CCDC DAC-TR-2020-101
December 2020

Recording Human Operator Data in Cyber Environments: User Activity Tracker (UAT) Technical Report

by Alex Poylisher, Matthew Witkowski, Vladislav D. Veksler, Blaine E. Hoffman, and Norbou Buchler

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so specified by other official documentation.

WARNING

Information and data contained in this document are based on the input available at the time of preparation.

TRADE NAMES

The use of trade names in this report does not constitute an official endorsement or approval of the use of such commercial hardware or software. The report may not be cited for purposes of advertisement.

CCDC DAC-TR-2020-101
December 2020

Recording Human Operator Data in Cyber Environments: User Activity Tracker (UAT) Technical Report

by Alex Poylisher and Matthew Witkowski
Perspecta Labs

Vladislav D. Veksler
DCS Corp.

Blaine E. Hoffman and Norbou Buchler
CCDC Data & Analysis Center

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED *(From - To)*
December 2020	Technical Report	November 2019–September 2020

4. TITLE AND SUBTITLE
Recording Human Operator Data in Cyber Environments: User Activity Tracker (UAT) Technical Report

5a. CONTRACT NUMBER
W911NF-14-D-0006

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Alex Poylisher, Matthew Witkowski, Vladislav D. Veksler, Blaine E. Hoffman, and Norbou Buchler

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Director
U.S. Army CCDC Data & Analysis Center
Human Systems Integration Division (FCDD-DAH-N)
6560 Surveillance Loop
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER

CCDC DAC-TR-2020-101

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
To conduct evaluations of individual and team operations in computing environments, sensors are need in appropriate places to collect data and provide it to researchers, experimenters, exercise leadership, and/or team leaders for analysis. Specifically, assessment of human performance requires data from the actions and behaviors of human operators in these settings, such as data from keyboard and mouse use and graphical-user-interface interactions including program launch, window manipulation, application focus, and commands executed. In addition to collecting this data, there is a need for a defined and known logging format to enable the creation and use of tools and interfaces that are capable of rapid parsing and visualizations of the data. In this report, we describe the User Activity Tracker (UAT) and the UAT Live tools. The former is a configurable tool for capturing human–computer-interaction data at the system level and capturing it in log files that can be preserved and analyzed both in real time and after the fact; the latter is a tool that capitalizes on UAT's logging format to provide automated analysis and data visualization.

15. SUBJECT TERMS
User Activity Tracker, UAT, automated data collection, human–computer interaction, HCI, cybersecurity, human behavior, logging

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Blaine E. Hoffman
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	34	19b. TELEPHONE NUMBER *(include area code)* (410) 278-5175

Table of Contents

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

This report describes the motivation, requirements, operating instructions, and design of the User Activity Tracker (UAT) developed at the U.S. Army Combat Capabilities Development Command (CCDC) Data & Analysis Center (DAC) in support of the Performance Assessment Suite for the Cyber Mission Force (PASS-CMF) project funded by the Office of Under Secretary of Defense for Research and Engineering (OUSD(R&E)). As the Applied Research and Engineering Partner for an Army cybersecurity collaborative research alliance bringing together Army organizations, industry, and academics, Perspecta Labs has been the lead developer of the tool over the course of the project. UAT has been created to serve the project goal of data-collection streams that enable analytics by providing a high-resolution, comprehensive, and timely recording of human–computer interaction (Figure 1). The latest version of UAT is 1.9.10 and has all of the features and capabilities described in this document. However development is ongoing, and future versions may have additional functionality or revised logging formats.

Figure 1. PASS-CMF's goals

The remainder of Section 1 gives the high-level motivation for UAT development and the set of high-level requirements. Section 2 contains the UAT user manual, including prerequisites, configuration, starting and stopping instructions, and current limitations for this tool. Section 3 describes the UAT architecture and implementation details.

1.1 Bottom Line Up Front

UAT is cross-platform software that runs a daemon process able to record user activity along with the respective state of the operating system (OS). UAT collects human–computer interaction events and stores these in compressible text format. Effectively, UAT records raw data from human operation of a system into a standardized data stream that can be used in analysis and exploration such as those used in performance assessment. The data logging enables detailed playback and quantitative analysis of user activity without the need to resort to large storage of images and video-screen recordings. For example, Figure 2 displays a sample screenshot of the type of analysis made possible by UAT (screenshot is of a separate UAT Live activity-analysis tool currently under development at CCDC DAC). In addition to a full Gantt chart of user activity (Figure 2, top left), UAT-collected data enables detailed playback (Figure

2, bottom left) of most human–computer interactions, including which windows are open and visible, where the windows are located, mouse location, clicks, and keyboard activity. Additionally, UAT makes it possible to construct a histogram of user interactions and a summary analysis of individual application use and between-application transition matrix (Figure 2, right). This level of detail of desktop user activity enables rich behavioral experiments and analysis, Human–Computer Interaction studies, and high-fidelity model and simulation development as well as Machine Learning-based user and user-activity classification.

