Ex-Tip: An Extensible Timeline Analysis Framework in Perl

Michael Cloppert

Ex-Tip: An Extensible Timeline Analysis Framework in Perl

GCFA Gold Certification

Author: Michael Cloppert [mike@cloppert.org]

Adviser: Dominicus Adriyanto

Accepted: May 16, 2008

Outline

Index of Figures

1. Introduction

1.1. Abstract

Digital forensic investigative needs extend well beyond the capabilities provided by classic timeline generation and analysis tools. In this paper, a simple, extensible, and portable timeline framework is discussed in detail. Dubbed Ex-Tip, it is shown that this tool can be used to provide basic timeline capabilities to any variety of input sources, with customizable output for human or programmatic consumption.

1.2. Background

Tools exist to construct timelines based on modify, access, and create times of files on various filesystems to aid in forensic investigations. Sleuthkit's *mactime* in concert with *fls* or *macrobber* is a common example. However, in most investigations, the timeline needs of the forensic analyst have become far more encompassing than simple file activity. Investigations often necessitate a step-by-step recreation of events pulling time data associated with Windows registry entries, anti-virus logs, intrusion detection systems, and any other data available to supplement filesystem activity. At times, both in the lab and in the field, investigators find new time-stamped data that warrants inclusion in a timeline, such as custom application logs. As the digital forensics field matures, the list of critical data available grows

longer, as does the number of timeline visualization tools available for data presentation. Adding to the complexity, the nature of these data sources is dynamic as software versions change.

All of this considered, one can see that a gap has emerged between the timeline data needed by analysts and flexible, portable tools available to easily consume this data - aggregation, normalization, and visualization, to be specific. This paper describes an extensible framework to achieve these ends, with plug-ins provided for common timeline data sources and output formats as proof-of-concept.

1.3. Prior & Related Work

As mentioned above, Brian Carrier's *Sleuthkit* toolset is the de-facto standard in open-source forensics tools as of the writing of this paper. Another powerful collection of free and open-source (FOSS) tools commonly used is Helix (http://www.e-fense.com/helix/). Helix contains Sleuthkit tools, amongst others, in the form of a bootable linux-based CD or DVD. Both contain utilities to generate timelines, but their input and output formats are limited, static, and cannot handle multiple data types.

Various commercial, off-the-shelf (COTS) tools contain basic timeline capabilities. For instance, Guidance Software's *EnCase Forensic* tool

(http://www.guidancesoftware.com/products/ef_index.aspx), in its various versions, provides a "timeline view." By default, this display includes file metadata. It can be expanded to include other file formats, but the barrier for entry is high in either cost or vendor-specific knowledge. Custom scripts must be written in Encase's own language. Additionally, Encase is not as portable a tool as the various FOSS tools mentioned above.

One existing extensible framework is *Zeitline*, a research project by Purdue University's CERIAS – Center for Education and Research in Information Assurance and Security (http://projects.cerias.purdue.edu/forensics/timeline.php). This FOSS tool is a graphical, java-based event analysis tool that enables timeline event analysis from various input sources and hierarchical clustering. Data adapters that have been written into this framework are classic *mactime* and UNIX syslog formats. While this shows promise, it is inactive (last updated in 2006), requires a relatively high barrier for entry to include other data sources, and does not provide for variable output formats.

The only other tool that brushes this application domain is a timeline tool called *EasyTimeline* (http://infodisiac.com/Wikipedia/EasyTimeline/). This multipurpose timeline tool takes a configuration file and graphically represents the results. It makes no attempts to address the other challenges presented in this paper.

2. The Ex-Tip Framework

Ex-Tip aims to address the challenges of normalization and aggregation with a flexible and portable tool. It is a complementary technology to many existing domain-specific tools such as those listed above.

2.1. Ex-Tip Design

Ex-Tip is written in Perl, meaning that any bootable environment that can run Perl can also run Ex-Tip, provided the required supporting libraries are included. All efforts have been made to remove the Perl-specific complexities of the framework outside of the adapter scripts, simplifying the job of the analyst attempting to extend Ex-Tip.

Fundamentally, Ex-Tip is a collection of four types of Perl scripts:

1. The main Perl script, ex-tip.pl
2. The Timeline module which contains the basic input/output/sorting functions of Ex-Tip, Timeline.pm.
3. The various input modules, adapters that convert data from their native format into the Ex-Tip hierarchical hash structure discussed later.
4. The various output modules, adapters that convert data from the Ex-Tip hierarchical

hash structure into whatever output format is required (textual, visual, custom file format, etc.).

