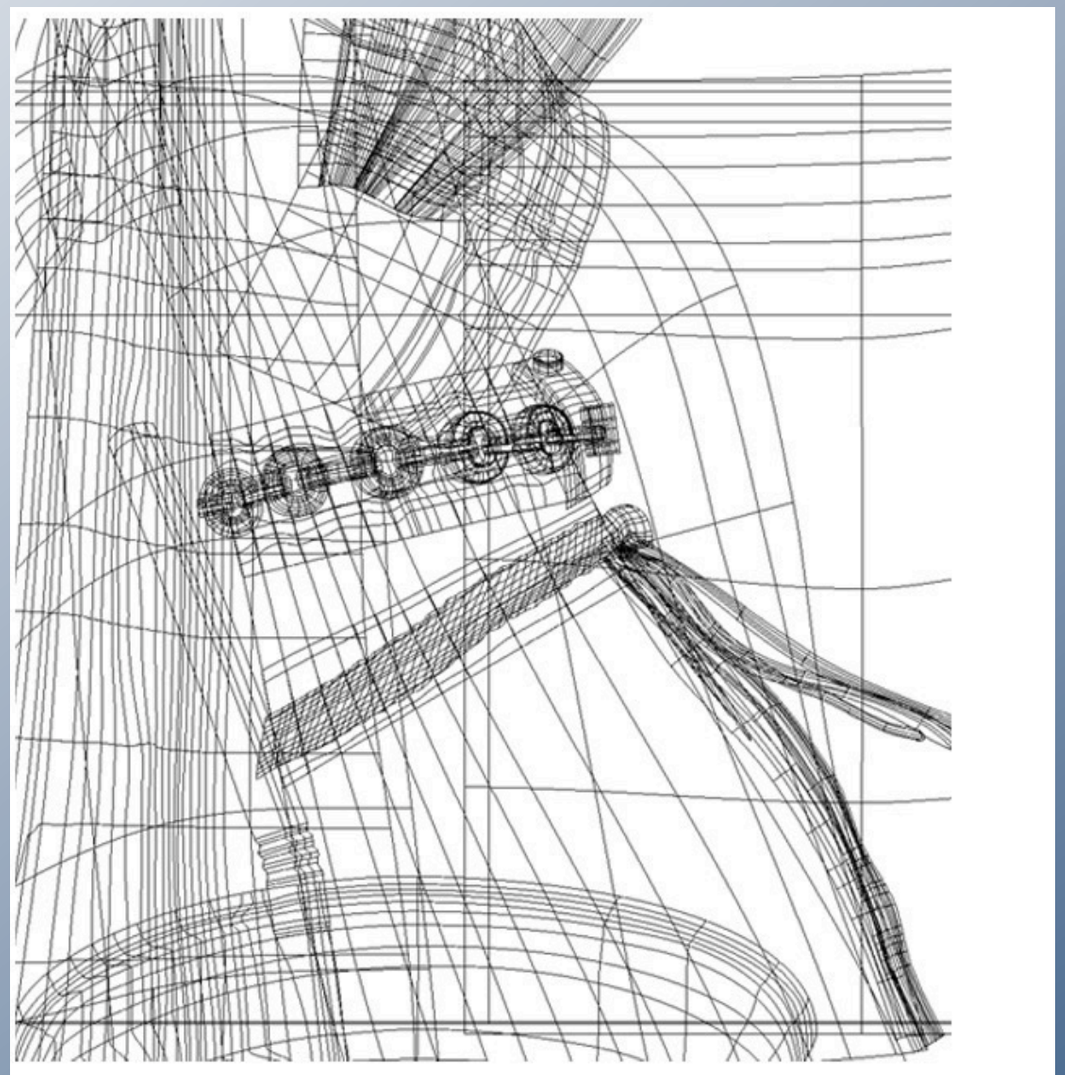
This tutorial introduces forensics and forensics data for security & usability practitioners.

Forensics is used to understand a system's past or current state.

- What did the person do with the computer?
- What did the user see?
- How was the computer hacked?

Working with forensics data is *hard*:

- Forensics data is typically raw, inconsistent, fragmented, and sometimes corrupt.
- Data sets are large
  - —*typically generated by last year's top-tier computers and networks...*
  - —*... analyzed with this year's top-tier computers and networks.*
- Forensics data is highly variable.
  - —*Many file systems; applications; etc.*
- Forensics problems span the abstraction stack.
  - —*machine code, HTML, JavaScript, Authentication, Naming, Storage, Networks, etc.*



Introducing digital forensics & investigations

Forensic definitions
The “magic camera”
Hypothesis-based investigation

Today “forensics” means the application of scientific methods to investigations.

forensic |fəˈrenzik; -sik|

- adjective
- of, relating to, or denoting the application of scientific methods and techniques to the investigation of crime : forensic evidence.
- of or relating to courts of law.

noun (forensics)

- scientific tests or techniques used in connection with the detection of crime.
- (also forensic) [treated as sing. or pl.] informal a laboratory or department responsible for such tests.

—*ORIGIN mid 17th cent.: from Latin forensis ‘in open court, public,’ from forum (see forum).*

There are *many* kinds of investigations that might involve forensics:

- Criminal — a murder.
- Civil — a lawsuit between companies.
- Internal corporate — employee termination.
- Computer crime — understanding how a computer was hacked.

Investigations need to be done by forensic examiners.

Forensic evidence is critical to many civil and criminal cases:

- Fingerprints & DNA
- Photograph of a crime scene
- SMS messages

But judges and juries can’t collect and examine physical evidence:

- They don't have the time.
- They don’t have the training.

Evidence may be open to interpretation.

US courts employ an adversarial process.

- Prosecution (or plaintiff) experts look for *evidence of wrongdoing.*
- Defense experts *refute the interpretation*
  - —*Evidence is not relevant (e.g. from a different crime)*
  - —*Evidence was improperly collected (contaminated)*
  - —*Evidence was misinterpreted (error in training or technique)*

In some cases, the Court may hire its own expert.

Broadly speaking, evidence is either *physical* or *digital*.

Physical evidence is based on physical objects.

- Blood & DNA
- Bullets, guns and ballistics
- Tire tread marks.

Digital evidence is evidence that has some kind of connection to computers.

There are many definitions for digital evidence:

- “Information stored or transmitted in binary form ... relied upon in court.” [Int02]
- “Information of probative value ... stored or transmitted in binary form.” [Sci05]
- “[D]ata of investigative value ... stored ... or transmitted by a computer.” [Ass05]
- “[D]ata ... that support or refute a theory of how an offense occurred or that address critical elements ... such as intent or alibi.” [Cas04]

Example: Digital files that show evidence of a physical crime.

- JPEGs showing child exploitation.
- Excel files tracking drug sales.
- Emails documenting a conspiracy

File analysis should be done by a forensic expert.

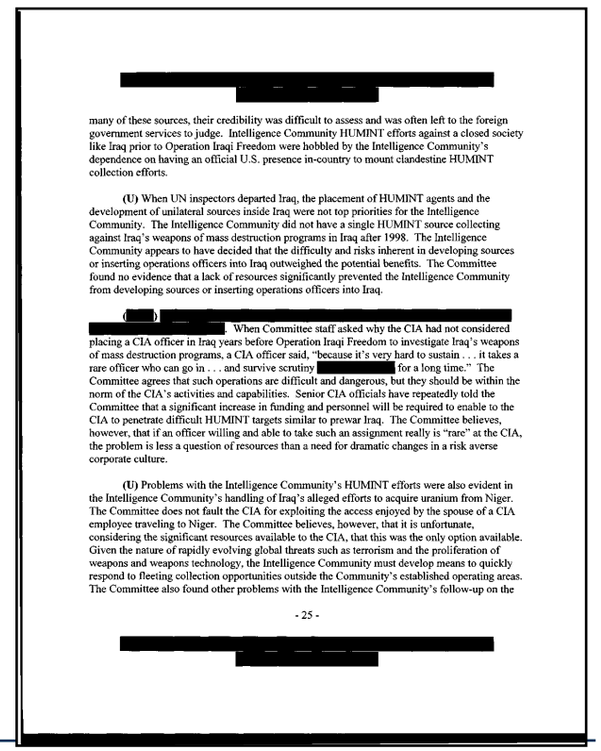
The forensic expert might:

- *Authenticate* the file.
  - —That it came from the subject's device.
  - —That it is a true and accurate copy.

- *Examine* the file.
  - —Note the file's *overt* file contents.
  - —Look for hidden data within the file.
  - —Determine if the data was created through normal processes or modified through some kind of extraordinary process (e.g. a hex editor).

- Prepare a report.

- Testify in court.



many of these sources, their credibility was difficult to assess and was often left to the foreign government services to judge. Intelligence Community HUMINT efforts against a closed society like Iraq prior to Operation Iraqi Freedom were hobbled by the Intelligence Community's dependence on having an official U.S. presence in-country to mount clandestine HUMINT collection efforts.

(U) When UN inspectors departed Iraq, the placement of HUMINT agents and the development of unilateral sources inside Iraq were not top priorities for the Intelligence Community. The Intelligence Community did not have a single HUMINT source collecting against Iraq's weapons of mass destruction programs in Iraq after 1998. The Intelligence Community appears to have decided that the difficulty and risks inherent in developing sources or inserting operations officers into Iraq outweighed the potential benefits. The Committee found no evidence that a lack of resources significantly prevented the Intelligence Community from developing sources or inserting operations officers into Iraq.

([redacted]) [redacted]. When Committee staff asked why the CIA had not considered placing a CIA officer in Iraq years before Operation Iraqi Freedom to investigate Iraq's weapons of mass destruction programs, a CIA officer said, "because it's very hard to sustain . . . it takes a rare officer who can go in . . . and survive scrutiny [redacted] for a long time." The Committee agrees that such operations are difficult and dangerous, but they should be within the norm of the CIA's activities and capabilities. Senior CIA officials have repeatedly told the Committee that a significant increase in funding and personnel will be required to enable to the CIA to penetrate difficult HUMINT targets similar to prewar Iraq. The Committee believes, however, that if an officer willing and able to take such an assignment really is "rare" at the CIA, the problem is less a question of resources than a need for dramatic changes in a risk averse corporate culture.

(U) Problems with the Intelligence Community's HUMINT efforts were also evident in the Intelligence Community's handling of Iraq's alleged efforts to acquire uranium from Niger. The Committee does not fault the CIA for exploiting the access enjoyed by the spouse of a CIA employee traveling to Niger. The Committee believes, however, that it is unfortunate, considering the significant resources available to the CIA, that this was the only option available. Given the nature of rapidly evolving global threats such as terrorism and the proliferation of weapons and weapons technology, the Intelligence Community must develop means to quickly respond to fleeting collection opportunities outside the Community's established operating areas. The Committee also found other problems with the Intelligence Community's follow-up on the

- 25 -

[redacted]

Even photographs may require interpretation

When were these photographs taken?

Were they faked?

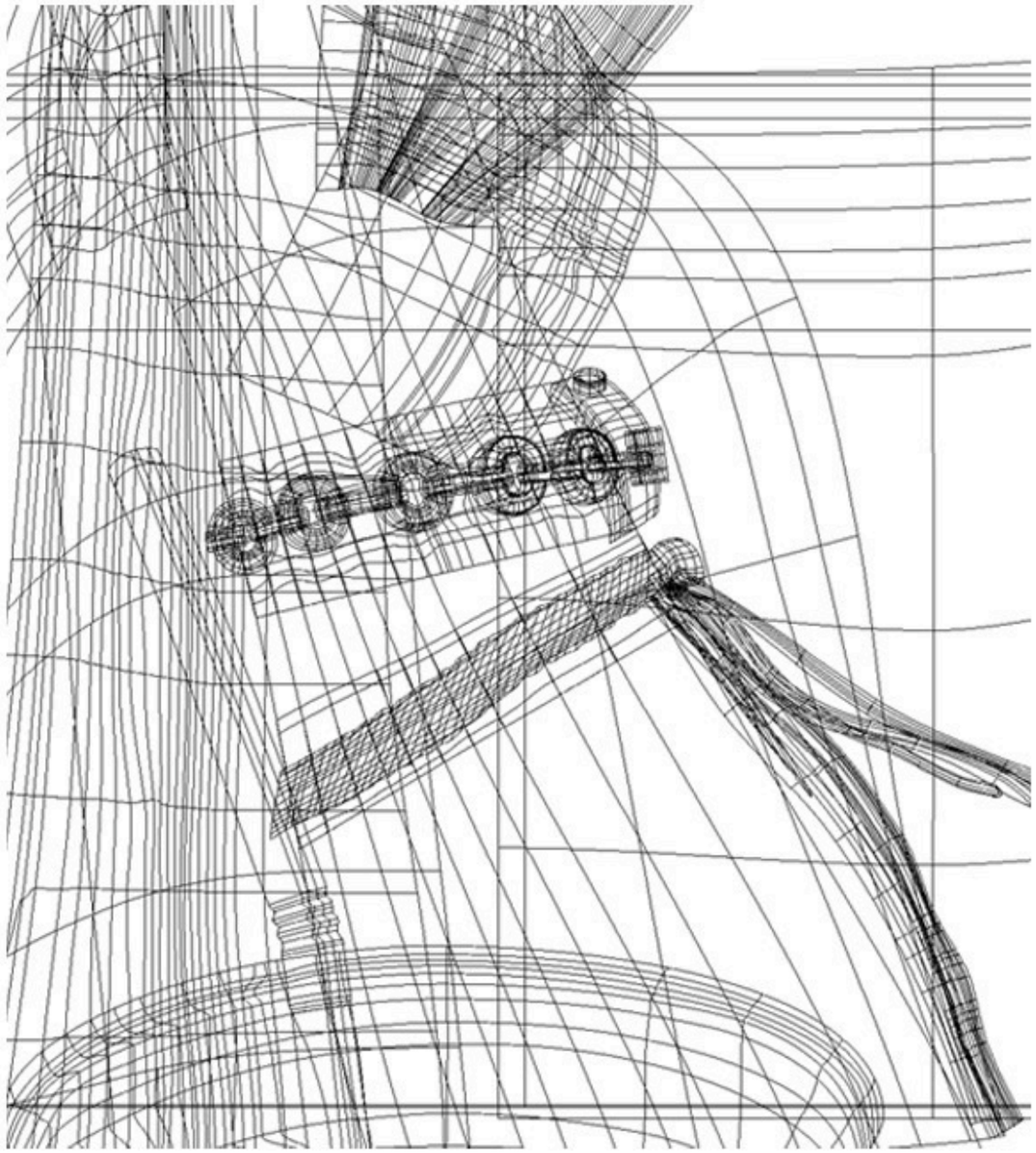
Stalin's darkroom tampered with the past.

Abel Yenukidze:

- Shot during the purges of 1936-1938
- Photo removed from official photographs by Stalin’s darkroom

- “The Commissar Vanishes: The Falsification of Photographs in Stalin’s Russia.”
  —*http://www.newseum.org/berlinwall/commissar_vanishes/*
  —*http://www.hoover.org/publications/digest/3531641.html*

Computer graphics are so realistic...
... that it's easy to mistake a simulated photo for reality.



Pisan Kaewma 2006

—Can Digital Photos Be Trusted, Steve Casimiro, 9/11/2005, popsci.com

—Seeing is Not Believing, Steve Casimiro, Popular Science, Oct. 2005,

Traditional forensics is dominated by the Locard Exchange Principle

Dr. Edmund Locard (1877-1966) - "Every contact leaves a trace."

- “**Wherever he steps**, whatever he touches, whatever he leaves, even unconsciously, **will serve as a silent witness against him**.

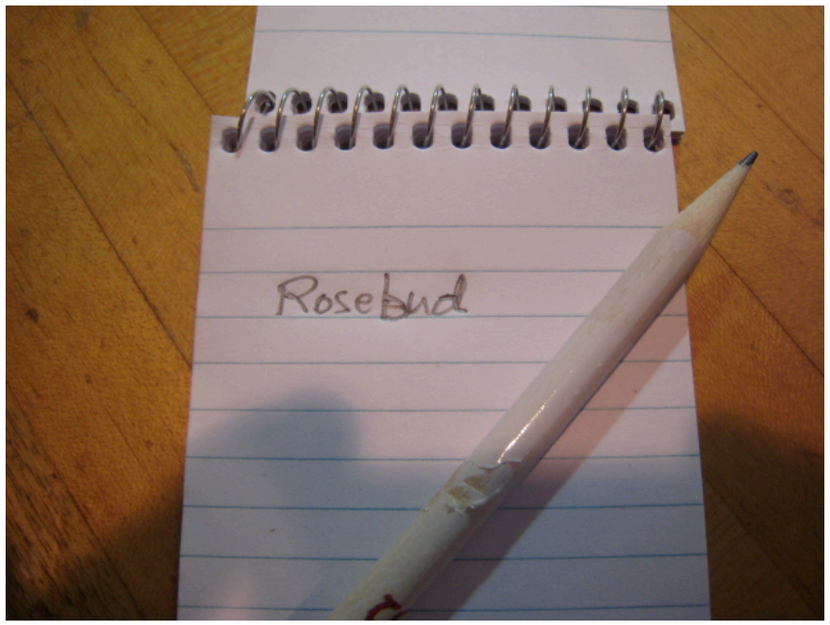
Not only his fingerprints or his footprints, but his hair, the fibers from his clothes, the glass he breaks, the tool mark he leaves, the paint he scratches, the blood or semen he deposits or collects.

All of these and more, bear mute witness against him. This is evidence that does not forget. It is not confused by the excitement of the moment.
It is not absent because human witnesses are. It is factual evidence.

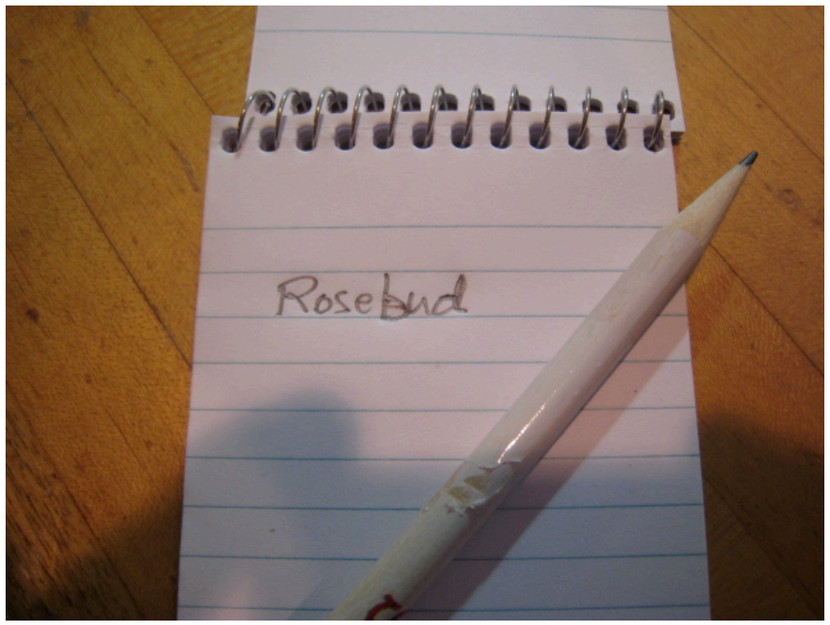
Physical evidence cannot be wrong, it cannot perjure itself, it cannot be wholly absent. Only human failure to find it, study and understand it, can diminish its value.

Every contact leaves a trace...

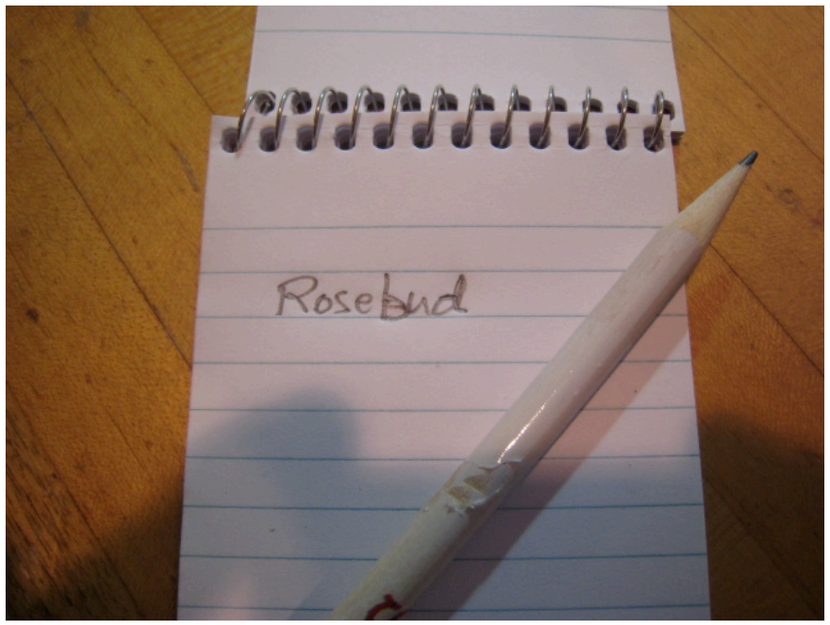
Every contact leaves a trace...



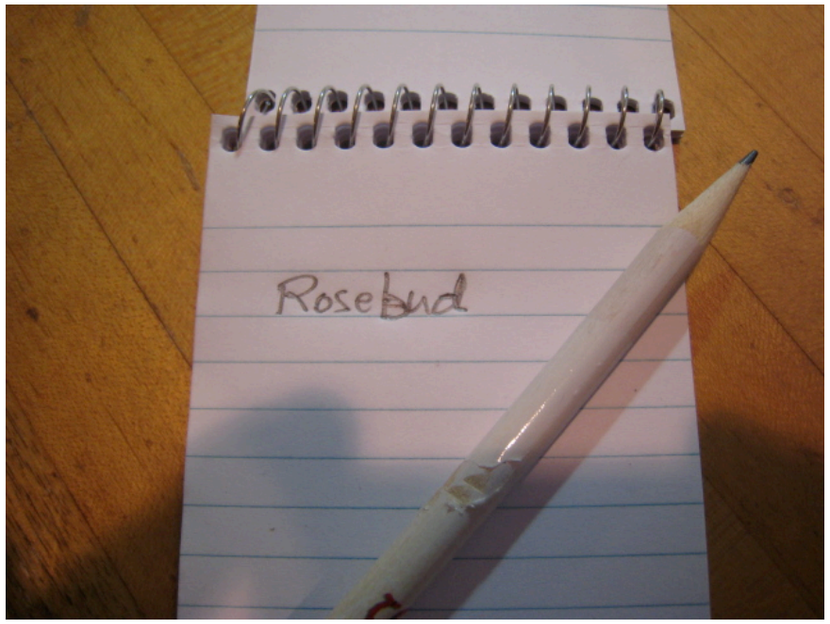
Every contact leaves a trace...



Every contact leaves a trace...



Every contact leaves a trace...



Digital forensics applies these principles to computers.

Some definitions for computer forensics/digital forensics:

- "Involves the preservation, identification, extraction, documentation, and interpretation of computer data."

  —*(Computer Forensics: Incident Response Essentials, Warren Kruse and Jay Heiser.)*

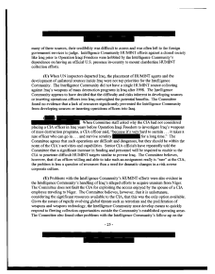
- "The scientific examination, analysis, and/or evaluation of digital evidence in legal matters."

  —*(Scientific Working Group on Digital Evidence, http://www.swgde.org)*

Digital evidence is:

- "Information stored or transmitted in binary form ... relied upon in court." [Int02]
- "Information of probative value ... stored or transmitted in binary form." [Sci05]
- "[D]ata of investigative value ... stored ... or transmitted by a computer." [Ass05]
- "[D]ata ... that support or refute a theory of how an offense occurred or that address critical elements ... such as intent or alibi." [Cas04]

If it involves computers, it's probably digital evidence



Most forensics relies on the analysis of *residual data* or *non-obvious data*.

Residual data:

- information left on a computer after processing is finished.
- Examples:

  —*deleted files — unlink() doesn't overwrite sectors when a file is deleted.*

  —*memory and swap files — free() doesn't overwrite memory no longer used.*

Non-obvious data:

- Web cache files
- System Log files
- Router log files.

These data sources are useful because:

- Most data is not encrypted
- The subject of the investigation is not aware of them.

Digital Forensics lets investigators go back in time...

A magic camera that can:

- View previous versions of files
- Recover “deleted” files
- Find out what was typed
- Report websites visited in the past

For example, The Sleuth Kit (TSK) can view and recover deleted files that have not been overwritten:

```
$ fls -o 51  nps-2009-canon2-gen6.raw 517
r/r 1029:   IMG_0044.JPG
r/r 1030:   IMG_0042.JPG
r/r 1031:   IMG_0003.JPG
...
r/r 1052:   IMG_0024.JPG
r/r * 1053: _MG_0025.JPG
r/r 1054:   IMG_0026.JPG
...
r/r 1057:   IMG_0029.JPG
r/r * 1058: _MG_0030.JPG
r/r 1059:   IMG_0031.JPG
```

But digital evidence is easily faked!

It is relatively easy to create fake evidence:

- Photoshop!
- Log in with some else's username and password.
- Run an attack through an open proxy.

Most data are not "doctored."

- But most data are not taken into court

If the interpretation is high-stakes...

- ... then *someone* has an interest in an incorrect interpretation.

This is true of all evidence...

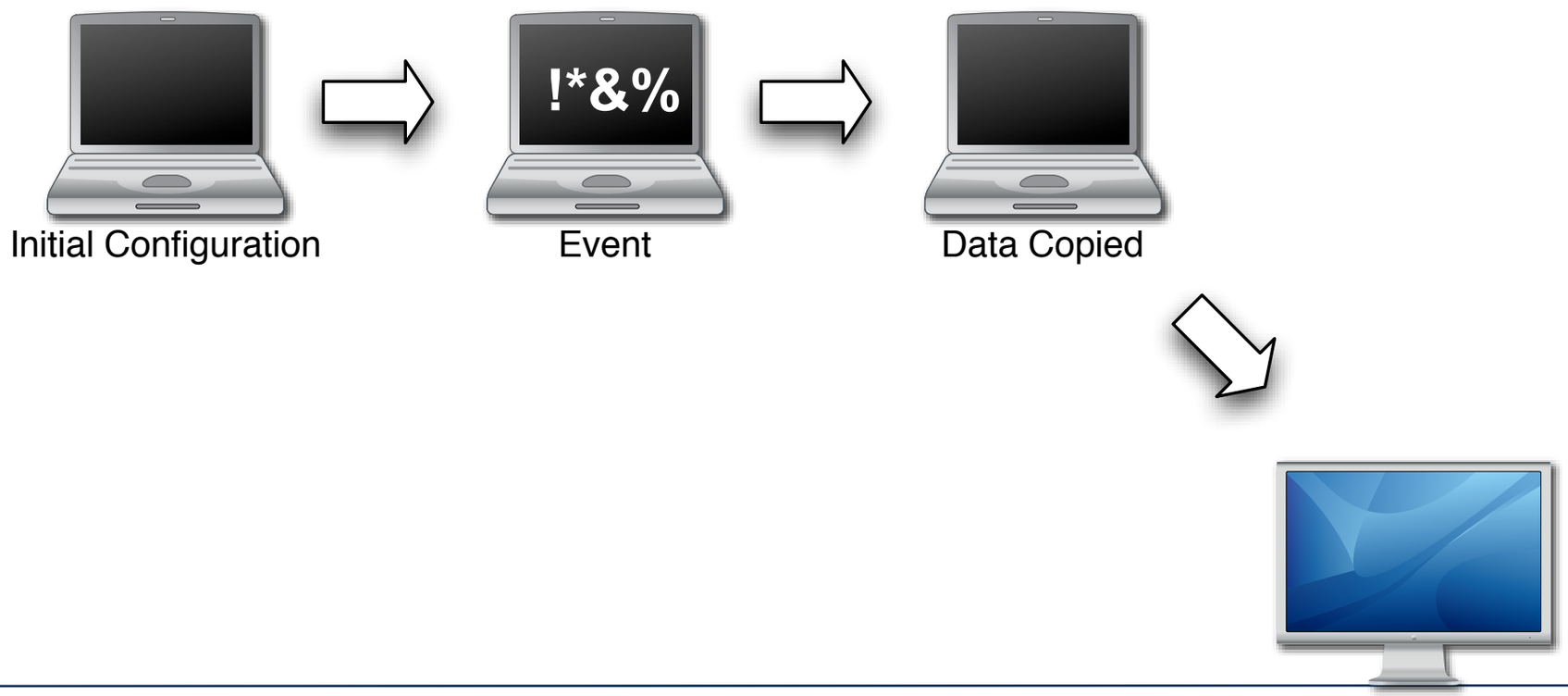
- It's especially easy to doctor digital data, because the tools are widely available.

When we look at a computer system, we build a *hypothesis* about the computer's past.

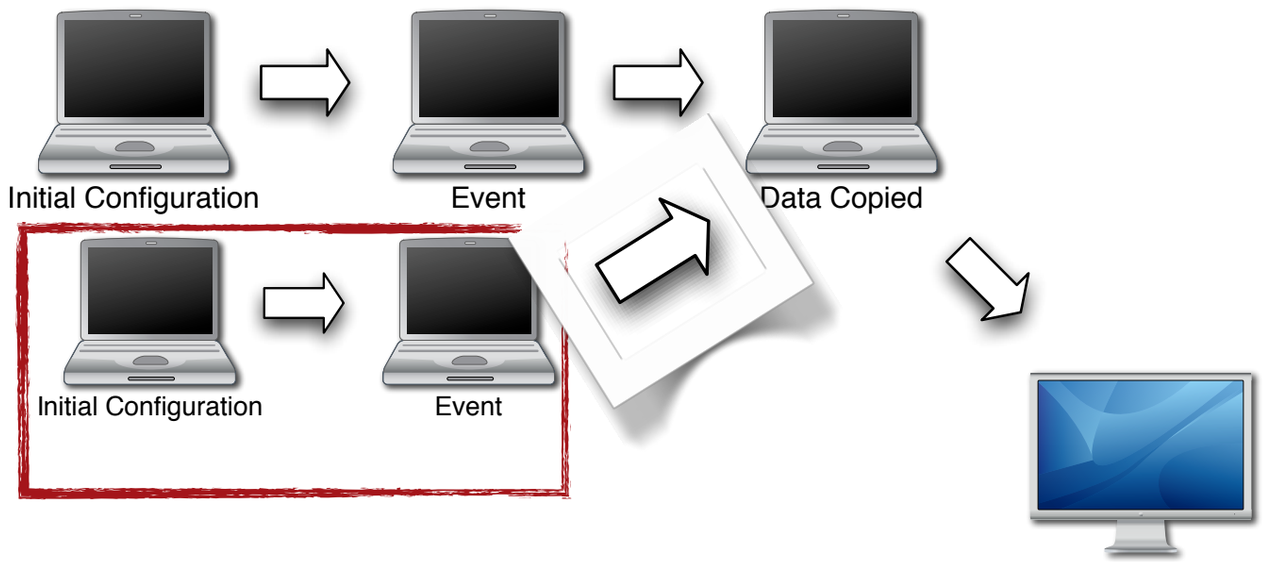
The hypothesis makes assumptions about:

- The system under investigation:
  - —*hardware (stock hardware? modified? firmware?)*
  - —*software (stock? custom? patch level?)*
- The flow of time
- The movement of evidence
- The system being used to investigate the data

Typical Forensics Workflow Process



But any piece of digital evidence can be explained by *multiple explanations.*



We assume:

- The event didn't fake the initial configuration
- Vulnerabilities we find were used by the attacker.
  - —*The attacker could have created a new vulnerability to hide what was actually used*
- We can can copy all of the computer's data
  - —*We can't get stuff out of L2 cache, some firmware, coprocessor, etc.*
- Our forensic tools are reliable
  - —*The attack might be invisible due to a bug in the forensic tool*

The most likely explanation may not be correct one.

Opportunities for tampering can be minimized by *proactively collecting evidence.*

Systems can record and retain:

- Log files — Recording events (syslog aggregation)
- Disk images (Snapshots)
  - —*Guidance Software's EnCase Forensic*
  - —*Access Data's FTK*
- Network packets and packet flows (Network Forensics)
  - —*Network Flight Recorder (NFR)*
  - —*NetIntercept (Niksun)*

Storage is cheap!

- A 1TB drive holds more than a week's worth of a consumer broadband traffic (@ 100%)

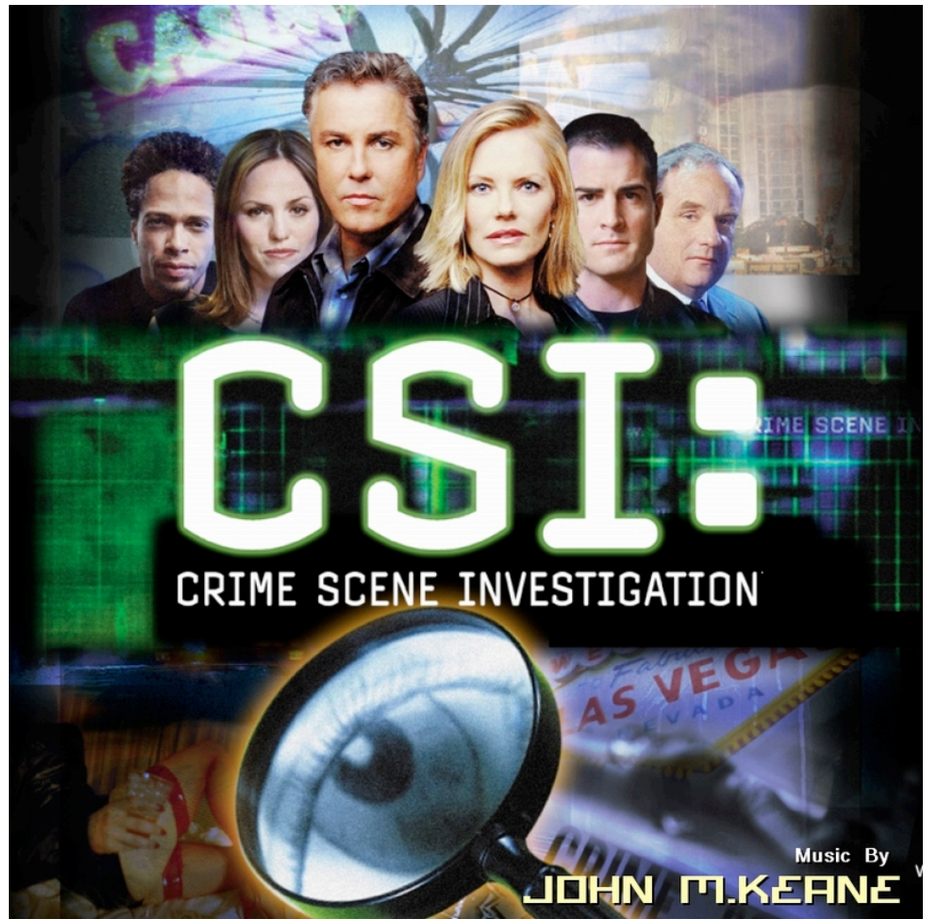
Proactive evidence allows investigators to discover:

- How a crime was committed
- Extent of damage / Presence of illegal activity
- Confirm/disprove an alibi

The “CSI Effect” causes victims and juries to have unrealistic expectations.

On TV:

- Forensics is swift
- Forensics is certain
- Human memory is reliable
- Presentations are highly produced



TV digital forensics:

- Every investigator is trained on every tool
- Correlation is easy and instantaneous
- There are no false positives
- Overwritten data can be recovered
- Encrypted data can usually be cracked
- It is impossible to delete anything

The reality of digital forensics is less exciting.

Every investigation is beset by problems:

- Data that is overwritten cannot be recovered
- Encrypted data usually can't be decrypted
- Forensics rarely answers questions or establishes guilt
- Forensics rarely provides specific information about a specific subject

Today’s Forensic tools are poor:

- Tools need to be guided by users to complete their tasks.
- Tools frequently chained together (A→B→C)
- The best tool may not be available
- Tools crash a lot

Forensics has many uses beyond the courtroom.

Data Recovery:

- Recover deleted files
- Recover data from physically damaged media

Testing and Evaluating:

- System Performance
- Privacy Properties & Tools
- Security Policies

Spot-check regulatory compliance:

- Internal information flows
- Data flow across network boundaries
- Disposal policies

Performance Evaluation

Conclusion:
Forensics and Digital Investigations

Scientific evidence requires interpretation to get it into a court room:

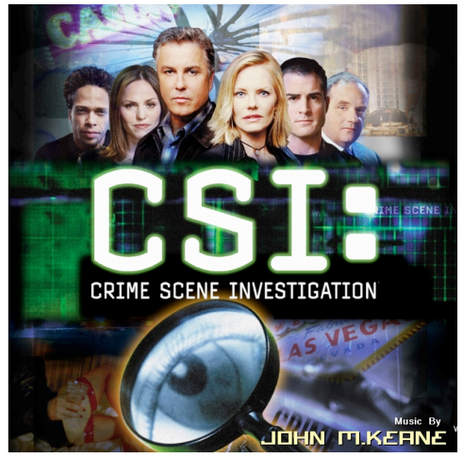
- You give a disk image to a jury

Digital evidence is easy to fake

- You can completely wipe a computer or restore its hard drive
  - —*You can't image and restore a physical crime scene*
- Digital tampering is intrinsically hard to detect
- The original data may be unavailable

Main uses today of digital forensics:

- Finding child pornography
- Recovering deleted files



Forensic Data Modalities

Where’s the digital forensic data?

Where’s the digital forensic data?

Where’s the digital forensic data?

Where’s the digital forensic data?

Where’s the digital forensic data?

Where’s the digital forensic data?

Where’s the digital forensic data?

Where’s the digital forensic data?

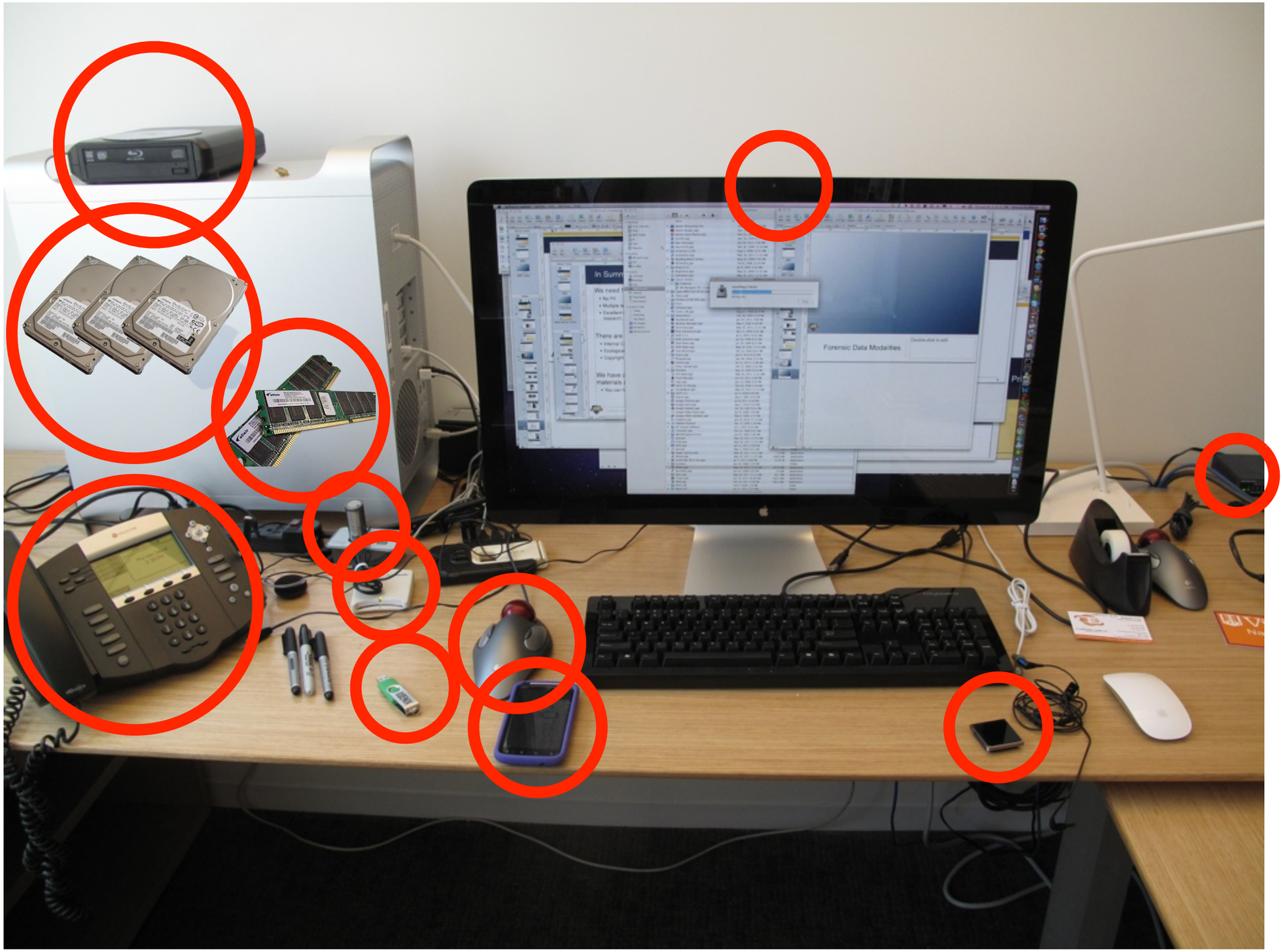
Where’s the digital forensic data?

Where’s the digital forensic data?

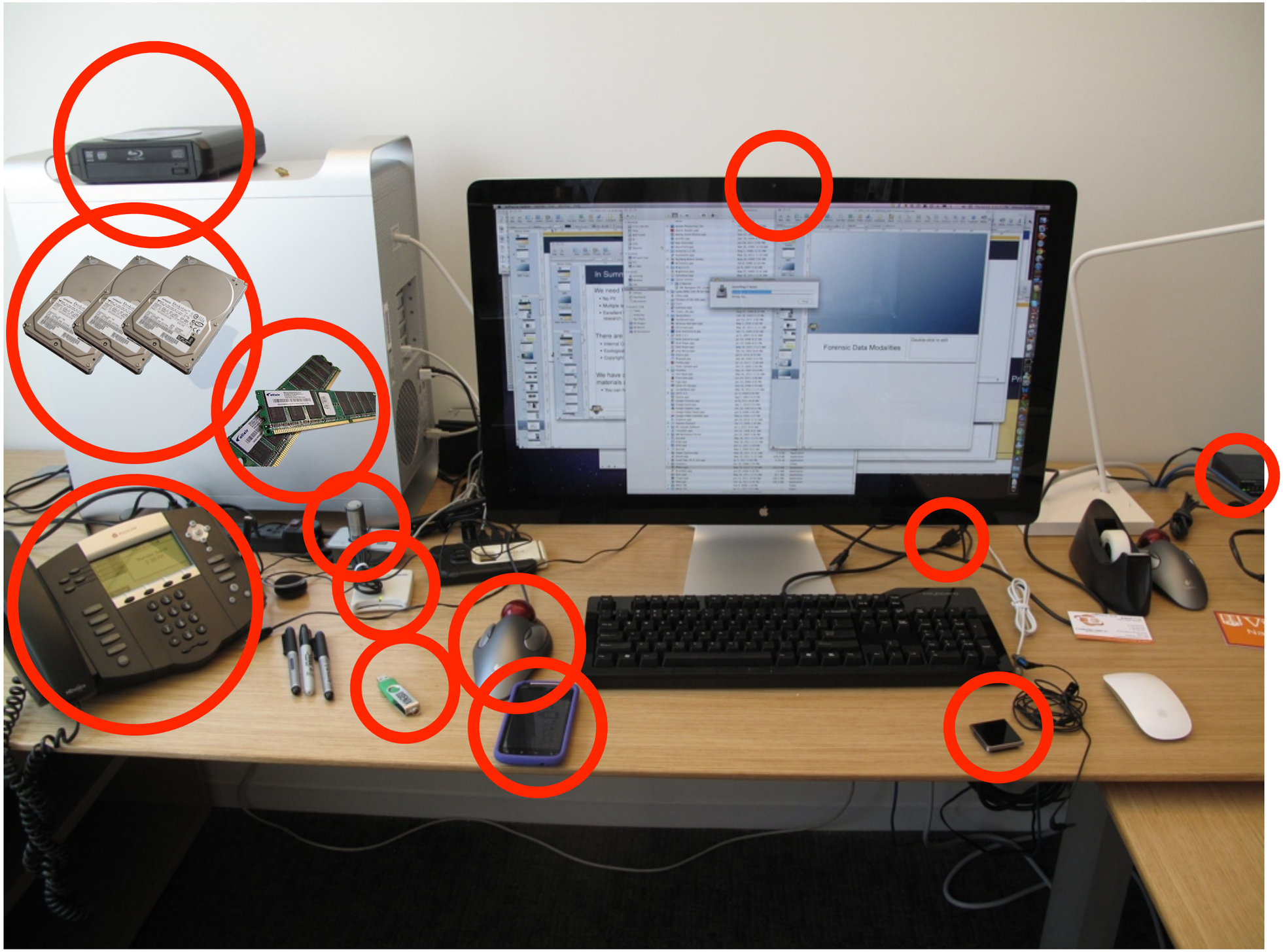
Where’s the digital forensic data?

Where’s the digital forensic data?

Where’s the digital forensic data?



Where’s the digital forensic data?



Hard drives (and disk images) are the most common form of digital evidence.

Typical hard drives store 250GB to 2TB.

A *logical dump* is a copy of all the files on the drive.

- Typically 0-1M files, 0-2TB in size.
- Frequently preserved as a ZIP or ZIP64 file.
- Commonly used in e-discovery.

A *physical dump* or *disk image* is sector-for-sector copy of the data.

