

Validation of Forensic Computing Software Utilizing Black Box Testing Techniques

Tom Wilsdon & Jill Slay
University of South Australia
South Australia
tom.wilsdon@unisa.edu.au

Abstract

The process of validating the correct operation of software is difficult for a variety of reasons. The need to validate software utilised as forensic computing tools suffers the same fate and is hampered to a greater extent with the source code of said tools usually not being accessible. Therefore a testing regime must be developed that offers a high degree of correctness and high probability of finding software faults with limited ability to view source code. Software testing is a complex component of software engineering in its own right. This complexity is encountered with an infinite number of environments posed by hardware, and co-hosted software; one can never determine an absolute range of tests to determine the software in each potential environment or custom setting. Therefore a finite set of tests are developed to validate the software. As the software being tested is not disclosed to the source code level the testing is developed around the functions of the software. Using techniques categorized in the black box methodology this regime must attempt to tender a validity status to the software's functions.

Keywords

Forensic Computing, Software, Testing

INTRODUCTION

The variety and number of digital devices available to the domestic user is growing at a phenomenal rate (Grabowski 1998; Etter 2001). Unfortunately such technology is not solely being used for the betterment of life. The misuse of devices to either conduct, or be used as a tool to conduct, crime is well publicised. The unfortunate reality is people embrace, interpret and abuse the added ability one can inherit from the acquisition of digital devices.

To date the significance of encountering digital devices by law enforcement agencies has yielded the creation of dedicated investigators dealing with the interrogation of said devices. A common misconception is that an investigation involving digital means is isolated to a high tech crime when in reality quite the opposite of this is true (Etter 2001).

One such example of this is the uptake and usage of mobile phones. A seemingly isolated device can produce associations between individual parties by reviewing incoming and outgoing communications on the handsets. Any two or more devices offer the ability to identify relationships between devices and subsequently their respective owners. Mobile phones are only one example and similar processes can be applied to computers, personal digital assistants (PDA), fixed line telephones (billing), email accounts, websites, closed circuit television cameras. All devices and many more, potentially offer the investigator the ability to create a series of relationships between individuals via their devices. The usual caveat of proving the individual used the device for a specific message, call, etc still exists.

What this translates to is the fact that evidence exists on countless devices and there must exist some way to extract this in a sound manner. Forensic computing investigators are in the frontline of the analysis of the devices. With this knowledge, law enforcement agencies have equipped their digital forensic investigators with a variety of tools and training to seize, analyse and report relevant information from a truly infinite number of sources.

Most of the tools utilized to conduct such investigations are proprietary software applications developed by specialised application vendors. As the trade secret is the application, they are not willing to disclose source code for testing and evaluation. It is unrealistic to expect companies to do so to law enforcement community, or other parties, to evaluate the 'under the bonnet' as the risk of is too great.

NEED FOR EVALUATION

The need for the evaluation of all forensic computing software is well documented (Reust 2006; Meyers & Rogers 2004). Evaluation frameworks for computer forensic tools do exist, such as NIST Computer Forensic Tool Testing programme and SWGDE, and their work is extremely comprehensive and scientifically sound. However, the output of such organizations is failing to satisfy the demand of the rapidly developing industry. It is therefore proposed that an alternative testing framework be developed and implemented to offer a similar, if not equivalent, level of testing but be more efficient with respect to time, output and financial requirements. Such testing alternatives have been proposed by numerous sources in the Australian context as no such evaluation exists beyond an individual's environment (Armstrong 2003; Wilsdon 2005). This papers proposes a high level architecture for such a streamlined framework.

It is important to reiterate that this evaluation is concentrated on the accuracy, reliability of forensic computing tools and not the processes associated to the investigation. The discipline is well documented with these processes and guidelines for investigators but evaluation of tools is lacking in comparison. The underlying reasons for this are not documented but the authors assume to be a lack of skills in the computer science domain, the financial burden, experience & time.