Figure 1 illustrates these modules and their relationship to one another. The use of certain parameters when calling ex-tip.pl result in function calls to Timeline.pl, which in turn leverages the various input & output adapter modules using the variables provided on the command line. The specifics of this relationship are described later.

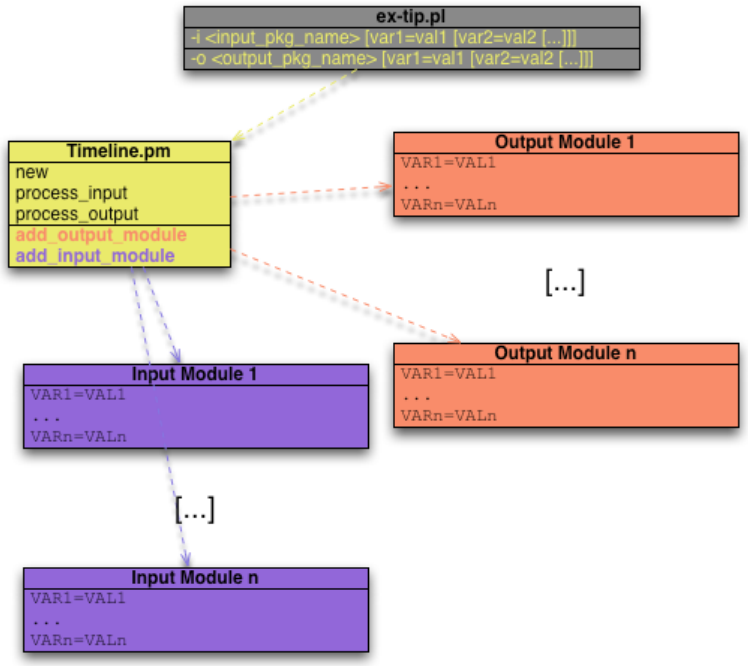

Figure 1: Generic Ex-Tip module relationship

Data is represented in Timeline.pm as a hierarchical hash structure conforming to that illustrated in Figure 2.

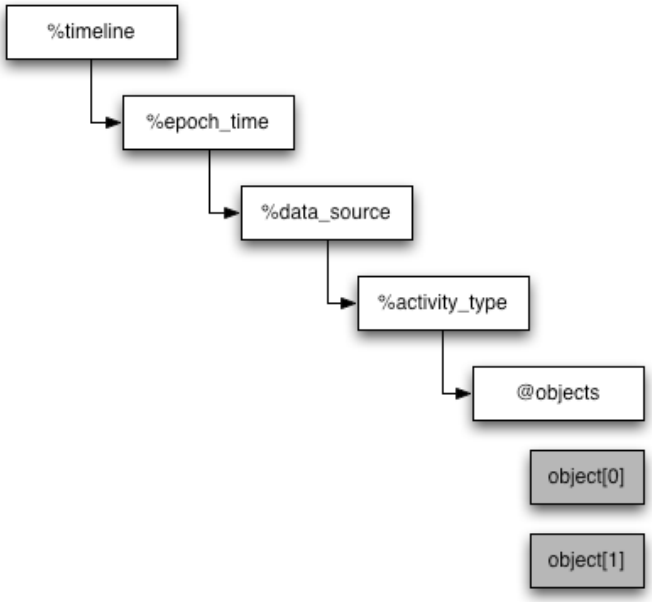

Figure 2: Ex-Tip Timeline Data Hierarchy

The timeline hash has as its children the epoch time[1] of every event inserted into Ex-Tip by the input module adapters. For each epoch time, a hash structure exists for every data source type. This is populated by the input module adapters, and can be any value the analyst desires. Underneath the {time, source} hash is another hash for each activity type that meets this criteria. Again, this is populated by the input module adapter and can take any

[1] "Epoch time," or "Unix time," is the date recorded as the number of seconds since 1/1/1970 00:00:00 GMT. It is typically represented as a 32-bit integer. It has been common in POSIX systems since the 1970's in various forms [Unix Programmer's Manual, 1st Ed.], and specified as a standard in ISO 8601.

value, although it is recommended that this be a single character. Finally, an array is associated with the {time, source, type} triplet for each object that shares these properties.

2.2. Ex-Tip Implementation

As written for this project, Ex-Tip 0.1 supports one output and three input module adapters, the dependencies of which are illustrated in Figure 3.

