MultiLog: A Tool for the Control and Output Merging of Multiple Logging Applications

Jonathan Woodruff and Jason Alexander

School of Computing & Communications

Lancaster University, Lancaster, United Kingdom

+44 (0)1524 510371 +44 (0)1524 510508

j.woodruff@lancaster.ac.uk j.alexander@lancaster.ac.uk

Abstract

MultiLog is a logging tool that controls, gathers, and combines the output, on-the-fly, from existing research and commercial logging applications or ‘loggers’. Loggers record a specific set of user actions on a computing device, helping researchers to better understand environments or interactions, guiding the design of new or improved interfaces and applications. MultiLog reduces researchers’ required implementation effort by simplifying the set up of multiple loggers and seamlessly combining their output. This in turn increases the availability of logging systems to non-technical experimenters for both short-term and longitudinal observation studies.

MultiLog supports two operating modes: ‘researcher mode’ where experimenters configure multiple logging systems, and ‘deployment mode’ where the system is deployed to user-study participants’ systems. Researcher mode allows researchers to install, configure log filtering and obfuscation, observe near real-time event streams, and save configuration files ready for deployment. Deployment mode simplifies data collection from multiple loggers by running in the system tray at user log-in, starting loggers, combining their output, and securely uploading the data to a web-server. It also supports real-time browsing of log data, pausing of logging, and removal of log lines.

Performance evaluations show that MultiLog does not adversely affect system performance, even when simultaneously running several logging systems. Initial studies show the system runs reliably over a period of ten weeks.

Introduction

Client-side logging software runs on a computer to automatically gather data on a user's interactions. This is now a standard method for recording user actions to understand behaviour and improve future interface design as it provides complete, accurate, and machine processable data that catalogues interactions. These loggers may be built into an application, be installed as an extension to an application, or run independent of any application. Typically they are focused on a particular aspect of user behaviour, such as web-browsing patterns (Montgomery et al., 2001), window-switching habits (Oliver et al., 2006), or navigation preferences (Juvina et al., 2004). Increasingly, longitudinal log analyses are also being used to inform the design of new interface artefacts: Alexander et al. (2009) derived the design of their *Footprints Scrollbar* from log analysis of within-document revisitation, while Tak et al. (2009) used longitudinal log data to inform window-switcher design. Hutchings et al. (2004) used the VibeLog software to log UI events in order to assess desktop complexity in single and multiple-monitor users. Unfortunately, as loggers typically monitor specific behaviours, researchers can find re-use difficult, as existing systems often do not capture all applicable actions. However, while multiple different loggers in combination can provide the required dataset, their configuration and management is time-consuming and combining their output difficult.

Logging is useful both within and beyond the HCI community. An example of a field outside the HCI community where logging is helpful to researchers is Interactive Information Retrieval (IIR) which also uses collected log data to analyse and draw conclusions from users' behaviour. Kelly et al. (2013) provide a deep overview of IIR evaluation studies and the use of logging as a data collection technique used within them. Other relevant areas which prompted

motivation for this article include: data mining such as work by Iváncsy et al. (2006) and audit logging in distributed systems (Yavuz et al., 2009).

To support experimenters in the deployment of multiple logging systems and ease later log analysis by standardising output, we created MultiLog. MultiLog (Figure 1) is a tool that simultaneously controls, gathers, standardises, and merges the output from pre-existing logging applications. Further, it supports both technical and non-technical experimenters in the deployment of longitudinal logging-based user studies by managing logger start-up, log filtering and obfuscation, and securely uploading log files. By easily running multiple logging systems MultiLog also encourages the re-use of pre-existing loggers.

The power of MultiLog stems from its ability to combine output from any pre-existing logging application (providing it has timestamp data) and automate the output combination in near real-time. This enables researchers to understand native PC use in detail by logging actions within multiple different applications (e.g. combining an email logger and file system logger would allow detailed inspection of how a user processes email attachments).

Further, MultiLog also provides the ability to log and combine events from outside the native PC environment, from devices such as eye-trackers and EEG sensors. This is supported through TCP/UDP logging (a common input method for external sensors). This further increases MultiLog's power, allowing researchers to experiment with otherwise complex software and hardware logging system setups. Examples include: investigating how applications on the user's PC interact with external devices or inputs (from the user) such as eye trackers, and finger pressure input. Logs from both the external devices and internal applications can be combined seamlessly in MultiLog to give a rich data set. Researchers could also use MultiLog to understand processing inefficiencies by combining CPU usage and number of applications

open/application interaction or even web server logs and those describing the temperature in a server room for example. MultiLog's flexibility allows non-technical researchers to quickly and easily combine the logging capabilities from pure-software, external sensor-driven, and even mobile systems.

We validated three key aspects of MultiLog: (1) That a diverse range of existing logging systems work successfully with MultiLog; (2) That MultiLog maintains data integrity; and (3) That resource consumption is acceptable. First, we checked that a wide-range of logging applications and external sensor systems were compatible with MultiLog (both in terms of start and stop configuration, and data capture). Second, data integrity checks were performed to ensure MultiLog successfully parsed and recorded all required data from the logging applications. Third, we conducted computing resource consumption tests to ensure MultiLog did not cause performance degradation when monitoring multiple loggers.

In this article we describe MultiLog and its architecture, test MultiLog with a range of off-the-shelf logging solutions, and conduct a performance analysis of the system. MultiLog is available to researchers via www.scc.lancs.ac.uk/MultiLog. The MultiLog software is free to download and use for non-commercial and research purposes.