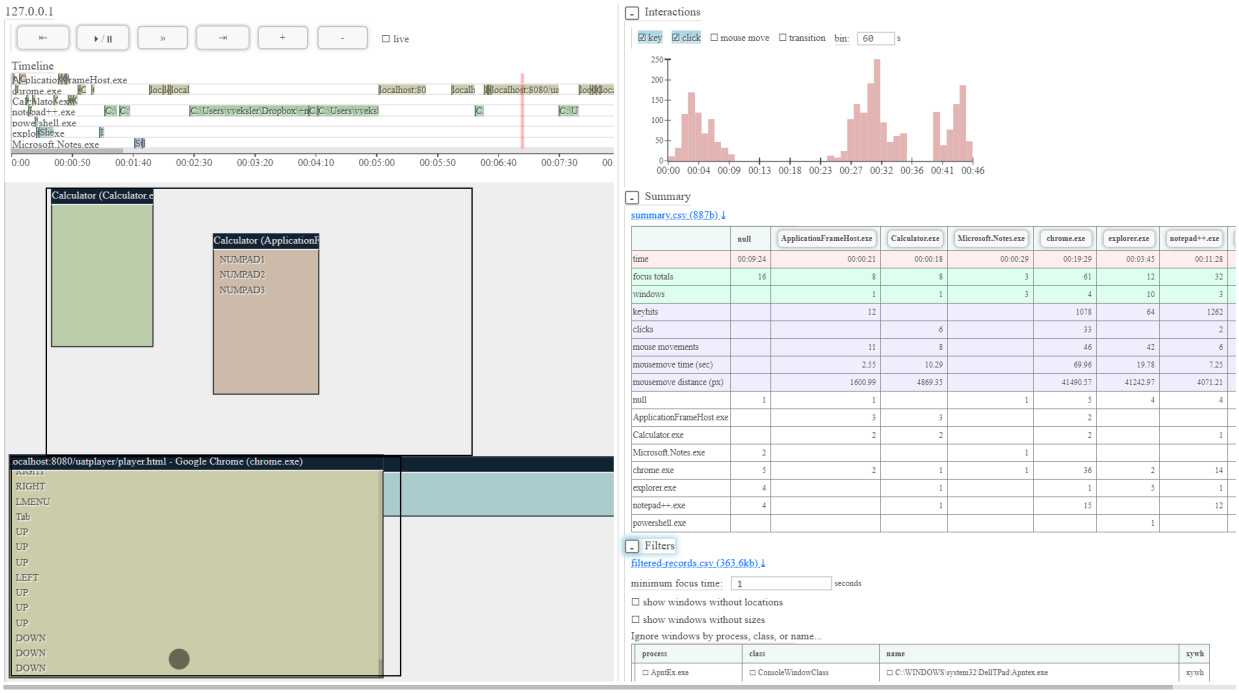

Figure 2. Screenshot of UAT Live data-analysis tool

1.2 Motivation for a New User Activity-Tracking Tool

The motivation for UAT originated from the impending end of support (in December 2020) of TechSmith's Morae usability testing tool, (TechSmith Corp., n.d.) widely used in the human-factors research community and initially considered for the PASS-CMF project. At the start of the project, PASS-CMF researchers formulated these key objectives of data collection for CMF-related user activity:

1. Support for *automated* processing of collected data to enable development of third-party analytics, post-experiment and near-real time.
2. *Experiment goal and application-agnostic* recording of *all* user activity, including actions that produce no visually observable effects, to provide ground truth for correlation with other cyber activity.

3. *Attribution* of user activity to cyber entities of interest (including machines, screens, and processes).
4. *Compact* data storage.
5. *Fine-grained* time resolution for the recorded events.
6. *Cross-platform* collection for major OSs, including Microsoft (MS) Windows and Linux variants.

Morae Recorder (the data-collection module of Morae) is only a partial match for the above requirements, as it is essentially a video-recording tool with *optional* recording of key presses and mouse activity, associated with processes, with the primary focus on manual event recognition by the human observers. While Morae Recorder's functionality can be extended via user-written plugins, this is not easy for a typical user.

The possible event types, apart from key presses and mouse clicks/movements, are predefined by the experimenter, so any automated post-processing is either limited to those, or requires expensive human effort (not serving Objectives 1, 2, and 3). Video recording is storage-heavy (not serving Objective 4). Event marking by human observers is subject to visibility, human error, and imprecise timing (not serving Objectives 2 and 5). Finally, the Morae Recorder is limited to MS Windows (not serving Objective 6).

The search for existing, publicly available alternative tools that would address all or most of these six objectives was unsuccessful, and PASS-CMF researchers decided to develop a new tool.

1.3 Use Case for UAT

PASS-CMF researchers constructed the first use case to drive UAT development around a fragment of a Cyber Protection Team engagement with elements of *Survey*, *Hunt* and *Protect* missions, taking place in an enterprise network where suspected adversarial Command and Control (C2)/exfiltration activity has been detected by an intrusion-detection system.

Figure 3 shows a fragment of the enterprise network topology, and both benign (green) and malicious (red) network flows. The dashed red line indicates the C2/exfiltration activity over Domain Name Service (DNS) tunnels. The DNS tunneling traffic is detected as suspicious by Security Onion sensors (Security Onion, n.d.), which is the starting point for the use case. The defenders operate from an administrative workstation (Ubuntu Linux/X11) and interact with multiple remote hosts via both graphical (remote desktop) and text-only (secure shell [SSH]) displays.

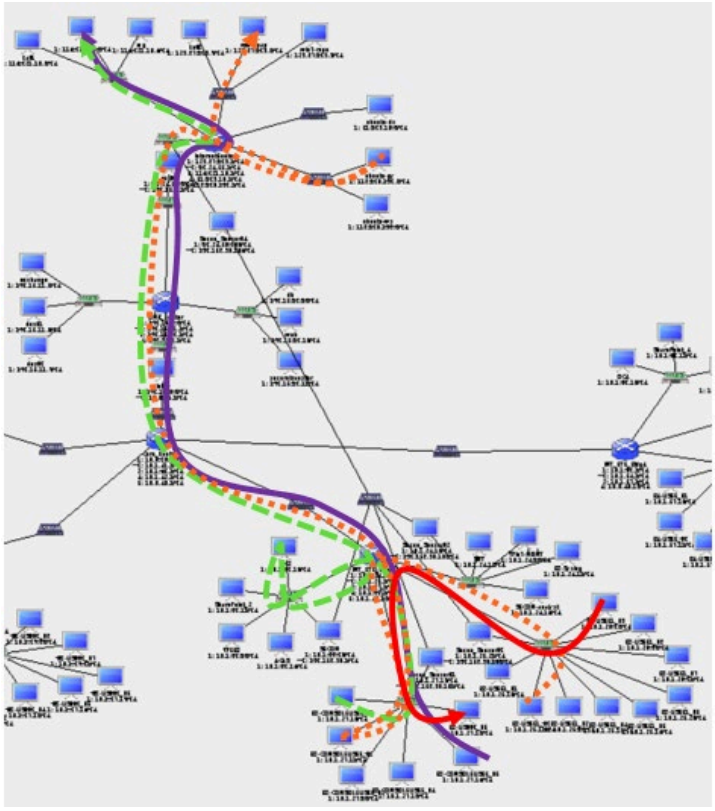

Figure 3. Benign and malicious activity in UAT use case

The defenders are alerted to the suspicious activity on the Security Onion console graphical user interface (GUI), observed via a remote desktop. Based on the alert analysis, the defenders decide to perform deep packet inspection on two firewalls (external and internal) and interact with the firewalls via remote Linux command line. The defenders diagnose the suspicious activity as exfiltration and decide to block the DNS tunnels used for exfiltration, via firewall rules, entered via the remote command line. A snapshot of the defender workspace during this interaction is shown in Figure 4.