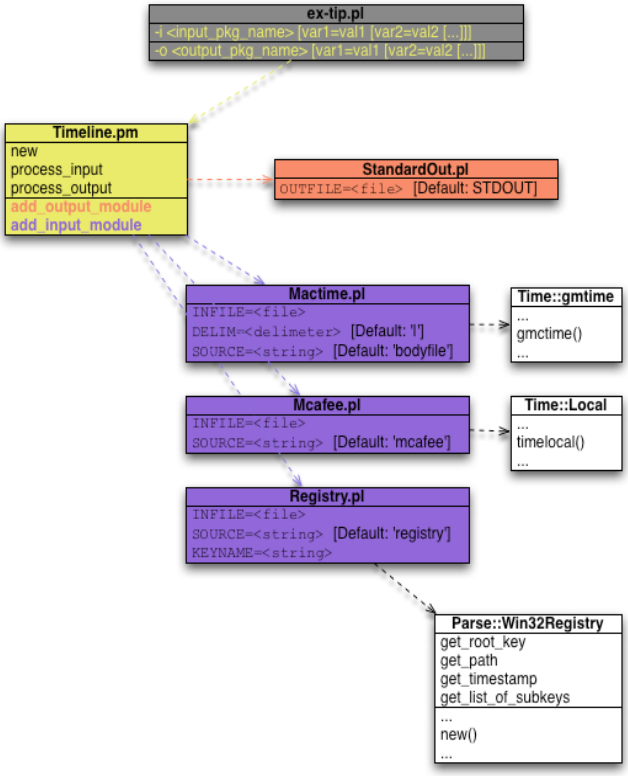

Figure 3: Ex-Tip 0.1 Implementation

Each of these modules will be discussed in detail in Section 3, with sample uses in Section 4.

An illustration of a sample instantiation of Ex-Tip is provided in Figure 4. This is a simplified example of how infected files that were detected by McAfee and modified would be represented in the hash structure described in the previous section. One file is modified for content, aka "Cleaned," the other is deleted outright. Perl's hierarchical hash capabilities allow for unlimited nesting of hashes. The implementation of Ex-Tip leverages this feature so that each timeline object has associated metadata of the entire hierarchy. The flexible nature of Perl's hashing capabilities makes it ideal for adding more object-associated metadata in the future.

