Journal of Digital Forensic Practice, 1:19–26, 2006

ISSN: 1556-7281 print / 1556-7346 online
DOI: 10.1080/15567280500541421

Taylor & Francis

ARTICLE

Digital Evidence: Challenging the Presumption of Reliability

Eric Van Buskirk
2826 N. 43rd Street, Phoenix, Arizona 85008, USA

Vincent T. Liu
1378 West Maria Lane, Tempe, Arizona 85284, USA

ABSTRACT There is a general tendency among courts to presume that forensic software reliably yields accurate digital evidence. As a judicial construct, this presumption is unjustified in that it is not tailored to separate accurate results from inaccurate ones. The authors illustrate this unfortunate truth by the presentation of two currently uncorrected weaknesses in popular computer forensic tools, methods, and assumptions. Some percentage of these forensic software errors (and ones like them) will necessarily have negative effects on parties, whether in terms of faulty criminal convictions or improper civil judgments. The authors argue that the collective value of these negative effects among parties is far larger than the costs of research and development required to prevent such negative effects. Under a purely rational economic approach to the law, this dynamic constitutes an inefficiency to be corrected through the proper application of rules. The authors advance two approaches to cure current defects. One is through the proper application of scientific jurisprudence to questions of digital evidence and the other is through some combination of certain broad market and social corrections.

KEYWORDS computer forensics, cybercrime, Daubert, digital evidence, EnCase, Federal Rules of Evidence, software reliability, source code

> Nobody knows how bad the state of software really is.
> It's as if, in constructing a building, all the drywall went up before anyone looked at what was inside.
>
> –William Guttman[1]

Eric Van Buskirk is a technical consultant, author, and attorney specializing in information security, computer forensics, and electronic discovery. He is licensed to practice law in Arizona, has a master's degree in analytic philosophy, and expects to receive his CISSP certification in March 2006. His writing on electronic discovery has been cited to the United States Supreme Court. He can be reached at evb@azbar.org.

Vincent T. Liu is a partner at Stach & Liu, a firm providing advanced IT security consulting to the Fortune 500 and international financial institutions. Vincent formerly worked as a security specialist with Honeywell International, as a consultant with the Ernst & Young Advanced Security Center, and as an analyst at the National Security Agency. Vincent holds a degree in Computer Science and Engineering from the University of Pennsylvania. He is lead developer for the Metasploit Anti-Forensics project, and a contributor to the Metasploit Framework. Vincent is a published author and has presented at BlackHat, ToorCon, and Microsoft BlueHat. He can be reached at mailto:vliu@stachliu.com.

INTRODUCTION

Should forensic software[2] be entitled to a judicial presumption of reliability? When, if ever, should courts compel non-party forensic software vendors to reveal proprietary source code to party experts in order to assure a fairer trial? And what does *reliability* mean in the context of digital evidence anyway?

These issues have been raised by a recent criminal prosecution in Florida. In *Florida v. Bjorkland*,[3] the defendant was charged with driving-under-the-influence of alcohol (DUI). In response, the defendant moved to exclude evidence of intoxication obtained from the Intoxilyzer 5000 breath-alcohol analysis computer (IT5000). Although the defendant advanced several theories in support of her motion, the broader issue presented by the case is whether experts should be entitled to audit software source code to ensure the reliability of digital evidence.

Bjorkland is captivating because it elicits conflicting values that directly affect the practice of computer forensics. Intellectual property rights holders fear that compelled source code disclosure will encourage competitors to use discovery procedures to conduct industrial espionage. Parties urge that compelled source code disclosure can be essential to ensuring the admission of accurate evidence. Courts are concerned that an order to compel source code disclosure in one case is likely to foster requests to audit the individual quality of engineering in, for example, an integrated circuit, a gate array, luminal, or paint.

