Why PCs Are Fragile and What We Can Do About It: A Study of Windows Registry Problems

Archana Ganapathi
University of California
Berkeley, CA

Yi-Min Wang
Microsoft Research
Redmond, WA

Ni Lao and Ji-Rong Wen
Microsoft Research
Beijing, China

Abstract

Software configuration problems are a major source of failures in computer systems. In this paper, we present a new framework for categorizing configuration problems. We apply this categorization to Windows Registry-related problems obtained from various internal as well as external sources. Although infrequent, Registry-related problems are difficult to diagnose and repair. Consequently they frustrate the users. We classify problems based on their manifestation and the scope of impact to gain useful insights into how problems affect users and why PCs are fragile. We then describe techniques to identify and eliminate such Registry failures. We propose health predicate monitoring for detecting known problems, fault injection for improving application robustness, and access protection mechanisms for preventing fragility problems.

1. Introduction

Windows-based personal computers (PCs) offer a platform for interesting and complex software interactions. However, undisciplined uses and sharing of persistent configuration data by Windows programs have made PCs more vulnerable to fragility. In particular, the Windows Registry stores large quantities of complex, undocumented and unprotected configuration data, making it the single most vulnerable component of the Windows operating system. Understanding and undoing Registry damage is a non-trivial task to the average user. The problem is as burdensome to system and application developers as it is to users.

In this paper, we characterize how Registry problems impact users and what can be done to alleviate and prevent future occurrences. We base our analysis on two sets of real-world failure data. The first set consists of 100 common Registry problems from a database of Product Support Services (PSS) email logs. The second dataset comprises 100 problems encountered by our colleagues and problems posted on various Web forums.

Through analysis of 200 cases, we develop a categorization framework that covers two orthogonal axes. The first dimension uses *problem manifestation* and *scope of impact* to understand the user's view of PC fragility. The second dimension suggests *monitoring techniques*, *fault injection tactics*, and *access protection mechanisms* to alleviate PC fragility.

An extended version of this paper including a survey of related work and monitoring tools is available in [1]. The remainder of this paper is organized as follows. Section 2 provides an overview of the Registry. Section 3 describes two fragility data sets. Sections 4 and 5 categorize problems based on manifestation, impact and mitigation techniques. Section 6 concludes.

2. Registry Overview

The Windows Registry provides a hierarchical, shared repository for named and typed configuration data. This data is accessed by the operating system as well as application software. The Registry is divided into *Registry hives* that contain sets of keys, sub-keys and items pertaining to components such as System, Software and Hardware. Registry hives distinguish between various per-user and system-wide settings. Each *Registry key* in a hive can be viewed as a directory that optionally contains sub-keys and *Registry items* that possess the actual configuration information as typed data. Each Registry item is accessed by navigating through a designated path of keys and sub-keys.

3. Fragility Data Sets

We collected data using two methods. First, we used text-mining tools to extract data from PSS logs. As logs contain limited root cause analysis information, our second dataset comprised problems encountered by our colleagues and Web forums users. We used the Strider Troubleshooter [3] to diagnose/reproduce these problems.

3.1. Text-Mined Data Set (TMDS) from PSS

We analyzed problems reported by emails to a PSS organization during several years spanning 3/20/1997 to 5/20/2003. PSS is a technical support organization that helps customers resolve problems by maintaining a knowledge base of known problems and solutions. Since

these email case logs do not explicitly identify Registry-related problems, we used text-mining to automatically extract potentially relevant cases and then manually eliminated cases with insufficient information.

A total of about 2,400,000 problems were reported during this time period, of which 101,900 (4.4%) contained references to a total of approximately 143,157 Registry keys/items. These references simply meant a Registry entry was present in the problem report, not necessarily that it was attributed as the cause for the problem itself. Only 5,266 unique entries were identified from these reports as multiple references to the same Registry entry could be present across various problems. We only chose problems with sufficient information for analysis, particularly the post-mortem summary descriptions provided by the PSS engineers. We extracted 10,405 summary-containing problems.

