The Power of Traceability

The ability to verify exactly what has happened (or is happening) inside applications at a particular moment has widespread usefulness in matters of consequence.

Paul H. Harkins
Harkins & Associates, Inc.
www.realtimeprogramaudit.com

An increasingly critical problem confronting corporate computing and programmer capability is the inability of computer programs to record and analyze the entire execution of the program and data in real time. This lack of traceability inhibits machine learning and autonomic (self-healing) computing possibilities, and prevents real-time and permanent analysis of exactly what executing program statements and data are actually processed. Therefore, as computers become millions of times more powerful and program environments become ever more complex and critical, the programmer is still using tools developed at the dawn of computing—tools like step debugging, reconstruction of events, and guessing--to attempt to <u>understand</u> what happened, rather than simply <u>observing</u> exactly what happened. Application 'observability' is a key and important trend in computing.

This paper presents a technique–-Electronic Program Auditing—and a software tool–The Real-Time Program Audit (RTPA)—designed to capture and record all of the executing source program statements in virtually any programming language, along with all of the data being processed and the moment-in-time of the statement execution without programmer intervention. This technique and unique tool provide a video-camera-like recording of the exact environment of the entire program execution (or selected conditions of the execution) as the program executes—capturing previously unrecorded critical information and thus enhancing the potential of machine learning, autonomic computing, and potentially artificial intelligence activities. This paper presents the RTPA technique and tool, some actual and possible applications, and evidence of the value of the approach.

1 Introduction

Today's computer hardware is incredibly powerful and becoming more powerful at an increasingly rapid pace [1]. A powerful new IBM supercomputer was announced in 2018 to have a peak performance speed of 200 Petaflops (200 quadrillion calculations per second) [2]. However, much of that processing power is wasted in typical corporate environments, as the latest software cannot come close to fully harnessing such processing power or fully utilizing the information generated.

Many common corporate computer software programming languages were introduced more than a decade ago (eg, Java in 1995); COBOL, introduced in 1959, has been in use for half a century and yet still supports many mission-critical applications. When many corporate languages were introduced, computers were millions of times less powerful than they are now, and software applications and environments were significantly more simple than the sophisticated applications, system architectures, and hardware now driving corporate computing.

Full *Electronic Program Auditing* is defined here as the complete real-time recording, auditing, and analysis of all executing computer source statements, all data processed, and the moment-in-time of each statement execution to electronic storage (normally disk). This is a comprehensive and powerful technique to enable today's software to take full advantage of hardware developments and application requirements. Electronic program auditing provides a video-camera-like, real-time permanent record of everything happening inside an application.

This ability to verify exactly what has happened (or is happening) inside an application at a particular moment has widespread usefulness in matters of consequence. Since full recording, auditing, and analysis of all program execution allows for a

permanent and unalterable record of the actual program execution, this technique could be used, for example, to: expose financial fraud; provide verification—without the possibility of alteration—of actual ballots cast in an election; and provide proof of transactions for Sarbanes-Oxley legislation [3] purposes. The importance of moment-to-moment verification of activity has already been recognized in other contexts, for example by the widespread use of video cameras to record and analyze virtually all public activity in order to provide a true, unaltered, real-time, and contemporaneous permanent record of significant activity. This emphasis on recorded and immutable verification has also carried over into computing, for example at transaction level with recent interest in blockchain technologies, and the more recent popularity of the term 'observability' [4].

Electronic program auditing was initially implemented by using a patented software invention—the Real-Time Program Audit (RTPA) [5]—to make existing programming languages capable of recording to disk the execution of all source statements, data processed, and the moment of time as a pre-processor to make the existing source statements smart enough to record their execution in real time, together with all data processed.

Figure 1 shows a partial source program in a free-format programming language that is an example for real-time program auditing. The executable source statements end with a semi-colon, similar to Java and C++ programming languages, and contain programming language commands that are audited based on the command, and program variables that are audited with the data processed by the executing statement. This source program reads a data file named DATAFILE and converts the program variable DATA to hexadecimal (two hexadecimal characters in a program variable **szhex** per input data character) using a function **cvthc**, then the program converts the hexadecimal variable back into the input variable character data with function **cvtch**. This sample source program is a modified source from www.rpgworld.com.

```
0001.00 H BNDDIR('QC2LE')
0002.00 H Dftactgrp(*NO)
0003.00  * Source program from www.rpgworld.com
0004.00 fdatafile  if   e            disk
0005.00
0006.00 D cvthc          PR                 extproc('cvthc')
0007.00 D  szRtnHexVar              65532A   OPTIONS(*VARSIZE)
0008.00 D  szSourceVal              32766A   CONST OPTIONS(*VARSIZE)
0009.00 D  nHexLen                    10I 0 VALUE
0010.00
0011.00 D cvtch          PR                 extproc('cvtch')
0012.00 D  szRtnCharVar             32766A   OPTIONS(*VARSIZE)
0013.00 D  szInputHex               65532A   CONST OPTIONS(*VARSIZE)
0014.00 D  nHexLen                    10I 0 VALUE
0015.00
0016.00 D szHex          S              40A
0017.00 D szChars        S              20A
0018.00 D Result         S              40A
0019.00
0020.00  /free
0021.00   // Source program example from www.rpgworld.com
0022.00         read datafile;
0023.00         dow not %eof(datafile);
0024.00   // convert character to hex
0025.00            cvthc(szHex : data : %len(data)*2);
0026.00         eval  result = szHex;
0027.00         if (szHex <> *blanks);
0028.00   // convert hex to character
0029.00    cvtch(szChars : szHex : %len(%TrimR(szHex)));
0030.00         eval  result = szChars;
0031.00         endif;
0032.00         read datafile;
0033.00         enddo;
0034.00         eval  *inlr = *on;
0035.00         return;
0036.00  /end-free
```

Figure 1. Source program for electronic program auditing.