The idea that the law would permit parties to compel the disclosure of business-critical trade secrets may be counterintuitive. However, although state laws vary, there is no specific federal "trade secret privilege" that absolutely prohibits the ability of parties to compel disclosure of trade secrets. For example, in *Coca-Cola Bottling Co. v. Coca-Cola Co.*, under the guise of preparing its experts, Coca-Cola Bottling Company successfully moved to compel The Coca-Cola Company to disclose the secret formula of the Coca-Cola brand soft drink.[4]

What may be even more surprising is that *non-parties* may be subject to orders to compel disclosure of proprietary information: Federal Rule of Civil Procedure 45(a) authorizes non-party discovery in civil cases,[5] whereas Federal Rule of Criminal Procedure 17(c) authorizes it in criminal ones.[6] Despite the potentially wide scope of federal discovery procedures, orders compelling the disclosure of non-party trade secrets are a relatively infrequent occurrence.

The purpose of this article is to present and discuss important legal and technical issues involved in the effort to ensure the reliability of forensic software and the accuracy of digital evidence. There is a general tendency among courts to *presume*—without the benefit of meaningful assurance—that forensic software can be trusted to yield accurate digital evidence. As a judicial construct, this presumption is unjustified in that it is not tailored to separate accurate results from inaccurate ones. This unfortunate result will be illustrated by the presentation of two current, but uncorrected, weaknesses in popular computer forensic tools, methods, and assumptions.

Some percentage of these software errors (or ones like them) will necessarily have negative effects on parties, whether in terms of faulty criminal convictions or improper civil judgments. These costs may be assigned monetary values. The authors argue that the collective value of negative effects caused by software defects among parties is far larger than the costs of research and development required to prevent such negative effects.

If there were some sustainable mechanism to force software developers to bear appropriate research and development costs, the net gain to society and the system of justice would significantly increase. The authors argue there are indeed two broad mechanisms to promote social net gains. One such method is through the proper application of scientific jurisprudence at the individual case level. However, judges must not bear sole responsibility for the quality of forensic software. The authors therefore argue that social policymakers can significantly increase the accuracy of forensic software and digital evidence through some combination of certain market and social corrections.

THE PRESUMPTION OF RELIABILITY

Courts have a seemingly unflappable faith in the ability of software to render reliable evidence. Broadly speaking, this "Presumption of Reliability" is well-entrenched in American law. For example, in *Olympic Ins. Co. v. H. D. Harrison, Inc.*, the court wrote that digital evidence had a "prima facie aura of reliability."[7] Likewise, in *United States v. Moore*, the court noted that "ordinary business circumstances . . . suggest trustworthiness. . . .

at least where absolutely nothing in the record in any way implies the lack thereof."[8] Similarly, in *California v. Martinez*, the court noted that testimony on the "acceptability, accuracy, maintenance, and reliability" of computer software is not prerequisite to admission of computer records.[9] And in *Missouri v. Dunn*, the appellate court agreed with the trial court in "concluding that [such] records were uniquely reliable in that they were computer-generated rather than the result of human entries."[10] Although the Presumption of Reliability may have found roots in the context of business records, it is by now so pervasive that it should be recognized to be co-extensive with the law of digital evidence itself.[11]

The Presumption of Reliability is difficult to rebut. Unless specific evidence is offered to show that the particular code at issue has demonstrable defects[12] that are directly relevant to the evidence being offered up for admission, most courts will faithfully maintain the Presumption of Reliability. But because most code is closed source and heavily guarded, a party cannot audit it to review its quality. At the same time, however, source code audits are perhaps the best single way to discover defects.[13]

This difficulty gives rise to an important question: if a party cannot gain access to source code without evidence of a defect, but cannot get evidence of a defect without access to the source code, how is a party to rebut the Presumption?[14] Rather than wrestle with, or even acknowledge, this conundrum, most courts simply presume that all code is reliable without sufficient analysis.

With only a few exceptions, even if a party does manage to raise some doubt as to the reliability of particular code, most courts will still admit into evidence output from such code, subject only to the caveat that any reliability concerns should affect the *weight* of the evidence, not its *admissibility*.[15] But, given the overwhelmingly prevalent belief that "machines are immune to human frailties, desires, and whims that can lead to erroneous information or misinterpretation,"[16] is this purported compromise of any value?

