Чудеса трассировки: Решение проблем с приложениями при помощи утилиты strace

Журнал «Хакер», 13.01.2011 0 38426

Содержание статьи

Представь ситуацию: ты поставил новую классную прогу, а она не запускается или безбожно тормозит. Или сетевой сервис падает при непонятных обстоятельствах. Досадно! Ситуация усугубляется тем, что ни в консольном выводе, ни в логах ничего интересного нет. Но и в этом случае можно предпринять ряд действий, которые, если и не помогут устранить проблему в запуске, то хотя бы позволят составить правильный баг-репорт.

Знакомство

Первый помощник в таком случае — это strace. Для тех, кто вдруг не читал статью в #10 за 2009 год («Танцы с бубном и напильником»), напомню, что работа strace заключается в перехвате и записи системных вызовов, выполненных процессом, а также полученных им сигналов. Strace может помочь в следующих ситуациях:

- если приложение отказывается работать из-за проблем с правами;
- если приложение не запускается из-за отсутствия какого-нибудь нужного файла;
- в некоторых случаях с помощью strace быстрее, чем с помощью tcpdump, можно обнаружить проблемы с сетевыми прогами;
- при проблемах с физическим или псевдоустройством (типа /dev/random или /dev/ audit) strace покажет последний незавершенный вызов;
- если надо отследить все файлы, к которым обращается приложение в процессе работы. Это может быть полезным, например, для составления профиля AppArmor или переноса приложения в среду chroot. В простейшем случае вызов strace выглядит следующим образом:

```
$ strace uname
execve("/bin/uname", ["uname"], [/* 36 vars */]) = 0
brk(0) = 0x1ed2000
access("/etc/ld.so.nohwcap", F_OK) = -1
ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb79f08a000
access("/etc/ld.so.preload", R_OK) = -1
ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=133660, ...}) = 0
...
### Попытка получить доступ к большому количеству
файлов, в основном из каталога /usr/
lib/locale/ru_RU.utf8
uname({sys="Linux", node="adept-laptop",
...}) = 0
...
```

По умолчанию весь вывод strace отправляет в stderr, что далеко не всегда удобно. Попросить strace писать вывод в файл можно с помощью опции '-o':

```
$ strace -o uname.strace uname
```

Первый системный вызов — execve: запуск файла на выполнение. В скобках передается команда с аргументами (если они есть) и количество переменных окружения, переданных процессу. По умолчанию strace не показы вает сами переменные окружения, но его можно попросить выводить более подробную информацию с помощью опции '-v'. Вызов возвратил 0 — значит все ok. В противном случае значение было бы -1.

Следующий интересный системный вызов — access: проверка прав пользователя на файл. В данном случае тестируется существование файла (о чем говорит режим проверки F_OK). На третьей строчке системный вызов вернул значение -1 (ошибка) и вывел ошибку ENOENT (No such file or directory). Это нормально, так как этот файл всего лишь служит для указания линковщику на использование стандартных неоптимизированных версий библиотек.

Как правило, с помощью вызова access проверяются только права на файл или существование самого файла, без каких-либо последующих манипуляций над файлом. Манипуляции над файлом всегда начинаются с системного вызова open, открывающего файл в одном из режимов (O_RDONLY, O_WRONLY или O_RDWR).

Вызов возвращает небольшое целое число — файловый дескриптор, который впоследствии будет использоваться другими вызовами (до того момента, пока не будет закрыт с помощью вызова close).

После открытия файла вызовом open происходит его чтение вызовом read или запись вызовом write. Оба вызова принимают файловый дескриптор, а возвращают количество прочитанных/записанных байт.

Вызов fstat предназначен для получения информации о файле (номер inode, uid, gid и т.д.)

Самый главный вызов в листинге выше — uname, который позволяет получить информацию о текущем ядре. Если трассировка uname занимает всего сотню строк, то трассировка серьезного приложения легко может занимать несколько тысяч строк. Читать такой лог — не самое большое удовольствие. Поэтому иногда лучше записывать в лог только определенные вызовы. Например, чтобы отследить все вызовы open и access (а на них следует обращать внимание в первую очередь при проблемах с запуском приложения):

```
$ strace -e trace=open,access \
-o strace.log uname
```

Вместо перечисления всех нужных вызовов можно использовать классы, состоящие только из специализированных вызовов: file, process, network, signal или ipc. Также можно писать в лог все вызовы, кроме одного.
Например, чтобы исключить вызов mmap:

```
$ strace -e trace=\!mmap -o strace.log uname
```

К сожалению, исключить из вывода сразу несколько вызовов не получится. Некоторые приложения в процессе работы любят наплодить большое количество дочерних процессов. По умолчанию strace игнорирует дочерние процессы, но это поведение можно изменить с помощью опции '-f'.

