Automating Disk Forensic Processing with SleuthKit, XML and Python

Simson Garfinkel, Ph.D.
Associate Professor
Naval Postgraduate School
http://faculty.nps.edu/slgarfin/

May 20, 2009
SADFE 2009

NPS is the Navy's Research University.

Location: Monterey, CA

Campus Size: 627 acres

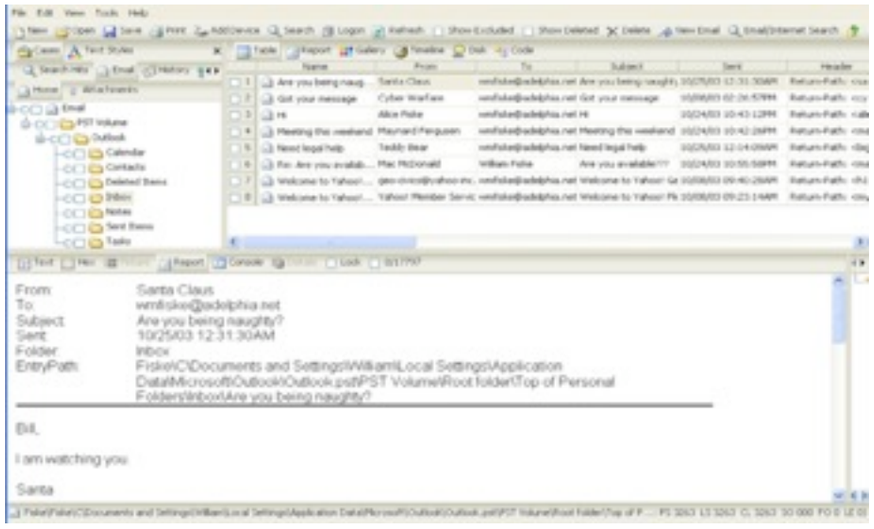
1500 Students:

- US Military (All 5 services)
- US Civilian (SFS & SMART)
- Foreign Military (30 countries)

4 Schools:

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

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



Encase:
- GUI Closed Source

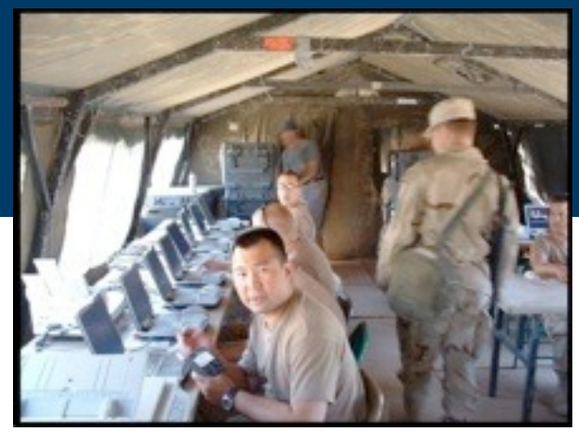
SleuthKit:
- Command-line Open Source

These tools are great for:

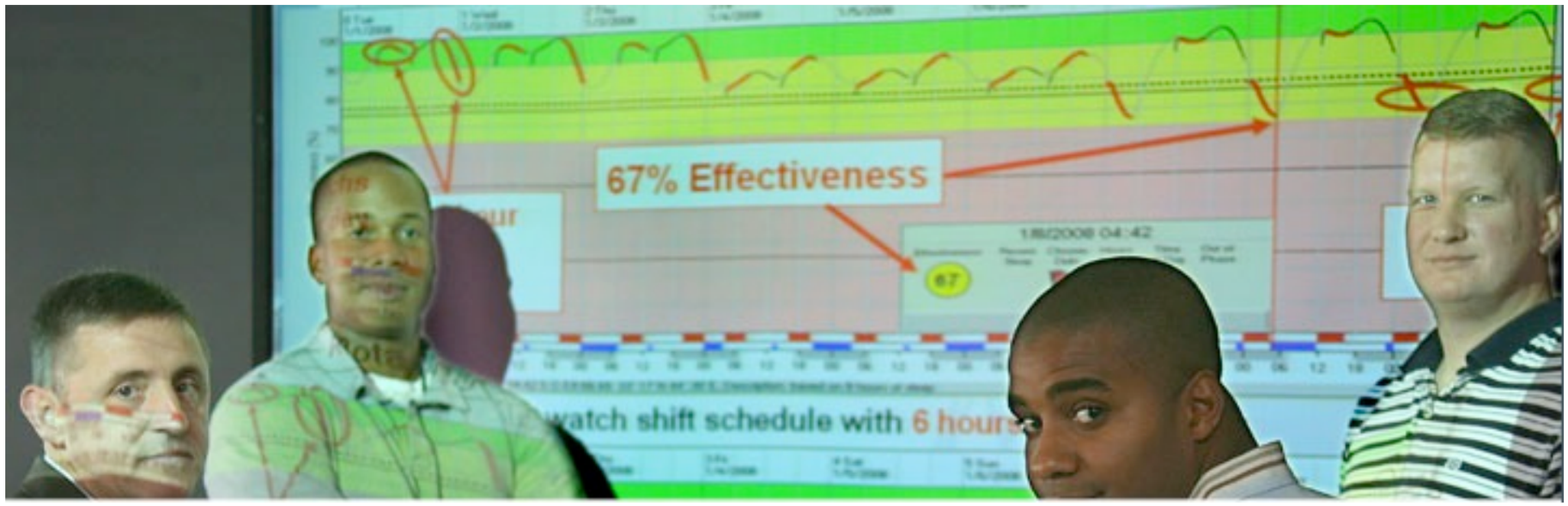
- FIle recovery
- Search

These tools were not created for research or automation.

Forensics *needs* research and automation.



Students (and researchers) need an easy-to-program environment for conducting forensic experiments.



It's *hard* to work with forensic data — All the details matter

- Many different file systems.
- Many different file types.

Good research requires working with large data sets.

- Even small "pilot studies" should be tested on multiple data sources.
- Otherwise, you aren't doing research on forensics — you are researching a particular object.

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++

But *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.

Faced with this, a standard approach is to leverage the database:

- Extract everything into an SQL database.
- Use multiple SELECT statements to generate reports.

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*.

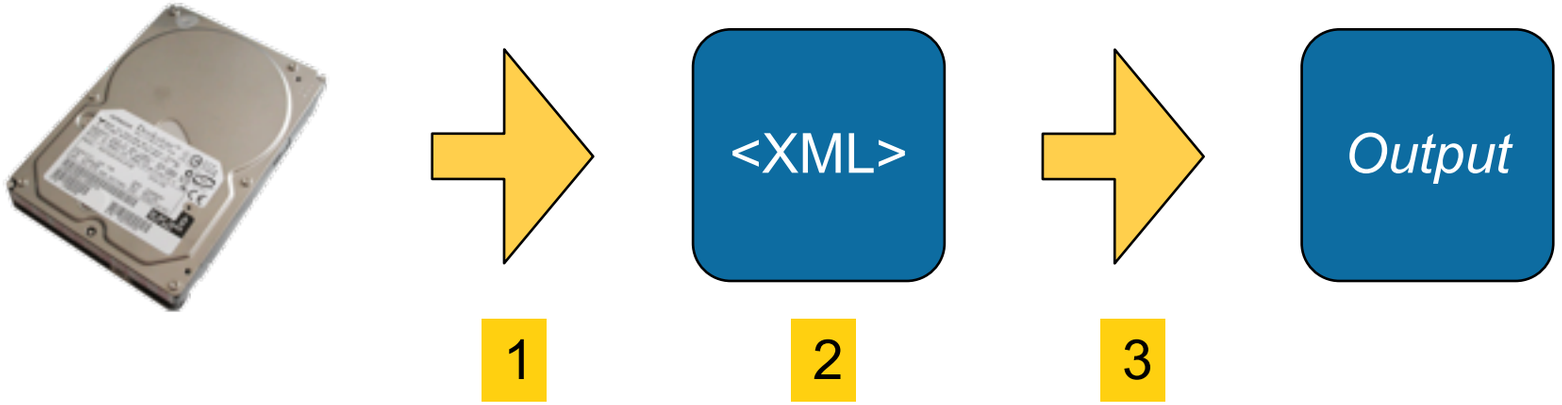
	Unsorted	Sorted
Files processed:	23,222	23,222
backwards seeks	12,700	4,817
Time to extract metadata:	19 seconds	19 seconds
Time to read files:	441 seconds	38 seconds
Total time:	460 seconds	57 seconds