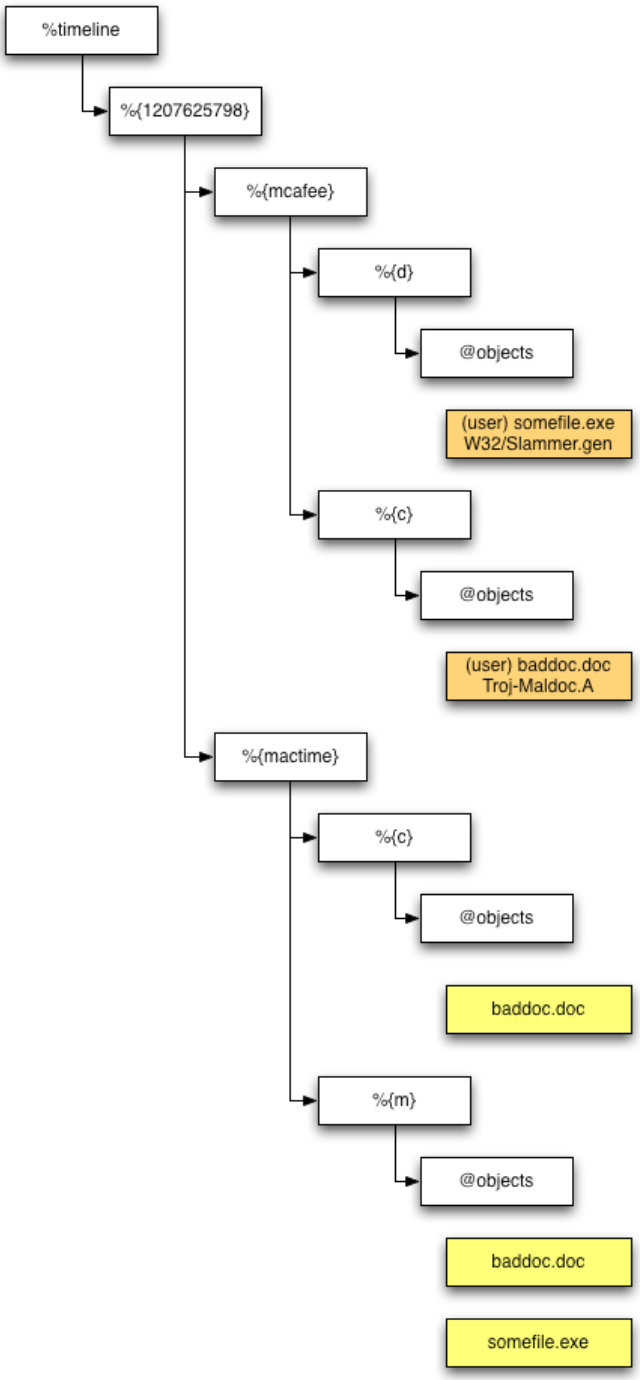

Figure 4: Example hash structure for a file deleted by McAfee

3. Ex-Tip Module Implementation

Ex-Tip modules are of type input or output. At least one of each must be specified in order for Ex-Tip to function. More than one of each type may be specified, but the same module may not be specified more than once.

3.1. Input Modules

The provided input modules with Ex-Tip 0.1 will be described in terms of their output by the StandardOut module, as this clearly reveals the underlying timeline hash structure that has been defined therein. The StandardOut module will be discussed in detail later, but readers familiar with the *mactime* tool's textual output will find that StandardOut is very similar.

Input modules are specified at the command line with the *–i* parameter to ex-tip.pl as follows:

```
$ ./ex-tip.pl –i <Module Name> [VAR1=VAL1 [VAR2=VAL2 […]]]
```

Module Name is the name of the module without the .pl extension. Input modules will typically require an input source be specified by variable assignment. Some input modules will have other options that can be specified with additional variable assignments. There is no limit to the number of variables that can be specified for a particular module.

Multiple input modules of different types can be specified with *–i*.[2] Variable assignments are made to the most recently named module specified by *–i*, and variable names need not be unique within Ex-Tip – only within each individual module.

3.1.1. Classic Mactime Body Files

The goal of the Mactime input module is to represent the basic functionality provided by The Sleuthkit's *mactime* tool. The module takes as input "body" files from the Sleuthkit tools *fls* and *ils*. The basic file action types introduced to Ex-Tip by this module are:

- File modified time - the last time the file's metadata was modified (i.e., MFT entry), denoted by capital "M."
- File access time - the last time the file's metadata was accessed, denoted by capital "A."
- File create time - the time when the file's metadata entry was created, denoted by capital "C."

Commensurate with Ex-Tip's framework, each file entry provided in the body files will

---

[2] Input modules of the same type cannot be specified twice in this version. If this happens, the second specification of a particular input module will replace the first.

necessarily have three separate entries: one each for Modified, Accessed, and Created, even if all three timestamps are identical.

The only variable supported by this module is *INFILE*, and it is a required parameter. This is the *mactime* bodyfile to be read by the module. The module has no dependencies.

Listing 1 is a truncated body file created via *fls* and *ils* with data describing 4 files.

```
0|sda/System.map-2.6.17-1.2142_FC4|0|20|33188|-/-rw-r--r--
|1|0|0|0|832577|1169014481|1152672733|1169014472|1024|0
0|sda/config-2.6.17-1.2142_FC4|0|21|33188|-/-rw-r--r--
|1|0|0|0|65917|1152672733|1152672733|1169014472|1024|0
0|sda/vmlinuz-2.6.17-1.2142_FC4|0|22|33188|-/-rw-r--r--
|1|0|0|0|1803676|1152672733|1152672733|1169014472|1024|0
0|<sda1-dead-22106>|0|22106|33152|-rw-------|0|0|0|0|1169014487|1169014487|1169014487|1024|0
```

Listing 1: Mactime input format

The same data, after having been imported into Ex-Tip and displayed via the StandardOut module, is shown in Listing 2.

```
Wed Jul 12 02:52:13 2006
        [bodyfile]      A       sda/System.map-2.6.17-1.2142_FC4
        [bodyfile]      A       sda/config-2.6.17-1.2142_FC4
        [bodyfile]      A       sda/vmlinuz-2.6.17-1.2142_FC4
        [bodyfile]      M       sda/config-2.6.17-1.2142_FC4
        [bodyfile]      M       sda/vmlinuz-2.6.17-1.2142_FC4
Wed Jan 17 06:14:32 2007
        [bodyfile]      C       sda/System.map-2.6.17-1.2142_FC4
        [bodyfile]      C       sda/config-2.6.17-1.2142_FC4
        [bodyfile]      C       sda/vmlinuz-2.6.17-1.2142_FC4
Wed Jan 17 06:14:41 2007
        [bodyfile]      M       sda/System.map-2.6.17-1.2142_FC4
Wed Jan 17 06:14:47 2007
        [bodyfile]      A       <sda1-dead-22106>
        [bodyfile]      M       <sda1-dead-22106>
        [bodyfile]      C       <sda1-dead-22106>
```