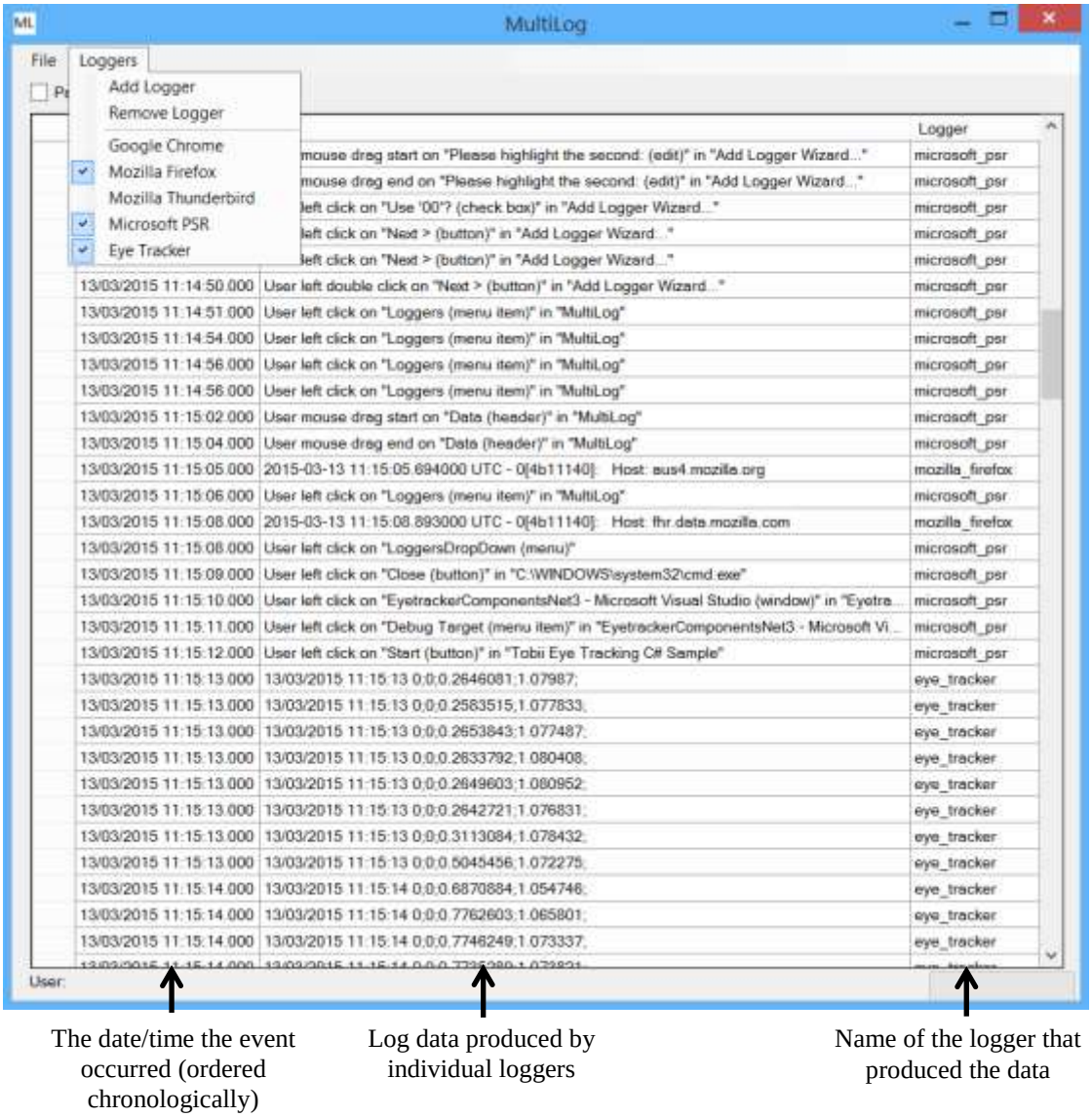

Figure 1: MultiLog's researcher mode with Microsoft PSR, Mozilla Firefox and Tobii Eye Tracker loggers enabled. This allows researchers to see the locations on screen users are looking while performing their everyday tasks.

Related Work

Logging of users' actions is valuable in understanding how people utilise applications and interfaces on their computers; the output can then be used to inform their future (re-)design. This

section reviews logging systems in general, a range of existing loggers, and previous work into merging output from different loggers.

Low-Level Loggers

Low-level loggers typically just record basic input actions such as mouse movements and keystrokes. Examples include *Actual Keylogger Software* (Actual Keylogger, 2014), *REFOG Free Keylogger* (REFOG, 2014) and *A Stealthy GPU-based Keylogger* (Ladakis et al., 2013). Such loggers allow researchers to analyse, for example, how people use certain keyboard shortcuts (Peres et al., 2004) or how fast they type (Kinkead, 1975). The biggest disadvantage of low-level loggers is their lack of context. Without knowing which buttons or menu items are clicked or which textboxes text is enter into, researchers can only make general statements about user behaviour.

High-Level Loggers

High-level loggers are either application-specific or generic and provide additional contextual information to that of low-level loggers. They are either targeted at a particular software application or more generally at an operating system.

Application-Specific Loggers

Application-specific loggers are sometimes developed and shipped as part of a software package and often encourage users to ‘opt-in’ to product improvement programmes (e.g. the (Adobe Customer Improvement Program, 2014) and (Microsoft Customer Experience Inprovement Program, 2009)). Other loggers, such as *Microsoft Outlook Logging* (Microsoft

Corporation, 2003) and *Microsoft Visual Studio Logging* (Microsoft Visual Studio, 2014) support logging for diagnostic and troubleshooting purposes. Log files are saved to disk and can be sent to the manufacturer for inspection. Examples from the research community include *AppMonitor*, a logger to document detailed events performed within Microsoft Word and Adobe Reader (Alexander et al., 2008) and *OpenOffice.org Interceptor*, an application which uses a hybrid technique to log events inside OpenOffice.org applications (Dostál et al., 2011).

An increasing number of applications include some form of high-level logging software. Browsers such as Google Chrome[1] contain an in-built logger that records a list of websites visited; such logging is invoked via the command line. Other applications, such as those from the Mozilla family, also support this level of logging. Unfortunately these loggers have to be manually started and stopped via the command line and work on extracting relevant lines from the output (depending on the research being implemented) needs to be conducted, which non-expert users may struggle with.

Application-specific loggers' tight integration with specific software means they can log detailed contextual events within the application, but cannot provide insights into the external context.

Generic Loggers

Generic loggers gather information from basic operating system events such as window focus changes and detecting when applications are started or stopped. Examples include *RUI* (Kukreja et al., 2006), *PyLogger* (Tak & Cockburn, 2009) and *VibeLog* (Oliver et al., 2006) that monitors the windows a user switches between. The tool Morae (TechSmith, 2015) is also relevant and collects data for market research purposes on usability and other types of pre-release

[1] www.google.com/chrome

product testing. While these cover a broad spectrum of basic actions users undertake during operating system-level interaction, the level of detail differs between loggers.

Generic loggers have the advantage of providing external contextual information as basic events across the whole computer system are monitored and they are not attached directly to a specific application.

Screen Recorders

Screen-recorders collect a series of screenshots or video feeds of the user's desktop providing full details of their actions both within and between applications. They are often complemented with a low-level logging system in order to provide accurate timing information that would be tedious to extract from the video stream (e.g. *Wintective* (Wintective, 2014)). The main limitation with screen-recorders is their resource requirements: recording for extended periods of time can be resource intensive for both CPU usage and storage. Examples include *CamStudio* (CamStudio, 2014), *Rylstim* (Rylstim Screen Recorder, 2014) and *Ezvid* (Ezvid, 2014) which collect video feeds of the user's desktop.