disk image: nps-2009-domexusers1

This talk presents a new approach for automated forensic analysis and research

The approach breaks forensic processing into three key parts:

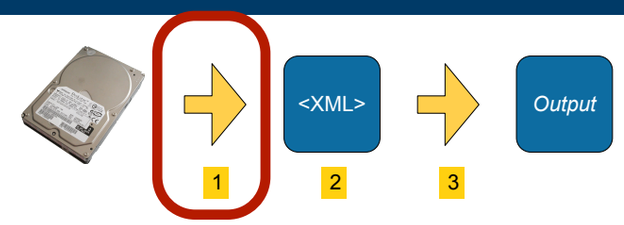
1. Extraction of forensic metadata.
2. Representation of the extracted metadata.
3. Processing.



You can start using this framework today.

You can easily expand it.

fiwalk extracts metadata from disk images.



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

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

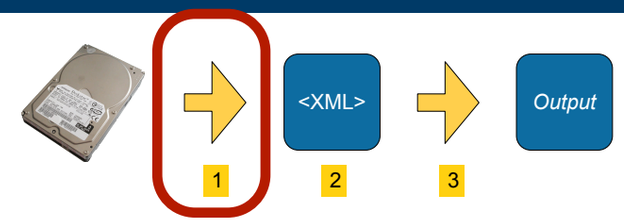
Features:

- 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
```

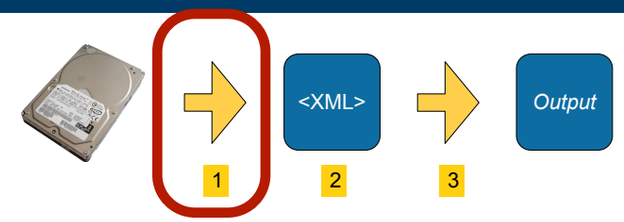
File System Metdata:

- -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
```

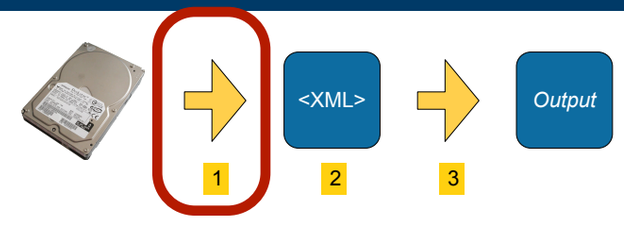
- *Plugins are run in a different process for safety.*
- *We have designed a native JVM interface which uses IPC and 1 process.*

Metadata extractors produce name:value pairs on STDOUT

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

Extracted metadata is automatically incorporated into output.

XML is ideally suited for representing forensic data.



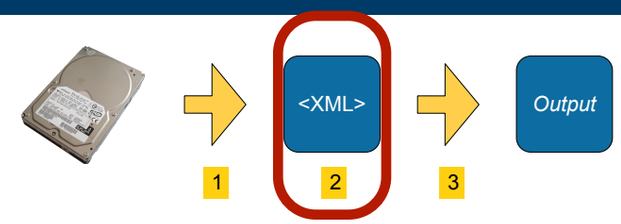
Forensic data is tree-structured.

- Case > Devices > Partitions > Directories > Files

- Files
  - *file system metadata*
  - *file meta data*
  - *file content*

- Container Files (ZIP, tar, CAB)
  - *We can exactly represent the container structure*
  - *PyFlag does this with "virtual files"*
  - *No easy way to do this with the current TSK/EnCase/FTK structure*
  - *(Note: Container files not currently implemented.)*

fiwalk produces three kinds of XML tags.



