Digital Forensics Tool Integration
28 JUN 2012

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

NPS is the Navy's Research University.

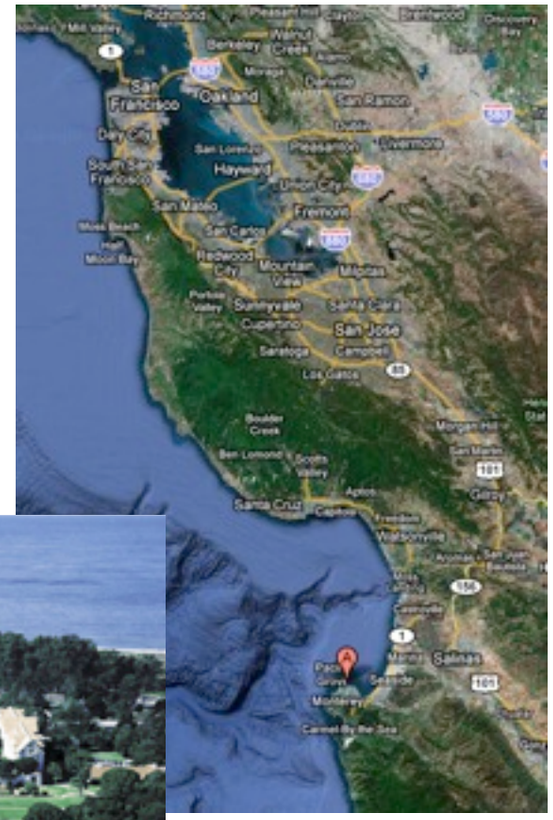
Location: Monterey, CA

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
- **"Residence" option for USG PhDs @ NPS.**

Current NPS research thrusts in digital forensics

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

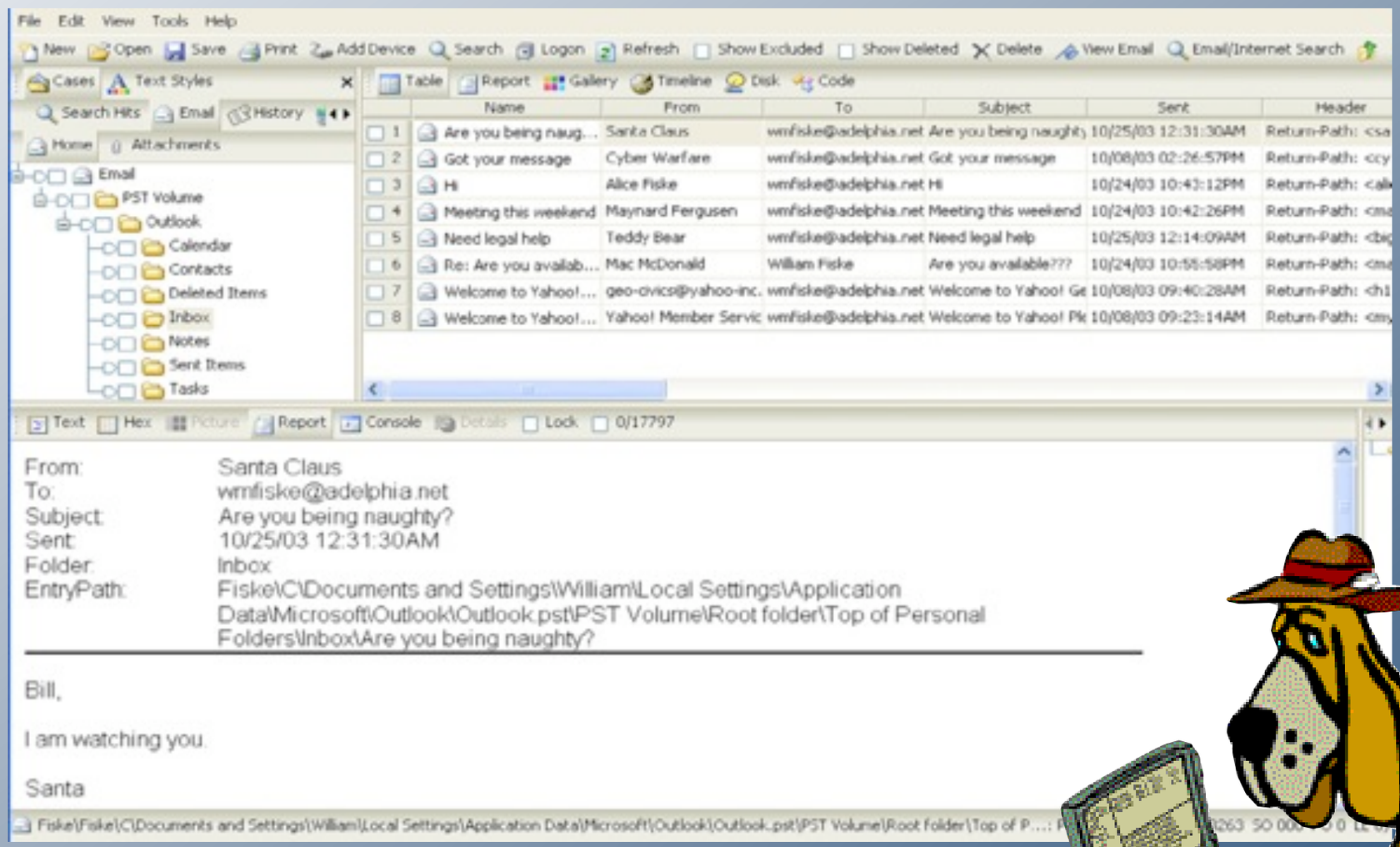
- Digital Forensics XML (DFXML)
- Disk Image ⇛ Power Point

Area #2: Bulk data analysis

- File system agnostic feature recovery (bulk_extractor)
- Statistical techniques (sub-linear algorithms)
- Similarity metrics

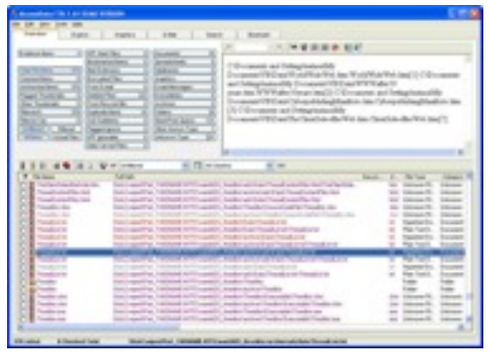
Area #3: Bringing “science” to digital forensics

- Forensic Corpora: redistributable disk images, packet dumps, file collections.