Future programming languages could easily incorporate this real-time RTPA audit recording, auditing, and analysis capability directly into the language itself. And the powerful business intelligence (BI) tools of today could be linked directly

into these programming languages in order to provide real-time analysis and facilitate machine learning, autonomic computing, and eventually artificial intelligence processes.

The processing transactions—relatively small in number in comparison with the exponentially increasing power of the computer—can easily be fully program-audited, stored, and analyzed with electronic program auditing and RTPA without noticeable processing overhead.

The reality of corporate business computing is that much computing processes a quite small number of transactions— typically, several thousands of customer orders, invoices, inventory transactions, employee payroll processing, etc., in a typical program execution, rather than millions or billions of transactions. And the number of transactions processed each year in a typical application, such as data on the students in a university, gets only incrementally larger even if the business grows rapidly. Another characteristic of corporate business computing is that the corporate databases such as customer, inventory item, employee, and transaction activity are typically changed or updated by normal transaction processing, making rerunning or reconstruction of the exact same processing conditions in exactly the same processing environment (including time) impossible.

State of the Art. There are many program step-through debuggers, interactive program debuggers, and capture-replay techniques and tools that attempt to display the details of program statement execution. All of these techniques have critical limitations in that they stop the program execution (as in debuggers), allow program and data alteration, or capture only selected parts of the program execution, and require programmer intervention and knowledge of the program. These debuggers can provide, at best, a tiny part of the information needed for program analysis. All of these techniques fail to provide the exact processing conditions as the original processing program, as at least the time has changed and other programs may be altering or have altered the data files. However, there is no known software other than the Real-Time Program Audit (RTPA) that provides full electronic program recording, auditing, and analysis of all executing program statements and all data processed in real time as the program executes, without intervention and without program or data alteration, while providing a permanent record of the original exact environment of the program execution for real-time or future analysis.

Advantages of This Approach. The objective of recording, auditing, and analyzing the entire program statement execution, all data processed, and the moment-in-time in real time (as in video-camera recording) is simple, paradigm-shifting, powerful, and takes advantage of advancing computer capability and greatly reduced cost. This objective has been proven to be easily achievable in multiple programming languages, and it does not require any programmer or operational intervention at the time of program execution.

Real-time recording, auditing, and analysis of the original program execution, with the exact data and conditions at original program execution, provides a permanent record of exactly what happened, thereby eliminating the need for program rerunning, debugging, and guessing what happened. And the wealth of information recorded from the executed source statements, data, and moment-in-time provides information for real-time autonomic computing, business analysis and optimization, and business intelligence that is not possible without full electronic program auditing.

The programmer and operations need not even be aware that electronic program auditing is being performed by the smarter enabled source program, which, like a concealed video camera, provides a permanent and unobtrusive record of events. Electronic program auditing, like that provided by RTPA, audits and records in real time, but does not change or alter the program statements or data, as does most debugging software.

Growing trends such as machine learning, artificial intelligence, autonomic computing, and real-time analytics processes more generally would benefit by using Real-time Program Audit (RTPA) output, including data processed to view not only externally available data now written to disk by other programs, but also the information of the executing source statements and all data processed as the majority of program computation and data is never written to external files. Only electronic program auditing records and audits the actual program statement computations and data content of every statement executed and thus eliminates the opportunity for fraud by manipulating summarized data. For example, the classic fraud of altering summarized voting machine tabulations by adding 25 votes to one candidate's totals and subtracting 25 votes from a competing candidates total vote count would be exposed (and thus prevented) by electronic program auditing, which audits the computation and summarization of each and every voter. Additionally, electronic program auditing as in RTPA provides the capability for all logically related sub-programs in an application to be sequenced by the moment-in-time each statement was actually executed, regardless of the architecture or structure of the programs.

2 Real-Time Program Audit Technique

The Real-Time Program Audit (RTPA) [5] software overall technique is to enhance the capability of source programs in virtually all corporate (mainstream) programming languages to record the execution of source program statements (including auditing all source statements executing in real time, all data processed, and the moment-in-time) to an independent audit log or receiver. This independent audit log file is normally a disk file.

Thus, the RTPA auditing provides full electronic program auditing by enabling or enhancing the input source program to audit itself, even if the programming language does not provide recording and auditing capability.

2.1 Overview of the Approach

This technique consists of two phases: (1) audit-enabling or enhancing the original input source program and the resulting executable object program to make it smart enough to completely audit its execution, and (2) executing the audit-enabled object program in the normal program execution environment to produce the real-time audit file, audit spool file, and real-time business analysis. Future programming languages could easily provide this auditing capability as part of the standard programming language capability by providing for the automatic auditing and recording of all of the source statements and data executed similar to current disk file journaling.