Per-Image tags

```
<fiwalk> – outer tag
<fiwalk_version>0.4</fiwalk_version>
<Start_time>Mon Oct 13 19:12:09 2008</Start_time>
<Imagefile>dosfs.dmg</Imagefile>
<volume startsector="512">
```

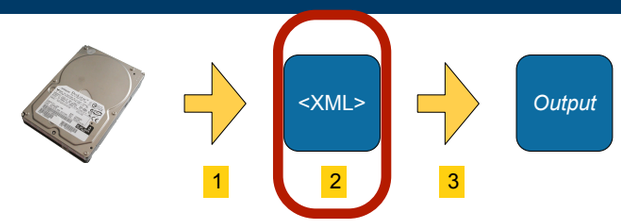
Per <volume> tags:

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

Per <fileobject> tags:

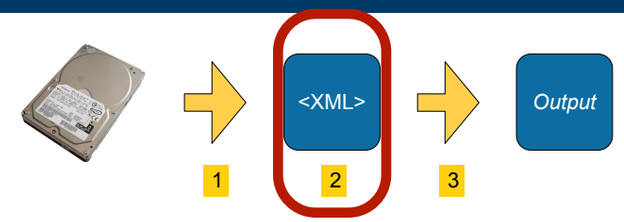
```
<filesize>4096</filesize>
<partition>1</partition>
<filename>linedash.gif</filename>
<libmagic>GIF image data, version 89a, 410 x 143</libmagic>
```

fiwalk XML example




```
<fileobject>
<filename>WINDOWS/system32/config/systemprofile/「开始」菜单/程序/附件/_rf55.tmp</
filename>
<filesize>1391</filesize>
<unalloc>1</unalloc>
<used>1</used>
<mtime>1150873922</mtime>
<ctime>1160927826</ctime>
<atime>1160884800</atime>
<fragments>0</fragments>
<md5>d41d8cd98f00b204e9800998ecf8427e</md5>
<sha1>da39a3ee5e6b4b0d3255bfef95601890afd80709</sha1>
<partition>1</partition>
<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>
</fileobject>
```

<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>
```

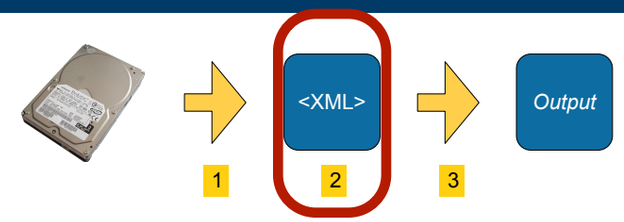
This file has two fragments:

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

Additional XML attributes may specify compression or encryption.

- Note: Currently <byte_runs> not provided for compressed or MFT-resident files.

XML incorporates the extracted metadata.



fiwalk metadata extractors produce name:value pairs:

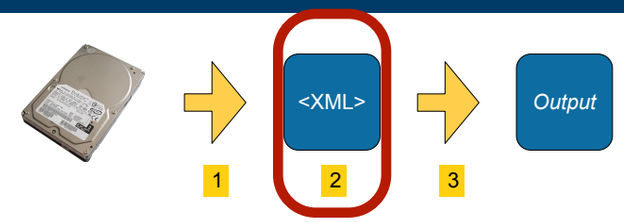
```
Manufacturer: SONY
Model: CYBERSHOT
Orientation: top - left
```

These are incorporated into XML:

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

— *Special characters are automatically escaped.*

Resulting XML files can be distributed with images.



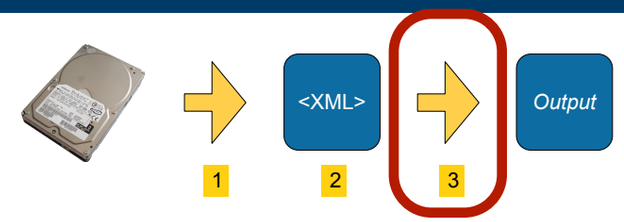
The XML file provides a key to the disk image:

```
$ 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: a Python module for automated forensics.



