strace

Practical Application Troubleshooting

- Practical Troubleshooting, not an in-depth guide to the various system calls...
- Linux -- strace 4.5.20
  - Applying to most distros; Centos, Debian, Suse, etc.
    - Original author(s) Paul Kranenburg
    - Developer(s) Dmitry Levin
    - Stable release 4.7 / May 2, 2012; 48 days ago
    - Written in C
    - Operating system Linux, FreeBSD
    - Type Debugging
    - License BSD

- Install
  - Centos
    - yum install strace
  - Debian & uBuntu
    - apt-get install strace
  - Others
    - Website http://sourceforge.net/projects/strace/

Challenges of Modern day IT

- Introduction to the "DevOps" environments
    - Easier access to Developers
- Modern Infrastructure
    - Infrastructure tools (Puppet, Chef, CFEngine, etc.) allow for easy deployment of new nodes. Once the environment has been setup, bootstrapping a VM (VMware, EC2, etc.) and complete application stack install in less than 30 minutes...
    - With a truly scalable infrastructure model we can replace a faulty node in less than 30 minutes!!!
- Can I afford to spend hours troubleshooting?

strace

- What is it?
- What does it do?
  - Plus some background information on system calls...
- How can you use it to make troubleshooting easier???

What is it?

- **strace: A useful diagnostic, instructional, and debugging tool.** *System administrators, diagnosticians and trouble-shooters will find it invaluable for solving problems with programs for which the source is not readily available since they do not need to be recompiled in order to trace them. Students, hackers and the overly-curious will find that a great deal can be learned about a system and its system calls by tracing even ordinary programs. And programmers will find that since system calls and signals are events that happen at the user/kernel interface, a close examination of this boundary is very useful for bug isolation, sanity checking and attempting to capture race conditions.*

What does it do?

- In the simplest case strace runs the specified command until it exits. It intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and its return value are printed to stdout or to the file specified with the -o option.

System Calls

The Linux Kernel offers User-Mode processes a set of interfaces to interact with hardware devices such as CPU, disks and printers. Putting an Extra Layer between the application and the hardware has several advantages.

- It makes programming easier by freeing application programmers from studying low-level programming characteristics of hardware devices.
- It greatly increases system security, because the kernel can check the accuracy of the request at the interface level before attempting to satisfy it.
- These types of Interfaces make programs more portable, because they can be compiled and executed correctly on every kernel that offers the same set of interfaces.

Common System Calls

- read()
- write()
- open()
- execve()
- connect()
- futex()

How can I apply it to make troubleshooting easier?

- Do you have an application that starts up and immediately dies without writing sufficient data to a log? Or even worse, an application that supplies little to no logging.
- Do you want to see why a process is consuming all cpu resources?
- Do you need to supply a debugging output of an application to a developer or bugtracking system?
- Do you have the desire to see exactly what an application or script is doing on your system?

- Let’s take a look at strace in action
    - Unfiltered strace
    - strace summary
    - Filtered *(and grep’ed)* strace

- Difficulties working with strace
  - Attaching to a process after it's running
    - Easily done by identifying the PID
  - Run an init script using strace
    - If the startup script runs a daemon, you may not trace the other processes
  - Run a startup daemon or startup application
    - Examples; mysqld_safe5, apachectl, etc.

A few use cases

- An application (job, script, etc.) has become unresponsive.
- Extracting data from a process (or script).
- Troubleshooting an application that uses encryption.

Case #1 -- Extracting data from a script

- A Physical server's hardware monitoring tool triggers an alert for Memory Chip.
- The hardware manufacturer asks you run a diagnostic utility to identify the error, the resulting output is placed in an encrypted zip archive.
- Curiosity gets the better of you and you spend two days attempting to crack the password with your best dictionaries.

Two Days later...

- Re-run the script using strace sending the output to a file
- Took about 5 minutes, but the password became instantly recognizable when the ‘execve’ system call ran the zip binary passing in the string “zzzwxyz”...
  - A google search would have revealed the answer much faster, but that’s too easy...