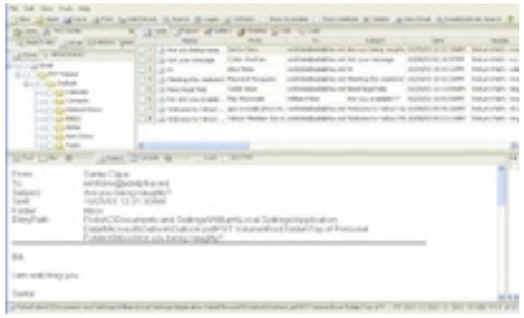


Forensic Tool Integration Today

Will FTK, EnCase, and SleuthKit find the same files on the same disk image?



FTK file set

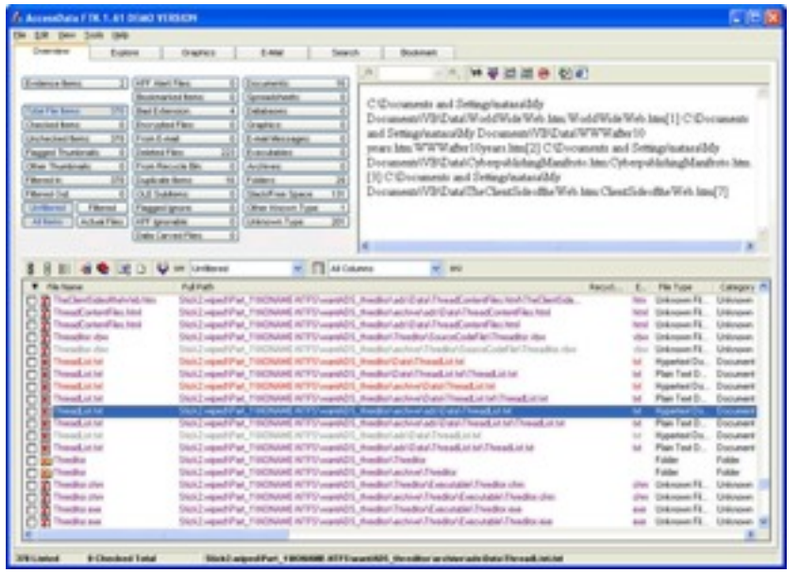


EnCase file set

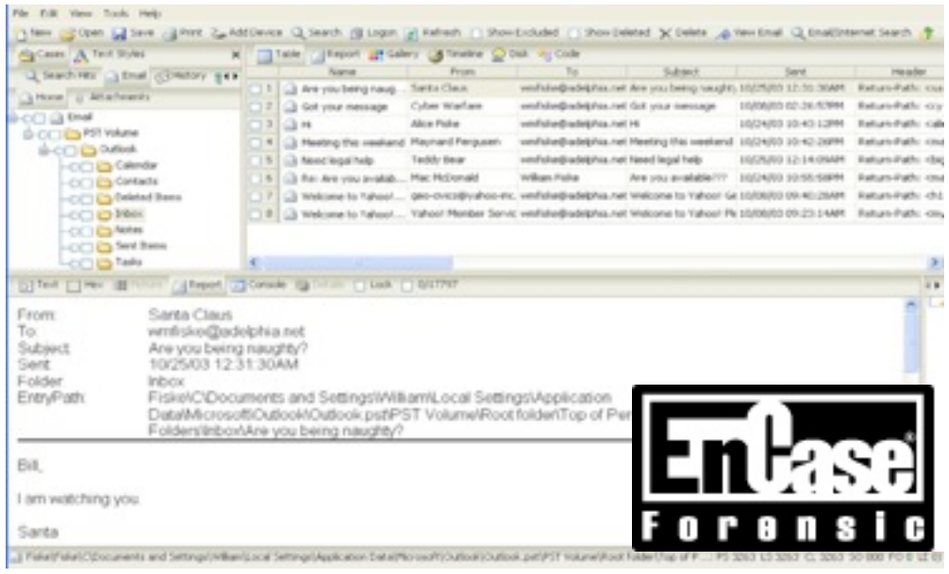
TSK file set

This is a surprisingly hard experiment to perform.

Commercial forensic tools are designed for investigations.



AccessData FTK



Guidance Software Encase

These tools are great for:

- File recovery
- Search
- Looking at disk sectors

Not so great for automation, interoperability, or research:

- Non-standard “scripting” languages.
- No standard format for reporting exploitation/extraction results

Open Source tools allow some automation; integration is mostly through text files

SleuthKit:
Open Source
Command-line C/C++
GUI Java

RegRipper: perl

RegRipper
www.RegRipper.wordpress.com

These tools are great for:

- File recovery
- Timeline generation
- Automating specific tasks with scripting — many language choices!

These tools lack interoperability and integration.

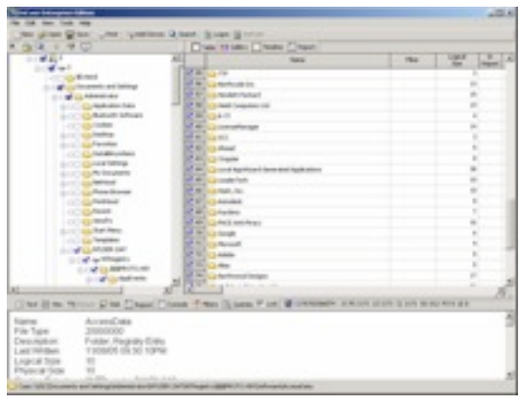
- No standard format for reporting exploitation/extraction results

Today's forensic pipeline is a three-step process.

Data

- Disk Images
- pcap files
- "files" (frequently in a ZIP archive)
- Lists of hash values (MD5 codes)

Tools



- EnCase (e-scripts)
- FTK (windows automation)
- Custom-written tools

Output

- Word, Excel & PDF files
- Text Files

Forensic input files are large.
Inputs require frequent re-processing

Disk Images:

- Copying TBs takes time and is increasingly error-prone.
- File extraction has to be repeated (and needs to be repeatable)

PCAP processing — done differently by different tools.