Listing 2: Mactime output format

3.1.2. McAfee Anti-Virus Logs

Three McAfee log file formats were considered for this anti-virus parser:

- On-Demand Logs
- On-Access Logs
- Access Protection Logs

These logs are voluminous, often containing statistics, informational, and warning messages intertwined with clean and delete messages. Access Protection logs, which contain messages about blocked execution based on behavioral rules, were especially difficult to digest notionally alongside more discrete data like file MAC times. Due to inconsistent log formatting, vague protection messages, and a general lack of availability of representative Access Protection logs, the decision was made to only include file clean and delete messages from On-Demand and On-Access logs. These are the two action types introduced to Ex-Tip: Clean, denoted by the lowercase "c," and Delete, denoted by the lowercase "d."

Modern versions of McAfee adhere to the following space-aligned format for log entries recording Clean and Delete messages:

```
<date> <time> <AM | PM> <Cleaned | Deleted> [(message)] <DOMAIN>\<userID>
<process> <file> <detection name> <(detection type)>
```

The mapping of these fields to Ex-Tip fields is articulated in Table 1.

Table 1: Field mappings from Mcafee to Ex-Tip

Ex-Tip Hash	`epoch_time`	`data_source`	`activity_type`	`object`
McAfee Data	<date>, <time>, <AM|PM> conversion by Time::Local	"mcafee"	<Cleaned = "c" | Deleted = "d">	<DOMAIN><userID>, <process>, <file>, <detection name>, <(detection type)>

The only required variable definition for this module is *INFILE*, which is the McAfee log file to be parsed. Note that the assumed timezone for this module is US Eastern. This is not controlled by a variable, but the script could be easily modified to do so. There is one package dependency for this module, *Time::Local*, available through CPAN (http://search.cpan.org/dist/Time-Local/).

Listing 3 is a sample McAfee log file, truncated to include lines of interest. The first lines that contain statistical or stateful information are ignored by the parser. Only the lines with "Cleaned" or "Deleted" specified are even processed.

```
1/11/2008       10:17:19 AM     Statistics:
1/11/2008       10:17:19 AM     Files scanned:  300903
1/11/2008       10:17:19 AM     Files detected:         130
1/11/2008       10:17:19 AM     Files cleaned:  0
1/11/2008       10:17:19 AM     Files deleted:  0
1/11/2008       11:26:07 AM             Engine version                          =       5200.2160
1/11/2008       11:26:07 AM             AntiVirus   DAT version                 =       5204.0000
1/11/2008       11:26:07 AM             Number of detection signatures in EXTRA.DAT =   None
1/11/2008       11:26:07 AM             Names of detection signatures in EXTRA.DAT  =   None
1/11/2008       11:32:46 AM     Not scanned  (scan timed out)   ASPENSYS\rhou
C:\WINDOWS\Explorer.EXE C:\Program Files\Quest Software\Toad for Oracle\TOAD.exe
1/11/2008       12:04:34 PM     Not scanned (The file is encrypted)     ASPENSYS\rhou
C:\Program Files\Internet Explorer\iexplore.exe C:\Documents and Settings\rhuo\Local
Settings\Temporary Internet Files\Content.IE5\0JHBAIR1\TodayLight[1].aspx\TodayLight[1]
[...]
1/13/2008       11:22:13 AM     Cleaned         ASPENSYS\rhou   C:\Program Files\Internet
Explorer\iexplore.exe c:\documents and settings\rhuo\local settings\temporary internet
files\content.ie5\wqe160nf\setup[1].exe  Adware-ZangoSA (Adware)
1/13/2008       11:22:15 AM     Deleted         ASPENSYS\rhou   C:\Program Files\Internet
Explorer\iexplore.exe C:\DOCUMENTS AND SETTINGS\RHUO\LOCAL SETTINGS\TEMPORARY INTERNET
```