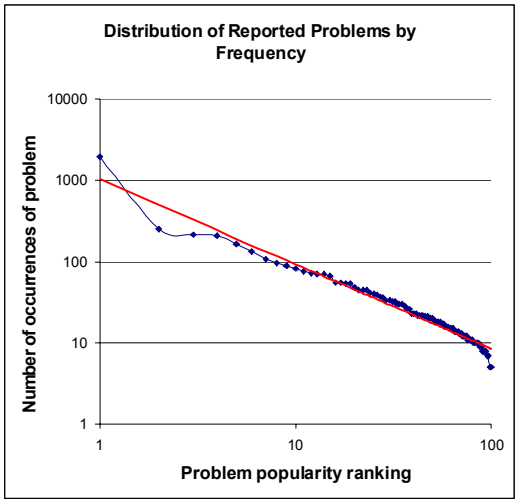

Figure 1: This graph presents a Zipf-like distribution of 100 most common problems from TMDS. Note that both X and Y axes follow a logarithmic scale.

We grouped multiple repetitions of problems by root cause Registry entry (as identified by text-mining tools) and selected 100 most common problems to analyze. The data approximately followed a Zipf distribution [2]. As shown by figure 1, most problems were infrequent while a small number of problems impacted a significant number of users. The number of occurrences of each selected problem ranged from 1,947 to 5. The top 100 cases represent approximately 5,379 (more than half) of 10,405 problems.

3.2. Strider-Verified Data Set (SVDS)

To further understand Registry fragility problems and experience their manifestation, we collected, reproduced, and resolved Registry problems using the Strider Troubleshooter [3, 4]. This tool performs a "diff" operation of two Registry snapshots (e.g. pre-problem good state and post-manifestation bad state), and intersects it with a trace of operations leading to problem manifestation. Finally, it provides a report that ranks the potential causal entries based on their likeliness of being the actual root cause.

We collected 100 problems from our colleagues as well as a helpdesk organization and Web forums such as TweakXP.com[1] and Registry Guide for Windows[2]. We specifically focused on problems relating to Windows XP SP1. For problems with known root causes, we reproduced the mis-configuration to study the symptoms. For problems without known solutions, we used the Strider Troubleshooter to identify the root causes and then manually verified the repairs. As there was no record of the number of people encountering and resolving each problem, we cannot report frequency.

4. Why PCs Are Fragile

4.1. Problem Manifestation

PC fragility manifestation on users' machines can be classified into seven distinct categories, enumerated below. In cases where a problem had several symptoms that mapped simultaneously into more than one category, we selected the most representative/impacting symptom.

(1) Unstable/unusable system – Certain Registry mis-configurations cause severe loss of critical functionality and/or open avenues for the system to be compromised. For example, changing the data of *HKEY_LOCAL_MACHINE\software\Microsoft\windows NT\currentversion\Winlogon\Userinit* from "C:\WINDOWS\system32\userinit.exe," to "C:\WINDOWS\system32\userinit.exe;" prevents user login to the machine, making the system unusable.

(2) Cannot perform a function or action – Sometimes, a user is unable to perform a desired task such as sending e-mail or invoking a program. For example, if *HKEY_CLASSES_ROOT\.mp3\PerceivedType* has data value other than "audio" (e.g. "text"), "Play All" in the "My Music" folder does not play .mp3 files.

(3) Unanticipated response – Users are often confused and frustrated when their action causes a new unanticipated response. For example, double clicking a folder opens a search-results window instead of the folder itself, when the *HKEY_CLASSES_ROOT\Directory\shell\(Default)* Registry item's data is empty.

(4) Unanticipated side-effect – Some side-effects are caused by bad program design. Moreover, what is acceptable within an application's specification can be an unpleasant side-effect to the user. For example,

[1] http://www.tweakxp.com
[2] http://www.winguides.com/forums

installing a new CD burner renames the "A" drive to "H" drive and produces an error when a user types "A:\" from Start menu→Run to open the floppy drive.

(5) Cannot locate user interface to perform a task – Sometimes, a user interface is concealed below several levels of menus. For example, Internet Explorer asks if it should save a user's password to a particular website. Once the user selects the *"Don't offer to remember any more passwords"* option in the pop-up dialog box, it is difficult for the user to restore Internet Explorer's settings so that it prompts to remember passwords.

