ANDY SEELY

system administration of command and control systems

Andy works for Northrop Grumman as the technical lead for the Global Command and Control System at the Headquarters, U.S. Central Command (J6), in Tampa, Florida. He taught Linux and UNIX courses at the University of Maryland in Europe for five years, has been a command and control system administrator for over a decade, and was the promoter for the 2006 GCCS System Administration and Engineering Conference. His wife Heather is his init process.

seelya@centcom.mil

IN THE MILITARY COMMAND AND CONtrol (C2) environment there are many missions that come together to allow a commander to fight, but it is the system administration mission that is the least understood. My day-to-day system administration tasks are typical of those in any systems support shop, but they have a unique flavor that forces me to think creatively about sysadmin problems. The C2 sysadmin focuses on maintaining totally reliable and robust systems; a system crash on one side of the world can mean death on the battlefield on the other side of the world. Yet a C2 system is highly conservative, and the C2 syadmin is not allowed access to many traditional tools such as compilers, debuggers, or network sniffers. In this article I explain how the limited environment and mission focus changes the way I and my colleagues approach the job of system administration.

Mission Orientation

My systems are common: a Sunfire v240 with 2 GB of RAM and two 800MHz processors, Dell 2650 rack-mounted servers, Gateway desktop workstations, etc. But put these systems in a C2 context and the term *mission oriented* forms the basis of all support efforts. It is from mission orientation that all other critical issues are derived. A C2 system is ultimately a system for which failure may mean the death of American soldiers far from the point of that system's support effort. The soldiers who depend on the products of a C2 system execute missions where life and death lie in the balance; a C2 system administrator has to keep this in mind every time a server is rebooted.

Mission orientation is relevant to industry in that it can bring a dedication and drive to systems support. Other systems support shops may exhibit these traits: Civilian law enforcement and emergency response, finance, and legal systems, for example, all may have the same life-or-death importance found in a C2 system. With the stakes this high, a mission-oriented C2 system presents an interesting, unique, and above all challenging system administration opportunity.

Critical Support Issues

Maintenance of C2 systems is focused on getting the data or functionality to the end user. All other considerations fall to a distant second place. Downtime for backup and recovery drills is not typically an option. Key goals in C2 support include uptime, alternate data paths, failover, and hardware backups. The bottom line to any C2 support is a simple, repeatable solution that may be implemented in a minimum amount of time by a less-than-seasoned system administrator.

Like any other systems shop, my uptime metrics are important and scrutinized closely by leadership. More importantly, C2 systems may be at the core of sustained military operations in such a way that uptime is the only thing that matters. Uptime is maximized through a close understanding of the relationship between the kernel and the application and the network interface. Under normal situations a server may appear to need a reboot. In critical situations a careful analysis may show that bouncing a network interface using ifconfig may be all that's needed. Manual restart of system services via /etc/init.d to allow system resources proper allocation may also serve to "reboot the process" instead of rebooting the computer. Unless the problem resides in the core of the system, memory thrashing, process table management problems, or I/O subsystem problems, it's likely the system itself does not need a reboot. The problem with this type of repair is that it requires a senior sysadmin to do the analysis and come to a conclusion quickly. If the problem causes degradation to the point of service interruption for longer than it would take for a normal reboot, then the window of opportunity has been missed.

Failover is a goal for any important server system, but it is not always as viable in a Government Off The Shelf (GOTS) software world as it is in a commercial environment. Commercial software may be explicitly designed to perform a failover function; DNS, SMTP, and HTTP services, for example, are easily set up in a pooled environment that will allow any given server to fail without the entire service suffering a failure. Unless it's specifically contracted to be so written, GOTS software does not typically allow this type of pooling. In most cases failover of a C2 system is a manual process. Certain failover points may be set up in hardware, such as disk mirroring and firmware instructions for boot order to recover automatically from a failed boot disk, multiple network interfaces on different switches to allow for failure of a network leg, or multiple power sources from different power legs to allow for inconsistencies in the power grid.

Systems may die for a variety of reasons, from software that degrades its own configuration, to hardware that stops passing electrons, to a hurricane that flattens the server room. When a system dies, there must be an alternative solution to the data-processing mission. In military terms this alternative routing is known as an ALTROUTE. When the software repairs, system reboots, and hardware swaps can't fix the problem or can't fix it fast enough, there must be an alternative. Every critical C2 system must have a viable and tested ALTROUTE plan in place. Regular exercising of that ALTROUTE is essential to ensure that procedures are viable, personnel knowledgeable, and the end user of the data can actually successfully accomplish the mission in an ALTROUTE configuration.

