Testing the Robustness of Windows NT Software*

Anup K. Ghosh, Matt Schmid, & Viren Shah
Reliable Software Technologies Corporation
21515 Ridgetop Circle, #250, Sterling, VA 20166
phone: (703) 404-9293, fax: (703) 404-9295
email: {aghosh,mschmid,vshah}@rstcorp.com
http://www.rstcorp.com

Keywords: black-box testing, robustness gaps, COTS software, test case generation, fault tolerance

Abstract

To date, most studies on the robustness of operating system software have focused on Unix-based systems. This paper develops a methodology and architecture for performing intelligent black-box analysis of software that runs on the Windows NT platform. The goals of the research are three-fold: first, to develop intelligent robustness testing techniques for Commercial Off-The-Shelf (COTS) software, second, to benchmark the robustness of NT software in handling anomalous events, and finally, to identify robustness gaps to permit fortification for fault tolerance. The Random and Intelligent Data Design Library Environment (RIDDLE) is a tool for analyzing operating system software, system utilities, desktop applications, component-based software, and network services. RIDDLE was used to assess the robustness of native Windows NT system utilities as well as Win32 ports of the GNU utilities. Experimental results comparing the relative performance of the ported utilities versus the native utilities are presented.

1 Introduction

Windows NT is rapidly becoming the development, engineering, and enterprise platform of choice. Windows NT systems are replacing Unix systems on the engineer's desktop because of the large suite of professional software development tools available for NT, the support for common desktop applications such as word processing and spread sheet software, and finally the low cost of the machines. Recent data shows that roughly twice as many workstation owners bought Windows NT compared to Unix this year. Windows NT shipments are 80% above last year, while Unix shipments are down 7 percent [1]. Despite the proliferation of NT Workstations in enterprise and sometimes mission-critical environments, little analysis of the software that comprises the NT platform has been performed. As a result, the decision to employ NT-based systems in critical applications is based on anecdotal evidence or trust in the software vendor rather than based on scientific study.

To date, analysis on the reliability of operating systems has been performed for commercial and free software variants of Unix. The next section summarizes the prior art in studying the reliability of operating system software. The purpose for studying the reliability of the operating system and its associated software is to determine to what extent confidence can be placed in the platform for running enterprise and mission-critical applications. A secondary goal is to identify potential gaps in robustness of the operating system or application software that may result in unreliable operation.

In this paper, an approach and environment for analyzing the robustness of Windows NT software are developed. The environment permits stress testing of application software, system utilities, COM/DCOM components, shared libraries, and system functions. The Random and Intelligent Data Design Library Environment (RIDDLE) enables analysis of commercial off-the-shelf (COTS) software by using black-box testing techniques. Unlike traditional black-box testing approaches, the approach developed here is to "stress test" software with unexpected, intelligently crafted test cases. The goal of this research is to determine what robustness gaps, if any, exist in Windows NT software. This paper presents the RIDDLE architec-

*This work is sponsored under the Defense Advanced Research Projects Agency (DARPA) Contract F30602-97-C-0117. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

ture and results from analyzing one class of NT software — system utilities. The experimental study compares the reliability of native Windows NT utilities to that of the Cygnus Win32 port of the widely distributed GNU utilities. The GNU utilities, known for their high reliability, have been ported from Unix to Windows NT. This study sheds light on the reliability of ported software before and after the port. The Fuzz research project, described next, performed a study of the reliability of the GNU utilities before they were ported to the Windows NT platform.

2 Prior art

Two research projects have independently defined the prior art in assessing system software robustness: Fuzz [5] and Ballista [3]. Both of these research projects have studied the robustness of Unix system software.

Fuzz, a University of Wisconsin research project, studies the robustness of Unix system utilities. Fuzz merely subjects a program to random input streams. The criteria for failure is very coarse, too. The program is considered to fail if it dumps a `core` file or if it hangs. After submitting a program to random input, Fuzz checks for the presence of a `core` file or a hung process. Though the methodology employed by Fuzz is simple, the results from their study were quite revealing [4, 5]. The 1995 study [5] found that the failure rate of the Unix utilities they tested were between 18% and 23% — down only a few percentage points from 25% - 33% failure rate documented in the 1990 study [4].

Ballista, a Carnegie Mellon University research project, studies the robustness of different Unix operating systems to handling exceptional conditions. Unlike the Fuzz research, Ballista focuses on assessing the robustness of operating system calls made frequently from desktop software. Rather than generating inputs to the application software that made these system calls, Ballista generates test harnesses for operating system calls that allow generation of both valid and invalid input.

The goal of the work presented in this paper is to assess the robustness of software commonly used on the Windows NT platform. By first identifying potential robustness gaps, this work will pave the road to isolating the source of potential unreliability in the Windows NT system.