The technical component of forensic computing can be seen as the most critical with respect to the effectiveness of an investigation. Without the correct tools a tradesman can only deliver a substandard product – this does not differ from the physical to the digital world. So what constitutes a forensic computing tool? To the authors current knowledge there is no definition as to what a forensic computing tool is. Many may have their opinions on this but for the purpose of this research a forensic computing tool is 'a software application, or a component of an application, that is utilized to perform some function on a digital device as part of a forensic computing investigation.'

While this definition is broad and non specific so to is the real world of forensic computing tools being embraced by investigators. Mainstream tools such as Guidance Software's EnCase or Access Data's Forensic Toolkit (FTK) are well known and utilised by most agencies. Further to these mainstream suites there are countless scripts and in-house developed applications that perform forensic tasks. Also the adoption of software developed for another purpose is being utilized more and more. Examples of this include system administration tools, scripts and even operating systems.

Such a range of tools with a variety of purposes increases the problem of testing – if we have one common tool with one clear objective the testing requirements is clearly defined. Unfortunately quite the opposite exists with many tools such as EnCase and FTK providing numerous functions. Complicating this further is the ability for users to create their own functions to extend the existing application. The creation of, and utilisation of existing, tools is growing at a phenomenal rate. It is naive to assume each tool and its functions can be evaluated. The currently testing of tools is a mere drop in the ocean suggesting we must optimise the process and harness all resources to address this shortcoming. Compounding this is the need to examine new devices regularly and their implications.

Holzmann explores the economics for software developers to include as many functions/features into their software and charts a graph suggesting the greater the features the greater the defects. It is then the onus of the developer to employ sufficient quality assurance mechanisms within the development life cycle to identify such defects and address these as required. According to the DFRWS 2005 Workshop Report one vendor of software suggested 'vendors cannot anticipate all uses in all situations therefore they can only test a finite set of permutations of systems and uses' (Reust 2006). This raises the question whether proprietary software is more

reliable and accurate than open source software. Numerous observers actively participate in this debate and at this time there is no clear evidence to suggest to the authors that one model is better than the other.

Holzmann also documents the benefit of testing to identify all defects to suggest there exist some point of time where a majority of defects have been identified within the software development life cycle. Testing beyond this point is unviable for the developer, as the main defects identified are the more likely to occur and cause potentially less impact. Unfortunately the remaining defects within the software are less likely to occur but if they do materialize the impact will be greater. (Holzmann)

So if the vendor is potentially distributing software tools with defects residing within their applications, why is it the forensic computing community's responsibility to identify such defects?

To address this question we must determine what the software, and its results, are utilised for. The word forensic can be categorized as suitable for use in courts of judicature, or in a public discussion (AAFS 1996). We must satisfy the most stringent standard. Therefore it is imperative the processes and procedures undertaken to derive results from a forensic computing investigation is sound in its foundations and execution ensuring results are admissible within the court room as evidence.

Software testing is a science that exists within the discipline of software engineering and computer science. It is a major component of the software development lifecycle and all software engineering methodologies have strong emphasis on testing in various stages of the lifecycle. Within the field of evaluation we have a limited capacity to test software as the lifecycle has evolved from development to deployment. While the testing component is still available, the ability to conduct testing to the same rigor as is available during the development stages is significantly reduced. In addition the accessibility to the source code is dubious at best.

Within this research an assumption has been made that testing is occurring on proprietary software and therefore source code is not available. The debate between open source and proprietary software is active and the authors do not wish to discount the added benefits of an open source testing model but this decision has been made to the following rationale:

- It is acknowledged open source applications offer a greater ability to truly examine the working of the application (due to the ability to examine the source code) but a standard or common denominator must be identified. The reason why a common standard must be identified is due to the objectives of the testing process to validate and verify forensic computing tools correctly and efficiently with as much standardisation as possible. Separating standards between open and closed source does have benefits but also introduces complications which will impact on the efficiency of the framework.
- Individual organizations can conduct lower level examinations to either independently validate or verify results from the validation result achieved.
- Crystal box testing has more available processes in place which may identify defects but it is hoped software engineering methodologies will identify these. The level of familiarity is greater to the developer than an independent party performing validation with constraints of time and resources.
- Creates a universal or standardised testing environment for both open source and proprietary solutions.