- Sessionization
- Decompression
- Decryption

Hash Sets

- No systematic way for labeling or annotating.
- Fortunately, we're all using MD5…

TSK 3.2 introduced tsk_loaddb, a tool for saving file information into an SQLite 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 1000,1;
1|11399|128|5|download-page[2].css|10243|5|1|1|5|790|1224542464|1224542464|
1224542674|1224542464|511|0|0
sqlite>
```

But SQLite is not a good match for many applications.

- Limited schema; hard to extend

Work product has limited use because of file formats

Microsoft Office Files (Word, Excel, PPT)

- Good for briefings.
- Excel has high value for analysts, but not for tool integration.

Adobe Portable Document Format

- Hard to recover data (“Dead Data”)
- Allows attaching files, but few people use this feature.

We rarely use these *output* as *input* for other tools.

We need higher-level *formats* and *abstractions*.

Signatures

- 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

- Friends lists

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

Forensic outputs lack *provenance*.

Provenance is a “history of ownership or location.”

- Long history of provenance in the art community.
- Increasingly used in scientific computing.

In forensic processing, we would like to track:

- Where we got the data
  - —*Sources*
  - —*Classification levels*
- Tools that processed the data
  - —*Version of the tool.*
  - —*Compiler used to compile the tool.*
  - —*Libraries linked with the tool.*
- Computer(s) on which processing was done.
  - —*Operating System*

We track some of this information now, but not systematically.

This talk presents Digital Forensics XML, an XML language for integrating forensic tools.

Why we need DFXML

DFXML's Structure

<dfxml>

Writing DFXML programs with dfxml toolkit

Tools that generate DFXML

<dfxml>

<fileobject>
<creator>

Digital Forensics XML solves forensic integration issues.

DFXML can be used to represent many kinds of forensic information

- Disk images — where did it come from?
- Files, file system metadata, MAC times, file hashes, sector hashes, etc.
- TCP flows
- Hash sets

DFXML provides provenance:

- ***Where the data came from; classification and use restrictions***
- ***Which tools were used; how the tools were made***.

Integration:

- Tools can both generate and consume DFXML.

DFXML powers research and operations

- Students at NPS — build projects without understanding EnScript, FTK or TSK
- Support has been added to numerous tools.

Why XML?

We aren't <religious> about XML, but...

- More programmers speak XML than "forensics."
- DFXML tags are reasonably self-documenting
- Creating an XML for forensics was very straightforward.
- As our tools improve, we can add new XML tags.
  - —*Namespaces can prevent tag conflicts.*
  - —*Old tools will ignore tags they don't understand.*

DFXML is an informal XML (at least for now)

- XML documents must be syntactically correct.
- "Validation is in the eye of the beholder."
- Experience tells us which tags and constructs are useful.

DFXML complements other XML forensics efforts

- MITRE — MACE
- Mantech OpenIOC

Goals for DFXML

1. Complement existing forensic formats.
2. Be easy to generate
3. Be easy to ingest
4. Provide for human readability
5. Be free, open and extensible
6. Provide for scalability
7. Adhere to existing practices and standards

Status today: DFXML is in production.

We have XML tags that describe:

- Files, file metadata, application metadata
- Hash codes.
- Partitioning schemes.
- TCP streams

We have toolkits:

- Python (2.7 & 3.2) modules for reading DFXML.
- C++ code for generating DFXML (including provenance) and reading DFXML

We have support:

- fiwalk, bulk_extractor, frag_find
- md5deep, sha1deep, hashdeep, etc.
- PhotoRec, StegCarver

Most of all, we have a good feeling about this.

We use the same XML tags in many different applications.

We have an API that makes it *easy* to do forensic processing.

We are finding bugs in programs when we add support for DFXML.

We are using DFXML files to debug programs we are developing.

DFXML structure

<dfxml>

<metadata> ***Dublin Core Metadata*** </metadata>

<creator> ***The program that made this DFXML*** </creator>

<configuration> ***Runtime Configuration*** </configuration>

<fileobjects>

<fileobject> ***Information about a file*** </fileobject>

</fileobjects>

<rusage> ***Runtime CPU usage information*** </rusage>

</dfxml>

<creator> provides provenance

```
<creator version='1.0'>
  <program>BULK_EXTRACTOR</program>
  <version>1.1.0_beta8</version>
  <build_environment>
    <compiler>GCC 4.2</compiler>
    <compilation_date>2011-11-19T23:27:21</compilation_date>
    <library name="afflib" version="3.6.9"/>
    <library name="libewf" version="20100805"/>
    <library name="exiv2" version="0.21.1"/>
  </build_environment>
  <execution_environment>
    <cpuid>
      <identification>GenuineIntel</identification>
      <family>6</family>
      <model>5</model>
      <stepping>5</stepping>
      <clflush_size>64</clflush_size>
      <nproc>16</nproc>
      <L1_cache_size>262144</L1_cache_size>
    </cpuid>
    <command_line>src/bulk_extractor -o dell1 /corp/drives/nist/nist-2004-
hacking/4Dell Latitude CPi.E01</command_line>
    <uid>501</uid>
    <username>simsong</username>
    <start_time>2011-11-20T04:34:27Z</start_time>
  </execution_environment>
</creator>
```

<fileobject> presents information about a file.

A “file” is a set of 0 or more bytes and metadata.

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

Can be on a disk, in a hash set, sent over a network, in an archive,...

```
<fileobject>
  <filename>casper/filesystem.manifest-desktop</filename>
  <filesize>32672</filesize>
  <inode>651</inode>
  <meta_type>1</meta_type>
  <mode>511</mode>
  <nlink>1</nlink>
  <uid>0</uid>
  <gid>0</gid>
  <mtime>2008-12-29T01:33:32Z</mtime>
  <atime>2008-12-28T05:00:00Z</atime>
  <crtime>2008-12-29T01:33:32Z</crtime>
  <byte_runs>
   <byte_run file_offset='0' fs_offset='5577728' img_offset='5609984' len='32672'/>
  </byte_runs>
  <hashdigest type='md5'>bd1b0831fcba1f22eff2238da96055b6</hashdigest>
  <hashdigest type='sha1'>7e072af67f8d989cc85978487b948048ac3c7234</hashdigest>
</fileobject>
```

A TCP flow is a file with <tcpflow> information.