3 RIDDLE

The Random and Intelligent Data Design Library Environment (RIDDLE) is an environment that was created for testing the robustness of COTS software on Windows NT systems. Because RIDDLE is used for analysis of COTS software, no access to source code is assumed. The environment uses intelligent black-box testing techniques to generate input for the application being tested. Because the goal of the testing is to assess the *robustness* of the software being tested, the input generated can be characterized as "anomalous". That is, the input generated by RIDDLE in most cases falls outside of the normal operational profile for the software being tested.

Test cases are generated with random, intelligent input using the input grammar of the component under analysis. Rather than simply generating random input that does not meet the basic syntax of the program's input, generating input intelligently using the input grammar of the component permits stress testing of more of the software's functions. Otherwise, simply testing a program with random, unstructured input tests little more than the program's input handler.

RIDDLE provides an environment to combine random input, malicious input, and boundary value conditions in the legal grammar of the program to test its behavior more thoroughly under anomalous conditions. In order to exercise more of a program's functionality and to test more of the program's response to anomalous input, RIDDLE makes use of syntactically correct test templates. A library of data generation functions is used to fill the test templates with different forms of anomalous data. The result is a syntactically correct usage of the program using anomalous input.

Each test template represents a different, but valid, usage pattern. For example, one usage of the `cp` utility is to copy one file to another file. One test template for this usage would look like "`cp` $SWITCHES $FILE_NAME $FILE_NAME". Each string that begins with a '$' (called a token) is a call to a function within the data generation library.

The grammar generator takes as input a definition of the program's input grammar written in a format similar to Backus-Naur Form (BNF). In addition to specifying production rules, the grammar definition can also specify the likelihood that a particular production rule will be chosen. The grammar generator uses a random number generator to produce random, but syntactically correct, test templates.

RIDDLE does not use an oracle for its analysis. That is, RIDDLE will not reveal whether the component executed correctly. Rather, RIDDLE uses assertions of incorrect exit codes, unhandled exceptions, hung processes, insecure behavior (such as wip-

ing files), or system crashes to determine whether a component is robust to anomalous input. The software component under test can be as fundamental as a system function in the NT operating system or as complex as a desktop application with a graphical user interface. RIDDLE provides the set of input generation functions to drive these components given an interface specification.

4 Experimental analysis

RIDDLE was used to perform robustness tests on two categories of Windows NT software. The first category is made up of Windows NT command line utilities that are supplied with the operating system. The utilities tested are `attrib`, `chkdsk`, `comp`, `expand`, `fc`, `find`, `help`, `label`, and `replace`. The second category of software that was tested was a group of GNU command line utilities that have been ported to the Windows NT operating system as part of the Cygnus GNU-Win32 project.[1] The ported GNU utilities tested are `cat`, `chmod`, `chksum`, `cp`, `ls`, `mv`, `rm`, and `wc`.

The GNU software was selected partially because of the results from the Fuzz analysis [5] and also because it represents a sample of software ported from one platform to another. In the Fuzz analysis, the GNU software fared the best in robustness testing compared to other commercial implementations of similar utilities. Because GNU tools are written and maintained by world-class programmers, this software should serve as the baseline for how robust good software can be. Notwithstanding the high caliber of its developers, it is important to note that the GNU tools were written for the Unix platform — not Windows NT. The testing results from Fuzz for GNU tools do not necessarily warrant their reliability on other platforms such as the Window NT platform. This study may shed light on the effects of porting reliable software on the reliability of the ported software.

The experimentation covered all combinations of the string lengths and character sets described above. In all, there were 64,000 tests run on the GNU utilities, and 114,000 tests run on the native Windows NT utilities. RIDDLE detected distinct termination states from the programs that were tested. The exit states determine when the program terminates normally, when the program is hung, and when the program terminates due to an unhandled exception. Three types of exceptions were caught by the RIDDLE monitor in these experiments: memory access violation exceptions, privileged instruction exceptions, and illegal instruction exceptions. If these exceptions arise during the execution of a program, then the program has failed to perform robustly by failing to handle the exception internally.

[1] See http://www.cygnus.com/misc/gnu-win32/ for the distribution.