When C2 hardware fails it must be repaired or replaced. Hardware warranty contracts can be expensive and, in some locations, impossible, and maintaining on-site hardware technicians can be even more expensive. There is a cost-benefit relationship where the variables are cost and mis-

sion restoration time. In most situations the greatest speed for return to service may be had at the lowest cost by simply having one or more complete spares on the shelf for every critical bit of hardware in service. This allows me to make rapid change-out of a failed server while buying a cheaper warranty plan for getting the failed system repaired.

My ultimate solution to satisfy service delivery and critical goals is one that may be accomplished by the most junior member of my team. Can my most junior sysadmin restart a service instead of a server? Can he or she implement the ALTROUTE procedure or pull the dead server out of the rack and replace it with a cold spare? Can this lowest-paid member of the team identify the critical points for C2 systems support? After years of C2 system administration work, I've come to truly appreciate that critical and simple are values that cannot be separated.

Dealing with GOTS

GOTS software development is not subject to the same pressures as open source software. Transparency is considered a bad thing, and I have almost never been granted access to source code. Because government contracts for development define clear boundaries for product deliverables, interoperability with other software and open APIs for system administration analysis just aren't there. I cannot assume that GOTS software will behave in a fashion consistent with commercial software or with related GOTS software, or even with different versions of itself. This leads to significant problems with integration and performance tuning. How do I right-size a server's physical RAM if there's no documentation on how much RAM a product requires to run correctly? How do I correctly configure the site border protection when the behavior of a product's network interface is so poorly understood that there are frequent and heated debates on such fundamental issues as whether the product uses TCP or UDP? How do I explain to the user community why their applications don't run correctly when they were developed for a workstation security model greatly different from their own?

Successful administration of GOTS requires a savvy, creative system administrator with a handy box of stock operating system tools, including truss, strings, files, snoop, ptree, and od. If the tool didn't ship with the OS, then I can't use it. C2 systems are subject to configuration management rules that sometimes make good sysadmin practices impossible. If I were to install gdb and Ethereal, for instance, I would be committing a security violation that could potentially shut my operation down. The solution is to be creative with the tools you have. The following are examples of some problem-solving techniques I've had to use in a Solaris and Bourne shell environment.

Problem: GOTS software listens on the wrong port. Suspect this is a configurable option but don't know where the configuration file is.

Solution: Find all ASCII files in the software's installation directory and grep for the bad port number:

```
find /path -print | xargs file | egrep -i "ascii|command|text|script" | awk -F:
'{print $1}' | while read fn; do; grep PORTNUM $fn > /dev/null && echo $fn
>> results; done
```

Problem: The configuration file is known and in a standard location, but the software does not change behaviors when configuration values are changed. Suspect more than one configuration file is on the system.

Solution: Either do a system-intensive global find for all files of the same name or truss the process on startup to find the specific file in use:

```
find / -name "config.file" -print &
truss -a -e -f /path/to/start 2>&1 | grep config.file
```

Problem: GOTS software turns out to be a GOTS wrapper around commercial software. Network behavior is on the right ports and protocols, but it is not behaving correctly. For example, my GOTS DNS turns out to be an older version of BIND and it doesn't resolve TLDs on my closed network segment.

Solution: Test your assumptions. Is BIND using the correct root name servers for the network? Snoop the network interface for the correct root server while doing a name resolution. Or just validate the hints file; commercial software rolled into a closed government network may not have been set up correctly in the first place.

```
snoop -v root.server.IP.address
```

Problem: GOTS software is delivered as an ELF binary with no documentation. Command line switches are suspected but not known. How do I get a help listing?

Solution: Most GOTS doesn't conform to the --help convention, but sometimes a product will try to correct you when you make a request it can't handle. Feeding the program gibberish will sometimes produce a usage report. If not, the strings command and some educated guesswork can help. Try to get a usage message with the following:

```
./program -asdf
./program /asdf
./program asdf
```

You can use strings to find common arguments:

```
strings program | egrep -i "debug|help|-v|-i"
```

Problem: GOTS software output is in a binary format that resists analysis. While troubleshooting, it's suspected that a data stream issuing from the software is corrupt.

