Plaso
Reinventing the super timeline.

DFIR Summit 2013
Who Am I

- Incident responder and a forensic analyst.
  - In other words a practitioner in the DFIR field.
- Software developer.
- Work for a small company called Google.
- Been endorsed on LinkedIn for:
  - balloon artist
  - certified arborist
  - party favors
  - tires
  - and many other things...
DFIR in a nutshell.

What is the goal?

How does timeline analysis fit in?
Enter the Super Timeline

Correlation
Context
Confidence
Temporal proximity.
Sorted based on time.
Good Ol' log2timeline

● **Highlights**
  ○ Written in Perl
  ○ Modules independent from other parts.
  ○ Single-threaded.
  ○ Engine called Log2Timeline.

● **Issues**
  ○ Timestamp stored with second precision.
  ○ Worked on files (not images).
  ○ Little information shared between parsers.
  ○ Events stored as a perl hash with limited structure.
  ○ Output usually in a text format.
  ○ Adding new features usually required heavy rewrite.
  ○ Harder to do filtering and post processing.
Plaso Langar Að Safna Öllu

- Framework named plaso.
- Log2timeline still main front-end.

Multi processed.
Structured events
Metadata in storage
Granular filtering
Parses image files
VSS parsing
Targeted collection
Tagging.
Goals

Make it easy to create and analyse super timelines
Front Ends

I DON'T ALWAYS CREATE NEW TOOLS

BUT WHEN I DO, I NAME THEM ALL USING ICELANDIC ACRONYMS

log2timeline
   Extract timelines.
psort
   Post processing.
plasm
   Tagging (for now)
pinfo
   Display storage metadata
pshell
   iPython shell (advanced)

Additional front-ends
preg - PoC for registry parsing.
pprof - profiling runtime, for devs.
Quick Overview

Preprocess

Collects information from the image/files/mount point that can be used during processing.

Collector

Process queue

Goes over the image/mount point to find all files that need to be processed.

Worker 1

storage queue

Process each extracted event. But it in a queue, sort the queue before flushing it to disk.

Worker N

storage file on disk

Process each file, file opened, classified and other files potentially extracted from it and also processed, parsed and events extracted.
How to Use the Tool?

```
log2timeline.py [OPTIONS] output_file input_file
```

```
log2timeline.py -o 63 /cases/12345/storage.dump /cases/12345/evil.dd
```

- **Parameters:**
  - `-o 63`: This is an image with sector offset 63.
  - `...storage.dump`: Location of storage file.
  - `...evil.dd` is the input (the image in this case).
  - `[--vss]`: Optional to include VSS parsing
What to collect?

Different approaches

Do actual events of interest get drowned?

What if missed something?
Targeted Collection

● "Sniper forensics".

● We have a set of questions.
  ○ And we know what data we need?

● Why include everything
  ○ ...when you obviously know what you are after?
Current Approach

- Collect browser history

(not a complete list)

```plaintext
/(Users|Documents And Settings)/.+/.+/AppData/Local/Google/Chrome/.+/History
/(Users|Documents And Settings)/.+/.+/Local Settings/Application Data/Google/Chrome/.+/History
/Users/./.+/.+/AppData/Local/Microsoft/Windows/History/History.IE5/index.dat
/Users/./.+/.+/AppData/Local/Microsoft/Windows/History/History.IE5/MSHist.+/.+/index.dat
/Users/./.+/.+/AppData/Local/Microsoft/Windows/History/Low/History.IE5/index.dat
/Users/./.+/.+/AppData/Local/Microsoft/Windows/History/Low/History.IE5/MSHist.+/.+/index.dat
/Users/./.+/.+/AppData/Local/Microsoft/Windows/Temporary Internet Files/Content.IE5/index.dat
/Users/./.+/.+/AppData/Local/Microsoft/Windows/Temporary Internet Files/Low/Content.IE5/index.dat
/Users/./.+/.+/AppData/Roaming/Microsoft/Windows/Cookies/index.dat
/Users/./.+/.+/AppData/Roaming/Microsoft/Windows/Cookies/Local index.dat
/Documents And Settings/.+/Local Settings/History/History.IE5/index.dat
/Documents And Settings/.+/Local Settings/Temporary Internet Files/Content.IE5/index.dat
/Documents And Settings/.+/Cookies/index.dat
/(Users|Documents And Settings)/.+/.+/AppData/Roaming/Mozilla/Firefox/Profiles/.+/places.sqlite
/(Users|Documents And Settings)/.+/.+/Local Settings/Application Data/Mozilla/Firefox/Profiles/.+/places.sqlite
```
Run the Tool

log2timeline.py -i -f browser_filter.txt history.dump /mnt/e01/ewf1

p.s. instead of parsing a disk image we could just as easily have used F-Response to connect to an image over the network, or for that matter GRR.
Then What?
Are we done now?
Is collection enough?

...well there is always the analysis
ANALYZING TIMELINES IS HARD

LET'S GO SHOPPING.
Let's talk Analysis

Let's review our initial goals and questions

Can we construct better filters?

What about tagging?

Then we can assess the data we already have
1. Review Initial Goal
Review Initial Goal/Questions.