(6) User interface disappears but functionality is preserved – In some cases, the user interface for a task is either absent or tainted. However, this task can be executed by other means such as command-line invocation. For example, a user may be unable to access Internet Options from Internet Explorer. However, the same configuration set is accessible from the Control Panel. Similarly, *Control Panel→Network Connections* may reveal nothing if a UI-related key is deleted, but all network connections remain intact.

(7) Program adaptation or automation is performed in an unexpected manner – Automation decisions made by some programs can be unintuitive to the user. For example, on some laptops, Microsoft PowerPoint fails to display a slide show when it incorrectly assumes, based on Registry information, that there exists a second monitor to which it should direct the video output.

Problem Manifestation	TMDS	SVDS
Unstable/unusable system	2 (143)	6
Cannot perform function/action	62 (4212)	32
Unanticipated response	18 (676)	23
Unanticipated side-effect	9 (196)	14
Cannot locate UI	1 (16)	9
UI disappears, functionality ok	3 (65)	12
Unexpected program adaptation	5 (71)	4

Figure 2: This table summarizes the manifestation categorization for the two data sets. The frequency of problems (out of 5,379) in TMDS is parenthesized.

The data in Figure 2 reveals that user inability to perform a function/action was the dominant form of reported problems. Among remaining categories, "Unanticipated response" has the highest case count. Intuitively, such problems are more frustrating to users than customization-related nuisances and user interface issues. Finally, the category of "Unstable/unusable system" contributes a non-trivial amount of problems; each problem needs to be carefully investigated as they cause a high degree of user frustration.

4.2. Scope of Impact

This category captures the impact of Registry problems on a machine's functionality:

(1) Impact Scope I – The impact can be system-wide or affecting only a particular user.

(2) Impact Scope II – A problem can impact a single application (e.g., Internet Explorer), multiple applications (e.g., all Microsoft Office applications) or the entire system. This information enables us to provide feedback to appropriate development groups (OS or applications) on potential tribulations to consider.

Impact Scope I	TMDS	SVDS
System-wide	71 (4312)	59
User-specific	29 (1067)	41
Impact Scope II		
Single Application	28 (994)	48
Multiple Applications	31 (3081)	16
System Level	41 (1304)	36

Figure 3: This table shows the impact scope of TMDS and SVDS problems. The frequency of problems (out of 5,379) in TMDS is parenthesized.

The categorization presented in Figure 3 suggests that reported system-wide problems are more common than user-specific problems. This observation implies that more mis-configurations are caused by application programs or OS components incorrectly updating Registry entries and/or non-robustly reading Registry data, than user actions inappropriately modifying settings. The approximate even distribution of problems between application-level and system-level impact suggests that tackling PC fragility problems requires coordinated efforts from both OS and application developers.

5. What We Can Do About PC Fragility

We identify three areas of focus to address the issue of PC fragility. For existing applications running on a released OS platform, a monitoring tool can detect known bad changes to Registry entries. For applications that are under development, a fault injection tool can help verify immunity to known problems. For designing a new operating system, we describe and evaluate access protection mechanisms that are most effective, according to our fragility data. Figure 4 summarizes the categorization of text-mined and strider-verified data based on monitoring, fault injection and access protection axes.

How to Monitor	TMDS	SVDS
Known bad entry	22 (2746)	35
Potential bad + symptom match	71 (2390)	60
Can't help	7 (243)	5
What to Inject		
Bad data	22 (832)	24
Data legal but considered bad	39 (933)	41
Item exists	3 (278)	9
Item missing	5 (136)	10

Key exists	12 (2454)	7
Key missing	5 (146)	9
Bad key	8 (263)	0
Bad sub-key	5 (325)	0
Type corrupt	1 (12)	0
How to Protect		
OS lockdown	9 (296)	20
Check rules upon modification	18 (659)	14
Copy on Write	3 (118)	1
Log changes	56 (1757)	58
Can't help	14 (2549)	2
Ignore	0 (0)	5

Figure 4: This table shows the number of TMDS and SVDS problems per category described in sections 5.1 to 5.3. The frequency of problems (out of 5,379) in TMDS is parenthesized.