Если вывод strace пишется в лог, то удобно использовать опцию '-ff', которая заставляет strace писать трассировку каждого процесса в отдельный лог вида filename.PID.
Еще одна весьма полезная возможность strace: с помощью опции '-p' и указания PID можно проводить трассировку работающего процесса. Можно даже соединиться сразу с несколькими процессами, указав опцию '-p' несколько раз. Вот такая конструкция запустит трассировку всех процессов apache:

```
# strace -f $(pidof apache2 | sed 's/\([0-9]*\)/-p \1/g')
```

Чтобы показать всю мощь strace, опишу несколько случаев из моей практики, в которых без помощи этой удивительной утилиты на поиск и устранение проблемы я потратил бы кучу времени.

Проблемы с правами

Давным-давно, когда апач еще был версии 1.3, а PHP — 4, переехал я на новый сервак. И практически сразу вылезла одна проблема — из PHP с помощью обычной функции mail не отправлялись письма. Заглянул в логи индейца — пусто, в логах сендмыла и системных логах — тоже ничего интересного. С точно такими же конфигами apache, PHP и sendmail на другом сервере все работало, значит, причина не в них. Пора расчехлять strace. Остановил апач и запустил трассировку:

```
# strace -f -o /tmp/apache2.strace \
/etc/init.d/apache2 start
```

После того, как скрипт отправки почты (с нехитрым названием mail.php) был несколько раз запущен из браузера, а apache остановлен, можно приступать к анализу лога.

```
$ grep mail.php /tmp/apache2.strace
5345 read(9, "GET /mail.php HTTP/1.1\r\nHost:
12"..., 8000) = 397
5345 stat("/var/www/mail.php", {st_mode=S_IFREG|0644, st_size=256, ...}) = 0
5345 lstat("/var/www/mail.php", {st_mode=S_IFREG|0644, st_size=256, ...}) = 0
5345 open("/var/www/mail.php", O_RDONLY) = 10
...
```

Здесь в каждой строчке первое поле — PID, второе — вызов с параметрами, третье — значение, которое вернул вызов. В большинстве случаев, если возвращаемое значение не отрицательное — вызов отработал без ошибки. То есть, grep по mail.php не дал ничего интересного, кроме PID-процесса (5345), который его обрабатывал. Что ж, запустим grep по PID:

```
$ grep 5345 /tmp/apache2.strace
5340 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f3bf2eada10) = 5345
...
5345 read(9, "GET /mail.php HTTP/1.1\r\nHost:
12"..., 8000) = 397
5345 stat("/var/www/mail.php", {st_mode=S_IFREG|0644, st_size=256, ...}) = 0
...
5345 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f3bf2eada10) = 5347
```

Опять ничего интересного по поводу ошибки. Но на последней строчке с помощью системного вызова clone создается дочерний процесс с PID 5347. Ух ты, квест! Grep по 5347:

```
$ grep 5347 /tmp/apache2.strace
5345 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f3bf2eada10) = 5347
...
5347 execve("/bin/sh", ["sh", "-c", "/usr/sbin/sendmail -t -i "], [/* 6 vars */] = -1 EACCES
(Permission denied)
```

Бинго! Для отправки почты используется /usr/sbin/sendmail, а вызов ругается на отсутствие прав. Причем права на sendmail выставлены корректно, а вот на /bin/sh — нет. Каким-то образом оказалось, что права на /bin/sh были 770 (при владельце и группе root), то есть пользователь www-data (от которого работал apache) не имел прав на выполнение. Корректировка прав исправила это недоразумение.

Проблемы с сетью

Иногда strace позволяет решать сетевые проблемы гораздо быстрее, чем tcpdump. В частности, с помощью strace очень удобно отслеживать, к каким сервисам и в каком порядке обращается приложение для определения имен.