2.2   Audit-Enabling the Original Input Source Program

Figure 2 shows the initial implementation of full electric program auditing in the Real-Time Program Audit (RTPA) patent [5] as a pre-processor that inputs source programs of audited programming languages, copies the source program to an enabled source program, and allows the enabled program source statements to be fully audited and recorded in real time during program execution together with all data processed, and outputs an expanded or enabled source program. Executable program object programs are compiled from the enabled source programs and have the capability of auditing themselves during program execution. The audit default is to record and audit all executing program statements and all data processed. However, extensive conditional auditing is also provided to allow focus and auditing on issues of interest.

RTPA provides real-time full recording and auditing capability of executing programs by examining every executable statement or command in the original source program, and adding the capability to log the source statement, the content of statement variables (the data processed by the executing statement), and the moment-in-time to an independent audit file when the statement is actually executed.

The basic process for RTPA auditing is quite similar for most programming languages, and has been defined in detail.

For the programming language being audited: define all valid language commands, or operation codes, and the auditing to be performed when processing a source statement using that command. For example, in COBOL the IF reserved word (command) would be defined, together with all other commands such as MULTIPLY, together with their RTPA auditing attributes.
1. For instance, the COBOL statement with a MULTIPLY command would be audited in real time in the enabled source program *after* the MULTIPLY command was executed, together with the data of program variables in the MULTIPLY source statement. The COBOL statement with an IF conditional command would be audited in real time in the enabled source program *before* the IF command was executed, together with the data of program variables in the IF source statement, in case the IF condition was not true, so the RTPA audit would show the contents of the IF statement variables that caused the IF condition statements not to be executed.
2. Define an audit recording file for logging all audit output. This file will be included in the audit-enabled source program.
3. Optionally define a printer spool file audit log.

For each original source program to be audit-enabled:
1. Copy the original input source program to an audit-enabled source program. The original input source program is not altered.
2. Compile the original source program and retrieve all detailed program information needed to fully audit the program and program variables.
3. Read the copied source program and audit every executable source statement so that it is recorded in the audit file when the statement is executed, together with the content of statement variables and the moment-in- time of execution.

4. Scan each executable source statement to identify the language command and program variables and special conditions such as complex statement groups such as IF, AND conditional statements.

5. Audit enable each executable source statement including the statement itself, program variables data processed, and the moment-in-time when the statement is executed, together with control information such as the program name.

6. Audit complex statements (commands) such Execute Format (EXFMT) which is a WRITE then READ as a group with the elapsed time between them (user wait time), and audit complex conditional statements like IF, AND as a group.

7. Audit comment statements to provide better program comprehension.

8. Time stamp the moment-in-time of the executing statement to the audit file.

9. Output control and analysis information including the program ID and statements group level to the audit file, so that real-time auditing and analysis may be easily accomplished.

10. Output the file definition for the independent audit file, which will contain the full electronic program audit of the executing program statements, and optionally output the printer spool audit file definition.

11. Optionally selectively audit based on audit criteria or conditions or audit output limits.

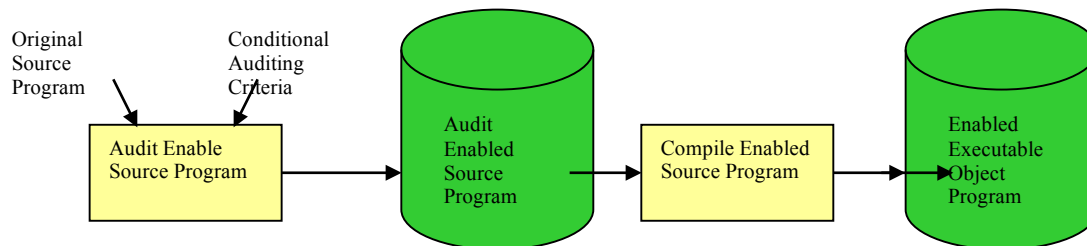12. Compile the audit-enabled source program and create an audit-enabled executable object program.

Figure 2. Real-Time Program Audit (RTPA) pre-processor overview.

Figure 3 shows a partial listing of the audit-enabled source program statements with RTPA inserted statement to record and audit every executing source program statement, all data processed, and the moment-in-time the statement was executed. The subroutines or subprocedures actually accomplishing the output to the RTPA audit file and RTPA audit analysis (spool) file, such as EXSR Z$INIT to initialize the audit file and analysis spool file, vary by the programming language being audited. However, these routines always audit and output the source statements being executed, the data content of program variables in the executing statement, date, and the moment-in-time the statement is executed, together with control information such as the program name.

Further information about this input example source program, the RTPA audit-enabled source program, and the RTPA audit file and audit analysis output for this CVTTOHEX example program can be found on www.realtimeprogramaudit.com website.

The audit inserted source statement EXSR Z$INIT; opens the audit file and initializes the inserted audit routines in the audit-enabled source program (and in the executable audit-enabled object program) and outputs audit control information including the program name, job number, and moment-in-time of the program activation. The inserted Z$INIT source statement also outputs optional spool file audit report headings which include the moment of program initiation. The inserted audit source statement EXSR Z$GENS; outputs the source statement to the audit file.

Note: The RTPA source statements inserted into the audit-enabled source program to accomplish full electronic program auditing could easily be inserted by the language compiler (processor) as part of a smarter audit-enabled programming language by the language vendor. A similar technique for accomplishing disk record journaling before-and-after images has long been available as part of the vendor-provided operating system software.