Log-Merging Software

Merging the data collected from multiple loggers is not new. IBM have in the past worked in the area of autonomic computing where they devised the *Generic Adapter Logging Toolkit* (Grabarnik et al., 2004) which provides a framework for transforming event-based system information into a standard format. Although it transforms event data into a generic format, it does not control the applications producing the log files, nor does it cater for researchers with only basic computer knowledge as an adapter (software module) has to be

written or, as seen in further work (Math et al., 2009), a regex defined for each file which is performance-intensive. A *Generic Log Analyser* (Shahzad, 2013) has also been developed as part of a master's project but instead focuses on tracking down issues in the mobile telecommunications area, it has no control over the applications, and requires an XML schema to be provided in order to parse the files.

Other commercial applications such as *Log Monitor* (Log Monitor, 2014) and *Tiny Log Monitor* (Tiny Log Monitor, 2014) bring logger output together into one application. *Log Monitor* allows the user to open multiple log files and watch them for changes while *Tiny Log Monitor* supports the same functionality but with the addition of adding regex patterns to format the output. In both these applications, the output from different log files is not saved, merged, uploaded, or log lines individually removal, all of which are key features of MultiLog.

Description of MultiLog

MultiLog is a research tool that controls, gathers, filters, and combines the output, on-the-fly, from existing research and commercial logging applications. It allows researchers to easily deploy multiple software logging systems to observe user behaviour in either short- or long-term user studies. Automatic log uploading facilitates large-scale data collection.

The system gathers log data on-the-fly: when a logger is enabled, MultiLog actively polls the corresponding log file (or listens on the specified TCP/UDP port) at an interval configurable by the researcher (set to one-minute by default) and checks for updates. If changes are detected (or new data is received on the open TCP/UDP socket), the relevant lines are extracted from the log file, formatted to MultiLog's pre-defined format (as shown in Figure 1), presented in the main interface, and written to an output database.

MultiLog is designed for two groups of users — researchers and study participants — with each having a distinct mode of operation: *Researcher Mode* and *Deployment Mode.* *Researcher Mode* provides the user with full control and configuration ability, while *Deployment Mode* is intended for user study deployments with settings controlled via a configuration file.

Researcher Mode

By default, MultiLog runs in *Researcher Mode*, where the user sees the full user interface, is able to add and remove loggers, can start and stop loggers, and can view the log output from all currently active loggers, as shown in Figure 1. This mode allows researchers to experiment with logger configurations, examine the combined output from loggers, and prepare logging environments for deployment during a user study.

A key feature of MultiLog is its 'plug and play' architecture that allows the researcher to 'plug-in' any existing logger, at any time. MultiLog will work with any existing logging application as long as the researcher can provide the executable name, start and stop commands, the location of the continually-updated log file (or port number if the logger outputs data to a TCP/UDP socket), the position of the timestamp within this output (or the attribute/element that contains the timestamp if the log is in XML format) and an idea of which log lines are required to appear in the output. The 'plug and play' architecture allows even non-technical researchers to quickly configure a series of logging systems. Once configured, the researcher can manually start and stop each logger through MultiLog's user interface, sending appropriate signals to the relevant logger.

Researchers can also choose to filter incoming log lines to reduce the amount of information collected. MultiLog supports filtering via line matching to include/exclude text provided by the researcher at configuration time.

The log file polling interval is configurable by the researcher. By default, this is set to one minute, selected as a result of a trade-off analysis between obtaining real-time data, without experiencing degradation in performance (during configuration, researchers will often wish to reduce this value to immediately see the result of their changes). Data received from loggers that output to TCP/UDP socket is automatically received and processed in real time and thus, the polling interval does not apply to these loggers.

Researchers can 'save' the current logger setup (enabled loggers, filters, and polling interval) and generate a configuration file ready to deploy the logger in *Deployment Mode*.

Deployment Mode

Deployment Mode helps researchers to quickly 'roll out' the application to many computers using MultiLog's executable and an editable configuration file. In this mode, no interface is displayed and the logger runs 'silently' in the user's system tray. The configuration file provides details of each logger to be run (name, executable location, start and stop commands, location of the log file (or port number if the logger outputs data to a TCP/UDP stream), timestamp, and filtering data). If the relevant flag is set inside this file, its contents are read by MultiLog on start-up and the relevant loggers are started with MultiLog minimised to the user's system tray.

Users can open the interface from the system tray icon, view logged actions, remove individual lines if they do not wish these to be uploaded, or pause logging completely. The log

lines are stored locally in a database that is automatically uploaded via a secure FTP connection to a server once daily.

In an effort to reduce privacy issues surrounding logging, MultiLog can hash the data part of a log line or detect and hash URLs. As an example, when URL hashing is enabled via the add logger wizard in *Researcher Mode*, the URL `http://www.bbc.co.uk/news/uk/` could appear as `http://www.bbc.co.uk/HGTRFDH`. When enabled, MultiLog detects and hashes the path part of the URL, preventing the exact website address from appearing in the output (although identical URLs will hash to the same value). Hashing of the data part of the log line is also set up in the add logger wizard where lines containing certain textual phrases can be hashed.

Deployment

MultiLog saves log data into a local SQLite database that is then uploaded to a server. The local database is then truncated to prevent large amounts of log data accumulating on the user's computer. The researcher configures the connection by providing the address, username, and password of the remote web server. Data can be extracted by non-technical researchers by using MultiLog's re-combination software which combines the output for a given user into a text file.

Summary

The main features of MultiLog are: (1) Two distinct modes of operation for different audiences; (2) Its 'plug and play' architecture allowing on-the-fly addition and removal of loggers; (3) On-the-fly gathering, combination, and display of logged data; (4) Fully featured *Deployment Mode* allowing it to start-up and run silently in the user's system tray, allowing user

'pausing' and where necessary removal of log data, and hashing to address privacy issues; (5) Log files are securely uploaded to a server on a daily basis.

MultiLog Architecture

MultiLog is written in C# in Microsoft Visual Studio 2013. It has been built to run on Windows 7, Windows 8 and Windows 8.1. MultiLog's high-level architecture is shown in Figure 2.