There are quite a few events that occur on a typical filesystem, and the ones you are interested in can easily hide amongst the millions you don't care about.
Hypothetical Example

● Imagine a very simple example.
  ○ And a boring one too...
● Internal investigation.
  ○ The user is suspected of studying up on computer forensics online (which is a terrible offense).
● Goal is:
  ○ Has the user been studying computer forensics?
Overall Goal
What to Collect

- This super exciting investigation's goal reduces to .... collecting browser history.
- Perform a targeted collection of browser history (as detailed in the slides before).
2. Tag Events
Tagging

- PLASM (Plaso Langar Ad Safna Minna)
- Tags events based on criteria.
- Simple definition file:

```
TAG NAME
  CONDITION (REGULAR FILTER)
ANOTHER TAG
  CONDITION 1
  CONDITION 2
```

Example

```
Application Execution
  data_type is 'windows:prefetch:prefetch'
```
Tagging Continued

plasm.py -t tag.txt some_case.dump
Applying tags...
DONE (applied 298 tags)

pinfo.py some_case.dump
...

Counter information:
  Counter: Total = 298
  Counter: Application Execution = 142
  Counter: Document Opened = 121
  Counter: File Downloaded = 40
  Counter: Document Printed = 15
3. Assess Data.
Assess Data

pinfo.py /cases/12345/boring_case.dump

... Time of processing: 2013-07-06T20:18:08 ...

    parsers = ['Symantec', 'PfileStatParser', 'WinRegistryParser', 'ChromeHistoryParser',
          'GoogleDriveParser', 'MsiecfParser', 'WinEvtParser', 'FirefoxHistoryParser', 'WinPrefetchParser',
          'WinLnkParser']

... Counter information:

    Counter: total = 24157
    Counter: MsiecfParser = 22067
    Counter: ChromeHistoryParser = 1952
    Counter: PfileStatParser = 136
2. Construct Filter
Filters... How Do They Work?

- Filters are modular.
  - Currently mostly different wrappers of the same filter.
- Available filters:
  - Event filter.
  - Filter list.
  - Dynamic filter.
- Event filter:
  "date > '2012-01-01 15:12:02' and parser contains 'prefetch' and (executable contains 'cmd' or executable contains 'evil')"
- Dynamic filter (wrapping event filter):
  "SELECT datetime, executable WHERE executable contains 'evil' "
Filters and Tags

- You can even filter based on tags.

```
psort.py -q allt_vss.dump "SELECT date,time,timestamp_desc,message
WHERE tag CONTAINS 'Application Execution' AND date > '2009-01-01'
LIMIT 1"
```

Checking to see if you got a legit copy of Windows

2011-08-14,08:00:48,Content Modification Time,[7045 / 0x00001b85] Record Number: 2185 Event Level: 4 Source Name: Service Control Manager Computer Name: WKS-WIN764BITB.shieldbase.local Strings: [u'Windows Activation Technologies Service' u'\%SystemRoot\%\system32\Wat\WatAdminSvc.exe' u'user mode service' u'demand start' u'LocalSystem']

Service starting - application executed
Construct Filters

1. Define the goal.
2. What sources provide the necessary information?
3. View sample source.
4. Determine which attributes need to be displayed.
5. Construct a test filter.
6. Wrap this all up in a nice little one-liner.
Let's Construct a Filter

● Let's get back to our lovely example.
● Find out if our employee researched forensics...
  ○ And where do you research things?
  ○ Let's examine search engine history.

● Sources:
  ○ Web history.
  ○ URL has to be from a known search engine.
  ○ Extract the search query itself.
Filter Construction, continued

● Let's skip some steps and assume we know all.
  ○ Browser history all has the source "WEBHIST"
  ○ URL's are stored in an attribute called "url"
  ○ Dynamic filter can display only that attribute.

● One example filter:

SELECT url WHERE source is 'WEBHIST' and url contains 'google.com' and url not contains 'mail.google.com'
Let's turn that into a one-liner

- Let's assume we all know our one-liners, and here it comes:

```bash
psort.py -q boring_case.dump "SELECT url WHERE source is 'WEBHIST' and url contains 'google.com' and url not contains 'mail.google.com'" | awk -F 'q=' '/search/ {if ($2 != "") { split($2,s,"&"); print s[1]}}' | xargs -0 string_scrub | sed -e 's/+/*g' | sort | uniq -c | sort -nr

small script to replace %2A, etc hex into characters.
```
...Or Create a Script

```python
plaso_extract_search_history.py boring_case.dump

== ENGINE: GoogleSearch ==
5  super timelines beginner
3  log2timeline
1  yout

== ENGINE: Gmail ==

== ENGINE: YouTube ==
2  log2timeline
1  log2timeline dummies
1  super timeline
1  super timeline forensics

== ENGINE: BingSearch ==
10  computer forensics for dummies
5  how to avoid computer forensics detection
3  super timelines for dummies
1  super timelines howot
1  super timelines howto
```
Post processing summary

- Go back to those questions of yours...
  - Construct filters that fit that criteria.
- Build up a set of filter rules.
  - For the initial "catch known stuff" analysis.
- Have few one-liners ready to go.
  - Exceptionally few examples on plaso site.
  - More to come...
- Learn to love the command line
  - Remember sed/awk/grep/... are your friends.
  - Or try out some of the GUI solutions like 4n6time.
Small Recap

● Plaso is a collection of libraries to:
  ○ Parse image files.
  ○ Provide an easy read-only access to files.
  ○ Collector to find and open files.
  ○ Parsers that know file structures.

● Extracting timelines is not the only purpose
  ○ Although main driving factor behind development
What Can I Do?

- Code contributions.
  - You'll love the code review process.
- Test the code and provide feedback.
- Throw some suggestion our way.
  - New parsers.
  - Modify the output.
- Develop documentation.
  - Usage, analysis or development tips.
Questions?

When Chuck Norris analyses timelines he uses psort.py storage.dump without filters