In many circumstances, it is not:

> [T]he reliability of a particular computer system or process can be difficult to assess. Programmers are fallible and can unintentionally or purposefully embed errors in their applications. Also, complex systems can have unforeseen operating errors, occasionally resulting in data corruption or catastrophic crashes. Possibly because of these complexities, courts are not closely examining the reliability of computer systems or processes and are evaluating the reliability of digital evidence without considering error rates or uncertainty.[17]

TWO EXAMPLES

File timestamps are one of the most heavily relied-upon techniques used by computer forensic examiners to timeline and identify suspicious behavior. On FAT-related file systems, timestamps are properties of a file that are composed of a Last Modified time (M), a Last Accessed time (A), and a Created time (C). On NTFS file systems, there is a fourth timestamp called the Entry Modified (E) time. Timestamps were designed to be used by the computer operating system for accounting purposes, but they have been adopted as an important investigative tool as well.

The M-timestamp is updated when file data changes, the A-timestamp is updated when the file is accessed, and the C-timestamp is updated when the file is created on the file system. On NTFS file systems, MACE times are stored in two different attributes in the Master File Table: the Standard Information (SI) attribute and the File Name (FN) attribute. Again, on NTFS file systems, the E-timestamp is updated when anything changes to the MFT record entry, including any modification to the first three MAC values.

How do timestamps play a critical role in cyber-investigations? If, for example, a suspicious file is found to have a creation date of Monday, November 28, 2005, at 3:15 pm, the theory of temporal locality would suggest that other suspicious files may have been modified, accessed, or created at or near the same time. If that is correct, proper investigative technique should lead the investigator to look for other files with temporally near timestamps in order to locate other suspicious files. But proper investigative techniques work only if computer forensic software accurately reports file timestamps.

EnCase Forensic Edition (EnCase) is a popular computer forensic software suite. Guidance Software, the corporate developer of EnCase, claims that "[c]omputer evidence recovered with EnCase software has been admitted into thousands of court proceedings in several countries and jurisdictions."[18] With such a vast market presence, the reliability of a substantial amount of digital evidence may be divined by examining the reliability of evidence yielded by EnCase.

The method by which EnCase analyzes file systems is easy enough to understand. After acquiring a hard drive image, EnCase reads the NTFS MFT records in order to determine all MACE values on the file system. In particular, EnCase by default obtains the MACE values by reading the SI attribute of each file record within the MFT. MACE values are displayed within the EnCase interface so that the cyber-investigator can use them to create a timeline of suspicious activity, and to detect other aberrant data.

If there is any question as to EnCase's ability to analyze timestamps properly, then the investigation of hundreds or thousands of cases could be affected. If, for example, there were methods by which timestamps could be undetectably modified, especially in ways previously unknown to the investigative community at large, then it might be the case that investigations were (or will be) aborted, that illegal activity was (or will be) concealed, or that innocence was (or will be) feigned. It is thus very important that forensic software be highly accurate and that its errors and rates of occurrence be known.

There have long been programs in existence that modify timestamp values on FAT-related and NTFS file systems. Research shows that these programs rely on the well-documented **SetFileTime** function call in the Windows Application Program Interface (API). However, the **SetFileTime** call only works undetectably against the M, A, and C (MAC) timestamps; it does not work on the fourth (E) timestamp. Because FAT-related file systems store only the first three timestamp values (i.e., MAC timestamps), traditional MAC modification programs have long been able to modify MAC values in ways that can damage an investigation. This fact should be well known to the cyber-investigative community.

Still, because MAC modification programs typically have been thought to modify MAC timestamps on NTFS file systems, but not the fourth timestamp (E), traditional computer forensic wisdom has held that MACE timestamps on NTFS file systems were generally safe from surreptitious modification attempts via the **SetFileTime** call. That is, traditional thought holds that if an attacker attempts to modify NTFS timestamps with any tool programmed to use the **SetFileTime** function call, such modification will fail because the E-timestamp will always update to the exact time that the modification was attempted. Thus, *previous* timestamp modification techniques were easily detectable because the E-timestamp would fall out-of-sync with the MAC timestamps. That is, with regard to NTFS file systems, traditional thinking holds that the temporal disjunction between the MAC values and the E-value could always be used to indicate the presence of timestamp modification attempts.