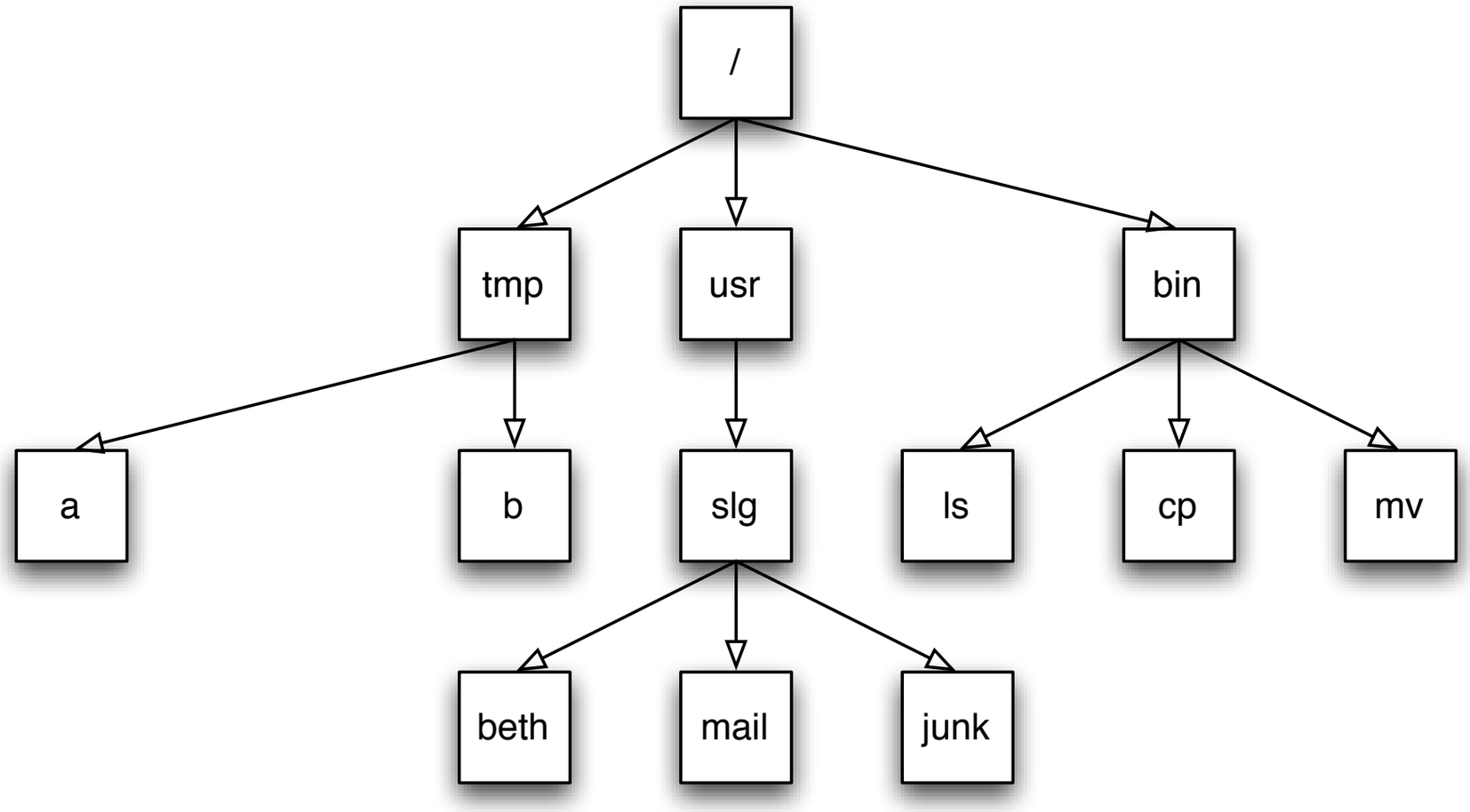
- Created with a *disk imaging tool* or *dd.*
  - —*FTK Imager*
  - —*EnCase Imager (filename.E01)*
  - —*dd:*

```
dd if=/dev/sda of=myfile.raw bs=64k conv=noerror,sync
```

- Typically authenticated with the MD5 hash of the disk image.

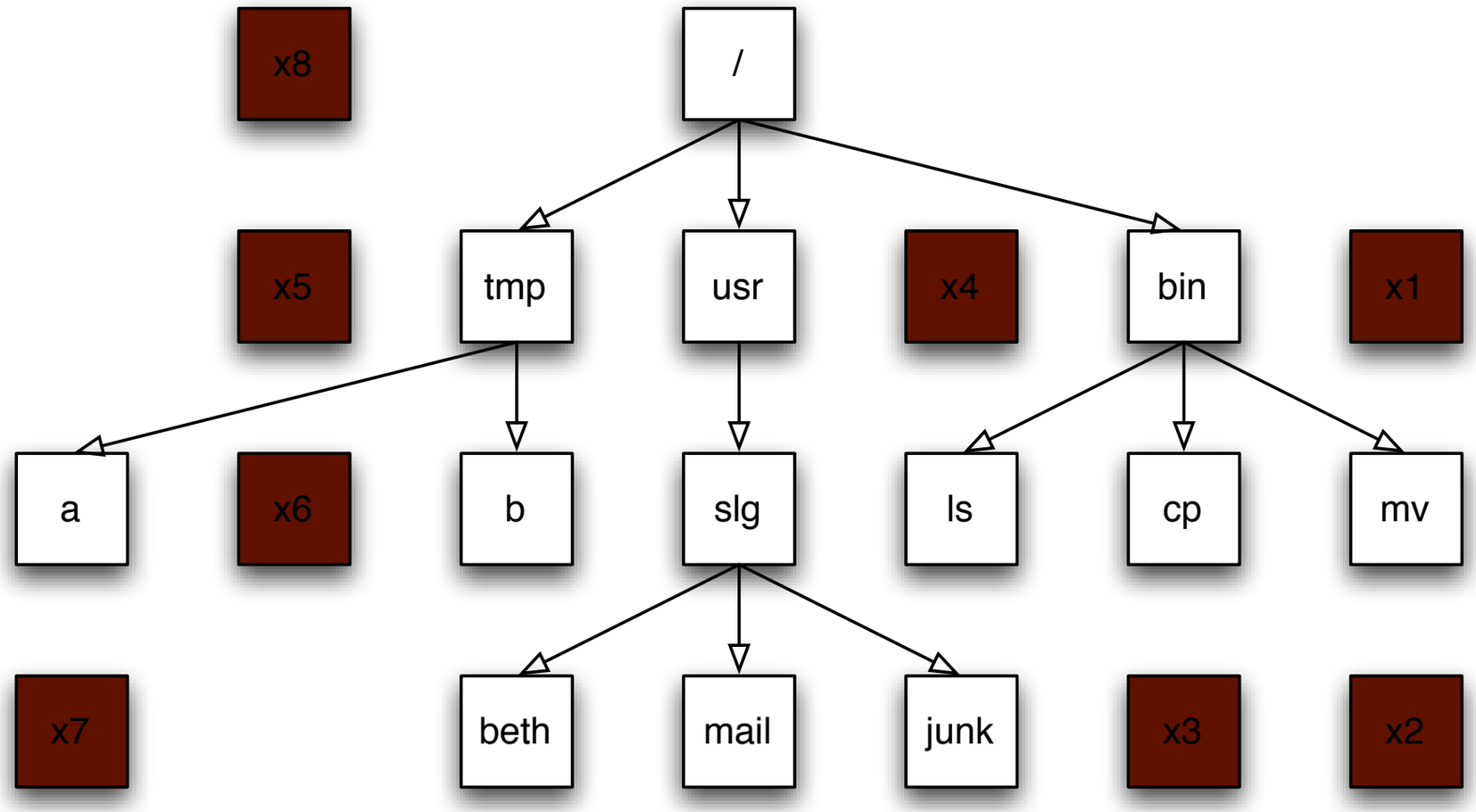
```
$ md5  nps-2009-canon2-gen6.raw
MD5 (nps-2009-canon2-gen6.raw) = 750b509d8fbed37a5213480aaccfdc61
$
```

If you *mount* a disk image, you will only see the allocated (“resident” or “overt”) files.



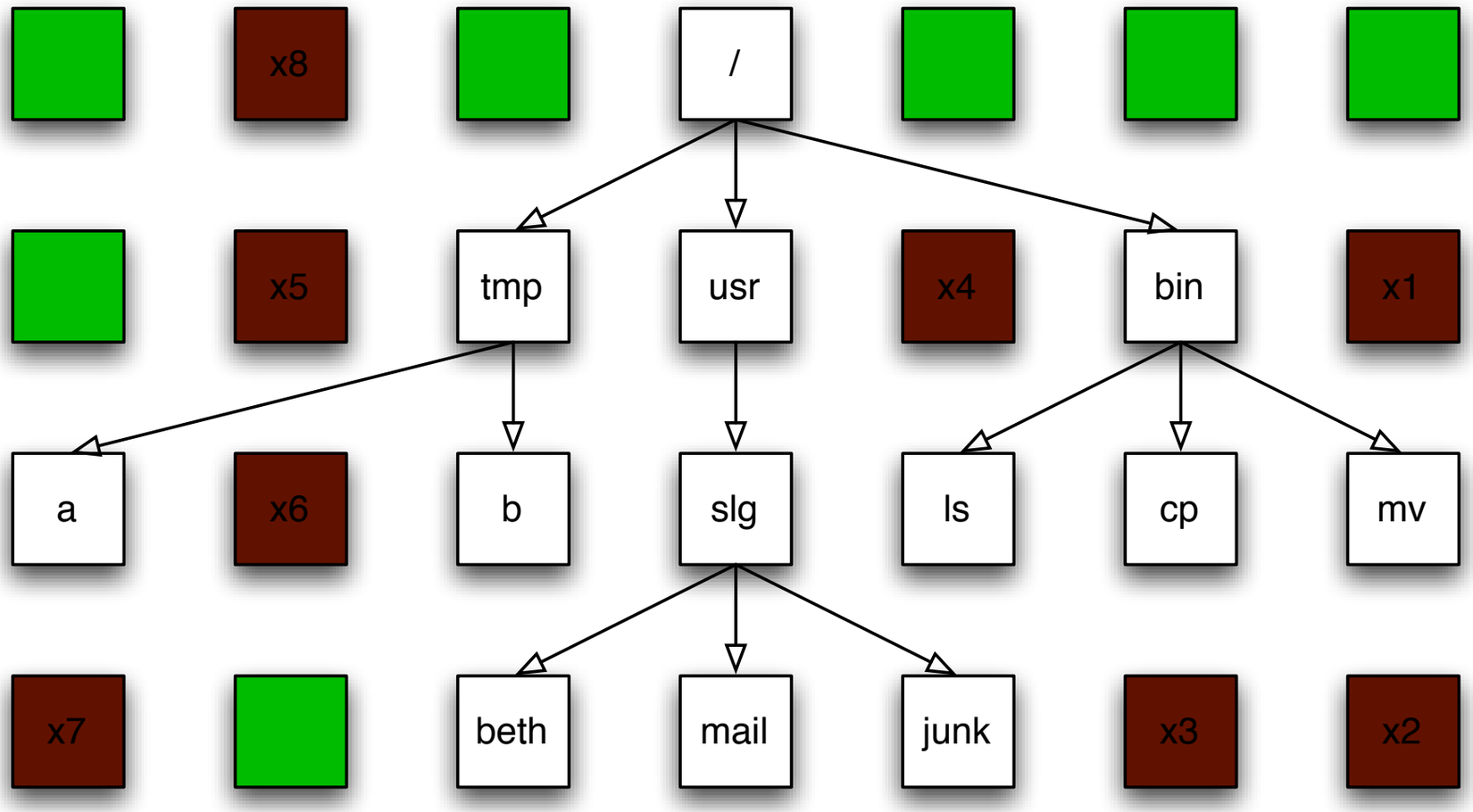
Resident Data

Data is on the disk that is not in the file system. It can only be recovered with forensic tools.



Deleted Data

Some sectors have “no data” and are blank.



No Data

Most forensic tools follow the same analysis steps.

Walk the file system to map out all the files (allocated & deleted).

For each file:

- Seek to the file.
- Read the file.
- Hash the file (MD5).
- Index file's text.

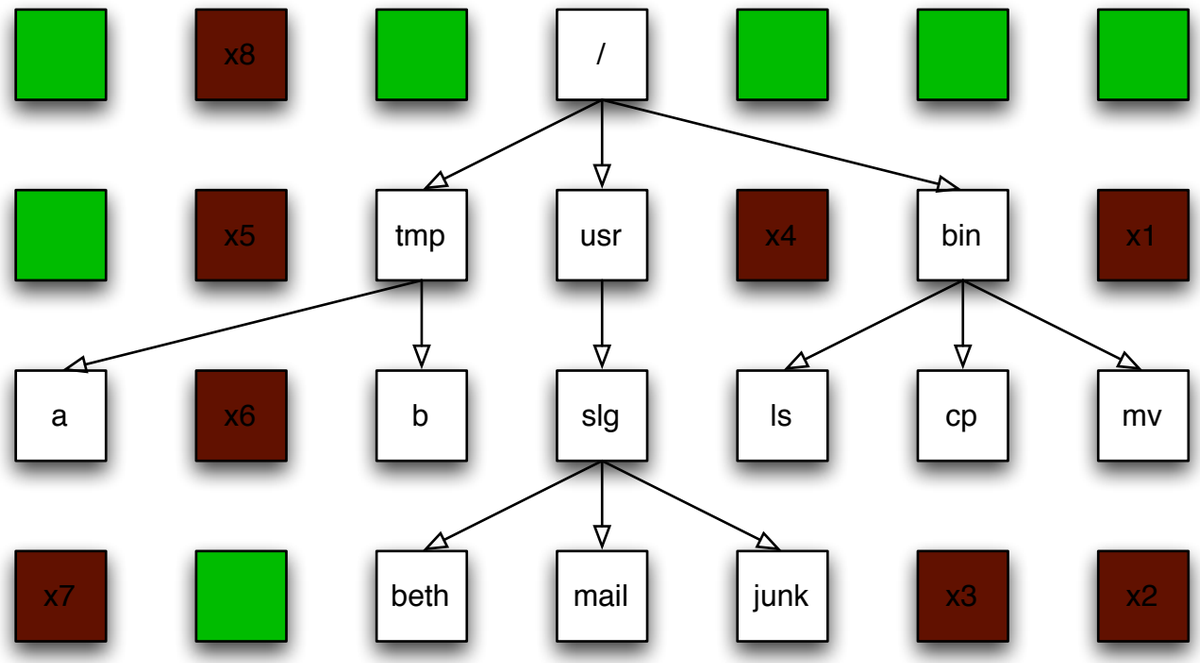
"Carve" space between files for other documents, text, etc.

Problem #1: Time

- 1TB drive takes 3.5 hours to read
  —*10-80 hours to process!*

Problem #2: Completeness

- Lots of residual data is ignored.
  —*Many investigations don't carve!*



Disk images are easy to acquire, but hard to work with.

Easy to acquire:

- Remove disk and image through a *write blocker*
- Boot a Linux “live CD” (e.g. Cain, SIFT or DEFT) and image to an external drive.
- Copy the *diskname.VHD* file from a Virtual Machine

Don’t forget the *metadata:*

- Serial number; manufacturer
- *When* the image was made
- *Who* made the image

Hard to work with:

- Files are *BIG* — typically ⅓ the size of the original disk image
- Special software required to extract files from disk image
- Special techniques required for most interesting data:
  - —*Deleted files*
  - —*Data between files*

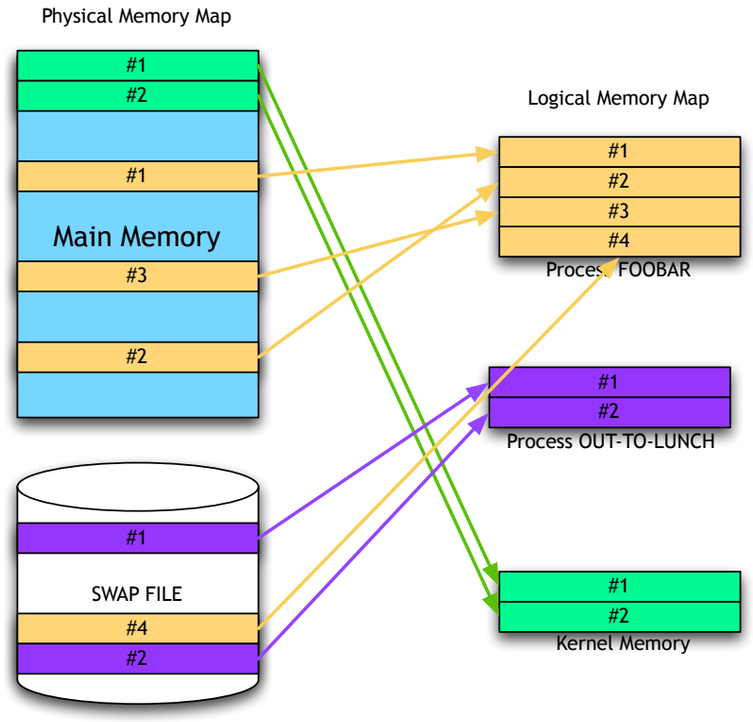
RAM analysis

The computer's RAM may contain:

- Discoverable evidence (e.g. logfiles, documents)
- Encryption keys
- Current network connections; Some kinds of malware
  - —*"Cold boot" attack lets you move memory between computers.*

Working with RAM is considerably harder than disk images:

- RAM contents are in constant motion.
- Physical memory map is radically different from logical map.
- Important information may not be in physical memory.
  - —*Registers*
  - —*Translation Lookaside Buffer (TLB)*
  - —*Swap File*



Two approaches to analysis:

- Understand the structures.
- Treat memory as "bulk data" and use *carving*.

Cell phones are an important source of digital evidence.

Cell phones are frequently used in the commission of crimes:

- Call history information & SMS messages
- GPS tracking information
- Apps and Documents

Cell cameras may:

- Document a crime
- Connect an image to the photographer

Cell phone data is exceedingly hard to work with:

- Multiple CPUs and memory systems. (SIM Chip; Cell memory; MMC)
- Multiple logical and physical dump formats.
- Different tools work with different phones.

Cameras (esp. built-in cameras)

Each camera records:

- An image (still or video)
- Unique pattern of defects in photo sensor

Many cameras also record:

- Manufacturer and Model number of camera.
- Unique serial number
- GPS location

JPEGs and MOV files are familiar, but:

- Software must handle 100,000s of photos at a time.
- JPEGs may be corrupt; contain attacks; etc.
- The most important information may be invisible or ignored by consumer software.

iPods, iPads, MP3 music players, and old-style PDAs

iPods are full-blown computers with:

- Storage
- Cameras
- Wireless networking

All manner of data can be stored:

- Music; Documents; Contacts

Old technology doesn't go away:

- Computers are used long after they leave the market.
- Forensic software must work with all versions of all computers.

USB storage devices:

USB devices are widely used and frequently encountered:

- Overt files
- Residual data from deleted files, applications, etc.
- Malware

Most USB storage systems are in FAT32 format.

Things to beware:

- A single USB device may contain multiple logical devices.
- Devices can have hidden partitions.
- Growing use of encryption.
- Logical vs. Physical
- Manufacturers do not release internal details

Where’s the digital forensic data?

Smart cards contain identity, cryptography, storage, and even applications.

Most smart cards have:

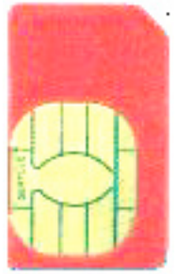
- Public and private keys.
- Some storage (16K-1GB)

Smart cards can also have:

- Loadable applications; Encrypted storage

GSM SIMs also have:

- Integrated circuit card identifier (ICCID)
- International mobile subscriber identity (IMSI)
- Authentication Key K1
- Phone book storage
- Multimedia storage
- IMEI - International mobile equipment identifier (*#06#)



Smart cards must be read with special readers.

- Most readers cannot access all of the data that the cards contain.

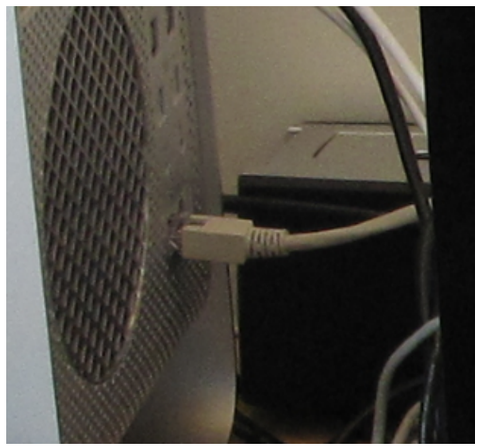
Network Devices and Network Traffic: Two kinds of forensic data.

Configuration data stored in the device.

- Flash configuration
- Volatile configuration

Data that moves over the network

- Packets
  - —*Full content*
  - —*Headers*
- Packet Flow Data
- Compressed Data
- Encrypted Data



Network data is examined for many purposes.

- It is *rarely if ever correlated with stored data or RAM.*

Optical media and drives:
Most crime scenes have multiple discs.

Optical media is deceptively ordinary:

- We all have experience with discs
- It seems “so 1990s”

But Optical Storage is quite complex:

- Multi-sessions may make older files invisible
- Discs may present different files under Windows, MacOS or Linux
- Blu-Ray has per-device encryption keys and content management

Optical discs are hard to work with:

- Manual handling
- Easily damaged
- Many tools do not examine all of the data

USB Devices

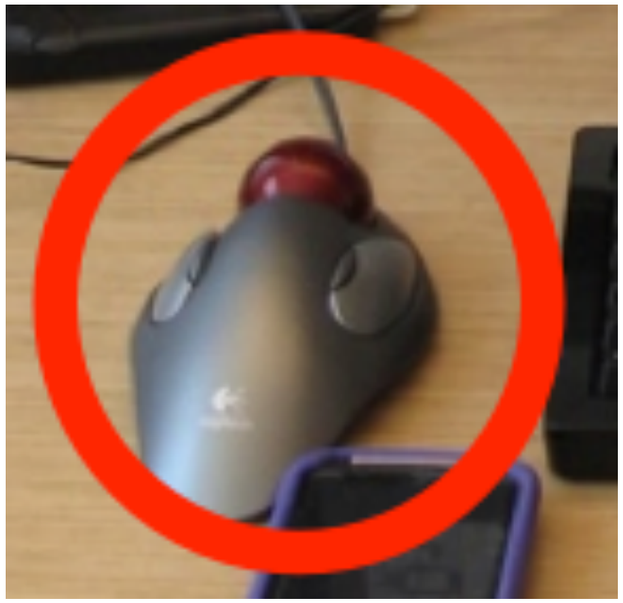
USB devices typically have

- Micro-controller & Firmware
- Unique identifier

—*"Tracing USB Device artefacts on Windows XP operating system for forensic purpose", Victor Chileshe Luo, School of Computing and Information Science, Edith Cowan University, Dec. 2007.*

The internals of USB devices are remarkably opaque:

- What other features are in your USB mouse?
- Why does a DROID phone present as a CDROM?
- Firewire allows DMA; does USB?



Desk Phones (especially VoIP phones)

Today desk phones are full-blown computer systems:

- CPU, RAM, Flash Storage & IP addresses
- Remotely-controllable microphones
- Built-in 100Mbps Ethernet Hub

Phones are examples of embedded devices

Other embedded devices include:

- Car event data recorders
- Smart meters
- Anything with a computer.

Embedded devices of forensic interest must be identified!

- They must be analyzed by individuals with domain-specific knowledge.

“Hidden” devices.

KEY GHOST
THE HARDWARE KEYLOGGER

Interface Security

THAWTE Authentic Site Secured by SSL

Ordering | Customer Support | Products | Company Info | Links | Helpdesk

We welcome

VISA

MasterCard AMERICAN EXPRESS

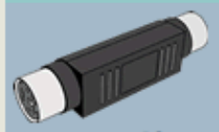
Home - Site Map

- Home
- Keylogger
- Reviews
- Demonstration
- Testimonials
- Photos
- Specifications
- FAQ
- Press releases
- Download
- Legal Disclaimer
- Affiliates
- Distributors

The KeyGhost Hardware Keylogger is a tiny plug-in device that records every keystroke typed on any PC computer.

learn more >>

KeyGhost Headlines



NEW! KeyGhost SX
New compact design. Huge 2,000,000 Keystroke capacity! Store and retrieve approx 12 months worth of typing. Patent Pending triple-speed download. Visit the website below for more information on this keylogger.
http://www.keyghost.com/sx

TimeDate Stamping KeyGhost SX

Click the link below to visit the **KeyGhost SX** website:
http://www.KeyGhost.com/SX

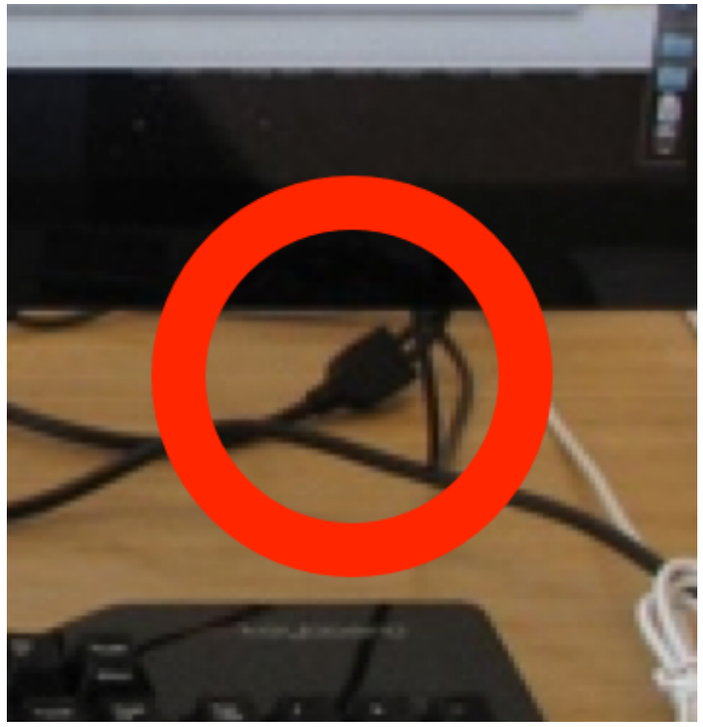
KeyGhost External Stand-alone Models
KeyGhost Home Edition 128K Flash Memory - $89
KeyGhost Std 512K Flash Memory - $99
KeyGhost Pro 1 Megabyte Flash Memory - $149
KeyGhost Pro SE 2 Megabyte Flash Memory - $199

ORDER NOW!
SSL Secure Site

KeyGhost Security Keyboards (all brand name)
KeyGhost Hardware KeyLogger Keyboards - from $129

KeyGhost USB Keylogger

Compatible with both PC and Mac USB keyboards. Click to read more...



BEFORE	AFTER
	

For security reasons, the photo (above right) is only a representation of what the KeyGhost key logger (stand alone unit) looks like. The actual KeyGhost key logger is injection moulded to look exactly like an EMC Balun.

There are *many* different kinds of digital forensics data.

In practice, tools (and practitioners) specialize:

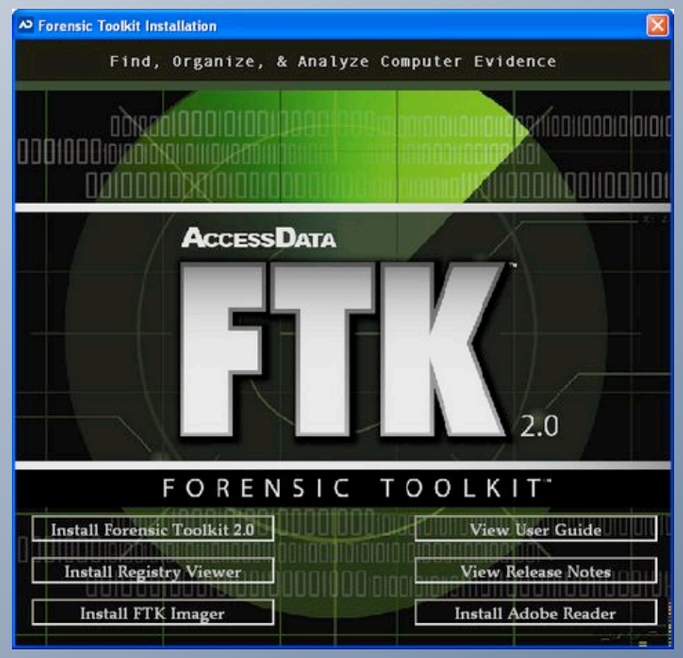
- Disk Images
- RAM
- Configuration Information
- Network Data
- Cell Phones
- Office Documents (Word; PDF; etc.)
- Multimedia Content (JPEGs, MPEGs, etc)

There are many commonalities:

- Logical vs. Physical
- Overt vs. Hidden

It's really hard to work with this data:

- Data extraction can be hard.
- Capacities are increasing — top-of-the-line computers to analyze top-of-the-line.
- Data is frequently corrupt.



Forensic Tools and Formats

Forensics Software: Commercial or Open Source?

Commercial (e.g. EnCase, FTK, etc.)

- Widely used in government & industry (+)
- Educational pricing usually available (+)
- Must be licensed for classroom (-)
- Complex user interface can detract from instruction (-)
- Typically runs just on Windows (-)
- Requires hardware license management (dongle) (-)

Open Source (e.g. The SleuthKit — TSK)

- Runs on Windows, Mac and Linux (+)
- No dongle (+)
- Good platform for further research (+)
- Less functionality than proprietary programs (-)
- Poor user interfaces (-)

We use both in our research.

There are many kinds of forensic data formats.

Disk Images

- Disk image files (MB to GB in size)

Packet Capture Files

- libpcap files

Memory Images

- raw files; debug files
- Swap files
- Hibernation Files

File Signatures

- Lists of hash codes (typically MD5)

File Lists

- Individual files (typically stored in a ZIP file or on a DVD).

There are many different disk image formats.

- RAW (**dd**) —
  - —*easiest format to work with; fast; very big*
  - —*Handled by all tools*
  - —*Many file systems (FAT32, ext2), cannot have files larger than 4GB*
- Split raw (file.000, file.001, file.002, etc.)
  - —*Not all tools can handle.*
- Encase (.E01) — compressed format developed by Expert Witness / Guidance Software
  - —*Compressed*
  - —*Evidence split across multiple “volumes” (file.E01, file.E02, etc.)*
  - —*Doesn't work with some tools (carvers, etc.)*
  - —*Supports "passwords" but not encryption.*

- AFF (Garfinkel 2005) — compressed open source format
  - —*Can store image as a single file (>2GB) or as multiple files (.afd format)*
  - —*Supports encryption and digital signatures; Extensible*
  - —*Poorly supported.*
  - —*Poor performance on certain Windows NTFS disk images.*

libewf: an open source library for reading EnCase files.

Libewf decodes .E01 files.

- libewf — C/C++ code
- jlibewf — Java E01 implementation
- cslibewf — C# implementation
- http://sourceforge.net/projects/libewf/files/

libewf must be compiled and installed *before* building other tools.

- Shared library (DLL or .so) or Static Library (.lib, .a)

libewf also includes command line tools:

- ewfacquire
  - —*simple disk imaging tool*
  - —*Will also convert RAW to E01*
- ewfinfo
  - —*Prints information about E01 files*
- ewfverify
  - —*Verifies the CRCs and MD5 of an E01 file*

We distribute disk images multiple formats at http://digitalcorpora.org

For disk images, we will use E01, AFF, and RAW (when reasonable)

- nps-2009-canon2-gen6 *A disk image from a 32MB Canon camera card.*

```
21039780294 2009-08-05 10:23 seed1.aff
20904705677 2009-08-05 00:10 seed1-redacted.aff
 1572833864 2011-07-16 08:39 seed1-redacted.E01
 1572846889 2011-07-16 08:47 seed1-redacted.E02
 1572832091 2011-07-16 08:53 seed1-redacted.E03
 1572831194 2011-07-16 08:59 seed1-redacted.E04
 1572861677 2011-07-16 09:05 seed1-redacted.E05
 1572850590 2011-07-16 09:10 seed1-redacted.E06
 1572835580 2011-07-16 09:14 seed1-redacted.E07
 1572849292 2011-07-16 09:18 seed1-redacted.E08
 1572839306 2011-07-16 09:21 seed1-redacted.E09
 1572840799 2011-07-16 09:23 seed1-redacted.E10
 1572862200 2011-07-16 09:27 seed1-redacted.E11
 1572859886 2011-07-16 09:32 seed1-redacted.E12
 1572833699 2011-07-16 09:38 seed1-redacted.E13
  713128482 2011-07-16 09:56 seed1-redacted.E14
80000000000 2009-08-05 08:02 seed1-redacted.raw
   93954789 2009-08-05 10:23 seed1.xml
11:07 ps14412:/corp/drives/nps/nps-2008-seed1$
```

The file extension describes the file format.

The file formats are incompatible but interconvertable.

```
  21039780294 2009-08-05 10:23 seed1.aff
  20904705677 2009-08-05 00:10 seed1-redacted.aff
   1572833864 2011-07-16 08:39 seed1-redacted.E01
...
    713128482 2011-07-16 09:56 seed1-redacted.E14
  80000000000 2009-08-05 08:02 seed1-redacted.raw
     93954789 2009-08-05 10:23 seed1.xml
11:07 ps14412:/corp/drives/nps/nps-2008-seed1$
```

The file extension describes the file format.

- E01 — *Expert Witness File Format (Guidance Software's EnCase)*
  - —*Basic compression (although not much here, as the card is filled)*
  - —*Some metadata: Examiner, Notes, MD5 of disk image.*
- AFF — Advanced Forensic Format (Garfinkel)
  - —*Better compression*
  - —*Arbitrary metdata*
- RAW — Raw disk image (dd)
- XML — Digital Forensics XML
  - — *"map" of the files in the disk.*

You can verify the integrity by computing the MD5 hash of the disk sectors.

Easiest approach — compute the MD5 of the raw file.

There are *many* MD5 commands:

```
$ md5 nps-2009-canon2-gen6.raw
MD5 (nps-2009-canon2-gen6.raw) = 750b509d8fbed37a5213480aaccfdc61

$ md5deep nps-2009-canon2-gen6.raw
750b509d8fbed37a5213480aaccfdc61  nps-2009-canon2-gen6.raw

$ openssl md5 nps-2009-canon2-gen6.raw
MD5(nps-2009-canon2-gen6.raw)= 750b509d8fbed37a5213480aaccfdc61
```

But you must validate the MD5 code using out-of-band information.

- It may be posted on the website.
- In actual practice, forensic examiners write the MD5 in their notebook.

The E01 file has a built-in MD5. You can display it with “ewfinfo”

```
$ ewfinfo nps-2009-canon2-gen6.E01
ewfinfo 20100805 (libewf 20100805, libuna 20100505,... , libcrypto 0.9.8)

Acquiry information
            Acquiry date:                   Mon Apr 12 11:12:32 2010
            System date:                    Mon Apr 12 11:12:32 2010
            Operating system used:          Darwin
            Software version used:          20090927
            Password:               N/A

EWF information
            File format:                    EnCase 6
            Sectors per chunk: 64
            Error granularity: 64
            Compression type:  no compression
            GUID:                           dc032794-bef0-2c45-8ede-8cc01ed31683

Media information
            Media type:                     removable disk
            Is physical:                    no
            Bytes per sector:  512
            Number of sectors: 60800
            Media size:                     29 MiB (31129600 bytes)

Digest hash information
            MD5:                            750b509d8fbed37a5213480aaccfdc61
```

The ewfverify command will verify the image integrity.

```
$ ewfverify nps-2009-canon2-gen6.E01
ewfverify 20100805 (libewf 20100805, libuna 20100505, ..., libcrypto 0.9.8)
Verify started at: Fri Jul 15 00:13:20 2011

This could take a while.

Status: at 0%.
        verified 32 KiB (32768 bytes) of total 29 MiB (31129600 bytes).

Status: at 1%.
        verified 320 KiB (327680 bytes) of total 29 MiB (31129600 bytes).
        completion in 18 minute(s) and 9 second(s) with 27 KiB/s (28299
bytes/second).

Status: at 64%.
        verified 19 MiB (20086784 bytes) of total 29 MiB (31129600 bytes).
        completion in 6 second(s) with 1.6 MiB/s (1729422 bytes/second).

Verify completed at: Fri Jul 15 00:13:32 2011

Read: 29 MiB (31129600 bytes) in 12 second(s) with 2.4 MiB/s (2594133 bytes/
second).

MD5 hash stored in file:       750b509d8fbed37a5213480aaccfdc61
MD5 hash calculated over data: 750b509d8fbed37a5213480aaccfdc61

ewfverify: SUCCESS
```

Likewise, the affinfo command will print information about an AFF file

```
$ affinfo nps-2009-canon2-gen6.aff
nps-2009-canon2-gen6.aff is a AFF file

nps-2009-canon2-gen6.aff
[skipping data segments]
                                        data
Segment                        arg      length    data
=======                    =========    ========    ====
badflag                            0         512   BAD SECTOR....9....5}..:.7.>....
badsectors                         2           8   = 0 (64-bit value)
afflib_version                     0           7   "3.5.5"
creator                            0           9   afconvert
aff_file_type                      0           3   AFF
acquisition_commandline            0          67   afconvert /corp/drives/nps/nps-2
pagesize                    16777216           0
sectorsize                       512           0
imagesize                          2           8   = 31129600 (64-bit value)
md5                                0          16   750B 509D 8FBE D37A 5213 480A ACCF DC61
sha1                               0          20   4742 C325 F105 83DA B1EB 4C55 D0D4
                                                   5AB3 BEB9 9EB3
image_gid                          0          16   258E FE7D 86A0 B08C BD89 8123 A206 9E22
acquisition_date                   0          20   2009-04-13 20:09:54.

Total segments:                   15   (15 real)
  Page  segments:                  2
  Hash  segments:                  0
  Signature segments:              0
  Null segments:                   0
$
```

The affverify will verify the file's integrity.

```
$ affverify nps-2009-canon2-gen6.aff
nps-2009-canon2-gen6.aff: no signing certificate present.

SHA1 stored in file:     4742c325f10583dab1eb4c55d0d45ab3beb99eb3
MD5 stored in file:      750b509d8fbed37a5213480aaccfdc61

  Read                0/      31129600 bytes; done in n/a
  Read         16777216/      31129600 bytes; done in  0:00:00

Calculated SHA1: 4742c325f10583dab1eb4c55d0d45ab3beb99eb3  VERIFIES
Calculated MD5:  750b509d8fbed37a5213480aaccfdc61          VERIFIES
$
```

Convert RAW to E01 with ewfacquire.

```
$ ewfacquire seed1-redacted.raw
ewfacquire 20101215

Storage media information:
Media size:             80 GB (80000000000 bytes)

Acquiry parameters required, please provide the necessary input
Image path and filename without extension: seed1-redacted
Case number:
Description: Multi-user Windows XP machine
Evidence number:
Examiner name: Simson Garfinkel
Notes:
Media type (fixed, removable, optical, memory) [fixed]: fixed
Media characteristics (logical, physical) [physical]: physical
Use compression (none, empty-block, fast, best) [none]: best
Use EWF file format (ewf, smart, ftk, encase1, encase2, encase3, encase4,
encase5, encase6, linen5, linen6, ewfx) [encase6]: encase6
Start to acquire at offset (0 >= value >= 80000000000) [0]:
The number of bytes to acquire (0 >= value >= 80000000000) [80000000000]:
Evidence segment file size in bytes (1.0 MiB >= value >= 7.9 EiB) [1.4 GiB]:
The number of bytes per sector (1 >= value >= 4294967295) [512]:
The number of sectors to read at once (64, 128, 256, 512, 1024, 2048, 4096, 8192,
16384, 32768) [64]:
The number of sectors to be used as error granularity (1 >= value >= 64) [64]:
The number of retries when a read error occurs (0 >= value >= 255) [2]:
Wipe sectors on read error (mimic EnCase like behavior) (yes, no) [no]:
```

AFF to RAW and vice-versa with affconvert:

RAW to AFF:

```
$ affconvert nps-2009-canon2-gen6.raw
convert nps-2009-canon2-gen6.raw --> nps-2009-canon2-gen6.aff
Converting page 1 of 1
md5: 750b509d8fbed37a5213480aaccfdc61
sha1: 4742c325f10583dab1eb4c55d0d45ab3beb99eb3
bytes converted: 31129600
Total pages: 2  (2 compressed)
Conversion finished.
$
```

AFF to RAW:

```
$ affconvert -e raw nps-2009-canon2-gen6.aff
convert nps-2009-canon2-gen6.aff --> nps-2009-canon2-gen6.raw
Converting page 1 of 1
bytes converted: 31129600
Conversion finished.
$
```

AFFLIB v3 added encryption & digital signatures

Encryption: each segment can be encrypted with a 256-bit AES key.

- AFFLIB automatically encrypts & decrypts each segment on read if possible.

Key can be specified as:

- passphrase that decrypts an **`afkey_aes256`** segment.
- X.509 certificate that decrypts a **`afkey_evp0`** segment.

Passphrase can be specified two ways:

```
$ export AFFLIB_PASSPHRASE='mypassphrase'
$ afinfo file://:mypassphrase@/filename.aff
```

Convert an encrypted AFF file to RAW:

```
$ export AFFLIB_PASSPHRASE='mypassphrase'
$ affconvert -e raw nps-2009-canon2-gen6.aff
```

—*or*

```
$ affconvert -e raw file://:mypassphrase@/nps-2009-canon2-gen6.aff
```

AFFLIB encryption example.

```
$ export AFFLIB_PASSPHRASE='password'
$ ./demo
$ afinfo file.aff
file.aff is a AFF file
file.aff: has encrypted segments

file.aff

                                        data
Segment                       arg       length    data
=======                    =========    ========    ====
badflag                            0         512   BAD SECTOR..2w..a.....A. ;...
badsectors                         2           8   = 0 (64-bit value)
afflib_version                     0           7   "3.5.8"
creator                            0           5   a.out
aff_file_type                      0           3   AFF
pagesize                    16777216           0
page0                             51           4   ....
imagesize                          2           8   = 65536 (64-bit value)
Bold indicates segments that were decrypted.

Total segments:                    9   (9 real)
  Page  segments:                  1
  Hash  segments:                  0
  Signature segments:              0
  Null segments:                   0
```

Without the passphrase, decryption is not possible.

```
$ unset AFFLIB_PASSPHRASE
$ afinfo -a file.aff
file.aff is a AFF file
file.aff: has encrypted segments

Segment                        arg      length    data
=======                    =========    ========    ====
badflag                            0         512   BAD SECTOR..2w..a.....A. ;...+
badsectors                         2           8   = 0 (64-bit value)
afflib_version                     0           7   "3.5.8"
creator                            0           5   a.out
aff_file_type                      0           3   AFF
affkey_aes256                      0          52   ........_.....4>.Nf..q..N..d.
pagesize/aes256             16777216           0
page0/aes256                      51          20   ....dswS.K...NL+....
imagesize/aes256                   2          24   +Y6..3f.......n.........

Total segments:                    9   (9 real)
  Encrypted segments:              3
  Page  segments:                  0
  Hash  segments:                  0
  Signature segments:              0
  Null segments:                   0
  Empty segments:                  0

Total data bytes in segments: 631
Total space in file dedicated to segment names: 107
Total overhead for 9 segments: 216 bytes (9*(16+8))
Overhead for AFF file header: 8 bytes
$
```

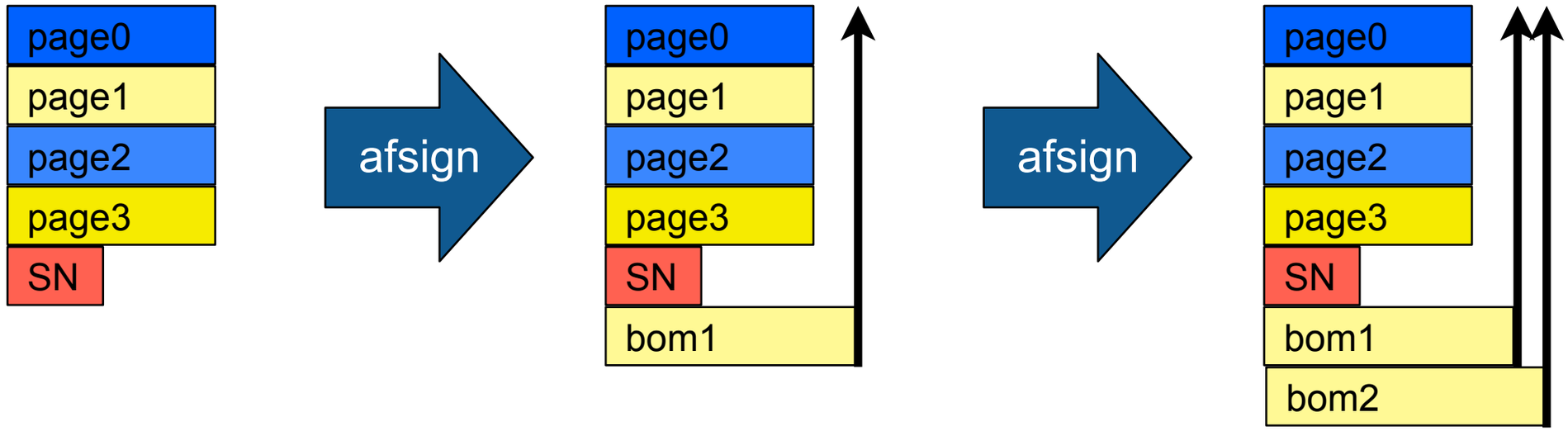
AFFLIBv3 also adds digital signatures and parity pages.

Signatures are as signed SHA256 values.

- Each segment's SHA256 is calculated.
- SHA256 values are signed using OpenSSL's EVP_Sign functions.

Signatures can be stored:

- In individual signature segments.
- In a new *Bill Of Materials (BOM)* segment.



- Multiple signatures can provide for chain-of-custody.
- **afsign** can also create a "parity page" for RAID-like reconstruction.

AFFLIBv3 status

AFFLIBv3 is in use today for research and education.

- Integrated with SleuthKit.

AFFLIB tools - A set of utilities for manipulating disk images.

- **affcat** — outputs an AFF file to stdout as a raw file
- **affcopy** & **affconvert** — segment-by-segment copying and verification (optional encryption)
- **affinfo** — prints details about the segments
- **affrecover** & **affix** — recovery of data within a corrupted AFF file
- **affsign** — signature tool
- **affverify** — verifies signatures
- **affcompare** — compares two disk images
- **affcrypto** — encrypt or decrypt a disk image in place
- **affdiskprint** — generates an XML-based "diskprint" for fast image comparison
- **affuse** — allows AFF images to be "mounted" as raw files on Linux
- **affsegemnt** — view or modify an individual segment

AFFLIBv3: strengths and weaknesses

Strengths:

- Single archive for storing all of the data and metadata
- Strong data integrity
- Compact archiving format (16MB segment size, optional LZMA)
- Supported by AccessData's FTK

Weaknesses:

- Performance.
  - —*16MB page size is problematic for some disk images due to MFT fragmentation.*
  - —*Caching is only solution at the present:*

```
export AFFLIB_CACHE_PAGES=24          # Dedicates 24*16=384MB to cache
export AFFLIB_CACHE_PAGES=64          # Dedicates 64*16=1GB to cache
```

- Only one disk image per file
  - —*Problem for lots of small devices*
- No way to package extracted files as a "logical" evidence file. (e.g. FILE.L01)
- Not supported by Guidance Software's EnCase

AFFv4 (under development) improves performance, provides for Logical files.