The elapsed time of complex source statements like EXFMT (Execute Format) is computed including the elapsed time it takes for a terminal user to respond to a WRITE then READ statement and the elapsed time it takes for external program calls to other programs or sub-procedures. The exact time a statement is executed and the elapsed time of the statement is invaluable in program traceability, analysis and optimization.

For example, the expanded audit capable source program contains the following source statements to read the DATA file input disk record and to audit the read statement and the data being read.

```
66          read datafile;
67                    Z$SRC# =  2    ;
68                    EXSR     Z$GENS;
69                    EXCEPT   Z$00002;
70                    IF       NOT %EOF;
71                    EXCEPT   Z$00002D;
72                    ENDIF;
```

The input source statement **read datafile;** reads the next record from the disk file.

The inserted audit statements below output the audit of the read datafile; statement when it is executed with the moment-in-time of the statement execution.

```
67                    Z$SRC# =  2    ;
68                    EXSR     Z$GENS;
69                    EXCEPT   Z$00002;
```

The inserted audit statements below output the audit of the actual data if a disk record was read.

```
70                    IF       NOT %EOF;
71                    EXCEPT   Z$00002D;
72                    ENDIF;
```

```
     61 C                    EXSR      Z$INIT
     62  /free
     63   // Source program example from www.rpgworld.com
     64               Z$SRC# =  1    ;
     65               EXSR      Z$GENS;
     66        read datafile;
     67               Z$SRC# =  2    ;
     68               EXSR      Z$GENS;
     69               EXCEPT    Z$00002;
     70               IF        NOT %EOF;
     71               EXCEPT    Z$00002D;
     72               ENDIF;
     73               Z$SRC# =  3    ;
     74               EXSR      Z$GENS;
     75        dow not %eof(datafile);
     76   // convert character to hex
     77               Z$SRC# =  4    ;
     78               EXSR      Z$GENS;
     79        cvthc(szHex : data : %len(data)*2);
     80               Z$SRC# =  5    ;
     81               EXSR      Z$GENS;
     82               EXCEPT    Z$00005;
     83        eval  result = szHex;
     84               Z$SRC# =  6    ;
     85               EXSR      Z$GENS;
     86               EXCEPT    Z$00006;
     87               Z$SRC# =  7    ;
     88               EXSR      Z$GENS;
     89               EXCEPT    Z$00007;
     90        if (szHex <> *blanks);
     91   // convert hex to character
     92               Z$SRC# =  8    ;
     93               EXSR      Z$GENS;
     94        cvtch(szChars : szHex : %len(%TrimR(szHex)));
     95               Z$SRC# =  9    ;
     96               EXSR      Z$GENS;
     97               EXCEPT    Z$00009;
     98        eval  result = szChars;
     99               Z$SRC# =  10   ;
    100               EXSR      Z$GENS;
    101               EXCEPT    Z$00010;
    102        endif;
    103               Z$SRC# =  11   ;
    104               EXSR      Z$GENS;
    105        read datafile;
    106               Z$SRC# =  12   ;
    107               EXSR      Z$GENS;
    108               EXCEPT    Z$00012;
    109               IF        NOT %EOF;
    110               EXCEPT    Z$00012D;
    111               ENDIF;
    112        enddo;
    113               Z$SRC# =  13   ;
    114               EXSR      Z$GENS;
    115        eval  *inlr = *on;
    116               Z$SRC# =  14   ;
    117               EXSR      Z$GENS;
    118               EXCEPT    Z$00014;
    119               Z$SRC# =  15   ;
    120               EXSR      Z$GENS;
    121        return;
```

Figure 3. Source program with RTPA auditing.

2.3   Execution of the Audit-Enabled Program

Figure 4 shows the processing of the audit-enabled executable Object Program with the output of Real-Time Program Audit output, including data processed and real-time audit analysis, business intelligence tools, and business analytics and optimization (BAO) tools. The Real-Time Program Audit Query analysis tool provides extensive real-time analysis of the audit output file, including the capability to view executing subprograms, source program statements, and program variable contents

of different programming languages in a logical application in moment-of-time sequence as the computer actually executes them.
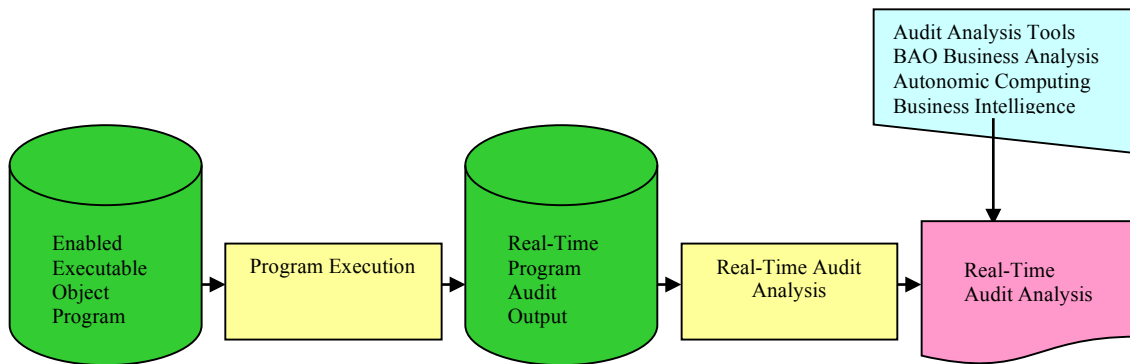

Figure 4. Real-Time Program Audit (RTPA) audit output and analysis.