However, recently published research[19] shows that by using the undocumented Windows system call, **NtSetInformationFile**, instead of the **SetFileTime** call, all four timestamps can be undetectably modified (i.e., fabricated). And this weakness is not limited to EnCase; it applies to any computer forensic software that makes use of the Windows API to read file timestamps. Not only is the weakness shared by other forensic software, but it is quite easy to exploit. One need not even be well-versed in software engineering to exploit this weakness, as a command line tool that does it almost effortlessly has already been written and published on the Internet.[20]

Additional research shows the existence of another weakness in the way forensic software such as EnCase employs the API. In particular, this weakness allows an attacker to impede the ability of EnCase (and others) to translate timestamps properly. EnCase and its ilk rely on the Windows API to convert binary timestamp values into human readable format. However, exploit code has been written that can subvert this conversion. Again, using the **NtSetInformationFile** function call, file timestamps can be modified so that they are *not* translatable into human readable values. Because EnCase (and others) depend on the Windows API to translate binary system time values into the human readable format, an attacker using this technique could hide all file timestamps from an investigator.

PROBLEMS AND SOLUTIONS

As it currently stands, the Presumption of Reliability has two important problems. First, it facilitates the admission of inaccurate digital evidence. When forensic software lacks independent, thorough, and scientific testing, and when it may have been prematurely sold to customers in beta form so as to meet quarterly sales requirements, there is little justifiable basis for presuming either reliability or unreliability–that is, we should not presume to know one way or the other.

There is a further problem with the Presumption of Reliability. By facilitating the admission of inaccurate results, the Presumption of Reliability is economically

inefficient in that it fails to force developers to internalize costs.[21] Instead, costs from code defects are passed on to parties in the form of faulty criminal convictions, improper civil judgments, lost opportunity, and the like. If these costs were assigned monetary values, in many cases the failure to correct various forensic software defects would impose on society a net loss.

To take a simple example, if the cost to a software developer (in terms of extra research and development) to fix a weakness is $50, but the cost to parties (in terms of lost property, freedom, or opportunity) of not fixing the weakness is $100, then the law should be used so as to force the developer to internalize the costs to correct the weakness.

Software such as this is economically inefficient, for it would be much less expensive to fix the defects rather than to pass them downstream to innocents. In the case of the two previously discussed API weaknesses, the cost to remediate these problems is minute compared to the downstream costs borne by the throng. Specifically, small sections of the EnCase code base could be updated to analyze timestamps with custom translations functions, so that it would not be necessary to rely on those built into the Windows API. Accordingly, forensic software subject to these weaknesses (and others like them) should be construed as economically inefficient.

Judges and social policymakers should use their powers in order to correct current inefficiencies in forensic software so as to improve the accuracy of digital evidence. There is a range of solutions to be considered, both at the individual case level as well as at the social policy or market level.

Case Level Improvements

At the level of cases and controversies, judges should more closely scrutinize digital evidence resulting from cyber-investigations. One way of elevating the foundational requirements for such evidence is to consider whether the software yielding it has been validated in accordance with the procedures set forth in *Daubert v. Merrell Dow Pharmecuticals*.[22] *Daubert* is a United States Supreme Court decision that considered the standards under which scientific evidence is admitted. In *Daubert*, the Court ruled in part that Rule 702 of the Federal Rules of Evidence imposes on trial judges a special obligation to ensure that scientific evidence is reliable.[23] The Court provided a four-part test to assist in this determination. In assessing reliability, trial courts applying federal law should consider: (1) whether the theory or technique has been reliably tested; (2) whether it has been subjected to peer review; (3) the known or potential rate of error of the theory or technique; (4) whether the technique is generally accepted. The holding of *Daubert* was subsequently extended to technical evidence in *Kumho Tire Co. v. Carmichael*.[24] Further, a trial court in Texas applied *Daubert*-like criteria to EnCase in finding it "reliable."[25] In so doing, the court implicitly acknowledged that different software and different software versions qualify as different "theories or techniques."[26]

