ARTICLE:

An approach to the judicial evaluation of evidence from computers and computer systems

By **Michael Jackson**

Introduction

In England & Wales, computer-derived evidence presented in a criminal case is assessed by the members of a jury, and in civil proceedings by a judge. Since 1999, when an earlier statutory provision was repealed, the common law presumption of *proper functioning of machines* has applied to computers. Thus, by default, the evidence is treated as reliable: if the evidence is contested, the burden of rebutting the presumption falls entirely on the party who denies its reliability.

This short paper is intended as a contribution to addressing some of the difficulties that can arise in such cases. It can be seen as an addition to the discussion in two articles published last year (Peter Bernard Ladkin, Bev Littlewood, Harold Thimbleby and Martyn Thomas CBE, 'The Law Commission presumption concerning the dependability of computer evidence', 17 *Digital Evidence and Electronic Signature Law Review* (2020) 1 – 14, and Peter Bernard Ladkin, 'Robustness of software', 17 *Digital Evidence and Electronic Signature Law Review* (2020) 15 – 24). Like those papers, it draws heavily on the example of the Post Office Horizon system that was at issue in *Bates v Post Office Ltd Rev 1* [2019] EWHC 3408 (QB).

The Horizon system supports the accounting for each Post Office branch for each Trading Period. At the start of each Trading Period the physical cash held at the branch, and the stock of such items as stamps and Lottery scratch cards, should agree with the data held in the Horizon system. During the Trading Period the sub-postmaster or sub-postmistress (SPM) enters into the Horizon system all sales (for example, of stamps) and payments (for example, of pension entitlements) made to customers. At the end of the Trading Period, the account for the branch as shown by Horizon should again agree with a physical audit of the cash and stock held in the branch. In the event of a discrepancy the Post Office might accuse the SPM of carelessness, mismanagement, or criminality. The claimants in *Bates v Post Office* maintained that the Post Office had wrongly held them responsible for discrepancies that were in fact due to defects in the Horizon system, and that they had been unjustly convicted in their earlier trials at first instance.

The task of rebutting, or defending, a presumption that a computer system has functioned properly can be approached in various ways. For example, reliability can be assessed from the system's operational history, or from the operational histories of other systems that share major characteristics. The present paper proposes an approach aimed at addressing the concern expressed by Fraser J at paragraph [826] of his judgment in *Bates v Post Office*. Postulating a hypothetical 'bug X' in the system, he wrote: 'Analysis and resolution of the correct and true situation of the branch accounts between the Post Office and the SPM for the trading period in question does not depend upon whether, in all the other millions of branch accounts, there was no such incidence of bug X. The correct analytical approach in my judgment is to consider the branch activity for that branch for that period; consider the evidence both for and against (1) the existence of bug X and (2) the likely cause of the discrepancy, bearing in mind both the burden and standard of proof; make findings on the cause of the discrepancy; and then apply those findings.'

Long Transactions

The approach proposed in this paper does not usually start with a specific postulated 'bug X'. Rather, it starts from the concept of a *long transaction*. Suppose, for example, that a sub-postmaster (SPM) sells a sheet of stamps to a customer for £240 and enters the sale into the Horizon system. At the end of the period, the Horizon accounts for that Sub-Post Office should show the SPM's stock of stamps reduced by the sheet sold, and cash in hand increased by £240. In the most usual meaning of the term *transaction*, the event in which the sheet of stamps is exchanged for the money would constitute the transaction of interest. In the approach proposed here, by the word *transaction* we will mean what may be called the *long transaction*, beginning with that exchange of stamps for money, and completed only when the appropriate updates have been made to the branch account data in the Horizon system and have been agreed between the SPM and the Post Office. Only when this completion has been achieved can the long transaction be ready, so to speak, to be archived in the branch history. Assuming no complicating factor that defers the completion because something must be carried over to the next Trading Period, the time necessary for a long transaction can be up to four or five weeks, these being the possible lengths of a branch Trading Period. A complicating factor that required, for example, suspending resolution of an apparent discrepancy to the next period, would extend the long transaction by that further period. This concept of a long transaction is particularly suited to the Horizon branch accounting system because it is the typical arena of dispute in particular cases of large discrepancy: often, the amount of the discrepancy is the amount of an identified large sale or payment to a customer, or a large transfer of funds or stock. The core activity of an SPM is conducting many long transactions simultaneously.