TESTING REGIME

The following sections refer to various standards to guide the testing procedure. The definitions of some testing terminology must be clarified prior to continuing – the IEEE Standard 610.12-1990 provides the following definitions;

Test – ‘An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.’

Validation – ‘The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.’

Validation and Verification (V&V) – ‘The process of determining whether the requirements for a system or component are complete and correct and the final system or component complies with specified requirements.’

It is assumed some independent agency will undertake the role of the tester for this process. Such an agency would have to be funded from either a consortium of government and potentially software companies. This structure suggests bias to vendors and developers that contribute to the consortium but the task of establishing such an agency will require major support from vendors and practitioners alike. Therefore their contribution may not come in the form of a monetary value but an in-kind contribution to establish an agency and its processes.

The role of the independent agency acting as the tester is assumed to be the following. Acquire software tools along with all documentation and knowledge required for the evaluation of the product. With respect to ISO 17025-2005 standard appropriate logging of this acquisition is mandatory and must be developed by the agency to suit their business model. Once acquisition is achieved the testing process begins. It is important that the goal of the testing is made apparent before the process evolves to any subsequent state. The goal of the testing is to evaluate a software product to act as a forensic computing tool and determine whether the tool’s function(s) operate correctly and accurately within a prescribed environment. The software product is referred to as a collective form but may consist of a set of functions and each of these functions are required to be identified, documented and tested.

Whittaker (2000) suggests the starting point prior to conducting tests is to plan the testing environment. In summary the four categories described are

1. Modelling the software’s environment
2. Selecting test scenarios
3. Running and evaluating the test scenarios
4. Measuring testing process

While this process appears self explanatory it is important that each of these categories are explored and customized to the proposed forensic computing evaluation architecture.

1. Modeling the Software Environment

Knowledge of the software must be included when preparing for the testing of a forensic computing tool’s function(s). This environment must be created with minimal possibility of contamination from any sources. Such sources of contamination may include remnant data, conflicting software, ill equipped testers, etc. ISO 17025-2005 describes clear guidelines to achieve such a state which should be considered when undertaking the process. It is imperative the environment must be able to be recreated (To recreate testing is discrepancies identified). Therefore the documentation must be precise and blatant. A process to document and achieve this requirement will be created along with testing to ensure the environment is uncontaminated prior to the testing.

2. Select Test Scenarios

The selection of applicable test scenarios is potentially the most important component of validating forensic computing software. A process to identify functions, requirements and test cases is examined later in this paper. The ability to select appropriate scenarios to be tested requires experience and knowledge of the tester (and agency). As the assumption for this research is that all applications will be tested as closed source the variety of test types is reduced considerably and limited to black box testing.

3. Running and Evaluating the Test Scenarios

The undertaking of the tests as selected in stage 2 and executed in the environment as derived from stage 1 requires careful observation and consideration in preparation, execution and evaluation. The preparation phase ensures the tester is familiar with the objectives of test and interacts with the system in a prescribed and documented manner. It is assumed that this prescribed manner is obtained from documentation supplied with the software or determined in stage 2. Complete logging of the execution of the tests is required and ISO 17025-

2005 provides framework for this requirement. Results must be recorded to allow the evaluation against expected outcomes and judgments must be considered to determine whether the function/feature has achieved the desired outcome. Results evaluation is explored in a later section of this paper.

4. Measuring the Testing Process

Within the context of the Whittaker publication the measurement of the testing process suggests that a progression of the testing is monitored and updated to reflect the actual state at any one time. Within the context of the testing of forensic computing tools it is deemed that the measuring testing process will have an alternative use. As forensic computing tools have the potential to contain multiple functions it is naive to assume that the all functions should be tested collectively. The reasoning for this judgment is due to the fact that some obscure functions offer no benefit in the real world or the process of testing these functions is too complex to consider if the demand does not exist. Within the context of forensic computing tool evaluation, it is therefore proposed that the measuring testing process tracks the functions tested and displays identified functions which have not been tested (Whittaker 2000).