What should proper *Daubert* testing mean in the context of forensic software? The independent source code review of all software is perhaps the single most revealing method to uncover defects. In many cases, however, the proprietary nature of source code means that it is unavailable for such review. In such cases, unless a court compels source code disclosure, *black box testing*–also referred to as *zero-knowledge application review*–becomes an indispensable way to validate particular software scientifically.

Black box testing is a software testing methodology in which assessors attempt to identify flaws in software without prior knowledge of the application's internal logic. Black box testing has at least two sub-components. First is *fuzzing*, which is the process of attempting a large range of application inputs in order to locate unexpected behavior. Second, *logic bypassing* involves attempts to subvert the intended operation of software by attempting inputs the programmer did not anticipate.

Black box testing is very important for software reliability assessment. Indeed, it was this form of testing that was used to discover the API weaknesses described above. Despite its advantages, however, black box testing has noteworthy limitations. First, it is very difficult to perform comprehensive black box testing because fuzzing is so time intensive. Second, because many applications are made available only to selected persons, it is often impossible to conduct independent research. Thus, it should be of great concern to the computer forensics community that there is a growing tendency among forensic software vendors to refuse to distribute some of their products to non–law enforcement personnel.[27]

The publication of computer forensic software reviews is another crucial way to validate computer forensic software for purposes of *Daubert*. To conform properly to scientific jurisprudence, however, it is important that reviewers conduct independent and verifiable accuracy testing in a variety of conditions. Published results that do not reflect these qualities should not be used by courts to assay the "reliability" of forensic software.[28]

Cross-validation of computer forensic tools is another important component of software reliability assessment. Cross-validation involves *cross-checking* the results of one software tool against the results of another; if the results are consistent, increased confidence in the reliability of both tools might be justified. Although cross-validation is important, it assumes the baseline tools are themselves reliable. As illustrated above, it may currently be a mistake to cross-validate one software tool reliant on the Windows API against another such tool. Although the National Institute of Standards and Technology has significantly improved cross-validation baselines for certain kinds of forensic software tools,[29] additional work and additional funding are needed to expand these baselines.

Social and Market Level Analysis

Perceived imperfections in the free market, intellectual properly law, and the profession of software and hardware engineering must be considered in the overall effort to improve the reliability of forensic software and accuracy of digital evidence.

The current free market is perhaps the primary contributor to current inefficiencies. If developers spent more time testing, significant errors could certainly be reduced without great cost.

> [D]evelopment schedules have been compressed well beyond the 25% point, design and code reviews have been reduced or eliminated (effectively eliminating the opportunity to discover and correct both design and coding errors), testers have been turned into developers (thus reducing further the possibility of finding and correcting the more subtle design and coding failures), and test time has been minimized in the rush to market. Obviously then, large numbers of software defects and design oversights are shipped to customers.[30]

Open source software has been proposed as a way to alleviate undesirable effects of a free market. *Open source* is a term used to describe "the conditions under which software source code used by computers is made available to others apart from the developer."[31] Open source software has been offered to improve the quality of digital evidence because it purportedly "makes it easier to verify whether or not a vulnerability exists that could have allowed the tampering, alteration, corruption, or forgery of information produced by the software. . . ."[32] If society is to continue to allow developers to use customers as beta-testers, would it not be a good idea at least to allow independent review and testing of the source code?