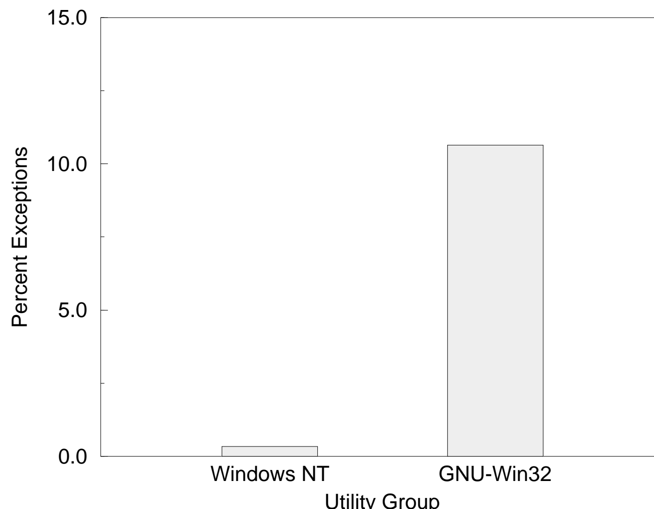

Figure 1: Percentage of unhandled exceptions for all test cases run against the native Windows NT and GNU-Win32 utilities. The vast majority of unhandled exceptions were memory access violations that result in the aborted execution of the program being tested.

Figure 1 summarizes the results of the testing of native Windows NT utilities and the GNU-Win32 utilities. In all the test cases run against the native Windows NT utilities, only 0.338% of the test cases resulted in failure according to our failure metric. On the other hand, the GNU-Win32 utilities exit with an unhandled exception 10.64% of the time. The distribution of exceptions favored memory access violations so heavily (approximately 7000 to 1 for GNU-Win32, and 100 to 1 for Windows NT) that the other types of exceptions are statistically insignificant.

Further analysis of the GNU-Win32 results show that the 10.64% failure rate is fairly consistent across the eight GNU utilities that were tested. The vast majority of the exceptions occurred when the character set being used for string generation was in the range [1,255] (excluding the NULL character). This is most likely due to the program's interpretation of special characters. The number of exceptions decreases dramatically when the character set is altered to include the null character, or when it consists only of printable characters in the range [33,127]. The null character may be interpreted as the termination char-

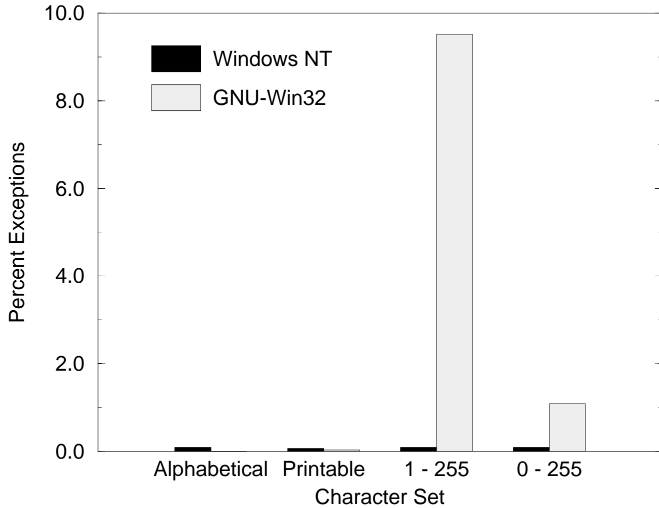

Figure 2: Distribution of unhandled exceptions among different content types. The character range [1,255] includes all characters except the NULL character. The range of the last column, [0,255], includes the NULL character.

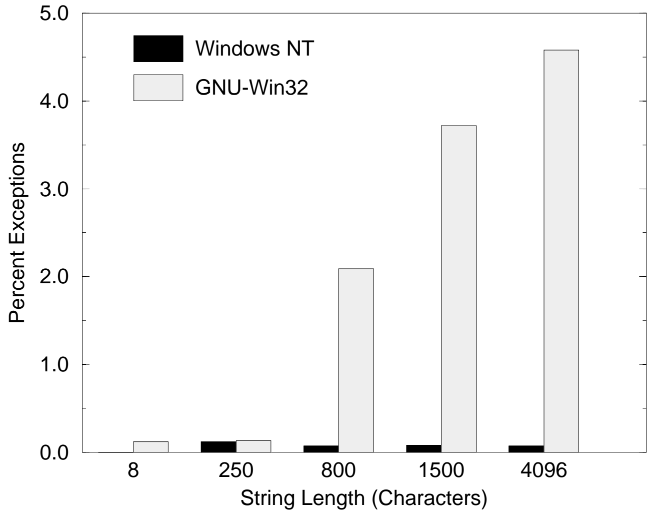

Figure 3: The percentage of test cases that resulted in exceptions as a function of the length of the input strings.

acter of a string, effectively limiting the length of the input. This would explain why there are fewer unhandled exceptions when this character is used in light of the correlation between length and exceptions (see Figure 3). Another possibility is that if the null character is interpreted as either the end of a string or the end of the parameter list, then the parameters may no longer constitute a valid use of the application and the utility may immediately reject the test case.

The distribution of exceptions by content type is shown in Figure 2. Clearly, the GNU-Win32 utilities are most vulnerable to input that is sampled from the character set range [1,255]. This set includes every printable and non-printable character except for the NULL character. Even very long length input in the alphabetical and printable set resulted in few exceptions. Instead, it is the combination of very long length with nearly the entire range of the character set (including non-printable characters) that resulted in the most unhandled exceptions.