Figure 5 shows the RTPA real-time audit output file from execution of the audit-enabled program. The audit output file is a *permanent contemporaneous record of the original program execution.*

The electronic program audit of the executed program contains the source program statements as they were actually executed, and shows the compile listing statement numbers as in the original compile listing of the original source program, together with the data executed by the program and the moment-in-time of the statement execution.

The audit output of the read statement below shows the **read** statement and the data read by the read statement with the moment-in-time of the read statement execution The DATA- line is the variable DATA followed by the contents of the variable DATA.

```
    22          read datafile;                                                   19.46.37.452
                                                                       File-      00002 Key-
    DATA-1234567890ABCDEFGHIJ
```

The **dow not (eof)** statement (do while not end-of-file) and the source statements 23 and 24 are executed if the read statement read an input record into the variable DATA.

The comment statement at line 24 **// convert character to hex** is audited each time the comment statement is executed. Auditing executing comment statements is very important in programmer comprehension of what actually happened in the executing program.

The **cvthc** statement converts the variable DATA into hexadecimal format (two characters per input character) and places the output in the variable szHex.

```
    23          dow not %eof(datafile);
    24   // convert character to hex
    25          cvthc(szHex : data : %len(data)*2);                              B01
                    F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
                          1234567890ABCDEFGHIJ
                                      1234567890ABCDEFGHIJ
```

```
Program-CVTTOHEX    Convert Character to Hex Data in PF DATAFILE     Obj Lib: Z$AUDITE    Initiated:  6/10/09
         CVTTOHEX    CVTTOHEX
 Job: 334040              User Profile: PHH           Source Type: RPGLE     Y   Source File/Library: QRPGLESRC
 Line#                                                                                     Do# SrcId
   21    // Source program example from www.rpgworld.com
   22         read datafile;
                                                                          File-     00002 Key-
 DATA-1234567890ABCDEFGHIJ
   23         dow not %eof(datafile);
   24    // convert character to hex
   25         cvthc(szHex : data : %len(data)*2);                                        B01
                  F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
                         1234567890ABCDEFGHIJ
                                     1234567890ABCDEFGHIJ
   26         eval  result = szHex;
                  F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
                         F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
   27         if (szHex <> *blanks);                                                      01
                  F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
   28    // convert hex to character                                                      01
   29         cvtch(szChars : szHex : %len(%TrimR(szHex)));                               B02
                  1234567890ABCDEFGHIJ
                         F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
                                         F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
   30         eval  result = szChars;
                  1234567890ABCDEFGHIJ
                         1234567890ABCDEFGHIJ
   31         endif;                                                                      02
   32         read datafile;                                                              02
                                                                          File-DATAFILE   Key-
 DATA-KLMNOPQRSTUVWXYZ!@#
   24    // convert character to hex
   25         cvthc(szHex : data : %len(data)*2);                                        B01
                  D2D3D4D5D6D7D8D9E2E3E4E5E6E7E8E95A7C7B40
                         KLMNOPQRSTUVWXYZ!@#
                                     KLMNOPQRSTUVWXYZ!@#
   26         eval  result = szHex;
                  D2D3D4D5D6D7D8D9E2E3E4E5E6E7E8E95A7C7B40
                         D2D3D4D5D6D7D8D9E2E3E4E5E6E7E8E95A7C7B40
   27         if (szHex <> *blanks);                                                      01
                  D2D3D4D5D6D7D8D9E2E3E4E5E6E7E8E95A7C7B40
   28    // convert hex to character                                                      01
 (truncated output)
   34         eval  *inlr = *on;                                                          01
                  1
   35         return;                                                                     E01
```

Figure 5. RTPA audit file output from Audit-Enabled Executable Object Program.

Real-time program audit analysis with the RTPA Query tool, conventional business intelligence (BI) tools, business analysis and optimization (BAO) tools, machine learning, and autonomic computing could access this real-time RTPA audit file and spool file and achieve real-time analysis while the program is executing. Audited source statements and source statement data including the contents of program variables provide unprecedented capability to analyze detailed program information never before available for analysis and action.

Figure 6 shows the right side of the audit output containing control information including the conditional do level, source statement number, change date, and time stamp of the moment-in-time of the execution of each statement.