Although open source has been offered as one way to improve the quality of forensic software and digital evidence, it is not yet clear how this would work in practice. Is the idea to establish open source cross-validation baselines against which proprietary tools are measured? If so, it is difficult to construe that offering as a complete solution: no matter how vast the offering of open source forensic software, there will likely always be a set of proprietary, closed source tools that do not have an open source equivalent. As long as that is true, the courts and the industry will always be faced with the challenge of authenticating the output from such proprietary ware. Secondly, although the open nature of open source software means source code *can be* reviewed,[33] it does not follow that such code will *actually* be reviewed.[34]

Next, various industry solutions have been proposed as a way to improve the software development process. The ISO/IEC TR 15504 software process assessment standard,[35] the Capability Maturity Model,[36] Harlan Mills' Cleanroom software development process,[37] and work by the Sustainable Computing Consortium,[38] have all been proposed as ways to improve the quality and reliability of software. For example:

> [d]ata from the use of [Harlan Mills' Cleanroom software development process] at NASA have shown 25 to 75 percent reductions in failure rates during testing. Use of Cleanroom also showed a reduction in rework effort so that only 5 percent of the fixes took more than an hour, whereas the standard process caused more than 60 percent of the fixes to take that long.[39]

Although industry efforts such as these are laudable, benefits will accrue only if the standards are comprehensive and mandatory and if developer participation is meaningfully monitored.

Professional licensing of software engineers is another way to improve the quality of software in general, as well as forensic software in particular.

> In the U.S., mandatory licensing has been used as a means to protect the public from malpractice by those offering services directly to the public, such as doctors, lawyers, civil engineers, contractors, day care workers, barbers, and surveyors. Many licensing advocates argue it would help promote software engineering into a profession and would safeguard society against incompetent engineers.[40]

To be effective, professional licensing would need to be free from inappropriate commercial influence. And as with the industry efforts described above, professional licensing standards would need to be properly developed, implemented, and monitored in order to be effective.

THE FLORIDA APPROACH

In *Bjorkland*, the Florida DUI case mentioned at the outset, the trial court granted the defense motion to compel disclosure of the IT5000 source code so as to improve the quality of evidence in that trial.[41] Accordingly, the prosecution was ordered to disclose to the defense expert the IT5000's firmware source code. The court ordered disclosure even though the prosecution did not have possession of it. Unsurprisingly, as of the date of this article, CMI, Inc., manufacturer of the IT5000, steadfastly refuses to make the IT5000 source code available to the prosecution. Without source code from the IT5000, the prosecution will be unable to make its case. The prosecution is therefore faced with the choice of either dismissing the DUI charges or appealing the ruling.

In spite of the *Bjorkland* court's attempt to improve the accuracy of the digital evidence, there are several notable problems with compelled source code disclosure. First, compelled disclosure leads to the familiar slippery slope. For example, based on *Bjorkland*, should Florida courts now allow parties to inspect the hardware instruction sets of radar gun signal processing chips every time a criminal speeding ticket is challenged? Alternatively, shall murder defendants be allowed to hold up a trial because the testifying expert wishes to audit the chemical properties of Hemaglow, a proprietary blend of luminal?

Second, because of its case-by-case nature, compelled disclosure of source code is repetitive and inefficient. That is, even if the IT5000 firmware source code passed muster in *Bjorkland*, a new audit could be requested each time a defendant were charged with DUI based on evidence obtained from a breath/alcohol machine. When this redundancy is extrapolated to tens of thousands of DUI trials nationwide, the inefficiencies quickly become enormous. Compelled source code disclosure should therefore be considered only as a last resort in cases where *Daubert* validation testing or other social or market corrections have failed.

CONCLUSION

Software is "unique among complex engineered products, insofar as we have very little ability to understand the inherent level of quality in the product."[42] This difficulty makes it tempting for courts to presume that digital evidence is reliable. This presumption results in the admission of evidence that is both inaccurate and economically inefficient. Compelled disclosure of non-party source code has recently been used to improve the quality of digital evidence. However, the case-by-case nature of this method is repetitive and costly. Courts can improve accuracy rates and avoid inefficiency by properly following the strictures of *Daubert*. Policymakers can help by implementing appropriate market or social corrections. Neither the application of scientific jurisprudence nor corrections to the social or market rules can guarantee error-free digital evidence–indeed, that is not the point. The issue is not how to attain perfection, but rather how to use the law to inject efficiency into a system of justice that is not sufficiently skeptical of that which is offered up as proof.