The Process of Evaluation

The process for evaluating software applications will consist of the following:

1. Acquirement of software
2. Identification of software functionalities
3. Development of test cases & reference sets
4. Development of result acceptance spectrum
5. Execute tests & evaluate results
6. Evaluation results release

It is imperative that all phases of the evaluation are transparent in the form of acquirement, testing, results reporting and evaluation. The process may consider input from a number of sources such as practitioners, vendors and academics to assist in each of the phases. An example of where such input will be beneficial is to determine the function(s) of a software application if no documentation is supplied, or is of substandard nature. The process of testing will generate documentation which then may be considered as a standard operating procedure (SOP) of types. This is just one such side effect of such a community engaging architecture.

The phases of evaluation are:

1. Acquirement of Software

The process of acquiring the relevant software applications to be evaluated is the beginning of the evaluation lifecycle. Once an application is 'acquired' the documentation begins satisfying ISO 17025-2005 and AS 4006-1992.

Once the initial documentation is acquired a signature of the software must be developed to ensure that any subsequent versions, patches or modifications are not misinterpreted as already tested. It is expected a MD5 checksum or similar could be used to generate such a unique signature.

2. Identification of Software Functionalities

Once the acquirement requirements are satisfied the identification of a set of the application's functionality is required. Only the identified functions will be tested.

If documentation is available when acquiring the software then this phase is much simpler. If no documentation is available then the tester must identify functions to be tested by whatever means available. This may include, but is not limited to, execution and trialling of the application, community input, vendor input or mining information about functions from other sources such as websites, discussion boards, guidelines etc.

A function is considered some component of the software which delivers a result when executed. This function can be complex in its nature or simple – but only functions identified in this phase will be tested and given an evaluation result at the completion of the subsequent phases.

If all functions are not identified within this phase the application can be retested at a later stage. Newly identified or untested functions can be appended to the test documentation.

All functions must be clearly documented as an item to be tested. This requires an experienced tester with knowledge of how the tools functions are utilised. It is imperative that any functions that act as a dependency to another function are documented as this will be reflected within the next phase of evaluation.

3. Development of Test Cases & Reference Sets

The development of test cases and reference sets is critical to determine the correctness of the application.

When all functions to be are identified during the previous phase, tests must be developed to determine the correct operation. The assumption has been made that all testing will be on applications treated as proprietary - access to the source code is not available. Therefore all tests will be based on black box testing techniques where the data is organised to test as many functions as identified into an image referred to as a reference set. The reference set will differ for each evaluation process determined by the functions being evaluated. Different applications with the same functions may use the same reference set.

Utilizing the black box testing technique the data in the test case is known and determined to be an example of real world scenario. The reference set will be developed by the tester and must not only test the real world example in isolation but must also test the true operation of the software. Examples of this may include encryption, modification of file headers, placing keywords across clusters, modifying data so that the expected format is not within standards – these are just a few examples of such tests that may be included.

Due to the critical nature of this testing phase input maybe sought from outside the test organization to determine what the real world case is and how to truly test the extent of the function. As with all phases the test phase reference sets will be made available so individuals can verify the software, and its functions, within their own environment.

It may become practice that once a reference set has been developed and documented this is shared with the community. The community can then assess whether the tests are appropriate and comprehensive enough to cover the real world usage. Request for comments would expire prior to the penultimate phase and all feedback would be considered before undertaking any subsequent phases of evaluation.

Test cases and reference sets not only are designed to test the functions identified but also the operation of the application as a whole. Focus is predominantly on determining the correctness of functions identified but it is acknowledged if the host software is incorrect then it must be determined whether this impacts on the functions operations. Therefore environment testing, hardware configuration, co hosted software and processes may be considered and tested to determine their impact. Such testing must be included within the documentation as faults or defects may directly, or indirectly, be attributed to the environment hosting the test bed.