The general principle of the approach proposed in this paper is to investigate the processing by the Horizon system of a completed long transaction that is the subject of dispute. The purpose of the investigation is to identify the most likely cause of the discrepancy. The investigation is a journey following the transaction processing through the system. To describe the terrain in which this journey takes place, the following sections present a simplified view of computers and computer systems.

Elements of a system

A *computer* is a physical machine that can execute *programs* and has *ports* at which it communicates with *devices* to receive and emit data and control signals. A keyboard, for example, is an input device; a screen is an output device, and so is a printer. A disk drive, typically holding structured representations of data, is also a device: it can both accept and provide data, for the same or different computers; it stores data and allows it to be accessed later on multiple occasions. Different devices can interact in different ways with *persons* in the world and with *things*. For example, in the Horizon system, a sub-postmaster is a person, and a Fujitsu software engineer is a person. In an information system, a *thing* is likely to be a document of some kind. In a cyber-physical setting (for example, in a passenger lift system), a thing might be a lift car, a call button, or a lift door. Things and persons in the world can interact with one another in various ways. Computers can interact only with devices, and only through devices with things and persons. How computers interact with devices is determined by programs, and programs themselves are placed in computers by interactions, ultimately with humans.

A *system* is an assemblage of *computers*, *devices*, *persons* and *things* connected as nodes in a network through which information may flow. As a trivial example, a pocket calculator used in a small shop by an employee P (a *person*) can be seen as the core of a computer system: it has a keyboard input device, a computer, and a screen display. P, preparing an invoice (a *thing*) for a handwritten order (another *thing*) received from customer C (another *person*), and wanting to calculate 17% of 435, presses the keys ‘1’, ‘7’, ‘%’, ‘*’, ‘4’, ‘3’, ‘5’, and finally ‘=’ on the keyboard. This data sequence flows from the keyboard to the computer; the computer executes its program to calculate the result; the computer then transmits the calculated result to the screen; the screen displays the result 73.95; P reads from the screen and writes the result by hand in a line item of the invoice.

Tracking computer system evidence

The preceding paragraph sketched a *long transaction,* from receipt of a customer order to the production of an invoice. Customer C might dispute the invoice line written by P, denying the *end-to-end* correctness of the transaction: that is, denying the correctness of the relationship between the initial order and the final invoice. The shopkeeper might respond that the system in operation has proved reliable over many years of such transactions: so the invoice is almost certainly correct. The system sketch given in the preceding paragraphs provides a street map that is essential for resolving the dispute: the disputed transaction can be *tracked* end-to-end along every step of its transmission path through the system.

Tracking the customer’s transaction along its processing path through the system invites an obvious comparison with the familiar recorded tracking of retail deliveries. The step-by-step tracking records of a retail purchase delivery have two purposes. First, the customer, given access to the records, can maintain confidence that the delivery is progressing. (This purpose has some relevance to our system concerns, discussed later in this paper.) Second, the cause of an apparent end-to-end failure can be more easily located and more effectively analysed, both by the disappointed purchaser and by the delivery's manager. The manager, of course, represents the owner and responsible operator of the delivery system.

In the trivial calculator system, as in retail deliveries, the possible causes of error are at the nodes of the network and at the connections between them. Customer C may have written wrong order details; P may have misread the order; P may have erred in keying in the required calculation; the keyboard may have transmitted the wrong data for a key press; the computer may have executed its calculation program incorrectly; the program itself may be erroneous; the display may have received wrong data from the computer; the display may have produced a wrong screen image of the result; P may have misread the screen; P may have written the wrong result on the invoice line. Each of these putative causes of a claimed end-to-end failure can be separately examined and evaluated at the relevant node and at its local connections.

The assessment at each node will be based on facts peculiar to it and to the current stage of the evolving transaction. For example, different technologies for keyboards and screens demand examination of different hypothetical diagnoses of possible error. A certain high speed laser copier offers an example of a remarkable (and completely unexpected) error. The copier's high speed was achieved by recognising areas of each page whose contents might later be exactly repeated on the same or another page. Storing such formatted areas, and recognising any later occurrences of the same content, allowed each later occurrence to be printed without expending the time needed to format the same content again. A bug in the copier's internal software occasionally produced a page in an accounts printout in which just one amount in a column was wrongly printed. The erroneous amount was the only difference from a previously encountered area, and the software had failed to detect the difference.[1]

Complexity in realistic systems

In general, realistic systems are far more complex than the trivial system of the pocket calculator. The Post Office Horizon system is a notable example. The Technical Appendix to the judgment in *Bates v Post Office* makes clear that the system is very complex and has been modified to accommodate many interventions whose explicit purpose is to repair discovered errors in operation.