Working with Disk Images using The Sleuth Kit (TSK)

The Sleuth Kit (TSK) is a tool for working with disk images.

Open source computer forensics toolkit

- Originally “The Coroner's Toolkit,” developed by Dan Farmer & Wietse Venema
- Rewritten and maintained by Brian Carrier:
- http://www.sleuthkit.org/

SleuthKit works directly with disk images:

- View files & directories in a forensically sound manner (without modifying evidence)
- View deleted files
- Physical location of information

Without forensic tools, viewing data can change it!

- "last viewed" and "last modified" times can be changed

  —*simply mounting a file system changes it.*
- Entries can be put into the registry
- Temp files can be created

TSK is *the* open source forensic standard.

Image Formats	raw, split-raw, AFF, EWF, etc.
Partitioning Schemes	DOS MBR, GPT, Apple, BSD, Solaris
File Systems	FAT 12/16/32; NTFS; ext2/3; UFS 1/2; ISO9660
Platforms	Linux, OSX, Windows, *BSD, Cygwin, Solaris

Current Shortcomings:

- No support for EXT4 or YAFFS2
- No support for encrypted file systems
- No support for RAID

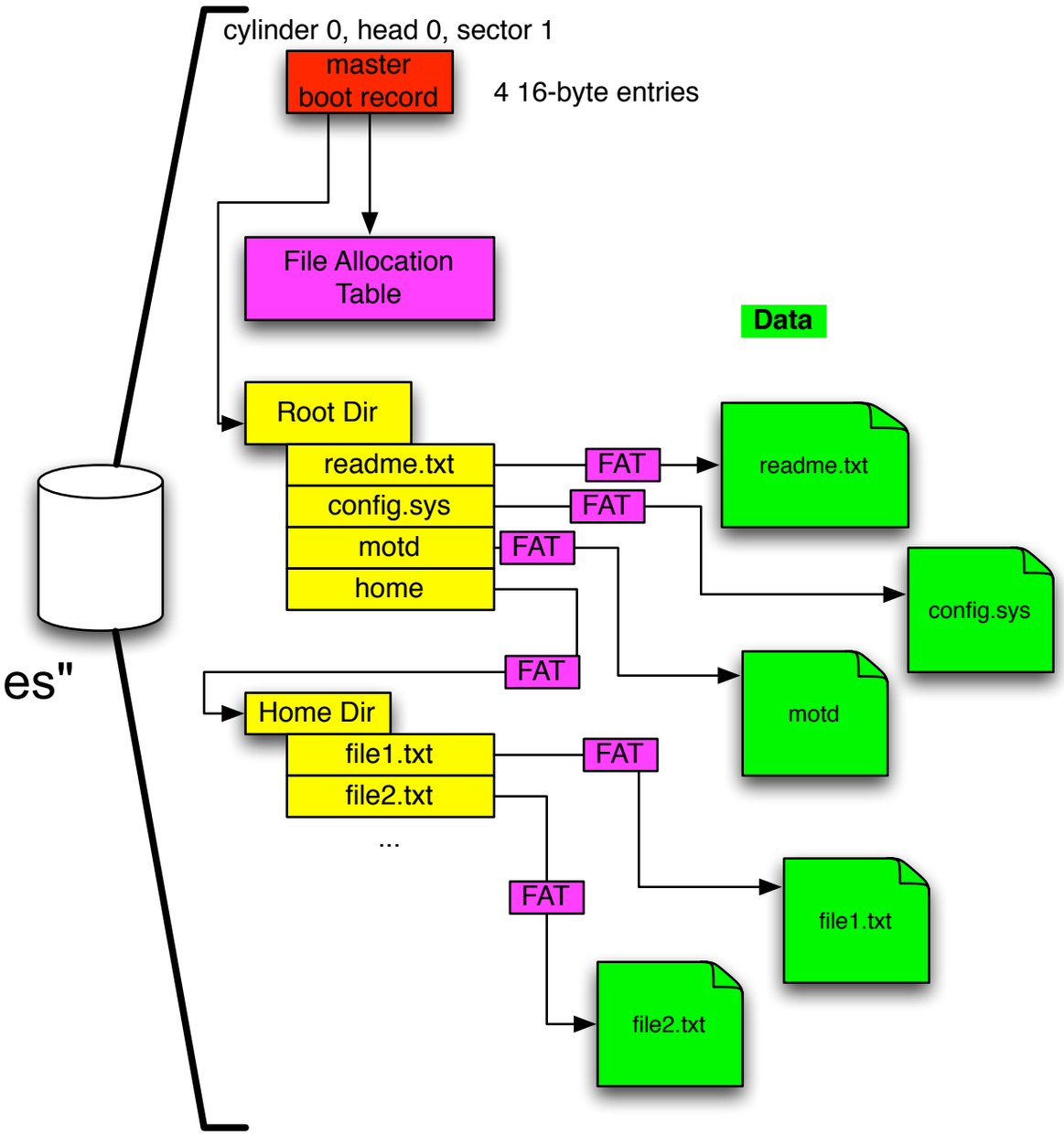
SleuthKit works with both data and metadata.

Data is the content of files.

Metadata tells how to work with the disk and the data.

- Partition table
- List of available sectors
- Directory information

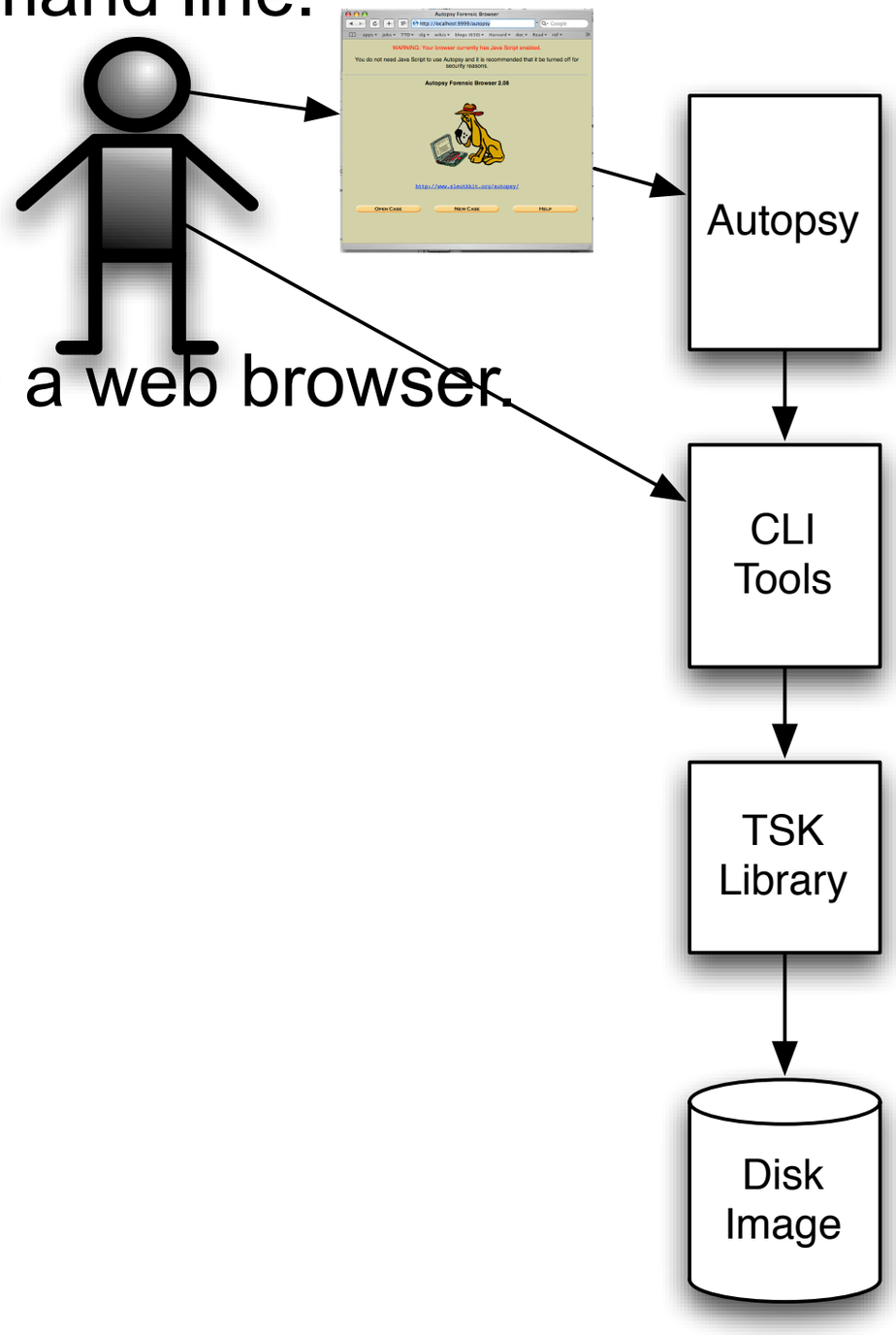
- Note: "Metadata" like EXIF and Word "properties" are considered *data* here.



TSK is a modular system

Most TSK commands are run from the command line.

- You can also write programs that call the library directly.



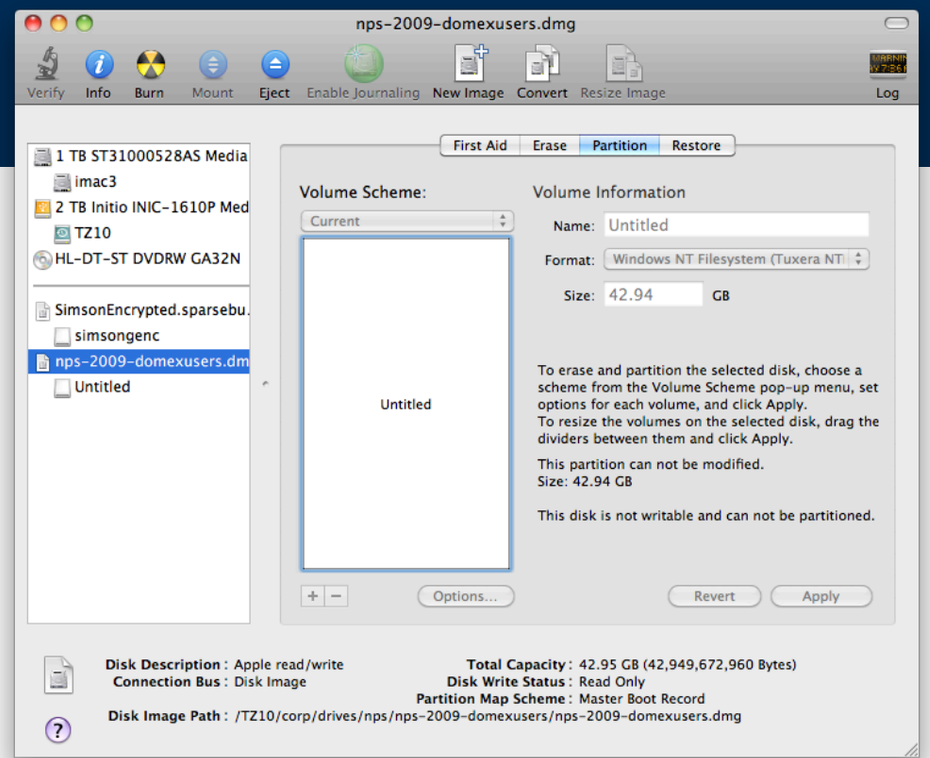
The Autopsy Forensic Explorer runs the commands and shows you the results in a web browser.

- Autopsy is being rewritten as a Java application.

The most commonly used commands are:

- mmls — list file systems
- fsstat — Status of a file system
- fls — list a directory
- istat — Status of a file
- icat — extract files

mmls: list the partitions



Disks contain one or more partitions.

- Each partition can contain one or more file systems

The mmls command will list the partitions:

```
$ mmls nps-2009-domexusers.raw
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000000062   0000000063   Unallocated
02:  00:00   0000000063   0083859299   0083859237   NTFS (0x07)
03:  -----   0083859300   0083886079   0000026780   Unallocated
$
```

Type “mmls -i list” to list the disk image formats *your mmls* supports:

```
$ mmls -i list
Supported image format types:
          raw (Single raw file (dd))
          aff (Advanced Forensic Format)
          afd (AFF Multiple File)
          afm (AFF with external metadata)
          afflib (All AFFLIB image formats (including beta ones))
          ewf (Expert Witness format (encase))
          split (Split raw files)
```

fsstat will give you the “statistics” about a file system. Be sure to use “-o offset” for partitioned disk images.

```
$ fsstat nps-2009-domexusers.raw
Cannot determine file system type
$ mmls nps-2009-domexusers.raw
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000000062   0000000063   Unallocated
02:  00:00   0000000063   0083859299   0083859237   NTFS (0x07)
03:  -----   0083859300   0083886079   0000026780   Unallocated
```

```
$ fsstat -o 63  nps-2009-domexusers.raw
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: NTFS
Volume Serial Number: 3CFCCD01FCCCB684
OEM Name: NTFS
Version: Windows XP

METADATA INFORMATION
--------------------------------------------
First Cluster of MFT: 786432
First Cluster of MFT Mirror: 5241202
Size of MFT Entries: 1024 bytes
Size of Index Records: 4096 bytes
Range: 0 - 36880
```

The “fls” command lets you list directories.

By default, fls shows the root directory:

```
$ fls -o 63  nps-2009-domexusers.raw
r/r 4-128-4:          $AttrDef
r/r 8-128-2:          $BadClus
r/r 8-128-1:          $BadClus:$Bad
r/r 6-128-1:          $Bitmap
r/r 7-128-1:          $Boot
...
r/r 7445-128-1:       AUTOEXEC.BAT
r/r 3516-128-3:       boot.ini
r/r 7444-128-1:       CONFIG.SYS
d/d 3524-144-6:       Documents and Settings
r/r 7446-128-1:       IO.SYS
r/r 25743-128-1:      IPH.PH
r/r 7447-128-1:       MSDOS.SYS
d/d 29222-144-1:      MSOCache
r/r 3487-128-3:       NTDETECT.COM
r/r 3483-128-3:       ntldr
r/r 27-128-1:         pagefile.sys
d/d 3993-144-6:       Program Files
d/d 29184-144-1:      RECYCLER
d/d 3522-144-6:       System Volume Information
d/d 28-144-6:         WINDOWS
d/d 36880:  $OrphanFiles
$
```

fls takes an optional directory argument:

```
...
d/d 3524-144-6:      Documents and Settings
...
$ fls -o 63  nps-2009-domexusers.raw 3524-144-6
d/d 10219-144-6:     Administrator
d/d 3526-144-6:      All Users
d/d 3525-144-7:      Default User
d/d 27708-144-5:     domex1
d/d 28463-144-5:     domex2
d/d 10146-144-6:     LocalService
d/d 3370-144-6:      NetworkService
$
$ fls -o 63  nps-2009-domexusers.raw 27708-144-5
d/d 27748-144-6:     Application Data
d/d 27747-144-5:     Cookies
d/d 27746-144-1:     Desktop
d/d 27745-144-1:     Favorites
d/d 27730-144-6:     Local Settings
d/d 27729-144-6:     My Documents
...
$ fls -o 63  nps-2009-domexusers.raw 27729
r/r 27820-128-1:     desktop.ini
d/d 27824-144-1:     My Music
d/d 27821-144-1:     My Pictures
r/r 35419-128-3:     This is a spreadsheet by domex user 1.xlsx
r/r 35424-128-3:     This is a spreadsheet sent by domex user 1.xlsx
r/r 35395-128-3:     This is a word document by domex user 1.docx
```

27708 = NTFS File #
144-5 = MFT Entry Attribute

The istat command gives information about an inode

```
r/r 35419-128-3:     This is a spreadsheet by domex user 1.xlsx
...
$ istat -o 63  nps-2009-domexusers.raw 35419-128-3
MFT Entry Header Values:
Entry: 35419        Sequence: 2
$LogFile Sequence Number: 157027741
Allocated File
Links: 2

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 1060  ()
Created:    Wed Oct 29 12:16:27 2008
File Modified:        Wed Oct 29 12:16:28 2008
MFT Modified:         Wed Oct 29 12:16:28 2008
Accessed:   Wed Oct 29 22:04:32 2008

$FILE_NAME Attribute Values:
Flags: Archive
Name: This is a spreadsheet by domex user 1.xlsx
Parent MFT Entry: 27729        Sequence: 1
Allocated Size: 12288          Actual Size: 8230
Created:    Wed Oct 29 12:16:27 2008
File Modified:        Wed Oct 29 12:16:28 2008
MFT Modified:         Wed Oct 29 12:16:28 2008
Accessed:   Wed Oct 29 12:16:28 2008
```

There is a lot more istat information than you expect.

```
r/r 35419-128-3:     This is a spreadsheet by domex user 1.xlsx
...
$ istat -o 63  nps-2009-domexusers.raw 35419-128-3
...
$OBJECT_ID Attribute Values:
Object Id: cdbb2629-0c00-aca6-11dd-a50ef743eede

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size: 72
Type: $FILE_NAME (48-5)   Name: N/A   Resident   size: 90
Type: $FILE_NAME (48-4)   Name: N/A   Resident   size: 150
Type: $OBJECT_ID (64-6)   Name: N/A   Resident   size: 16
Type: $DATA (128-3)   Name: N/A   Non-Resident   size: 8230  init_size: 8230
139149 139150 139372
$
```

The icat command outputs inode contents to stdout:

```
$ icat -o 63  nps-2009-domexusers.raw 35419-128-3 > /tmp/sheet.xlsx
$ ls -l /tmp/sheet.xlsx
-rw-r--r--  1 simsong  wheel  8230 Jul 16 12:58 /tmp/sheet.xlsx
```

You can use the “file” command to verify the file type:

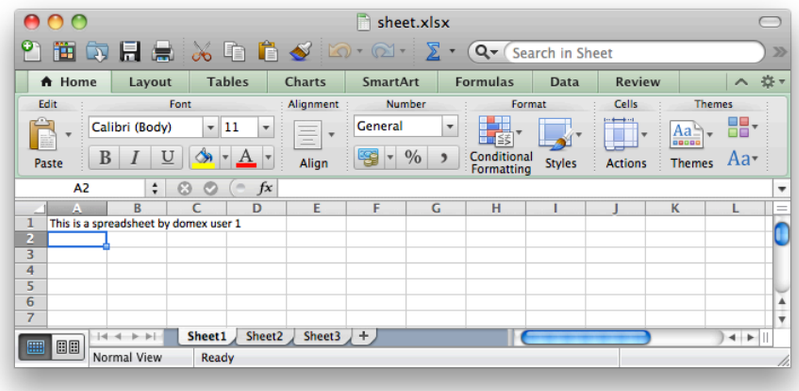
```
$ file /tmp/sheet.xlsx
/tmp/sheet.xlsx: Zip archive data, at least v2.0 to extract
$
```

Once you have a file, you can open it or view it.

On a Mac, you can “open” the file with the “open” command.

—*This is usually a bad idea...*

```
$ open /tmp/sheet.xlsx
```



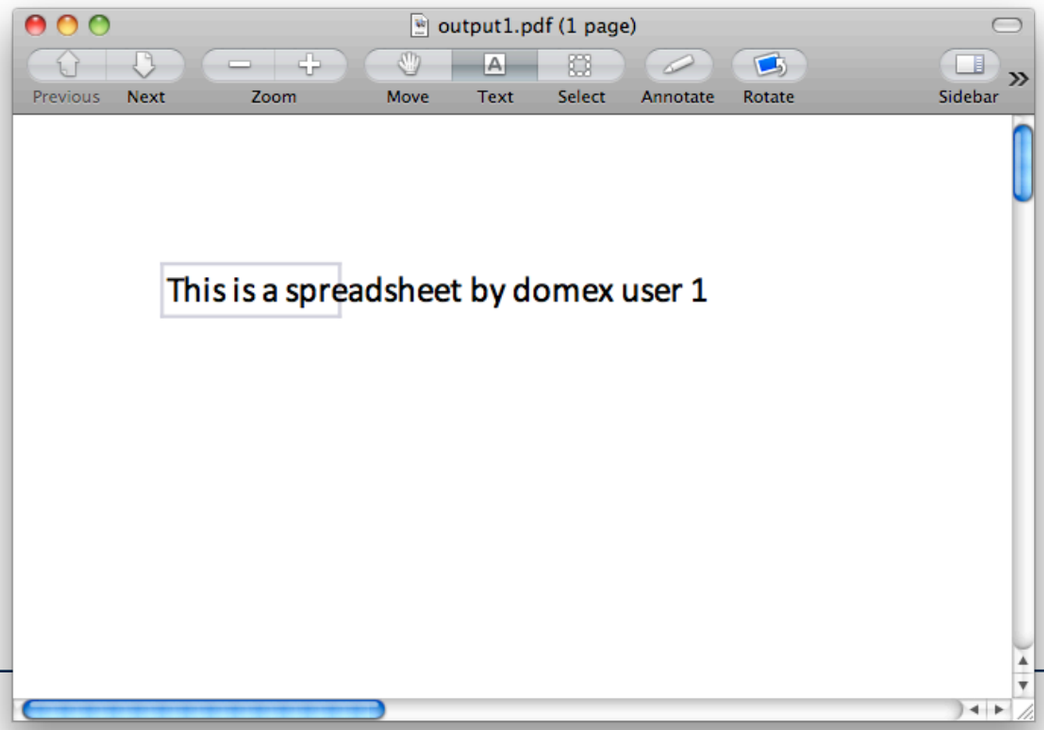
You can generate HTML with qlmanage:

```
$ mkdir sheet
$ qlmanage -o sheet -p /tmp/sheet.xlsx
Testing Quick Look preview with files:
                /tmp/sheet.xlsx
* /tmp/sheet.xlsx produced a preview with data of type public.html
$ ls -l sheet/sheet.xlsx.qlpreview/
total 64
-rw-r--r--  1 simsong  wheel    189 Jul 16 13:02 Attachment1.png
-rw-r--r--  1 simsong  wheel   3456 Jul 16 13:02 Attachment2.png
-rw-r--r--  1 simsong  wheel   8115 Jul 16 13:02 Attachment3.js
-rw-r--r--  1 simsong  wheel    336 Jul 16 13:02 Attachment4.html
-rw-r--r--  1 simsong  wheel    569 Jul 16 13:02 Attachment5.html
-rw-r--r--  1 simsong  wheel    336 Jul 16 13:02 Attachment6.html
-rw-r--r--  1 simsong  wheel   3729 Jul 16 13:02 Attachment7.css
-rw-r--r--  1 simsong  wheel   1044 Jul 16 13:02 Preview.html
-rw-r--r--  1 simsong  wheel  26328 Jul 16 13:02 PreviewProperties.plist
```

Many forensic applications require PDFs.

You can convert HTML to PDF with wkhtmltopdf and pdftk:

```
$ grep Attachment..html sheet/sheet.xlsx.qlpreview/PreviewProperties.plist
                                <string>Attachment6.html</string>
                                <string>Attachment4.html</string>
                                <string>Attachment5.html</string>
$ wkhtmltopdf sheet/sheet.xlsx.qlpreview/Attachment5.html output1.pdf
Loading pages (1/6)
Counting pages (2/6)
Resolving links (4/6)
Loading headers and footers (5/6)
Printing pages (6/6)
Done
$ pdftk output1.pdf output2.pdf output3.pdf output.pdf
$ open output.pdf
```



<dfxml>

Digital Forensics XML

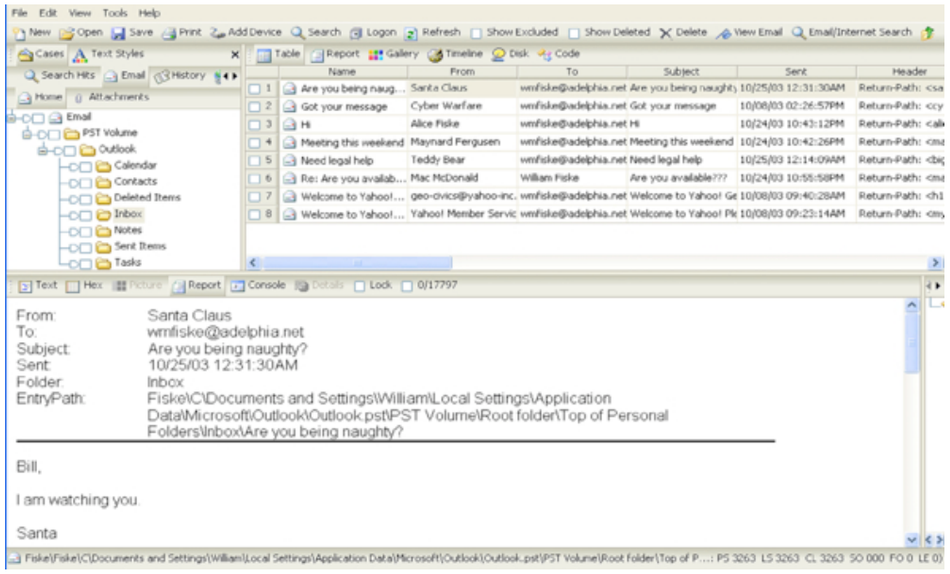
Remember that XML file?

```
21039780294 2009-08-05 10:23 seed1.aff
20904705677 2009-08-05 00:10 seed1-redacted.aff
 1572833864 2011-07-16 08:39 seed1-redacted.E01
 1572846889 2011-07-16 08:47 seed1-redacted.E02
 1572832091 2011-07-16 08:53 seed1-redacted.E03
 1572831194 2011-07-16 08:59 seed1-redacted.E04
 1572861677 2011-07-16 09:05 seed1-redacted.E05
 1572850590 2011-07-16 09:10 seed1-redacted.E06
 1572835580 2011-07-16 09:14 seed1-redacted.E07
 1572849292 2011-07-16 09:18 seed1-redacted.E08
 1572839306 2011-07-16 09:21 seed1-redacted.E09
 1572840799 2011-07-16 09:23 seed1-redacted.E10
 1572862200 2011-07-16 09:27 seed1-redacted.E11
 1572859886 2011-07-16 09:32 seed1-redacted.E12
 1572833699 2011-07-16 09:38 seed1-redacted.E13
  713128482 2011-07-16 09:56 seed1-redacted.E14
80000000000 2009-08-05 08:02 seed1-redacted.raw
   93954789 2009-08-05 10:23 seed1.xml
```

The XML file contains a “map” of every file in the disk image.

- The format is Digital Forensics XML (DFXML), an XML application we have been developing for the past four years.

Today's forensic tools are designed for performing forensic investigations.



Encase:
- GUI Closed Source
- EnScript

SleuthKit:
- Command-line Open Source
- C/C++ API

These tools are great for:

- File recovery
- Search

Not so great for automation, interoperability, or research.

Automation requires a forensics “language.” Standardized *formats* and *abstractions*.

Today we have limited formats and abstractions:

- Disk images — raw & EnCase E01 files
- Packet Capture files — BPF format
- Files — distributed as files or as ZIP for collections of files
- File Signatures — List of MD5 (or SHA1) hashes in hex with no context.
- “Selector Lists” — Lists of email address, CCNs, etc. (typically ASCII, rarely in Unicode)

We need new structured formats for distributing:

- Signatures Metrics (parts of files; n-grams; piecewise hashes; similarity metrics)
- File Metadata (e.g. Microsoft Office document properties)
- File system metadata (MAC times, etc.)
- Application Profiles (e.g. collections of files that make up an application.)
- Internet and social network information

Creating, testing, and adopting schema and formats is hard work.

Today there is no good match between forensic tools and the needs of researchers.

Several of today's tools allow some degree of programmability:

- EnCase — EScript
- PyFlag — Flash Script & Python
- Sleuth Kit — C/C++

Writing programs for these systems is hard:

- Many of the forensic tools are not designed for easy automation.
- Programming languages are *procedural* and *mechanism-oriented.*
- Data is separated from actions on the data.

TSK 3.2 introduced tsk_loaddb, a tool for saving file information into an SQLlite DB.

tsk_loaddb:

- Walks all file systems in an image; extracts metadata into an SQLite3 database.
- Use multiple SELECT statements, you can generate reports:

```
$ tsk_loaddb
usage: tsk_loaddb [-vVk] [-i imgtype] [-b dev_sector_size] [-d output_dir]
image [image]
            -k: Don't create block data table
$ mkdir out
$ tsk_loaddb -k -d out nps-2009-domexusers.raw
$ ls -l out
total 3784
-rw-r--r--  1 simsong  staff  3872768 Jul 16 14:29 nps-2009-domexusers.raw.db
$ sqlite3 out/nps-2009-domexusers.raw.db
...
sqlite> .tables
tsk_db_info      tsk_fs_info      tsk_image_names  tsk_vs_parts
tsk_fs_files     tsk_image_info   tsk_vs_info
sqlite> select * from tsk_fs_files limit 1000,3;
1|11399|128|5|download-page[2].css|10243|5|1|1|5|790|1224542464|1224542464|
1224542674|1224542464|511|0|0
1|11531|128|5|downloading[1].htm|10243|5|1|1|5|29158|1224542571|1224542570|
1224542571|1224542571|511|0|0
1|11489|128|4|downloading[2]|10243|5|1|1|5|395|1224542571|1224542571|
1224542571|1224542571|511|0|0
sqlite>
```

Digital Forensics XML:
An approach for standardizing forensic metadata

XML is well suited to forensics:

- We can represent a wide variety of data **today**.
- As our techniques improve, we can add new XML tags.
- More programmers speak XML than “forensics.”

Today we have XML tags to describe:

- Files and file metadata.
- Hash codes.
- Partitioning schemes.
- Application metadata.
  - —*We can use the same XML tags in many different applications.*
  - —*We can develop APIs to leverage the XML from python, perl, Java, etc.*

More expressive than SQL; our implementation has more data.

The <fileobject> tag is the most important in disk forensics.

The DFXML <fileobject> tag describes information about a file.

- File name, size, and hash codes.
- Physical Location on the disk.
- Provenance

Simple example:

```
<fileobject>
<filename>samplefile.bin</filename>
<filesize>9014</filesize>
<mtime format='time_t'>1297835303.0</mtime>
<ctime format='time_t'>1297835303.0</ctime>
<atime format='time_t'>1299631657.0</atime>
<hashdigest type='MD5'>8dfcbdce6562602911990bdfd661415a</hashdigest>
<byte_runs>
  <run file_offset='0' len='9014' fs_offset='6553600' img_offset='6585856'/>
</byte_runs>
</fileobject>
```

Multiple <fileobject>s can be used for a list of hashes.

A hash list might be include metadata about the hashes, but lack timestamp and physical placement info:

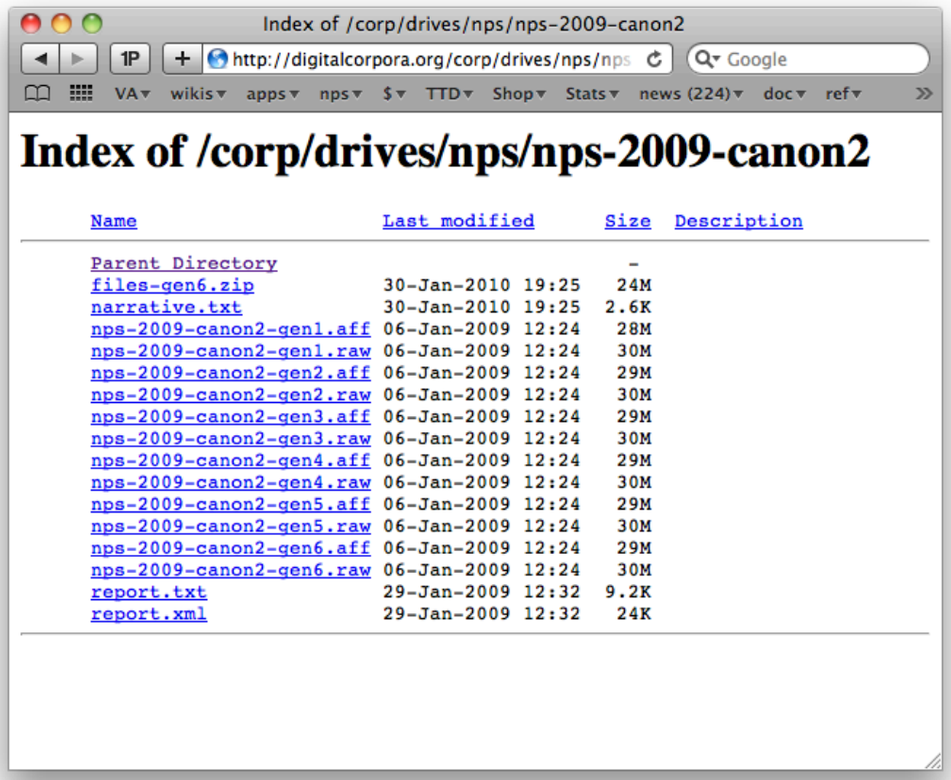
```
<?xml version='1.0' encoding='UTF-8'?>
<dfxml xmloutputversion='0.3'>
<metadata xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns='http://afflib.org/fiwalk/'
  xmlns:dc='http://purl.org/dc/elements/1.1/'>

<classification>UNCLASSIFIED</classification>
<dc:type>Hash Set</dc:type>
<dfxml xmloutputversion='0.3'>

<fileobject>
<filename>demo1.bin</filename>
<filesize>1718</filesize>
<hashdigest type='MD5'>8e008247fde7bed340123f617db6a909</hashdigest>
</fileobject>

<fileobject>
<filename>demo2.bin</filename>
<hashdigest type='MD5'>c44293fdb35b6639bdffa9f41cf84626</hashdigest>
</fileobject>

</dfxml>
```

DFXML is a convenient way to annotate disk images that we distribute.

With DFXML we can quickly describe:

- File systems in the image.
- Number of files on a disk, their names and hash values.
- Human languages in use.
- Distribution of file types.
- Location of files on the disk.
- Etc.

This is more extensible than SQLite.



We have a growing list of tools that use DFXML

Generating DFXML:

- **fiwalk** — Creates DFXML from disk images. (Based on SleuthKit)
- **frag_find** — Hash-based carving; DFXML indicates where the files are in the disk image.
  —*Used for malware detection, reassembling RAIDs, data exfiltration detection.*
- **dfxml_tool** — Generates DFXML hash lists from files.

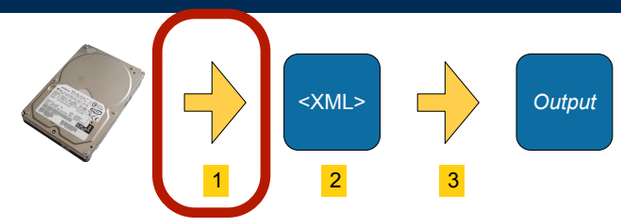
Consuming DFXML:

- **imap.py** – Prints a “map” of a disk image.
- **iverify.py** — Reports if the DFXML file matches a disk image.
- **iredact.py** — Removes or alters sensitive files in a disk image.
- **iblkfind.py** — Reports the file that maps to a given disk sector.
- **idifference.py** — Reports difference between two disk images.
- **iexport.py** — Exports the unallocated sectors.
- **iextract.py** — Extracts files of a given type.
- **igrep.py** — Reports the files in a disk image that match a string
- **ihistogram.py** — Fast histograms of the files on the disk

This is part of the fiwalk distribution, at http://afflib.org/

fiwalk extracts metadata from disk images.

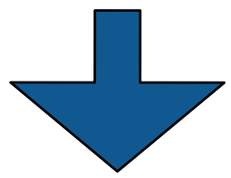
fiwalk is a C++ program built on top of SleuthKit




```
$ fiwalk [options] -X file.xml imagefile
```

Features:

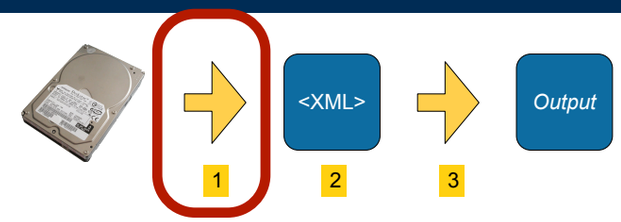
- Finds all partitions & automatically processes each.
- Handles file systems on raw device (partition-less).
- Creates a single output file with forensic data data from all.



Single program has multiple output formats:

- XML (for automated processing)
- ARFF (for data mining with Weka)
- "walk" format (easy debugging)
- SleuthKit Body File (for legacy timeline tools)
- CSV (for spreadsheets)*

fiwalk provides limited control over extraction.



Include/Exclude criteria:

- Presence/Absence of file SHA1 in a Bloom Filter
- File name matching.

```
fiwalk -n .jpeg /dev/sda          # just extract the .jpeg files
```

File System Metdata:

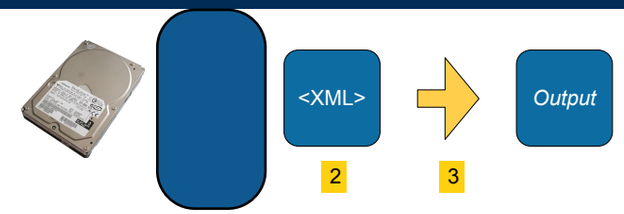
- -g — Report position of all file fragments
- -O — Do not report orphan or unallocated files

Full Content Options:

- -m — Report the MD5 of every file
- -1 — Report the SHA1 of every file
- -s *dir* — Save files to *dir*

fiwalk has a plugable metadata extraction system.

Configuration file specifies Metadata extractors:



- Currently the extractor is chosen by the file extension.

```
*.jpg   dgi     ../plugins/jpeg_extract
*.pdf   dgi     java -classpath plugins.jar Libextract_plugin
*.doc   dgi     java -classpath ../plugins/plugins.jar word_extract
```

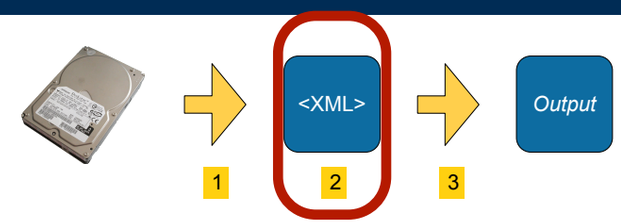
- Plugins are run in a different process for safety.
- We have designed a native JVM interface which uses IPC and 1 process.

Metadata extractors produce name:value pairs on STDOUT

```
Manufacturer: SONY
Model: CYBERSHOT
Orientation: top - left
```

Extracted metadata is automatically incorporated into output.

fiwalk produces four kinds of XML tags.



Per-Run (provenance) tags:

```
<fiwalk_version>0.4</fiwalk_version>
<Start_time>Mon Oct 13 19:12:09 2008</Start_time>
<library name="tsk" version="3.1.0b1"></library>
```

Per-Image tags:

```
<Imagefile>dosfs.dmg</Imagefile>
<volume startsector="512">
```

<volume> tags:

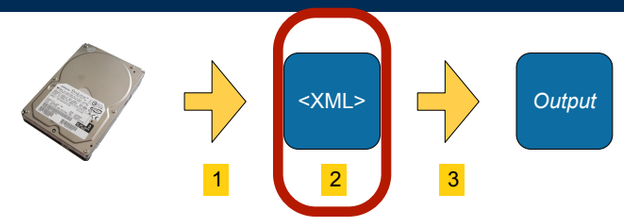
```
<Partition_Offset>512</Partition_Offset>
<block_size>512</block_size>
<ftype>4</ftype>
<ftype_str>fat16</ftype_str>
<block_count>81982</block_count>
```

<fileobject> tags:

```
<filesize>4096</filesize>
<partition>1</partition>
<filename>linedash.gif</filename>
<libmagic>GIF image data, version 89a, 410 x 143</libmagic>
```

<byte_runs> specifies data's physical location.

One or more <run> elements may be present:



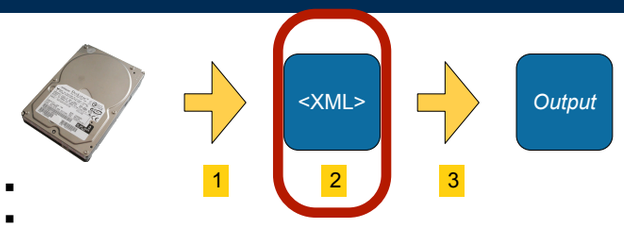

```
<byte_runs type='resident'>

  <run file_offset='0' len='65536'
       fs_offset='871588864' img_offset='871621120'/>

  <run file_offset='65536' len='25920'
       fs_offset='871748608' img_offset='871780864'/>

</byte_runs>
```

This file has two fragments:

- 64K starting at sector 1702385 (871621120 ÷ 512)
- 25,920 bytes starting at sector 1702697 (871780864 ÷ 512)

Additional XML attributes may specify compression or encryption.

XML incorporates the extracted metadata.



fiwalk metadata extractors produce name:value pairs:

```
Manufacturer: SONY
Model: CYBERSHOT
Orientation: top - left
```

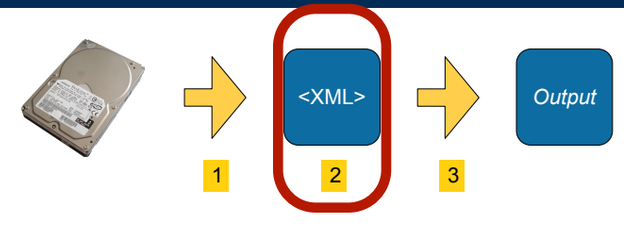
These are incorporated into XML:

```
<fileobject>
...
<Manufacturer>SONY</Manufacturer>
<Model>CYBERSHOT</Model>
<Orientation>top - left</Orientation>
...
</fileobject>
```

—*UTF-8 — Special characters are automatically escaped.*

Resulting XML files can be distributed with images.

The XML file provides a key to the disk image:

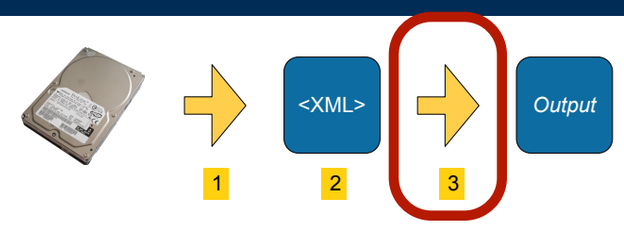



```
$ ls -l /corp/images/nps/nps-2009-domexusers/
-rw-r--r--  1 simsong  admin  4238912226 Jan 20 13:16 nps-2009-realistic.aff
-rw-r--r--  1 simsong  admin    38251423 May 10 23:58 nps-2009-realistic.xml
$
```

XML files:

- Range from 10K — 100MB.
  - —*Depending on the complexity of the disk image.*
- Only have files & orphans that are identified by SleuthKit
  - —*You can easily implement a "smart carver" that only carves unallocated sectors.*

fiwalk.py and dfxml.py: Python modules for automated forensics.



Key Features:

- Can automatically run fiwalk with correct options if given a disk image
- Reads XML file if present (faster than regenerating)
- Creates and consumes **fileobject** objects.

Multiple interfaces:

- SAX callback interface

```
fiwalk_using_sax(imagefile, xmlfile, flags, callback)
```

—*Very fast and minimal memory footprint*

- SAX procedural interface

```
objs = fileobjects_using_sax(imagefile, xmlfile, flags)
```

—*Reasonably fast; returns a list of all file objects with XML in dictionary*

- DOM procedural interface

```
(doc,objs) = fileobjects_using_dom(imagefile, xmlfile, flags)
```

—*Allows modification of XML that's returned.*

—*Slow and memory intensive.*

The SAX and DOM interfaces both return fileobjects!

The Python **`dfxml.fileobject`** class is an easy-to-use abstract class for working with file system data.

Objects belong to one of two subclasses:

```
fileobject_sax(fileobject)     # for the SAX interface
fileobject_dom(fileobject)     # for the DOM interface
```

Both classes support the same interface:

```
fi.partition()
fi.filename(), fi.ext()
fi.filesize()
fi.uid(), fi.gid(), fi.metatype(), fi.mode()
fi.ctime(), fi.atime(), fi.crtime(), fi.mtime(), fi.dtime(), fi.times()
fi.sha1(), fi.md5()
fi.byteruns(), fi.fragments()
fi.content()
fi.tempfile()
```

Don’t use DOM unless you need to *modify the DOM*.

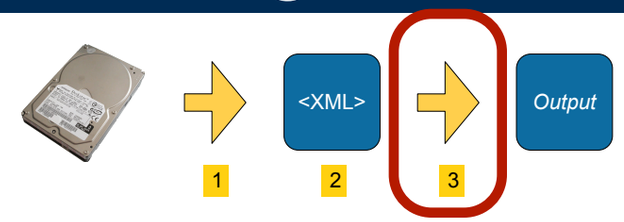
Example: igrep.py

```
import fiwalk

if __name__=="__main__":
    import sys

    from optparse import OptionParser
    parser = OptionParser()
    parser.usage = '%prog [options] image.iso  s1'
    parser.add_option("-d","--debug",help="debug",action="store_true")
    (options,args) = parser.parse_args()

if len(args)!=2:
    parser.print_help()
    sys.exit(1)

(image,data) = args

def process(fi):
    offset = fi.contents().find(data)
    if offset>0:
        print "%s (offset=%d)" % (fi.filename(),offset)

fiwalk.fiwalk_using_sax(imagefile=image),callback=process)
```

igrep.py in action

```
$ python igrep.py nps-2009-canon2-gen6.raw Firmware
DCIM/100CANON/IMG_0044.JPG (offset=1228)
DCIM/100CANON/IMG_0042.JPG (offset=1228)
DCIM/100CANON/IMG_0003.JPG (offset=1228)
DCIM/100CANON/IMG_0043.JPG (offset=1228)
DCIM/100CANON/IMG_0045.JPG (offset=1228)
DCIM/100CANON/IMG_0046.JPG (offset=1228)
DCIM/100CANON/IMG_0007.JPG (offset=1228)
DCIM/100CANON/IMG_0047.JPG (offset=1228)
DCIM/100CANON/IMG_0009.JPG (offset=1228)
DCIM/100CANON/IMG_0038.JPG (offset=1228)
DCIM/100CANON/IMG_0011.JPG (offset=1228)
DCIM/100CANON/IMG_0048.JPG (offset=1228)
DCIM/100CANON/IMG_0013.JPG (offset=1228)
DCIM/100CANON/IMG_0049.JPG (offset=1228)
DCIM/100CANON/IMG_0050.JPG (offset=1228)
DCIM/100CANON/IMG_0016.JPG (offset=1228)
DCIM/100CANON/IMG_0017.JPG (offset=1228)
DCIM/100CANON/IMG_0018.JPG (offset=1228)
DCIM/100CANON/IMG_0019.JPG (offset=1228)
DCIM/100CANON/IMG_0051.JPG (offset=1228)
DCIM/100CANON/IMG_0021.JPG (offset=1228)
DCIM/100CANON/IMG_0022.JPG (offset=1228)
DCIM/100CANON/IMG_0023.JPG (offset=1228)
DCIM/100CANON/IMG_0024.JPG (offset=1228)
DCIM/100CANON/IMG_0026.JPG (offset=1228)
```

Example:
Get all of the file objects for files < 100 bytes in length.