```
<fileobject>
  <filename>074.125.019.104.00080-192.168.001.102.50955</filename>
  <filesize>2792</filesize>
  <tcpflow startime='2008-10-06T13:54:54.638913Z'
          endtime='2008-10-06T13:54:54.638913Z'
          src_ipn='74.125.19.104'
          dst_ipn='192.168.1.102'
          packets='6'
          srcport='80' dstport='50955'
          family='2' out_of_order_count='3' />
</fileobject>
<fileobject>
  <filename>192.168.001.102.50955-074.125.019.104.00080</filename>
  <filesize>655</filesize>
  <tcpflow startime='2008-10-06T13:54:54.621853Z'
          endtime='2008-10-06T13:54:54.621853Z'
          src_ipn='192.168.1.102'
          dst_ipn='74.125.19.104'
          packets='7'
          srcport='50955' dstport='80'
          family='2' out_of_order_count='0' />
</fileobject>
```

The <filename> is where the file's bytes are in the file system.

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>
```

Additional annotations can be added as needed

We can start with a simple representation:

```
<fileobject>
  <filename>casper/filesystem.manifest-desktop</filename>
  <hashdigest type='md5'>bd1b0831fcba1f22eff2238da96055b6</hashdigest>
</fileobject>
```

...and add additional information later...

```
<fileobject>
  <filename>casper/filesystem.manifest-desktop</filename>
  <filesize>32672</filesize>
  <hashdigest type='md5'>bd1b0831fcba1f22eff2238da96055b6</hashdigest>
  <hashdigest type='sha1'>7e072af67f8d989cc85978487b948048ac3c7234</hashdigest>
</fileobject>
```

<byte_runs> describes where the data is located.

```
<byte_runs>
  <byte_run offset="0"     img_offset="114688"  len="4096"/>
  <byte_run offset="4096"  img_offset="1523712" len="4096"/>
  <byte_run offset="8193"  img_offset="6356992" len="3512"/>
</byte_runs>
```

These can be be used for sector hashes:

```
<byte_runs>
  <byte_run offset="0"     img_offset="114688"  len="4096">
     <hashdigest type="md5">e07910a06a086c83ba41827aa00b26ed</hashdigest>
  </byte_run>
  <byte_run offset="32768" img_offset="1523712" len="4096"/>
     <hashdigest type="md5">9a8ad92c50cae39aa2c5604fd0ab6d8c</hashdigest>
  </byte_run>
  <byte_run offset="65536" img_offset="6356992" len="3512"/>
     <hashdigest type="md5">2b00042f7481c7b056c4b410d28f33cf</hashdigest>
  </byte_run>
</byte_runs>
```

<byte_run> can have both `img_offset` and `fs_offset`:

- Redundant, but it makes processing easier!

<msregistry> describes Windows registry values

```
<msregistry>
  <key name="HKEY_CURRENT_USER" root="1">
    <key name="Console">
      <mtime>2010-03-24T10:10:10-04:00Z</mtime>
      <value name="ColorTable00" type="REG_DWORD" value="0"/>
      <value name="ColorTable01" type="REG_DWORD" value="8388608"/>
...
      <value name="WindowSize" type="REG_DWORD" value="1638480"/>
      <value name="WordDelimiters" type="REG_DWORD" value="0"/>
    </key>
  </key>
</msregistry>
```

Key things that we can represent:

- Registry key modification times.
- Physical location of registry entries within the hive or in unallocated space.
- Disconnected registry entries.

We can build dramatically more powerful registry analysis programs by splitting the work between multiple programs (and programmers).

DFXML uses ISO8601 timestamps (e.g. 2010-03-24T10:10:10-04:00Z)

ISO 8601 has significant advantages over Epoch-based timestamps

- Represent times with or without time zones
- Can represent leap seconds
  - —*1997-06-30T23:59:60Z*
  - —*1998-12-31T23:59:60Z*
- Can represent clocks that are set incorrectly.
- Can represent systems with improper timezone handling.

Apparent disadvantages:

- Timestamps are 20 bytes instead of 4 (or 8) bytes.
- Processing timestamps takes longer (but the added time is not significant).

In our testing, these are not significant disadvantages.

Using DFXML

Generating DFXML
Consuming DFXML

There are several ways to use DFXML

Provenance <creator>

- DFXML C++ class (Mac, Linux & Windows)

File Objects (<fileobject>) can be produced with many tools

- fiwalk — uses SleuthKit to extract <fileobjects> from disk images
- md5deep (v4.0) — Hashes files and generates DFXML files
- PhotoRec & StegCarver — DFXML file reports where files were found
- frag_find — hash-based carving tool

DFXML can be consumed with programs written in Python or C++

- A SAX-based system allows rapid processing of gigabyte-sized XML files
- Same API for Python & C++ allows for rapid prototyping and then easy scaling up

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*

  —*Can be memory intensive with a lot of objects*

- DOM procedural interface

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

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

  —*Slow and memory intensive.*

C++ interface mirrors the SAX callback interface

The <fileobject> XML tag maps to Python and C++ fileobject classes.

The Python **`dfxml.fileobject`** class is an easy-to-use class for working with <fileobject> data.

Objects are created when the </fileobject> tag is processed

- Provided as arguments to a callback.
- Automatically garbage collected if not stored

The class supports a wide range of method calls:

```
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()
```

The callback structure allows processing gigabyte-sized XML files.

Example: igrep.py — grep through a disk image

```
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.

```
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

Generating DFXML

fiwalk extracts <fileobjects> 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)

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.

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.

XML namespaces can be used to prevent conflicts.

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 — 2GB
  - —*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.*

Other ways to generate DFXML

hashdeep, md5deep, sha1deep

- Hashing programs originally written by Jesse Kornblum
- Multithreaded since v4.0
- Create DFXML files of file names, sizes, MAC times, and hash values
- Only allocated files

tcpflow

- tcpip session reconstruction originally written by Jeremy Elson

PhotoRec & StegCarver

- Carving tools
- Use DFXML to say where the files were recovered from
- Allows results of different carving tools to be cross-validated.

Find out more about DFXML!

Download the paper:

- http://simson.net/clips/academic/2012.DI.dfxml.pdf

Try out the fiwalk program:

- https://github.com/kfairbanks/sleuthkit

Add support to your software!

bulk_extractor 1.3 status report!