Solution: Capture a known-good data stream and compare using od and diff. Use the -c option to od to display the human-readable elements of the data stream:

```
od -c stream.one > od.one
od -c stream.two > od.two
diff od.one od.two
```

Problem: GOTS software is doing significant IPC. There is a failure in the main process, but it's suspected that the problem is with a subsystem. As usual, there's no documentation.

Solution: Use ptree to analyze the way the processes fork() and isolate situations where a subprocess is hanging awaiting IPC input; zombie processes are a good indicator that IPC is being handled poorly in the software. Use truss -f to follow system calls in each fork and identify problems with resource management; for example, a parent calls may wait() on a child or the child may be blocked indefinitely owing to a dead IPC pipe:

```
truss -f -o /tmp/trussfile ./start.program &
[PID]
/usr/proc/bin/ptree PID
```

GOTS software is most commonly developed under government contract. Occasionally the government will use a commercial or open-source software solution but wrap the software in a GOTS package for consistency. I find this to be a special challenge; to fix the GOTS package, I have to be extra-intimate with the standard software when I walk in the door. Full understanding of the way a software package works "in the real world," as we say in the C2 shop, is critical to debugging why it doesn't work for us. I try to isolate problems with the core software configuration, the GOTS software delivery, or system integration in a GOTS environment, in that order.

Dealing with Legacy Systems in a Conservative Environment

The C2 environment is highly conservative. Change happens slowly and for good reason. The latest and greatest system introduces more problems than it solves:

1. Bringing a new system online requires significant planning to ensure minimum downtime, and it never goes the way you expect it to. Why introduce risk into a stable system?
2. Current systems are better understood by the operator and administrator than any new system.
3. Legacy software may be less of a hot target for hackers, crackers, and script kiddies, whereas the newest thing is the one everyone wants to break.
4. Newer systems tend to have more and newer features, which can lead to information and option overload. Missions may be enhanced by a greater range of options, but mission operators may spend so much time figuring out and playing with the options that the mission falls to the side. Administrators end up responding to "problems" about new features that have nothing to do with the mission.

The key to system administrator survival in this environment is to focus on the service delivery mission and not get distracted by new features and new software. The latest, from my C2 perspective, is not always the greatest.

Keeping Personal Skills Up with the Industry

It's a real challenge to be a serious system administrator in a fast-moving industry. Being a C2 system administrator makes keeping up feel to be impossible. How do I maintain status as a professional system administrator while balancing the requirement to maintain systems that seem like they never change? I've found the keys to be an awareness of the movement of the greater industry and a focus on generalized, creative skills. I believe that limitations breed creativity, and this can be a bonus for the C2 sysadmin. Being a pathological optimist doesn't hurt either.

My key points for keeping up in industry include the following:

1. Professional association memberships. Although I certainly don't have time to read every magazine that shows up in the mailbox, I do have time for abstracts. Professional organizations are the heartbeat of the industry, and even a casual exposure can help.

2. Certifications. Some folks think a certification is not worth the effort, but keeping an OS certification current forces a C2 sysadmin to follow changes in the OS and demonstrates commitment to the profession.
3. Training, seminars, and conferences. Subject to funding, of course, technical interchange with broader industry is the single most beneficial thing a C2 sysadmin can do to prevent professional atrophy.
4. Home hobby systems. I may not be supporting the cutting edge on the job, but you can bet I do at home.
5. Involvement with the open source community, IRC channels, local and user groups; these are the grassroots of the whole industry.

Serving a Higher Mission

At the military commands where I've worked, I've become dubiously famous for my verbal stream editing. I never speak a sentence that doesn't come through the command `sed -e 's/problem/challenge/g'`. But I truly believe that the difference between a problem and a challenge is whether you control the situation or it controls you. The issues my team and I face every day with GOTS software, limited tools, critical missions, and atrophy of skills are all challenges when we approach them with creativity and personal responsibility. There are few sysadmin jobs with such an impact on the lives of others. A good sysadmin in the industry is not necessarily a good sysadmin in the C2 world, but a good C2 system administrator is likely to have great potential in industry. The creative thinking required by closed systems more than makes up for the lack of cutting-edge systems, while the mission orientation encourages drive and focus uncommon in the industry.