Kernel read error

Illegal Operation

Wudu code 725

A fatal exception 0E has occurred at 0028:002A0201 in VXD I96(04)+00000000

Fatal Error

24 mm

(96pt)

0,945 inch

transfer

General Protection Fault

Error Messages: What's the Problem?

Real-world tales of woe shed some light.

Computer users spend a lot of time chasing down errors—following the trail of clues that starts with an error message and that sometimes leads to a solution and sometimes to frustration. Problems with error messages are particularly acute for system administrators (sysadmins)—those who configure, install, manage, and maintain the computational infrastructure of the modern world—as they spend a lot of effort to keep computers running amid errors and failures.

Over the past few years, we have spent time observing the problems and practices of sysadmins and have found them frequently frustrated or led astray by error messages. In fact, our data indicates that as much as 25 percent of a sysadmin's time may be spent following blind alleys suggested by poorly constructed and unclear messages.¹ We hope that once developers know how much time end users (such as sysadmins) spend dealing with error messages and how error messages shape problem-solving behavior, they will spend more time crafting messages to try to improve overall productivity.

In this article, we detail two examples of sysadmins hunting for the causes of specific errors. In one case, error and informational status messages led our sysad-

Error Messages: What's the Problem?

min astray, whereas in the other the messages may have helped, but only serendipitously. Because systems are so big and complex, individual errors are often reported far removed from their initial causes. This lack of coordination between error reporting and error origin often leads to incorrect human reasoning about root causes. One simple help to sysadmins (and other users) would be to report errors in context.

SYSTEM ADMINISTRATION

Sysadmins design, configure, troubleshoot, and maintain complex computer systems consisting of dozens of components (e.g., database management systems, Web servers, application servers, and load balancers) and hundreds of servers distributed across multiple networks and operating system platforms. Because the computational infrastructure of modern life depends on sysadmins performing their work nearly flawlessly, studying human error and problem solving in the context of large systems is important. Though people are often blamed for system failures,² we must recognize that sysadmin work places high cognitive demands on its practitioners—they must troubleshoot systems, making sense of millions of log entries (containing error, warning, and informational status messages) by controlling thousands of configuration settings and performing tasks that take hundreds of steps. Sysadmin work also places high social demands on its practitioners, who need interpersonal skills to communicate effectively so they can solve problems quickly.

Despite the importance of sysadmins, few studies report on their particular activities and practices.³ Because of the lack of in-depth studies of this critical user group, we conducted field studies in large corporate data centers, observing the organization, work practices, tools, and problem-solving strategies of many kinds of sysadmins.⁴ For our purposes, field studies offer insights into work that cannot be found in focus groups, lab studies, or surveys alone, as they can show us day-to-day interpersonal interactions not visible in isolated laboratory settings, particularly for problem solving.

We conducted seven field studies of database and Web sysadmins at large industrial service delivery centers. Two

researchers participated in each visit, which lasted three to five days. Typically, we followed one sysadmin per day as he or she worked in the office, attended meetings, and so on. One researcher took notes and occasionally asked questions, while the other video-recorded the sysadmin's interactions with the computer, other people, and information sources. We asked our participants to speak aloud while working, which they often did. We collected physical and electronic materials and took pictures of the artifacts in the work environment. We recorded, reviewed, and analyzed approximately 200 hours of video.

We now turn to two problem-solving examples that illustrate sysadmin experiences with error and other system messages. (Details of the products, customers, and sysadmins have been obscured to remove any identifying information.)

EXAMPLE 1: "I AM JUST A LITTLE CONFUSED"

In this first example, the customer had installed software for providing secure data delivery. This software has two parts: a *player* instance running on one server and a *maestro* instance running on another server. Through the player instance, "junctions" can be set up to back-end data servers, which provide the data for the applications. Communication between the servers was done through ports regulated by a firewall. The customer requested that a second player be added to improve performance.⁵ This work involved creating a new player instance, configuring the firewall to permit communication between the maestro and the new player instance, and setting up junctions for the back-end servers.