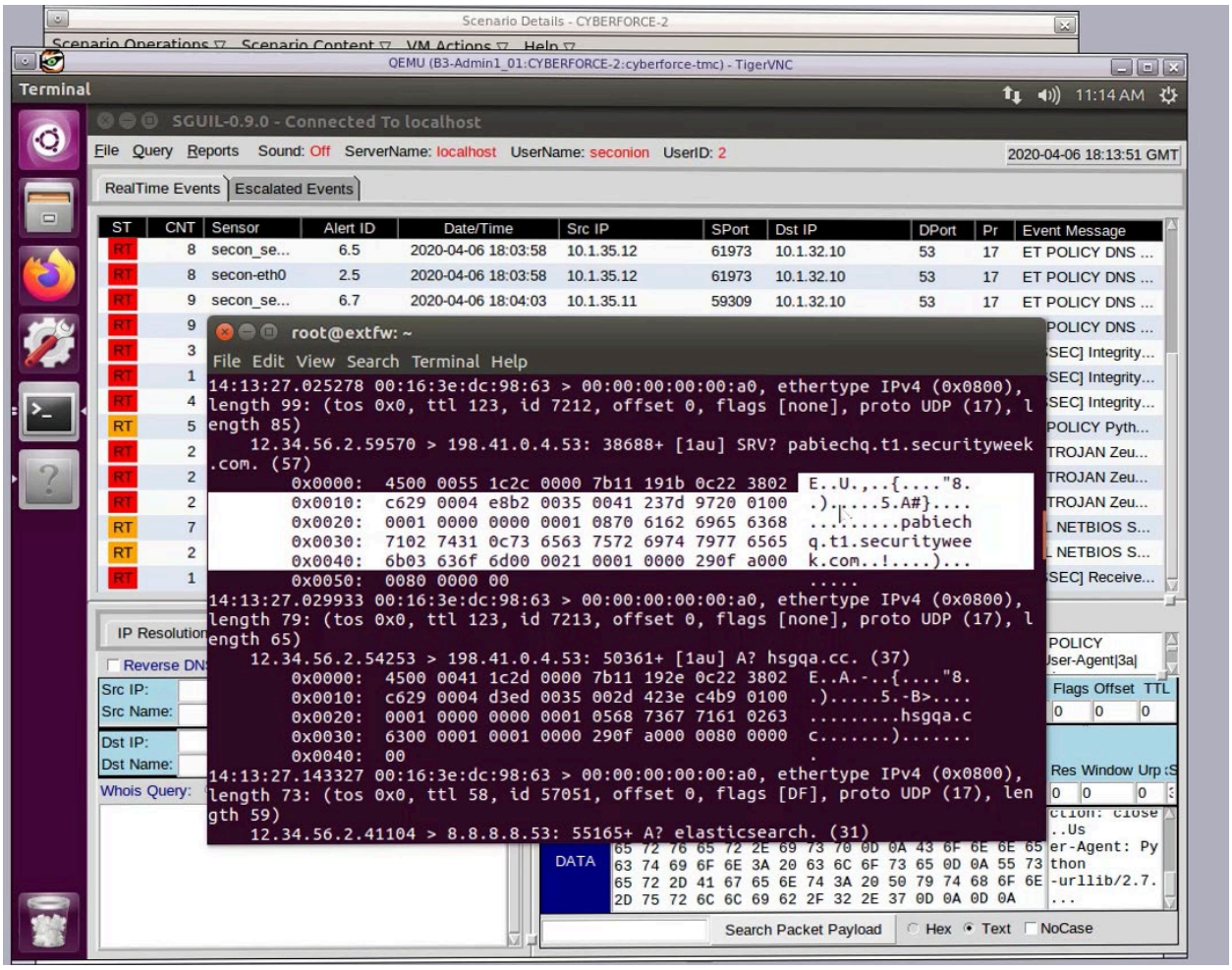

Figure 4. Screenshot of defender workspace in the use case

The defenders then identify the source host of the exfiltration traffic (an MS Windows client machine), and proceed to interact with that host (via MS Windows remote desktop) to find the malicious process responsible for the exfiltration and its executable file, to be quarantined for future examination.

The use case has been chosen to test realistic defender interaction via both GUI and command line with multiple machines (local desktop and remote desktops and terminals) and OSs (Linux and MS Windows variants).

1.4 PASS-CMF User Activity-Tracking Requirements

The following set of high-level requirements (Rs) grew organically while the tool was developed, driven by the six objectives defined in Section 1.2), feature sets of other tools, available instrumentation mechanisms, and user experience with the UAT prototype.

- **R1**: record user activity for graphical displays on a machine and all monitors (screens) for each display, including relative position of monitors (screens).
- **R2**: attribute recorded user activity to display-unique, window-system-native window identifiers.
- **R3**: attribute recorded user activity to machines and OS process identifiers.
- **R4**: record used activity time with millisecond resolution.
- **R5**: record window system events related to window creation and destruction.
- **R6**: record window system events related to window position, size, and visibility, including minimizing and maximizing.
- **R7**: record window system events related to window focus.
- **R8**: record window system events related to mouse movement, scrolling, and mouse button clicks, including event coordinates.
- **R9**: record window system events related to key presses, including key codes and character representation, if applicable.
- **R10**: record shell commands (strings entered by the user).
- **R11**: record shell input—shell command strings *after* shell aliasing and substitutions (i.e., the final form of the command about to be executed).
- **R12**: record shell commands and shell input for the shells that are not started from graphical terminals (e.g., started from local consoles or via remote login such as SSH).
- **R13**: record window system events related to screen resolution or position changes.
- **R14**: record window system events related to window visibility.
- **R15**: enable user-configurable choice of which events to record.
- **R16**: record user activity to local files or to a remote Hypertext Transport Protocol (HTTP) server.
- **R17**: enable user-configurable choice of the recording target (R15), and disable UAT startup if none is configured.
- **R18**: generate a local log file per display and rotate log files.
- **R19**: record with minimal impact on user experience and system resources.
- **R20**: support modern 64-bit MS Windows and Linux OSs, with an easy extension path for other OSs in the future.