Figure 5 and Figure 6 are actually the left-hand and right-hand sections of the same 198 position audit output, and together illustrate the executing source statement, all data processed by the executing statement, the moment in time of the executing statement, the date and time, and control information including the program do level and the program id. The audit output can ne queried and analyzed immediately in real-time or analyzed later with a variety of existing analytic and business intelligence tools.

```
o Hex Data in PF DATAFILE        Obj Lib: Z$AUDITE   Initiated:  6/10/09 19.46.37.452    PAGE   1    RPGLE     Y

 PHH            Source Type: RPGLE      Y   Source File/Library: QRPGLESRC Z$AUDIT      CVTTOHEX   JOB 297949
                                                              Do# SrcId ChgDat   Seq#    Time
 www.rpgworld.com                                                    080612   2100 19.46.37.452
                                                                            19.46.37.452
                                        File-     00002 Key-


                                                                            19.46.37.452
                                                                080611   2200 19.46.37.452
%len(data)*2);                                          B01      080612   2300 19.46.37.452
8F9F0C1C2C3C4C5C6C7C8C9D1
890ABCDEFGHIJ
     1234567890ABCDEFGHIJ
                                                                080611   2400 19.46.37.452
F0C1C2C3C4C5C6C7C8C9D1
5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
                                                          01    080611   2500 19.46.37.452
C1C2C3C4C5C6C7C8C9D1
                                                          01    080612   2600 19.46.37.452
 : %len(%TrimR(szHex)));                                 B02     080611   2700 19.46.37.452
FGHIJ
3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
          F1F2F3F4F5F6F7F8F9F0C1C2C3C4C5C6C7C8C9D1
                                                                080611   2800 19.46.37.452
IJ
0ABCDEFGHIJ
                                                          02    080611   2900 19.46.37.452
                                                          02    080612   3000 19.46.37.452
                                        File-DATAFILE   Key-

                                                                080611   2200 19.46.37.456
%len(data)*2);                                          B01      080612   2300 19.46.37.456
9E2E3E4E5E6E7E8E95A7C7B40
RSTUVWXYZ!@#
     KLMNOPQRSTUVWXYZ!@#
                                                                080611   2400 19.46.37.456
E3E4E5E6E7E8E95A7C7B40
 (output truncated)
                                                          E02   080611   3100 19.46.37.456
                                                          01    080611   3200 19.46.37.456

                                                          E01   080611   3300 19.46.37.456
```

Figure 6. RTPA audit output right side of report with the moment-in-time of the executing statement.

2.4 Additional Technical Details

This section discusses several of the most relevant technical details of this approach that relate to its implementation into programming languages with varying capabilities to produce audit files (normally disk) and audit spool (printer) files and to allow business analysis and optimization tools (BAO) to be utilized for autonomic computing.

2.4.1 Assumptions
First, this Real-Time Program Audit (RTPA) electronic program audit technique should work for virtually all current corporate programming languages that have the capability to produce an audit file as a disk file output, and optionally a printer spool file output in the programming language. Programming languages without the capability of producing an audit file as a disk file output or a printer spool file, such as the IBM Control Language Program (CLP), can currently be audited with this technique using calls to sub-procedures or subprograms to a language such as COBOL that does provide auditing capability. Future implementation of real-time program auditing in the vendor distributed language would provide this auditing capability in a manner transparent to the programmer for both the executing source statements and for all the data processed with an audit file similar to the current vendor supplied disk journal file for before and after images of disk record updates, additions, and deletes.

Second, the audit enabling statements and techniques currently inserted into the copied, expanded, and enabled source program should not alter the normal execution of the source (enabled object) program. Again, this could be easily incorporated by the

vendor of the programming language directly into the language without requiring a separate preprocessor step to audit-enable the source program. Then, audit processing would produce a hugely beneficial and transparent result (similar to video-camera recording) of virtually all corporate computer processing.

Third, the program execution processing time of the audit-enabled statements, including creation of the real-time audit file, incurs very little additional processing time, as is evidenced in the audit output in Figure 5. Real-life execution of audit-enabled program processing has shown that audit processing requires minimal additional time or overhead for typical corporate application environments for which the original processing time for an application is only a few processing seconds. Additionally, audit recording and logging may be turned on or off based on conditions or the size of the audit file. The emergence of solid-state disk drives (SSDs), which have throughput of some twenty times that of conventional disk drives, ensures that RTPA auditing overhead will be insignificant in the future. That, coupled with the immense and largely unused processing capacity, presents a huge opportunity for pervasive implementation of full electronic program auditing.

3 Possible Applications of This Technique

Seven possible applications of this real-time program audit technique are discussed:
(1) full electronic program auditing of computer program execution;
(2) debugging and analysis of computer program execution;
(3) business analysis and optimization (BAO) of executing program statements and data;
(4) verification of information being valued as a business asset [6];
(5) real-time machine learning, autonomic computing, and AI using executing program statements and data;
(6) vote-count tabulation verification; and
(7) security and authentication verification [7].

3.1 Electronic Program Auditing of Program Execution

This technique, together with the dramatic increase in power and decrease in cost of computing, increases a paradigm shift in how computing is accomplished and analyzed. Full electronic program recording and auditing should become the normal method of corporate computing, making obsolete many of the ancient and inefficient techniques in use today and greatly multiplying the understanding and capability of programmers. This full electronic program audit information would be available as a permanent and unalterable record of the details of exactly how the program executed and exactly how the data were processed.

3.2 Debugging and Analysis of Program Execution

Step-program debuggers and interactive program debuggers were developed literally at the dawn of computing in the 1950s, when programs were small and relatively simple, and only one program at a time was processed by the computer. These debuggers are also slow and labor intensive, requiring programmer intervention by manually stopping the program and they require program knowledge to be used. And, the debuggers can never exactly re-create the original environment, since at least the time and computing environment will have changed by the time the debugging begins. Debuggers also require speculation—guessing about what might have happened or where the problem might be. Real-time electronic program auditing totally eliminates the need for step debuggers and eliminates the need for guessing what happened during program execution.

3.3 Business Analysis and Optimization (BAO) of Executing Program

Much critical business-analysis information processed in executing programs, including exactly how the information is computed statement by statement, is simply not recorded, and therefore is not available for business analysis and business intelligence tools.

For example, end-user screen error messages are typically not recorded, but electronic program auditing records all of the end-user error messages and all the data displayed to the end user and entered by the end user. This error-message

recording, auditing, and analysis allows needed end-user training to be customized to the end user and pinpoints how a series of end-user errors can result in error or abnormal processing.