Stream-based forensics:
Scan the disk from beginning to end; do your best.

0 → 1TB

3 hours, 20 min to *read* the data

1. Read all of the blocks in order.
2. Look for information that might be useful.
3. Identify & extract what's possible in a single pass.

Primary Advantage: Speed

No disk seeking.

Potential to read and process at disk's maximum transfer rate.

Potential for intermediate answers.

Reads all the data — allocated files, deleted files, file fragments.

- Separate metadata extraction required to get the file names.

bulk_extractor and DFXML

DFXML is used for:

- Provenance — On the cluster, we re-run bulk_extractor as necessary
- Performance testing — Each module reports # of calls, time
- Checkpointing and restarting

DFXML is *not* used for:

- Feature Files — We wanted a file format that was compact and easy-to-parse
- Three fields on each line
  - —*Offset*
  - —*Feature*
  - —*Context*
  - —*[File name]*
- In retrospect, we probably should have used XML!

San Luis Obispo
Sincerely, California.

http://www.sanluisobispovacations.com/

A bulk_extractor Success Story

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 defends were unsophisticated and lacked knowledge.

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

- Typically, it would take several days to conduct a proper forensic investigation.

bulk_extractor found actionable evidence in 2.5 hours!

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

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.

Faster than conventional tools. Finds data that other tools miss.

Runs 2-10 times faster than EnCase or FTK *on the same hardware.*

- bulk_extractor is multi-threaded; EnCase 6.x and FTK 3.x have little threading.

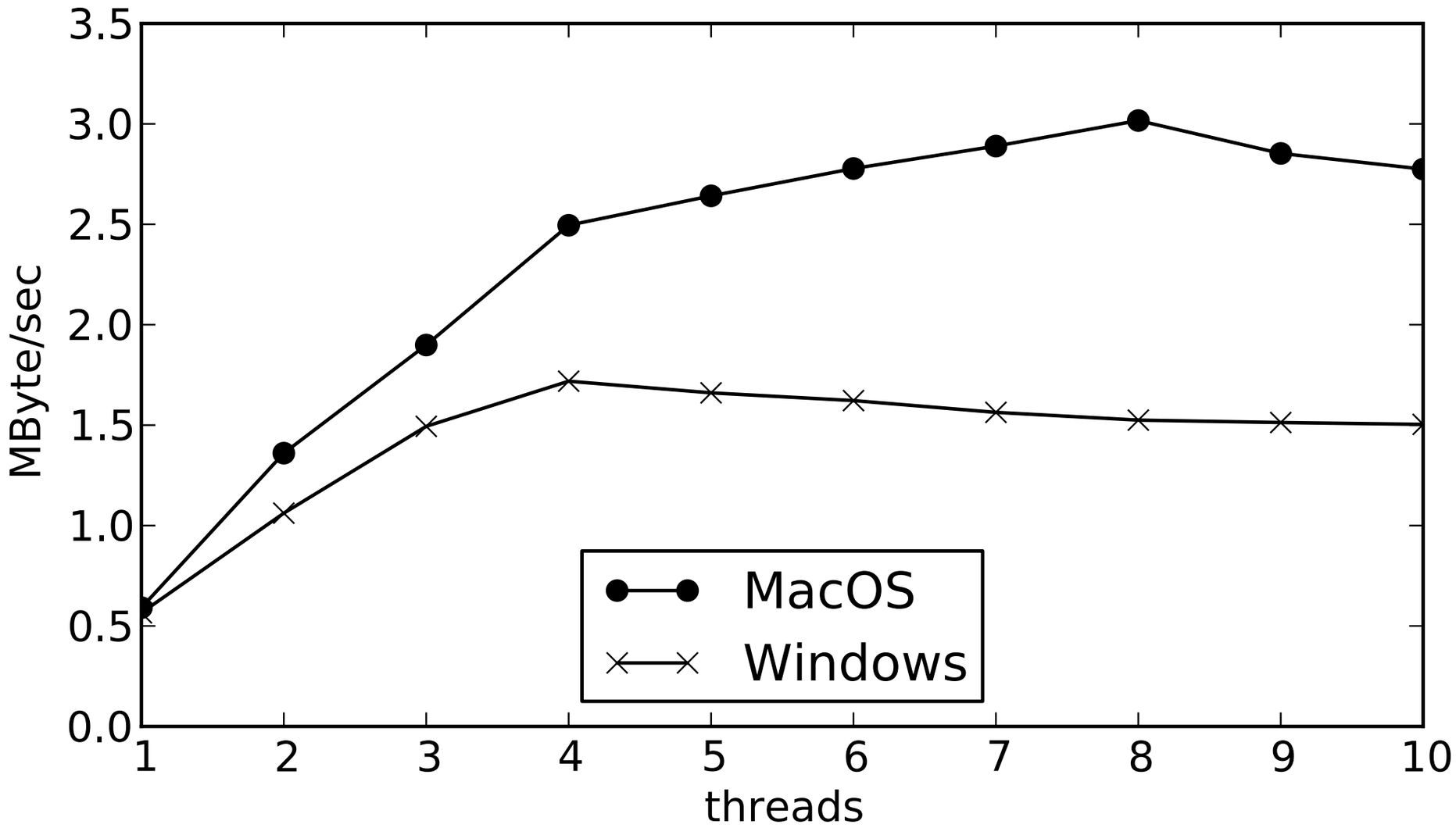
Finds stuff others miss.

- “Optimistically” decompresses and re-analyzes all data.
- Finds data in browser caches (downloaded with zip/gzip), and in many file formats.

Presents the data in an easy-to-understand report.

- Produces “histogram” of email addresses, credit card numbers, etc.
- Distinguishes primary user from incidental users.

bulk_extractor performance scales linearly with # cores.



MacBook Pro with 4 physical cores, 4 hyper-threaded cores

bulk_extractor has *multiple* feature extractors. Each scanner runs in order. (Order doesn't matter.)