As within the previous phase if a function is determined to be excluded on an evaluation cycle this can added at a later time – it is essential once again that any dependants on this function(s) are evaluated together.

4. Development of Result Acceptance Spectrum

Result acceptance spectrum is a metric used to assess the test results against what was expected. As the testing method is black box and the reference sets and test cases have been developed all results of tests are known. Dependant on the environment (law enforcement, military, civil, etc) of tool use determines the level of accuracy required. It is therefore expected that numerous levels of acceptable ranges will be identified allowing a greater number of practitioner domains benefit from the evaluation process.

ISO 14598.1-2000 provides a methodology to document the result acceptance spectrum by dividing the potential result set into 4 mutually exclusive groupings; exceeds requirements, target range, minimally acceptable and

unacceptable. How these groupings are determined must include previously undertaken phases including phases 2 & 3. From these phases the metric of acceptance and expected can be identified and documented – this may differ for example between law enforcement and military domains and this would require input from the respective communities.

If a function does not meet the acceptance range then that function and all dependant, and co-dependant functions, will be rendered as incorrect. Therefore it is essential that dependencies of functions are recognized in the prior phases as this may cause one function to a ‘pass’ evaluation status when coupled with the incorrect function will render the outcome as invalid.

5. Execute Tests & Evaluate Results

The execution of the tests is undertaken by the tester and each process of testing will be documented as per ISO 17025-2005 and AS 4006-1992 requirements. Any unexpected intervention required will be documented and assessed as this may bear impact with results.

All results from all tests will be collected and assessed against the acceptance spectrum compiled in the previous phase. All functions found to be below the acceptable range will be evaluated as ‘failed’ and documented. All functions found to achieve in, or above, an acceptance rating will be evaluated as ‘passed’ with all documentation included.

It is possible for a function to pass evaluation without passing all test cases. As described in phase 3 extraneous tests will be executed with the intention to identify limitations of the application. These will be included with any evaluation and have a significant component of the documentation exploring the limitation so all users can recognize this.

6. Evaluation Results Release

Upon conclusion of the previous phases all results must be made available to the community. Once a tool is evaluated it is not immune from future evaluation or re-evaluation.

Furthermore if the application is modified in any capacity (patches, updates, etc) the evaluation is not automatically inherited. Therefore software patches usually released by vendors to address identified defects will render their already evaluated application as unevaluated. If this evaluation framework is adopted it will force vendors to supply a product with a greater emphasis on quality to the consumer. If patches are frequent the use of an unevaluated tool may motivate a consumer to select a competitor’s product due to more ‘uptime’.

Discussion

As highlighted in the DFRWS Final Report 2005 (Reust 2006) the ability to test tools in a variety of environments is limited due to both the financial and time constraints. The current models of tool testing frameworks (NIST CFTT, etc) do not offer a solution to this as there methodologies are testing in isolation. Whittaker (2001) documented the invisible user in a paper highlighting the fact that it is difficult for test environments to truly recreate all potential live environments. Furthermore systems, particularly operating systems, are prone to react in unexpected manners and these actions are difficult to identify and simulate. Although software is deterministic in isolation once the interaction occurs with the host and other software the results can vary.

The evaluation framework proposed offers an evaluation of a software application identified as a forensic computing tool at one point of time in one environment. Therefore, by releasing the evaluation framework phases to all, this allows for individual organizations and practitioners to undertake the same testing process within their environment. This is referred to verification and their results should resemble the evaluation results which are referred to as validation results. If they do not correspond it must be determined if the error lies within the validation/evaluation process, the verification process or both. If the error is determined to exist in the validation process then the evaluation status of the software tool, and all functions, must be revoked immediately until revalidation is completed and verified.

With this knowledge it is imperative that tools are tested in the greatest combinations of environments as possible. A study being conducted by the authors of digital forensic laboratories currently within Australia highlighted that no two laboratories used examination workstations of the same specification. An example is some workstations using single core processors while others used dual core within the same organisation. Write blockers varied in brand and connectivity along with countless other hardware and software variations.