Listing 3: McAfee Anti-virus input format

Listing 4 is the same data after having been imported into Ex-Tip and displayed via the StandardOut module. The mapping of McAfee fields to Ex-Tip fields is made clear in this example.

```
Sun Jan 13 16:22:13 2008
        [mcafee]        c       (ASPENSYS\rhou) C:\Program Files\Internet Explorer\iexplore.exe
c:\documents and settings\rhuo\local settings\temporary internet
files\content.ie5\wqe160nf\setup[1].exe : (Adware)Adware-ZangoSA
Sun Jan 13 16:22:15 2008
        [mcafee]        d       (ASPENSYS\rhou) C:\Program Files\Internet Explorer\iexplore.exe
C:\DOCUMENTS AND SETTINGS\RHUO\LOCAL SETTINGS\TEMPORARY INTERNET
FILES\CONTENT.IE5\WQE160NF\SETUP[1].EXE : (Adware)Adware-ZangoSA
```

Listing 4: McAfee output format

3.1.3. Windows Registry Keys

Windows registry files can be valuable in forensic investigations as well. It isn't

commonly known, but all registry keys have a last modified timestamp associated with them that can be crucial in identifying the time across which suspicious activity occurred during an intrusion-related investigation (as compared to, say, an investigation related to pornography). While this date isn't always valuable, as it is only associated with the registry key and not values and is *only* a last-modified timestamp, there are times that this supplemental information can be invaluable, such as in cases where intruders modify file timestamps – amongst others. It is important to note that the files which contain the timestamps are raw, binary registry files; not registry hive exports that are ascii text.

The Registry module accepts the following variables:

- *INFILE* – A required variable, this is the binary registry file that will be parsed
- *KEYNAME* – This is the name of the registry hive that was imported. It is arbitrary text (no spaces), and will be prepended to the key upon import.

In order to parse the windows registry file format, the module leverages the work done by James Macfarlane in *Parse::Win32Registry*, available from CPAN (http://search.cpan.org/~jmacfarla/Parse-Win32Registry-0.30/). This module can be problematic, and will sometimes generate warning or error messages. It also does not work in some environments (Cygwin, for example), however as tested in Fedora the module will parse most modern Windows registry files, and will not terminate when errors or warnings are

encountered. There are a variety of existing perl scripts that parse Windows registry files, but unfortunately all are problematic to one degree or another. *Parse::Win32Registry* gave the author the fewest problems.

Listing 5 shows a portion of a registry file, as displayed by the Ex-Tip StandardOut module after import. To generate this output, the variable assignment *KEYNAME=SOFTWARE* was used when calling ex-tip.pl.

```
Wed Mar 14 14:40:37 2007
        [registry]      M       SYSTEM\ControlSet001\Enum\Root\LEGACY_PARTMGR
        [registry]      M       SYSTEM\ControlSet001\Hardware Profiles\0001
        [registry]      M       SYSTEM\ControlSet003\Enum\Root\LEGACY_PARTMGR
        [registry]      M       SYSTEM\ControlSet003\Hardware Profiles\0001
Wed Mar 14 14:40:43 2007
        [registry]      M       SYSTEM\ControlSet001\Enum\Root\LEGACY_MUP
        [registry]      M       SYSTEM\ControlSet001\Enum\Root\LEGACY_NDIS
        [registry]      M       SYSTEM\ControlSet003\Enum\Root\LEGACY_MUP
        [registry]      M       SYSTEM\ControlSet003\Enum\Root\LEGACY_NDIS
```

Listing 5

3.2. Output Modules

Output modules are specified at the command line with the *–o* parameter to ex-tip.pl as follows:

```
$ ./ex-tip.pl –o <Module Name> [VAR1=VAL1 [VAR2=VAL2 […]]]
```

Module Name is the name of the module without the .pl extension. This usage is identical to that of input modules, simply with a different sentinel switch. As with input modules, there is

no limit to the number of variables that can be specified for a particular module.

Multiple output modules of differing types can also be specified with *–o*.[3] Variable assignments are made to the most recently named module specified by *–o*, and variable names need not be unique within Ex-Tip – only within each individual module.

3.2.1. StandardOut: Classic Mactime Output Format

The classic Mactime output format in the StandardOut module was designed to emulate the timeline data display provided by The Sleuthkit's *mactime* tool. Listings provided throughout this paper serve as examples of this output format.