Scanners can be turned on or off

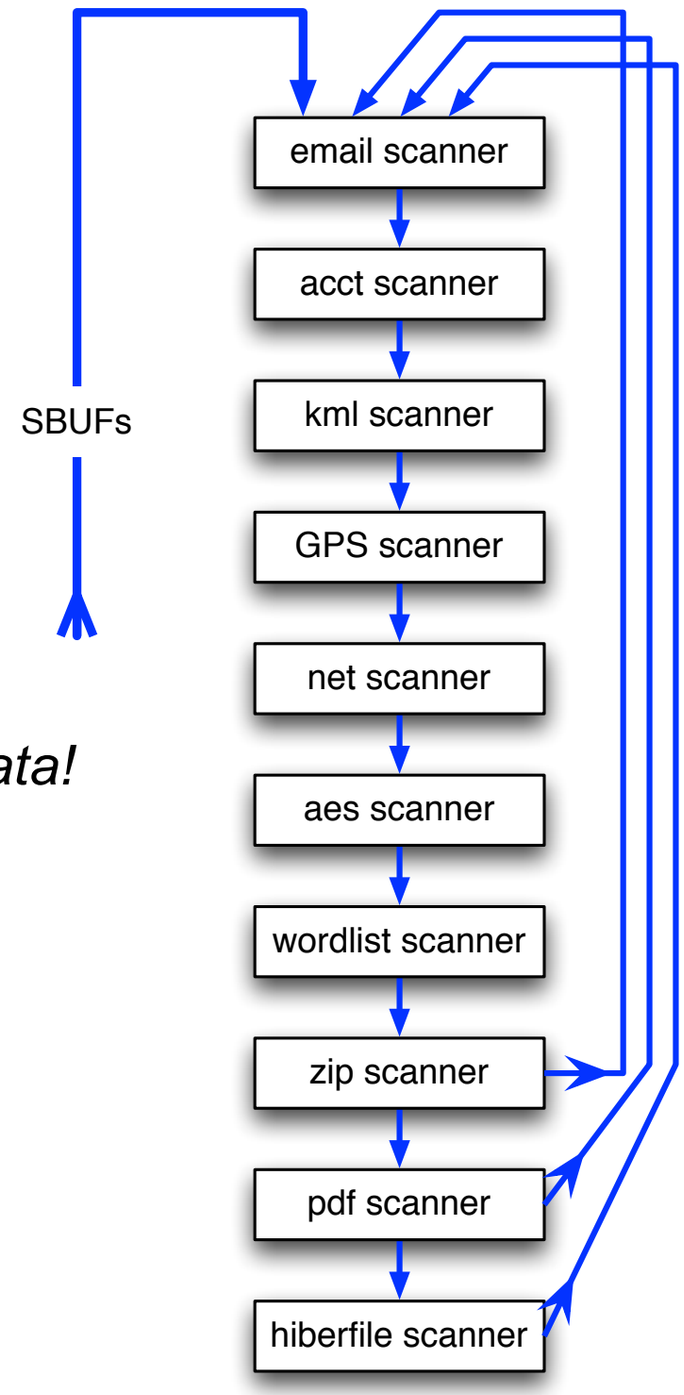
- Useful for debugging.
- AES key scanner is *very slow* (off by default)

Some scanners are *recursive.*

- *e.g.* scan_zip will find zlib-compressed regions
- An **sbuf** is made for the decompressed data
- The data is re-analyzed by the other scanners
  - —*This finds email addresses in compressed data!*

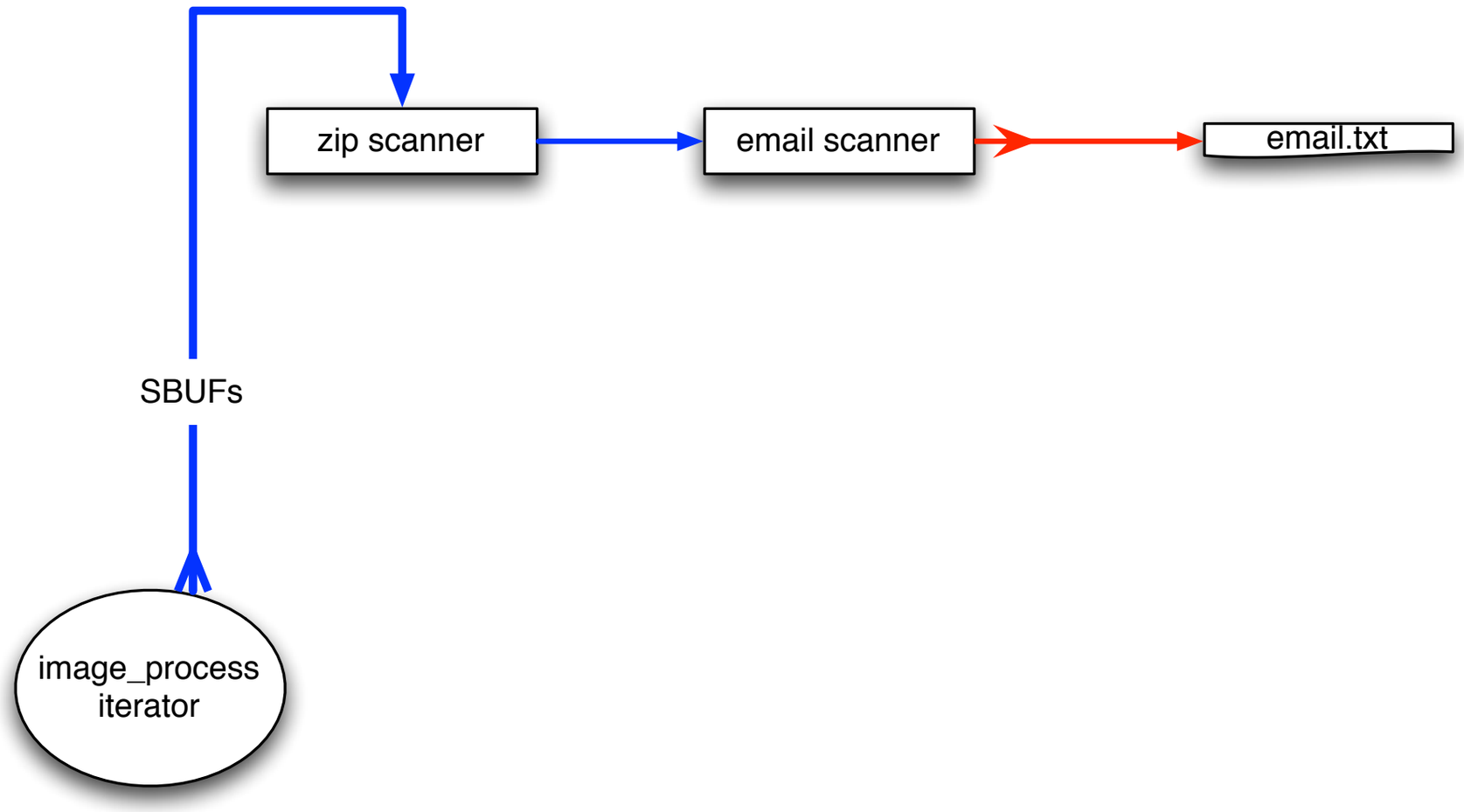
Recursion used for:

- Decompressing ZLIB, Windows HIBERFILE,
- Extracting text from PDFs
- Handling compressed browser cache data



Recursion requires a *new way* to describe offsets. bulk_extractor introduces the "forensic path."

Consider an HTTP stream that contains a GZIP-compressed email:



We can represent this as:

```
11052168704-GZIP-3437   live.com    eMn='domexuser1@live.com';var srf_sDispM
11052168704-GZIP-3475   live.com    pMn='domexuser1@live.com';var srf_sPreCk
11052168704-GZIP-3512   live.com    eCk='domexuser1@live.com';var srf_sFT='<
```

bulk_extractor: current status and future goals

Scanners:

- accts
- aes
- base64
- ccns
- email headers
- **elf**
- exif
- find
- gps
- gzip
- net
- hiberfile
- httpheader
- json
- kml
- net
- pdf
- **vcard***
- winprefetch
- wordlist
- zip
- **windir***
- **winpe**

Currently under development:

- bulk (detects encrypted data)
- RAR, RAR2
- LZMA
- BZIP
- Improved handling of Unicode.
- NTFS

bulk_extractor: get it today!
Download from http://afflib.org/

AFFLIB
Open Source Computer Forensics Software

HOME | CONTACT | ABOUT

Welcome to AFFLIB.ORG

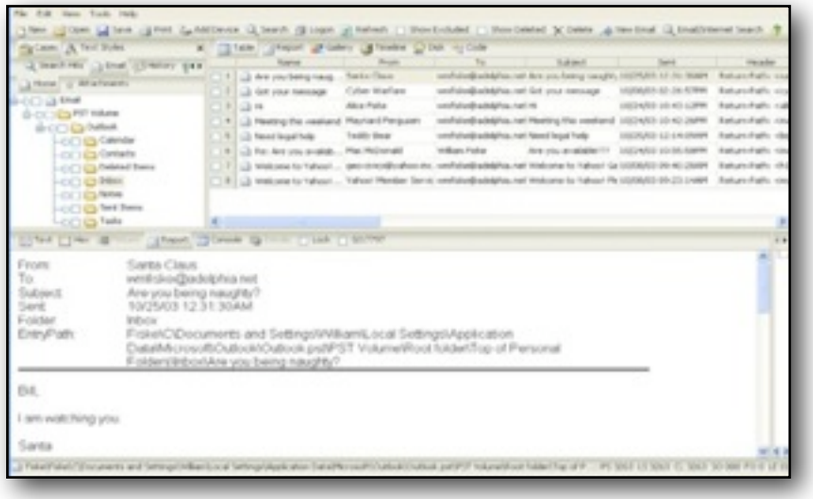
This server is the distribution site for currents and archival releases of forensic software by Simson L. Garfinkel. All of the software distributed at this server is either covered by a liberal Open Source license agreement or is in the public domain.

Name	Description	Lifetime Downloads	Download Current Version
AFFLIB	AFF Library and Tools	37473	afflib-3.6.12.tar.gz *Mon, 23 May 2011 22:16:41 +0000*
	Pre-compiled Windows Executables		afflibwin-3.5.0.zip *Sun, 17 Jan 2010 00:21:58 +0000*
bloom	NPS Bloom filter package (includes frag_find)	4882	bloom-1.4.6.tar.gz *Sun, 10 Apr 2011 17:55:35 +0000*
fiwalk	File and Inode Walk Program	6718	fiwalk-0.6.15.tar.gz *Mon, 18 Jul 2011 03:38:43 +0000*
Bulk Extractor	Bulk Email and URL extraction tool	7830	bulk_extractor-1.1.0.tar.gz *Mon, 28 Nov 2011 16:45:07 +0000*
	pre-compiled for Windows		bulk_extractor-windows-1.1.0.zip *Mon, 28 Nov 2011 16:39:21 +0000*
	Context Stop List		feature_context.1.0.zip *Mon, 27 Sep 2010 11:16:56 +0000*
ATA Raw	Linux user-level ATA raw command utility	2989	ataraw-0.2.1.tar.gz *Mon, 05 Oct 2009 05:53:39 +0000*
tcpflow	TCP/IP session demultiplexer	1407	tcpflow-1.0.4.tar.gz *Thu, 24 Nov 2011 19:21:17 +0000*

http://simson.net/working/bulk_extractor.pdf

Creating Forensic Corpora

Digital forensics is at a turning point.
Yesterday's work was primarily *reverse engineering*.



Key technical challenges:

- Evidence preservation.
- File recovery (file system support); Undeleting files
- Encryption cracking.
- Keyword search.