Key Features:

- Automatically runs fiwalk with correct options if given a disk image
- Reads XML file if present (faster than regenerating)
- Creates **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)`

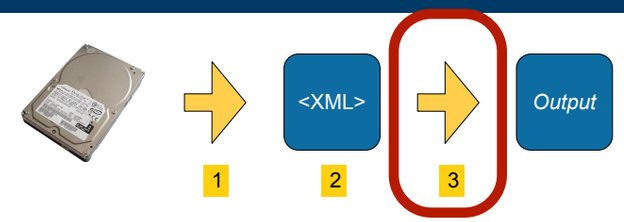
  — *Reasonably fast; returns a list of all file objects with XML in dictionary*

- DOM procedural interface

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

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

The SAX and DOM interfaces both return fileobjects!



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

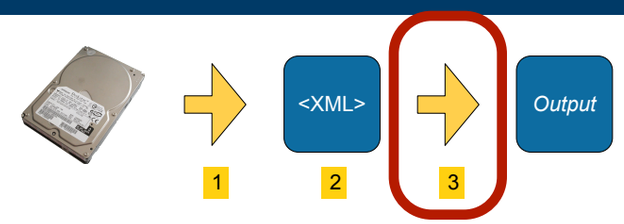
Objects belong to one of two subclasses:

`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.ctime(), fi.atime(), fi.crtime(), fi.mtime()*
- *fi.sha1(), fi.md5()*
- *fi.byteruns(), fi.fragments()*
- *fi.content()**

Example: calculate average file size on a disk



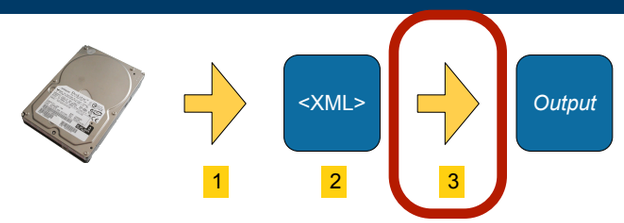
Using DOM interface:

```
import fiwalk

objs = fileobjects_using_sax(imagefile, xmlfile, flags)
print "average file size: ",sum([fi.filesize() for fi in objs]) / len(objs)
```

(For the Python-impaired:)

```
import fiwalk

objs = fileobjects_using_sax(imagefile, xmlfile, flags)
sum_of_sizes = 0
for fi in objs:
    sum_of_sizes += fi.filesize()
print "average file size: ",sum_of_sizes / len(objs)
```

Example: Find and print all the files 15 bytes in length.



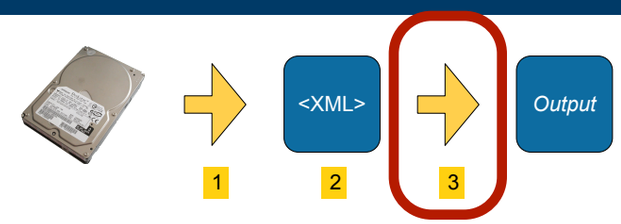
Using DOM interface:

```
import fiwalk

objs = fileobjects_using_sax(imagefile, xmlfile, flags)
for fi in filter(lambda x:x.filesize()==15, objs):
    print fi
```

(For the Python-impaired:)

```
import fiwalk

objs = fileobjects_using_sax(imagefile, xmlfile, flags)
for fi in objs:
    if fi.filesize()==15:
        print fi
```

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>
```

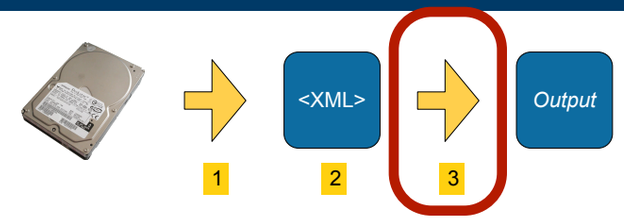
Becomes:

```
[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 calculating hash

We are building several interconnected applications with this framework.

imap.py

- reads a disk image or XML file and prints a “map” of a disk image.

igroundtruth.py

- reads multiple disk images (different generations of the same disk)
- uses earlier images as “maps” for later images.
- Outputs new XML file

iverify.py

- Reads an image file and XML file.
- Reports which files in the XML file are actually resident in the image.

iredact.py

- reads a disk image (or XML file) and a “redaction file”
- Produces new disk image.