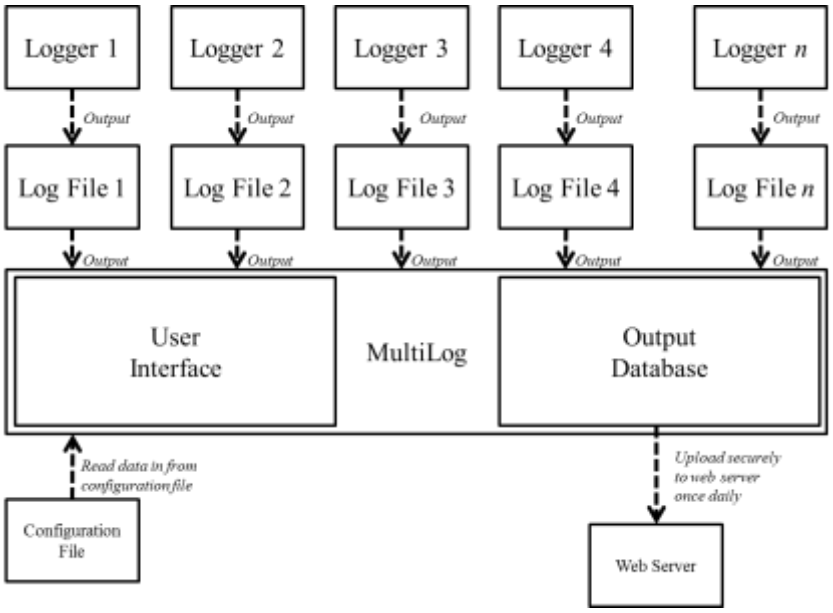

Figure 2: MultiLog architecture

Overview

In its simplest terms, the pre-existing logging applications continue to operate as normal and MultiLog captures their output and merges it into a single interface and output database. MultiLog regularly monitors the logging application's log file's contents (by default once a minute, but this is configurable) or listens on the specified TCP/UDP port for a stream of continuous data. When changes are detected or new data is received, it extracts the data from the last line it read to the end of the file or processes the new TCP/UDP data. Internally, MultiLog

keeps a record of the last position read in each log file. This reduces the overhead incurred by continually reading entire log files (that often become large).

MultiLog can handle three types of logging applications: *Command-Line Loggers, Stand-Alone Loggers* and *TCP/UDP Loggers*. Command-line loggers use additional flags to enable internal logging mechanisms on start-up (e.g. Google Chrome[2], Mozilla Firefox[3] and Mozilla Thunderbird[4]); stand-alone loggers are either dedicated logging applications or are applications that are pre-configured to log events during interaction (e.g. Microsoft PSR); TCP/UDP loggers are systems which send continuous data on specified TCP/UDP sockets. Examples of this type of loggers from the HCI community include eye trackers or EEG systems. The following sections describe how these logger types 'plug-in' and interact with MultiLog.

Command-Line Loggers

Many applications now ship with internal logging mechanisms built into the application. These are typically used when trying to trace program bugs and are enabled by setting appropriate command-line flags when starting the application. For example, a user can enable logging in the Google Chrome browser from the command line by running the command `chrome.exe --enable-logging --v=1` in the directory that contains the Chrome executable file. Chrome then generates a continually updated log file of website visits along with other browser events inside the user's home directory. MultiLog comes pre-configured for logging with popular command-line loggers such as Google Chrome, Mozilla Firefox, and Mozilla Thunderbird and allows researchers to add additional command-line loggers through the add logger wizard.

[2] http://www.chromium.org/for-testers/enable-logging
[3] https://developer.mozilla.org/en-US/docs/Mozilla/Debugging/HTTP_logging
[4] https://wiki.mozilla.org/MailNews:Logging

The main disadvantage of command-line loggers is that they typically cannot be 'stopped' while the application is running (for example, to stop command-line initiated logging in Google Chrome, the user must close and restart the application without the command-line arguments). When a user *pauses* MultiLog logging from its interface, and it is logging from an 'unstoppable' application, MultiLog discontinues log polling of that application and keeps track of the last allowable log line. When logging resumes, MultiLog only continues log line reporting from the time logging was re-enabled (and does not back-read events that occurred during the paused time). When an individual logger is *suspended* or *removed* from MultiLog, MultiLog will provide appropriate warning messages before attempting to close the application.

Stand-Alone Loggers

Stand-Alone loggers are independent applications that record activities within one or more applications or systems, or themselves generate logging information as part of their normal operation. The Microsoft Problem Steps Recorder (PSR) is a typical example that records a wide selection of log events across the whole operating system such as clicks and menu selections, key-presses and shortcuts. MultiLog can typically start and stop external loggers without interfering with monitored applications. Researchers also add stand-alone loggers via the add logger wizard.

TCP/UDP Loggers

Unlike command-line or stand-alone loggers that record data into log files or databases, external sensors typically communicate their data via a TCP/UDP port. MultiLog also supports

logging data through this mechanism. Loggers that use TCP/UDP communication are common in the HCI community and are often incorporated in devices such as eye trackers or EEG systems. MultiLog supports the same add, remove, and pause operations on TCP/UDP loggers as with command-line and stand-alone logger. The primary difference in operation is that TCP/UDP loggers always update in real-time (and do not require a polling interval)—this prevents excess and unnecessary data buffering.

Set up & Use

MultiLog allows researchers to add any pre-existing research or commercial logging application to its list of active loggers. Loggers are added through a wizard interface by specifying certain information about the logging application, such as: its name, log file location or TCP/UDP details if the logger uses sockets, how to start and stop it, and an idea of which log lines are required to appear in the output. Once added, a logger can be started and stopped via the MultiLog interface. Once all required loggers are set up, deployment mode can be configured ready for deployment onto participants' computers. This allows the researcher to generate a configuration file for the currently active loggers. Once created, this file, along with the MultiLog executable, can be dropped onto multiple participants' machines ready for data collection. The configuration is a plain-text file that can also be manually edited if required.

MultiLog's Parsing of Log Files

MultiLog supports any logging application that produces line-by-line plain text or XML log files, or data received on TCP/UDP sockets. MultiLog parses each new log line, extracts the timestamp, and treats the remainder of the line as 'data'. XML data is flattened into a single line

entry (by extracting the elements/attributes marked as ‘timestamp’ and ‘data’ by the researcher at configuration time); consistent formatting for all input streams makes post-collection analysis simpler. TCP/UDP streams are continuously received (and most are appropriately pre-timestamped) and therefore can be added to the MultiLog interface and database in real time. MultiLog only requires that it can read and understand the timestamp; the ‘data’ portion of the log line may be pre-encrypted by the logging application.