For successful error detection and correction, knowing the content of program variables that are currently not recorded can be crucial. Consider the need for comparison of the estimated carton weight of packed products to the actual packing-line real-time scale carton weight, including the estimated weight of each product. This comparison is critical to successful detection of cartons not matching the estimated carton weight. Real-time recording of all program variable contents at the moment of time of statement execution allows real-time analysis of products in this error carton with products in other error cartons—signaling a potential error in the estimated weight of a particular product, without the need to open the error carton. This allows for the kind of rapid and sure resolution of problems, like errors in estimated weights of a product that may be changed and corrected at any time during the day that would not be possible without this detail information—and it enables the prevention of future packing-weight errors for cartons of the product. Current applications not providing this program-auditing capability simply cannot provide this real-time analysis and correction capability, and result in massive and time consuming error detection and correction procedures and missed shipments.

3.4 Verification of Information Being Valued as a Business Asset

The increasing importance of data as a business and financial asset asset is now well established [6]. However, the importance of data as an asset taken together with the ongoing potential risk of corporate financial frauds, requires that summarized data must be verifiable with drilldown not to other summarized data (as is normally the output of computer processing), but to the actual program statements and program variable contents used to create the information. Program auditing is the only technique that will accomplish this ultimate drilldown to the original and unalterable audit record created during program execution.

3.5 Real-time Machine Learning, Autonomic Computing, and AI Using Executing Program Statements and Data

In order for machine learning, autonomic computing, and AI processes to be effective and self-managed, it is obvious that the executing program must have access to, and utilize, real-time detail information that is often computed in the executing program, such as error messages, and is not available externally outside the program, and never output to disk as part of the application. The RTPA audit file, and spool file do provide all of this internally created program information and make this information immediately available for a full range of business intelligence tools and business analysis and optimization (BAO) tools. These analysis tools can be utilized by the executing program in real time to provide autonomic computing, as in the case of end-user screen-error message analysis and action.

3.6 Vote-Count Tabulation Verification

Real-time program auditing could be utilized in vote-count tabulation verification to ensure that all votes counted by a computer program, such as mark sensed voting sheets or electronic voting, are verified with a *permanent and unalterable* audit record of *actual votes* as each vote is accumulated. This auditing technique could be very easily implemented, and, if mandated, would eliminate actual and much attempted electronic-voting machine and online voting fraud at every level of voting.

3.7 Internet Security and Authentication Verification

The emergence of the Internet and remote processing technologies such as cloud computing, has greatly increased the susceptibility of computer programs to hacking, unauthorized use, and the resulting damage by this hacking of computers at every level of business and government, as the computing is external to the corporate location. Cybersecurity is a top priority for every business organization [7].

Electronic program auditing and analysis provides unalterable and potentially permanent and accessible proof of exactly what was processed at the program executing statement level; what data were processed at the executing statement level; the moment-in-time of the executing statement level; and the user ID or program name and object information responsible for the processing activity. This, the ultimate drilldown for security and authentication, provides real-time audit information for

autonomic computing security and authentication, analysis, and corrective action during the execution of the program and later for detailed review and analysis.

4 The Tool: Real-Time Program Audit (RTPA)

The Real-Time Program Audit (RTPA) is a system or method of full electronic program auditing [5]. The specific implementation of full electronic program auditing is undertaken with the RTPA software tool. The first RTPA software tool has been written in the RPG programming language and is implemented for several programming languages, including RPG, COBOL, and the IBM Control Program Language (CLP). As previously mentioned (see Section 2.4.1), RTPA development in other programming languages is possible. The original source program is not altered in any way, and the copy of the source program is audit-enabled without programmer intervention, so that full auditing is output to an audit file when the audit-enabled program is executed.

During execution of the audit-enabled executable object program, the program is intended to execute the original source statements exactly as in the original source program. The inserted RTPA source statements are executed in line with the executing original source program statements, merely and only to record the executing source program statement, the content of program variables in the executing original source statement, and the moment-in-time of execution of the original source program statement. The executable object program of some programming languages such as RPG, COBOL, and CLP have a program template that includes the source program statement and program variable definitions to allow for step through or interactive debugging and program variable analysis. Implementation of full electronic program auditing by software vendors in the operating system compilers would utilize this available information for audit recording, rather than the auditing preprocessor having to save this information in the audit-enabled executable object program. Thus, software vendor-supplied full electronic program auditing capable compiles would provide the capability to output the executing source statement to the audit file when the source statement was executed and the moment–in-time of execution as part of the operating system.

5 Empirical Evaluation

The feasibility and value of full electronic program auditing in the corporate programming environment was first tested by manually inserting the RTPA additional source statements to a copy of a large production original source program. When the resulting expanded source program was compiled to an expanded object program and executed in a normal corporate processing environment, the resulting real-time audit file and audit spool file was extremely useful in the real-time analysis of the program. This manually inserted program auditing was performed on several important source programs and greatly simplified the development and maintenance and support of these programs.

No step-debugging or program rerunning was required to solve programming issues, since the audit file and audit spool file had already produced the information needed for analysis.

Once the concept and value of full electronic program auditing was tested, the manual auditing process was generalized and automated as in today's RTPA software implementations, and the automated auditing implementation was generalized for other programming languages.