The StandardOut module has no required parameters. If specified, the *OUTFILE* variable will cause the module to write all output to a file. Otherwise, the module prints all output to *STDOUT*. In order to print epoch time in human-readable format, StandardOut depends on the *Time::gmtime* module available from CPAN (http://perldoc.perl.org/Time/gmtime.html).

[3] Output modules of the same type cannot be specified twice in this version. If an output module is specified twice with different parameters, the second specification will replace the first.

4. Practical Ex-Tip

Ex-Tip should be easy to use for any analyst comfortable with a shell prompt. However, to remove any uncertainty, this section describes its installation and use.

4.1. Getting Ex-Tip

A static copy of the proof-of-concept code is stored at http://www.cloppert.org/ex-tip-0.1.zip for the purposes of evaluating this practical. It will also be stored and maintained long-term thanks to Sourceforge.net at http://sourceforge.net/projects/ex-tip. Ex-Tip will be released under the Gnu Public License (GPL), according to the terms and conditions documented in full at http://www.gnu.org.

4.2. Ex-Tip Installation

Before Ex-Tip is used, all module dependencies must be resolved. For version 0.1, this means installing the following from CPAN:

- *Time::Local*
- *Time::gmtime*
- *Parse::Win32Registry*

The most likely problems one is likely to encounter with Ex-Tip involves installation and

use of the dependent modules. *Time::gmtime* and *Time::Local* are part of the standard Perl libraries. *Parse::Win32Registry* is not, and CPAN testers have documented problems in some environments. All modules can be installed using CPAN as illustrated in Listing 6, which demonstrates the installation of Time::Local.

For proper operation of ex-tip.pl, all associated Perl scripts and modules must be copied to the user's *PATH*, or current working directory (CWD). This includes:

- ex-tip.pl
- Timeline.pm
- All modules. For version 0.1:
    - StandardOut.pl
    - Mcafee.pl
    - Mactime.pl
    - Registry.pl

```
[root@webservices ~]# perl -MCPAN -e shell
Terminal does not support AddHistory.

cpan shell -- CPAN exploration and modules installation (v1.7602)
ReadLine support available (try 'install Bundle::CPAN')
cpan> install Time::Local
CPAN: Storable loaded ok
Going to read /root/.cpan/Metadata
  Database was generated on Tue, 14 Aug 2007 22:36:47 GMT
CPAN: LWP::UserAgent loaded ok
Fetching with LWP:
[…]
Writing /usr/lib/perl5/5.8.8/i386-linux-thread-multi/auto/Time/Local/.packlist
Appending installation info to /usr/lib/perl5/5.8.8/i386-linux-thread-multi/perllocal.pod
  /usr/bin/make install  -- OK

cpan>
```

Listing 6: Installation of Time::Local using CPAN

4.3. Ex-Tip Usage Example

A few usage examples are given here to support descriptions of Ex-Tip given earlier in the paper. The most basic use, emulating The Sleuthkit's *mactime* functionality of taking in a bodyfile and printing its contents in human-readable format to screen, is as follows:

./ex-tip.pl –i Mactime INFILE=/some/dir/bodyfile.txt –o StandardOut

In order to take this output and write it to disk, the output can be redirected using the greater-than symbol "**>**", or specified as a parameter to StandardOut, as shown respectively below:

./ex-tip.pl –i Mactime INFILE=/some/dir/bodyfile.txt –o StandardOut **>** output.txt

./ex-tip.pl –i Mactime INFILE=/some/dir/bodyfile.txt –o StandardOut

OUTFILE=output.txt

Multiple input and output modules of different types can be specified. In Listing 7, all three input modules are used, with the StandardOut module writing the result to disk. The output of Ex-Tip is also included here.

```
$ ./ex-tip.pl -i Mcafee INFILE=./mcafee_samples/OnAccessScanLog.txt -i Mactime
INFILE=test.body -i Registry INFILE=./SYSTEM KEYNAME=SYSTEM -o StandardOut
OUTFILE=./test_output.txt
Adding input packages
        Adding package Mcafee
                Setting 'INFILE'='./mcafee_samples/OnAccessScanLog.txt'
        Adding package Mactime
                Setting 'INFILE'='test.body'
        Adding package Registry
                Setting 'INFILE'='./SYSTEM'
                Setting 'KEYNAME'='SYSTEM'
Adding output packages
        Adding package StandardOut
                Setting 'OUTFILE'='./test_output.txt'
Processing package Mcafee
Processing package Mactime
Processing package Registry
Processing package StandardOut
        Writing to output file ./test_output.txt
$
```

Listing 7: Three input and one output modules example use

4.4. Limitations and Considerations

As noted earlier, Parse::Win32Registry is problematic in some operating environments. It works as tested in Fedora Core 4, but will occasionally generate very verbose warnings or errors for some registry files. In cases such as this, most registry data will still be imported

and the scripts will not terminate. Redirecting STDERR to /dev/null will allow Ex-Tip to proceed to the end of the file so that output validation is possible – recommended in these cases.

