Plaso - reinventing the super timeline.

Kristinn Gudjonsson
Who Am I?

- Incident responder and a forensic analyst
- Software developer
- Work every now and then for Google
- Been endorsed on LinkedIn for:
  - balloon artist
  - certified arborist
  - party favors
  - tires
  - and many other things
Why Rewrite log2timeline?

• Few issues came up that required a rewrite
  • Does not scale easily
  • Single-threaded
  • Only second precision
  • Output not structured
  • Hard to add new features

• Why rewrite in Python?
  • Easier to get external contributors
  • Easier to integrate with other projects (TSK, Volatility™, GRR)
  • Most new forensics tools/libraries/scripts are released in Python
  • Google doesn’t like Perl
  • Easier to maintain than Perl code
Goal With New Version

- Make it easier to create a timeline
- Automate parts of the analysis
  - Tagging/categorization
  - Statistical analysis and reports
  - Clustering/Grouping together events that belong to the same user action
- Create a set of useful libraries for others to use
  - For one-off scripts using parts of the feature set
  - To integrate the functionality into other tools
- Make the tool scalable
  - Both using cores on machine and across machines
- Not just focus on timelines
  - Current examples: image_export and preg
OSDF

What Does Plaso Provide?

- Scalability (more to come)
- Structured events
- Ability to trace where an event was extracted from
- Metadata stored
- Granular filtering
- Directly parse disk images (TSK)
  - Moving to offload that to a new project, pyvfs
- VSS Parsing
- Targeted and kitchen-sink collection
- Tagging of events
How Open Source is it?

- All code stored on Google Code
  - https://code.google.com/p/plaso
- All code review done in public
  - https://codereview.appspot.com
- Most if not all design documents open to dev group
  - https://groups.google.com/forum/#!forum/log2timeline-dev
- Documentation actively updated
  - http://plaso.kiddaland.net
  - http://blog.kiddaland.net
- Roadmap open to all
  - http://goo.gl/7x4pli
New Release - 1.0.2alpha

- Halloween brings with it riding witches and other treats
  - Most notably a new plaso release
- Introducing version 1.0.2alpha
  - AKA the spooky release
OSDF Highlights

- Over 16 new parsers introduced
  - Sometimes a thin line between a plugin and a parser
- Three new output modules
- Two new front-ends
- Several enhancements
- Ready for replacing the 0.X branch
Front-Ends Included

log2timeline
- Extract timelines.

psort
- Post processing.

plasm
- Tagging (for now)

pinfo
- Display storage metadata

pshell
- iPython shell (advanced)

preg
- Registry parsing

pprof
- Profiling runtime, for devs.

image_export
- Exporting files out of an image
Other tools

- As you may have noticed all of the UIs are CLI
  - Nothing else in the world?
- Due to easy integration into other tools focus is on CLI and backend
  - Others are welcome to make their own UI’s
- Example UIs
  - 4n6time
  - GRR (coming soon)
How To Run the Tool?

log2timeline.py [OPTIONS] output_file input_file

log2timeline.py -o 63 [--vss] /cases/12345/storage.dump /cases/12345/evil.dd

- **Parameters**
  - `-o 63`: This is a disk image and the partition starts at sector offset 63
  - Could also use `--partition 2`
  - `[--vss]`: Optional, include information from VSS
  - `storage.dump`: This is the path of the storage file
  - `evil.dd`: This is the input, the disk image
What to Collect?

The Kitchen Sink

Do actual events of interest get drowned?
What to Collect?

The Targeted Approach

What if missed something?
What to Collect?

Build this timeline, brick by brick

The “let’s start small” Approach
Targeted Collection

- Collect browser history
  - Sample target file, does not include all sources

/$(Users|Documents And Settings)/.+/$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/Low/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
Targeted Collection

- Collect few registry files
  - Again not meant as a complete list, just an example

/\(\text{Users|Documents And Settings}/.+/\text{NTUSER.DAT\)}
/\{\text{sysregistry}\}/\text{SOFTWARE}
/\{\text{sysregistry}\}/\text{SYSTEM}
/\{\text{sysregistry}\}/\text{SAM}
/\{\text{sysregistry}\}/\text{SECURITY}
How To Use the Tool

```
log2timeline.py -o 63 -f filter_file.txt browser_storage.dump /mnt/e01/ewf1
```