The most significant trend in the data collected from the tests performed on the GNU-Win32 utilities is the increase in unhandled exceptions as the length of string increases as illustrated in Figure 3. The graph of the GNU-Win32 exception ratios show that as the length of input is increased from 8 to 4096 bytes, the number of exceptions rises dramatically indicating a failure to handle anomalous input within proper input grammar. Significantly fewer exceptions occurred when the length of the string used was either 8 or 250 characters. Because the exception that occurred most often was a memory access violation, the cause is most likely an over-written buffer that placed an illegal pointer on the program stack. In other words, the instruction pointer that was overwritten with the long input probably points to a region of the memory that is inaccessible for the program, or it may point to data that is not a valid instruction opcode. This result points to potential vulnerabilities in the GNU utilities to buffer overrun attacks. Buffer overrun attacks are one of the most significant security-related flaws that are most often exploited in practice [2, 6]. The Fuzz study also pointed out the relative vulnerability of programs to unconstrained input [5]. However, the assertion that these programs are vulnerable to buffer overrun attacks has not been investigated in this study.

The data collected from the tests run on the native Windows NT utilities paints a very different picture. Of the nine utilities that were tested, only two of them, `comp` and `expand` produced any exceptions. The `expand` utility had a failure pattern similar to the GNU-Win32 utilities. It failed more often when the strings were longer and the character sets were more complex. The `comp` utility failed most frequently when the character set was alphabetic, and the string length

was 250.

Further investigation of the experiments performed on the `comp` utility demonstrated that RIDDLE's intelligent data generation component exercised sections of the program that could not be tested using simply random (unintelligent) data generation. The strategy of combining valid and anomalous data together in a syntactically correct application usage was responsible for many of the exceptions that were being thrown by the `comp` utility that compares the content of two files. One test case that caused `comp` to fail is using valid and invalid file names for the program's arguments. When the data generator produces a valid file name for the first argument and an invalid file name with an alphabetic string of 250 characters for the second argument the utility terminates with a memory access violation exception. This exception is not generated when the data generator produces an invalid file name with less than 250 characters or greater than 255 characters. Furthermore, the first argument to `comp` must be a valid file name for this exception to be generated. In all other cases, the program exits gracefully with a standard "file not found" type of error. In a random testing approach, legitimate file names would not be constructed for either argument leaving this failure uncovered. This result is anecdotal evidence that the mix of valid and anomalous data has tested a portion of the `comp` utility that could not be tested using only random data.

5 Conclusions

This paper describes an environment and approach to testing Windows NT software for robustness to unexpected, anomalous input. The importance of robustness testing was established by prior research on Unix-based systems using Fuzz and Ballista testing environments. To our knowledge, RIDDLE is the first research tool to be applied outside of the operating system vendor to test the robustness of NT software.

The principle employed by RIDDLE is to stress test software with syntactically valid, yet anomalous input. To this end, an intelligent data generator that supports the target component's input grammar together with random data generators is developed. In addition, a monitor component that observes unreliable, unsafe, or insecure behavior is also used during the testing.

To date, RIDDLE has been applied to native Windows NT and GNU Win32 utilities. The results show that the native Windows NT utilities had far fewer failures to anomalous input than the GNU Win32 utilities. This result is contrary to some of the other published results in analyzing the reliability of GNU utilities against commercial software [4, 5]. One significant distinction, however, is that the GNU utilities were written for the Unix platform and ported to the Windows NT platform. The results indicate that a utility that may be highly reliable on one platform may suffer from problems in porting to a different platform.

These results represent testing of one class of Windows NT software — system utilities. RIDDLE is currently being developed to support testing of network servers, shared libraries, desktop applications, and OLE/COM/DCOM components. Future research will involve testing these other classes of NT software as well as exploring robustness gaps to determine their potential to be exploited into security holes.

References

[1] M.R. Anderson. News bulletin. Online. http://www.tapsns.com, February 3 1998.

[2] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Symposium*, pages 63–78, San Antonio, TX, January 1998.

[3] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz. Comparing operating systems using robustness benchmarks. In *Proceedings of the 16th IEEE Symposium on Reliable Distributed Systems*, pages 72–79, October 1997.

[4] B.P. Miller, L. Fredrikson, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, December 1990.

[5] B.P. Miller, D. Koski, C.P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. Fuzz revisted: A re-examination of the reliability of UNIX utilities and services. Technical report, University of Wisconsin, Computer Sciences Dept, November 1995.

[6] E.H. Spafford. The Internet worm program: An analysis. *Computer Communications Review*, 19(1):17–57, January 1989.