Identifying Bugs In Digital Forensic Tools

Brian Cusack
Edith Cowan University

Alain Homewood
Auckland University of Technology

DOI: 10.4225/75/57b3c3befb86c
11th Australian Digital Forensics Conference. Held on the 2nd-4th December, 2013 at Edith Cowan University, Perth, Western Australia

IDENTIFYING BUGS IN DIGITAL FORENSIC TOOLS

Brian Cusack[1,2], Alain Homewood[1]
[1]Auckland University of Technology, Auckland, New Zealand
[2]Security Research Institute, Edith Cowan University, Perth, Australia
brian.cusack@aut.ac.nz, alain.homewood@aut.a.nz

Abstract
Bugs can be found in all code and the consequences are usually managed through up-grade releases, patches, and restarting operating systems and applications. However, in mission critical systems complete fall over systems are built to assure service continuity. In our research we asked the question, what are the professional risks of bugs in digital forensic tools? Our investigation reviewed three high use professional proprietary digital forensic tools, one in which we identified six bugs and evaluated these bug in terms of potential impacts on an investigator's work. The findings show that yes major brand name digital forensic tools have software bugs and there is room for improvement. These bugs had potential to frustrate an investigator, to cost time, to lose evidence and to require compensatory strategies. Such software bugs also have the potential for malicious exploitation and anti-forensic use.

Keywords
Bugs, Risk, Digital, Forensic, Tools, Work

INTRODUCTION

A software bug is weakness in a computer program either by code or design that produces an incorrect or unexpected result, or causes it to behave in unintended ways (Garfinkel, 2007). The research question regards the value of these vulnerabilities for anti-forensic hacks or the implications for the preservation and presentation of evidence (Hilley, 2007). Exploiting software bugs can occur in many ways. The focus of our interest was fuzzing exploitations. Fuzzing is the process of providing intentionally invalid data to an application in an attempt to trigger an error or fault condition of some kind. This type of activity can be classified as anti-forensic as the consequences can block evidence, counterfeit evidence, confound investigation, frustrate processes, and confuse analysis. Code execution is an integral part of software tool functionality and the associated vulnerabilities require securing. We used fuzzing to create malformed data structures through methods such as randomly replacing single bytes. In its simplest form fuzzing can consist of simply randomly replacing bytes in a data structure; at its most advanced it requires manipulating specific byte locations with knowledge of the properties of a data structure. We used a set of mutations that are designed to exploit typical programming mistakes commonly found in software. An example of one of these mutations is replacing a sequence of NUL bytes with random values of the same length. Fuzzing was performed on a number of file formats such as JPEG images and PDF documents with the goal of detecting problems with the built in file viewers in the forensic tools. Fuzzing was also performed on file system structures in an attempt to reveal issues with the methods used by forensic tools to interpret file systems (Sutton, Green, & Amini, 2001; Harris, 2006).

A second technique used was manual targeted manipulation of data formats. Targeted manipulation is the process of modifying specific portions of a data structure guided with detailed knowledge about the data structure. Two data structures were targeted for testing; individual files and file system structures. Individual files were targeted in an attempt to again locate issues with a tool built in file viewer. File systems and entire disk images were also targeted in an attempt to locate issues with the techniques used to analyse file systems. Function based software testing uses standardised and benchmarked input data but fuzzing addresses the residual risk inherent in such testing. Importantly we identified a number of bugs in several different types of tool and this report focuses on six of those bugs in one tool. The paper is structured to deliver a brief background on software bugs, the testing methods, the results and a discussion of the anti-forensic risks posed.