Our sysadmin, George, started his work by confirming that the network team had configured the firewall to open up ports (7137 from the player instance to the maestro instance, and 7236 in the other direction). He then copied the command to create a new player instance from a document and pasted it onto the command line:

```
m_web create inst2 -m 7137
```

The command returned no error, so George launched the command processor for maestro and then asked for a list of player instances:

```
m_admin> server task mplayer list
...
mplayer_inst2
...
```

Assured that the new player instance was in the list, he added a junction (/jump) for a back-end server at IP address 1.2.3.4:

```
m_admin> server mplayer_inst2 add
-t tcp -h 1.2.3.4 -l -s -b ignore /jump
```

This command failed:
Could not perform the administration request.
Error: Could not connect to server (status: 0x1234A123)

After several unsuccessful attempts to resolve the error, George needed help. Through his manager, he contacted Adam, the architect for this customer solution. He described the problem to Adam by phone:

I created the second player instance in the same server as the original instance. No trouble there. When I went to create the junction to the back-end server from that instance, it gave me a message something like “Could not find server” or “Could not connect to server.” I made sure it was running. I am not sure why it won’t accept the junction. It is not a problem with the server. It is a problem with the additional instance.

George then did a Web search for further information on the error and read Adam what he found:
Error: 0x1234A123 (305439011) Could not connect to server
Action: Make sure the server is running and accepting connections.
Name: m_s_socket_connectnot

Noticing that this suggests a problem with sockets, Adam told George that it might be a connection problem. George agreed:

Right, but I don’t know why. I do a server list and it shows it is there. I do m_status and it shows it is running. I do a grep on the server and I can see the process. So I know it is running.

George believed that the instance creation was successful because the m_web create command had succeeded and the server list commands showed the server. The apparent success of the instance creation and the ambiguity of the original error message led George to believe that the connection problem might have had something to do with the different references to the new player instance he saw in various messages.

In the m_web create command, he specified inst2 as the name of the new instance. Yet the output of the server list command showed mplayer_inst2 as the instance name. When he ran m_status, the server name appeared as workplace_mplayer_inst2. Upon examining the configuration file, he saw yet another variation on the player instance name. George raised his concerns about these differences to Adam:

I am just a little confused. I think the problem may be in the discrepancies with the naming conventions. You know, I see different names in different places.

Adam suggested he try another command to create

the instance. This time, however, George substituted a variation of the server name he had seen, and he got a different error message:

Could not perform the administration request.
Error: Server not found (status: 0x1256A123)

At first, George thought this was the same error message and reported it as such to Adam. A few seconds later, though, George realized that the message was reported immediately, whereas previously, it had taken a while. He told Adam:

This one spits right back—like almost immediately, so it doesn’t even like that at all. The other one kind of hangs there for a little bit and then spits it back.

Adam and George decided to look in the log files. Adam noticed some errors and asked George about them. George said:

Failed recovered, failed recovered. We see that in all our player instances in our environment. It has something to do with the firewall timeout. It is not what’s causing this problem, though.

What neither George nor Adam noticed were socket failures reported in the log file in a cryptic manner (gsk_secure_soc_write failed) among a thousand other messages. These errors were mixed in with the typical “failed recovered” errors that George was used to seeing, which likely led him to overlook the socket messages.

The problem was finally solved by George’s colleague, who discovered that maestro was trying to communicate with player on port 7137—but that the firewall had been opened *only* in the opposite direction (player to maestro) on that port. George (and others) spent nearly three hours setting up the second player instance, a job that should have taken only minutes.