2. USER MANUAL

2.1 Prerequisites

UAT collects information from low-level system events, and some prerequisites are required to make this possible on all operating systems.

2.1.1 Linux

1. Bash Shell Version 4.2 and up

2.1.2 MS Windows

1. Install .NET Framework 4.5
2. Install Windows Management Framework 5.0

2.2 Installation

The User Activity Tracker is bundled into a single zip file containing a configuration file, readme file, license file, Linux shared library, and an executable for each supported OS.

The following setup steps with administrative privileges are needed to run UAT. Most of the steps are required only for shell command logging.

2.2.1 Linux

1. Required for shell command logging—run the following command as root:

   ```
   echo 0 > /proc/sys/kernel/yama/ptrace_scope
   ```

2. Required for shell command logging—update the user's shell configuration file *and* profile configuration file to include the following lines:

   ```
   set –x
   set -v
   ```

3. Required for window-to-process mapping—update `/etc/ld.so.preload` to include the following line:

   ```
   <PATH_TO_LIBRARY>/libxcreatewindow.so
   ```

4. Required for shell command logging—comment out the following lines in `/etc/bash.bashrc`:

   ```
   #  enable bash completion in interactive shells
        #if ! shopt -oq posix; then
   #  if [ -f /usr/share/bash-completion/bash_completion ]; then
   #    . /usr/share/bash-completion/bash_completion
   ```

```
#  elif [ -f /etc/bash_completion ]; then
#    . /etc/bash_completion
#  fi
#fi
```

5. Required for shell command logging—move or rename the following files:

```
/usr/share/bash-completion/bash_completion
/etc/bash_completion
```

6. Required for shell command logging—append the following lines to `/etc/bash.bashrc`:

```
exec 5> /dev/null
export BASH_XTRACEFD=5
```

2.2.2 MS Windows

1. Required for shell command logging—set the group policies within Administrative Templates > Windows Components > Windows PowerShell (Dunwoody, 2016):
2. Set the group policies within Administrative Templates > Windows Components > Windows PowerShell
3. "Turn on PowerShell Script Block Logging": Enabled
4. "Turn on PowerShell Transcription": Enabled

2.3 Startup

UAT is distributed as an executable that can be started from the command line or a shell script, which could also be included in service configuration files (e.g., on MS Windows). It takes a single mandatory argument to specify the configuration file; paths can be added to the executable and config file names as needed:

```
UAT -c <config.json>
```

2.4 Configuration

The configuration file uses the standard JavaScript Object Notation schema (JSON Schema, n.d.) to adjust event tracking. Configuration allows for different attributes to be set for events, and to enable or disable them. The way these events are logged is also configurable. These different configurable components are split into two sections labeled "events" and "logging".

2.4.1 Event Configuration

The events configuration allows the user to enable/disable different events. The events section is split up into different categories titled "window", "mouse", and "keyboard". The rest of this

section describes each of the three categories. The enabled event configuration items will dictate most of the logging behavior of UAT. Section 2.4.8 has details on the format of data logged.

2.4.2 Window Events

The window section configures logging for the events related to GUI windows. Table 1 lists the currently supported events and their configurable attributes. Event-specific information logged for window events contains window title, process information, coordinates, and/or window size as appropriate.

Table 1. Currently Supported Events and Configurable Attributes—GUI

Event Type	Event Semantics	Attributes	Value
create	Creation of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
destroy	Destruction of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
focus_in	Focusing of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
focus_out	Focusing out of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
resize	Resizing of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
		pseudo_timeout	A float JSON value that represents the timeout value, in seconds, for the pseudo timer. See the Pseudo Events section for more info.
move	Movement of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
minimize	Minimizing of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
maximize	Maximizing of a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
restore	Restoring a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
resolution	Change in screen resolution or position.	enabled	A Boolean JSON value to enable or disable logging of the event.
draw	Drawing or undrawing of windows. This event can assist the UAT event consumer in detecting window visibility.	enabled	A Boolean JSON value to enable or disable logging of the event.

2.4.3 Mouse Events

The mouse section configures logging for the mouse-related events. Table 2 lists the currently supported events and their configurable attributes. The event-specific logging for a mouse event contains the screen coordinates of the action.

Table 2. Currently Supported Events and Configurable Attributes—Mouse

Event Type	Event Semantics	Attributes	Value
left_click	Releasing the left mouse button in a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
left_click_down	Pressing the left mouse button down in a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
right_click	Releasing the right mouse button in a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
right_click_down	Pressing the right mouse button down in a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
move	The movement of the mouse in a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.
		delta	A positive integer JSON value. This sets the distance in pixels, from the previous logged position, for the event to fire.
scroll	The scrolling of the mouse wheel in a GUI window.	enabled	A Boolean JSON value to enable or disable logging of the event.

2.4.4 Keyboard Events

The keyboard section configures logging for the keyboard events. Table 3 lists the currently supported event and its configurable attributes. Each keyboard event will include the character of the key and its integer value, as reported by the OS.

Table 3. Currently Supported Event and Configurable Attribute—Keyboard

Event Type	Event Semantics	Attributes	Value
keypress	Pressing a key on the keyboard.	enabled	A Boolean JSON value to enable or disable logging of the event.

2.4.5 Shell Events

The shell section contains logging configuration for shell-related events. UAT currently supports "command" for MS Windows `PowerShell`, and both "command" and "input" for `bash` on Linux variants. The "command" event captures the shell input as typed, before any unaliasing and variable expansion performed by the shell. The "input" event captures the shell input *after* unaliasing and variable substitution and captures the command as it will actually be executed. In both cases, the event-specific information logged will contain what each type captures as a string.