5.1. Monitoring Techniques

During the interim period between Registry-problem discovery and consequent repair by developers, PC monitoring is essential. Maintaining a knowledge base of known bad predicates from past Registry troubleshoots provides an invaluable resource for problem diagnosis and prevention. Monitoring can be performed either periodically or by registering to receive change notifications. Such protective measures can prevent problems from causing eventual failures and ease troubleshooting after failure occurence.

In general, a Registry entry's presence or absence of data can positively identify problems. For example, consider the SirCam virus changing the data of the *HKEY_CLASSES_ROOT\exefile\shell\open\command\(Default)* Registry item from *"%1" %** to *C:\recycled\sirc32.exe "%1" %**. This change would generate a "File Not Found" error when the user double-clicks a .exe file. The presence of data *C:\recycled\sirc32.exe "%1" %** can be monitored to positively identify the problem. Similarly, the absence of *HKEY_CLASSES_ROOT\CLSID\{00021401-0000-0000-C000-000000000046}\shellex\MayChangeDefaultMenu* can be monitored to positively identify the problem of dysfunctional shortcuts that fail to launch an application when the user double clicks them.

The breakdown in Figure 4 shows that many of the problems we studied can be immediately flagged upon detection of known bad Registry entries. In other cases, the software used by the user and the underlying environment dictate if the data of a Registry entry is problematic. Furthermore, many configurations are purely for purposes of customization; an unusual configuration in one user's perspective may not necessarily be a problem, rather a valid choice made by a different user or the same user at an earlier time. In such cases, the monitor must record the "potentially bad" Registry entry and await user complaint to perform symptom-based troubleshooting. It is important to not warn the user upon detecting every such known "potentially bad" entry as numerous false positives can eventually lead the user to disable the monitoring tool. For example, if *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial* is set to 1, some versions of Windows Media Player's "Open URL" function will fail even if the user has Internet connectivity. However, some users do not use this application and may set *EnableAutodial* to 1 for preferred functionality of other Internet applications. The monitor is unsure whether the user intended to make this change, as it is a valid customization. Alternately, the monitor can be fully prepared to point to the entry as the root cause if and when the user complains with a symptom matching previously reported occurrences.

Policy-related settings also belong to the "Potential bad entry + symptom matching" category and account for a significant percentage of the cases we studied. In a corporate environment, policy-related Registry entries allow the IT organization to disable certain functions on employees' desktops to increase stability and simplify maintenance. Many such settings have caused user complaints as they could not perform the functions that were normally available to them outside the corporate environment. Obviously, we should not warn all employees about potential problems when the IT organization implements a new policy because most people may never use the disabled function. Nevertheless, our monitor can quickly point to the root cause when a user actually complains with a matching symptom.

We note that a small percentage of problems are categorized as "Can't help". These problems primarily consist of highly case-specific data corruption that are difficult to monitor and generally valid settings that cause problems only in specific situations. For example, an application window may not display properly if window position-related Registry data is corrupted; a browser proxy setting may cause confusing network connection problems if the user inadvertently uses it in an environment where the specified proxy cannot be found.

We believe our monitoring methodology is feasible as there are over a dozen rule-based monitoring tools already in use. Registry Mechanic[3], Registry Healer[4] and Registry Medic[5] are a few examples of tools that incorporate rule-based mechanism to identify dubious Registry entries. However, these tools monitor upon user demand (not passively and continuously in the

[3] http://www.winguides.com/regmech
[4] http://www.fixregistry.com/regheal
[5] http://www.iomatic.com/products/product.asp?ProductID=registrymedic

background). Furthermore, these tools do not distinguish known bad and potential bad entries; they allow users to decide between fixing and ignoring every detected dubious entry. Moreover, they do not have a dynamic rule update mechanism similar to anti-virus signature update and a user interface for specifying symptoms.

5.2. Fault Injection Approach

Fault injection is a useful task that tests a program's fault handling capability. For applications or OS components that run on platforms with known fragility problems, we can test robustness by injecting known problems during development. Such fault injection can also be used to test the Registry-monitoring tool.

