Scripting with the Sleuthkit

Jon Stewart
Big Picture Guy, Lightbox Technologies, Inc
http://www.lightboxtechnologies.com
http://codeslack.blogspot.com
Twitter: @codeslack

"Most developers are *morons*,
and the rest are *a**holes*.

A**holes read specs with a fine-toothed comb, looking for loopholes, oversights, or simple typos. Then they write code that is meticulously spec-compliant, but useless.

Morons don't read specs until someone yells at them. They take a few examples they find "in the wild" and write code that seems to work.

Morons work from examples, ship code, and get yelled at. But when they get around to reading the spec, [some] magically turn into advocates and write up tutorials on what they learned from their mistakes. These people are called *experts*."

— Mark Pilgrim, *Why Specs Matter*, 2004

author, *Dive Into Python*, *Dive into HTML5*

I was an expert

- 6.5 years at Guidance Software
- Dual-hatted: developer & consultant
- If it involves EnScript, it's probably my fault.
  - Sorry.
- Also did *a lot* of eDiscovery casework.
- But this is not CEIC...

1959–1962

Super RECORD
LIVING STEREO
VICTOR
SHP-5074
Stereo Orthophonic High Fidelity Recording
橋〜ソニー・ロリンズ
The Bridge
Sonny Rollins

Confession: I am a moron

- Left GSI on December 22, 2009
- Wanted to write forensics code in Python, JavaScript, C++, Ruby, Go, Scala, Haskell
  - maybe even some Java
  - but please, no Perl
- Have been learning the Sleuthkit library
- and all kinds of other open source things

This presentation is for the morons.

(Not the a**holes.)

libtsk

- libtsk is a C/C++ library for getting Sleuthkitty
- The docs are really good. (Thanks, Brian!)
  - http://www.sleuthkit.org/sleuthkit/docs/api-docs/index.html
- If you know C++, inherit from TskAuto
  - Automates getting access to every file.
- *Waaaaaaaaaay* better to use libtsk than command-line utilities for batch processing
  - caching, process launching, parsing, caching...

TskAuto: declaration

```
#include <tsk3/auto/tsk_auto.h>

class FsWalker: public TskAuto {
public:
  FsWalker(ostream& out);

  virtual TSK_FILTER_ENUM filterFs(TSK_FS_INFO *fs_info);

  virtual TSK_RETVAL_ENUM processFile(TSK_FS_FILE *fs_file, const char *path);

  unsigned int NumFiles;

private:
  ostream& Out;
  string  FsInfo,
          Null,
          CurDir;

  TSK_FS_INFO* Fs;

  unsigned int CurDirIndex;
};
```

TskAuto: kicking off

```
FsWalker walker(cout);
if (0 == walker.openImage(numSegments,
                          segmentsStringArray,
                          TSK_IMG_TYPE_DETECT,
                          0)) // auto-detect blocksize
{
  walker.findFilesInImg();
}
```

TskAuto: callbacks

```
TSK_FILTER_ENUM FsWalker::filterFs(TSK_FS_INFO *fs)
{
  stringstream buf;
  buf << j(string("fs")) << ":{"
      << j("byteOffset", fs->offset, true)
      << j("blockSize", fs->block_size)
      << j("fsID", bytesAsString(fs->fs_id,
               &fs->fs_id[fs->fs_id_used]))
      << "}";
  FsInfo = buf.str();
  Fs = fs;
  return TSK_FILTER_CONT;
}
```

TskAuto: callbacks

```
TSK_RETVAL_ENUM FsWalker::processFile(TSK_FS_FILE* file, const char* path) {
 ++NumFiles;
 ++CurDirIndex;
 if (0 != CurDir.compare(path)) {
  CurDirIndex = 0;
  CurDir.assign(path);
 }
 // cerr << "beginning callback" << endl;
 try {
  if (file) {
   Out << "{" << FsInfo;
   if (path) {
    Out << j("path", string(path));
   }
   if (file->meta) {
    // need to come back for name2 and attrlist
    TSK_FS_META* i = file->meta;
    Out << ", \"meta\":{"
      << j("addr", i->addr, true)
      << j("atime", i->atime)
      << j("content_len", i->content_len)
      << j("crtime", i->crtime)
      << j("ctime", i->ctime)
      << j("flags", i->flags)
      << j("gid", i->gid);
    if (i->link) {
     Out << j("link", string(i->link));
    }
```

TskAuto is C++, not scripting

- Yep
- If you know some C++, TskAuto is easy.
- If you don't know C++, this gives you a taste.

Strategy: Get Data Out

APIs are good for one language... but not for all

fiwalk

- Simson Garfinkel
- Outputs filesystem data as xml, other formats
- Python interface!
- Depends on TSK, Afflib, libewf
- Download
  http://afflib.org/downloads/fiwalk.tar.gz

```
tar xfvz fiwalk.tar.gz
cd fiwalk-0.5.7
./configure && make
sudo make install
```