The redaction language is flexible.

Language: {CONDITION} {ACTION}

Conditions:

- FILENAME *filename*
- FILEPAT *file*name*
- DIRNAME dirname/
- MD5 *d41d8cd98f00b204e9800998ecf8427e*
- SHA1 *da39a3ee5e6b4b0d3255bfef95601890afd80709*
- FILE CONTAINS *user@company.com*
- SECTOR CONTAINS *user@company.com*

Actions:

- FILL *0x44*
- ENCRYPT
- FUZZ *(changes instructions but not strings)*

We have also built a USB transfer kiosk.

The kiosk:

- Reads a USB drive using fiwalk & fiwalk.py
- Displays list of files in GUI
- Transfers selected files to “quarantine” *without mounting the disk image*.
- Virus scans
- Transfers scanned files to SMB server *without mounting the file server.*

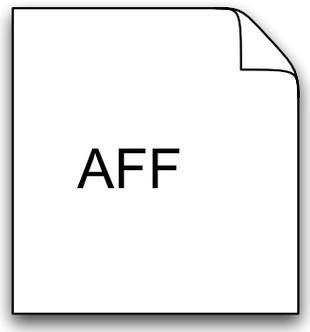
Key features:

- Functionality could not be implemented without forensic tools
- fiwalk & fiwalk.py allows forensics to be abstracted away
- Kiosk program is mostly GUI, not forensics
  - *filelist.py* — *110 lines*
  - *kiosk.py* — *368 lines*
  - *loginpanel.py* — *70 lines*
  - *smb.py* — *90 lines*
  - *watcher.py* — *152 lines*

Publishing XML for disk images enables our remote exploitation methodology...

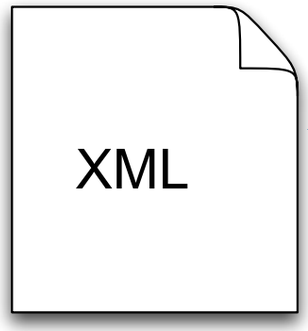
Extract metadata in Boston.
Search from Monterey.
Just download what you need.

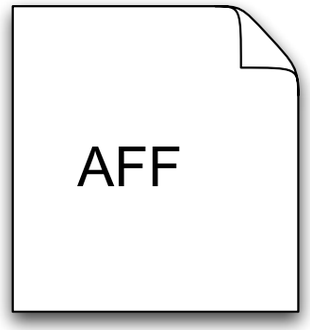
Publishing XML for disk images enables our remote exploitation methodology...



Extract metadata in Boston.
Search from Monterey.
Just download what you need.

Publishing XML for disk images enables our remote exploitation methodology...





Extract metadata in Boston.
Search from Monterey.
Just download what you need.

Publishing XML for disk images enables our remote exploitation methodology...

http

XML

VE RI TAS

AFF

Extract metadata in Boston.
Search from Monterey.
Just download what you need.

Publishing XML for disk images enables our remote exploitation methodology...

http

XML

VE RI TAS

xmlrpc

AFF

Extract metadata in Boston.
Search from Monterey.
Just download what you need.

Publishing XML for disk images enables our remote exploitation methodology...

http

XML

VE RI TAS

xmlrpc

AFF

Extract metadata in Boston.
Search from Monterey.
Just download what you need.

In summary, XML and Python can make forensic research and application development easier.

fiwalk — Batch procesisng of disk images.

XML — A widely understood data model.

python tools — Easy to create new forensic applications.

Available from http://www.afflib.org/

Acknowledgments:

- NPS:
  - *Jessy Cowan-Sharp*
  - *George Dinolt*
  - *Beth Rosenberg*
- NIST
- Anonymous Reviewers

Questions?