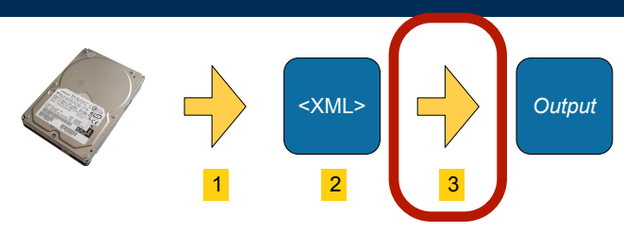
Using SAX interface:

```
import fiwalk,dfxml

imagefile = open("/corp/drives/nps/nps-2008-jean/nps-2008-jean.E01")
def process(fi):
    if fi.filesize()<100:
        print fi.filename(),fi.filesize()

fiwalk.fiwalk_using_sax(imagefile=imagefile,callback=process)
```

Produces:

```
$ python x.py
$BadClus 0
$Extend/.. 56
$Extend/$ObjId 0
$Extend/$Quota 0
$Extend/$Reparse 0
$Secure 0
$Volume 0
. 56
...
Documents and Settings/Administrator/Cookies/administrator@ads.cnn[2].txt 96
Documents and Settings/Administrator/Cookies/administrator@c.msn[2].txt 68
...
Documents and Settings/Administrator/Cookies/administrator@www.msn[1].txt 85
...
```

The fileobject class allows direct access to file data.



byteruns() is an array of “runs.”

```
<byte_runs type='resident'>

  <run file_offset='0' len='65536'
       fs_offset='871588864' img_offset='871621120'/>

  <run file_offset='65536' len='25920'
       fs_offset='871748608' img_offset='871780864'/>

</byte_runs>
```

Becomes:

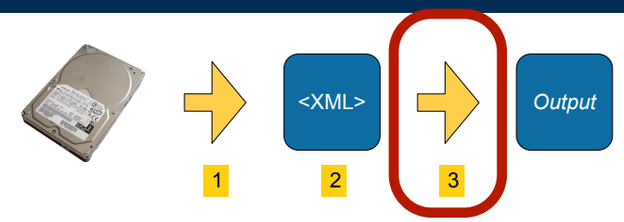
```
[byterun[offset=0; bytes=65536], byterun[offset=65536; bytes=25920]]
```

Each byterun object has:

```
run.start_sector()   – Starting Sector #
run.sector_count()
run.img_offset       - Disk Image offset
run.fs_offset        - File system offset
run.bytes            - number of bytes
run.content()        - content of file
```

The fileobject class allows direct access to file data.

byteruns() returns that array of “runs”
for both the DOM and SAX-based file objects.




```
>>> print fi.byteruns()
[byterun[offset=0; bytes=65536], byterun[offset=65536; bytes=25920]]
```

Accessor Methods:

```
fi.contents_for_run(run)       – Returns the bytes from the linked disk image
fi.contents()                  – Returns all of the contents
fi.file_present(imagefile=None)– Validates MD5/SHA1 to see if image has file
fi.tempfile(calMD5,calcSHA1)   – Creates a tempfile, optionally calc. hash
```

Question: how much time can we save in forensic analysis by processing files in *sector order?*

Currently, forensic programs process in directory order.

```
for (dirpath,dirnames,filenames) in os.walk("/mnt"):
    for filename in filenames:
        process(dirpath+"/"+filename)
```

Advantages of processing by sector order:

- Minimizes head seeks.

Disadvantages:

- Overhead to obtain file system metadata (but you only need to do it once).
- File fragmentation means you can't do a perfect job:

Using the architecture presented here, I performed the experiment.

Here's most of the program:

```
t0 = time.time()
fis = fiwalk.fileobjects_using_sax(imagefile)
t1 = time.time()
print "Time to get metadata: %g seconds" % (t1-t0)

print "Native order: "
calc_jumps(fis,"Native Order")
fis.sort(key=lambda(a):a.byteruns()[0].img_offset)
calc_jumps(fis,"Sorted Order")
```

With this XML framework, it took less than 10 minutes to write the program that conducted the experiment.

Answer: Processing files in sector order can improve performance *dramatically*.

	Unsorted	Sorted
Files processed:	23,222	23,222
backwards seeks	12,700	4,817
Time to extract metadata:	19 seconds	19 seconds
Time to read files:	441 seconds	38 seconds
Total time:	460 seconds	57 seconds

disk image: nps-2009-domexusers1

DFXML: Current Status

Working today:

- Programs for producing and consuming DFXML.
- A set of tags that can represent:
  - —*Files & Metadata*
  - —*Hashes*
  - —*Time*
  - —*Bloom Filters*
- Coming:
  - —*Windows Registry*

What we are doing with DFXML:

- Using DFXML to annotate disk images
- Using DFXML in a cluster/HPC environment.

What we need:

- Support from tool vendors (primary carvers)
- Increased use within the research community.

InaiMathi Regular: ஞ

Latin Small Letter Phi: ɸ

White Chess Queen: ♕

Snowman: ☃

Negative Circled Number Eleven: ⓫

Eject Symbol: ⏏

Arabic Letter Seen with Three Dots Below and Three Dots Above: ﭫ

Katakana Letter Zu: ズ

Understanding Unicode

We no longer live in an ASCII world.

One of the great things about DFXML is you can grep for filenames:

```
$ grep filename nps-2009-domexusers.xml |grep domexuser2
...
nps-2009-domexusers.xml:      <filename>Documents and Settings/domex2/Local
Settings/Temporary Internet Files/Content.IE5/0LYRGTUN/CASRADUP.com
%25252Fmail%25252F%25253F%2526service%253Dmail%2526ltmpl%253Ddefault%26hl
%3Den%26dEM%3Ddomexuser2</filename>
nps-2009-domexusers.xml:      <filename>Documents and Settings/domex2/Local
Settings/Temporary Internet Files/Content.IE5/W1QV09Y3/CANALGTL.com
%25252Fmail%25252F%25253F%2526service%253Dmail%2526ltmpl%253Ddefault%26hl
%3Den%26dEM%3Ddomexuser2</filename>
$
```

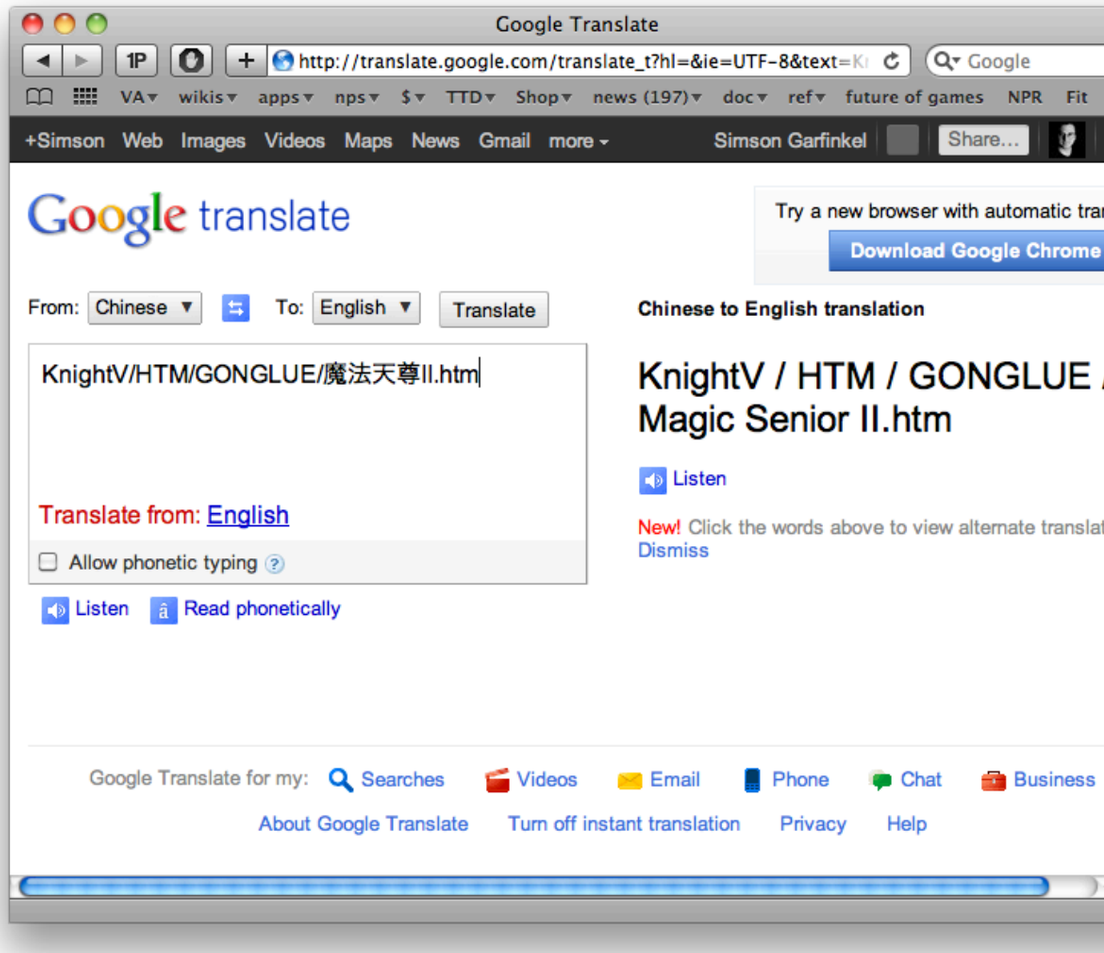
With drives from China, you sometimes see this:

```
$ grep filename cn1-1-10.xml | sed s/<filename.?>//g
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔法天尊II.htm
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔幻精灵2.htm
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔幻精灵二.htm
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔幻天下.htm
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔唤精灵.htm
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔唤精灵C.htm
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔界之泉.htm
cn1-1-10.xml:  Program Files/Kingsoft/KnightV/HTM/GONGLUE/魔神战记2.htm
```

When you see words like 魔法天尊, you ask questions...

What does KnightV/HTM/GONGLUE/魔法天尊II.htm mean?

- Use Google Translate:



How can 魔法天尊 display in Terminal?

How can I copy&paste 魔法天尊?

Counting the bytes just adds to one's confusion.

Let's put it in a file.

- The Kanji behaves like regular characters:

```
$ cat "KnightV/HTM/GONGLUE/魔法天尊II.htm" > filename.txt

$ ls -l filename.txt
-rw-r--r--  1 simsong  staff  39 Jul 16 19:47 filename.txt
$ cat filename.txt
KnightV/HTM/GONGLUE/魔法天尊II.htm
```

- But if you count the number of characters, there are only:
  - *22 Letters ("KnightVHTMGONGLUEIIhtm")*
  - *+ 3 slashes ("///")*
  - *+ 1 period (".")*
  - *+ 1 newline ("\n")*
  - *+ 4 Kanji ("魔法天尊")*
  - *= 30 characters, not 39*
- You only get 39 if you count 3 bytes for each Kanji.
  - *22+3+1+1+4*3 = 39*

Looking at the byte level is more confusing still.

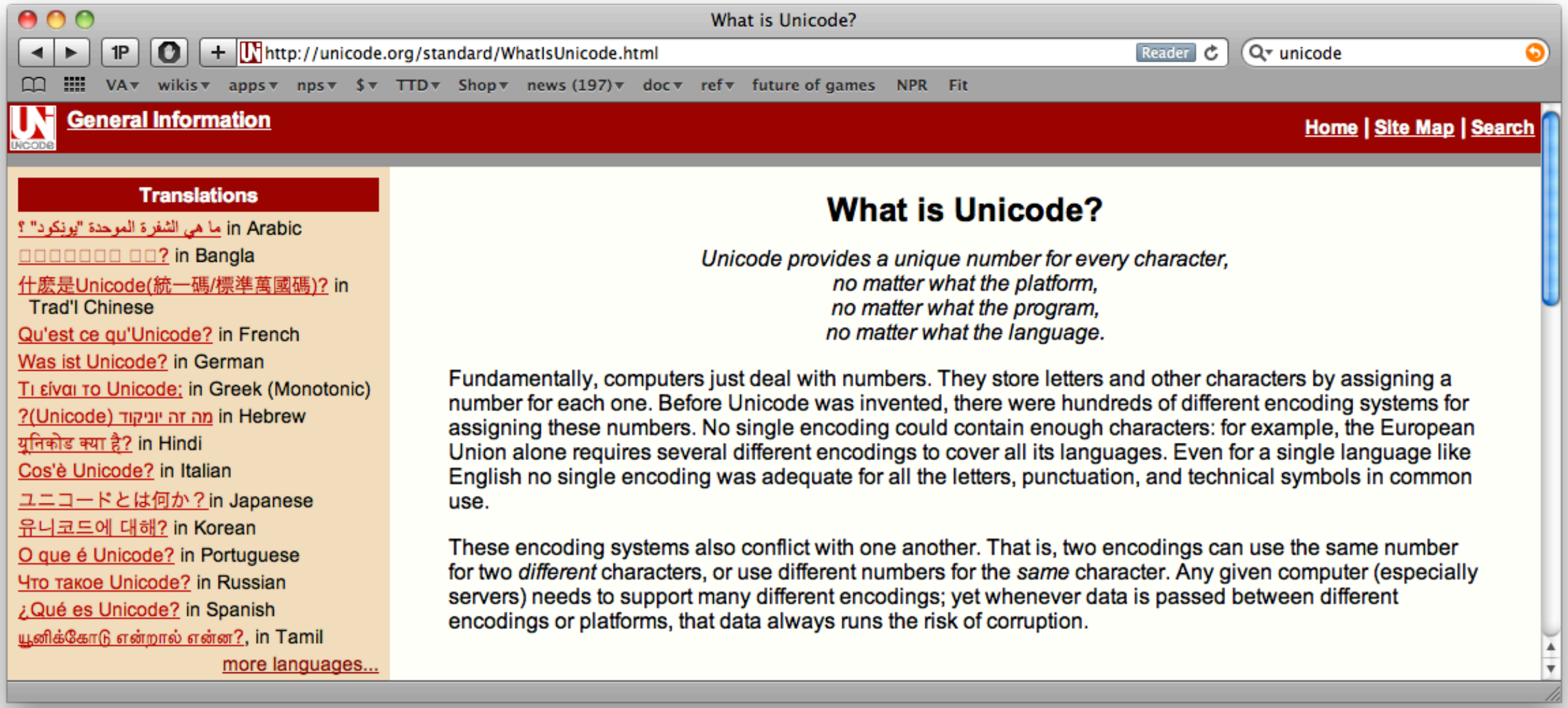
```
$ cat filename.txt
KnightV/HTM/GONGLUE/魔法天尊II.htm
$ xxd filename.txt
0000000: 4b6e 6967 6874 562f 4854 4d2f 474f 4e47  KnightV/HTM/GONG
0000010: 4c55 452f e9ad 94e6 b395 e5a4 a9e5 b08a  LUE/............
0000020: 4949 2e68 746d 0a                        II.htm.
$
```

Assuming 3 bytes per Kanji, we get:

```
魔 = e9 ad 94
法 = e6 b3 95
天 = e5 a4 a9
尊 = e5 b0 8a
```

魔法天尊 are examples of Unicode characters.

Unicode was created as a single system to represent all the characters of all the world's languages.



Currently we are on Unicode version 6.0.

A "Code" is a system for converting one piece of information to another.

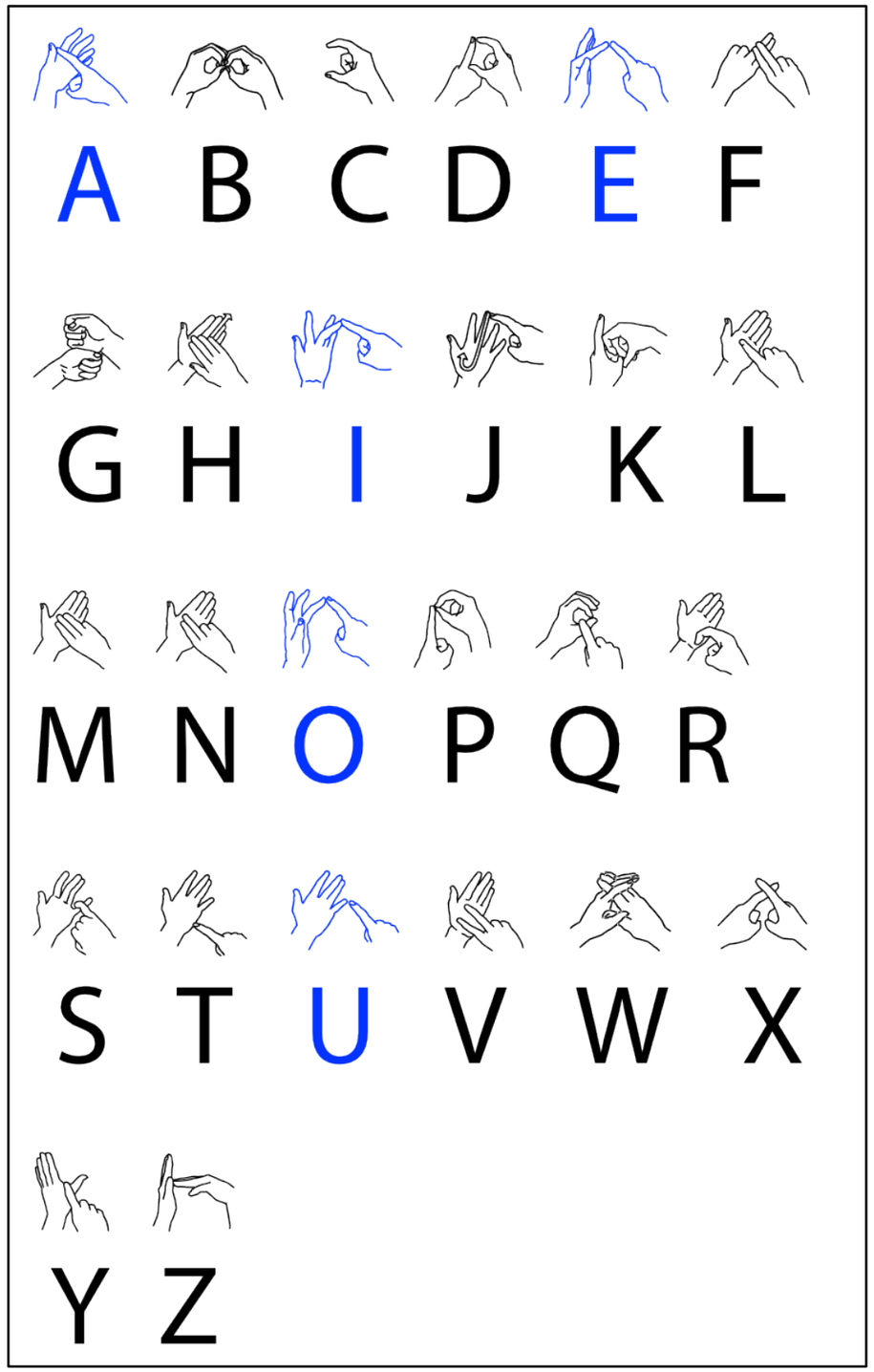
There are many codes:

- "Modem Codes" — Values that a modem returns to identify itself.
- ASCII — American Standard Code for Information Interchange
- Unicode — Modern interchange code.

  —*Note: These days codes are rarely used for security because they are easily broken.*

A *code book* is a list of codes and their meanings.

In computing, a *code point* is a particular number and its graphical representation.



American Standard Code for Information Interchange (ASCII) was developed in the early 1960s.

ASCII improved on codes in use at the time (e.g. Baudot):

- UPPERCASE and lowercase letters.
- No “shift” character to switch from letters to numbers.
- Many more symbols — !”#$%&’ ()*+,-./:;<=>?@[\]^_`{|}~
- 32 control characters — many to support interactive computing.
- 7-level code allows 1 bit for parity on 8-bit systems.

ASR 33
7-bit ASCII

The decimal set:

0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	del

```
0-25 = ^A through ^Z
   1 = 0000 0001 = ^A
  65 = 0010 0001 = A
  96 = 0110 0001 = a
```

7-bit ASCII was used on 8-bit computers.

ASCII was a 7-bit code.

- By the 1980s, transmission was reliable enough to use all 8 bits without parity.
- Bottom 128 codes were usually ASCII.
- The top 128 codes were used for different purposes in different areas:
  - —*European accents.*
  - —*Line drawing*
  - —*Greek, Cyrillic, Hebrew, etc.*

When IBM introduced the IBM PC in 1981, it used “Code Page 437” for the top 128 characters.

IBM gave each region of the world a different code page.

	–0	–1	–2	–3	–4	–5	–6	–7	–8	–9	–A	–B	–C	–D	–E	–F
8–	Ç 00C7 *128*	ü 00FC *129*	é 00E9 *130*	â 00E2 *131*	ä 00E4 *132*	à 00E0 *133*	å 00E5 *134*	ç 00E7 *135*	ê 00EA *136*	ë 00EB *137*	è 00E8 *138*	ï 00EF *139*	î 00EE *140*	ì 00EC *141*	Ä 00C4 *142*	Å 00C5 *143*
9–	É 00C9 *144*	æ 00E6 *145*	Æ 00C6 *146*	ô 00F4 *147*	ö 00F6 *148*	ò 00F2 *149*	û 00FB *150*	ù 00F9 *151*	ÿ 00FF *152*	Ö 00D6 *153*	Ü 00DC *154*	ø 00F8 *155*	£ 00A3 *156*	Ø 00D8 *157*	× 00D7 *158*	ƒ 0192 *159*
A–	á 00E1 *160*	í 00ED *161*	ó 00F3 *162*	ú 00FA *163*	ñ 00F1 *164*	Ñ 00D1 *165*	ª 00AA *166*	º 00BA *167*	¿ 00BF *168*	® 00AE *169*	¬ 00AC *170*	½ 00BD *171*	¼ 00BC *172*	¡ 00A1 *173*	« 00AB *174*	» 00BB *175*
B–	░ 2591 *176*	▒ 2592 *177*	▓ 2593 *178*	│ 2502 *179*	┤ 2524 *180*	Á 00C1 *181*	Â 00C2 *182*	À 00C0 *183*	© 00A9 *184*	╣ 2563 *185*	║ 2551 *186*	╗ 2557 *187*	╝ 255D *188*	¢ 00A2 *189*	¥ 00A5 *190*	┐ 2510 *191*
C–	└ 2514 *192*	┴ 2534 *193*	┬ 252C *194*	├ 251C *195*	─ 2500 *196*	┼ 253C *197*	ã 00E3 *198*	Ã 00C3 *199*	╚ 255A *200*	╔ 2554 *201*	╩ 2569 *202*	╦ 2566 *203*	╠ 2560 *204*	═ 2550 *205*	╬ 256C *206*	¤ 00A4 *207*
D–	ð 00F0 *208*	Ð 00D0 *209*	Ê 00CA *210*	Ë 00CB *211*	È 00C8 *212*	ı 0131 *213*	Í 00CD *214*	Î 00CE *215*	Ï 00CF *216*	┘ 2518 *217*	┌ 250C *218*	█ 2588 *219*	▄ 2584 *220*	¦ 00A6 *221*	Ì 00CC *222*	▀ 2580 *223*
E–	Ó 00D3 *224*	ß 00DF *225*	Ô 00D4 *226*	Ò 00D2 *227*	õ 00F5 *228*	Õ 00D5 *229*	µ 00B5 *230*	þ 00FE *231*	Þ 00DE *232*	Ú 00DA *233*	Û 00DB *234*	Ù 00D9 *235*	ý 00FD *236*	Ý 00DD *237*	¯ 00AF *238*	´ 00B4 *239*
F–	SHY 00AD *240*	± 00B1 *241*	‗ 2017 *242*	¾ 00BE *243*	¶ 00B6 *244*	§ 00A7 *245*	÷ 00F7 *246*	¸ 00B8 *247*	° 00B0 *248*	¨ 00A8 *249*	· 00B7 *250*	¹ 00B9 *251*	³ 00B3 *252*	² 00B2 *253*	■ 25A0 *254*	NBSP 00A0 *255*

Codes 0-127: ASCII

Codes 128-255: Localized Codes

Code page 850: Latin 1

Code page 737: Greek:

	–0	–1	–2	–3	–4	–5	–6	–7	–8	–9	–A	–B	–C	–D	–E	–F
9–	Ρ 3A1	Σ 3A3	Τ 3A4	Υ 3A5	Φ 3A6	Χ 3A7	Ψ 3A8	Ω 3A9	α 3B1	β 3B2	γ 3B3	δ 3B4	ε 3B5	ζ 3B6	η 3B7	θ 3B8
A–	ι 3B9	κ 3BA	λ 3BB	μ 3BC	ν 3BD	ξ 3BE	ο 3BF	π 3C0	ρ 3C1	σ 3C3	ς 3C2	τ 3C4	υ 3C5	φ 3C6	χ 3C7	ψ 3C8
B–	░ 2591	▒ 2592	▓ 2593	│ 2502	┤ 2524	╡ 2561	╢ 2562	╖ 2556	╕ 2555	╣ 2563	║ 2551	╗ 2557	╝ 255D	╜ 255C	╛ 255B	┐ 2510
C–	└ 2514	┴ 2534	┬ 252C	├ 251C	─ 2500	┼ 253C	╞ 255E	╟ 255F	╚ 255A	╔ 2554	╩ 2569	╦ 2566	╠ 2560	═ 2550	╬ 256C	╧ 2567
D–	╨ 2568	╤ 2564	╥ 2565	╙ 2559	╘ 2558	╒ 2552	╓ 2553	╫ 256B	╪ 256A	┘ 2518	┌ 250C	█ 2588	▄ 2584	▌ 258C	▐ 2590	▀ 2580
E–	ω 3C9	ά 3AC	έ 3AD	ή 3AE	ϊ 3CA	ί 3AF	ό 3CC	ύ 3CD	ϋ 3CB	ώ 3CE	Ά 386	Έ 388	Ή 389	Ί 38A	Ό 38C	Ύ 38E
F–	Ώ 38F	± B1	≥ 2265	≤ 2264	Ϊ 3AA	Ϋ 3AB	÷ F7	≈ 2248	° B0	∙ 2219	· B7	√ 221A	ⁿ 207F	² B2	■ 25A0	A0

Code pages complicate processing because different code pages show the same text differently!

There are many code pages:

- 437 — Original IBM PC code page
- 737 — Greek
- 775 — Estonian, Lithuanian and Latvian
- 850 — "Multilingual (Latin-1)" (Western European languages)
- 852 — "Slavic (Latin-2)" (Central and Eastern European languages)
- …

This text in code page 437:

naïve

Becomes this text in code page 737:

"naMve"

—*Note: that "M" is character code 8B; it is not an "M" (code 4D)*

Problems with code pages:

- No intrinsic coding of current code page.
- Hard to get symbols from multiple code pages.
- No obvious way to handle Chinese, Japanese, Korean, or Vietnamese (CJKV)
- Lack of standardization
- Some vendors implemented "shift."

Unicode was developed as a single coding standard for all of the world's languages

Project started in 1987 at Xerox and Apple.

- Originally called for 16-bit characters (limit of 65,535 symbols)
- Expanded to handle code points 0 through 10FFFF (1,114,112 total) to cover ancient languages.

Goals:

- Compatibility with existing systems.
- Clean "round trip" to legacy codings.
- Stability.
- No "shift" characters.
- Code for *graphemes*, not *glyphs* (e.g., '𝒶' and 'a' have the same unicode—U+0061)
- "Han unification" — A single set of characters for identical kanji in Chinese, Japanese, Korean, and Cantonese

Unicode Today

96,000+ characters in Unicode v4 (more in v5)

Used:

- In every major operating system
- In most office programs
- In XML, HTML, etc.
- In Java, C++, C#

Problems:

- Implementations are incomplete
- Not all programmers have implemented all the rules.
- Multiple codings (UTF-8, UTF-16) mean that code that works sometimes with some codings doesn't work other times with other codings.

Today Unicode 6.0 is widely used.

Unicode has 1,114,112 *code points* ranging from 0 to 10FFFF.

Most Unicode characters are 16-bit characters.

- U+0041 is "A" "LATIN CAPITAL LETTER A." *Just like ASCII*
- U+0042 is "B" *Just like ASCII*
- U+0495 is "ક" *Gujarati letter KA*
- U+20AC is "€" *Euro*
- U+FE4A is "‾" *Centerline Overline*

Unicode 4.0 has characters for *every living human language.*

- *Arabic* العربية *left-to-right*
- *Hebrew* עִבְרִית
- *Japanese* 日本語

Unicode 5.0 added support for dead languages.

- Excellent demo online at http://www.fileformat.info/info/unicode/

Unicode is divided into 17 planes, each with 65,536 code points.

Only a few code points are actually used:

Plane	Range	Name
0	U+0000 to U+FFFF	Basic multilingual Plane (BMP)
1	U+10000 to U+1FFFF	Supplementary Multilingual Plane (SMP)
2	U+20000 to U+2FFFF	Supplementary Ideographic Plane (SIP)
3 - 13	*Unassigned*	
14	U+E0000 to U+EFFFF	Supplementary Special-purpose Plane (SSP)
15	U+F0000 to U+FFFFF	Private Use Area (PUA)
16	U+100000 to U +10FFFF	Private Use Area (PUA)

Each Unicode code point can be represented with multiple encodings.

Most Unicode text is encoded as UTF-8

- Variable-length code; ASCII characters code as ASCII
- Arabic, Armenian, Cyrillic, Coptic, Greek, Syriac &Tāna: 2 characters
- Chinese, Japanese, Korean & Vietnamese: 3 characters
- Other: 4 (or more)

Unicode	Byte1	Byte2	Byte3	Byte4	example
U+0000-U+007F	0*xxxxxx*				'$' U+0024 → 0**0100100** → 0x24
U+0080-U+07FF	110*yyyxx*	10*xxxxxx*			'¢' U+00A2 → 110**00010**,10**100010** → 0xC2,0xA2
U+0800-U+FFFF	1110*yyyy*	10*yyyyxx*	10*xxxxxx*		'€' U+20AC → 1110**0010**,10**000010**,10**101100** → 0xE2,0x82,0xAC
U+10000-U+10FFFF	11110*zzz*	10*zzyyyy*	10*yyyyxx*	10*xxxxxx*	U+10ABCD → 11110**100**,10**001010**,10**101111**,10**001101** → 0xF4,0x8A,0xAF,0x8D

http://en.wikipedia.org/wiki/UTF-8

Most "modern" web pages download with UTF-8

www.apple.com:

Source of: http://www.apple.com/startpage/

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
<html lang="en-us">
<head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <meta http-equiv="pics-label" content='(pics-1.1 "http://www.icra.org/rati
        <meta name="Author" content="Apple Inc.">
        <meta name="viewport" content="width=1024">
        <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7">
        <title>Apple - Start</title>
        <meta name="omni_page" content="Start - Index">
        <meta name="Category" content="">
        <meta name="Description" content="">
        <link rel="alternate" href="http://images.apple.com/main/rss/hotnews/hotnev
        <link rel="stylesheet" href="http://images.apple.com/global/styles/base_itv
        <link rel="stylesheet" href="http://images.apple.com/startpage/styles/start
</head>
<body>
        <script src="http://images.apple.com/global/nav/scripts/shortcuts.js" type
<script type="text/javascript" charset="utf-8">
        var searchSection = 'global';
        var searchCountry = 'us';
</script>
<div id="globalheader">
        <!--googleoff: all-->
        <ul id="globalnav">
                <li id="gn-apple"><a href="/">Apple</a></li>
```

Find: UTF | Next | Previous | Highlight all | Match case

Line 10, Col 49

UTF-16 codes most characters as 2 bytes. UTF-16 is the original Unicode representation

Widely used by:

- Microsoft (in memory and on disk) for filenames
- Text in *some* Microsoft documents.
- Web pages authored in Chinese and Japanese.

Code plans 1 through 16 are encoded with U+D800 to U+DBFF

- Character U+10000 becomes 0xD800 0xDC00

Beware:

- UTF-16 can be coded two ways—big endian or little-endian.
- The Byte Order Mark (Zero-Width No-Break Space) U+FEFF is used at the beginning of a file to specifies byte order:

  —*big-endian — FE FF*

  —*little-endian — FF FE*
- If text is accompanied with encoding of UTF-16BE or UTF-16LE, BOM is ignored.
- Forensic data rarely has a BOM—and it can be wrong.

Many problems araise with Unicode.

Legacy problems:

- Implementations are incomplete
- Not all programmers have implemented all the rules.
- Multiple codings (UTF-8, UTF-16) mean that code that works sometimes with some codings doesn't work other times with other codings.

Ongoing problems

- Behavior of strings becomes complex and may depend on the locale.
- Complex rules for:
  - —*case conversion (toUpper(), toLower(), toTitle())*
  - —*String comparison (isUpper(), isLower(), isTitle())*

Complex rules for:

- bidi
- coalition
- line and paragraph breaks (U+2028 LINE SEPARATOR and U+2029 PARAGRAPH SEPARATOR)
- search/string matching

Consider Arabic:
There are multiple unicode glyphs for the same letter.

Different versions are used in different applications.

- For editing, the *general* form is used.
- For printing, the *isolated, final, medial* or *initial* forms might be used.
- For searching, *any form* needs to match

Each form has a different character code.

General: م

Isolated: م

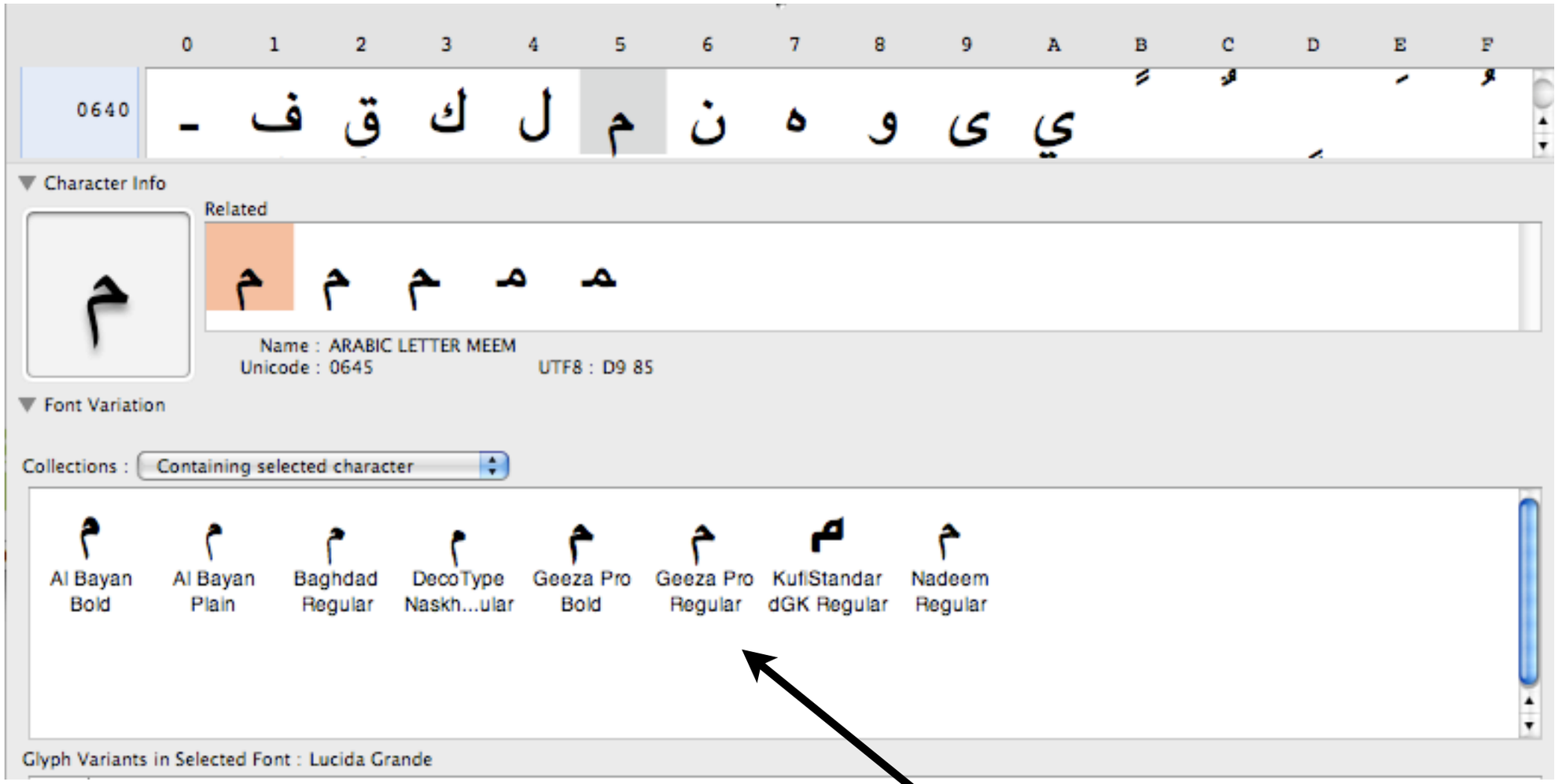
Final: ـم

Medial: ـمـ

Initial: مـ

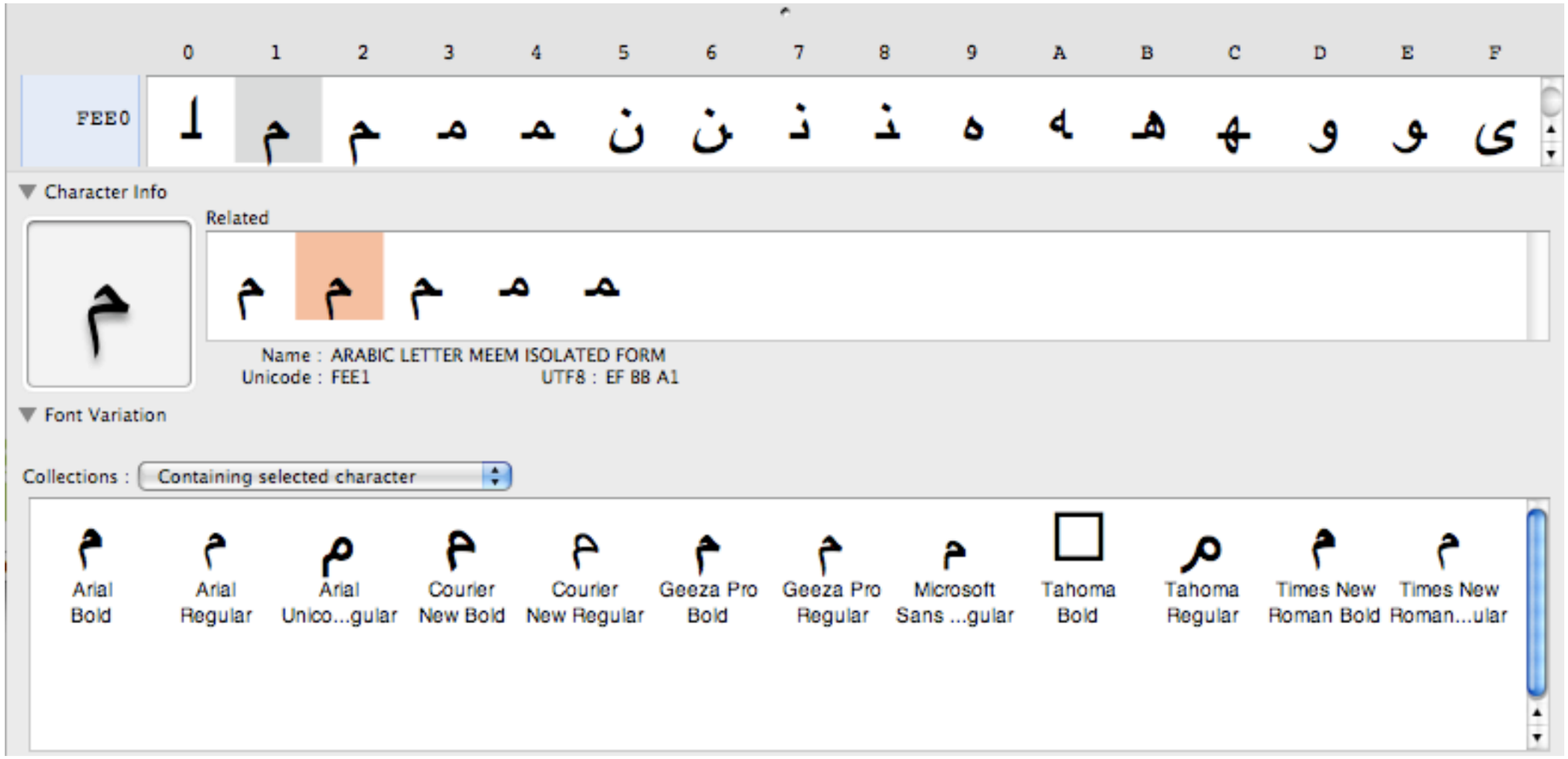
Arabic MEEM

U+0645: ARABIC LETTER MEEM

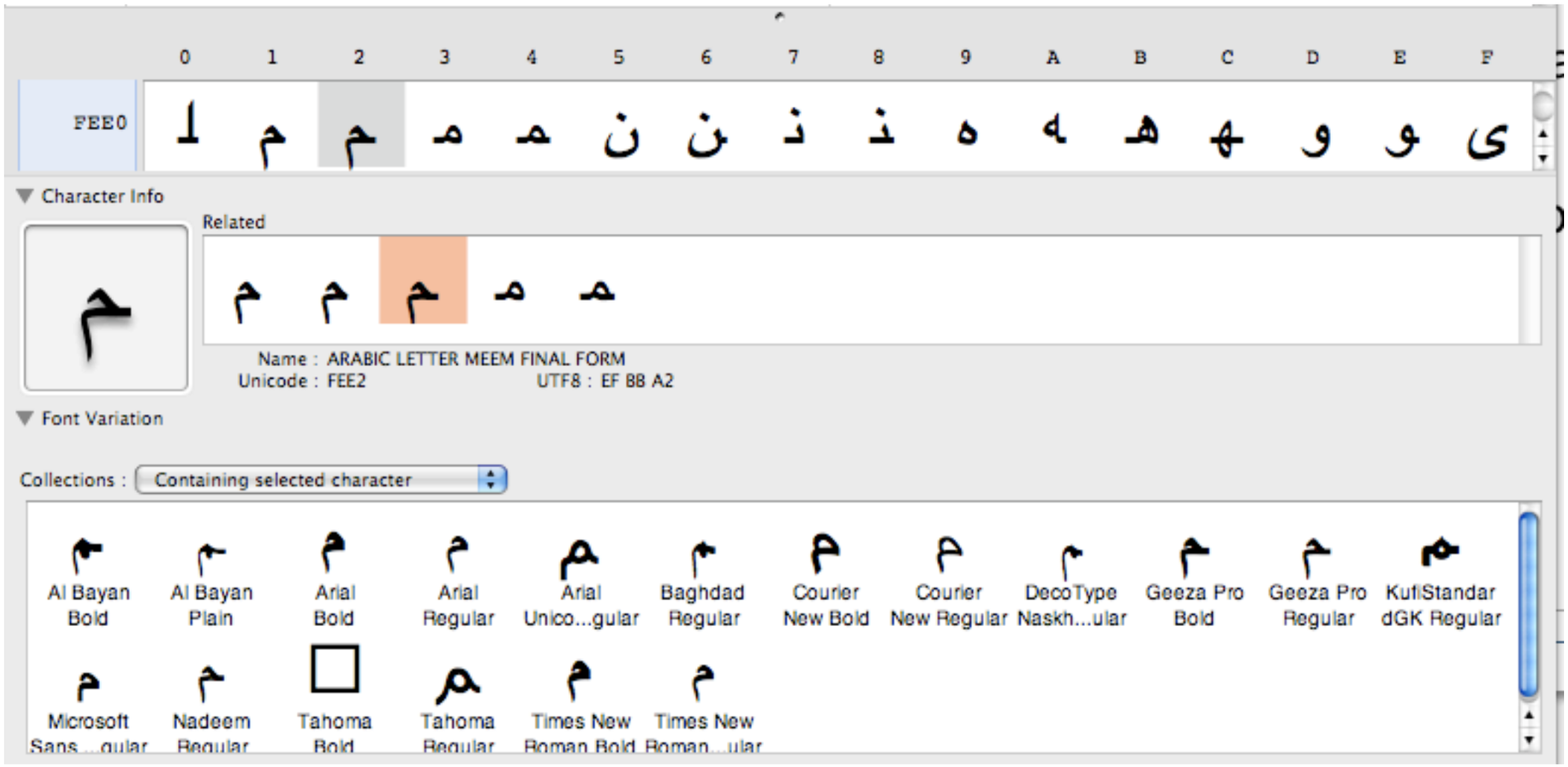


Different fonts render the MEEM differently!

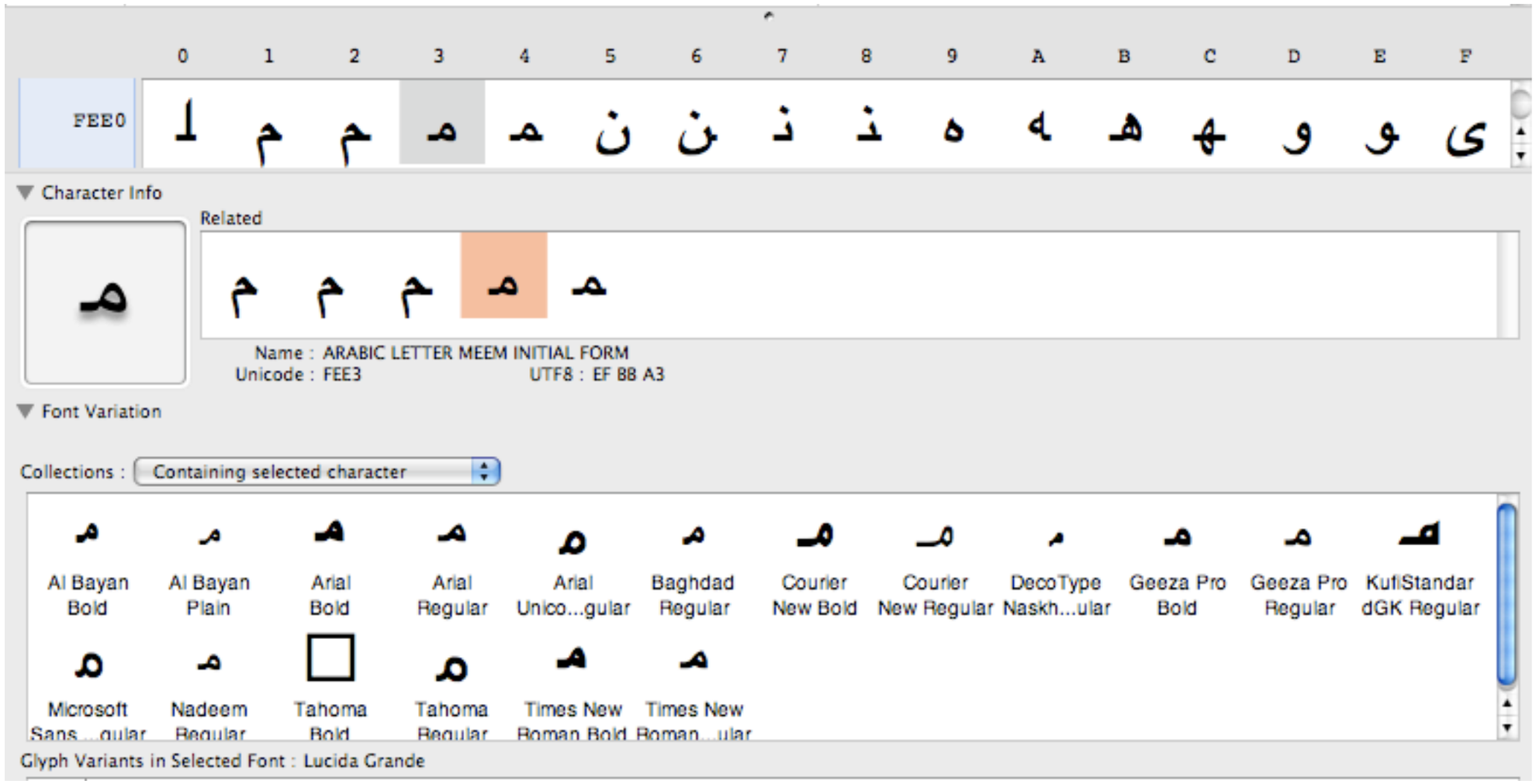
U+FEE1: MEEM ISOLATED FORM



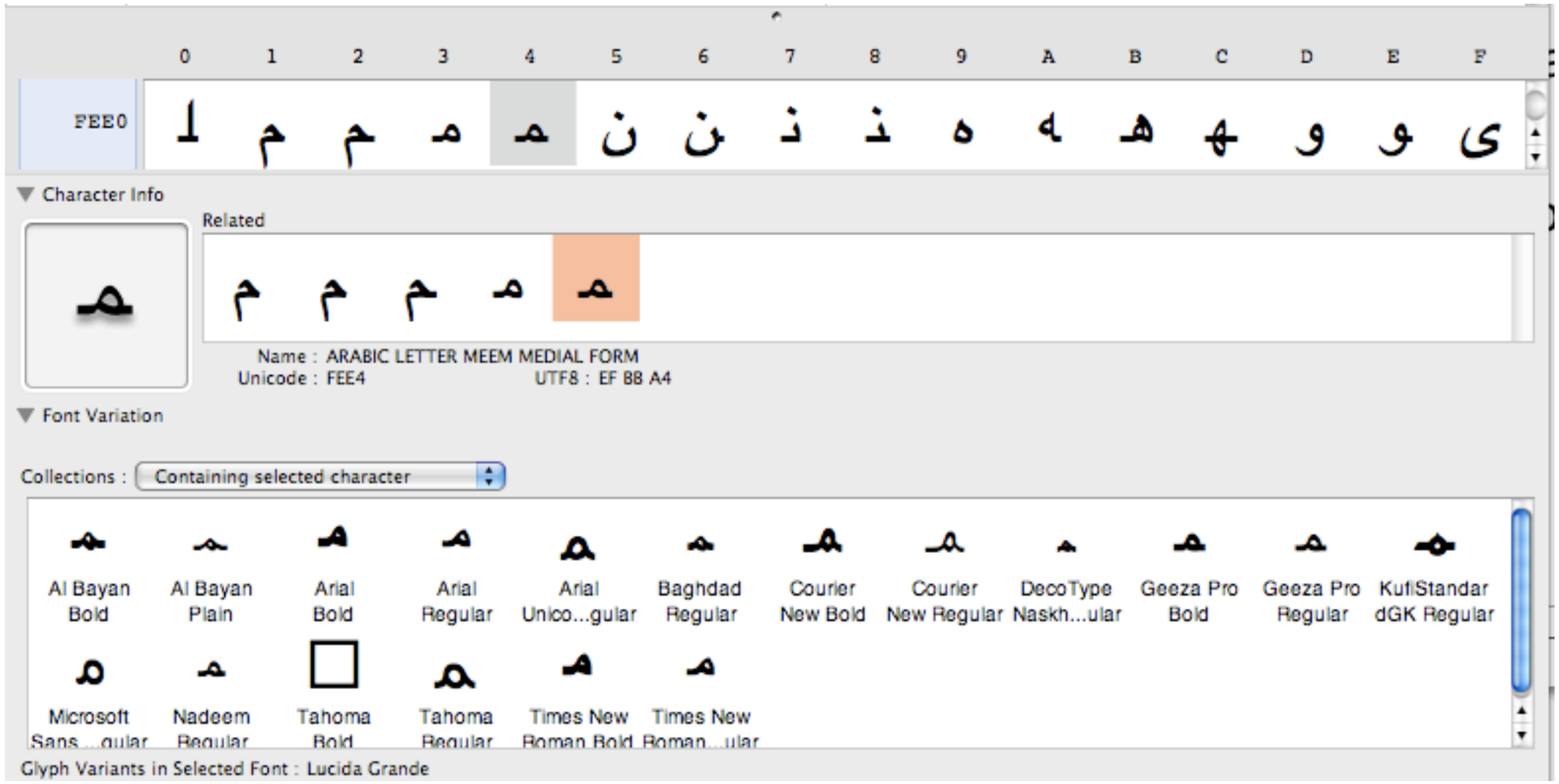
U+FEE2: MEEM FINAL FORM



U+FEE3: MEEM INITIAL FORM



U+FEE4: MEEM MEDIAL FORM



Unicode characters with diacritical marks can be constructed with 1 code point or two.

The ñ character can be coded two ways:

- U+00F1: ñ (LATIN SMALL LETTER N WITH TIDLE)

or:

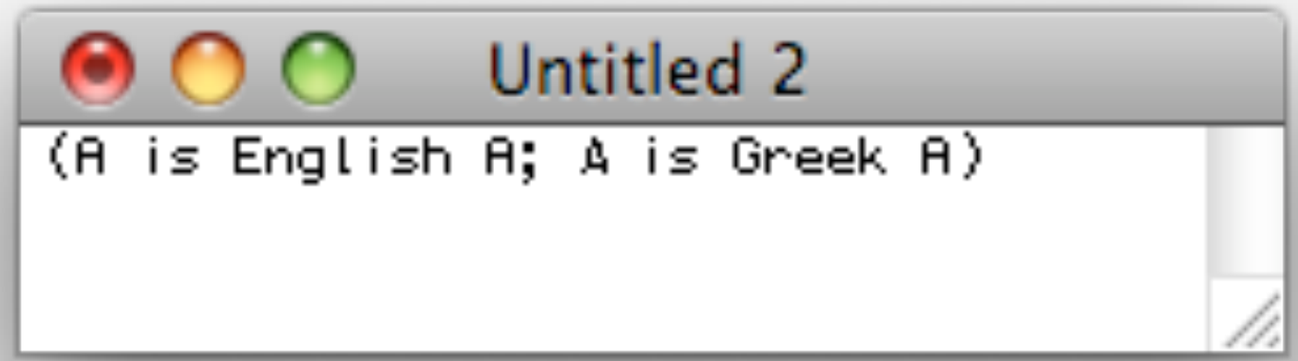
- U+0303: ~ (COMBINING TILDE)
- U+006E: n (LATIN SMALL LETTER N)

This creates 10 different byte sequences that display as ñ:

- U+00F1 in UTF-8, UTF-16LE, UTF-16BE, UTF-32BE, UTF32-LE
- U+0303 and U+006E UTF-8, UTF-16LE, UTF-16BE, UTF-32BE, UTF32-LE

Unicode has HCI-SEC problems.

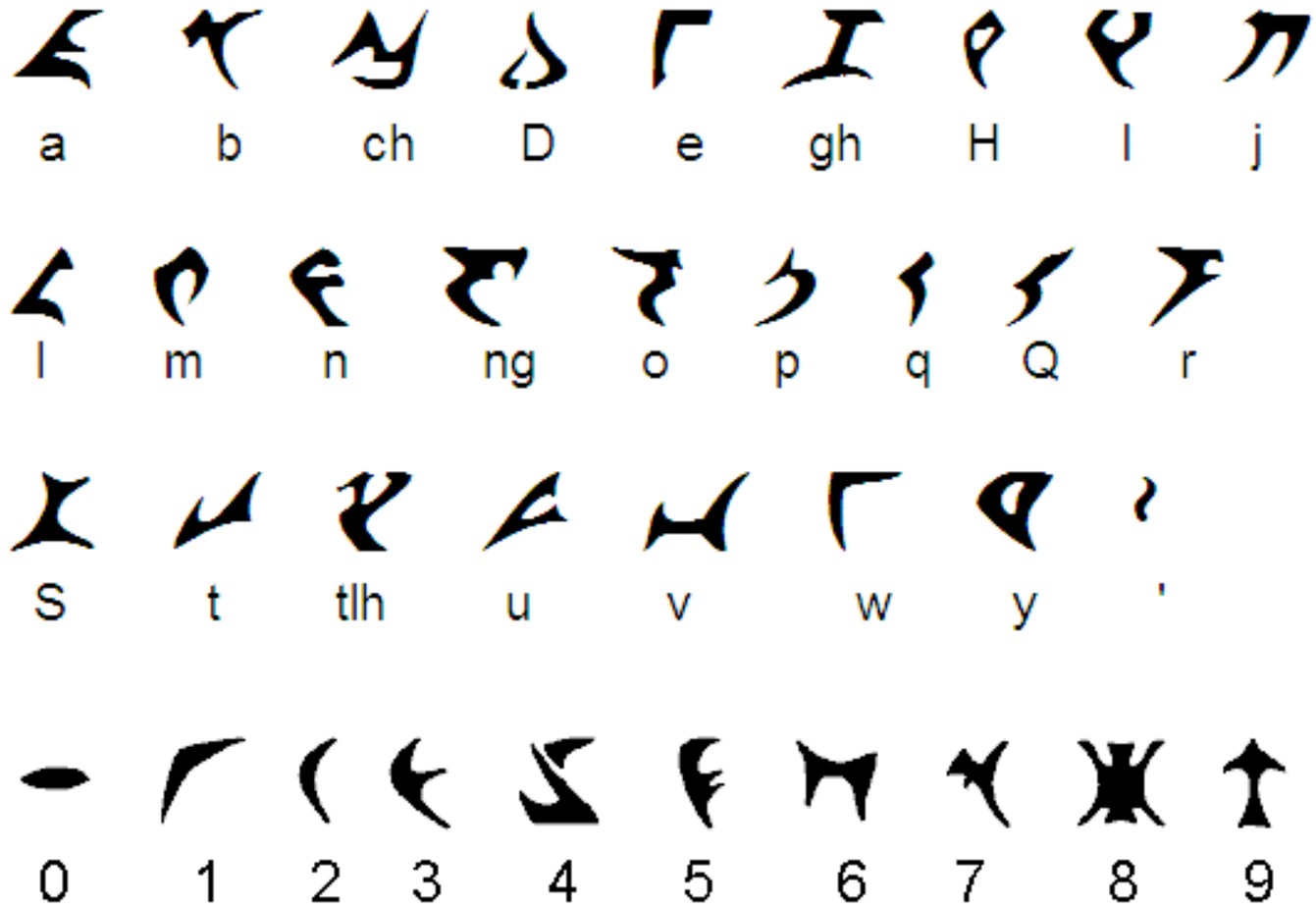
A and A are different characters (A is English A; A is Greek A)



This leads to both database problems and phishing attacks.

Unicode has *political* problems.

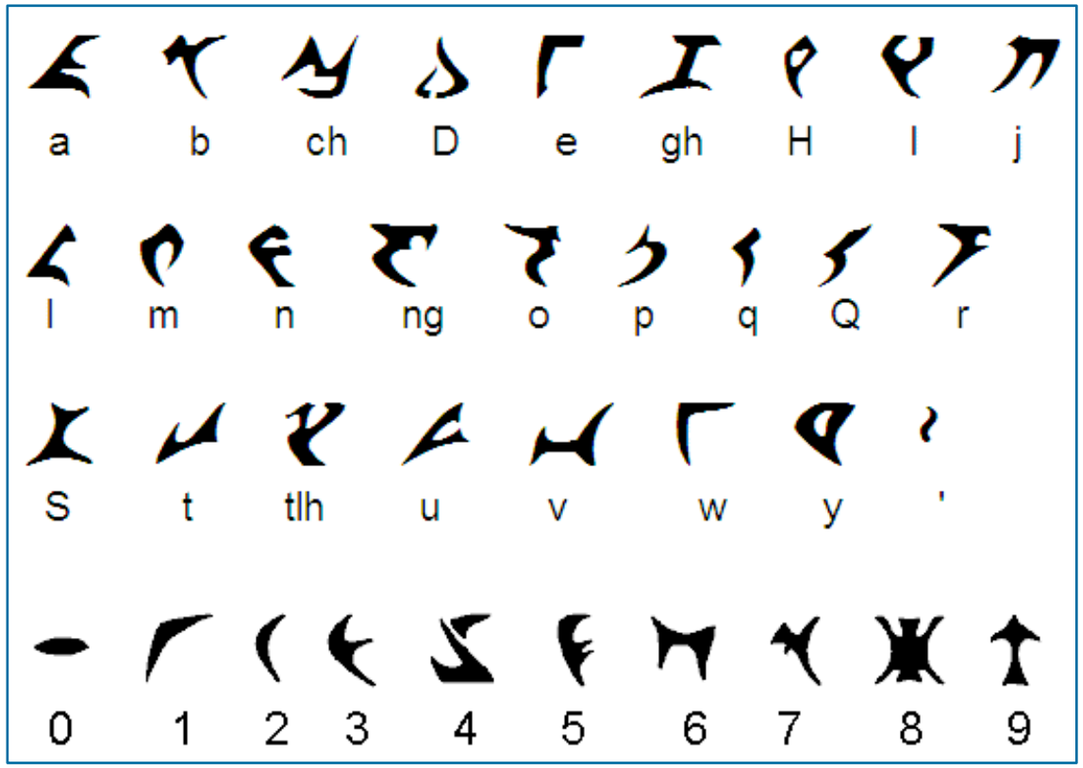
What should we do about this language?



Who decides what's a language?

Invented languages like Klingon.

- *The Klingon Dictionary* sold 300,000 copies.



Dying languages.

Dead languages.

Most of these problems matter less with the expanded code space that Unicode 5.0 and 6.0 provides.

Things to know as a programmer...

Behavior is complex and may depend on the locale. Use built-in functions for:

- case conversion
- String comparison
- isUpper(), isLower(), isTitle()
- toUpper(), toLower(), toTitle(), etc.
- Unicode strings should know about valid line-breaks

In forensics, we frequently have *invalid* Unicode codings:

```
>>> print unicode(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xff in position 0:
ordinal not in range(128)
```

I generally have all programs output in UTF-8 and escape or suppress it as necessary:

```
def make_utf8(s):
    # Try the most basic conversion first. If they work, we return
    try:
        if type(s)==unicode:
            return s.replace("\\","\\\\").encode('utf-8')
        elif type(s)==str:
            return unicode(s,'utf-8').replace("\\","\\\\")
        else:
            return unicode(str(s),'utf-8').replace("\\","\\\\")
    except UnicodeDecodeError:
        pass
    except UnicodeEncodeError:
        pass
    ret = ""          # Didn't work, so convert character-by-character
    for ch in s:
        if(ord(ch)<ord(' ')):
            ret += "\\%03o" % ord(ch)
            continue
        try:
            ret += ch.encode('utf-8').replace("\\","\\\\")
            continue
        except UnicodeDecodeError:
            pass
        except UnicodeEncodeError:
            pass
        ret += "\\%03o" % ord(ch)
    return ret
```

So let's look at 魔法天尊 again.

```
$ cat filename.txt
KnightV/HTM/GONGLUE/魔法天尊II.htm
$ xxd filename.txt
0000000: 4b6e 6967 6874 562f 4854 4d2f 474f 4e47  KnightV/HTM/GONG
0000010: 4c55 452f e9ad 94e6 b395 e5a4 a9e5 b08a  LUE/............
0000020: 4949 2e68 746d 0a                        II.htm.
$
```

Those 3 bytes per Kanji were UTF-8:

```
魔 = e9 ad 94
法 = e6 b3 95
天 = e5 a4 a9
尊 = e5 b0 8a
```

The letters could also be coded in UTF-16 or UTF-32:

```
魔 = U+9B54 = 9b 54 = 00 00 9b 54 = demon, evil spirits; magic power
法 = U+6CD5 = 6c d5 = 00 00 6c d5 = law, rule, regulation, statute; France,
天 = U+5929 = 59 29 = 00 00 59 29 = sky, heaven; god, celestial
尊 = U+5C0A = 5c 0a = 00 00 5c 0a = respect, revere, venerate; honor
```

I created those demos with Python

```
>>> "魔".decode('utf-8')
u'\u9b54'
```

“u” is the Python Unicode character type.

```
>>> a = "法"
>>> print a,type(a)
法 <type 'str'>
>>> b = a.decode('utf-8')
>>> print b,type(b)
法 <type 'unicode'>
>>> print b.encode('utf-16')
???l
>>> print a.encode('utf-16')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xe6 in position 0:
ordinal not in range(128)
>>> print len(a)
3
>>> print hex(ord(a[0])),hex(ord(a[1])),hex(ord(a[2]))
0xe6 0xb3 0x95
>>>
```

In summary

ASCII and Unicode:

- Display is easier than search
- Information may not display correctly, and you may not know it.

For further information:

- http://www.unicode.org/standard/principles.html
- http://www.unicode.org/versions/Unicode5.1.0/
- http://www.unicode.org/notes/tn23/
- http://www.unicode.org/faq/
- http://macchiato.com/slides/UnicodeMyths.pdf
- http://unicode.org/standard/tutorial-info.html

Memory Forensics

What was *really* happening on the subject's computer?

RAM analysis

The computer's RAM may contain:

- Discoverable evidence (e.g. logfiles, documents)
- Encryption keys
- Current network connections; Some kinds of malware
  - —*"Cold boot" attack lets you move memory between computers.*

Windows and MacOS deny access to memory for security.

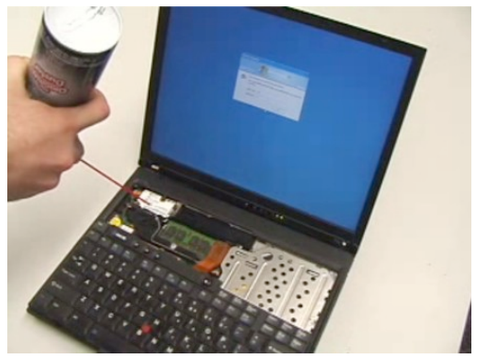
- A current list of tools is at: http://www.forensicswiki.org/wiki/Tools:Memory_Imaging

Linux memory acquisition is relatively easy

- Just read /dev/mem; /dev/kmem

Write the RAM to:

- Server on the Network (best)
- External storage (USB stick—leaves evidence under Windows)
- File on the same system (worst)



Control of memory *is* control of the computer.

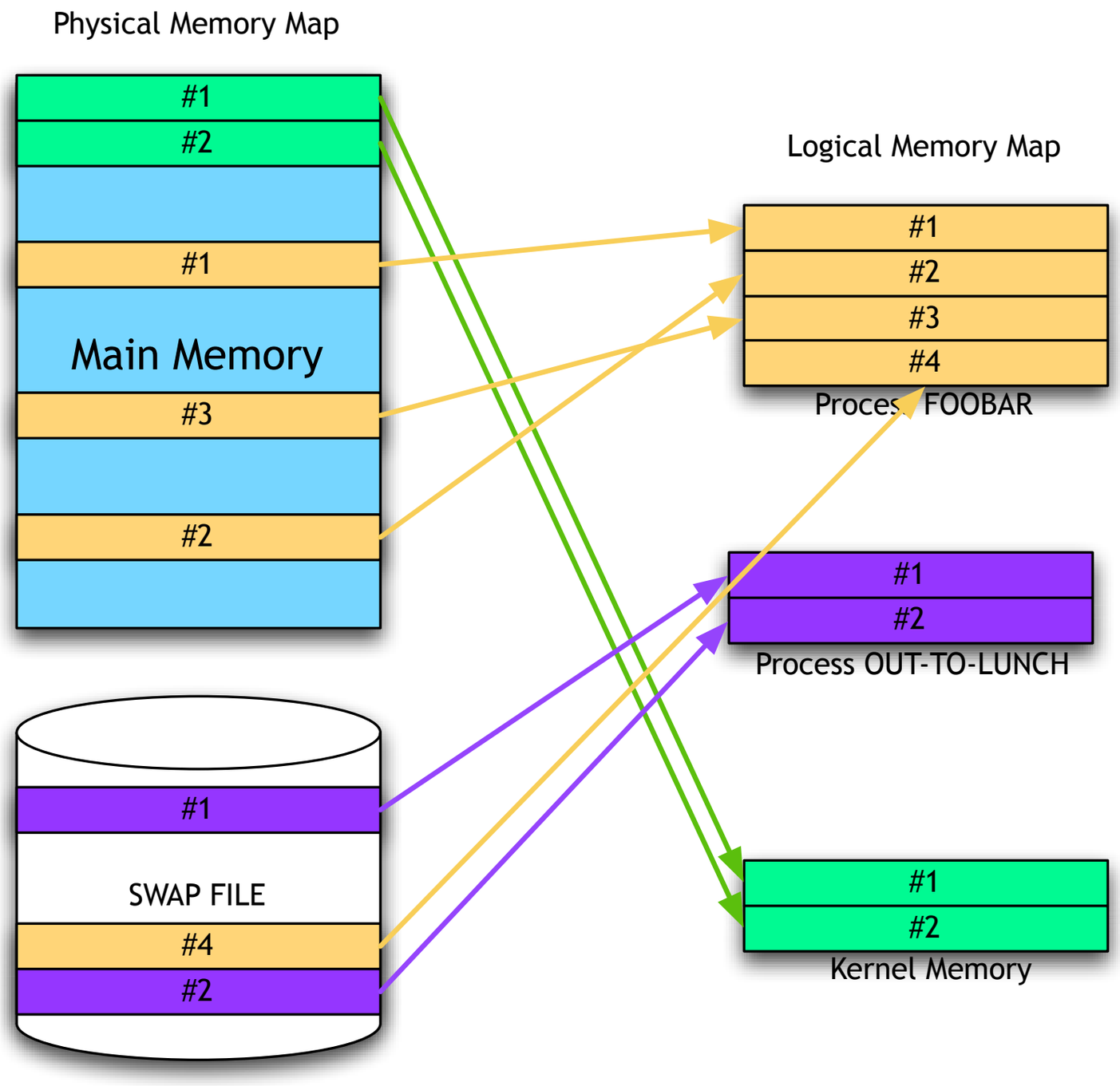
Reading:

- Contents of the screen
- Cryptographic Keys
- Passwords (BIOS & programs)
- Current Running Programs
- Remnants of previously run programs
- Open TCP/UDP ports
- Cached data
- Hidden data

Writing:

- Patch programs on the fly
- Change security levels
- Install malware

Two memory views: "Physical" and "Logical"



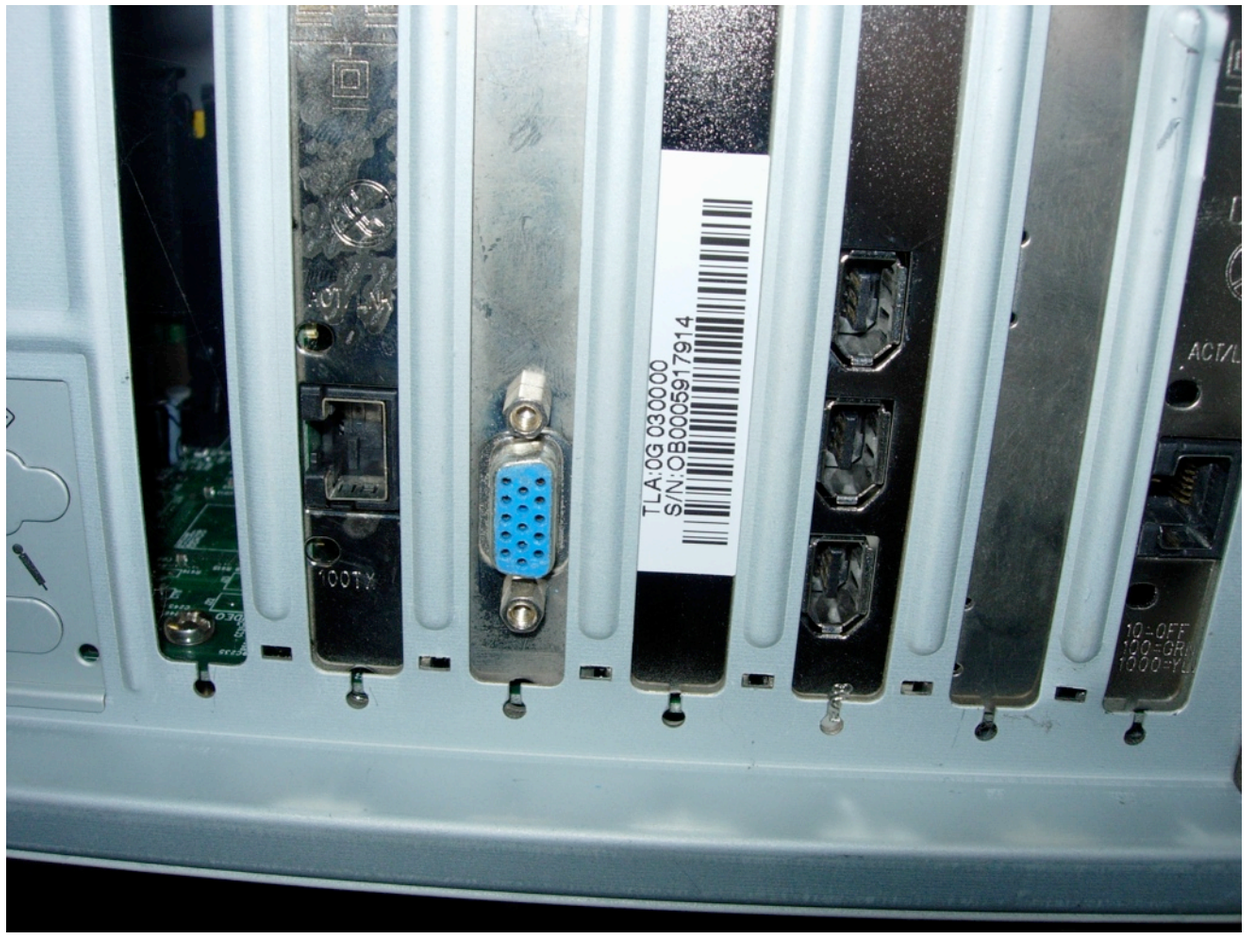
Memory can be acquired "LIVE" or "DEAD"

Swap space on live or dead systems

- PAGEFILE.SYS
- /private/var/vm/swapfile
- Swap Partitions
- *Suspend/Resume*

Live Memory:

- /dev/mem
- /proc/kcore
- \\.\PhysicalMemory
- \\.\DebugMemory
- Device Drivers
- Special programs (WinEn)
- Hardware memory imagers
- Firewire (provides DMA)



Options for RAM acquisition.

Hardware Acquisition (very hard to do)

- Special-purpose PCI card
- Firewire / PATA / SATA
- Cold Boot Attack

Software Acquisition

- User-level program (Windows XP2)
- User-level program with device driver (Windows XP3, Vista, Windows 7)

Hibernation Files and Virtual Machines

- hiberfil.sys
- VMWare stores "Ram" in ***FILENAME*.vmem**

Potential problems with acquiring live memory:

Speed:

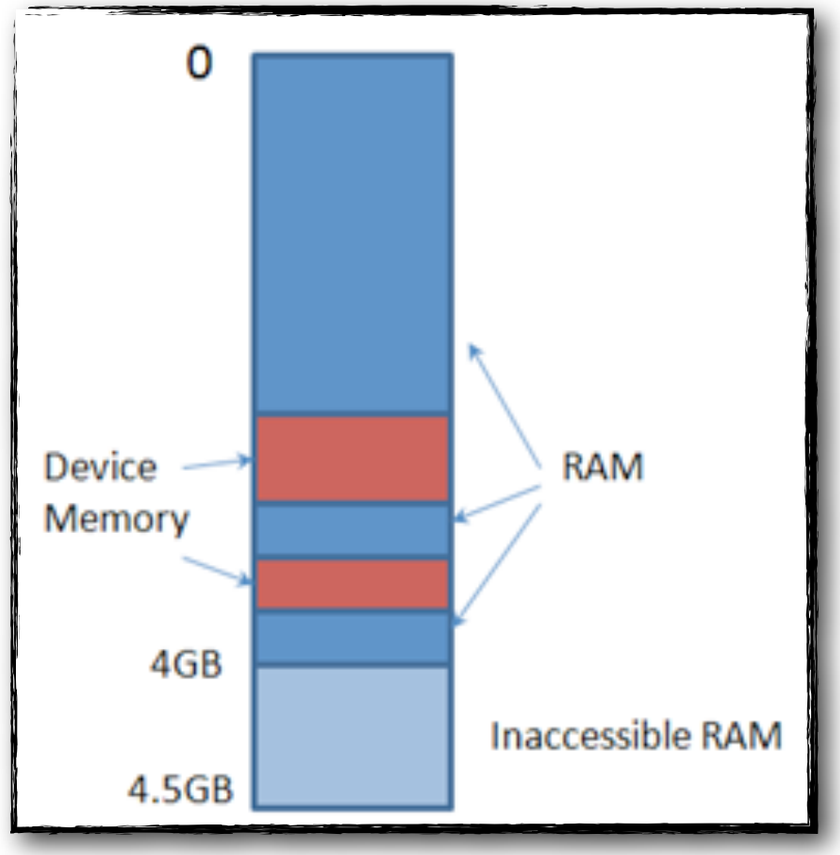
- Memory changes fast; it won't be consistent.

Availability:

- Software methods can be blocked by attacker.

Integrity:

- Software changes the memory map
- You can't get all the memory



Memory Analysis Techniques

Look for ASCII and UNICODE strings.

- strings(1), grep

"File carving"

- foremost, scalpel
- Princeton key search program

Identify and interpret kernel or program data structures

- Convert Windows memory image to Microsoft crashdump format, then analyze with standard debugging tools (WinDbg):
  - —*http://computer.forensikblog.de/en/2006/03/dmp_file_structure.html*
  - —*http://www.shakacon.org/talks/NFI-Shakacon-win32dd0.3.pdf*
- KnTTools (George Garner)
- Volatility, by Volatile Systems (http://wwwvolatilesystems.com)
- Idetect

KnTTools (Windows), by George M. Garner, Jr.

KNTDD - Acquires memory

- Acquisition to removable drive or network
- Cryptographic integrity checks, auditing
- Conversion to Microsoft crash dump format
- Remote deployment as a service

KnTList - Lists Kernel Structures

- Reconstructs virtual address space
- Drives, Device Objects, System Tables
- Threads, access tokens, handle table, objects, etc.
- Outputs as text and XML

http://forensic.seccure.net/

http://gmgsystemsinc.com/knttools/

WMFT - Windows Memory Forensic Toolkit

Enumerates processes, modules, libraries

Finds hidden data (including some rootkits)

Detailed information:

- Access tokens
- Handles
- Processes
- Modules

http://forensic.seccure.net/

Idetect (Linux)

Displays detailed information for each process

Enumerates all process-related structures

Can work on memory image or live system

- http://forensic.seccure.net/tools/idetect.tar.gz
- http://forensic.seccure.net/pdf/mburdach_digital_forensics_of_physical_memory.pdf

Lots more information about memory forensics, including 53-page presentation:

- http://forensic.seccure.net (2006)

Volatility: An open source tool for analyzing windows memory dumps

Created by Aaron Walters and Nick L. Petroni

- Open Source (unlike prior systems)
- Written in Python

Extracts:

- Image date & time
- Memory map for each running process
- Network sockets
- DLLs loaded for each process
- Lots more.

https://www.volatilesystems.com/VolatileWeb/volatility.gsp

http://volatility.tumblr.com/

My Windows machine has 512MB of RAM:

```
-r-xr-xr-x    1 simsong  simsong       1013 Oct  5 15:22 image.dd_audit.log
-r-xr-xr-x    1 simsong  simsong  536866816 Oct  5 15:22 image.dd
-r-xr-xr-x    1 simsong  simsong         73 Oct  5 15:22 image.dd.md5
```

```
$ cat image.dd.md5
74eb8e6cdaa43589e0b27449bd7ac03f [\\\\.\\PhysicalMemory] *v:\\image.dd
$
```

Volatility commands:

```
$ python volatility
Supported Commands:
        connections     Print list of open connections
        connscan        Scan for connection objects
        datetime        Get date/time information for image
        dlllist         Print list of loaded dlls for each process
        files           Print list of open files for each process
        ident           Identify image properties such as DTB and VM type
        modules         Print list of loaded modules
        pslist          Print list of running processes
        psscan          Scan for EPROCESS objects
        sockets         Print list of open sockets
        sockscan        Scan for socket objects
        strings         Match physical offsets to virtual addresses
        thrdscan        Scan for ETHREAD objects
        vaddump         Dump the Vad sections to files
        vadinfo         Dump the VAD info
        vadwalk         Walk the vad tree
```

volatility pslist -f *filename*: See the processes

```
$ python volatility pslist -f winxp.mem
Name                 Pid    PPid   Thds   Hnds   Time
System               4      0      57     187    Thu Jan 01 00:00:00 1970
smss.exe             612    4      3      19     Wed Aug 13 00:09:58 2008
csrss.exe            660    612    12     370    Wed Aug 13 00:10:01 2008
winlogon.exe         684    612    18     519    Wed Aug 13 00:10:02 2008
services.exe         728    684    16     269    Wed Aug 13 00:10:02 2008
lsass.exe            740    684    20     344    Wed Aug 13 00:10:02 2008
vmacthlp.exe         888    728    1      25     Wed Aug 13 00:10:03 2008
svchost.exe          904    728    17     196    Wed Aug 13 00:10:03 2008
svchost.exe          1020   728    10     269    Wed Aug 13 00:10:05 2008
svchost.exe          1056   728    55     1237   Wed Aug 13 00:10:06 2008
svchost.exe          1200   728    4      73     Wed Aug 13 00:10:07 2008
svchost.exe          1364   728    15     212    Wed Aug 13 00:10:13 2008
spoolsv.exe          1496   728    11     117    Wed Aug 13 00:10:15 2008
VMwareService.e      1796   728    4      139    Wed Aug 13 00:10:16 2008
searchindexer.e      1976   728    20     678    Wed Aug 13 00:10:17 2008
wscntfy.exe          276    1056   1      37     Wed Aug 13 00:10:22 2008
explorer.exe         480    456    13     351    Wed Aug 13 00:10:23 2008
VMwareTray.exe       548    480    1      37     Wed Aug 13 00:10:24 2008
VMwareUser.exe       556    480    3      184    Wed Aug 13 00:10:24 2008
Eraser.exe           572    480    3      90     Wed Aug 13 00:10:24 2008
ctfmon.exe           580    480    1      71     Wed Aug 13 00:10:24 2008
WindowsSearch.e      704    480    10     238    Wed Aug 13 00:10:25 2008
alg.exe              1108   728    6      105    Wed Aug 13 00:10:26 2008
imapi.exe            1336   728    5      118    Wed Aug 13 00:10:29 2008
ftk.exe              568    480    7      293    Wed Aug 13 00:24:39 2008
```

volatility files -f *filename*: See the open files

```
$ python volatility pslist -f winxp.mem
Pid: 4
File   \pagefile.sys
File   \Documents and Settings\NetworkService\NTUSER.DAT
File   \WINDOWS\system32\config\SECURITY
File   \WINDOWS\system32\config\software
File   \WINDOWS\system32\config\SECURITY.LOG
File   \Documents and Settings\NetworkService\ntuser.dat.LOG
File   \WINDOWS\system32\config\software.LOG
File   \WINDOWS\system32\config\system
File   \WINDOWS\system32\config\system.LOG
File   \WINDOWS\system32\config\default
File   \WINDOWS\system32\config\default.LOG
File   \WINDOWS\system32\config\SAM
File   \WINDOWS\system32\config\SAM.LOG
File   \Documents and Settings\NetworkService\Local Settings\Application Data
\Microsoft\Windows\UsrClass.dat
File   \Documents and Settings\NetworkService\Local Settings\Application Data
\Microsoft\Windows\UsrClass.dat.LOG
File   \Documents and Settings\Administrator\NTUSER.DAT
File   \Documents and Settings\Administrator\Local Settings\Application Data
\Microsoft\Windows\UsrClass.dat.LOG
File   \
File   \Documents and Settings\Administrator\ntuser.dat.LOG
File   \Documents and Settings\Administrator\Local Settings\Application Data
\Microsoft\Windows\UsrClass.dat
```

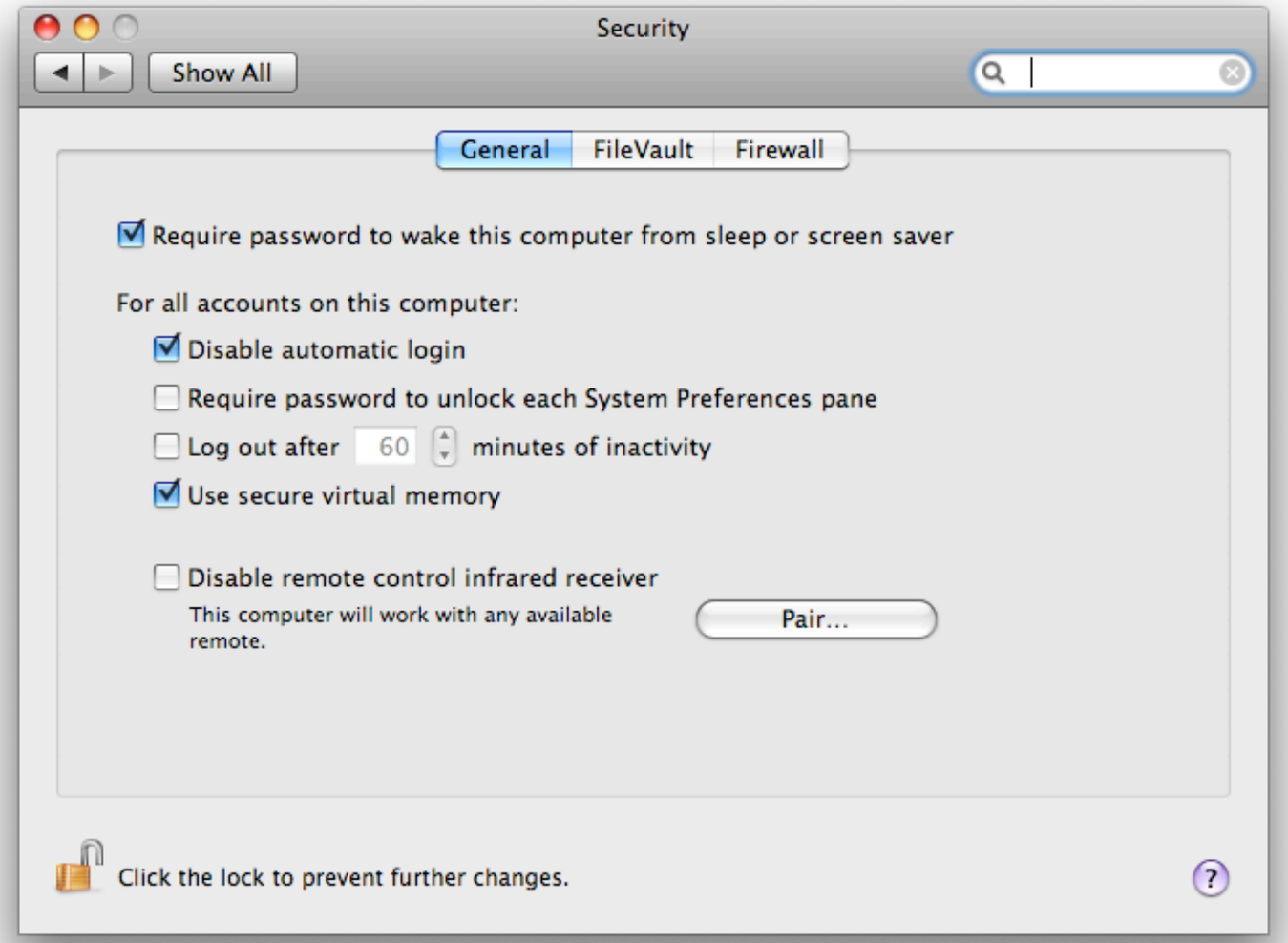
Use strings(1) to find the printable strings...

```
$ strings image.dd | grep ..............|head -10
Invalid partition ta
r loading operating system
Missing operating system
X509_REQ_add1_attr_by_txt
X509_REQ_add_extensions
X509_REQ_add_extensions_nid
X509_REQ_check_private_key
X509_REQ_delete_attr
X509_REQ_digest
```

Use strings(1) detect JPEG files...

```
$ strings filename.jpg
JFIF
ICC_PROFILE
appl
mntrRGB XYZ
acspAPPL
appl
-appl
rXYZ
gXYZ
...