MultiLog assumes that log files (or TCP/UDP streams) are continually updated by the logging application and that data is written in a linear manner (i.e. for loggers using a log file, the applications do not rewrite or insert lines earlier in the file). Our testing (see ‘Validation’) showed this to be the case for the vast majority of loggers. We built a re-usable ‘work-around’ to support Microsoft PSR’s unusual output generation. PSR’s output file is only generated when the application is stopped, so MultiLog regularly (currently once a minute – determined by the current polling interval) stops and immediately restarts PSR in order to obtain an output file. This feature can be increased or reduced by changing MultiLog’s general polling interval value from the interface.

MultiLog Validation

To confirm that MultiLog behaved as expected under various conditions we examined three aspects of the system: (1) That a diverse range of existing logging systems work successfully with MultiLog; (2) That MultiLog maintains data integrity; and (3) That resource consumption is acceptable.

Compatibility with Existing Logging Systems

We tested a diverse range of publically available logging systems to check their compatibility with MultiLog. The selection of loggers, and the results of these tests are shown in Table 1. The majority (23/33) work successfully with MultiLog. Of the remainder that partially worked or did not work the main issues identified were: (1) The log file is locked by the Operating System so is inaccessible by MultiLog (Microsoft Visual Studio (Microsoft Visual Studio, 2014)); (2) The timestamps in the log files measure time since the logger started, not a general measure of time (Inputlog (Leijten et al., 2005), Translog (Hansen, 1997) & WebQuilt (University of Washington)); (3) The logger spreads log data across multiple log files (Mendeley Desktop log files (Mendeley) & Kidlogger (Kidlogger)); and (4) All data in the file is compressed/encrypted so MultiLog cannot access a timestamp to order events (Skype log files (Skype), SoftActivity Keylogger (Soft Activity) & Revealer Keylogger (Logixoft)). The first (locked log file) and last (encrypted file) issues are outside the control of MultiLog. We solved issue 2—timestamps from start time—by adding support for MultiLog to use the last modified time on the log file as the timestamp; Issue 3—multiple log files—will be addressed in future work. Issue 4 is a limitation of MultiLog's approach to extracting log information.

We also include in table 2, output from a text (Google Chrome), XML (Microsoft PSR) and UDP (Arduino light sensor) logger providing an example of the raw output data produced by the logging application and the data after MultiLog has parsed it.

Table 1

A list of loggers tested with MultiLog. Y = logger works as expected, N = logger does not work, ! = logger partially works but there are known issues as detailed in the Notes column.

Logger Name		***Output***	***Notes***
Microsoft Problem Steps Recorder (PSR) (Microsoft Corporation)	Y	XML	
Google Chrome in-built logger (Google Chrome, 2014)	Y	Text	
Mozilla Firefox in-built logger (Mozilla Firefox)	Y	Text	
Mozilla Thunderbird in-built logger (Mozilla Thunderbird)	Y	Text	
Drag (developed by MultiLog authors)	Y	Text	Records mouse drags.
Drag-and-Drop (developed by MultiLog authors)	Y	Text	Records Drag-and-Drop actions including the name of the object being dragged, where it was dragged from and where it was dragged to.
Window Switch (developed by MultiLog authors)	Y	Text	Records window switches and the name, dimensions and size of windows.
Clipboard (developed by MultiLog authors)	Y	Text	Records cut, copy and paste actions.
Process Start Monitor (developed by MultiLog authors)	Y	Text	Records the time when a new process was started by the user or system.
WEKA Data Mining (Hall et al., 2009)	Y	Text	
AppMonitor (Alexander et al., 2008)	Y	Text	
Windows Update log files (Windows Update)	Y	Text	
VMWare log files (VMWare)	Y	Text	
Microsoft Outlook log files (Microsoft Outlook, 2014)	Y	Text	
Adobe ARM log files (Adobe)	Y	Text	
Internet Explorer Maintenance (brndlog) log files (Internet Explorer)	Y	Text	
Windows DTC install log files (Microsoft Distributed Transaction Coordinator)	Y	Text	
User Logger (User Logger)	Y	Text	
JEdit (Eklundh et al., 2003)	Y	Text	

TeamViewer (TeamViewer, 2014)	Y	Text	
Ubuntu log files (alternatives.log, auth.log, dpkg.log, kern.log and syslog) (Ubuntu)	Y	Text	
Tobii Eye Tracker	Y	UDP	Data continuously received on UDP port 11000. Processed by MultiLog in real time (as and when received).
Arduino Sensor	Y	UDP	Data continuously received on UDP port 8888. Processed by MultiLog in real time (as and when received).
Microsoft Visual Studio (Microsoft Visual Studio, 2014)	!	Text	When this logger runs, the log file is locked. As a result, MultiLog can only process it when the logger is stopped and the file is unlocked.
Inputlog (Leijten & Waes, 2005)	!	XML	The timestamp in the logger is a measure of the time since the logger was started not a general measure of time. As a result, MultiLog cannot determine chronological order unless the user/researcher specifies to use the last modified date from the file as the timestamp. Data is created across multiple log files. Only the file listed as the log file when the logger is added will be included initially. In order to include subsequent files, new loggers would have to be added via the Add Logger wizard.
Translog (Hansen, 1997)	!	XML	The timestamp in the logger is a measure of the time since the logger was started not a general measure of time. As a result, MultiLog cannot determine chronological order unless the user/researcher specifies to use the last modified date from the file as the timestamp. Data is created across multiple log files. Only the file listed as the log file when the logger is added will be included initially. In order to include subsequent files, new loggers would have to be added via the Add Logger wizard.
Mendeley Desktop log files (Mendeley)	!	Text	Data is created across multiple log files. Only the file listed as the log file when the logger is added will be included initially. In order to include subsequent files, new loggers would have to be added via the Add Logger wizard.
Kidlogger (Kidlogger)	!	XML/ HTML	Data is shown every minute for a period of one day. After this time, Kidlogger creates a new file and another logger would have to be added to MultiLog through the Add Logger wizard to reflect this.
WebQuilt (University of Washington)	!	Text	The timestamp in the logger is a measure of the time since the logger was started not a general measure of time. As a result, MultiLog cannot determine chronological order unless the user/researcher specifies to use the last modified date from the file as the timestamp.
Skype log files (Skype)	N	Text	Data is hashed/encrypted and MultiLog is unable to un-hash/decrypt it.
PersonalVibe (Microsoft Research)	N	Database	Data is held in a database and MultiLog does not handle databases.
SoftActivity Keylogger (Soft Activity)	N	Text	Data is hashed/encrypted and MultiLog is unable to un-hash/decrypt it.
Revealer Keylogger (Logixoft)	N	Text	Data is hashed/encrypted and MultiLog is unable to un-hash/decrypt it.