A number of factors conspired against George: First, the problem was not detected immediately (when the player instance was created), but manifested itself only later (when the back-end junctions were created). Second, the error messages did not specify which system could not communicate with which other system (saying simply, “Could not connect to server”). Third, different parts of the system referred to the player instance with different names (including inst2, mplayer_inst2, and workplace_mplayer_inst2). Fourth, the logs were cluttered with “normal errors,” making it difficult for George and Adam to pick up on the specific messages (about socket failures) that might have helped. In the end, it was pretty difficult to discover the misconfigured ports and firewall from the trail of clues provided by the error messages.

Error Messages: What's the Problem?

EXAMPLE 2: "I OFFICIALLY GIVE UP"

In our second example, a customer has upgraded its *webmonitor* application, but then complained of not being able to view object spaces through that application. Our sysadmin, John, began by trying to verify the customer complaint. To access the webmonitor application on the remote machine, *olympus*, John first had to connect to an intermediary machine, *liaison*, from which he could view the object spaces through either a Web-based interface or command-line commands typed directly into the console. John could connect to *liaison* using either of two login accounts, each with different access privileges. At first, John knew of only one login account. To try to view the object spaces, John first logged in to webmonitor through the Web interface, but this resulted in an error: **Operation is not authorized**

To work around this, John tried to telnet to *olympus* to check the object spaces directly through a command-line query. When he tried to use the telnet client, however, he found no connection settings for *olympus*, and he could not create a new connection—several attempts failed silently.

He then tried to verify that webmonitor was functioning properly by restarting the application. Restarting solves many computer problems, as stopping eliminates a corrupt state and restarting creates a known, working state. While trying to restart, however, John hit another problem: webmonitor's setup required specific information about the system, such as database settings, which John did not know. To find this information, he had to perform a command-line query from *olympus*. John attempted to telnet to the server again, but still could not create a connection.

All these problems prevented John from verifying that the object spaces existed—making it difficult for him to determine whether the issue was in the front-end webmonitor interface or in the back-end object spaces. Furthermore, because John was unable to access the database settings required to restart webmonitor, he could not know if a simple restart would solve the problem.

Later, John discovered in an old e-mail message that his team had two login accounts to connect to *liaison*. He

reconnected to *liaison* using the other account, and this time the telnet client listed possible connections to *olympus*. On connecting to *olympus*, John ran the database query to retrieve the database settings needed to reconfigure webmonitor and confirmed that the object spaces did indeed exist on *olympus*.

When logging in to do the command-line query, John mistyped his login name, which resulted in the following: **Login failed. You have used an invalid username, login, or client certificate.**

This message led John to suspect that an invalid certificate might be the cause of the customer's problem. He enabled webmonitor to download all required certificates automatically. After restarting, there was no change in behavior. To verify that certificates were necessary to view the application object spaces through *olympus*, John renamed the *olympus* certificates and queried the object spaces through the command line. The name change did not affect the object spaces, so John restored the certificates to their original names. Following a suggestion from a colleague, John then copied certificates from a similar server and added them to the existing certificates on *olympus*. He again checked to see whether this resolved the problem, but it did not.

After restarting webmonitor one more time, John did a Web search for more information. Unable to find anything useful, he began to compare configuration settings between *olympus* and a similar, working server. Finding the compared settings identical, John said, "I officially give up," and asked his colleague to look at the problem.

In fact, our data did not show us what the problem ultimately was. We do know, however, that of more than two-and-a-half hours that John spent troubleshooting, about 20 percent was wasted following error and system messages to no avail. Sometimes, lack of messages stymied John (when connections to *olympus* failed silently). At other times, the messages contained little detail for him to follow up on (e.g., "Operation not authorized"). In the end, he was left grasping at straws—he knew something was wrong, but he had no idea what it was or where the problem might be found.

MESSAGES PLAY A FUNDAMENTAL ROLE

Error, warning, status, and other information messages play a fundamental role in the way people reason about computer failures. By examining how computer users really behave, we found that messages in fact determine what users think and do when confronted with problems. That is, messages do not simply alert users to problems, they guide problem-solving behavior.