Today's work is increasingly *scientific.*

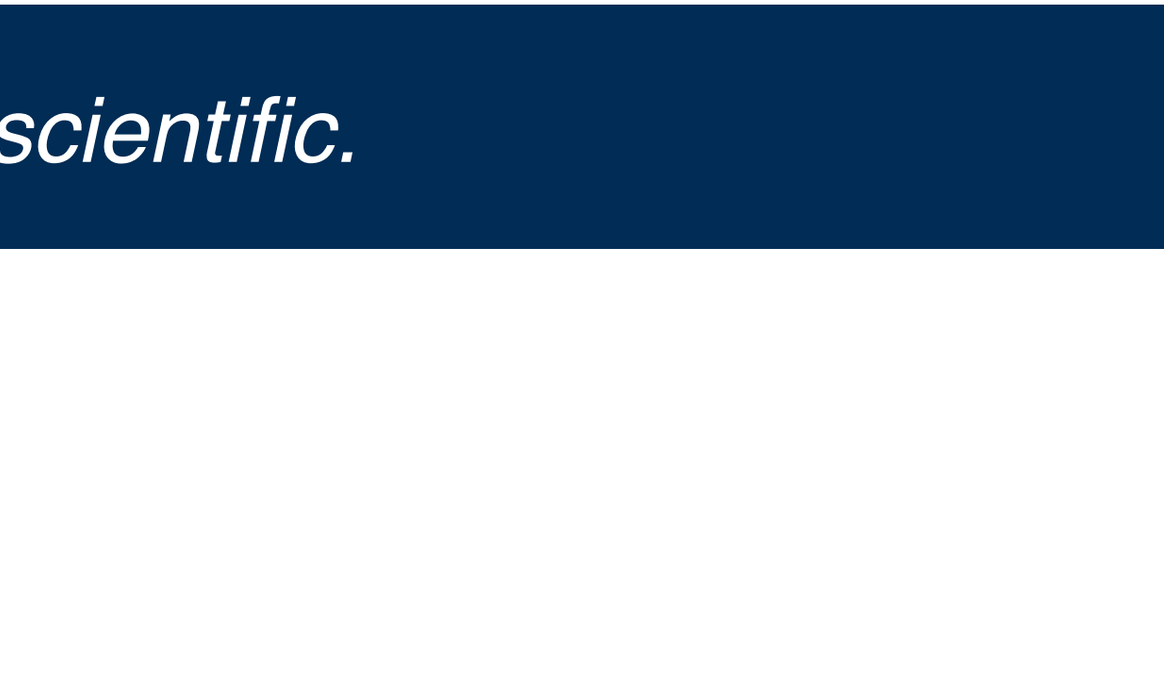
Evidence Reconstruction

- Files (fragment recovery carving)
- Timelines (visualization)

Clustering and data mining

Social network analysis

Sense-making



Science requires the *scientific process.*

Hallmarks of Science:

- Controlled and repeatable experiments.
- No privileged observers.

Why repeat some other scientist's experiment?

- Validate that an algorithm is properly implemented.
- Determine if ***your*** new algorithm is better than ***someone else's*** old one.
- (Scientific confirmation? — perhaps for venture capital firms.)

We can't do this today.

- People work with their own data

  —*Can't sure because of copyright & privacy issues.*

- People work with "evidence"

  —*Can't discuss due to legal sensitivities.*

We do science with “real data.”

The Real Data Corpus (30TB)

- Disks, camera cards, & cell phones purchased on the secondary market.
- Most contain data from previous users.
- Mostly acquire outside the US:
  - —*Canada, China, England, Germany, France, India, Israel, Japan, Pakistan, Palestine, etc.*
- Thousands of devices (HDs, CDs, DVDs, flash, etc.)

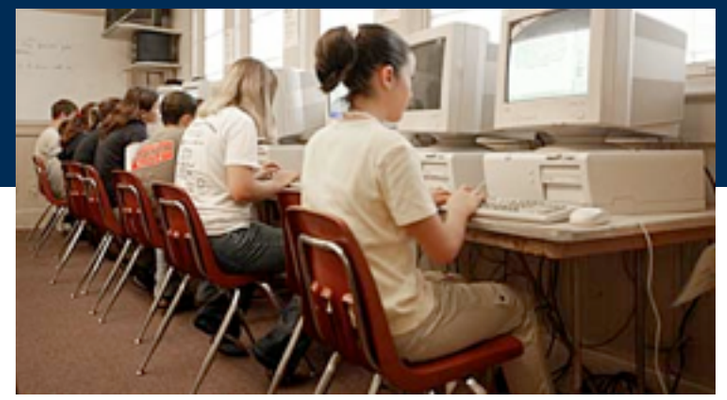
Mobile Phone Application Corpus

- Android Applications; Mobile Malware; etc.

The problems we encounter obtaining, curating and exploiting this data mirror those of national organizations

—*Garfinkel, Farrell, Roussev and Dinolt, Bringing Science to Digital Forensics with Standardized Forensic Corpora, DFRWS 2009*
http://digitalcorpora.org/

Digital Forensics education needs fake data!



To teach forensics, we need complex data!

- Disk images
- Memory images
- Network packets

Some teachers get used hard drives from eBay.

- Problem: you don't know what's on the disk.
  - —*Ground Truth.*
  - —*Potential for illegal Material — distributing porn to minors is illegal.*

Some teachers have students examine other student machines:

- Self-examination: students know what they will find
- Examining each other's machines: potential for inappropriate disclosure

We assemble data that can be freely redistributed.

Files from US Government Web Servers (500GB)

- ≈1 million heterogeneous files
  - —*Documents (Word, Excel, PDF, etc.); Images (JPEG, PNG, etc.)*
  - —*Database Files; HTML files; Log files; XML*
- Freely redistributable; Many different file types
- This database was surprising difficulty to collect, curate, and distribute:
  - —*Scale created data collection and management problems.*
  - —*Copyright, Privacy & Provenance issues.*

Advantage over flickr & youtube: persistence & copyright




<abstract>NOAA's National Geophysical Data Center (NGDC) is building high-resolution digital elevation models (DEMs) for select U.S. coastal regions. … </abstract>

<abstract>This data set contains data for birds caught with mistnets and with other means for sampling Avian Influenza (AI)….</abstract>

Our fake data can also be freely redistributed.

Test and Realistic Disk Images (1TB)

- Mostly Windows operating system.
- Some with complex scenarios to facilitate forensics education.

—*NSF DUE-0919593*

University harassment scenario

- Network forensics — browser fingerprinting, reverse NAT, target identification.
- 50MB of packets

Company data theft & child pornography scenario.

- Multi-drive correction.
- Hypothesis formation.
- Timeline reconstruction.

—*Disk images, Memory Dumps, Network Packets*

Download this and more from http://digitalcorpora.org/ !

In conclusion...

DFXML is a powerful tool for integrating forensic tools.

- And we are using it!

bulk_extractor is a powerful tool for getting information out of data

- Disk images
- memory dumps
- cell phones
- etc.

We have “fake data” and “real data” that we can share!

- Disk images
- Memory dumps
- cell phones
- packet captures

Questions?

Simson L. Garfinkel
email: slgarfin@nps.edu
phone: 202-649-0029
http://simson.net/