TOOL BUGS

Digital forensic investigators typically rely on one or two tools to conduct their investigation. The reliance on a small number of tools is because of costs, user confidence and the requirement in the community to have standardised tools that can be tested and confirmed to produce reliable results (Guo, Slay, & Beckett, 2009). Tool risk types fall into three categories: failure to validate data, denial of service attacks and fragile heuristics (Lui & Stach, 2006). The tools risk of improper input validation before performing a process leads to a common example of a technique that exploits software by a buffer overflow. A buffer overflow occurs when a program is writing data to a buffer in memory but overruns the buffers boundary and overwrites the adjacent memory area. The result of buffer overflow is that the program may exhibit erratic behaviour including crashes and memory access errors. In cases where the data being written to memory is under control of the user it may be possible for the user to control what code is currently being executed and to execute their own arbitrary code. Many investigators have probably experienced an unexpected crash or erratic behaviour when using a digital forensic tool; the crash is likely due a software bug in the tool that doesn't properly validate input data. One of the main reasons for the existence of software bugs in digital forensic tools is that digital forensic tools must be able to acquire data from multiple types of device and then analyse, search and display thousands of different data formats (Charters, 2009). Denial of service attacks also occur with tools and refer to the ability of an attacker to exhaust an available resource of memory and CPU time. Once the resource has been exhausted then the service provided by tool is denied meaning the specific forensic analysis task being performed stops. An example of a denial of service attack against digital forensic tools that is commonly referred to is "42.zip". 42.zip is a small zip file that is 42KB in size; however 42.zip contains multiple levels of recursively nested zip files inside of itself which when fully extracted contain 4.5PB of data. If the mounting process was to encounter 42.zip then the system would keep extracting until it ran out of resources such as hard drive space and memory.

Fragile heuristics refers to the processes digital forensic tools use to determine the type or structure of a data object. Essentially digital forensic tools often have to make educated guesses about what type of data they are processing or how data is structured. For example when a tool conducts a file signature analysis it first examines the file extension and file header and then performs a comparison with a list of known signatures. The risk of relying on heuristic processes is that they can be easily circumvented by tools such as Transmogrify in order to hide files. There is also the risk of a denial of service attack or file hiding techniques being possible through the creation of a large number of false positives. For example an attacker could create a large number of text files that start with "PK". Because text files have no header a file signature analysis a tool will report the text files as zip files and the large number of false positives could prevent functionality such as a file mount from working effectively or could divert the investigators attention from legitimate zip files containing relevant evidence (Carrier, 2002).

Tool related risks can be mitigated through two main approaches; firstly the use of multiple tools and secondly the production of better tools. The use of multiple tools is a simple solution however the cost in time and money of purchasing tools, training and performing the same work twice prohibits many investigators from being able to use multiple tools. The monetary cost factor can be reduced by the use of open source forensic tools which have come a long way in terms of functionality and usability in recent years. Open source tools also have the benefit of anyone being able to fix software bugs that pose an anti-forensic risk. The use of multiple tools also greatly helps mitigate the risk of improperly validated data. However there are only a limited number of ways for digital forensic tools to perform a task which results in tools sharing common methods and techniques; the end result being multiple tools that are vulnerable to the same denial of service and fragile heuristic attacks. Many anti-forensic techniques can be overcome by improving and fixing bugs in existing digital forensic tools. Denial of service attacks such as the use of 42.zip should be intelligently detected and handled by all tools. The heuristic systems behind processes like file signature analysis can be improved by looking beyond the header and footer of a file and the identification of known file structures within the file in order to identify its type (Hadnagy, 2010; Stamm, 2010).

TEST CRITERIA

We tested a range of proprietary and open source tools by adopting best practices from the literature. The testing data was deliberately constructed to dislodge bugs in the tools. Twenty malformed data sets of images, email, containers, internet artefacts, windows files and log files were used as input data. Acceptance spectrums adapted from literature were employed to determine the results of each test case (see Table 1).

Table 1: Default Acceptance Spectrum

Result	Acceptance Spectrum	Mapped Hypothesis
Pass	Exceeds expectations	H1: No software bugs are detected.
Pass	Meets expectations	H2: Software bugs are detected but they do not present an anti-forensic risk.
Fail	Unacceptable	H3: Software bugs are detected that present a minor anti-forensic risk.
Fail	Critically unacceptable	H4: Software bugs are detected that present a critical anti-forensic risk.

The acceptance spectrums contained a range of possible outcomes for each test case beyond a simple pass or fail result and further refinements were made to granulate (see Tables 2 & 3). A number of reference sets were created by first identifying a benign input file for a sub-function. A reference set was then generated by creating a number of malformed files based off of the input file. The benign files were either taken from an existing image of forensic corpora or by manually creating benign files and in this way the reference sets all contain various numbers of files at five specific malformation percentages being 0.1%, 0.2%, 0.5%, 1% and 2%. A range of malformation percentages was chosen to generate a wide range of possible malformations in the reference sets. One file type may generate an error condition at a malformation percentage of 0.1%; however another file type may only generate an error condition at a malformation percentage of 2%. These metrics gave further insight into expected performances.