NOTES

1. Alorie Gilbert, *Newsmaker: Fixing the Sorry State of Software*, CNET News, Oct. 9, 2002 (interviewing William Guttman, Director, CyLab Sustainable Computing Consortium, Carnegie Mellon University), at http://news.com.com/2008-1082-961370.html.
2. "Software" as used in this article includes traditional software instruction sets, as well as those found in firmware, and hardware.
3. Florida v. Bjorkland, No. 2004 CT 014406 SC (Sarasota County, 2005).
4. Coca-Cola Bottling Co. v. Coca-Cola Co., 107 F.R.D. 288 (D. Del. 1985).
5. See Fed. R. Civ. P. 45, Committee Notes, 1991 Amendment *Subdivision (a)* ("Fourth, *Paragraph (a)(1)* authorizes the issuance of a subpoena to compel a non-party to produce evidence independent of any deposition."); *cf.* Fed. R. Civ. P. 26(c)(7) (court *may* grant protective order to limit disclosure of trade secrets, but is not required to do so).
6. Fed. R. Crim. P. 17(c)(1). See, e.g., United States v. Iozia, 13 F.R.D. 335, 338 (E.D.N.Y. 1952); see also C. Wright, Federal Practice and Procedure: Criminal § 274 (1982).
7. Olympic Ins. Co. v. H. D. Harrison, Inc., 418 F.2d 669, 670 (5th Cir. 1969).