Possible sources of complexity are many. Their consequences may frustrate attempts to identify the processing path of a particular transaction; or the complexity at a certain computer node may be beyond the reach of any feasible analysis. Such outcomes should not lead merely to abandoning the effort to detect error in the transaction in hand, but rather to acknowledging explicitly that the tracking failure strongly suggests that the system cannot offer good evidence of its proper functioning, at least in respect of that transaction. The following paragraphs name four aspects and sources of complexity (among a much larger number) and mention some potential and actual consequences.

Complex topology. The topology of a system is analogous to a street map for the delivery of retail purchases: it shows constraints on the possible paths of data stream processing, in which the network connections are the streets and the nodes are street junctions. The topology of the trivial system is linear: the nodes of the network are configured in a single line in which data can flow only in one direction. (More exactly: if, as we should, we consider the person P to be a node of the whole system, the topology is a circle in which each end-to-end transaction path begins and ends at the same node P.) In a complex topology, each successive computer node, in addition to operating on the transaction's data, can also vary its subsequent path.

Quasi-persistent mutable data. A software program executed by a computer has local data stored within the computer (for example, partial results obtained during a computation): while the program is being executed these values can be used, updated, and

[1] A discussion of the copier and its behaviour by D. Kriesel, Xerox scanners/photocopiers randomly alter numbers in scanned documents, is available at https://www.dkriesel.com/en/blog/2013/0802_xerox-workcentres_are_switching_written_numbers_when_scanning.

output. When program execution is complete these local variables and their values are lost. A disk drive device, by contrast, can hold structures of mutable data whose current values remain available to any computer to which the device is currently attached and whose program reads or writes data of the structure. For example, a product database may be held on a disk drive containing details of a set of products and their prices. At different times these details can be read or written by different programs being executed on the same or different machines. If some path step of a transaction is at a computer node that can vary the subsequent path, the relevant quasi-persistent data values available to the computer at that point in time must be identified and taken into account in tracking and analysing the processing of the transaction.

Replaceable software. The software text executed by the computer node in the pocket calculator is not replaceable. If it is unsatisfactory the computer (in effect, the whole calculator) must be replaced. During the operational lifetime of a realistic complex system, the software at a computer node may be replaced by a new or modified version for any of several reasons. A modification may be needed because a bug has been found and must be corrected. In a system with several computer nodes, the software of more than one node may require correction and replacement, to bring the system to a consistent state of proper functioning. However, there is a difficulty. If software is replaced at nodes N1 and N2, some transactions may have progressed beyond N1 but not yet reached node N2. Eventually, the paths of these transactions may include the old N1 and the new N2: this combination may fail to provide proper functioning.

Dynamic topology. The topology of a system may be changed in many ways: the easiest and commonest way is by disconnecting and reconnecting disk drive devices. Because two computers can be simultaneously connected to the same disk drive, these devices in effect allow two computer nodes to be connected by paths that were not envisaged by the system designers and may have undesirable consequences. A notable example occurred in the control system at Unit 2 of the Edwin I. Hatch nuclear power plant in Georgia USA.[2] The plant had both control systems and business systems. One business system monitored diagnostic data from a control system. A software update on this business system was designed to synchronise data on both systems; but the synchronisation process erroneously reset some data on the control system, causing a shutdown of the reactor. In this incident, the business system was connected to the control system solely to read the control system's data and should not have been able to reset it. In the Horizon system Fujitsu engineers were able to connect to the system data for the explicit purpose of changing it. They made changes to repair the effects on branch data of errors they had discovered in the programs; they were also able, more generally, to rebuild branch accounting data after equipment failures, power outages, and other disruptive incidents.

Some aspects of tracking

Long transactions. The concept of a long transaction, presented in this paper, is particularly suited to the Horizon branch accounting system because a long transaction is the typical arena of dispute in cases of large discrepancy. Often, the amount of the discrepancy is the amount of an identifiable large sale or payment, or a transfer of funds or stock in the Trading Period concerned. The application of the concept, however, is more general than this explanation may suggest. In systems of many different kinds the functional system behaviour is largely, though not entirely, understandable in terms of episodes that can be understood and analysed individually. In a hotel system, for example, *Guest Check In* and *Guest Check Out* may be such episodes. In an automotive system the Automatic Park and other features may be episodes. If these episodes are made explicit in the system design, and also in its implementation, their processing in the system may be more easily tracked and analysed.