Table 2: Summary of reference sets

Reference Set ID	Contents
RS-IMAGE-01	Malformed BMP, GIF, JPG and PNG images
RS-EMAIL-01	Malformed PST email containers
RS-EMAIL-02	Malformed NSF email containers
RS-EMAIL-03	Malformed MBOX email containers
RS-EMAIL-04	Malformed DBX email containers
RS-CONTAINER-01	Malformed ZIP file containers
RS-CONTAINER-02	Malformed GZIP file containers

RS-CONTAINER-03	Malformed TAR file containers
RS-CONTAINER-04	Malformed RAR file containers
RS-CONTAINER-05	Malformed BZIP2 file containers
RS-INTERNET-01	Malformed Firefox history/bookmark databases
RS-INTERNET-02	Malformed Internet Explorer history databases
RS-INTERNET-03	Malformed Opera history databases
RS-INTERNET-04	Malformed Safari history databases
RS-INTERNET-05	Malformed Chrome history databases
RS-WINDOWS-01	Malformed Windows Link files
RS-WINDOWS-02	Malformed Windows Recycle Bin (INFO2) records
RS-LOG-01	Malformed Windows Legacy Event Logs
RS-LOG-02	Malformed Windows Event Logs

Table 3: Test requirements

Requirement ID	Description
EC.EP.01	The “Thumbnail Creation” sub-function shall be able to handle malformed images without generating an error condition
EC.EP.02	The “Find Email” sub-function shall be able to handle malformed email files without generating an error condition
EC.EP.03	The “Expand Compound Files” sub-function shall be able to handle malformed compound files without generating an error condition
EC.EP.04	The “Find Internet Artifacts” sub-function shall be able to handle malformed internet artefacts without generating an error condition
EC.EP.05	The “Windows Artifact Parser” sub-function shall be able to handle malformed Windows artefacts without generating an error condition
EC.EP.06	The “Windows Event Log Parser” sub-function shall be able to handle malformed Windows event logs without generating an error condition

THE RESULTS

Our report here focuses on one tool as an illustration of what may be expected when testing digital forensic tools. The outcome shows that the tool performs inconsistently and across the different tools different performances were found. In this instance two tests exceed expectations and four were unacceptable on the adopted acceptance spectrum (see Figure 1). The question then arises as to the implications of such inconsistencies. We further resolved the tool tests into issues that were identified during testing (see Table 4). Principally a dominant set of occurrences showed no issues but disturbingly a greater number of occurrences reported problems for the uninterrupted use of the tool. These included a high number of crashes indicating that abstract complexities that could not be resolved by the software. In addition buffer over-runs were found in large files and a number of unexplained exits from analysis were noted. When challenged by the malformed input data internal errors occurred that either froze the screen or error messages were reported. These results show that fuzzing is able to disclose bugs within code and that the stability of digital forensic tools may be questioned.

Table 4: Summary of acceptance spectrum determinations

Test Case	Result	Acceptance Spectrum
TC.01	Pass	Exceeds expectations
TC.02	Fail	Unacceptable
TC.03	Fail	Unacceptable
TC.04	Fail	Unacceptable
TC.05	Pass	Exceeds expectations
TC.06	Fail	Unacceptable

Further analysis showed that throughout the testing there were four distinct types of issues were identified. The most common type of issue seen was a complete crash resulting in the Windows operating system presenting an error message. The error message indicates that the operating system has detected a fatal exception has occurred in the tool executable. Windows performed a memory dump of the application’s memory and then ended the executable. A crash has the potential to be a significant issue for an application and could result in anti-forensic risks such as code execution which could lead to compromising the system and evidence. In test case TC.03 while processing reference set RS-CONTAINER-01 the tool exited unexpectedly without an error message appearing from either the tool or Windows OS. An internal error message occurred during test case TC.06 while processing reference set RS-LOG-02. An internal error message is an indication that the tool has encountered an exception and has been able to handle it gracefully without needing to end the executable. In this particular case it remained in working state with reduced functionality. However, testing was abandoned during test case TC.02 while processing reference set RS-EMAIL-04 due to the creation of unusually large cache files. In the case of RS-EMAIL-04 the logical evidence files being created were exceptionally large. The issue seen with RS-EMAIL-04 is likely due to the manipulation of internal data structures in a DBX file by the fuzzing process. The main risk of the creation of large cache files is that an investigator will run out of room to store the cache files and the evidence processing may need to be cancelled and repeated. This again is a time cost.