Table 2

A list of the three loggers (with three different types of output) along with example raw and MultiLog-parsed data.

Logger Name	***Raw Data***	***MultiLog-parsed Data***
Google Chrome	[9056:14968:0806/141932:VERBOSE1:resource_loader.cc(335)] OnResponseStarted: https://docs.google.com/offline/backgroundshell#ouid=ud3488c2d87270738	08/06/2015 14:19:32.000 OnResponseStarted: https://docs.google.com/offline/backgroundshell#ouid=ud3488c2d87270738
Microsoft PSR	<EachAction Time="17:36:43"> and <Description>User left click in "MultiLog.docx - Microsoft Word"</Description>	08/06/2015 17:36:43 User left click in "MultiLog.docx - Microsoft Word"
Arduino Light Sensor (As the logger is written by us, data is constructed in MultiLog's universal format in the pure logger code.)	100	08/06/2015 17:18:34 Data: 100

Data Integrity

Data integrity ensures that MultiLog is recording all of the required events and that it is recording all of the data associated with these events. The following method was used for validating data integrity: (1) MultiLog was configured to run one logger without filtering, see Table 3; (2) A series of interactions with the monitored application were performed; (3) The resulting events recorded by MultiLog were extracted from its database and the original log file generated by the logging software was copied; (4) MultiLog's output was transformed back into the format generated by the logging application by an additional piece of software; (5) The original log file and the transformed output from MultiLog were compared using a file difference checker[5].

We applied this methodology to a variety of loggers as illustrated in Table 3. For plain text output, all files were identical except for blank lines (which MultiLog automatically removes) and a handful of characters, such as the single quote, which were removed as they interfere with MultiLog's database.

Nine further trials were performed on XML loggers. This time, the raw log file was run through a third-party XML processor[6] to extract the timestamp and data and then formatted to match that produced by MultiLog. The difference checker was then used to check for differences. On observing the output, all files were identical as described in Table 3.

Three additional trials were conducted with the Arduino light sensor UDP logger. The functionality was altered to also write logs to the Arduino output window so a comparison between the raw data and the data received in MultiLog could be completed. Data over a one

[5] www.diffchecker.com
[6] www.xpathtester.com/xpath

minute period was checked and matched (MultiLog did not drop packets and timestamps were correct).

Table 3

Loggers used for output validation.

Test Logger	***Number of test repetitions***	***Output type (text/XML)***	***Average length of interaction***	***Average number of log lines produced***	***Results***
Google Chrome	3	Text	1 minute	7315	All lines identical except those which were omitted automatically by MultiLog because they did not contain a timestamp.
Mozilla Firefox	3	Text	1 minute	883	All lines identical.
Mozilla Thunderbird	3	Text	1 minute	353	All lines identical except those which were omitted automatically by MultiLog because they did not contain a timestamp, those that contained the single quote character which had to be removed to avoid database clashing and non-standard ASCII characters.
Microsoft PSR	3	XML	1 minute	80	Both files identical.
Inputlog	3	XML	1 minute	2192	Both files identical.
Translog	3	XML	1 minute	172	Both files identical.

Resource Consumption

Finally, MultiLog's impact on system performance (% processor time) was tested. Various tests were conducted using an average CPU tool[7]. Table 4 shows average CPU utilisation when running MultiLog with a number of different logging applications. These loggers were run firstly with user interaction with the PC (so events were generated) and secondly with no user interaction (when the PC was idle but the loggers were still running). All tests were run with a log polling interval of one minute. We found that, on average, 0.069% CPU was used when running MultiLog with one logger. This increased to 1.390% when running four. We also observed that increasing logger output directly impacts on performance.

[7] www.boray.se/software/averagecpu/

Table 4

Loggers used for performance validation. All tests were run for a period of 15 minutes apart from the Arduino light sensor test which was run for 30 minutes. The time intervals selected were relatively short because each test requires continuous interaction within that application, something current time constraints did not permit. 'Interaction' refers to the PC being utilised for the duration of the test. 'No interaction' means the PC was idle for the duration of the test. The tests performed with the Arduino light sensor were performed on a different machine due to hardware constraints. The no interaction figure provides a baseline from which the overheads can be calculated.

Configuration	***Average CPU usage (%)***	
	No interaction	***Interaction***
MultiLog + 1 logger (Microsoft PSR)	0.0169	0.0698
MultiLog + 2 loggers (Microsoft PSR and Mozilla Firefox)	0.0170	0.6267
MultiLog + 3 loggers (Microsoft PSR, Mozilla Firefox and Google Chrome)	0.1417	0.8925
MultiLog + 4 loggers (Microsoft PSR, Mozilla Firefox, Google Chrome and Mozilla Thunderbird)	0.0525	1.3902
MultiLog + 1 logger (Arduino light sensor) – high frequency data test (every 3 millisecond constant stream)	0.0018	6.8038

Initial Deployment

We also deployed MultiLog to a small number of users over a period of ten weeks to observe if it could run over extended periods of time to allow long-term studies to be conducted. MultiLog successfully ran for this period of time collecting and uploading data from anywhere between one and five loggers. This period of deployment allowed us to correct small interaction bugs we discovered such as the red 'X' wrongly closing the application rather than minimising it to system tray and various start up issues. We found that over the initial deployment period, web browser logs for example were producing a substantial amount of data, producing over 1000 lines in a matter of minutes. Much of this data was simple HTTP requests and not relevant to the research questions we are addressing and so it was decided to remove these for future user studies. When the appropriate loggers had been selected for the more detailed and extended deployment, we observed an average of 2000 log lines per hour with a database size of between 600 to 1000KB when user interaction is frequent. The loggers we tested with during initial deployment (with the average log lines per hour in brackets) were: Microsoft PSR (600), Drag-and-Drop (0-10), Window Switch (120), Clipboard (0-20) and Process start (450). Generally an interaction event is captured by more than one logger (and therefore more than one log line). For example, opening a new window would be captured by the Microsoft PSR (general logging), Window Switch (when the window came into focus) and Process start (when a new process was started) loggers.

Discussion

Approach

When evaluating the best approach to combining the output from existing logging applications, a trade-off between using log-merging software such as MultiLog has to be evaluated against whether it makes sense to develop a new logger from scratch. Under certain circumstances, such as when very specific data has to be collected that is not currently available from existing loggers, it would make sense (if time and finance permitted) to write a new logging application. However for general HCI user-based studies, log-merging software such as MultiLog facilitates the re-use of groups of existing logging applications.