Please see important OS-specific configuration notes in Sections 2.2.1 and 2.2.2. Table 4 lists the events and their attributes.

Table 4. Events and Attributes for MS Windows and Linux

Event Type	Event Semantics	Attributes	Value
command	The command entered into a shell.	enabled	A Boolean JSON value to enable or disable logging of the event.
input	The expanded form of the command entered into a shell.	enabled	A Boolean JSON value to enable or disable logging of the event.

2.4.6 Pseudo Events

Pseudo Events are the synthetic events that UAT itself creates, and as such they do not correspond to an actual event generated by the window system. These events will be prepended with "P-".

Currently, UAT creates six pseudo-event types, "P-N", "P-WR", "P-WM", "P-WMAX", "P-WMIN", and "P-WRES", to immediately log the window size, location, and state for any windows that exist at the time UAT starts. These pseudo-events are generated without waiting for an actual resize event or move event. When a window creation event is detected, UAT starts a timer using the configurable "pseudo_timeout" value (see Window Events configuration in Section 2.4.2). UAT will then create a "P-WR", "P-WM", and a "P-WMAX", "P-WMIN", or "P-WRES" event when *all* three of the following conditions have been met for the window: the timer expires, a destroy event for that window has not been detected, and neither a window resize event nor a window move event for that window has been detected.

In order to assist in detecting visibility, the UAT creates the pseudo-event type, "P-DR". If the UAT detects a cloak, show, or hide event for a window, then it generates a "P-DR" for the triggering window. UAT then attempts to generate this event for each of the triggering window's child windows. A child window will only generate this event if their current draw state does not match their previous state.

2.4.7 Logging Configuration

UAT supports local and remote logging. At least one of the logging types must be enabled for UAT to start. These can be configured in the "local" and "remote" sections of the "logging" configuration.

2.4.8 Local Logging

Local logging creates a directory that houses the logs for each display and a system log directory used to track events that do not belong to a display. The first recorded event of each run of the UAT will create a new log with the following format: `(Configured Name)-Year-Month-Day-Hour-Minute`. The UAT has a built-in rotating logger, and it will automatically roll logs over to a new file when the log file reaches the size of 1 MB. The rollover index ("`.#`") is append to the end of the "`.tsv`" extension.

Table 5 lists the currently supported attributes for local logging.

Table 5. Currently Supported Attributes—Local Logging

Attribute	Value
enabled	A Boolean JSON value to enable or disable local logging.
filename	A string JSON value to name the logfile.
Directory	A string JSON value that is used as the path to store local log files.

Local Log File Format

A tab-separated values (`tsv`) file will be generated that contains all of the events that have occurred since UAT start. The first row of the log file will always contain the header record for the data. Every row after the header is an event that has been received. Each row will have the data for each of the following seven data elements, as applicable and when available:

1. The **time** the event occurred in milliseconds since the Epoch, in the time zone of the machine running UAT.
2. The **id** (identification) of the window that the event took place in. A window id of –1 means the window does not exist and the event belongs to a session without a display.
3. The **class** of the window that the event took place in, as supplied by the underlying window system.
4. The **name** of the window that the event took place in.
5. The event type.
6. The **event-specific information** in comma-separated values.
7. The **name** of the process the window belongs to.

Table 6 lists the event types' fields and associated semantics. Table 7 lists event-specific information.

Table 6. Event Types and Semantics

Event Type Field	Semantics
C	Left Mouse Click Up
CD	Left Mouse Click Down
RC	Right Mouse Click Up
RCD	Right Mouse Click Down
M	Mouse Move
S	Scroll
K	Keypress
N	Window Creation
P-N	Pseudo Window Creation
D	Window Destruction
WM	Window Move
WR	Window Resize
P-WR	Pseudo Window Resize
P-WM	Pseudo Window Move
WMIN	Window Minimize
WMAX	Window Maximize
WRES	Window Restore
P-WMIN	Pseudo Window Minimize
P-WMAX	Pseudo Window Maximize
P-WRES	Pseudo Window Restore
R	Screen Resolution/Position Change
FI	Focus In
FO	Focus Out
SC	Shell Command
SI	Shell Input
P-DR	Pseudo Window Draw

Table 7. Event-Specific Values: Type and Extra Parameters

Event Type	Extra Parameters
C	Coordinates relative to the window's top left corner in the format x,y.
CD	Coordinates relative to the window's top left corner in the format x,y.
RC	Coordinates relative to the window's top left corner in the format x,y.
RCD	Coordinates relative to the window's top left corner in the format x,y.
M	Coordinates relative to the window's top left corner in the format x,y.
S	Direction of scroll, and coordinates relative to the window's top left corner in the format x,y.
K	A positive integer keycode followed by the character representation.
N	A positive integer representing the process id that the window belongs to, and a Boolean representing if the window is drawn. If the pid belongs to a remote machine, it will be prepended with the machine name.
P-N	A positive integer representing the process id that the window belongs to, and a Boolean representing if the window is drawn. If the pid belongs to a remote machine, it will be prepended with the machine name.
WM	The window's new coordinates relative to the parent window's top left corner in the format x,y.
P-WM	The window's new coordinates relative to the parent window's top left corner in the format x,y.
WR	The window's new width and height represented by integers in the format: width,height.
P-WR	The window's new width and height represented by integers in the format: width, height.
R	An integer for the screen's width, integer for the screen's height, integer for the screen id, and integer x,y coordinates relative to the display's top left corner.
FI	The window's coordinates relative to the parent window's top left corner in the format x,y.
SC	An integer for the terminal process id, an integer for the shell process id, the name of the shell, and the entered command.
SI	An integer for the terminal process id, an integer for the shell process id, the name of the shell, and the expanded command.
P-DR	A Boolean representing if the window is drawn.