Step size. The end-to-end view of a long transaction is one extreme choice of step size for tracking: everything that happens, from initiation to completion, is regarded as a single step at a single node, which may prove to consist of the whole system. Evaluating the reliability of the end-to-end relationship evidenced by the system includes assessment of the whole system's reliability or trustworthiness in its role as the single node. A smaller step size allows a structure of assessments of smaller nodes. For example: even in the pocket

[2] Brian Krebs, [ISN] Cyber Incident Blamed for Nuclear Power Plant Shutdown, *InfoSec News*, Thu Jun 05 2008 - 22:28:34 PDT, at http://lists.jammed.com/ISN/2008/06/0025.html.

calculator example it is in principle possible to consider transmission of data at a port between the computer and a connected device as a step, in which the computer port is the significant node. It can be argued that in their respective contexts, the statistical reliability of a port is more significant to the purpose of tracking than the statistical reliability of the whole system. Broadly, a smaller node has a smaller variety of possible behaviours, and hence fewer opportunities to discriminate between one transaction and another.

Auditing and error logging. Auditing and error logging practices are essential in the operation of a complex system, if only to provide records of changes to the system fabric and hence to its complex topology. For this reason, they are often built into modern operating systems and database systems as standard features. For transaction tracking such records are indispensable, but they do not serve the tracking purpose directly. The black box that is mandatory in commercial aircraft, and may become commonplace, or even mandatory, in motor vehicles, is more to the purpose. The characteristic properties of a black box are that: (a) it records cockpit conversations and pilot commands to the flight control system for later examination; (b) its records are direct, ensuring that their reliability is not open to dispute; (c) the box itself is independent of the aircraft's other systems and their complexities and vicissitudes; and (d) it is effectively invulnerable to interference of any kind from people or ambient conditions. Broadly it satisfies the stipulations of article 10 of the Model Law on Electronic Transferable Records (2017)[3] of the United Nations Commission on International Trade Law.

Software programs. The notorious propensity of software programs to harbour bugs of many kinds made them an obvious subject for fruitful research in computer science. The aim was to reduce bugs in general and to improve software development theory and practice to the point of eliminating some kinds of software bug entirely. It also makes them a natural first port of call in exploring a system's suspected failure to function properly. But to concentrate investigation and diagnosis of a system failure on the software program texts alone would be to look for the lost sixpence under the streetlamp because the light is brighter there. Many complexities and consequent failures are caused by software replacement, dynamic system topology, and the data mutations and movements in time and space made possible by database devices. In general, the potential and actual effects of these factors are not mentioned in software program texts.

Paper summary and argument

Motivated by reports of the Horizon system, and by the judgment of Fraser J in the case of *Bates v Post Office*, a highly simplified view of computer-based systems, and of the structure of the transactions that they process, has been presented. This view could perhaps be likened to proposing a garden trowel as a tool to move a mountain: the power and capacity of the tool are too small by far for the task in hand. The justification for such a suggestion is, as always, that an extreme simplification may sometimes offer or provoke useful insights into a problem of great complexity.

The problem to be addressed is not the problem of developing complex computer-based systems: addressing that problem is a continuing challenge for a world-wide community of computer scientists and software engineers and programmers. Rather, the problem is a forensic difficulty, signally demonstrated in the Horizon cases, of challenging and evaluating evidence derived from a complex computer-based system.[4] The number of distinct states that such a system, including its databases, can get into is effectively infinite. So, too, is the number of distinct behaviours it can exhibit in its interactions with the physical and human world.

When, in a specific case, evidence derived from a very complex system is to be evaluated, the general approach briefly described in this paper offers two advantages. First, the scope of the necessary investigation is restricted to the case in hand. Second, within that restricted scope the investigation can be sharply focused on the proper functioning of the system in processing the particular steps in the transaction in question.

[3] Date of adoption: 13 July 2017.

[4] For a high-level discussion of the position in the United States of America, see Steven M. Bellovin, Matt Blaze, Susan Landau and Brian Owsley, 'Seeking the Source: Criminal Defendants' Constitutional Right to Source Code,' 17 Ohio State Tech. L.J. 1, 38 (2021).

An approach to the judicial evaluation of evidence from computers and computer systems

Michael Jackson is a Visiting Research Professor at the Department of Computing, Open University, Milton Keynes, and researcher in software engineering.