In the first example, George observed that the player instance appeared to have different names in the output of different commands. Given these differences, he believed (reasonably) that there was a problem with the names. But these inconsistencies among status messages simply misled George. This was just bad design. One design flaw was that the first indication of a problem appeared several commands after the problem actually occurred. An error message indicating the misconfiguration could have appeared immediately after the new player instance was created.

Forming connections between events that appear at different times and places is especially difficult.⁶ The error was not only reported much later, but also contained little specific information on the context of the problem. The message “Could not connect to server” was particularly unhelpful, as at least three servers were in the immediate problem context. Which server could not connect to which other server?

What’s more, the error message was not only ambiguous but also similar to other messages. When George tried another command, he got the error message, “Server not found,” which was in fact different from the original error (“Could not connect to server”), but this went almost unnoticed because it looked so similar. This could be particularly severe in the case of log output, as logs typically contain lots of messages. In the case of George, we saw how two people could not recognize the new messages in the log, because they were hidden behind the hundreds of “typical” error messages.

There was a similar case of bad design in the second example. Why did connections to the olympus server fail silently? If John had gotten some hint that his login name was invalid, he might have spent less time trying to log in and more quickly found alternative login names. Even the error message resulting from the mistyped login name—“Login failed. You have used an invalid username, login, or client certificate”—was unhelpful. Which is it: username, login, or certificate? The fact that this message led John to wonder about client certificates is little consolation, since this message resulted from a simple typo logging in, rather than from the actual problem under consideration.

All these observations suggest that messages are often not well coordinated with problems. One simple solution would be to have a mechanism that traps errors when they occur and propagates them up the stack to the user, along with the contextual information collected from the parts of the system that the error passes through. The two examples outlined here were typical in that different parts

of the systems came from different vendors, suggesting that any such error-reporting scheme must be an industry standard if error messages are to actually help people find and fix problems in the real world. Q

ACKNOWLEDGMENTS

Many thanks to Eben Haber for help with the analysis of Example 1, and to Madhu Prabaker for help with the analysis of Example 2.

REFERENCES

1. Barrett, R., Haber, E., Kandogan, E., Maglio, P. P., Prabaker, M., and Takayama, L. A. 2004. Field studies of computer system administrators: Analysis of system management tools and practices. To appear in *CSCW 2004* (Computer-supported Cooperative Work).
2. Oppenheimer, D. 2003. The importance of understanding distributed system configuration. In *System Administrators are Users, Too: Designing Workspaces for Managing Internet-scale Systems*. Eds. R. Barrett, M. Chen, and P. P. Maglio. CHI 2003 Workshop.
3. Barrett, R., Chen, M., and Maglio, P. P. 2003. *System Administrators are Users, Too: Designing Workspaces for Managing Internet-scale Systems*. CHI 2003 Workshop.
4. Barrett, R., Maglio, P. P., Kandogan, E., and Bailey, J. 2004. Usable autonomic computing systems: The administrator’s perspective. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*.
5. Maglio, P. P., Kandogan, E., and Haber, E. 2003. Distributed cognition and joint activity in collaborative problem solving. In *Proceedings of the Twenty-fifth Annual Conference of the Cognitive Science Society*. Boston.
6. Decortis, F., de Keyser, V., Cacciabue, P. C., and Volta, G. 1991. The temporal dimension of man-machine interaction. In *Human-Computer Interaction and Complex Systems*. Eds. G. R. S. Weir and J. L. Alty, 51-72. San Diego: Academic Press.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

PAUL P. MAGLIO manages human systems research at the USER (User System Ergonomic Research) group of IBM Almaden Research Center. He has a Ph.D. in cognitive science from the University of California at San Diego. His undergraduate degree is in computer science and engineering from MIT.

ESER KANDOGAN is a research staff member in the USER group at the IBM Almaden Research Center.

© 2004 ACM 1542-7730/04/1100 \$5.00