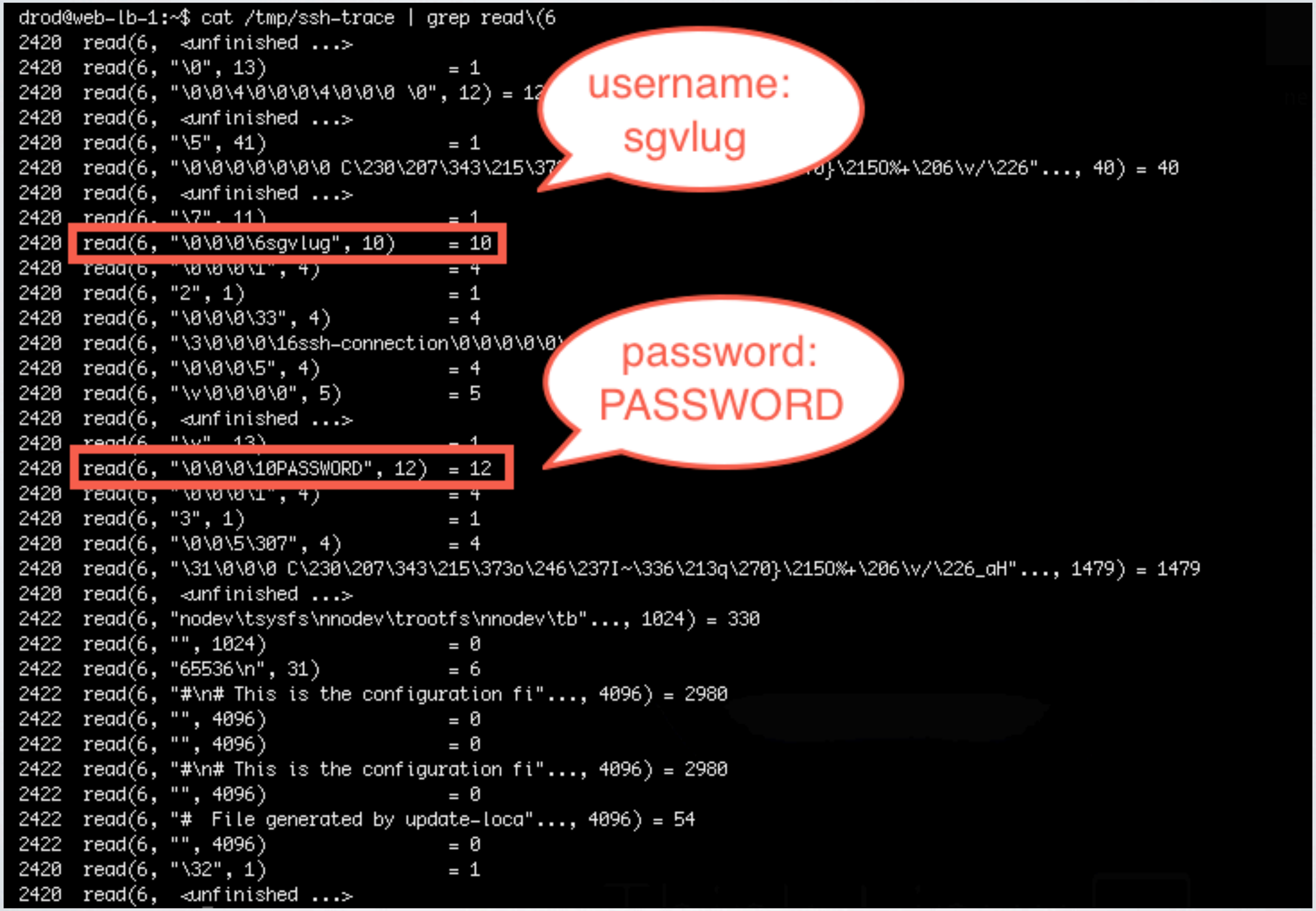
Case #2 -- Extract encrypted data from an application in plain text.

- SSH sends data encrypted across the network.
- Capturing the connection yields only encrypted data.

SSHD trace

- ps aux | grep sshd
- strace -f -o /tmp/ssh-trace -e read -p <pid>
- tail -f /tmp/ssh-trace | grep read\(6
  - tail -f /tmp/ssh-trace | grep read\(6 | cryptcat -k secret servername 44444

SSHD trace



```
drod@web-lb-1:~$ cat /tmp/ssh-trace | grep read\(6
2420  read(6,  <unfinished ...>
2420  read(6, "\0", 13)                = 1
2420  read(6, "\0\0\4\0\0\0\4\0\0\0 \0", 12) = 12
2420  read(6,  <unfinished ...>
2420  read(6, "\5", 41)                = 1
2420  read(6, "\0\0\0\0\0\0\0 C\230\207\343\215\37[illegible]0}\2150%+\206\v/\226"..., 40) = 40
2420  read(6,  <unfinished ...>
2420  read(6, "\7", 11)                = 1
2420  read(6, "\0\0\0\6sgvlug", 10)      = 10
2420  read(6, "\0\0\0\1", 4)           = 4
2420  read(6, "2", 1)                  = 1
2420  read(6, "\0\0\0\33", 4)          = 4
2420  read(6, "\3\0\0\0\16ssh-connection\0\0\0\0\0[illegible]
2420  read(6, "\0\0\0\5", 4)           = 4
2420  read(6, "\v\0\0\0\0", 5)         = 5
2420  read(6,  <unfinished ...>
2420  read(6, "\v", 13)                = 1
2420  read(6, "\0\0\0\10PASSWORD", 12)  = 12
2420  read(6, "\0\0\0\1", 4)           = 4
2420  read(6, "3", 1)                  = 1
2420  read(6, "\0\0\5\307", 4)         = 4
2420  read(6, "\31\0\0\0 C\230\207\343\215\373o\246\237I~\336\213q\270}\2150%+\206\v/\226_aH"..., 1479) = 1479
2420  read(6,  <unfinished ...>
2422  read(6, "nodev\tsysfs\nnodev\trootfs\nnodev\tb"..., 1024) = 330
2422  read(6, "", 1024)                = 0
2422  read(6, "65536\n", 31)           = 6
2422  read(6, "#\n# This is the configuration fi"..., 4096) = 2980
2422  read(6, "", 4096)                = 0
2422  read(6, "", 4096)                = 0
2422  read(6, "#\n# This is the configuration fi"..., 4096) = 2980
2422  read(6, "", 4096)                = 0
2422  read(6, "#  File generated by update-loca"..., 4096) = 54
2422  read(6, "", 4096)                = 0
2420  read(6, "\32", 1)                = 1
2420  read(6,  <unfinished ...>
```

Sanitize your Bug Reports!