Limitations

While MultiLog is robust and versatile, there are certain types of log output that it cannot handle. For example, log data that is split across multiple lines or lines without an accompanying timestamp. In order for MultiLog to handle plain text files and data received on TCP/UDP sockets, there must be one event per line/packet, each with an accompanying timestamp. Currently, only plain-text, TCP/UDP and XML-based loggers are supported. However, this could be extended in the future to handle other well-known data formats such as databases and encrypted data.

Certain low-level conditions also prevent or obstruct MultiLog from parsing logs. These include applications (such as Microsoft PSR) only generating their log files when the application is stopped, writing to log files in a non-linear manner, and locking the log file until the logging is terminated by the user. While work-arounds exist for these cases, all result in non-optimal presentation of log events to the user.

Command-line logging is typically not persistent across sessions—the application must always be started with the command line flags for logging to continue across sessions (this differs from system-wide loggers that automatically detect when applications of interest are started). In these settings, MultiLog will start an application once, but cannot prevent the user from starting the application without the flags, e.g. through shortcuts on task or menu bars. To achieve full coverage, researchers should alter the command line flags on all shortcuts available to the user. Not all logging applications employ this method of functioning (for example Microsoft Problem Steps Recorder (Microsoft Corporation), Window Switching, Drag-and-Drop, Clipboard and Process start (developed by MultiLog authors) and so researchers should check how individual loggers are started/stopped and adjust shortcuts accordingly.

Advice for Logger Developers

MultiLog has been released publically to allow researchers to take advantage of the logger control and log merging it facilitates. Developers wishing to make their logging application compatible with MultiLog should follow these guidelines: (1) The output is in either text or XML format, or writes data continuously to a TCP/UDP stream; (2) If a log file is used, data is written in a linear manner, appending new information to the end; (3) Each element of data has an accompanying timestamp and that this timestamp at least includes the year, month, day, hour, minute and second of when the event occurred; (4) The log file should not be locked, therefore preventing MultiLog from accessing it; (5) Ensure that if the logger is started with command-line arguments, it can also be successfully started without these for normal operation.

Combining Historical Logs

MultiLog's architecture does not require that the application itself was active during log generation. This means researchers can use MultiLog to standardise and combine historical logs, offline logs (such as from mobile devices), or data streams. This facilitates an array of new data analysis: from the process of revisiting multi-input experimental data-sets (e.g. combining eye-gaze data with system interactions) to aiding system administrators perform diagnostics by combining login, performance, and error logs to better understand failures.

Beyond Desktop Logging

While initially designed to reduce the technical barriers to multi-logger set up on desktop computers, MultiLog enables researchers to go beyond the realms of the desktop, by allowing them to easily integrate external data streams. This allows researchers to easily combine traditional application logging with environmental or on-body sensors to build a more comprehensive understanding of user interactions, contexts, and environments.

MultiLog's longitudinal data recording features mean that it can sit at the core of larger-scale deployments, especially when these are created with bespoke, non-standard components. For example, smart homes environments combine multi-manufacturer hardware and software to create a single system: MultiLog could, for example, be used to monitor and combine into one output: temperature, lighting levels, the temperature of the oven or fridge, which TV channel is currently selected, which web page a user is currently browsing on a PC and anything else within the house that produces a time stamped log output.

Conclusion

Event logging in desktop applications provides researchers with a tool to help understand how people interact with interfaces to facilitate improvements for future development. This article described MultiLog, a system which simultaneously controls, gathers, and combines the output from multiple existing research and commercial logging applications. MultiLog does not require technical expertise to configure or deploy. The MultiLog software is available from the website.

References

Actual Keylogger. (2014). Actual Keylogger Software for Windows - Free Download. from http://www.actualkeylogger.com/

Adobe. Adobe ARM Logging. from http://www.adobe.com/devnet-docs/acrobatetk/tools/PrefRef/Windows/Updater-Win.html

Adobe Customer Improvement Program. (2014). Adobe Customer Improvement Program. from http://www.adobe.com/limited/apip.html

Alexander, J., Cockburn, A., Fitchett, S., Gutwin, C., & Greenberg, S. (2009). *Revisiting read wear: analysis, design, and evaluation of a footprints scrollbar*. Paper presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Boston, MA, USA. http://dx.doi.org/10.1145/1518701.1518957

Alexander, J., Cockburn, A., & Lobb, R. (2008). AppMonitor: A tool for recording user actions in unmodified Windows applications. *Behavior Research Methods, 40*(2), 413-421. doi: 10.3758/BRM.40.2.413

CamStudio. (2014). CamStudio - Free Screen Recording Software. from http://camstudio.org/

Dostál, M., & Eichler, Z. (2011). *A research framework for performing user studies and rapid prototyping of intelligent user interfaces under the OpenOffice.org suite*. Paper presented at the Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems, Pisa, Italy. http://dx.doi.org/10.1145/1996461.1996511

Eklundh, K. S., & Kollberg, P. (2003). Emerging discourse structure: computer-assisted episode analysis as a window to global revision in university students' writing. *Journal of Pragmatics, 35*(6), 869-891. doi: 10.1016/S0378-2166(02)00123-6

Ezvid. (2014). Screen Recorder >> ezvid.com. from http://www.ezvid.com/screen-recorder

Google Chrome. (2014). Google Chrome Logging. from http://www.chromium.org/for-testers/enable-logging

Grabarnik, G., Salahshour, A., Subramanian, B., & Ma, S. (2004). *Generic adapter logging toolkit.* Paper presented at the Autonomic Computing, International Conference on.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explor. Newsl., 11*(1), 10-18. doi: 10.1145/1656274.1656278

Hansen, G. (1997). Success in translation. *Perspectives: Studies in Technology, 5*(2), 10.

Hutchings, D. R., Smith, G., Meyers, B., Czerwinski, M., & Robertson, G. (2004). *Display space usage and window management operation comparisons between single monitor and multiple monitor users*. Paper presented at the Proceedings of the working conference on Advanced visual interfaces, Gallipoli, Italy. http://dx.doi.org/10.1145/989863.989867

Internet Explorer. Internet Explorer brndlog.txt Log File. from http://blogs.msdn.com/b/askie/archive/2012/06/20/internet-explorer-maintenance-brndlog-txt-what-is-it-and-how-to-use-it-when-troubleshooting.aspx

Iváncsy, R., & Vajk, I. (2006). Frequent pattern mining in web log data. *Acta Polytechnica Hungarica, 3*(1), 77-90.