To perform this task, we collect information regarding the Registry key/item present/missing that cause the problem or bad data or item type that instigate the problem. For example, when the Registry sub-key *{-5b4dae26-b807-11d0-9815-00c04fd91972}* is present under *HKEY_CLASSES_ROOT\clsid* the Internet Explorer menu bar (which contains File, Edit, View etc. options) is missing. Similarly, when the *CLSID* Registry item is missing under *HKEY_CLASSES_ROOT\MIME\Database\ContentType\ text/x-component*, the System Restore program in Windows XP fails to display dates of previous restore points. Demonstrating data corruption, in Microsoft Money Deluxe 2003, a maximized window would display properly, but a "restore down" regular window would be invisible on the screen if corrupted binary data is injected into *HKEY_CURRENT_USER\software\microsoft\money\11.0\mainwindow*.

Figure 4 depicts that over half of the problems related to bad data in a Registry entry (the first two rows combined). Settings recorded in the data field are read/written more often than a key or item is created or deleted. Therefore, injecting known bad and random data values for Registry items is a fruitful fault-injection technique. The large number of occurrences in the "Key exists" category were contributed by the top PSS problem. Perhaps popularity ranking of problems can be used to generate realistic fault loads.

We have successfully injected Registry faults using the Windows command line *Reg* operation. We generate batch files containing Registry modifications based on fault predicates from our problem database. A challenge in performing fault injection is that some Registry changes require an application restart, Explorer restart, re-login, or system reboot to take effect. Such information must be encoded in the rule-generating database to ensure correctness of injection tests.

5.3. Access Protection Mechanisms

Ultimately, we would like to have an OS that can protect the Registry from bad changes and eliminate the fragility problem. The OS can provide the highest level of protection by **locking certain entries**. Only the OS can modify such entries; permissions for users and applications are read-only. Many configurations critical to the system must never be modified and thus should be locked. For example, the *.mp3* key must have a *PerceivedType* of "audio", the *.jpg* key must have a *PerceivedType* of "image", and the *.htc* key must always exist to allow dynamic HTML to work correctly. There is an additional category in which the entries cannot even be modified by regular OS components. For example, some EXE/DLL files and Registry entries can only be modified by patch installation programs. However, we lacked information to make this distinction among OS-level permissions.

The next level of protection is **rule-checks upon Registry entry modification**. It is important to note the difference between such protection predicates and monitoring predicates described previously: protection predicates must be supplied either by OS developers to detect universally known bad conditions or by application developers to enforce application-specific must-satisfy conditions; in contrast, monitoring predicates can be supplied by anyone to detect known bad conditions associated with known problems.

For example, when the Default Registry item under *HKEY_CLASSES_ROOT\HTTP\DefaultIcon* is changed from *"%SystemRoot%\System32\Url.dll,0"*, to the path of a non-existent .dll file, Internet Explorer's Favorites link and address bar link icons are missing and are instead replaced by the "unknown file type" icon. It would be beneficial if the OS can perform an existential verification of the path/file indicated every time an item's data is modified under that key. Another essential rule variety would verify acceptable ranges of values for certain Registry item's data. For example, the *ScheduledInstallDay* Registry entry under *HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\ Windows\WindowsUpdate\AU* (for configuring automatic updates) denotes a day of the week and must fall within the range of 0 and 7 (inclusive), where 0 denotes all days and 1-7 denote the day of the week. Thus, upon modification, we must ensure that the new data does not exceed these bounds. As another example, the *s1159* and s2359 Registry entries under *HKEY_CURRENT_USER\ControlPanel\International* denote what postfix to display (AM or PM) for the time and can be customized by the user. If either Registry entry's data value exceeds 9 characters, dates (both the calendar and e-mail send/receive) in Outlook disappear. So it is useful to check if the newly updated values for such string-type Registry items meet specific character limits, if any exist. Finally, we have observed several

problems due to lack of robustness in handling Registry items with empty data, necessitating the OS to enforce non-empty data-field rules. For example, when the *HKEY_LOCAL_MACHINE\Software\ Microsoft\Internet Explorer\View Source Editor\Editor Name\(Default)* Registry item's data is empty, right clicking an Internet Explorer window and selecting view source opens the user's Desktop folder. This item's data should be set to any desired default view source editor (e.g., Notepad.exe or Winword.exe) but should never be empty. Since the user has the flexibility to choose any editor, the entry cannot be locked by the OS, rather a non-empty data rule must be enforced.