8. United States v. Moore, 923 F.2d 910, 915 (1st Cir. 1991).
9. California v. Martinez, 990 P.2d 563, 581 (2000).
10. Missouri v. Dunn, 7 S.W.3d 427, 432 (1999).
11. See generally, Williford v. Texas, 127 S.W.3d 309 (2004) (finding computer forensic software to be reliable); Ohio v. Scurti, 153 Ohio App. 3d 183 (2003) (allowing "expert" testimony on software analytical functions by company CEO, although CEO had only "limited knowledge" of the relevant software); Wisconsin v. Busch, 576 N.W.2d 904 (1998) (affording presumption of accuracy to untested device, where it was composed of hardware that was "essentially the same" as the device's predecessor); Lattarulo v. Georgia, 401 S.E.2d 516, 519 (1991) (holding that "Intoximeter 3000 machine test results are based on accepted scientific theory or rest upon the laws of nature," thereby confusing the accuracy of the general method with the accuracy of a particular instance); California v. Lugashi, 205 Cal.App. 3d 632 (1988) (presuming digital evidence to be reliable); New Jersey v. Yerkes, 458 A.2d 1345, 1347 (1983) (finding that where the Breathalyzer 900 has already been certified for use, the model 900A need not be certified as well, where differences between the two models "consisted of [only] the addition of an automatic timing device and the substitution of a nullmeter for a galvanometer"); see also Fed. R. Evid. 901(b)(9) (providing for the authentication of evidence obtained through an automated processes or systems); McCormick on Evidence § 294 (John W. Strong et al., eds., 4th ed. 1992) (stating that with regard to the accuracy of business records, "the courts have moved in the direction of not imposing rigid requirements.").
12. There are generally two kinds of software defects: design defects and coding defects. With design defects, the product works as designed, but the design is flawed. With coding defects, the code itself is flawed, and the program does not behave as intended.
13. Erin E. Kenneally, *Gatekeeping Out Of The Box: Open Source Software As A Mechanism To Assess Reliability For Digital Evidence*, 6 Va. J.L. & Tech. 13, ¶ 94 (2001).
14. Kenneally, 6 Va. J.L. & Tech. 13 (2001).
15. United States v. Catabran, 836 F.2d 453, 458 (9th Cir. 1988) (stating questions "as to the accuracy of [computer] printouts, whether resulting from incorrect data entry or the operation of the computer program, as with inaccuracies in any other type of business records, would [affect] only the weight of the printouts, not their admissibility.").
16. Keneally, *supra* note 13, at ¶ 39.
17. Eoghan Casey, *Error, Uncertainty, and Loss in Digital Evidence*, 1 Int'l. J. Digital Evidence, Issue 2 (2002), at http://www.ijde.org/archives/docs/02_summer_art1.pdf.
18. Guidance Software Website, at http://www.guidancesoftware.com/lawenforcement/ef_index.asp (last visited Nov. 29, 2005).
19. Vincent T. Liu, Presentation at the BlackHat conference (July 28, 2005).
20. Timestomp, at http://www.metasploit.com/projects/antiforensics/.
21. *See generally* Ronald Coase, *The Problem of Social Cost*, 3 J.L. & Econ. 1 (1960).
22. Daubert v. Merrell Dow Pharm., Inc., 43 F.3d 1311 (9th Cir. 1995).
23. *Id*.
24. Kuhmo Tire Co. v. Carmichael, 526 U.S. 137 (1999).
25. Williford v. Texas, 127 S.W.3d 309 (2004).
26. *Id*.
27. For example, Guidance Software makes their Field Intelligence Module available only to law enforcement, military and other government agencies involved in computer forensics. The same is true for iLook Investigator software.
28. *Cf. Williford*, 127 S.W.3d 309 (admitting digital evidence based in part on unscientific product evaluation in a magazine article).
29. Computer Forensic Tool Testing Project, National Institute of Standard and Technology, at http://www.cftt.nist.gov/.
30. John R. Michener, et al., *"Snake-Oil Security Claims": The Systematic Misrepresentation Of Product Security In The E-Commerce Arena*, 9 Mich. Telecomm. & Tech. L. Rev. 211, 228 (2003).
31. Erin E. Kenneally, *Gatekeeping Out Of The Box: Open Source Software As A Mechanism To Assess Reliability For Digital Evidence*, 6 Va. J.L. & Tech. 13, ¶ 13 (2001).
32. *Id*. at ¶ 94.
33. *Id*. at ¶ 125 (stating "[c]ode under the Open Source model is subject to falsification. . . .").
34. See David H. Kaye, *On "Falsification" and "Falsifiability": The First* Daubert *Factor and the Philosophy of Science*, 45 Jurimetrics J. 473, 478 (2006) (stating that "[t]ypically . . . it is falsification, and not falsifiability, that matters when it comes to admissibility.").
35. See http://isospice.com/standard/tr15504.htm.
36. "A Capability Maturity Model is a way of measuring how well developed management processes are. Organizations that purchase software can refer to the formal assessment criteria of the provider to demonstrate the reliability of acquired software." Ajoy Ghosh, Presentation at the Incident Response & Forensics Workshop APEC-Tel 29 (March 21, 2004), at http://www.apectel29.gov.hk/download/irf_04a.pdf.
37. See for example John Foreman, *Cleanroom Software Engineering*, Software Technology Roadmap, 1997, http://www.sei.cmu.edu/str/descriptions/cleanroom_body.html.
38. The Sustainable Computing Consortium (SSC) was founded at Carnegie Mellon University (CMU) in May 2002 and became part of CyLab at CMU in 2004. See http://www.cylab.cmu.edu/default.aspx?id=2.
39. Barry Boehm & Victor R. Basili, *Software Defect Reduction Top 10 List*, Computer, Vol. 34, No. 1, Jan. 2001, at 136.
40. Association for Computing Machinery, *A Summary of the ACM Position on Software Engineering as a Licensed Engineering Profession* (July 17, 2000), at http://www.acm.org/serving/se_policy/selep_main.html.
41. Florida v. Bjorkland, No. 2004 CT 014406 SC (Sarasota County, 2005).
42. Alorie Gilbert, *Newsmaker: Fixing the Sorry State of Software*, CNET News, Oct. 9, 2002 (interviewing William Guttman, Director, CyLab Sustainable Computing Consortium, Carnegie Mellon University), at http://news.com.com/2008-1082-961370.html.