Juvina, I., & Oostendorp, H. v. (2004). *Predicting user preferences: from semantic to pragmatic metrics of Web navigation behavior*. Paper presented at the Proceedings of the conference on Dutch directions in HCI, Amsterdam, The Netherlands. http://dx.doi.org/10.1145/1005220.1005233

Kelly, D., & Sugimoto, C. R. (2013). A systematic review of interactive information retrieval evaluation studies, 1967–2006. *Journal of the American Society for Information Science and Technology, 64*(4), 745-770. doi: 10.1002/asi.22799

Kidlogger. Kidlogger. from http://kidlogger.net/

Kinkead, R. (1975). Typing Speed, Keying Rates, and Optimal Keyboard Layouts. *Proceedings of the Human Factors and Ergonomics Society, 19*(2), 159-161. doi: 10.1177/154193127501900203

Kukreja, U., Stevenson, W., & Ritter, F. (2006). RUI: Recording user input from interfaces under Windows and Mac OS X. *Behavior Research Methods, 38*(4), 656-659. doi: 10.3758/BF03193898

Ladakis, E., Koromilas, L., Vasiliadis, G., Polychronakis, M., & Ioannidis, S. (2013). *You Can Type, but You Can't Hide: A Stealthy GPU-based Keylogger*. Paper presented at the EuroSec'13, Prague, Czech Republic. http://www.cs.columbia.edu/~mikepo/papers/gpukeylogger.eurosec13.pdf

Leijten, M., & Waes, L. V. (2005). *Inputlog: A logging tool for the research of writing processes*. Paper presented at the Working Papers 2005011, University of Antwerp, Faculty of Applied Economics. http://www.researchgate.net/publication/4983063_Inputlog_A_logging_tool_for_the_research_of_writing_processes/file/60b7d519ce23bcc0db.pdf

Log Monitor. (2014). Log Monitor - CodeProject. from http://www.codeproject.com/Articles/19062/Log-Monitor

Logixoft. Revealer Keylogger. from http://www.softactivity.com/spy-software.asp

Math, M. M., Seetha, M., Kulkarni, U. P., & Yardi, A. R. (2009). Generic Log Adapters- A Step towards building a Parser Based Self Healable Autonomic System. *International Journal of Recent Trends in Engineering, 2*(3), 102-107.

Mendeley. Mendeley Logging. from http://support.mendeley.com/customer/portal/articles/227943-how-do-i-locate-mendeley-desktop-s-log-files-

Microsoft Corporation. Microsoft Problem Steps Recorder. from http://windows.microsoft.com/en-gb/windows7/how-do-i-use-problem-steps-recorder

Microsoft Corporation. (2003). What is the enable logging (troubleshooting) option? - Outlook. from http://office.microsoft.com/en-gb/outlook-help/what-is-the-enable-logging-troubleshooting-option-HA001174266.aspx

Microsoft Customer Experience Inprovement Program. (2009). Microsoft Customer Experience Improvement Program. from http://www.microsoft.com/products/ceip/en-US/default.mspx

Microsoft Distributed Transaction Coordinator. Microsoft Distributed Transaction Coordinator Log File. from http://technet.microsoft.com/en-us/library/cc759136%28v=ws.10%29.aspx

Microsoft Outlook. (2014). Microsoft Outlook Logging. from http://office.microsoft.com/en-gb/outlook-help/what-is-the-enable-logging-troubleshooting-option-HA001230421.aspx

Microsoft Research. PersonalVibe. from http://research.microsoft.com/en-us/downloads/0ea12e49-8b29-4930-b380-a5a00872d229/

Microsoft Visual Studio. (2014). /Log (devenv.exe). from http://msdn.microsoft.com/en-us//library/ms241272.aspx

Montgomery, A., & Faloutsos, C. (2001). Identifying Web Browsing Trends and Patterns *Internet Watch*.

Mozilla Firefox. Mozilla Firefox Logging. from https://developer.mozilla.org/en-US/docs/Mozilla/Debugging/HTTP_logging

Mozilla Thunderbird. Mozilla Thunderbird Logging. from https://wiki.mozilla.org/MailNews:Logging

Oliver, N., Smith, G., Thakkar, C., & Surendran, A. C. (2006). *SWISH: semantic analysis of window titles and switching history*. Paper presented at the Proceedings of the 11th international conference on Intelligent user interfaces, Sydney, Australia. http://dx.doi.org/10.1145/1111449.1111492

Peres, S. C., Tamborello, F. P. I., Fleetwood, M. D. I., Chung, P. I., & Paige-Smith, D. L. I. (2004). Keyboard Shortcut Usage: The Roles of Social Factors and Computer Experience. *Proceedings of the Human Factors and Ergonomics Society, 48*(5), 803-807. doi: 10.1177/154193120404800513

REFOG. (2014). REFOG Free Keylogger. from http://www.refog.com/free-keylogger.html

Rylstim Screen Recorder. (2014). Rylstim Screen Recorder - Reocrding On-Screen Manipulations to AVI File | Sketchman Studio. from http://www.sketchman-studio.com/rylstim-screen-recorder/

Shahzad, K. (2013). *Generic Log Analyser.* (Master of Science), Lulea University of Technology. Retrieved from https://pure.ltu.se/portal/files/42129705/LTU-EX-2013-42114123.pdf

Skype. Skype. from http://www.skype.com/

Soft Activity. SoftActivity Keylogger. from http://www.softactivity.com/spy-software.asp

Tak, S., & Cockburn, A. (2009). *Window Watcher: a visualisation tool for understanding windowing activities*. Paper presented at the Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7, Melbourne, Australia. http://link.springer.com/chapter/10.1007/978-3-642-03658-3_25

TeamViewer. (2014). TeamViewer - Free Remote Control, Remote Access & Online Meetings. from http://www.teamviewer.com/

TechSmith. (2015). Morae. from https://www.techsmith.com/morae.html

Tiny Log Monitor. (2014). Tiny Log Monitor. from http://www.tinylogmonitor.com/

Ubuntu. LinuxLogFiles - Community Help Wiki. from https://help.ubuntu.com/community/LinuxLogFiles

University of Washington. WebQuilt. from http://dub.washington.edu:2007/pubs/#webquilt

User Logger. User Logger. from http://chemware.co.nz/usrlog.htm

VMWare. VMWare Log Files. from http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1021806

Windows Update. Windows Update Log Files. from http://support.microsoft.com/kb/902093

Wintective. (2014). Wintective. from http://www.oocities.org/wintective/

Yavuz, A. A., & Peng, N. (2009, 7-11 Dec. 2009). *BAF: An Efficient Publicly Verifiable Secure Audit Logging Scheme for Distributed Systems.* Paper presented at the Computer Security Applications Conference, 2009. ACSAC '09. Annual.