fiwalk –x <evidence file>

```
<fileobject>
   <filename>CEIC 2009 Presentation Files/Acquiring Data Off a Dead Drive-Wiechman-5-20-2009.pdf</filename>
   <partition>1</partition>
   <id>4</id>
   <name_type>r</name_type>
   <filesize>5640434</filesize>
   <alloc>1</alloc>
   <used>1</used>
   <inode>522</inode>
   <meta_type>1</meta_type>
   <mode>511</mode>
   <nlink>1</nlink>
   <uid>0</uid>
   <gid>0</gid>
   <mtime>2009-04-29T14:00:22Z</mtime>
   <atime>2009-10-24T04:00:00Z</atime>
   <crtime>2009-04-30T20:34:58Z</crtime>
   <byte_runs>
    <byte_run file_offset='0' fs_offset='303104' img_offset='303104' len='5640434'/>
   </byte_runs>
   <hashdigest type='md5'>ffbe96f14ba547a83e29a51fa373ce29</hashdigest>
   <hashdigest type='sha1'>1879ae8d0851f3114be9b99462af10872d6280b1</hashdigest>
  </fileobject>
```

fsrip

- moron project created by me
- **Not ready for primetime; don't use this**
- spits out line-oriented json instead of xml
- http://github.com/jonstewart/fsrip.git

```
Install scons
git clone git://github.com/jonstewart/fsrip.git fsrip
cd fsrip
scons
export LD_LIBRARY_PATH=../fsrip/deps/lib
```

fsrip dumpfs <evidence file>

```
{
  "fs":{
    "byteOffset":0,
    "blockSize":512,
    "fsID":"c51bd431"
  },
  "path":"CEIC 2009 Presentation Files/",
  "meta":{
    "addr":522,
    "atime":1256356800,
    "content_len":8,
    "crtime":1241123698,
    "ctime":0,
    "flags":5,
    "gid":0,
    "mode":511,
    "mtime":1241013622,
    "nlink":1,
    "seq":0,
    "size":5640434,
    "type":1,
    "uid":0
  },
  "name":{
    "flags":1,
    "meta_addr":522,
    "meta_seq":0,
    "name":"Acquiring Data Off a Dead Drive-Wiechman-5-20-2009.pdf",
    "shrt_name":"ACQUIR~1.PDF",
    "type":5,
    "dirIndex":2
  },
  "attrs":[
    {
      "flags":3,
      "id":0,
      "name":"",
      "size":5640434,
      "type":1,
      "rd_buf_size":0,
      "nrd_allocsize":5668864,
      "nrd_compsize":0,
      "nrd_initsize":5640434,
      "nrd_skiplen":0,
      "nrd_runs":[
        {
          "addr":592,
```

xml

- xml validates structure of file
- xml can handle just about any encoding
- hierarchical/tree structure very familiar
- lots of good tool/library support
- pretty good web browser support

J'SON
life on life

line-oriented json

- a teensy bit terser than xml
- works with other CLI utilities

  fsrip dumpfs badguy.dd | wc –l
- great language support
- great web browser support
- flat structure allows for simple parallelism

```
#!/usr/bin/env python

import sys
import json

line = ''
for line in sys.stdin:
  line = line.strip()
  obj = json.loads(line)
  if "name" in obj and "path" in obj:
    print obj["path"] + obj["name"]["name"]
```

{"fs":{"byteOffset":0,"blockSize":512,"fsID":"c51bd431"},"path":"CEIC 2009 Presentation Files/", "meta":{"addr":522,"atime":1256356800,"content_len":8,"crtime":1241123698,"ctime":0,"flags":5,"gid":0,"mode":511,"mtime":1241013622,"nlink":1,"seq":0,"size":5640434,"type":1,"uid":0}, "name":{"flags":1,"meta_addr":522,"meta_seq":0,"name":"Acquiring Data Off a Dead Drive-Wiechman-5-20-2009.pdf","shrt_name":"ACQUIR~1.PDF","type":5,"dirIndex":2}, "attrs":[{"flags":3,"id":0,"name":"","size":5640434,"type":1,"rd_buf_size":0,"nrd_allocsize":5668864,"nrd_compsize":0,"nrd_initsize":5640434,"nrd_skiplen":0, "nrd_runs":[{"addr":592,"flags":0,"len":11072,"offset":0}]}]}

Ordering and Iteration

- Recursive descent of filesystem is easy, but...
  - Subvert the dominant paradigm!
- If processing file data, ordering files by extents leads to less disk seeking (c.f. Garfinkel)
- Flat structure lets you chop up data for xargs multi-core power!

```
fsrip dumpfs badguy.dd | xargs –P 8 ./myscript.py
```

- Iteration is where you drop in multithreading

Writing Parsers

- Try to separate I/O from parsing
  - i.e., Foo reads data, Bar parses that data
- Make your parsers simple, just output a nicer form of the data. Don't analyze/filter it.
- Provide a provenance, i.e., bytes on disk
  - tell me where the data came from
- Write unit tests
- Too many duplicate parsers!

Reading and Writing

- icat <evidence file> <meta address>
- That is all ye know on earth, and all ye need to know.
- One could design a daemon around icat, to cut down on process/evidence overhead
  - Ordering disk access is probably a bigger win
- Write to a different device than from where you're reading!

RAM and algorithms

- Don't be afraid to use a lot of it
  - e.g., All hash values in NSRL will fit in RAM
- Try to use a *constant* amount of RAM
  - Won't crap out when you throw a RAID at it
- O(n log n) is a lot like O(n)...

  ...compared to $O(n^2)$

Databases are a Gateway Drug

- tsk_loaddb <evidence file>
- Generates a sqlite database with metadata
- Query to your heart's content
- http://dfsforensics.blogspot.com/2011/01/exploring-sleuthkits-new-tskloaddb.html

Thanks!