- The output from strace can be used in malicious ways.
- Passwords that would normally be encrypted can be seen as clear text strings...

Case # 3 -- Application has become unresponsive

- A Java Business intelligence application (job) normally takes 4 hours to run (manipulates data on several databases larger than 200gb).
- The job took 14 hours to run one weekend after a massive influx of new business.
- When I arrived Monday morning, the monitoring system showed an error with the job. I had a higher priority issue that needed to be addressed immediately...

- The ‘chef-client’ failed to complete a run with the error, “‘cannot ‘umount’ a directory due to open files”.
    - This is a High Priority Error!

- root@servername-2:/root# mount
- /dev/sda1 on / type ext3 (rw,errors=remount-ro)
- proc on /proc type proc (rw,noexec,nosuid,nodev)
- sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
- udev on /dev/ type tmpfs (rw,mode=0755)
- devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=620)

  ****** The next line is the problem ******

- //fileserver-3/Dir2/Dir1 on /App-directory/path type cifs (rw,mand)

- root@servername-2:/root# lsof | grep App-directory
- java   26472 appuser 278w   REG   /App-directory/path/Filename.zip (deleted)

- root@servername-2:/root# strace -p 26472
- Process 26472 attached - interrupt to quit
- **futex(0x7f102b5e89d0, FUTEX_WAIT, 26473, NULL**
- ... (15 minutes)
- ... (30 minutes)
- ... (60 minutes)

- root@servername-2:/root# ps axjf
- *Interpreted...*
- The process (java -- 26472) is in an uninterruptable sleep state and is multi-threaded
- The parent processes are also in an uninterruptable sleep state
- This is the Child. Not a parent.

- It's time to kill the process...

Case #4 -- Troubleshooting an Application that uses Encryption

- Secure SFTP ch'rooted to the user's home directory.
- Packet Capture is not as helpful due to the encryption.
- Even using the -vvv option on the ssh client did not help.
  - Debug 3: EOF recvd

Isolating the SFTP issue

- The sftp client logs into the system using public key authentication.
- The sftp client can "cd", "put" and "get" files...
- Listing directory contents (e.g. the "ls" command) causes immediate disconnection of the client.
- It could be permissions, but in this case it was not the cause.

Identifying the root cause with strace

- Centralized Authentication was used in this scenario.
- The chroot’ed environment does not have access to the necessary resource.
    - In this case it was the socket file of the authentication module.

Attaching to a process

- The "-p" option allows you to attach to a running process using the PID
- Careful when using in a production environment
  - Take the node out of the pool before attaching strace
  - Passing incorrect options into strace can cause the application to crash.

Examples of strace

- **strace -f -c** process
- **strace -e poll,select,connect,recvfrom,sento,socket** nc site.domain.com 80
- **strace -f -s 1024 -e open,read,write,access,execve -o /tmp/script.trace** script
- for x in `pgrep app`; do **strace -f -o /tmp/app-pid-$x.trace -p $x** ; done
    - -f option
        - Follow Forks
    - -c option
        - Summary -- Statisics
    - -e option
        - Print only specific Expressions
    - -o option
        - Output file

Using strace to solve service start-up errors

- Identify how the service starts (init script, launched by daemon, etc.)
- Is the "-f" option enough, or is a 'for' loop more appropriate?
  - for i in `pgrep <processname>`; do strace -f -p $i -o >> <appname>$i.trace; done
- Modify the execution path to include your strace command
  - Standard
    - /usr/bin/mysqld_safe > /dev/null 2>&1 &
  - Modified
    - **strace -f -o /tmp/mysql.trace** /usr/bin/mysqld_safe > /dev/null 2>&1 &

Wrap-up

- Make your life easier, use tools...
- The right tool for the right job... Yes, but who can argue against carrying a multi-function pocket-knife that can help to quickly resolve many different types of challenges?
- Virtualize the application. Multiple nodes allows you to have multiples points of comparison and frequently a ‘known-good’ state for troubleshooting.

Thank you to:

- Paul Kranenburg -- Original Creator
- Dmitry Levin -- Current Developer
- Wikipedia -- Tons of information
- The wonderful folks that help to organize and manage SCaLE

- David Rodriguez -- *otherwise known as...* 'D Rod'
  - Linux Systems Engineer -- managing Linux DB & app servers
  - 3+ years as a Linux Systems Engineer
  - 5+ years as a Windows Systems Engineer
  - B.S. in Information Systems Security
  - A.S. in Computer and Electronic Engineering