Performance and memory footprint of Ex-Tip were not considered for this proof-of-concept. The purpose of this version was merely to demonstrate the benefits of a tool with the properties documented herein. As a result, it is possible that large data sets may not execute as quickly as expected. Provided sufficient memory is available, however, data set size should not prohibit the tool from executing successfully.

This early version of the tool does not permit multiple input or output modules of the same type to be used. For instance, multiple registry files cannot be imported using '-i Registry'. In such a case, the last specification will be the only one instantiated by Ex-Tip.

5. Conclusion & Future Work

Ex-Tip is without a doubt incomplete. The most immediate need to facilitate adoption of the framework is the addition of more modules. Of particular use would be Windows Eventlog and other Anti-Virus vendor file format parsers. Other hurdles exist within the framework itself between this proof-of-concept and a full release version, such as the capability to use more than one instance of each input or output plugin, e.g. adding binary SYSTEM and SOFTWARE hives with the Windows Registry plugin. The implementation should be studied with performance considerations in mind. It is likely that computational and memory efficiency gains can be achieved.

While the current realization of the framework does have limitations, this approach proves the value of an extensible framework for forensic timeline creation. It saves the valuable time of digital forensic investigators by solving the problems of data normalization, ordering, and visualization, and as a consequence reduces the cost of such investigations. It is adaptable to various visualization tools with open file formats, and can be extended to meet the needs of future input file types through the creation of a simple adapter. Written in Perl, and well-documented, this approach gives investigators easy access to the framework's extensibility so that it may be leveraged ad-hoc or in the field in classic forensic operating environments.

6. References

Comprehensive Perl Archive Network, CPAN (http://www.cpan.org)

The Sleuth Kit & Autopsy: Digital Investigation Tools for Linux and Other Unixes (http://www.sleuthkit.org/)

ISO 8601:2004, Date and Time Format (http://www.iso.org/iso/date_and_time_format)

Thompson, K., and Ritchie, D.M., *Unix Programmer's Manual*, Bell Labs, 1971.

Last Updated: July 3rd, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Amsterdam August 2020 Part 1	Amsterdam, NL	Aug 03, 2020 - Aug 08, 2020	Live Event
SANS Reboot - NOVA 2020	Arlington, VAUS	Aug 10, 2020 - Aug 15, 2020	Live Event
SANS FOR508 Canberra August 2020	Canberra, AU	Aug 17, 2020 - Aug 22, 2020	Live Event
SANS Amsterdam August 2020 Part 2	Amsterdam, NL	Aug 17, 2020 - Aug 22, 2020	Live Event
SANS Virginia Beach 2020	Virginia Beach, VAUS	Aug 30, 2020 - Sep 04, 2020	Live Event
SANS Philippines 2020	Manila, PH	Sep 07, 2020 - Sep 19, 2020	Live Event
SANS London September 2020	London, GB	Sep 07, 2020 - Sep 12, 2020	Live Event
SANS Baltimore Fall 2020	Baltimore, MDUS	Sep 08, 2020 - Sep 13, 2020	Live Event
SANS Munich September 2020	Munich, DE	Sep 14, 2020 - Sep 19, 2020	Live Event
SANS Network Security 2020	Las Vegas, NVUS	Sep 20, 2020 - Sep 25, 2020	Live Event
SANS Northern VA - Reston Fall 2020	Reston, VAUS	Sep 28, 2020 - Oct 03, 2020	Live Event
SANS San Antonio Fall 2020	San Antonio, TXUS	Sep 28, 2020 - Oct 03, 2020	Live Event
Oil & Gas Cybersecurity Summit & Training 2020	Houston, TXUS	Oct 02, 2020 - Oct 10, 2020	Live Event
SANS OnDemand	OnlineUS	Anytime	Self Paced
SANS SelfStudy	Books & MP3s OnlyUS	Anytime	Self Paced