A sample local log fragment (captured on an MS Windows host) is shown in Figure 5.

```
1597684831398   44040193        Nautilus-desktop        nautilus-desktop                P-N     2028,False                              nautilus-desktop
1597684837162   46150614        Gnome-terminal          Terminal                        P-WRES                                          gnome-terminal-server
1597684841141   46155420        Gnome-terminal          Terminal                        P-WMIN                                          gnome-terminal-server
1597684843226   50335560        Gnome-terminal          Terminal                        P-WMAX                                          gnome-terminal-server
1597684921717   50331654        Gnome-terminal          user@user-VirtualBox: ~         C       424,249                                 gnome-terminal-server
1597684921643   50331654        Gnome-terminal          user@user-VirtualBox: ~         CD      424,249                                 gnome-terminal-server
1597684960085   44040202        Nautilus                Desktop                         RC      587,552                                 nautilus-desktop
1597684960006   44040202        Nautilus                Desktop                         RCD     587,552                                 nautilus-desktop
1597684961163   44040202        Nautilus                Desktop                         M       591,515                                 nautilus-desktop
1597684969113   50331906        Gnome-terminal          user@user-VirtualBox: ~         S       down,540,352
1597927650168   54525955        Firefox                 Mozilla Firefox                 K       25,w                                    MainThread
1597927650794   54525955        Firefox                 Mozilla Firefox                 K       25,w                                    MainThread
1597927651014   54525955        Firefox                 Mozilla Firefox                 K       25,w                                    MainThread
1597927651722   54525955        Firefox                 Mozilla Firefox                 K       60,.                                    MainThread
1597927652565   54525955        Firefox                 Mozilla Firefox                 K       42,g                                    MainThread
1597927652831   54525955        Firefox                 Mozilla Firefox                 K       32,o                                    MainThread
1597927652969   54525955        Firefox                 Mozilla Firefox                 K       32,o                                    MainThread
1597927653083   54525955        Firefox                 Mozilla Firefox                 K       42,g                                    MainThread
1597927653213   54525955        Firefox                 Mozilla Firefox                 K       46,l                                    MainThread
1597927653356   54525955        Firefox                 Mozilla Firefox                 K       26,e                                    MainThread
1597927653588   54525955        Firefox                 Mozilla Firefox                 K       60,.                                    MainThread
1597927653957   54525955        Firefox                 Mozilla Firefox                 K       54,c                                    MainThread
1597927654068   54525955        Firefox                 Mozilla Firefox                 K       32,o                                    MainThread
1597927654168   54525955        Firefox                 Mozilla Firefox                 K       58,m                                    MainThread
1597927655242   54525955        Firefox                 Mozilla Firefox                 K       36,Return                               MainThread
1597684914846   44040202        Nautilus                Desktop                         N       2780,True                               nautilus-desktop
1597685313542   50335560        Gnome-terminal          user@user-VirtualBox: ~         D                                               gnome-terminal-server
1597685312368   50335560        Gnome-terminal          user@user-VirtualBox: ~         WM      95,381                                  gnome-terminal-server
1597685013940   50335560        Gnome-terminal          Terminal                        WR      1853,920                                gnome-terminal-server
1597685302151   16777218                                gnome-screensaver               P-WM    -100,-100                               gnome-screensaver
1597685302152   16777218                                gnome-screensaver               P-WR    10,10                                   gnome-screensaver
1597685006752   50335560        Gnome-terminal          Terminal                        WMIN                                            gnome-terminal-server
1597685013913   50335560        Gnome-terminal          Terminal                        WMAX                                            gnome-terminal-server
1597685296536   704                                                                     R       1920,975,65,0,0
1597685315037   50331906        Gnome-terminal          user@user-VirtualBox: ~         FI      891,391
1597685315008   50331654        Gnome-terminal          user@user-VirtualBox: ~         FO                                              gnome-terminal-server
1597684999039   50331654        Gnome-terminal          user@user-VirtualBox: ~         SI      2979,4127,bash,ls --color=auto          gnome-terminal-server
1597684999040   50331654        Gnome-terminal          user@user-VirtualBox: ~         SC      2979,4127,bash,ls                       gnome-terminal-server
1597774085725   50339911                                                                P-DR    True
```

Figure 5. Sample local log showing fragments of event trace

2.4.9 Remote Logging

Remote logging bundles several events data together and sends a POST request to the address specified in the configuration. The first entry in the bundled data will always be the header data. When the application closes any remaining events that have not been logged will be bundled and sent. Table 8 shows the currently supported attributes.

Table 8. Currently Supported Attributes and Values—Remote Logging

Attribute	Value
Enabled	A Boolean JSON value to enable or disable remote logging.
Address	A string JSON value that represents the address of the HTTP to send the POST request to.
buffer_size	An integer JSON value that represents the number of events to be sent that are buffered before a POST request is sent.

2.4.10 Developer Logging

Developer logging is used to record UAT debugging information. Table 9 lists the currently supported attributes.

Table 9. Currently Supported Attributes and Values—Developer Logging

Attribute	Value
level	A string JSON value to alter the level of logging. The following values are supported: VERBOSE, TRACE, DEBUG, INFO, WARNING, ERROR, and CRITICAL.
format	A string JSON value to format the log messages.

2.5 Shutdown

UAT can be killed like any other process in the OS. The three most common ways to kill the process are by closing the terminal UAT was started in, pressing <CTRL-c> in that terminal, or sending a kill signal to the UAT process.

2.6 Resource Considerations

On computer systems with heavy user activity (e.g., multiuser shared systems), UAT local file logging (Section 2.4.8) can create substantial disk load. Possible workarounds include a) reducing the number of logged event types, b) reducing the resolution of recording for mouse movement, c) switching to remote logging, and d) allocating dedicated resources.

2.7 Limitations

The current UAT implementation has a few limitations, including the following:

1. The MS Windows version supports only one display, and the logs will always have a directory for "Display 0". A system logs directory will also be created, but is not populated with events.
2. Shell Input is not supported for the MS Windows version of UAT.
3. Shell Command and Shell Input are only supported by the Bash shell in the Linux version of UAT.
4. Shell Command is only supported by 32-bit PowerShell in the MS Windows version of UAT.
5. Entered commands (unexpanded) are echoed in Bash terminals.
6. In MS Windows, events are only tracked on OBJID_WINDOW objects (Microsoft Corporation, 2018).
7. The local log file rotation limit is not configurable.

3. ARCHITECTURE AND IMPLEMENTATION

This section describes the current UAT architecture and implementation.