$ strings image.dd | grep -i JFIF | head -10
JFIF
JFIF
.jfif:
.jfif
HKLM,"%PATH_ALLOWEDIMGEXTS%",".jfif",0x10001,0x1
ijjgijggjfifjjgijijjjjjigjijgjiiijijjiiiffjijjjjjjijjijijjiijiijjjjiigfijjjjjijjjjjjgi
jjjj0
JFIF
JFIF
.jfif:
HKCR,".jfif",,,"pjpegfile"
$
```

Don't believe MacOS "Secure Virtual Memory"




```
$ ls -l /private/var/vm
total 4259840
-rw------T  1 root  wheel  4294967296 May  3 13:51 sleepimage
-rw------T  1 root  wheel    67108864 May  4 00:08 swapfile0
```

Summary: Memory Forensics

Memory forensics analysis:

- Analysis of live memory & suspended memory
- Bulk analysis & high-level analysis

Advantages:

- Gets around disk encryption
- No systems have encrypted memory (yet)

Disadvantages:

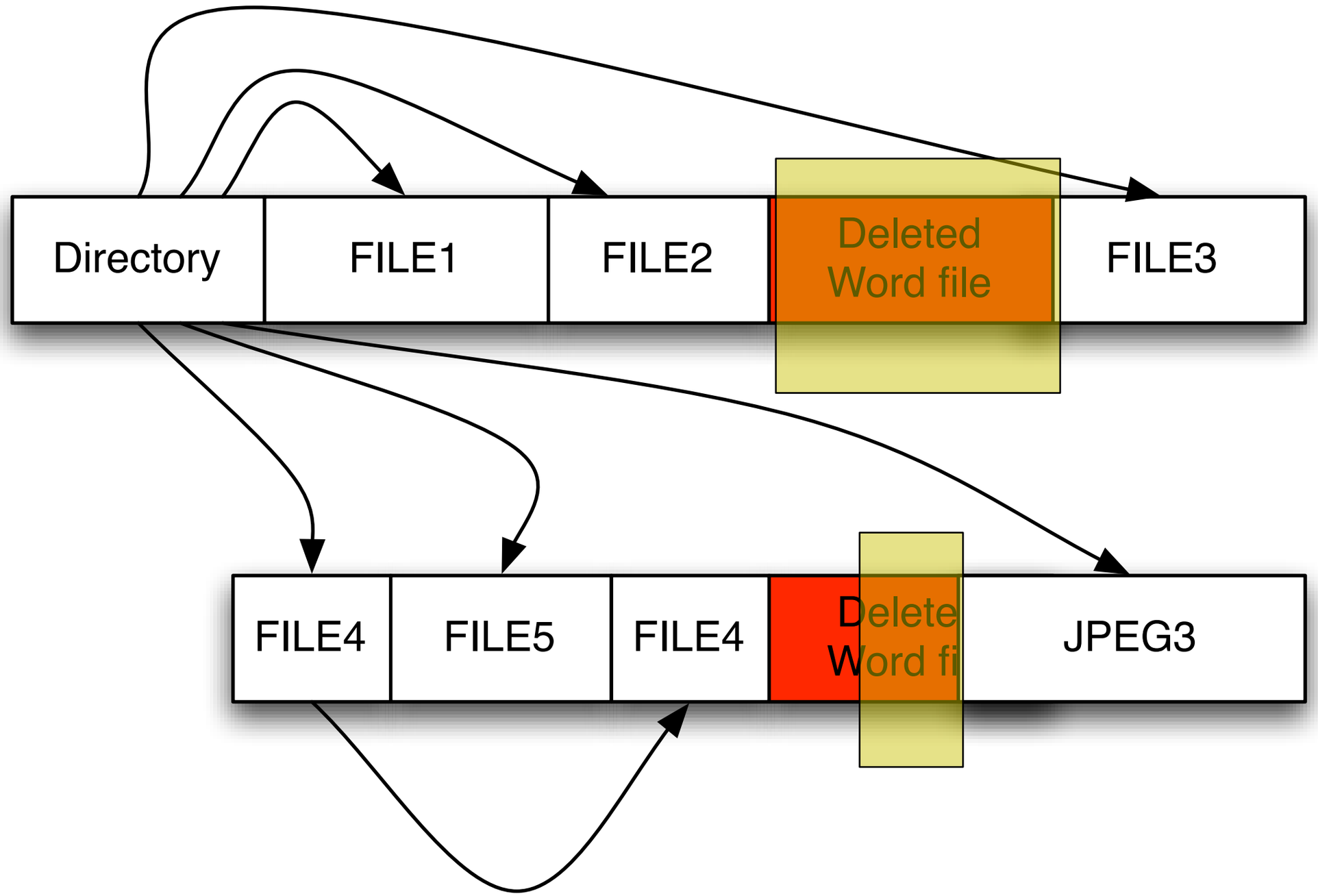
- Operating system specific.
- Tools are very primitive, but getting better.

See also:

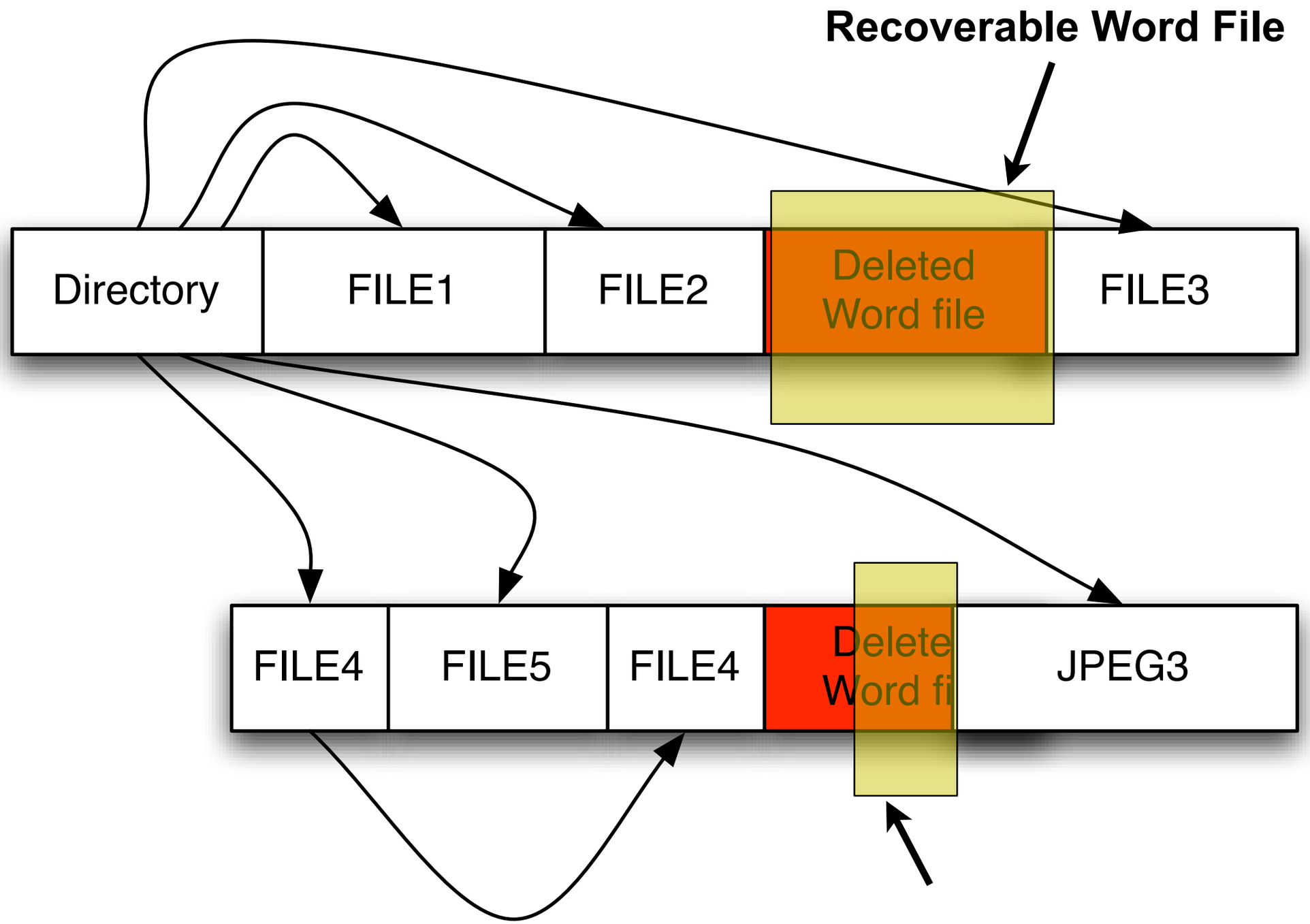
- http://www.forensicswiki.org/wiki/Windows_Memory_Analysis

Carving

“Carving” searches for objects based on content, rather than on metadata.



“Carving” searches for objects based on content, rather than on metadata.



Carving is a powerful tool for finding useful pieces of information.

What can be carved:

- Disks & Disk Images
- Memory
- Files of unknown format (to find embedded objects)

Objects that can be recovered:

- Images
- Text files & documents
- Cryptographic Keys
- Email addresses, Credit Card, Numbers, etc.

Why carve?

- Directory entries are overwritten
- Directory entries are damaged
- File formats aren’t known

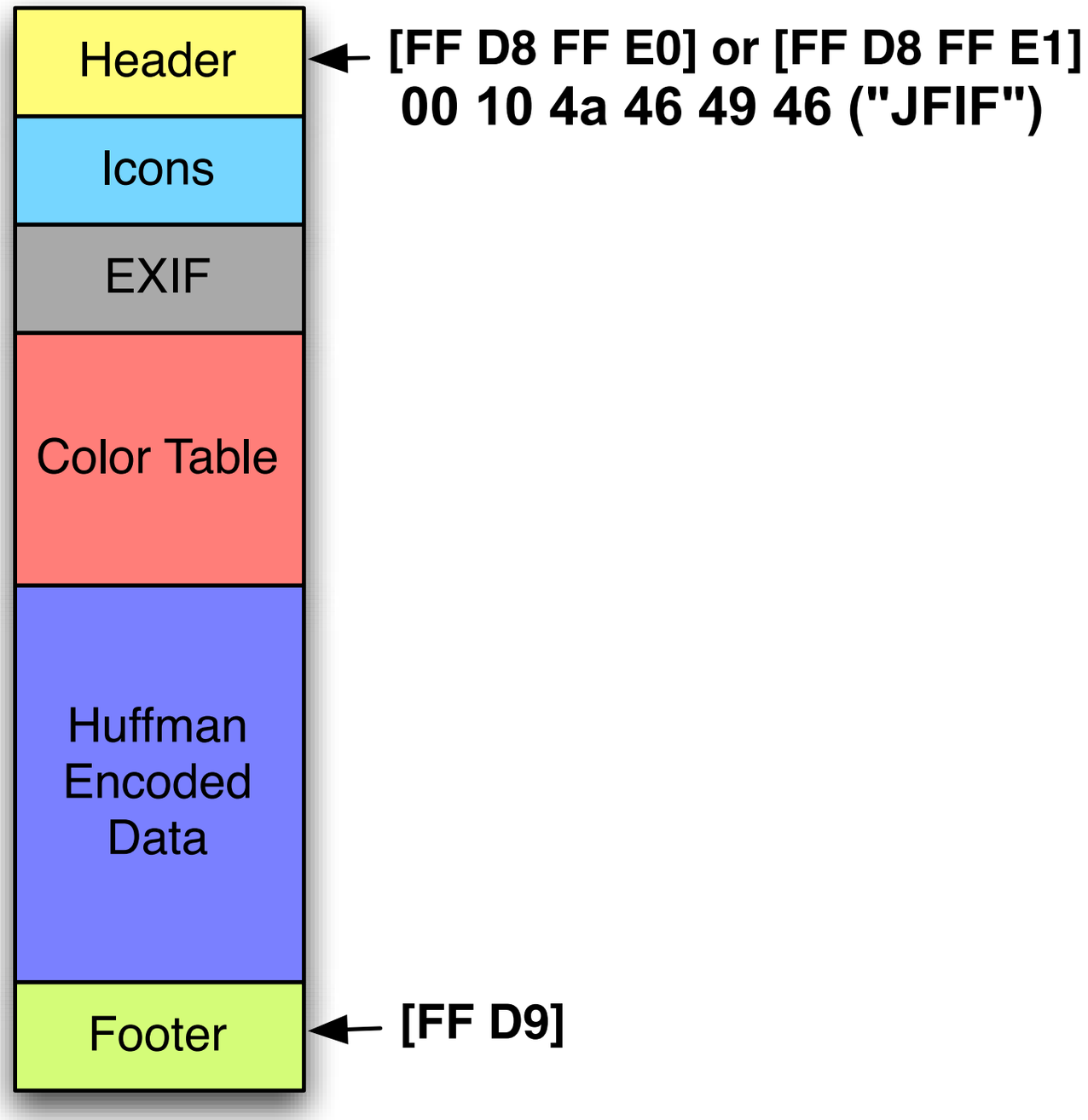
Example: Carving JPEG Files

JPEGs are container files

- Standard Header
- Standard Footer
- Embedded Images

Carving strategy:

- Find all headers
- Find all footers
- Save sectors to files



Header/Footer carving involves saving the data between a known header & known footer.

Disk Sectors ➔

Header/Footer carving involves saving the data between a known header & known footer.

Disk Sectors ➔

Header/Footer carving involves saving the data between a known header & known footer.

Disk Sectors ➔

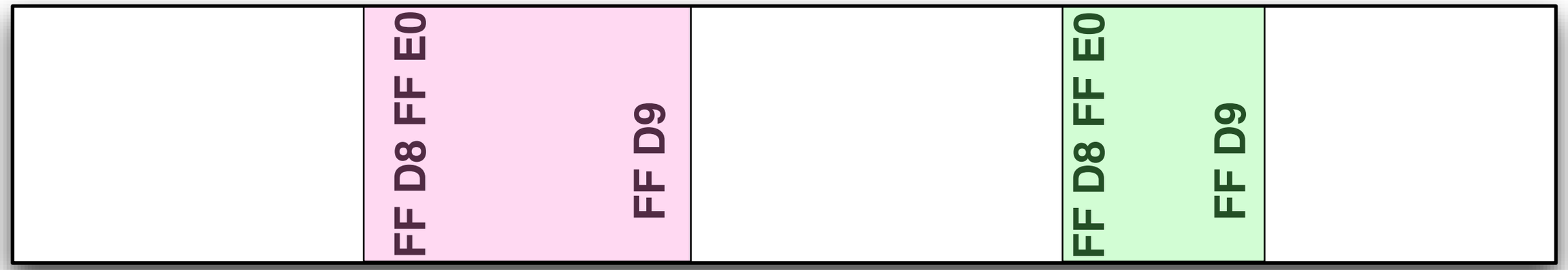
Header/Footer carving involves saving the data between a known header & known footer.

This strategy is used by foremost and scalpel.

Disk Sectors ➔

Header/Footer carving involves saving the data between a known header & known footer.

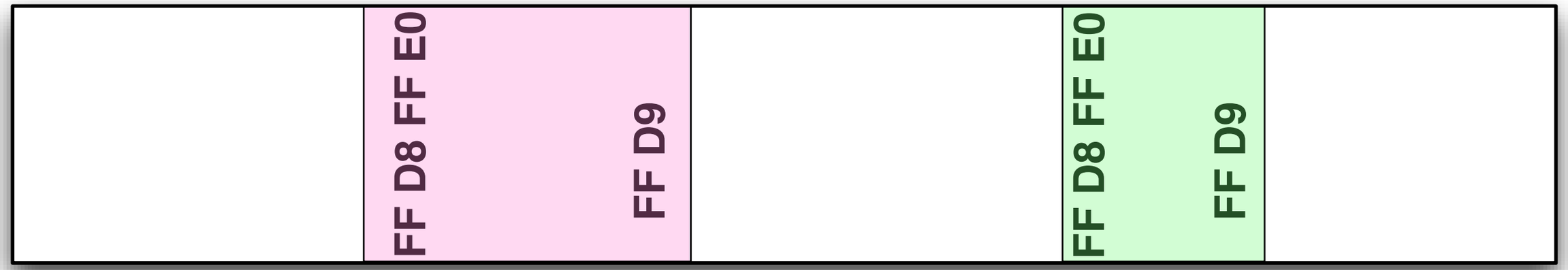
This strategy is used by foremost and scalpel.

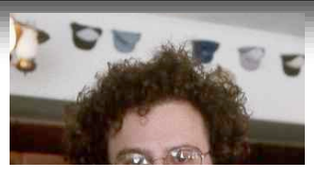


Disk Sectors ➔

Header/Footer carving involves saving the data between a known header & known footer.

This strategy is used by foremost and scalpel.

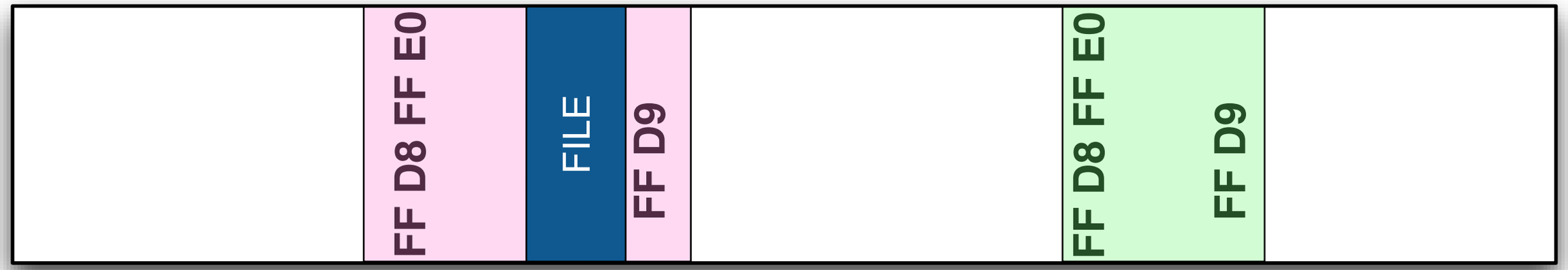


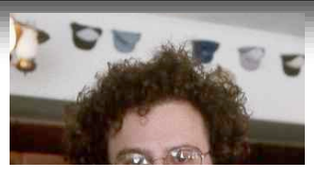


Disk Sectors ➔

Header/Footer carving involves saving the data between a known header & known footer.

This strategy is used by foremost and scalpel.

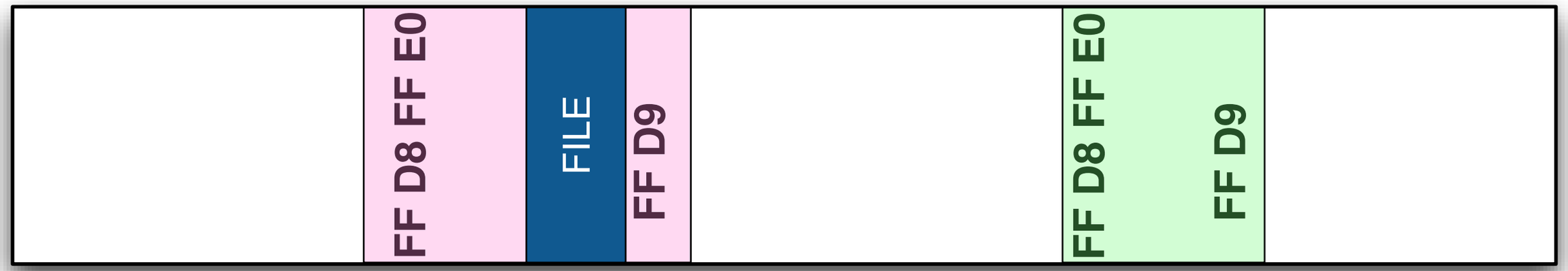


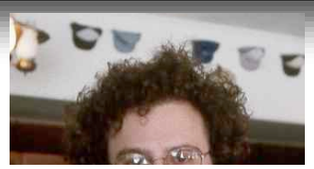


Disk Sectors ➔

Header/Footer carving involves saving the data between a known header & known footer.

This strategy is used by foremost and scalpel.





Disk Sectors ➔

Possible explanations:

1. **This file may be fragmented.**
2. **The file may have been overwritten.**

If the file is fragmented, it can be recovered with *fragment recovery carving*

Carving tools available today:

Open Source:

- **Foremost -** Developed by Jesse Kornblum and Kris Kendall at AFOSI
- **Scalpel** - Improved version of Foremost, by Golden G. Richard III
- **CarvFS** - Virtual file system for carving
- **PhotoRec** - Recovers lost photos from hard drives
- **RevIT & S2** - Experimental carvers developed for DFRWS 2006 carving challenge

Commercial:

- **Adroit Photo Recovery** — Amazing, but only works on JPEGs
- **EnCase -** comes with some eScripts that will carve
- **DataLifter** - File Extractor Pro

Let's use scalpel to carve a memory dump...

```
$ tar xfz scalpel-1.60.tar.gz
$ cd scalpel-1.60;make bsd
gcc -Wall -O2   -D__OPENBSD  -c helpers.c
gcc -Wall -O2   -D__OPENBSD  -c scalpel.c
gcc -Wall -O2   -D__OPENBSD  -c files.c
gcc -Wall -O2   -D__OPENBSD  -c dig.c
gcc -Wall -O2   -D__OPENBSD  -c prioque.c
gcc -Wall -O2   -D__OPENBSD  -c base_name.c
gcc -Wall -O2   -D__OPENBSD  -o scalpel helpers.o scalpel.o files.o dig.o
prioque.o base_name.o -lm
$
```

Edit the Scalpel config file to look for JPEGs...

```
#-----------------------------------------------------------
# GRAPHICS FILES
#-----------------------------------------------------------
#
# GIF and JPG files (very common)
        gif     y       5000000         \x47\x49\x46\x38\x37\x61        \x00\x3b
        gif     y       5000000         \x47\x49\x46\x38\x39\x61        \x00\x3b
        jpg     y       200000000       \xff\xd8\xff\xe0\x00\x10        \xff\xd9
#
#
# PNG
        png     y       20000000        \x50\x4e\x47?   \xff\xfc\xfd\xfe
#
#
# BMP   (used by MSWindows, use only if you have reason to think there are
#        BMP files worth digging for. This often kicks back a lot of false
#        positives
#
        bmp     y       100000  BM??\x00\x00\x00
#
# TIFF
        tif     y       200000000       \x49\x49\x2a\x00
# TIFF
        tif     y       200000000       \x4D\x4D\x00\x2A
```

Footer

Case Sensitive Header/Footer

Extension

Max Size

Header

Run scalpel with memory image as input file...

```
$ ./scalpel -c scalpel.conf -o outdir1 ~/image.dd
Scalpel version 1.60
Written by Golden G. Richard III, based on Foremost 0.69.

Opening target "/Users/simsong/image.dd"

Image file pass 1/2.
/Users/simsong/image.dd:  19.5% |**********  |  100.0 MB    00:21 ETA
```

Scalpel's output is verbose..

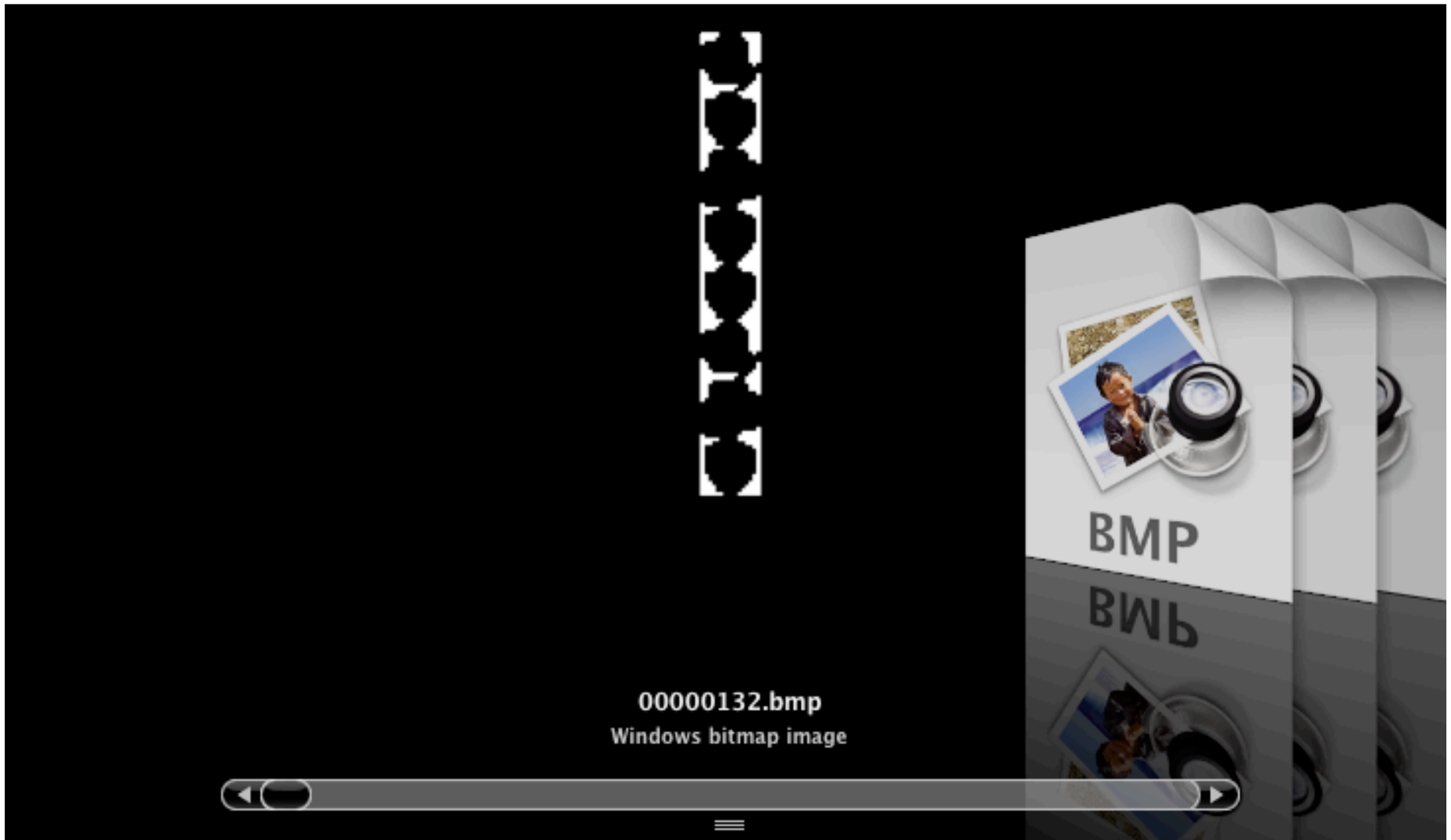
```
Carve lists built.  Workload:
gif with header "\x47\x49\x46\x38\x37\x61" and footer "\x00\x3b" --> 9 files
gif with header "\x47\x49\x46\x38\x39\x61" and footer "\x00\x3b" --> 103
files
jpg with header "\xff\xd8\xff\xe0\x00\x10" and footer "\xff\xd9" --> 15 files
png with header "\x50\x4e\x47\x3f" and footer "\xff\xfc\xfd\xfe" --> 5 files
bmp with header "\x42\x4d\x3f\x3f\x00\x00\x00" and footer "" --> 32 files
tif with header "\x49\x49\x2a\x00" and footer "" --> 2 files
tif with header "\x4d\x4d\x00\x2a" and footer "" --> 3 files
Carving files from image.
Image file pass 2/2.
/Users/simsong/image.dd: 100.0% |
*****************************************************|  512.0 MB
00:00 ETAProcessing of image file complete. Cleaning up...
Done.
Scalpel is done, files carved = 169, elapsed = 45 seconds.
$ ls -l outdir
total 12
-rw-r--r--    1 simsong  simsong  10055 Oct  5 18:36 audit.txt
drwxr-xr-x   34 simsong  simsong   1156 Oct  5 18:35 bmp-4-0/
drwxr-xr-x   11 simsong  simsong    374 Oct  5 18:36 gif-0-0/
drwxr-xr-x  105 simsong  simsong   3570 Oct  5 18:36 gif-1-0/
drwxr-xr-x   17 simsong  simsong    578 Oct  5 18:35 jpg-2-0/
drwxr-xr-x    7 simsong  simsong    238 Oct  5 18:35 png-3-0/
drwxr-xr-x    4 simsong  simsong    136 Oct  5 18:35 tif-5-0/
drwxr-xr-x    5 simsong  simsong    170 Oct  5 18:35 tif-6-0/
$
```

Scalpel creates an “audit file” with information about what it found.

```
Scalpel version 1.60 audit file
Started at Sun Oct  5 18:35:20 2008
Command line:
./scalpel -c scalpel.conf -o outdir1 /Users/simsong/image.dd

Output directory: /Users/simsong/scalpel-1.60/outdir1
Configuration file: /Users/simsong/scalpel-1.60/scalpel.conf

Opening target "/Users/simsong/image.dd"

The following files were carved:
File                 Start                Chop          Length          Extracted
From
00000132.bmp         14597258             YES           100000          image.dd
00000010.gif         11806720             NO              1416          image.dd
00000009.gif         11804672             NO              2004          image.dd
00000012.gif         50000312             NO                44          image.dd
00000011.gif         42857896             NO               332          image.dd
00000015.gif         62047623             NO                53          image.dd
00000014.gif         60672512             NO               371          image.dd
00000013.gif         60672208             NO                61          image.dd
…
00000017.gif         65827840             NO               477          image.dd
00000016.gif         65826816             NO               451          image.dd
00000000.gif         66591424             NO              3537          image.dd
00000113.jpg         74222592             NO            129055          image.dd
00000112.jpg         74219520             NO              2383          image.dd
```

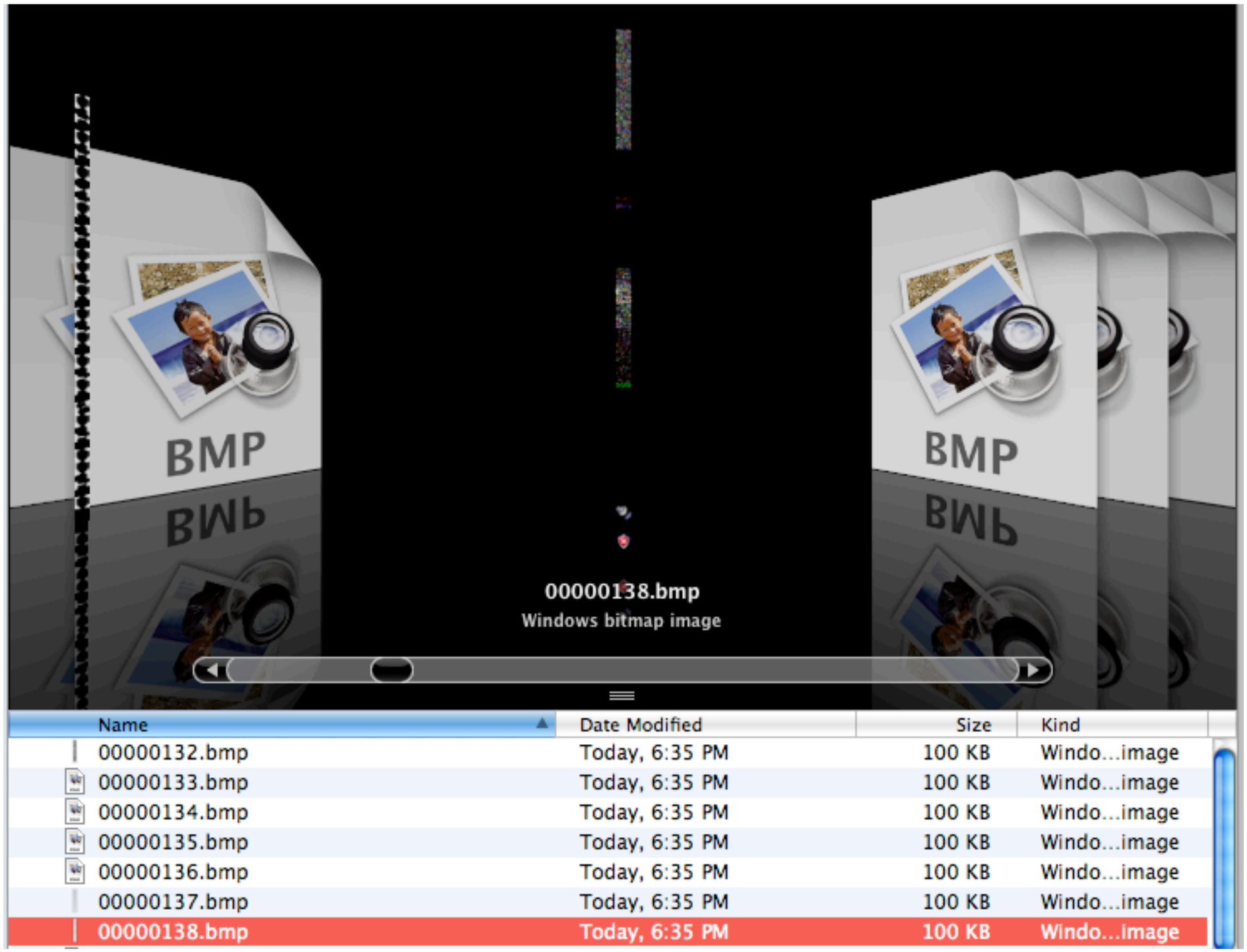
And here's some of what we found...

And here's some of what we found...

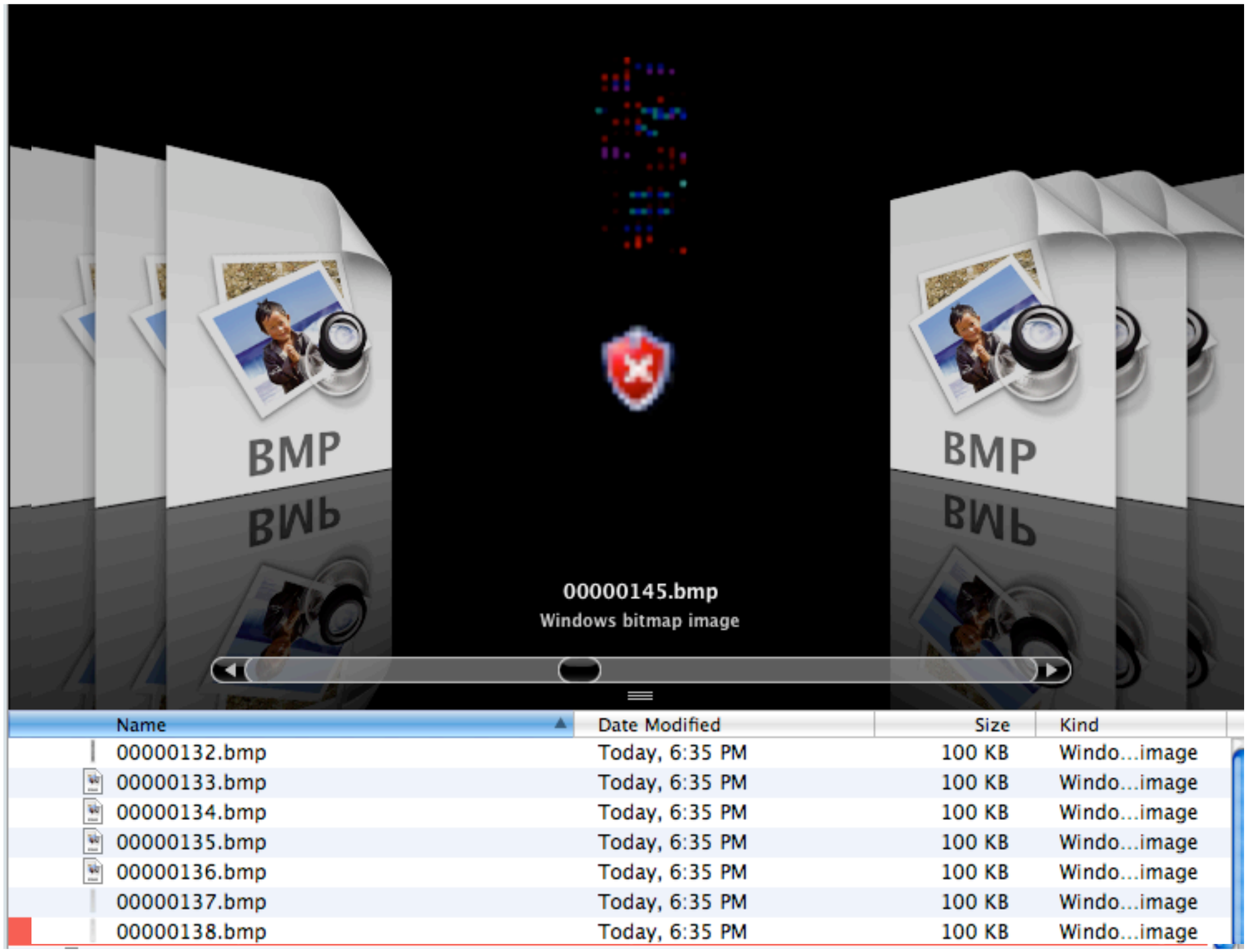


Name	Date Modified	Size	Kind
00000132.bmp	Today, 6:35 PM	100 KB	Windo...image
00000133.bmp	Today, 6:35 PM	100 KB	Windo...image
00000134.bmp	Today, 6:35 PM	100 KB	Windo...image
00000135.bmp	Today, 6:35 PM	100 KB	Windo...image
00000136.bmp	Today, 6:35 PM	100 KB	Windo...image

And here's some of what we found...

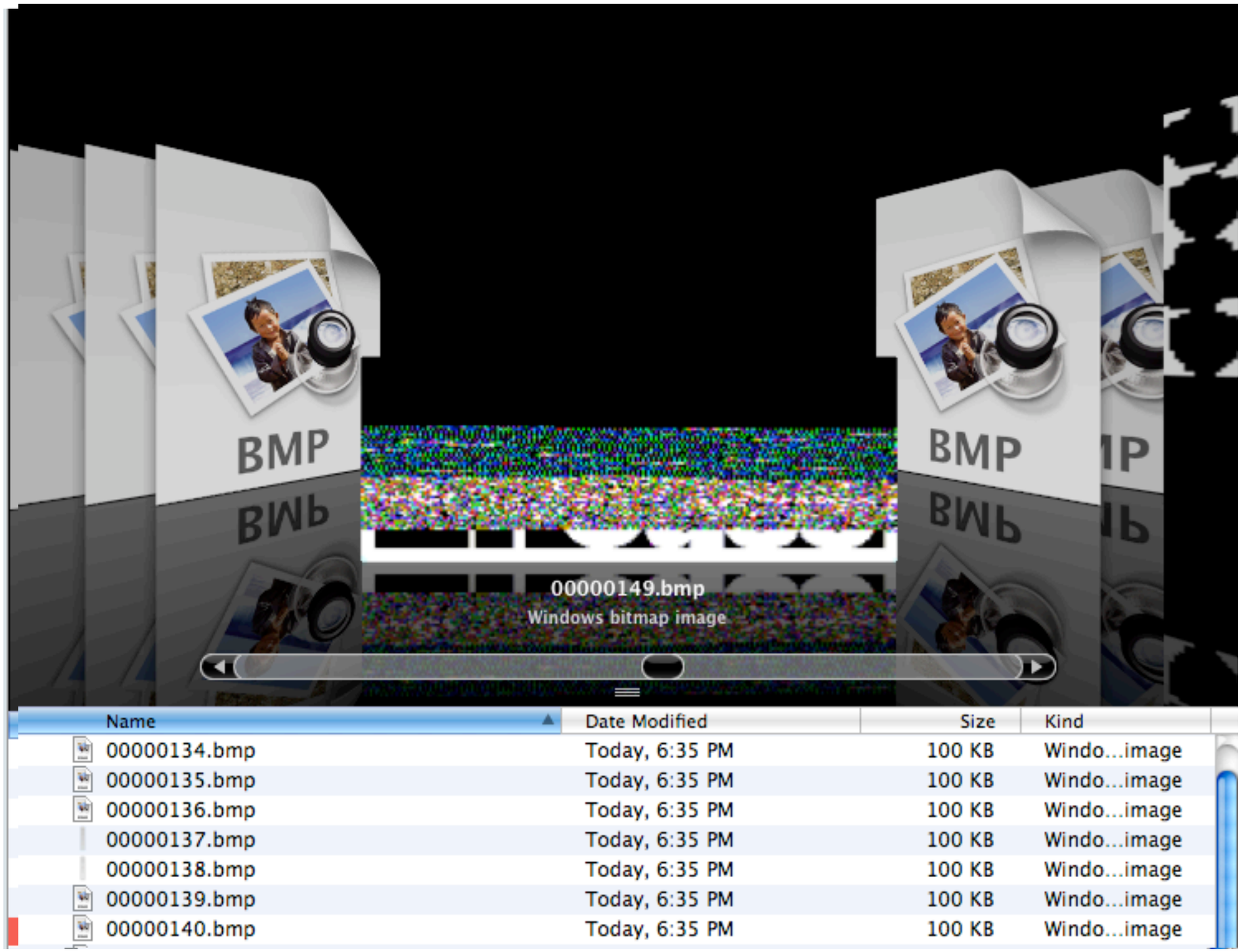


And here's some of what we found...



Name	Date Modified	Size	Kind
00000132.bmp	Today, 6:35 PM	100 KB	Windo...image
00000133.bmp	Today, 6:35 PM	100 KB	Windo...image
00000134.bmp	Today, 6:35 PM	100 KB	Windo...image
00000135.bmp	Today, 6:35 PM	100 KB	Windo...image
00000136.bmp	Today, 6:35 PM	100 KB	Windo...image
00000137.bmp	Today, 6:35 PM	100 KB	Windo...image
00000138.bmp	Today, 6:35 PM	100 KB	Windo...image

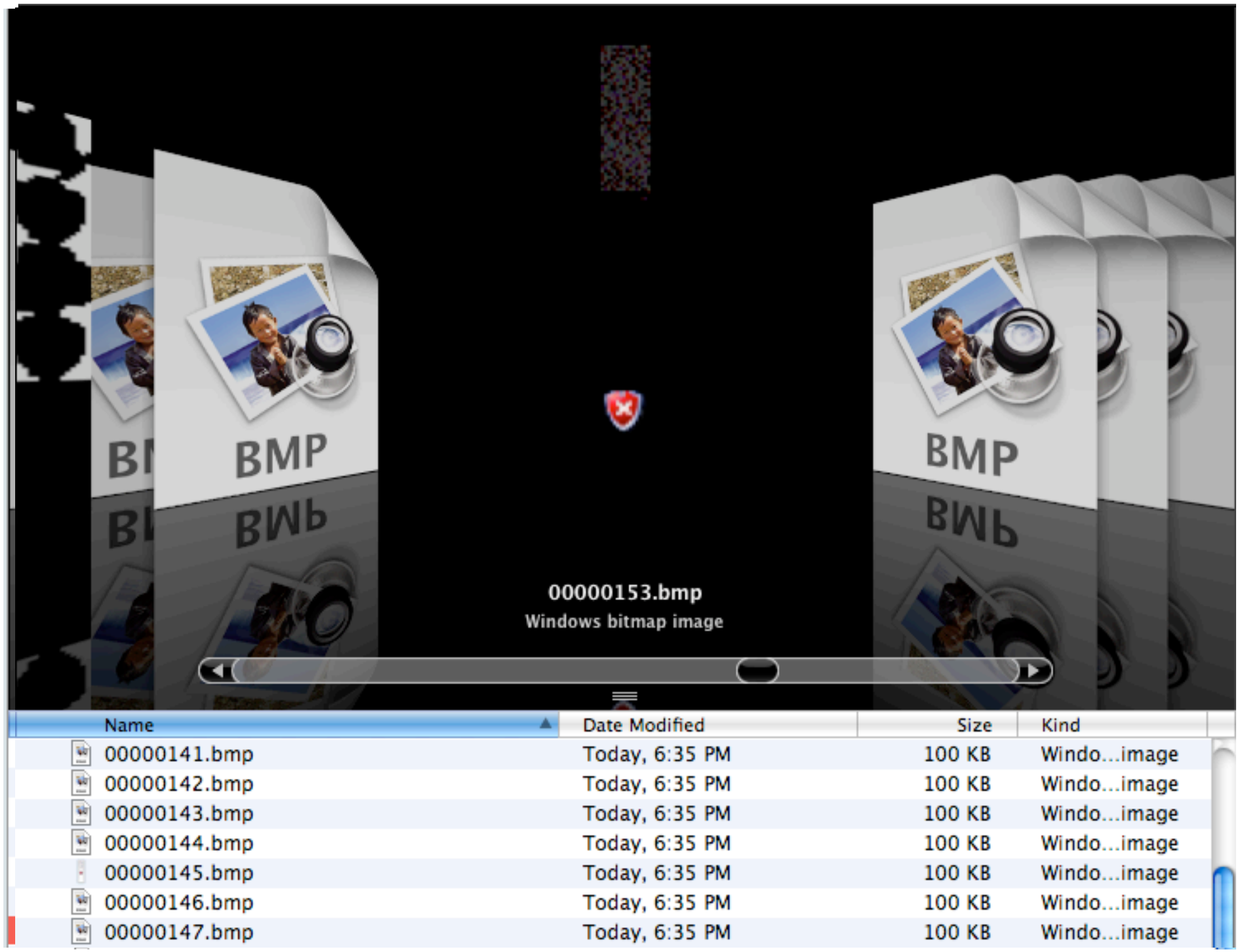
And here's some of what we found...



And here's some of what we found...

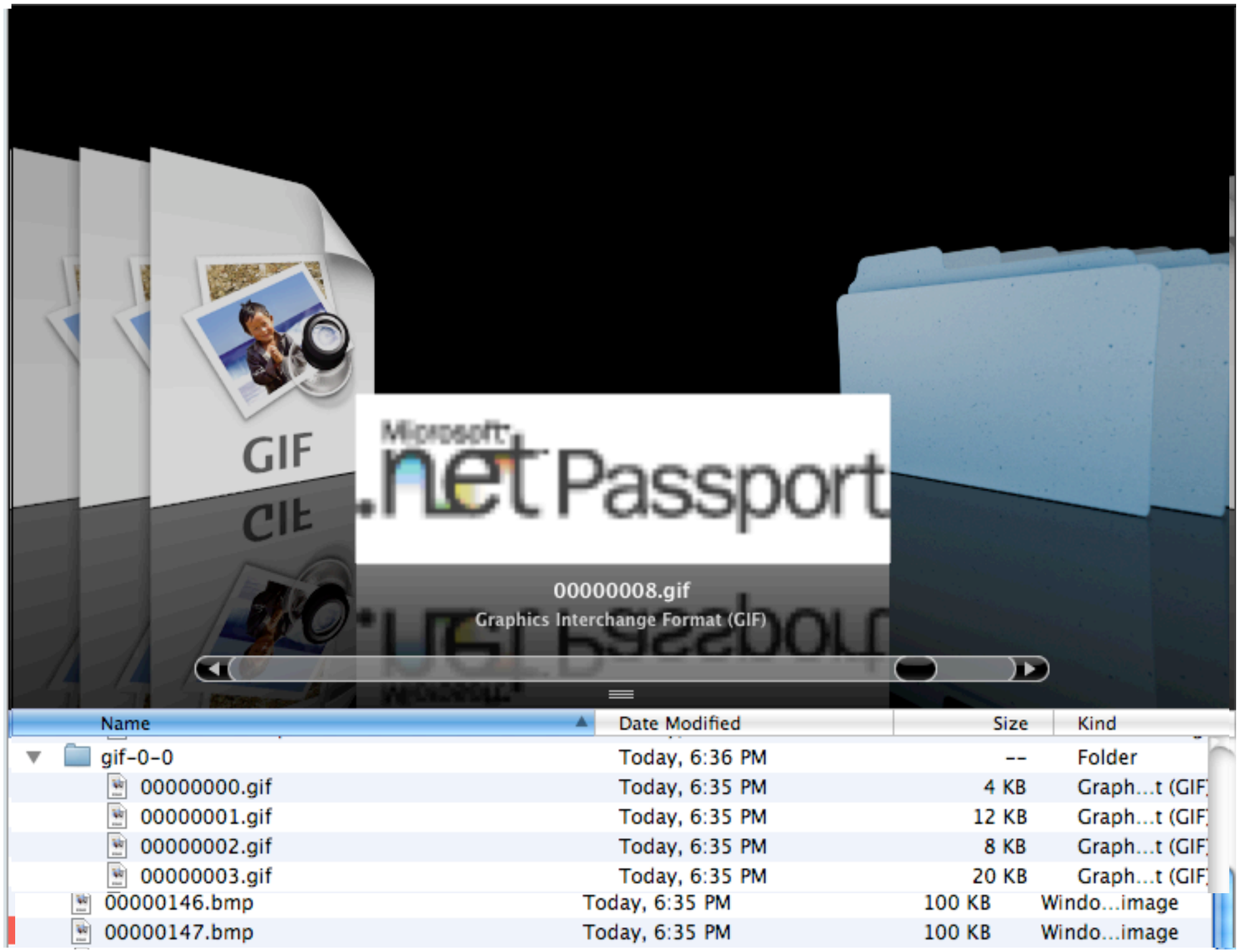
Name	Date Modified	Size	Kind
00000141.bmp	Today, 6:35 PM	100 KB	Windo...image
00000142.bmp	Today, 6:35 PM	100 KB	Windo...image
00000143.bmp	Today, 6:35 PM	100 KB	Windo...image
00000144.bmp	Today, 6:35 PM	100 KB	Windo...image
00000145.bmp	Today, 6:35 PM	100 KB	Windo...image
00000146.bmp	Today, 6:35 PM	100 KB	Windo...image
00000147.bmp	Today, 6:35 PM	100 KB	Windo...image

And here's some of what we found...

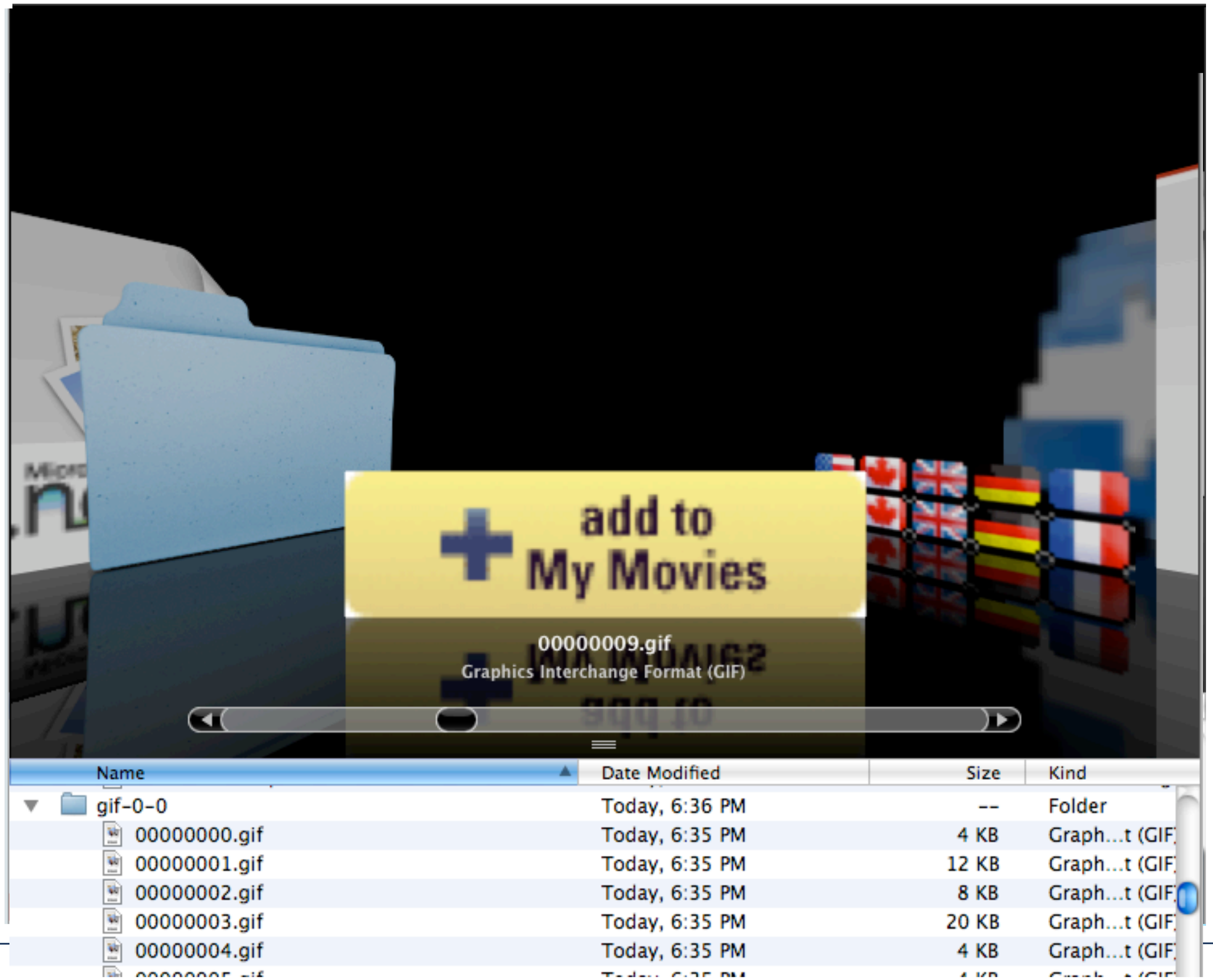


Name	Date Modified	Size	Kind
00000141.bmp	Today, 6:35 PM	100 KB	Windo...image
00000142.bmp	Today, 6:35 PM	100 KB	Windo...image
00000143.bmp	Today, 6:35 PM	100 KB	Windo...image
00000144.bmp	Today, 6:35 PM	100 KB	Windo...image
00000145.bmp	Today, 6:35 PM	100 KB	Windo...image
00000146.bmp	Today, 6:35 PM	100 KB	Windo...image
00000147.bmp	Today, 6:35 PM	100 KB	Windo...image

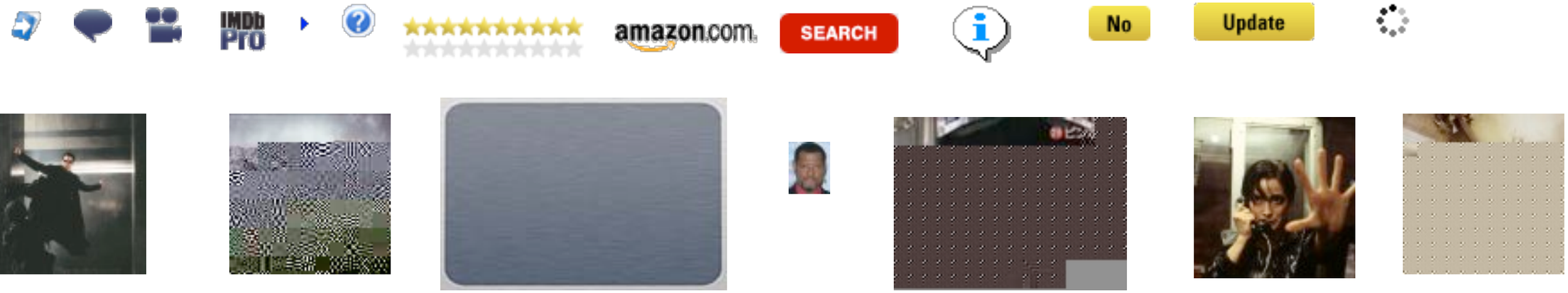
And here's some of what we found...



And here's some of what we found...



More photos…(actual size)
Note: No images more than 4K (due to page size)

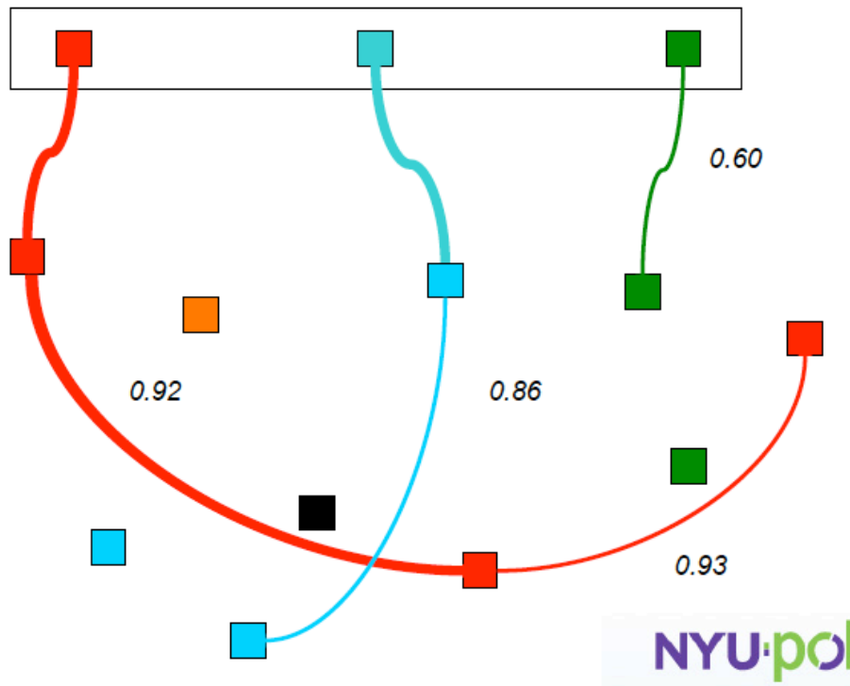


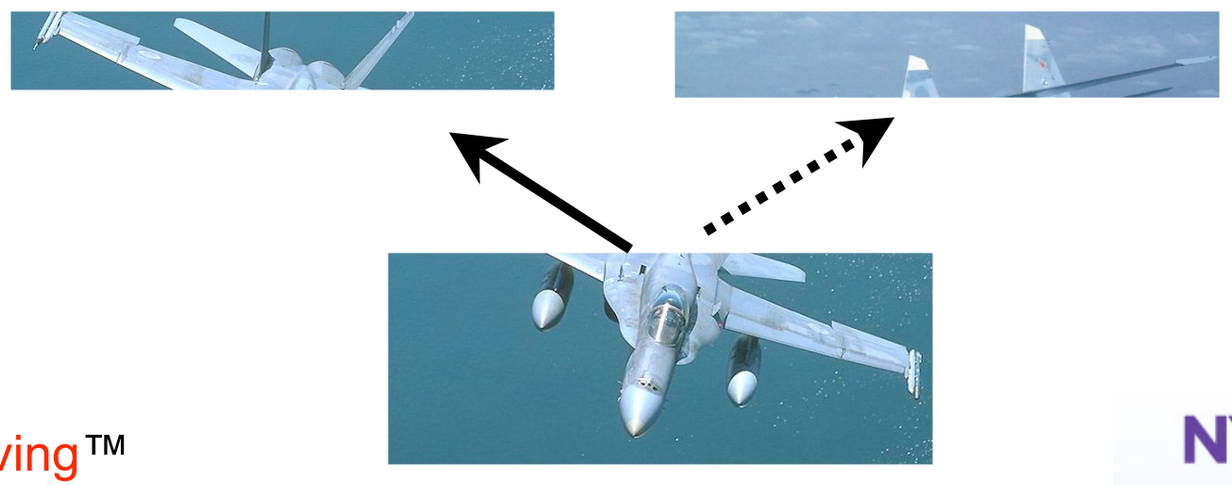
Adroit constructs paths through sequential hypothesis testing (SHT)

Incorrect paths:

- Do not decompress
- Have sudden changes between scan lines.

http://digital-assembly.com/





bulk_extractor is a multi-threaded carver that looks for email addresses (and other information)

Written in C, C++ and Flex

- Command-line tool.
- Linux, MacOS, Windows (compiled with mingw)

Key Features:

- Uses regular expressions and rules to scan for:
  - —*email addresses; credit card numbers; JPEG EXIFs; URLs; Email fragments.*
- Recursively re-analyzes ZIP components.
- Produces a histogram of the results.
- Multi-threaded.
  - —*Disk is "striped" into pages*
  - —*Results stored in mostly-ordered “feature files”*
- Work with evidence files of *any size* and on *limited hardware.*

bulk_extractor output: text files of "features" and context.

email addresses from domexusers:

```
48198832  domexuser2@gmail.com    tocol>____<name>domexuser2@gmail.com/Home</name>____
48200361  domexuser2@live.com     tocol>____<name>domexuser2@live.com</name>____<pass
48413829  siege@preoccupied.net   siege) O'Brien <siege@preoccupied.net>_hp://meanwhi
48481542  danilo@gnome.org        Danilo __egan <danilo@gnome.org>_Language-Team:
48481589  gnom@prevod.org         : Serbian (sr) <gnom@prevod.org>_MIME-Version:
49421069  domexuser1@gmail.com    server2.name", "domexuser1@gmail.com");__user_pref("
49421279  domexuser1@gmail.com    er2.userName", "domexuser1@gmail.com");__user_pref("
49421608  domexuser1@gmail.com    tp1.username", "domexuser1@gmail.com");__user_pref("
```

Histogram:

```
n=579   domexuser1@gmail.com
n=432   domexuser2@gmail.com
n=340   domexuser3@gmail.com
n=268   ips@mail.ips.es
n=252   premium-server@thawte.com
n=244   CPS-requests@verisign.com
n=242   someone@example.com
```

bulk_extractor success:
City of San Luis Obispo Police Department, Spring 2010

District Attorney filed charges against two individuals:

- Credit Card Fraud
- Possession of materials to commit credit card fraud.

Defendants:

- arrested with a computer.
- Expected to argue that defendants were unsophisticated and lacked knowledge.

Examiner given 250GB drive *the day before preliminary hearing.*

- In 2.5 hours Bulk Extractor found:
  - —*Over 10,000 credit card numbers on the HD (1000 unique)*
  - —*Most common email address belonged to the primary defendant (possession)*
  - —*The most commonly occurring Internet search engine queries concerned credit card fraud and bank identification numbers (intent)*
  - —*Most commonly visited websites were in a foreign country whose primary language is spoken fluently by the primary defendant.*
- Armed with this data, the DA was able to have the defendants held.

Eliminating false positives:
Many of the email addresses come with Windows!

Sources of these addresses:

- Windows binaries
- SSL certificates
- Sample documents

```
n=579    domexuser1@gmail.com
n=432    domexuser2@gmail.com
n=340    domexuser3@gmail.com
n=268    ips@mail.ips.es
n=252    premium-server@thawte.com
n=244    CPS-requests@verisign.com
n=242    someone@example.com
```

It's important to suppress email addresses not relevant to the case.

Approach #1 — Suppress emails seen on many other drives.

Approach #2 — Stop list from bulk_extractor run on clean installs.

Both of these methods *white list* commonly seen emails.

- Operating Systems have a LOT of emails. (FC12 has 20,584!)
- Should Linux developers email addresses be invisible to our tools?

Approach #3: Context-sensitive stop list.

Instead of extracting just the email address, extract the context:

- Offset: **351373329**
- Email: **zeeshan.ali@nokia.com**
- Context: **ut_Zeeshan Ali <zeeshan.ali@nokia.com>, Stefan Kost <**

- Offset: **351373366**
- Email: **stefan.kost@nokia.com**
- Context: **>, Stefan Kost <stefan.kost@nokia.com>_____sin**

—Here "context" is 8 characters on either side of feature.

We created a context-sensitive stop list for Microsoft Windows XP, 2000, 2003, Vista, and several Linux.

Total stop list: 70MB (628,792 features; 9MB ZIP file)

Applying it to domexusers HD image:

- # of emails found: 9143 ➔ 4459

without stop list		with stop list	
n=579	domexuser1@gmail.com	n=579	domexuser1@gmail.com
n=432	domexuser2@gmail.com	n=432	domexuser2@gmail.com
n=340	domexuser3@gmail.com	n=340	domexuser3@gmail.com
n=268	ips@mail.ips.es	n=192	domexuser2@live.com
n=252	premium-server@thawte.com	n=153	domexuser2@hotmail.com
n=244	CPS-requests@verisign.com	n=146	domexuser1@hotmail.com
n=242	someone@example.com	n=134	domexuser1@live.com
n=237	inet@microsoft.com	n=91	premium-server@thawte.com
n=192	domexuser2@live.com	n=70	talkback@mozilla.org
n=153	domexuser2@hotmail.com	n=69	hewitt@netscape.com
n=146	domexuser1@hotmail.com	n=54	DOMEXUSER2@GMAIL.COM
n=134	domexuser1@live.com	n=48	domexuser1%40gmail.com@imap.gmail.com
n=115	example@passport.com	n=42	domex2@rad.li
n=115	myname@msn.com	n=39	lord@netscape.com
n=110	ca@digsigtrust.com	n=37	49091023.6070302@gmail.com

http://afflib.org/downloads/feature_context.1.0.zip

bulk_extractor: Implemented as a set of C++ classes

Forensic Buffers and Path:

- sbuf_t — Holds data, margin, and forensic path of each page.
- pos0_t — Path of byte at sbuf[0]

```
100                     Offset at 100 bytes.
100-GZIP-500            At offset 100, GZIP compressed, 500 bytes further in
```

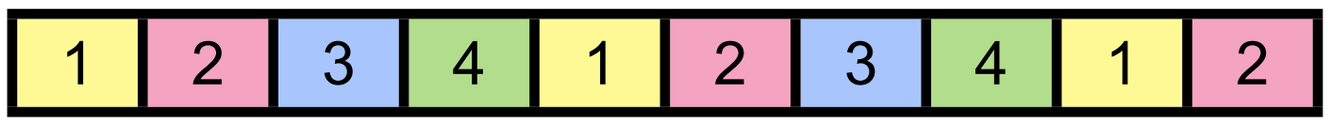
- feature_recorder — Holds output for each feature type

Plug-In Scanner System

- Each scanner is a C++ function that can be linked or loaded at run-time
- Simple scanners look for features in bulk data and report them
  - —*scan_accts, scan_aes, scan_bulk, scan_ccns2, scan_email, scan_exif, scan_find, scan_headers, scan_net, scan_wordlist*
- Scanners can instantiate files:
  - —*scan_kml*
- Scanners can be recursive.
  - —*scan_base64, scan_gzip, scan_hiberfile, scan_pdf, scan_zip*

bulk_extractor: Speed from multi threading

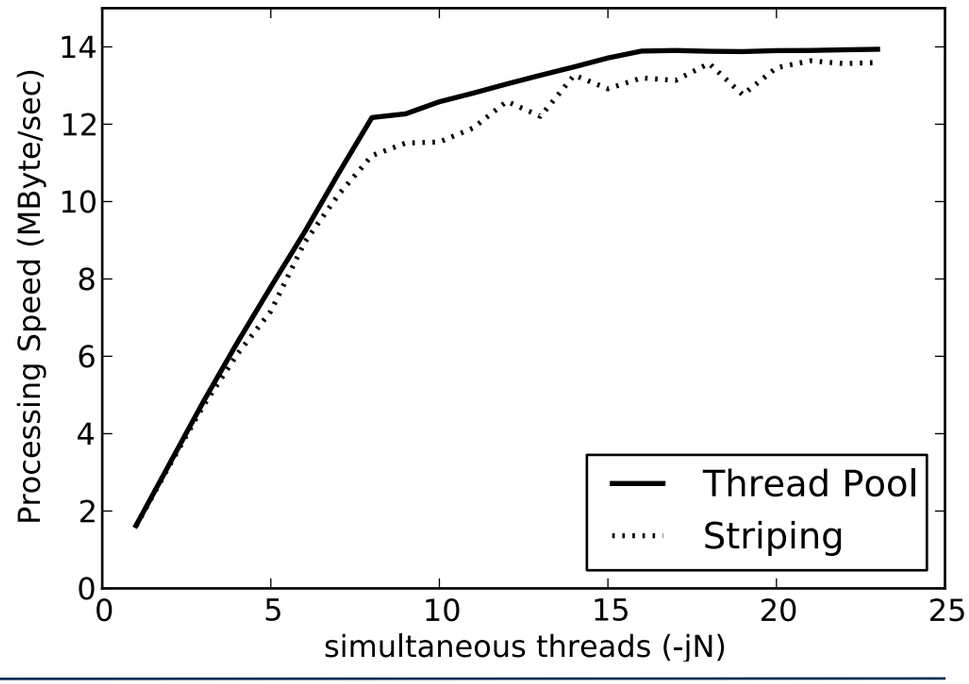
Primary thread:



- Iterator reads “pages” of forensic data and passes each page to a “worker.”
- Iterators available for:
  - —*raw & splitraw files*
  - —*AFF, E01*
  - —*Directory Hierarchies.*
- MD5 is computed automatically as data is read (source validation).
- Generates DFXML file with:
  - —*Tool compile and runtime provenance.*
  - —*Status reports of what is found, errors, etc.*

Worker Threads:

- One per core.
- Automatically figures out how many cores you have.



Bulk_extractor's magic — opportunistic decompression

Most forensic tools recover:

- allocated files
- "deleted" files
- carving of unallocated area.

bulk_extractor uses a different methodology:

- Carving and Named Entity Recognition
- Identification, Decompression and Re-Analysis of compressed data.

This helps with:

- hibernation files and fragments (hibernation files move around)
- swap file fragments
- browser cache fragments (gzip compression)

Post-processing the feature files

The feature files are designed for easy, rapid processing.

- Tab-Delimited
  - —*path, feature, context*
- Text (UTF-8)

bulk_diff.py: prepares difference of two bulk_extractor runs.

- Designed for timeline analysis.
- Developed with analysts.
- Reports “what’s changed.”
  - —*Actually, “what’s new” turned out to be more useful.*
  - —*“what’s missing” includes data inadvertantly overwritten.*

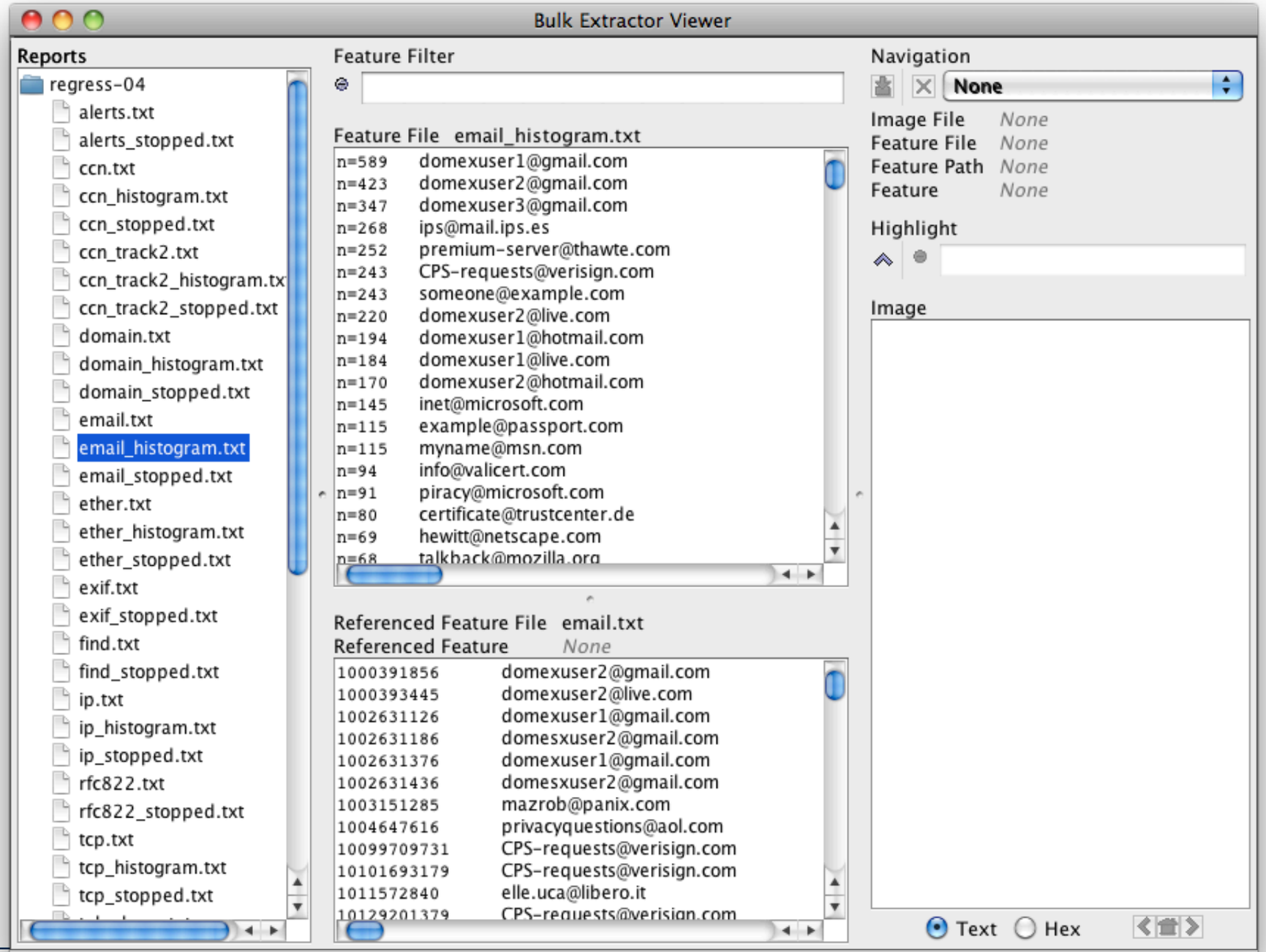
identify_filenames.py: Reports files responsible for features.

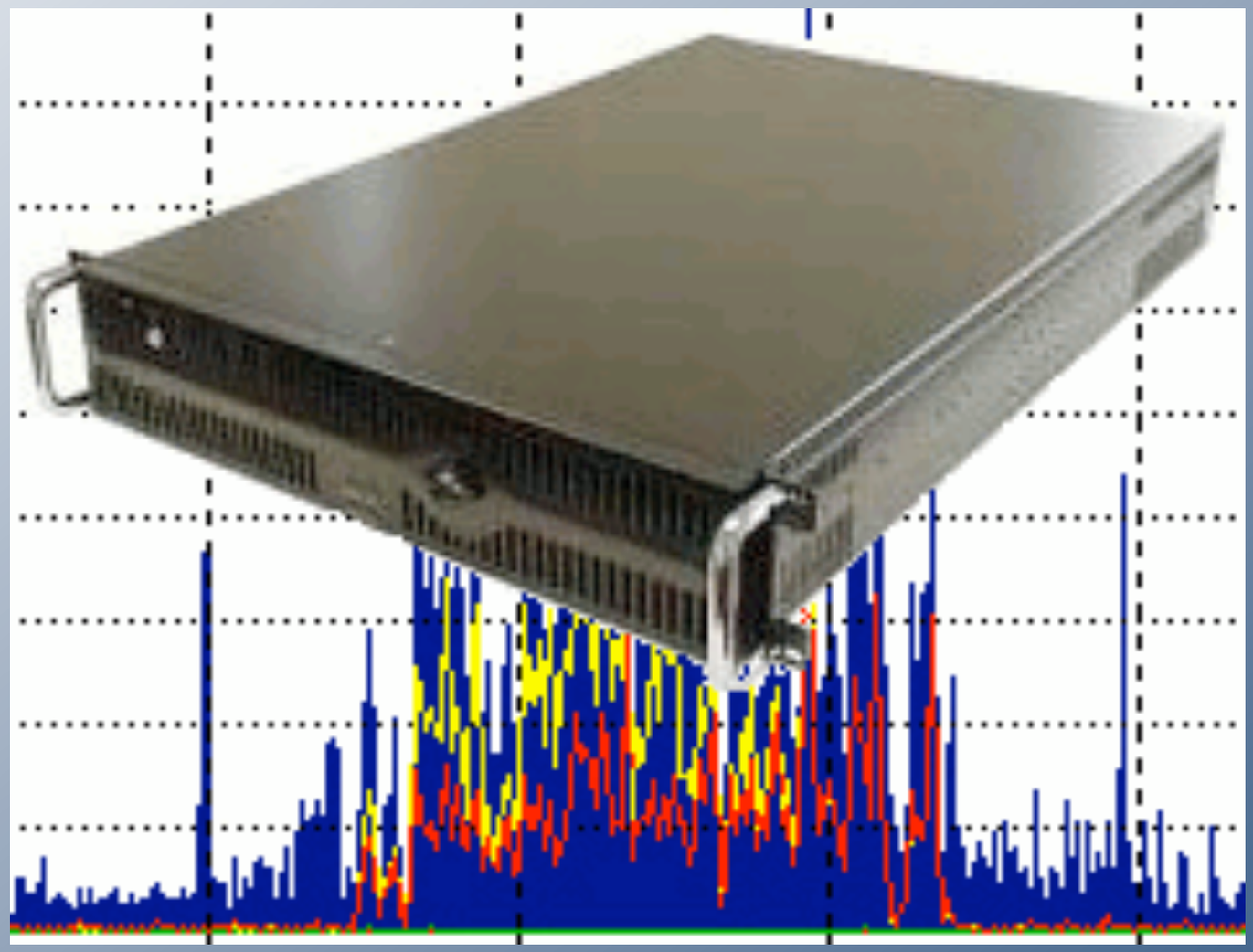
- Requires DFXML run (fiwalk) for disk image.
- Currently a two-step process; could be built in to bulk_extractor

bulk_extractor GUI

100% Java

Uses bulk_extractor to view contents of compressed containers.



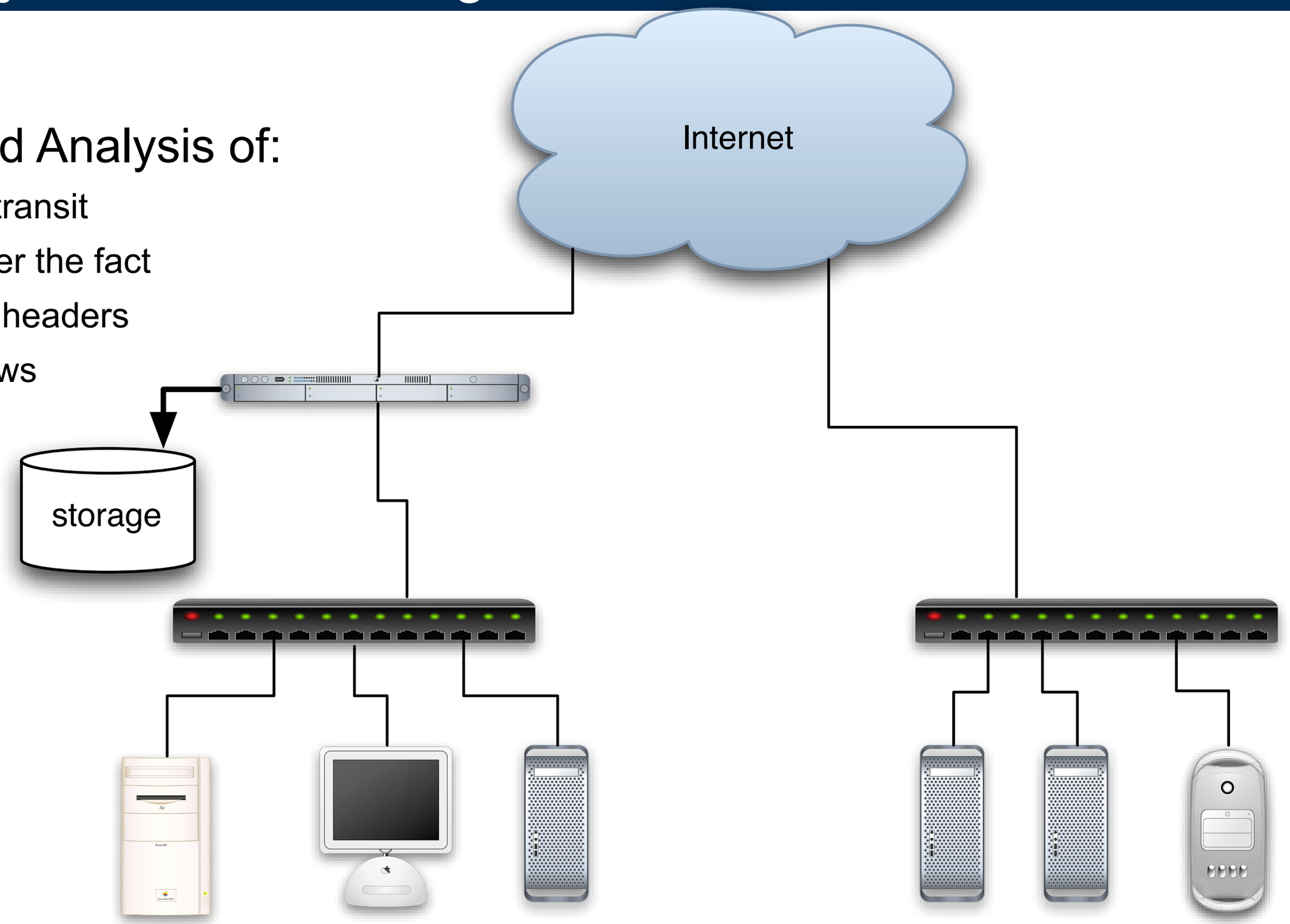


Working with Network Data

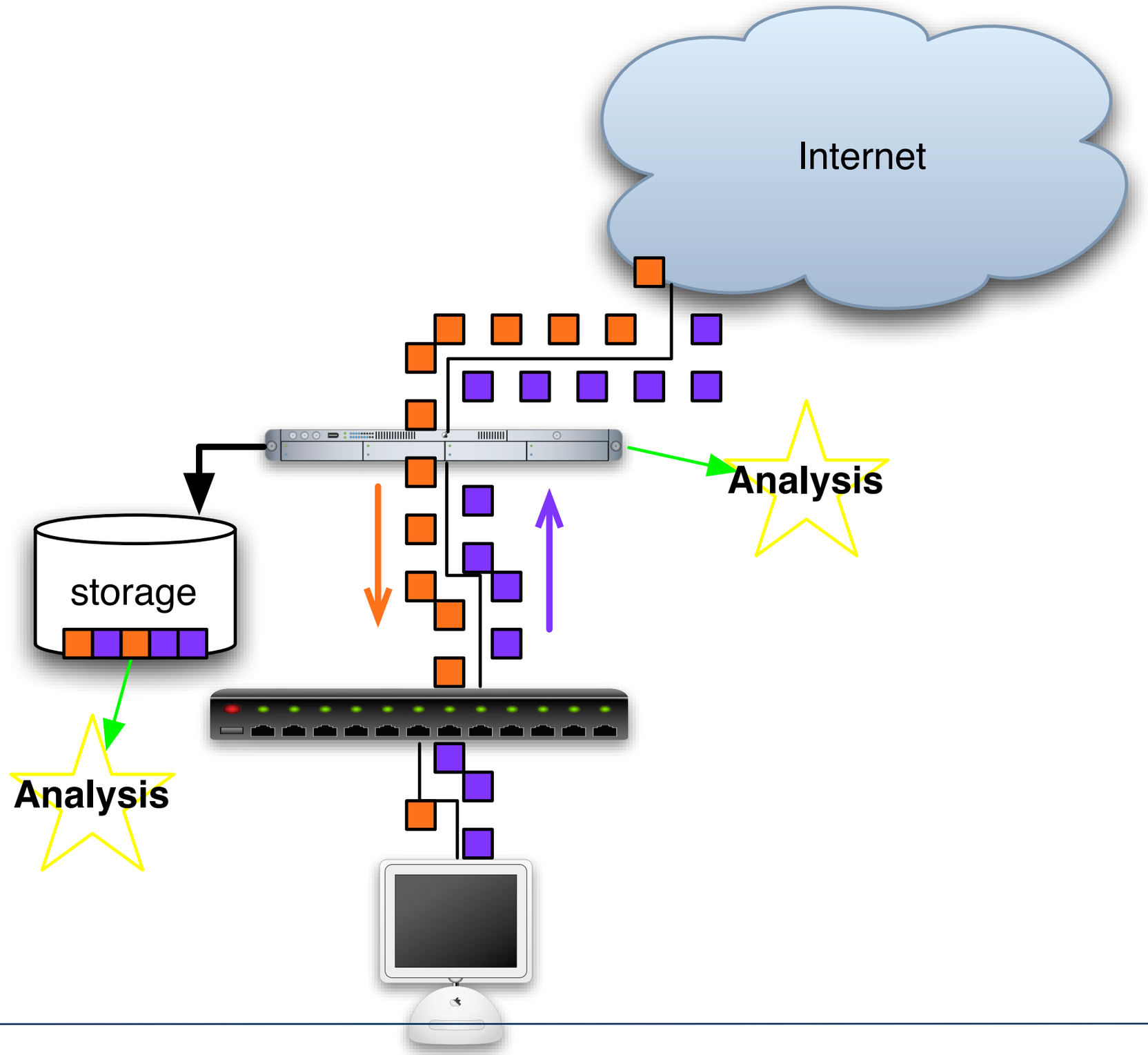
“Network Forensics” has many different meanings.

Capture and Analysis of:

- packets in transit
- packets after the fact
- just packet headers
- network flows
- log files



Packets can be analyzed in flight or after capture.



Packet monitoring is similar to wiretapping.

Passive Monitoring Options:

- Use an ethernet “hub” with a packet sniffer.
- Set up a switched monitoring port.
- Full-duplex networks may require two monitoring ports.

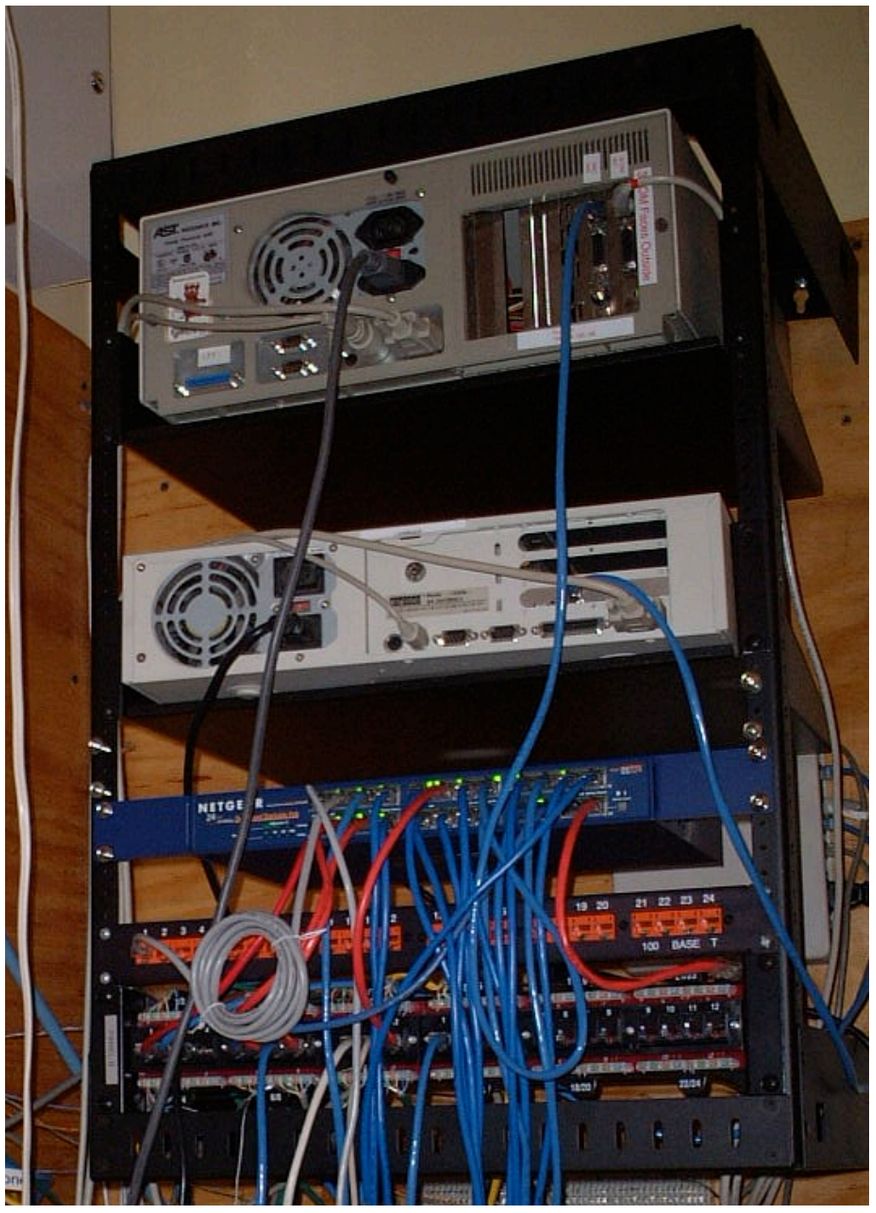
Active Monitoring Options:

- Monitor with a proxy or router.
- Monitor packets at endpoints

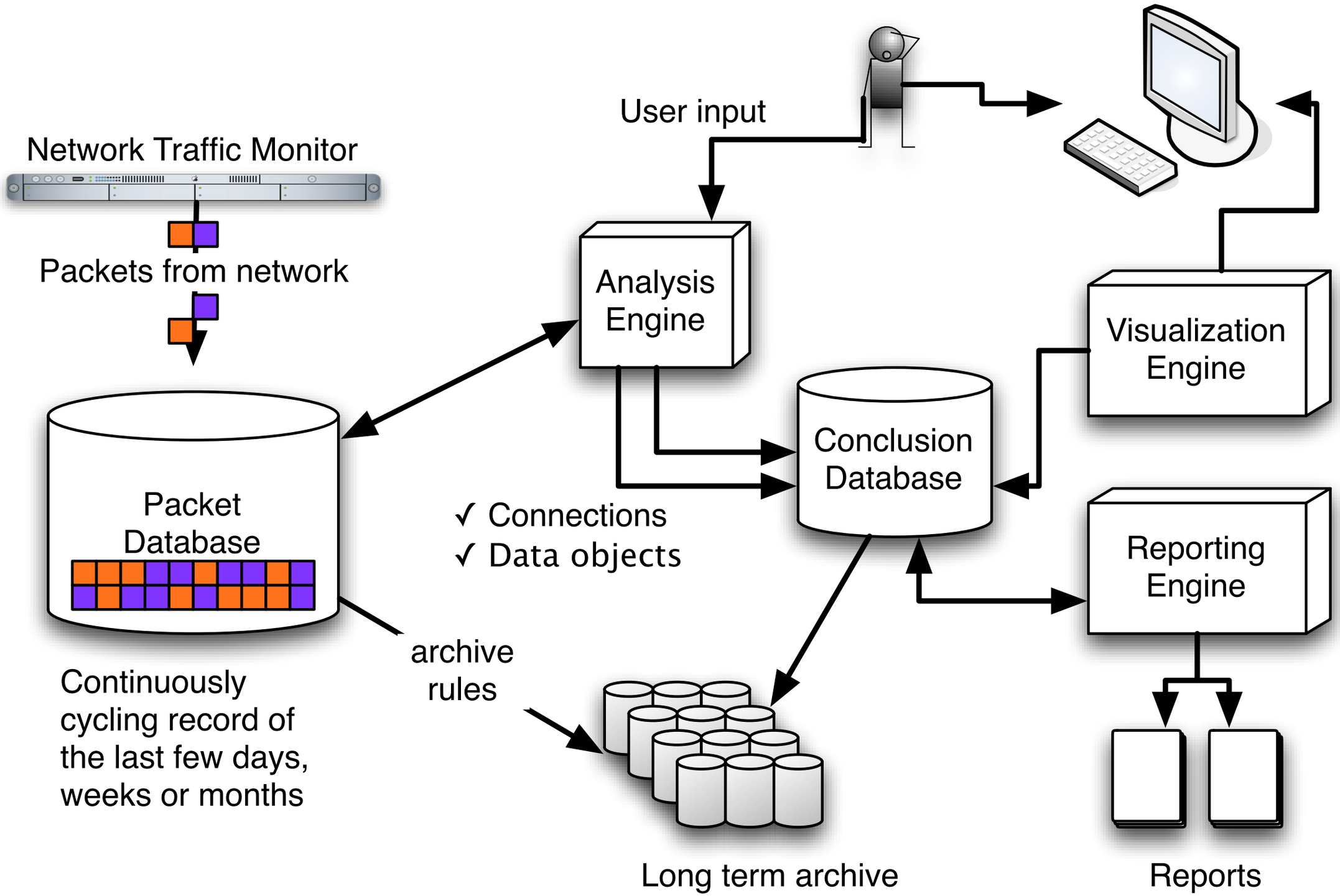
Critical uses:

- Attack assessment
- Policy enforcement

“A DVR for an Internet connection.”



Network Forensics Architecture



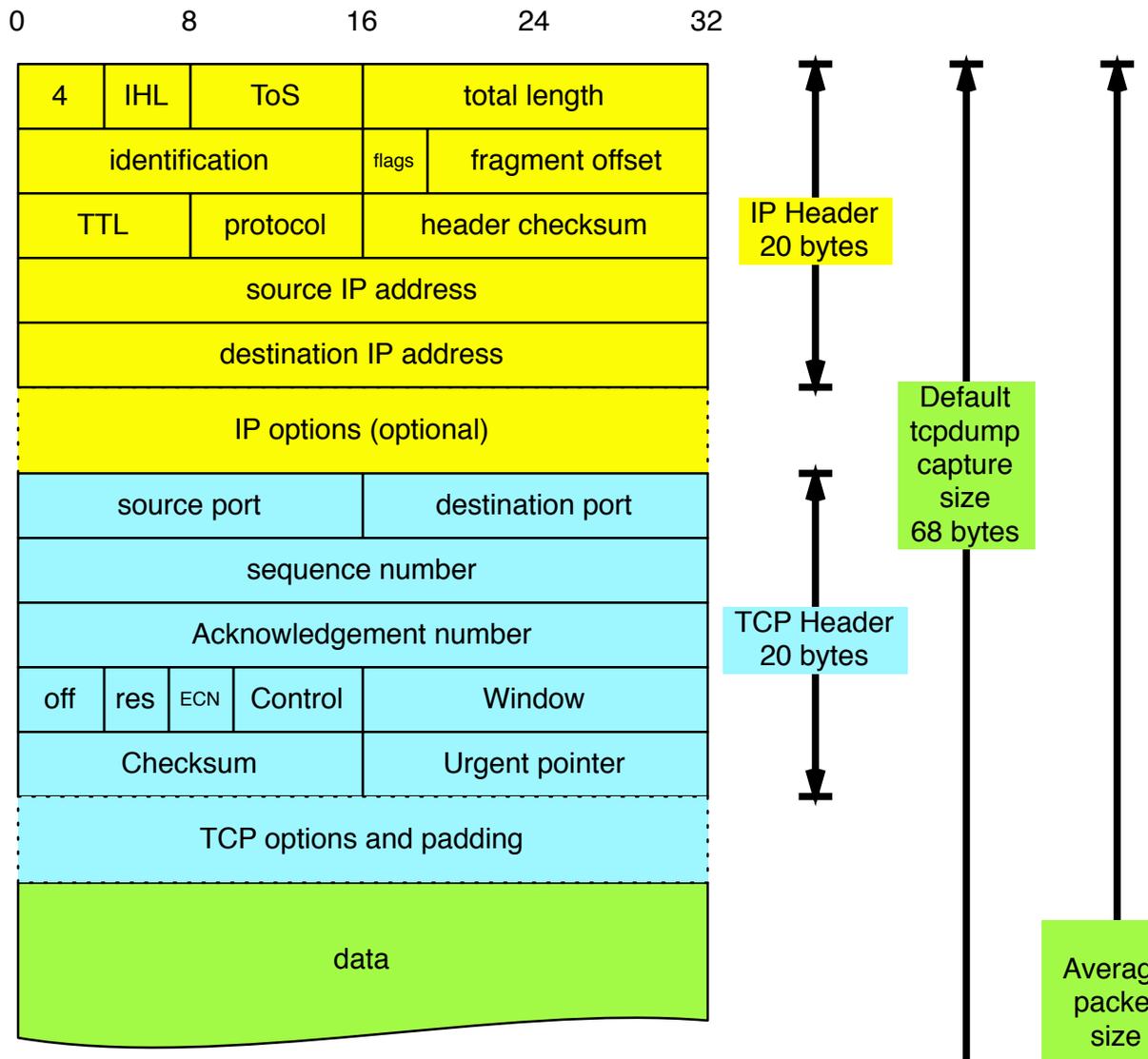
It is completely reasonable to capture all the packets.

In 1991, Los Alamos National Laboratory captured all information in and out of the lab's T1 on DAT tape (8 gigabytes/day @ 50% utilization)

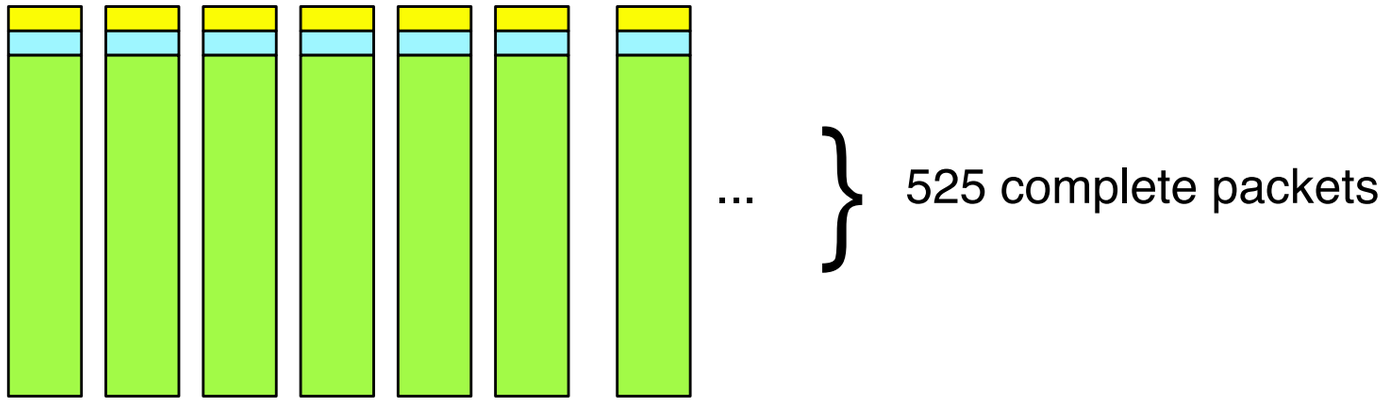
Connection	GB/Day (50%)
T1	8 GB
10 Mbit	54 GB
T3	170 GB
OC3	512 GB
OC12	2,000 GB

- Disks have gotten bigger faster than network connections have gotten faster.
- This is an engineering problem.
- Once implemented, it can also be privacy problem.

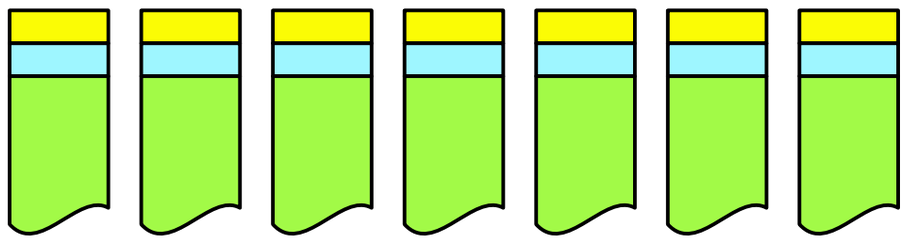
Systems can capture the *entire packet* or *just the packet header*



Downloading a web page transmits many packets over multiple TCP connections.



With just headers, you can only get source, destination, size, timestamps, ports, etc.



... } 525 packet headers