It has been demonstrated that—through its simplification of programming and analysis and its provision of real-time program data, internal to the program, that had never before been available—full electronic program auditing (as in the RTPA implementations), represents a paradigm shift in programming and computing.

Though the need for structured programming is widely acknowledged and useful to programmers viewing the static program source structure, full electronic program auditing diminishes this need, since all the executing statements from many routines or sub-procedures in a logical application written in several programming languages can be sequenced in the moment-of-time the statement was executed, regardless of program organization. The emergence and astonishing capability of online search engines to almost instantly gather together the widely dispersed information to satisfy a search request is an example of how a logical request for related data created at different times does not rely on the data being stored next to or near each other.

Electronic Program auditing could be utilized in virtually all corporate processing environments and in virtually all corporate applications, including all financial applications. Virtually all of these corporate applications are processed in a multiprocessing environment, where multiple independent jobs are running simultaneously in the computer, and where each program is running in a single processing thread.

Indeed, thousands of original source programs have been audit-enabled using the Real-Time Program Audit (RTPA) software, and then audited using the resulting expanded audit-enabled executable object program in normal processing environments. Typically, an original source program of several thousand source statements is copied, audit-enabled into an audit-enabled source program as in Figure 2 and then compiled into an audit-enabled executable object program as in Figure 4, in less than five elapsed seconds on a small corporate computer.

6 Related Work

The author knows of no other implementation of full electronic program auditing software, as in the Real-Time Program Audit (RTPA) software, in the corporate programming environment. However, the import of 'observability' is rapidly growing in the computing community [4].

The many debuggers and capture-and-replay tools provide useful techniques for program review and analysis, but they sometimes allow program and data modification, and they do not provide a full and complete real-time and permanent audit record of the entire program execution of all executing source statements and all data processed as output of the executed programs.

New technical products and processes that utilize and implement the immense benefits of full electronic program auditing are needed, particularly those that capitalize on the largely unused capacity of current technology to fully implement the rapidly developing initiatives of machine learning, AI, and autonomic computing. There are numerous application possibilities to enhance and maximize data as an asset, real-time business analysis and optimization (BAO) technology, as well as to facilitate software development practices such as DevOps.

7 Conclusion

This paper has described a technique for implementing full electronic program auditing and autonomic computing with the Real-Time Program Audit (RTPA) software. Of the many new applications made possible with electronic program auditing, seven have been suggested. Finally, the initial implementations of full electronic program auditing in several

corporate programming languages (RPG, COBOL, and CL) were presented, as well as the effective and straightforward process to implement electronic program auditing in other corporate programming language.

One significant conclusion is that the entire world of corporate computing can be made more productive and simple by harnessing the power of the computer to fully electronic program-audit, record, and analyze the execution of all computer programs.

Productivity tools that are now conventional—such as the Microsoft Word Spelling and Grammar tool and the security and virus tools that run in the background of Word document processing—are key examples of the significant benefits provided by smarter processing with productivity tools. Full electronic program auditing, as in the Real-time Program Audit (RTPA), provides smarter processing that is revolutionary and more comprehensive—indeed, it will work industry-wide.

Mandated standards and regulations in many industries have evolved over the years to provide needed safety and security for all who use cars, such as the automobile industry requirements for airbags and collision protection standards, and health care strict safety standards. These mandated standards have allowed much greater safe use of automobiles worldwide, while greatly reducing risks and accidents. As demand for 'observability' increases, it is now feasible that corporate computing and corporate and government institutions develop and implement effective and powerful real-time safety, security and auditing standards and capability for all corporate computing through tools such as full electronic program auditing.

References

[1] U.S. Department of Energy, Oak Ridge National Library (2018): 'ORNL Launches Summit Supercomputer'.

https://www.ornl.gov/news/ornl-launches-summit-supercomputer

[2] Informa USA, Inc. Data Center Knowledge. (2018) "IBM, Nvidia Build 'World's Fastest Supercomputer' for US Government".

https://www.datacenterknowledge.com/supercomputers/ibm-nvidia-build-world-s-fastest-supercomputer-us-government

[3] U.S. Government Publishing Office. (2002) 'Public Law 107-204 July 30, 2002'.

www.govinfo.gov/content/pkg/STATUTE-116/pdf/STATUTE-116-Pg745.pdf

[4] Sridharan, Cindy. (2017) 'Monitoring and Observability'.

https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c

[5] United States Patent and Trademark Office. Patent for Real-Time Program Audit (RTPA).

http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=b6e598167639a3b7ef6dba403805b4cd1977f95afa0be5ebb17a3997ca7255e52Fnetahtmlb6e598167639a3b7ef6dba403805b4cd1977f95afa0be5ebb17a3997ca7255e52FPTOb6e598167639a3b7ef6dba403805b4cd1977f95afa0be5ebb17a3997ca7255e52Fsrchnum.htm&r=1&f=G&l=50&s1=6,775,827.PN.&OS=PN/6,775,827&RS=PN/6,775,827

[6] Gartner, Inc. (2017) 'Gartner Says Within Five Years, Organizations Will Be Valued on Their Information Portfolios'.

https://www.gartner.com/en/newsroom/press-releases/2017-02-08-gartner-says-within-five-years-organizations-will-be-valued-on-their-information-portfolios

[7] Gartner, Inc. (2018) 'Gartner Top 6 Security and Risk Management Trends for 2018'.

https://www.gartner.com/smarterwithgartner/gartner-top-5-security-and-risk-management-trends/