3.1 Architecture

Figure 6 shows the UAT architecture, interfaces, and window-system/OS-instrumentation hooks, with the UAT code proper show in light blue. UAT is an event-driven daemon, reacting to the window system events naturally produced by the window systems, and events produced specifically by UAT instrumentation components (e.g., shell-related events).

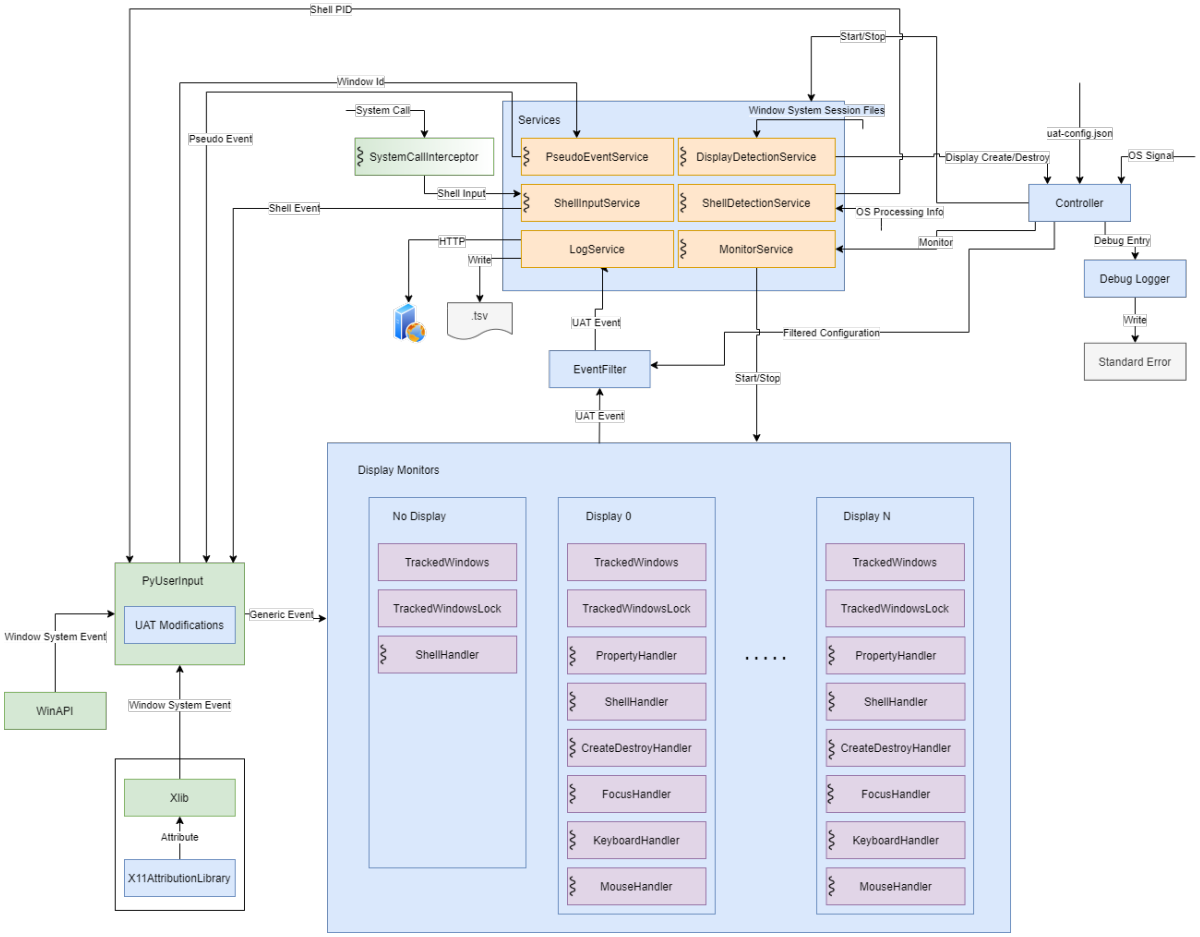

Figure 6. UAT architecture

At the top level, the **Controller** reads in the UAT configuration file, starts and stops Services, and outputs debugging information via the Debug Logger.

The **Display Detection Service** is responsible for start- and run-time detection of graphical display creation and destruction events. The **Shell Detection Service** is responsible for start- and run-time detection of shell startup and exit. The **Shell Input Detection Service** is

responsible for detecting shell input. The **Pseudo Event Service** is responsible for producing pseudo-events (see Section 2.4.6).

The **Monitor Service** is responsible for detecting and recording events per display, including window system events (window, keyboard, mouse, focus, and position/size) and shell-related events, taken from the Shell **Input Detection Service**, as the latter are recorded per display. Note the special "No Display" monitor, dedicated to processing shell events for the shells not started via a graphical terminal.

The **Log Service** writes the UAT-detected events to a local log file and to a remote HTTP server, if configured (Section 2.4.7).

The **X11 Attribution Library** is a Linux-specific component that ensures the X11 window system records OS-related information of interest to UAT (currently machine and process identifiers) in window attributes, for later extraction by Display Monitors.

Finally, the **Event Filter** ensures only the event types of interest to the user, as specified in the UAT configuration file, are recorded.

3.2 Implementation

This section describes the starting point of our implementation and also a discussion of select problems encountered during the UAT implementation along with the current solutions.

3.2.1 Starting Point

The majority of events of interest to UAT are naturally produced by the underlying window systems, and can be subscribed to by a program other than the actual window system's clients. Given the cross-platform requirements, we chose the cross-platform (MS Windows, Linux, MacOS) Python library `PyUserInput` (GitHub, 2013) as the starting point for UAT development due to its open-source nature, existing mouse and keyboard event-handling capabilities, extensibility, and a high-level, common implementation language. `PyUserInput` is in turn based on `PyXlib`, (Python XLib, n.d) which wraps the most common X11 library, `Xlib`.

Consequently, it was convenient to extend `PyUserInput` to subscribe to other window system events of interest to UAT *and* route non-window system events via `PyUserInput` for uniformity.

3.2.2 Select Problems and Current Solutions

This section documents several important implementation problems along with the current solutions.