```
10:52:16.294858 IP 192.168.1.102.58754 > www2.cnn.com.http: S
10:52:16.370616 IP www2.cnn.com.http > 192.168.1.102.58754: S
10:52:16.370700 IP 192.168.1.102.58754 > www2.cnn.com.http: .
10:52:16.371114 IP 192.168.1.102.58754 > www2.cnn.com.http: P
10:52:16.455120 IP www2.cnn.com.http > 192.168.1.102.58754: .
10:52:19.956986 IP i7.cnn.net.http > 192.168.1.102.58755: .
10:52:19.961475 IP i7.cnn.net.http > 192.168.1.102.58755: .
10:52:19.981228 IP cnn1.dyn.cnn.com.http > 192.168.1.102.58766:
10:52:19.983731 IP cl4.cnn.com.http > 192.168.1.102.58761: P
```

With the full packets, you can get all the content.

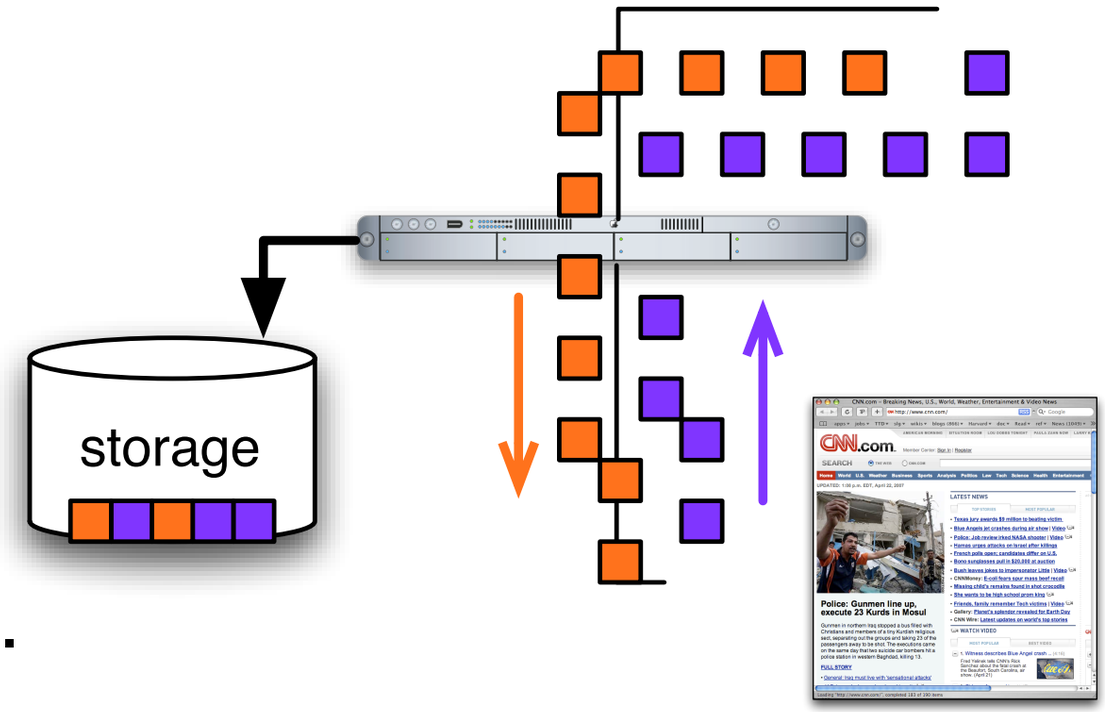
Some vendors call this “deep packet inspection”
or “deep packet analysis.”

Primary use is to discover inappropriate data transfer or service use:

- Use of outside chat or web mail services.
- Leaking protected health Information.
- Restrict information

Good for debugging networks

- Duplicate requests
- Incomplete transactions
- Discovery of vulnerabilities without scanning
- Cleartext usernames & passwords



Also good for finding privacy violations.

Full-content “deep analysis” solutions:

Open Source

- Wireshark
- Snort
- Squil

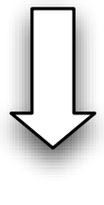
Commercial:

- NetWitness
- Q1Labs
- NIKSUN NetDetector

http://www.netwitness.com/

Packet headers can be used to reconstruct “flows”

```
10:52:16.294858 IP 192.168.1.102.58754 > www2.cnn.com.http: S
10:52:16.370616 IP www2.cnn.com.http > 192.168.1.102.58754: S
10:52:16.370700 IP 192.168.1.102.58754 > www2.cnn.com.http: .
10:52:16.371114 IP 192.168.1.102.58754 > www2.cnn.com.http: P
10:52:16.455120 IP www2.cnn.com.http > 192.168.1.102.58754: .
10:52:19.956986 IP i7.cnn.net.http > 192.168.1.102.58755: .
10:52:19.961475 IP i7.cnn.net.http > 192.168.1.102.58755: .
10:52:19.981228 IP cnn1.dyn.cnn.com.http > 192.168.1.102.58766:
10:52:19.983731 IP cl4.cnn.com.http > 192.168.1.102.58761: P
```

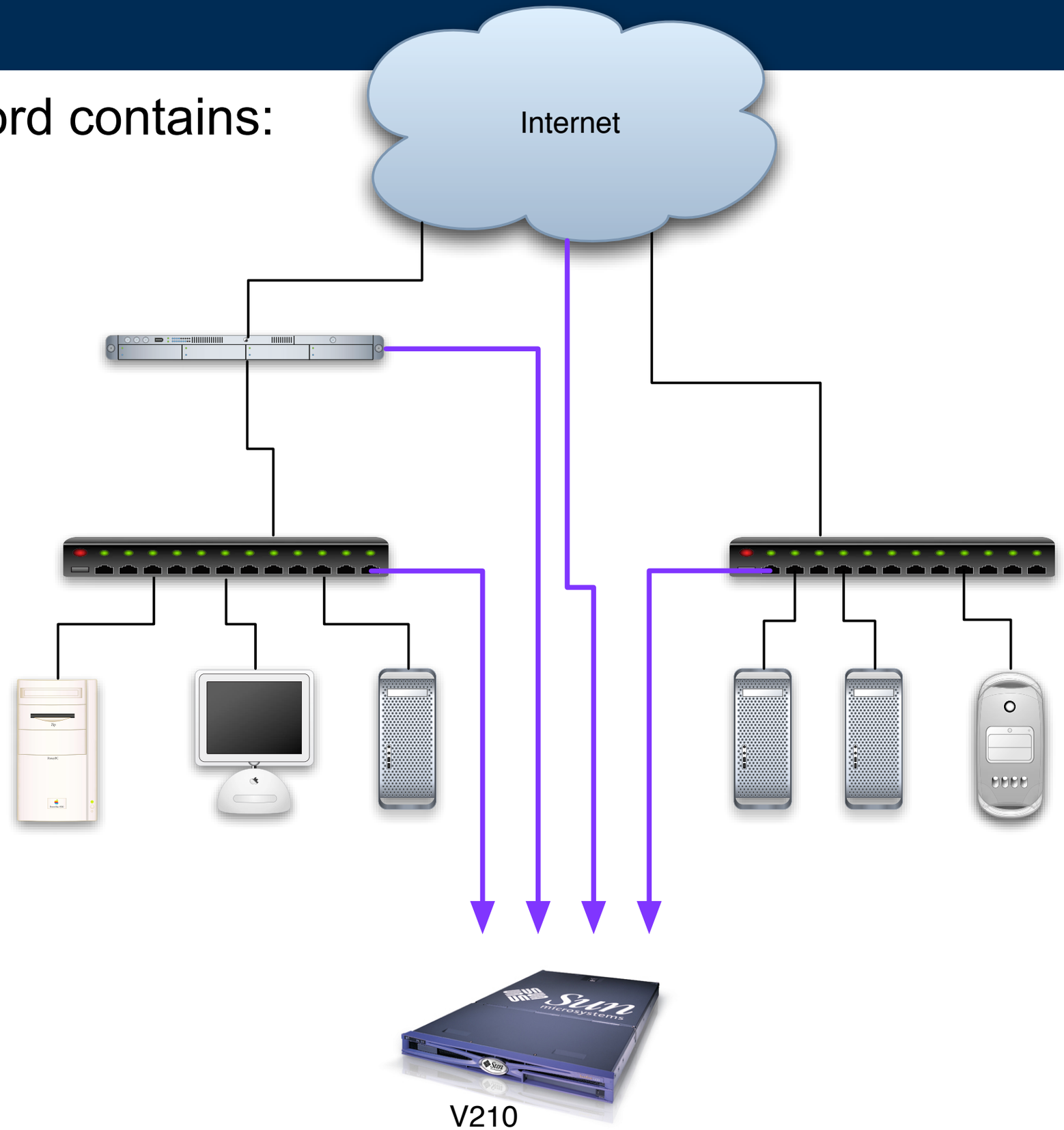


Count	Source	>	Destination
46	i7.cnn.net.http	>	192.168.1.102.58755
34	192.168.1.102.58755	>	i7.cnn.net.http
26	69.22.138.51.http	>	192.168.1.102.58776
24	www2.cnn.com.http	>	192.168.1.102.58754
21	192.168.1.102.58776	>	69.22.138.51.http
19	192.168.1.102.58765	>	i7.cnn.net.http
17	64.236.29.63.http	>	192.168.1.102.58758
17	192.168.1.102.58754	>	www2.cnn.com.http
16	i7.cnn.net.http	>	192.168.1.102.58765
14	192.168.1.102.58759	>	64.236.29.63.http
13	72.32.153.176.http	>	192.168.1.102.58769
13	192.168.1.102.58769	>	72.32.153.176.http
13	192.168.1.102.58758	>	64.236.29.63.http
12	64.236.29.63.http	>	192.168.1.102.58759
10	64.236.29.63.http	>	192.168.1.102.58778
10	64.236.29.63.http	>	192.168.1.102.58757

Many switches and routers will report “netfow” data directly.

Each Cisco NetFlow record contains:

- Total bytes & packets
- S&D IP addresses
- S&D ports (UDP or TCP)
- flags
- start & end time
- min & max packet size
- VLANs & ifaces
- Vendor proprietary data



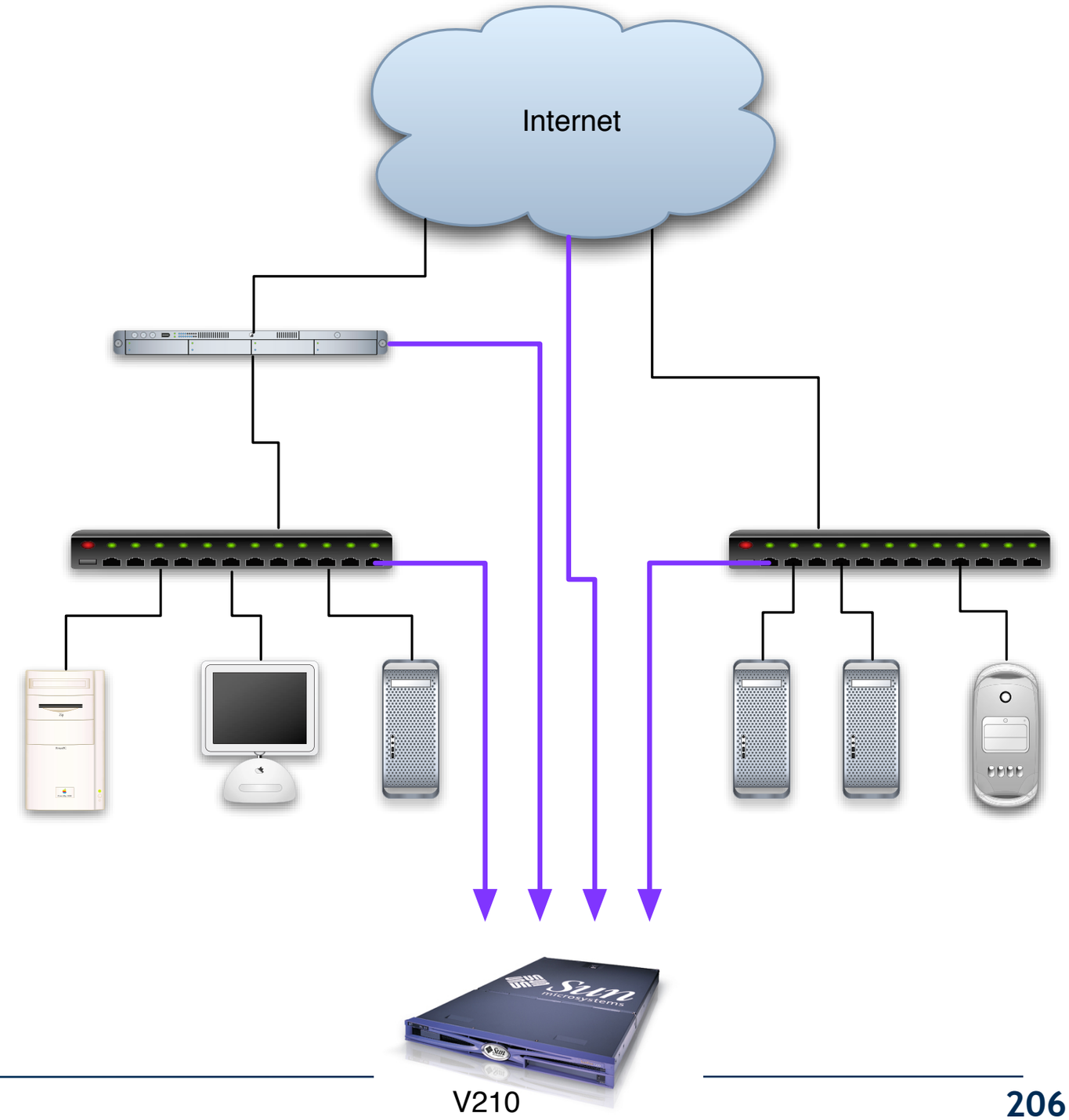
Flow-based systems are “blind” to data

Advantages:

- More economical
- Finds rogue servers and consultants
- More privacy-sensitive

Can’t discover:

- Missing encryption
- Inappropriate encryption
- Protocols on wrong ports
- Leaking specific documents



Flow data can still be a privacy problem

Flow data can reveal:

- When somebody went to work, left for home, etc.
- Which websites a person visited (but not perfectly).
- Applications that were used.

Flow data can be readily combined with other information:

- DHCP logs
- Mail logs

Tools for working with network data: Command line & GUI

Command-line tools are effective when:

- Working with a small amount of data (<100GB)
- Looking for a novel attack (something the GUIs don't recognize)
- With a skilled operator
- Mostly open source freeware

GUI Tools:

- Better for exploring large data sets
- Mostly proprietary

Command-Line Tools

Most Unix tools are based on **bpf** and **libpcap**

- Uniform way of getting packets from the network.
- Allows capture filters.
- Defines the "tcpdump" file format (header + (timestamp + packet)*)

Tool chest:

- **tcpdump** — User-level interface to libpcap
- **tcpflow** — Reassembles TCP streams (doesn't handle fragments)
- **tcpick** — Text-based TCP stream sniffer
- **tcpillust** — Graphical TCP illustration tool (requires X)
- **tcpshow** — Decodes all the headers
- **tcpslice** — extracting and combining TCPdump files

tcpdump can be used to capture packets, filter packets, or display their contents.

usage: tcpdump [-i interface] [-w output] [other options] [expression]

other useful options:

`-A`	Print each packet in Ascii
`-C 5`	Automatically roll over the output file every 5 MB
`-D`	Print interfaces on the system that tcpdump can use
`-N`	Don't print domain names, just host names.
`-w` *`file`*	Write to an output file.
`-r` *`file`*	Read from an input file.
`-s 4096`	Capture 4096 bytes of each packet (default is 68)
`-Z` *`user`*	Run the capture process as "user," rather than as root.

Typical expressions:

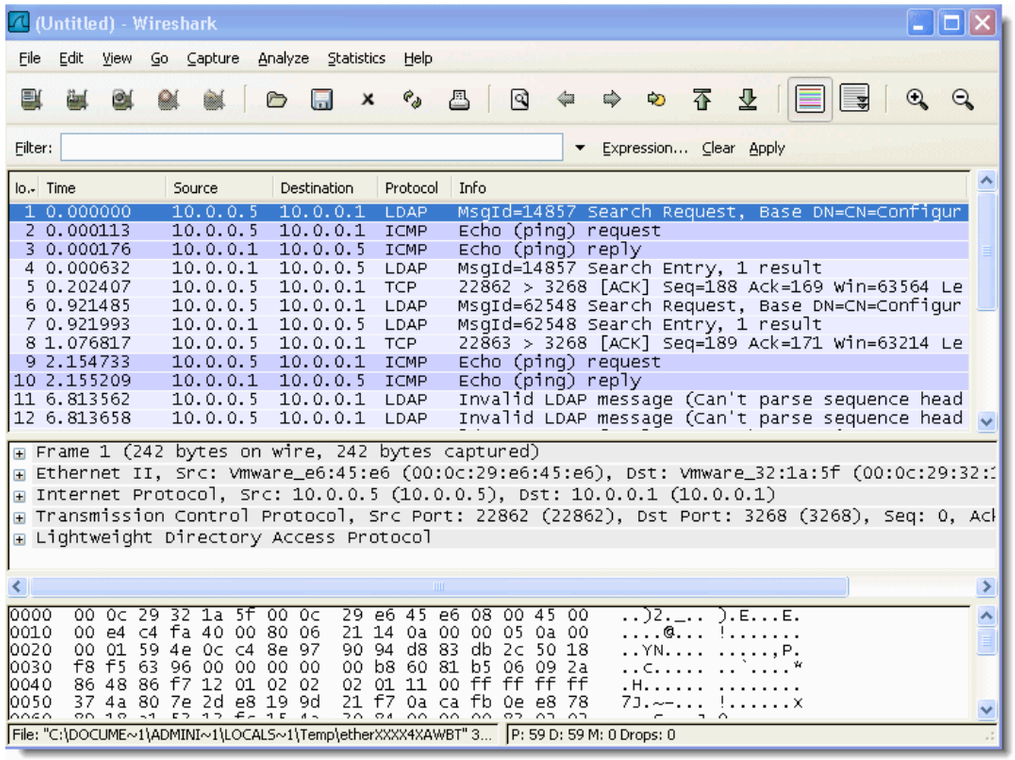
`host nitroba.com`	Packets to or from nitroba.com
`dst host nitroba.com`	Packets destined for nitroba.com
`src host nitroba.com`	Packets from nitroba.com
`port 52`	Packets for port 52
`udp`	Just the udp packets

Type "man tcpdump" for full information for your system.

Wireshark is mostly a GUI built on top of tcpdump. It captures packets and displays them in more detail.

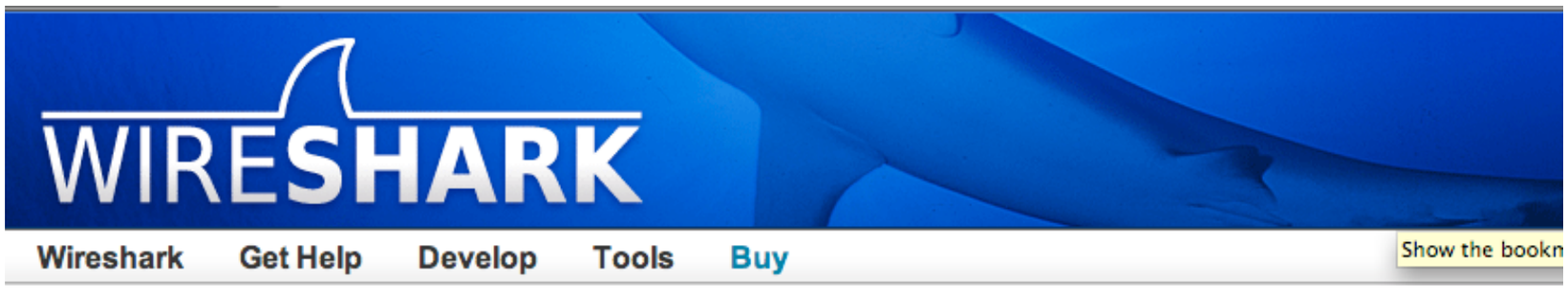
Advantages:

- Free
- Packet decoders for hundreds of packet types
- Decompresses compressed data on the fly
- Decrypts many protocols:
  - — *IPsec, ISAKMP, Kerberos, SMPv3,*
  - —*SSL/TLS, WEP and WPA/WPA2*



Disadvantages:

- Designed for packet analysis, not correlation.
- Only analyzes 1 tcp connection at a time.
- Flow reconstruction packet-by-packet is very time consuming.



Demo with real packets

Nitroba University Harassment Scenario Packet Dump file.

- http://domex.nps.edu/corp/scenarios/2008-nitroba/nitroba.pcap

Preview the file with tcpdump:

```
$ tcpdump -r nitroba.pcap -c 7
reading from file nitroba.pcap, link-type EN10MB (Ethernet)
21:51:07.095278 IP 192.168.1.64.42760 > 74.125.19.83.http: Flags [F.], seq
4261057042, ack 3039099121, win 65535, options [nop,nop,TS val 712729540 ecr
980517270], length 0
21:51:07.103728 IP 74.125.19.83.http > 192.168.1.64.42760: Flags [F.], seq 1,
ack 1, win 431, options [nop,nop,TS val 980523165 ecr 712729540], length 0
21:51:07.114897 IP 192.168.1.64.35011 > 74.125.19.19.http: Flags [P.], seq
2964083668:2964085019, ack 3683809881, win 65535, options [nop,nop,TS val
712729541 ecr 988904514], length 1351
21:51:07.139448 IP 74.125.19.19.http > 192.168.1.64.35011: Flags [.], ack
1351, win 393, options [nop,nop,TS val 988915614 ecr 712729541], length 0
21:51:07.319680 IP 74.125.19.19.http > 192.168.1.64.35011: Flags [P.], seq
1:1215, ack 1351, win 393, options [nop,nop,TS val 988915792 ecr 712729541],
length 1214
21:51:07.321990 IP 192.168.1.64.35011 > 74.125.19.19.http: Flags [.], ack
1215, win 65460, options [nop,nop,TS val 712729543 ecr 988915792], length 0
21:51:07.326517 IP 192.168.1.64.35011 > 74.125.19.19.http: Flags [F.], seq
1351, ack 1215, win 65535, options [nop,nop,TS val 712729543 ecr 988915792],
length 0
```

Use tcpflow to reconstruct the individual flows

```
$ ls -l
total 55720
-rw-r--r--  1 simsong  staff  57054792 Oct  6  2010 nitroba.pcap
$ tcpflow -r nitroba.pcap
$ ls -Tl | head -10 | sed s/^.............................//
total 117972
      3648 Jul 22 02:03:17 2008 004.071.104.187.00080-192.168.015.004.35950
       462 Jul 22 01:57:24 2008 004.078.212.029.00080-192.168.015.004.35458
       462 Jul 22 01:59:20 2008 004.078.212.029.00080-192.168.015.004.35712
    136520 Jul 22 00:29:55 2008 008.012.217.125.00080-192.168.015.004.32822
     23943 Jul 22 00:29:55 2008 008.012.217.125.00080-192.168.015.004.32824
     17995 Jul 22 00:29:55 2008 008.012.217.125.00080-192.168.015.004.32828
     20064 Jul 22 00:29:55 2008 008.012.217.125.00080-192.168.015.004.32830
       879 Jul 22 00:51:03 2008 008.012.221.123.00080-192.168.015.004.33298
       213 Jul 21 21:57:42 2008 012.129.147.065.00080-192.168.001.064.34023
$
```

Notice:

- tcpflow puts each side of the connection in its own file.
- Timestamps of the file are the time that the first packet was sent

Each packet flow tells a story

```
$ head 004.078.212.029.00080-192.168.015.004.35458
HTTP/1.1 302 Found
Connection: close
Date: Tue, 22 Jul 2008 05:57:45 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Pragma: no-cache
Location: http://mail.live.com/
Cache-Control: no-cache
Pragma: no-cache
$
```

This packet flow is an HTTP request

- To a http://mail.live.com/
- On July 22, 2008

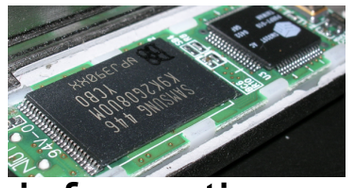
Embedded timestamps and tokens make it exceedingly difficult to create fake data.

Working with Android Phones

You've got an Android Phone. Now what?

SIM:

- Identity information.
- Possibly Address Book or SMS records from a previous phone
- On-board Flash (256M-2GB)



- Android file system (YAFFS2)
- Call history; messages; position information; network information; etc
- Downloaded applications & application data

Removable Flash (1GB-32GB)

- Downloaded applications & application data
- Media (songs; video; images); Documents
- Information from other computers (remember, phone can be a "thumb drive")

RAM (256M-1GiB)

- Linux; Dalvik (Java) VM; user programs
- May be only way to recover encryption keys, passwords, etc.

Two approaches for Android Forensics: Online & Offline

Online Analysis: Use Android to analyze Android

Enable USB debugging and debug with Android Debug Bridge (adb)

—*http://developer.android.com/guide/developing/tools/adb.html*

- Load an application that extracts data to your analysis machine
- RAM

—*Physical Dump of NAND flash*

Offline Analysis: Analyze Android as a storage system

- Analyze SDCard as a traditional FAT file system
- Logical analysis of YAFFS2 files

—*Less to get, but easier to get at*

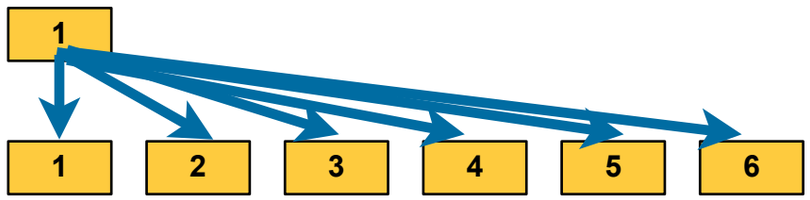
Which approach you choose depends on:

- Your goals — conviction, discovery, research
- Your skill level & available tools
- Legal requirements (i.e.: will the results be used in court?)

Flash memory is very different from traditional RAM.

Defining characteristics:

- Memory written in blocks (100s-1000s of bits per block — think “sectors”)
  —*Must be erased before it can be written*
- Memory erased in *pages* (10,000s of bits per page — think 4K pages)
- Each bit has limited lifetime (typically 1000 — 100,000 cycles)
- Therefore, writes must be *wear leveled*



NOR flash (not always present)

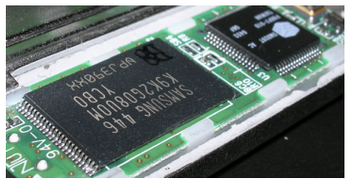
- True random access (direct execution)
- Low-density (expensive)
- Boot code can execute directly out of NOR

NAND flash (always present)

- Block-oriented access
- High density (Single Layer Cells & Multi Layer Cells)
- ROM boot code (in the microprocessor) can copy NAND into RAM and execute.

There are two-approaches for remapping.

File Level — Flash File System



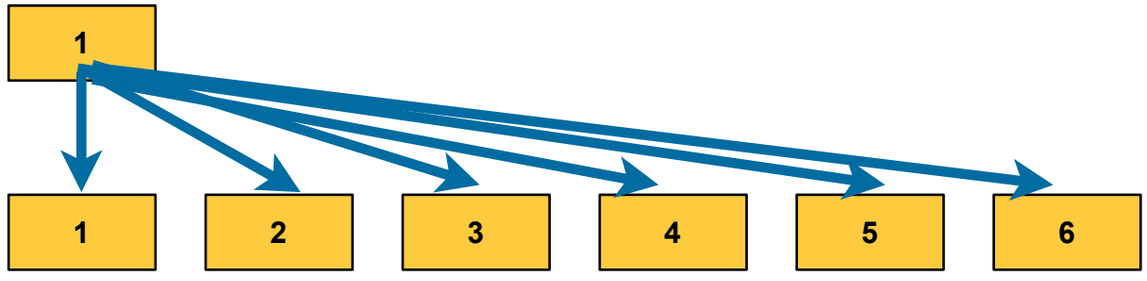
- Operating system directly controls writing & erasing.
- Files may be proactively moved to assist in leveling
- JFFS2 (Journaling Flash File System #2); YAFFS (Yet Another Flash File System); YAFFS2

Block Level — Flash Translation Layer

- Flash device appears as a block device
- Operating system rewrites as normal
- “Flash Translation Layer” transparently remaps & erases as necessary
- Used by all SD cards and SSDs

“TRIM” Command

- Tells FTL that a sector will not be read again
- Lets OS give SD/SSD “hint.”
- Implemented in Windows 7 and Linux ext4



Wear leveling means you can recover data *after it is deleted and overwritten*.

Assume this sequence of events:

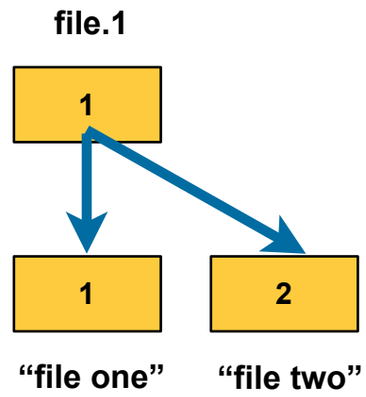
```
echo “file one” > file.1
echo “file two” > file.2
dd if=file1 of=file.2
```

These commands are executed at the *logical layer*

YAFFS2 would rewrite the directory entry for “file.2” to point at the new flash pages

- *A SSD or SDCard would rewrite the FTL so that the logical block # pointed to by the file.2 directory entry pointed to the new data*

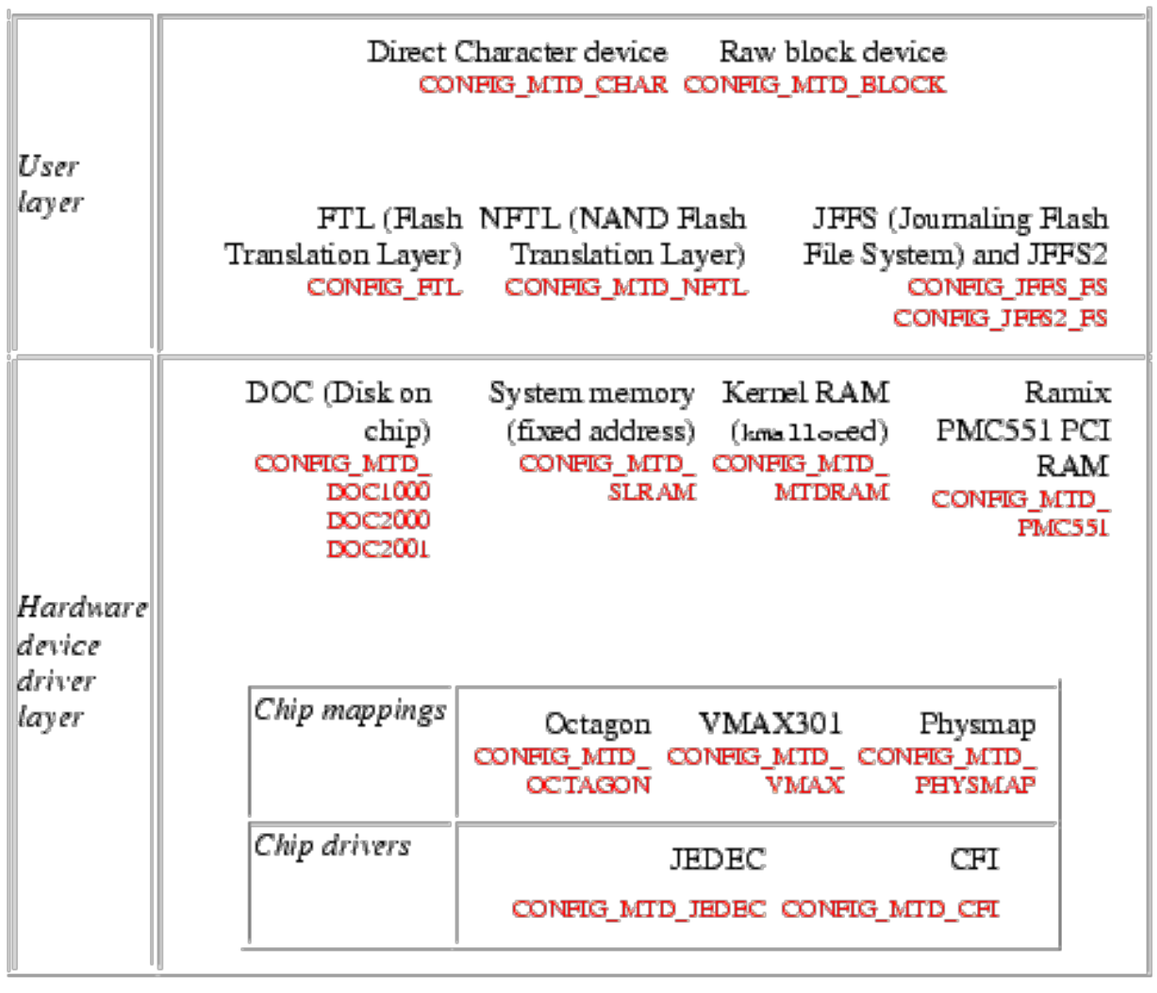
If you can access the *physical* layer, *you can recover the previous contents of file.2*



Android uses the Linux Memory Technology Device (MTD) to access flash memory.

The MTD has a Flash Translation Layer...

- ... but flash file systems (JFFS, JFFS2, YAFFS and YAFFS2) go directly to the hardware layer.



http://www.stlinux.com/howto/Flash/MTD

“Logical” vs. “Physical” dump.

A logical dump is a dump of the *records* or *files*

From data providers

- From walking the file system
- adb pull /*dir* local # don’t pull /proc

A physical dump is a dump of *sectors* or *pages*

YAFFS and YAFFS2:

—*raw is the individual flash pages*

—*16-bytes of Out-of-Band information stored every 512, 1024, or 2048 bytes must be removed*

—*Requires a YAFFS/YAFFS2 implementation to extract files*

—*FAT32 (or NTFS)*

—*raw is the individual disk “sectors” (512 or 4096 bytes)*

—*Requires FAT32 implementation to extract files*

—*Mount with a loop-back device to access allocated files*

- Use SleuthKit, EnCase, or FTK to access *deleted files*.

File formats typical on Android Phones

SQLite data files

- Public domain database holds SQL Schema, Tables, Rows, Columns
- Journal stored in secondary file
- Most of today's tools ignore the journal and deleted data

Internal log (circular buffer in memory)

- Log Collector (http://code.google.com/p/android-log-collector/)
- logcat

```
adb shell logcat > log.txt
```

- aLogCat

Text log files

- Some third party programs (e.g. DropBox) may store text logs
- Does the base Android system create text log files?

Using Sleuthkit for Android Forensics

Approach #1: MicroSD card

- Remove the MicroSD card and examine with SleuthKit
- Important: Use a *write blocker* to prevent modification to the SD card
- Advantage: Easy-to-do; no change to SD card
- Disadvantage: Will not read encrypted .apk files; shutting down may wipe important info

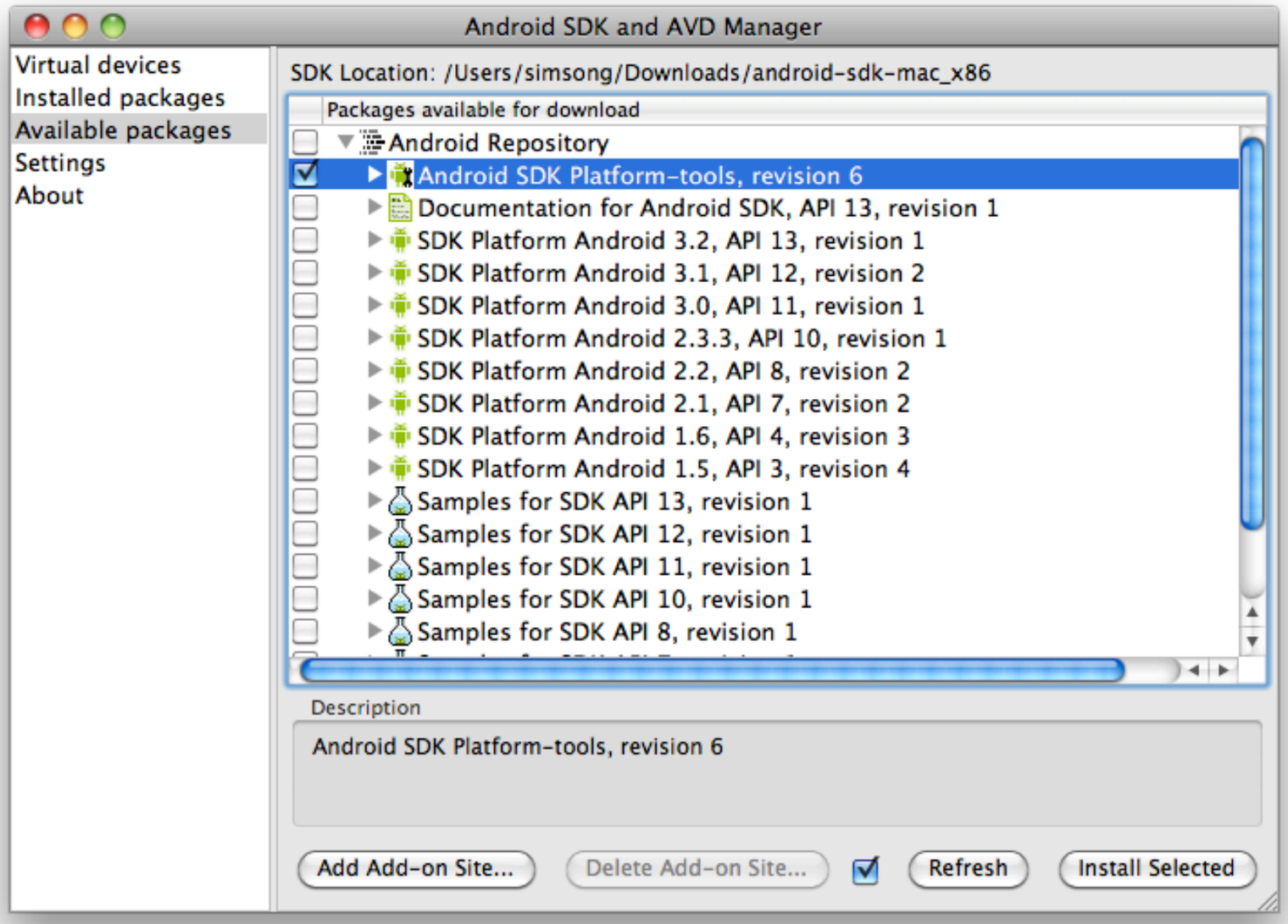
Approach #2: Analyze the Android device via USB

- Attach the Android device to your computer and select “USB Storage.”
- One or more partitions corresponding to the Android device *may appear*
- Question: Can we use a *write blocker* to prevent modification? (I don’t know)
- Advantage: Easy-to-do
- Disadvantage: May change Android device *even with write blocker*

Approach #3: Dump the Android device and analyze offline.

The easiest way to work with an Android Phone is with Google's Android Developer Kit

- Download the kit from developer.android.com
- Unzip the installer
- Run "tools/android" and install adb



To use “adb”, your phone *must* have USB debugging enabled.

Settings / Applications / Development / USB debugging

- NOTE: This allows anyone who has your phone to bypass the PIN lock.

Put platform-tools/ in your path.

```
$ export PATH=$PATH:./platform-tools/
$ adb devices
List of devices attached
HT163T524323          device
```

Execute remote commands with “adb shell”

```
$ adb shell ls -l
drwxr-xr-x root     system              2011-07-16 10:16 app-cache
dr-x------ root     root                2011-07-16 10:16 config
lrwxrwxrwx root     root                2011-07-16 10:16 sdcard -> /mnt/sdcard
drwxr-xr-x root     root                2011-07-16 10:16 acct
drwxrwxr-x root     system              2011-07-16 10:16 mnt
lrwxrwxrwx root     root                2011-07-16 10:16 etc -> /system/etc
drwxrwx--x system   system              2011-07-16 10:16 vendor
drwx------ root     root                2011-07-16 22:09 devlog
drwxrwx--- system   cache               2011-07-16 17:39 cache
-rw-r--r-- root     root          4311  1969-12-31 19:00 ueventd.rc
...
```

You can do a surprising amount of forensics with just a few commands

Look around:

```
$ adb shell ls -l
drwxr-xr-x root     system            2011-07-16 10:16 app-cache
dr-x------ root     root              2011-07-16 10:16 config
lrwxrwxrwx root     root              2011-07-16 10:16 sdcard -> /mnt/sdcard
...
$ adb shell lsof
...
```

Send files to Android:

```
$ adb push local remote
```

Get files from Android:

```
$ adb pull remote local
```

Ideas:

- Look for sqlite databases
- Root the phone to get access to all files.

Android Forensics References

- “Recovery of Deleted Data from Flash Memory Devices”, Capt. James Regan, Master’s Thesis, Naval Postgraduate School, 2009. http://simson.net/clips/students/09Sep_Regan.pdf
- “Android Forensics: Simplifying Cell Phone Examinations,” Lessard & Kessler, Small Scale Digital Device Forensics Journal, Vol. 4, No. 1, September 2010, http://www.ssddfj.org/papers/SSDDFJ_V4_1_Lessard_Kessler.pdf
- http://viaforensics.com/category/android-forensics/
- http://viaforensics.com/android

Concluding Remarks

Now you can work with computer forensics data!

What you learned:

- Digital Forensics is like a magic camera, but it can be easily faked
- There are many kinds of forensics data—more than people know how to analyze
- Everything is getting harder

Tradeoffs:

- Logical vs. Physical dumps
- Allocated vs. Unallocated data
- Live vs. Dead acquisition and analysis

Techniques:

- Metadata analysis and extraction
  - —*Walking file systems with SleuthKit*
  - —*Digital Forensics XML*
- Carving and Bulk Data Analysis
- Conversion to HTML and PDF

There are many data types

- Data that we worked with:
  - —*Disk Images and Digital Forensics XML*
  - —*Android Phones*
  - —*IP Packets*
  - —*SQLite databases*

- Data types we did not explore:
  - —*Windows registry.*
  - —*Internal elements in multimedia files (JPEG, MOV, etc)*
  - —*Machine Code*

Other resources

Presentations you may find useful:

- Day-long Forensics Tutorial at ACSAC 2009
  —*http://simson.net/ref/2009/ACSAC%202009%20forensics.pdf*
- This presentation, online:
  —*http://simson.net/ref/2011/2011-07-20%20Working%20with%20Forensic%20Data.pdf*
- Bulk_Extractor:
  —*http://simson.net/ref/2011/2011-06-14%20bulk_extractor.pdf*
- Android Forensics:
  —*http://simson.net/ref/2011/2011-07-12%20Android%20Forensics.pdf*

Other websites:

- Digital Corpora - http://digitalcorpora.org
- AFFLIB, source for fiwalk & bulk_extractor — http://afflib.org/
- Forensics Wiki — http://forensicswiki.org/
- Open Source Forensics Database — http://www.opensourceforensics.org/
- SleuthKit — http://sleuthkit.org/