Other beneficial side effects of the proposed framework are that it offers a centralized agency the ability to register practitioners to create a community. This allows for the collection of input from numerous domains and this can be included within the validation process. Documentation maybe created, or extended, if new functions are not already documented and practitioners maybe exposed to a library of tools if these are free for use.

Further work

Establishing an Testing Authority

The formation and allocation of resources to implement the described evaluation process is required immediately. The funding for a project such as this should not be solely the onus of a government but should be a consortium of government, private industry and academia to put in place the processes to create such an authority.

Automated Reference Set Generator Tool

An automated tool to create mountable disk images with dynamic content is the concept behind this tool. When a tool is to be validated the functionalities are identified and represented in the preceding screens of the automated tool. Once all functionalities are included at the initial execution of the software the application generates a dynamic image creating custom reference set for the image and embedding these into a disk image. Upon completion of execution a unique identifier for an image is created and a corresponding report documenting all embedded test cases such as keywords, partition types, file structures, deleted files, slack space, cross cluster files, etc.

When this image is used for validation the corresponding report is available to testers to allow for a comprehensive assessment of the tool being validated. This will significantly improve the efficiency of the evaluation process; particularly phase 3 of the previously described framework.

The automated reference set generator tool can also be used by vendors to increase their testing domain with minimal increase in time and cost.

CONCLUSION:

The proposed evaluation framework utilizes black box testing techniques via reference sets and test cases to determine the correctness of software applications utilized as forensic computing tools. While the testing techniques are fully described it is important to describe the process that encompasses the tests to appreciate the benefit this offers to the forensic computing community.

The proposed framework has numerous benefits such as a streamlined process, community input, multiple environment testing and a community point of contact. It addresses shortcomings of other similar frameworks and extends their capabilities. It must be determined if such a framework will function and be a truly complete solution to the needs of the forensic computing community. Whether this framework is the solution or not the gap between tools and their evaluation is widening at an alarming rate.

REFERENCES:

AAFS. (1996) *What is Forensic Science?* [online], American Academy of Forensic Sciences, <http://www.aafs.org/>

Armstrong, C. (2003) *Developing A Framework for Evaluating Computer Forensic Tools*, Curtin University of Technology

AS 4006-1992 *Software Test Documentation*

Etter, B. (2001) *The Forensic Challenges of E-Crime*. Australasian Centre for Policing Research.

Grabowski P (1998) *Crime and technology in the global village*. Internet Crime, University of Melbourne, Victoria

Holzmann, G. *Economics of Software Verification*. Bell Laboratories

IEEE Std 610.12-1990 *IEEE Standard Glossary of Software Engineering Terminology –Description*

AS/NZ ISO/IEC 14598.1-2000 *Information technology - Software product evaluation - General overview*

AS/NZ ISO/IEC 17025-2005. *General requirements for the competence of testing and calibration laboratories*.

Meyers, M. & Rogers, M. (2004) *Computer Forensics: The Need for Standardization and Certification*. International Journal of Digital Evidence, Vol 3, Issue 2

Reust, J. (2006) *DFRWS 2005 Workshop Report*.

Whittaker, J (2000) *What is software testing? And Why Is It So Hard?* IEEE Software, January/February 2000.

Whittaker, J. *Software's Invisible Users*. IEEE Software May/June 2001.

Wilsdon, T. & Slay, J. (2005). *Digital Forensics: Exploring Validation, Verification & Certification*. First International Workshop for Systematic Approaches to Digital Forensic Engineering

COPYRIGHT

Wilsdon, Tom & Slay, Jill ©2006. The author/s assign SCISSEC & Edith Cowan University a non-exclusive license to use this document for personal use provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive license to SCISSEC & ECU to publish this document in full in the Conference Proceedings. Such documents may be published on the World Wide Web, CD-ROM, in printed form, and on mirror sites on the World Wide Web. Any other usage is prohibited without the express permission of the authors