3.2.3 X11 Display Detection and Authentication

With the variety of X11 server implementations and OS-specific configurations, there is no uniform way to detect the appearance and disappearance of X11 displays. We therefore address the *most common case* of X11 server sockets (Unix domain sockets used to communicate between the X11 server and its clients within one host), stored in the `/tmp/.X11-unix` directory.

The Display Detection Service, on startup and then periodically, reads this directory and for each socket file in the directory uses `lsof` and `ps` to obtain the command line of the process that owns the socket (the X11 server). The Display Detection Service then parses the X11 server command line to obtain the name of the server's `Xauthority` file, with the credentials needed to authenticate to that X11 server, in order for `PyUserInput` to be able to subscribe to the window system events on that server.

If a new socket file is detected and the corresponding X11 server process exists, the Display Detection Service declares the display created. If a known socket file disappears from the directory, or its corresponding X11 server process is not found, the Display Detection Service declares the display gone.

3.2.4 Shell Input Detection

The input of a command into a shell has no simple generic solution. There are multiple types of shells and many different versions of each shell. A simple approach such as LD_PRELOAD would not work unless we created a custom library for each type of shell and each version of the shell to ever exist. We therefore had to develop our own strategy that could reliably obtain the command for us.

The Shell Input Detection Service is started at UAT startup and is updated with process identifiers as new shells are detected by the UAT. In the Linux version of the Shell Input Detection Service, a separate thread is created for each process identifier so that they can safely be monitored by the `python-ptrace` library (Stinner, n.d.). Meanwhile, the Windows version attaches these to the `winappdgb` library (Vilas, n.d.) but does not create a thread for each process identifier.

On Linux, the Shell Input Detection Service is notified by the debugger in the python-ptrace library when a write system call is made. The service will then dictate if the write is from the shell by parsing for a specific format that contains a "+" at the beginning of the text parameter.

On Windows, the Input Detection Service is notified by the `winappdbg` library using system call hooks. We defined a "`pre_WriteFile`" hook, and our service receives all

"WriteFile" system calls that belong to the shells we attached. The input is then parsed, and the service filters out system calls that do not contain the user's input.

3.2.5 X11 Per-Window Process and Machine Attribution

There is no requirement for X11 clients to populate the process identifier window attribute _NET_WM_PID, yet the UAT requires such attribution for all windows.

We therefore adapted the libxcreatewindow library (GitHub, 2016) that applies a post-hook on XCreateWindow(), XCreateSimpleWindow() and XReparentWindow() Xlib library calls, via the LD_PRELOAD mechanism used transparently with all X11 clients on any system running UAT.

4. CONCLUSION AND FUTURE WORK

In this report, we covered the features and capabilities of the User Activity Tracker tool, an application that records various data at the system level to capture human behavior within computing environments. By taking advantage of OS integration, UAT can log not only the raw data from input devices (keyboard and mouse) but also more high-level data such as commands invoked in shells and task sequences in application and GUI interactions. Additionally, we covered the format and structure of UAT configuration settings that support flexible deployment and operation as well as the format of logged data recorded by UAT. The development and evolution of UAT has led to more formal definitions and structuring of the data it logs and exploration of its use in both human-driven and automated analyses.

The reader can use this report as a manual of UAT installation and configuration on a target platform and a manual for its use and operation. As of this writing UAT is in Version 1.9.10 and continuing to be improved and updated. As such, future work will include defining and outlining additional features and capabilities aligned with the goals of the project and UAT development as well as covering changes or additional details in UAT logging and operation. Alongside UAT, this document also serves as a reference for the log files themselves to support the creation of tools to parse and analyze UAT data, such as the UAT Live tool.

5. REFERENCES AND DOCUMENTS

GitHub, Inc. (2013, January 30). *PyUserInput Library* [Computer software]. https://github.com/PyUserInput/PyUserInput

GitHub, Inc. (2016, February 28). *ld-preload-xcreatewindow-net-wm-pid* [Computer software]. https://github.com/deepfire/ld-preload-xcreatewindow-net-wm-pid/blob/master/ld-preload-xcreatewindow.c

Dunwoody, M. (2016, February 11). *Threat research: greater visibility through PowerShell logging*. FireEye, Inc. https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html

JSON Schema. (n.d.). *The JSON Schema*. https://json-schema.org

Microsoft Corporation. (2018, May 31). *Microsoft Active Accessibility object identifiers*. https://docs.microsoft.com/en-us/windows/win32/winauto/object-identifiers

Python XLib developers community. (n.d.). *The python xlib library* [Computer software]. https://github.com/python-xlib/python-xlib

Security Onion. (n.d.). https://securityonion.net.

Stinner, V. (n.d.). *The python-ptrace library* [Computer software]. https://github.com/vstinner/python-ptrace

TechSmith Corporation. (n.d.) *Morae Tutorials*. https://www.techsmith.com/tutorial-morae.html.

Vilas, M. (n.d.). *The winappdbg library* [Computer software]. https://github.com/MarioVilas/winappdbg

Appendix A – List of Acronyms

C2	Command and Control
CCDC	U.S. Army Combat Capabilities Development Command
DAC	Data & Analysis Center
DNS	Domain Name Service
GUI	graphical user interface
HTTP	Hypertext Transport Protocol
JSON	JavaScript Object Notation
MS	Microsoft
OS	operating system
OUSD(R&E)	Office of Under Secretary of Defense for Research and Engineering
PASS-CMF	Performance Assessment Suite for the Cyber Mission Force
R	requirement
SSH	secure shell
UAT	User Activity Tracker

Appendix B – Distribution List

Organization

U.S. Army CCDC Data & Analysis Center
FCDD-DAS-LBG/T Resetar-Racine
6896 Mauchly Street
Aberdeen Proving Ground, Maryland 21005-5071

U.S. Army CCDC Data & Analysis Center
FCDD-DAH-N/B E Hoffman
FCDD-DAH-N/N Buchler
6560 Surveillance Loop
Aberdeen Proving Ground, MD

U.S. Army CCDC Army Research Laboratory
FCDD-RLD-DCI/Tech Library
2800 Powder Mill Rd.
Adelphi, MD 20783

Defense Technical Information Center
ATTN: DTIC-O
8725 John J. Kingman Rd.
Fort Belvoir, VA 22060-6218