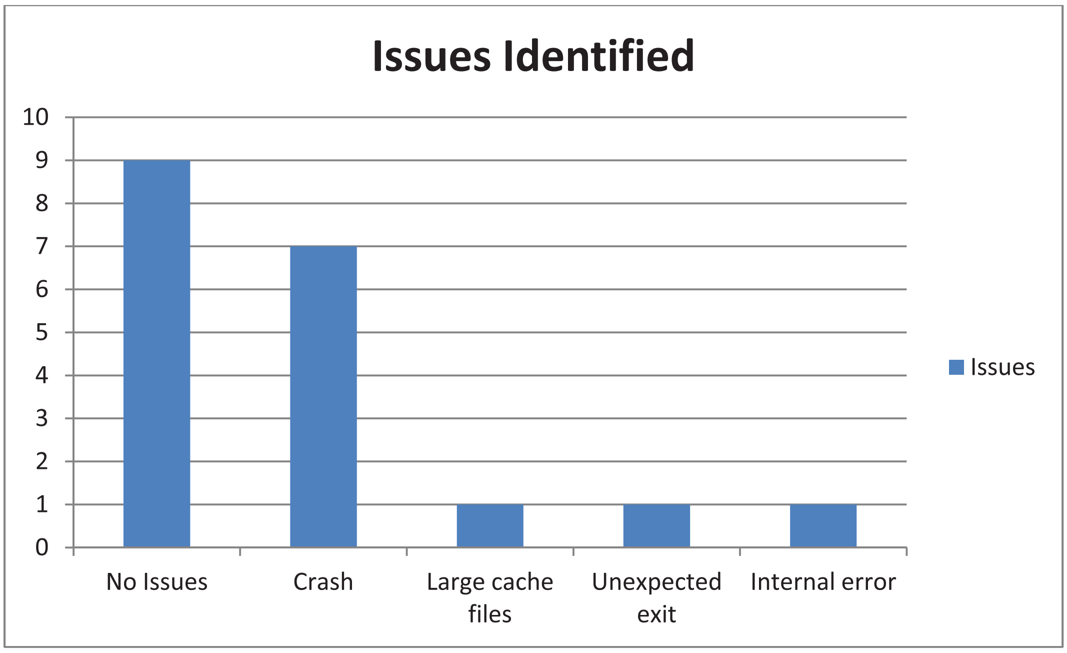

Figure 1: Summary of types of issues

IMPLICATIONS AND COUNTERMEASURES

The acceptance spectrum concept was adopted to determine the severity of anti-forensic risk and to assure that anti-forensic risks are not incorrectly judged to be harmless or critical due to insufficient differentiation. The hypothesis testing in the case revealed that the majority of the test cases performed were determined to be unacceptable. Essentially the software bugs detected have been determined to fall in the middle of the acceptance spectrum and present a minor anti-forensic risk. The result is good from the point of view that the software bugs have not been incorrectly labelled as harmless or critical. However the result is ultimately unsatisfying in determining the actual anti-forensic implications. The acceptance spectrum result does nothing to inform investigators if they should be concerned about a particular software bug. For example one of the bugs was analysed further and shown to be a code execution vulnerability while the rest of the bugs remained as minor risks. Care should be taken to understand if a risk has been accurately represented or if further analysis could determine if the risk has been under or over stated.

The most prevalent issue identified in the case was seven instances of crashing where the tool suffered a fatal error that it couldn't recover from. Two instances of the similar issues of internal errors and exiting unexpectedly were also identified. A suspect knowing they are about to be investigated and knowing the basics of the forensic process might deliberately plant several malformed compound files on his computer. An investigator could spend a considerable amount of time running tools only to encounter a deliberately malformed compound file that causes a crash. The investigator has now lost a significant amount of processing time and has been prevented from using the Evidence Processor functionality to automatically expand compound files. If the investigator was to try using the Evidence Processor again they would encounter another crash and lose even more time. At this stage the investigator is forced to take counter measures to handle the malformed compound files. The simplest option would be to manually expand the compound files one by one so that less work is lost if a malformed compound file is encountered. Another approach is to use the debugging features of the software and work with the software vendor to identify and fix the software bug. As an immediate solution the software vendor could possibly assist in identifying the problem files so that they can be isolated. A third option is to use an alternative tool to perform the analysis on the exhibit with the hope that different tools will have different software bugs.