The third level of protection considers **copy on write**. Systems, as well as various applications, maintain their own copies of a Registry entry so that their customizations do not interfere. For example, some third party printers depend on the *HKEY_LOCAL_MACHINE\SYSTEM\ currentcontrolset\control\print\monitors\<3rd party monitor>* Registry key to exist for proper printer functionality. However, the existence of one such third party's keys may cause problems for other third party printer monitors, such as being unable to identify the printer monitor and consequently being unable to print. In this scenario, it would be beneficial for each third party printer monitor to have its own set of keys that do not interfere with the functionality of others and copy on write helps isolate each printer monitor's view of this Registry entry. However, there are limitations to this protection option. Registry entries that provide a *logical notion* of some configuration can employ this method whereas other entries that reflect *physical conditions* (e.g., existence of hardware) cannot.

Finally, many Registry settings are legal, but may cause behaviors that are perceived as failures by a user when operating specific software in certain environments. Figure 4 shows that more than half of these problems cannot be prevented through the above-mentioned access protection mechanisms (see the last three rows). At best, we can **log Registry entry changes to facilitate troubleshooting**. This mechanism provides the ability to analyze phases of change or instability of a Registry entry and incorporate this information into Strider's root cause analysis. For example, when Firewall Client is disabled, the user may experience good network performance for all applications except Instant Messenger. While the performance degradation is inconvenient to the user, disabling the Firewall Client (by modifying the Registry entry *HKLM\Software\Microsoft\Firewall Client\Disable*) is a perfectly valid configuration decision that must be allowed by the system. Predicate construction is complicated by the fact that Disable=1 and Disable=0 are both valid configurations. Thus, we must resort to logging any changes to this Registry entry.

Many problems could not be helped as they were untraceable to a Registry entry modification operation; they were caused by left-over Registry entries from software uninstallation or system rollback. A few problems, categorized as "Ignore", were either too application-specific or too expensive to protect.

6. Conclusions

We have attempted to answer the question *"Why Are PCs Fragile and What Can We Do About It?"* by providing a categorization framework based on 200 Windows Registry problems. While Registry-related problems are not the dominant cause of system failures, they are a source of major frustration. By classifying problem manifestation, we exposed a wide variety of effects that create the image of PC fragility. These manifestation effects range from instability of an entire system to lost functionality including simple user interface issues. For a majority of these cases, monitoring techniques based on known-bad predicates are useful for detecting potential problems. However, we must be careful not to over-burden users with too many false positives. In many cases, waiting for a user complaint with a matching symptom before pinpointing a problem is more appropriate. Similarly, a majority of problems cannot be prevented through any of the access protection mechanisms that are proposed because their root-cause Registry entries had legal data that cause fragility problems only under specific circumstances. This result demonstrates the importance of developing effective Registry troubleshooters and maintaining a comprehensive knowledge base of problems, reducing the impact of PC fragility on total cost of ownership and user satisfaction.

References

[1] A. Ganapathi, Y. M. Wang, N. Lao, J. R. Wen, "Why PCs Are Fragile and What We Can Do About It: A Study of Windows Registry Problems," MSR-TR-2004-25, March 2004.

[2] W. Li, "Random texts exhibit Zipf's-law-like word frequency distribution", *IEEE Transactions on Information Theory*, 38(6), pp.1842-1845, 1992.

[3] Y. M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang, "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support," *Proc. Usenix Large Installation Systems Administration (LISA) Conference*, pp. 159-171, October 2003.

[4] Y. M. Wang, C. Verbowski, and D. R. Simon, "Persistent-state Checkpoint Comparison for Troubleshooting Configuration Failures," *Proc. Int. Conf. Dependable Systems and Networks (DSN)*, June 2003.