- Same parameters as before, except using “-f”

```
-f FILE_FILTER, --file_filter FILE_FILTER
List of files to include for targeted collection of
files to parse, one line per file path, setup is
/path/file - where each element can contain either a
variable set in the preprocessing stage or a regular
expression
How To Use the Tool

log2timeline.py --partition 2 -f /cases/filters/browser.txt /cases/12345/plaso.dump image.dd

- Do some review, notice I might want registry information

log2timeline.py --partition 2 --use_old_preprocess -f /cases/filters/registry.txt /cases/12345/plaso.dump image.dd

- Review again, reiterate until done
  - The --use_old_preprocess indicates you don’t want to regenerate preprocessing data but rely on previous find
Can we start looking at the timeline now?
The tool stores all the data in a compressed container. Need to use “psort” to convert the output.

Available choices (as of now):
- L2tCSV - the default output of 0.X branch of log2timeline
- MySQL4n6 - PoC MySQL connection for 4n6time
- Dynamic CSV (default output)
- Rawpy - “raw” output of the python event
- Raw - a string representation of the raw protobuf
- SQL4n6 - a SQLite database used by 4n6time
- Pstorage - save back into a plaso storage
- Have different requirements?
  - Write your own output module
  - Ask the developers to add one for you

- Most common parameters:
  - -o FORMAT: choose the output module
  - -w OUTPUTFILE: the path to the output file
  - PLASOFILE: the path to the storage file
  - FILTER: filter the output data set

- Other parameters:
  - --slice/--slicer: time slices
  - -z TIMEZONE: present timestamps in a different timezone than UTC
  - -q: Silence a quick runtime statistics in the end
Basic Psort

`psort.py mystorage.dump`

- Dumps out all the content in CSV to STDOUT

`psort.py -w l2t.csv -o l2tcsv mystorage.dump`

- Dump all the content of the storage into a L2tCSV file
ANALYZING TIMELINES IS HARD

LET'S GO SHOPPING.
pinfo - Where Metadata Counts

- pinfo presents metadata stored in a plaso storage file

**pinfo.py [-v] storage.dump**

- Prints out information such as
  - When and how the tool was run
  - What parameters were turned on
  - What parsers were loaded
  - Total count of events inside storage
  - Count of events extracted from each parser

- Verbose information includes
  - Information from pre-processing
  - Counters from each store
Filtering

- Filters in plaso are modular
  - Current implementations are mostly wrapper around the same filter
- Available filters:
  - Event filter
  - Filter list
  - Dynamic filter (affects output)

- Example event filter
  "date > '2012-01-01 15:12:02' and parser contains 'prefetch' and (executable contains 'cmd' or executable contains 'evil')"

- Example dynamic filter
  "SELECT datetime, executable WHERE executable contains 'evil' "
Tagging

- **PLASM (Plaso Langar Að Safna Minna)**
- Tags events based on defined criteria
  - [https://code.google.com/p/plaso/source/browse/#git%2Fextra](https://code.google.com/p/plaso/source/browse/#git%2Fextra)
- Simple definition file

<table>
<thead>
<tr>
<th>TAG NAME</th>
<th>CONDITION (REGULAR FILTER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANOTHER TAG</td>
<td>CONDITION 1</td>
</tr>
<tr>
<td></td>
<td>CONDITION 2</td>
</tr>
</tbody>
</table>

**Example**

Document Printed

(data_type is 'metadata:hachoir' OR data_type is 'olecf:summary_info') AND timestamp_desc contains 'Printed'
Tagging In Action

```
plasm.py tag --tagfile=tag_windows.txt mystorage.dump
Applying tags…
DONE (applied 157 tags)
```

- Run pinfo to see all applied tags

```
pinfo.py mystorage.dump
```

Counter information:

- Counter: Total = 157
- Counter: Application Execution = 91
- Counter: Startup Application = 20
- Counter: AutoRun = 20
- Counter: Document Opened = 20
- Counter: Document Printed = 18
- Counter: File Downloaded = 16

...
The keyword here is “tag contains ‘TAG NAME’”
Tying it All Together

1. Run log2timeline.py on a disk image
2. Run plasm to add tagging
3. Run psort.py to show us application executions
4. Run psort again, this time using a slice

Examine the record found earlier:

```
2012-04-05T17:01:00.148000+00:00,Last Time Executed,Windows Job,
Application: cmd /c c:\windows\system32\spinlock.exe Scheduled by:
SYSTEM Run Iteration: ONCE
```

```
psort.py -q --slice "2012-04-05 17:01:00" mystorage.dump
```

- Displays all records that occurred 5 minutes before and after this timestamp.
Some future ideas*:
- Create an analysis plugin framework (and plugins)
- Change many of the parsers into simpler plugins
- Create codelabs to make it easier for new developers
- Add clustering/grouping of events
- Integrate the tool with GRR
- Support artifacts

* Visit plaso.kiddaland.net and click roadmap for more details
What Can I Do?

- Glad you asked… short answer, plenty
- Contribute code
  - You’ll love the code review process
- Testing the tool and providing feedback
- Throw some suggestions our way
  - Missing some parsers?
  - Output is not intuitive?
  - Need a feature?
- Love documentation?
  - We’d love to accommodate you
  - Plenty of documentation that still needs writing
Questions?

When Chuck Norris analyses timelines he uses

psort.py
storage.dump without filters