By analysing this crash scenario we can identify a number of different types of risk factors present. The most obvious risk factor is a tool risk that was due to a failure to validate data. The digital forensic tool has failed to validate the data it is processing from the compound file and this has resulted in a fatal error occurring. There is also an element of process risk that shows reliance on standardised processes makes it easier to target anti-forensic attacks. Expanding compound files is going to be part of many people's forensic process and therefore it becomes a promising function to target with an anti-forensic attack. A number of counter measures were presented in the scenario however these counter measures are dependent on human risk factors and an investigator who hadn't encountered similar issues before might simply keep rerunning the tool and expect it to work the second time. The software bugs identified that caused a crash, unexpected exit or internal error present an anti-forensic risk that is easily identifiable by an investigator. When the tool crashes in the middle of running the Evidence Processor it's obvious that something has gone wrong and some action needs be taken to remedy the situation. There was one software bug identified that isn't as obvious. The "Find Email" function of the Evidence Processor results in the creation the of large cache files. The "Find Email" function parses email container files and extracts the individual emails out to a logical evidence file to allow for further analysis. A software bug was identified with the processing of DBX email containers which resulted in unusually large logical evidence files being created. In the test case the reference set of 3 GB of DBX email containers expanded to full a 1.8 TB drive before the process was cancelled after 24 hours of processing time.

If such files were to be encountered in actual investigation there would be no indication of what had happened until they run out of available hard drive space. The anti-forensic implications are similar to a crash in that the investigator loses a significant amount of time. However the loss of time is potentially much larger as the investigator is not immediately notified that something has gone wrong. Also the investigator has to spend additional time diagnosing the issue to figure out that the tool has created large cache files and then to clean up the consequences. The investigator can use similar counter measures to those used to counter a crash however the creation of large cache files falls into the category of denial of service attacks where the investigator is being prevented from using a resource. The software bugs identified do not currently pose a significant threat to evidential integrity or the security of examiner machines. Although there may be potential for more severe anti-forensic risk the only demonstrable risk is that of disrupting or preventing investigations from occurring. There are viable counter measures available to the anti-forensic risks identified however these may be expensive in terms of examination time and cost of additional resources and tools.

CONCLUSION

Our research question arose because of an unproven concern that bugs in software could be exploited for anti-forensic activity. A further worry was the potential mis-representation or damage to evidence by vulnerabilities in both open source and proprietary tools. The research ran a number of test cases in which deliberately malformed data was input into digital forensic tools in an attempt to locate software bugs. The software bugs once located were then analysed and the potential for different exploitations identified. We were successful in discovering a number of software bugs that resulted in unusual behaviour from different tools including behaviours that prevented evidence acquisition, crashing while searching or displaying incorrect evidence as well as evidence not being displayed.

REFERENCES

Carrier, B. (2002, October). *Open Source Digital Forensic Tools: The Legal Argument.* Retrieved February 20, 2012, from http://www.cs.plu.edu/courses/netsec/arts/osf.pdf

Charters, I. (2009). *The Evolution of Digital Forensics: Civilizing the Cyber Frontier.* Retrieved 01 11, 2012, from A Perspective on the History of Digital Forensics: http://www.guerilla-ciso.com/wp-content/uploads/2009/01/the-evolution-of-digital-forensics-ian-charters.pdf

Garfinkel, S. (2007). Anti-Forensics: Techniques, Detection and Countermeasures. *Proceedings of the 2nd International Conference on Information Warfare & Security* (pp. 77-84). Monterey, California: Academic Conferences Limited.

Guo, Y., Slay, J., & Beckett, J. (2009). Validation and verification of computer forensic software - Seaching function. *Digital Investigation*, S12-22.

Hadnagy, C. (2010). *Social Engineering: The Art of Human Hacking.* Indianiapolis: Wiley Publishing, Inc.

Hilley, S. (2007). Anti-forensics with a small army of exploits. *Digital Investigation, 4*, 13-15.

Harris, R. (2006). Arriving at an anti-forensics consensus: Examining how to define and the control the anti-forensics problem. *Digital Investigation, Volume 3*, S44-S49.

Liu, V. T., & Stach, P. (2006). Defeating Forensic Analysis. *CEIC 2006.*

Stamm, M. C., Tjoa, S. K., Lin, S. W., & Liu, R. K. (2010). Anti-forensics of JPEG compression. *2010 IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)* (pp. 1694 - 1697). IEEE Conference Publications.

Sutton, M., Greene, A., & Amini, P. (2007). *Fuzzing: Brute Force Vulnerability Discovery.* Upper Saddle River, NJ: Addison-Wesley.