Однажды я сменил IP для одного домена, но, несмотря на то, что dig выдавал мне правильный новый IP, firefox все еще ломился на старый. Трассируем:

```
$ strace -f -e trace=network firefox xakep.ru
7879 socket(PF_FILE, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
7879 connect(3, {sa_family=AF_FILE, path="/var/run/nscd/socket"}, 110) = 0
7879 sendto(3, "\2\0\0\0\r\0\0\0\7\0\0\0passwd\0",
19, MSG_NOSIGNAL, NULL, 0) = 19
```

Вызов connect из листинга показывает, что Firefox сначала обращается к сервису NSCD (кэширующий демон) для разрешения имен, а только потом, если NSCD ничего не выдаст — к DNS. На ноутбуке NSCD только мешается, поэтому я его смело удалил, после чего огнелис нашел правильный айпишник.

Проблемы с псевдоустройствами

Бывает, что какое-то приложение просто виснет, не выдавая никаких ошибок и завершаясь только по kill. Или работает, но тормозит на, казалось бы, простейшей операции. Приведу пример: есть старенький Debian Etch, на нем squid из репозитория с простой NCSA аутентификацией и SAMS для удобного управления. После создания пользователя через SAMS при релоадже squid долго тормозит на операциях добавления пользователей.

```
# strace -f -o /tmp/samsdaemon /etc/init.d/samsd start
...
15773 13:16:03 stat64("/etc/squid/ncsa.sams", {st_mode=S_IFREG|0644, st_size=314, ...}) = 0
15773 13:16:03 open("/etc/squid/ncsa.sams", O_RDONLY|O_APPEND|O_LARGEFILE) = 3
15773 13:16:03 close(3) = 0
15773 13:16:03 open("/dev/random", O_RDONLY) = 3
```

На последнем вызове система задумывается больше, чем на минуту. Значит, проблема в /dev/random. SAMS применяет его для создания хешей паролей пользователей. Самое простое решение — использовать /dev/urandom, который гораздо быстрее, чем /dev/random.

Сажаем nginx в песочницу

Безопасности много не бывает, поэтому никакая дополнительная ступень защиты лишней не будет. Достаточно популярный и простой в реализации механизм минимизации урона от взлома — запуск приложения в chroot. Процесс переноса приложения в песочницу не сложен, если воспользоваться strace и еще одной полезной утилитой — ldd (показывает список совместно используемых библиотек ELF-файла). Покажу на примере, как запускать в chroot популярный на просторах рунета веб-сервер nginx.

Предположим, что nginx (последней на момент написания статьи версии 0.8.40) уже собран с параметрами по умолчанию и лежит в /usr/local. Список библиотек, которые нужны ему для работы:

```
# ldd /usr/local/nginx/sbin/nginx
```

```
linux-gate.so.1 => (0xb7789000)
libcrypt.so.1 => /lib/i686/cmov/libcrypt.so.1
(0xb7751000)
libpcre.so.3 => /usr/lib/libpcre.so.3 (0xb7728000)
libssl.so.0.9.8 => /usr/lib/i686/cmov/libssl.so.0.9.8 (0xb75d4000)
libcrypto.so.0.9.8 => /usr/lib/i686/cmov/libcrypto.so.0.9.8 (0xb7cde000)
libz.so.1 => /usr/lib/libz.so.1 (0xb75bf000)
libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7464000)
libdl.so.2 => /lib/i686/cmov/libdl.so.2 (0xb7460000)
/lib/ld-linux.so.2 (0xb778a000)
```

Переносим эти библиотеки в заранее созданное chroot-окружение (например, /chroot/nginx). Дальше, чтобы удостовериться в том, что у нас есть все необходимые библиотеки, нужно с помощью ldd посмотреть также зависимости скопированных библиотек. Кроме библиотек nginx'у нужны еще некоторые конфиги и логи. Получим список необходимых файлов:

```
# strace -e trace=open /usr/local/nginx/sbin/nginx
open("/etc/ld.so.cache", O_RDONLY) = 3
open("/lib/i686/cmov/libcrypt.so.1", O_RDONLY) = 3
open("/usr/lib/libpcre.so.3", O_RDONLY) = 3
open("/usr/lib/i686/cmov/libssl.so.0.9.8", O_RDONLY)
= 3
open("/usr/lib/i686/cmov/libcrypto.so.0.9.8",
O_RDONLY) = 3
...
open("/etc/passwd", O_RDONLY|O_CLOEXEC) = 4
open("/etc/group", O_RDONLY|O_CLOEXEC) = 4
open("/usr/local/nginx/logs/access.log", O_WRONLY|O_CREAT|O_APPEND|O_LARGEFILE, 0644) = 4
open("/usr/local/nginx/logs/error.log", O_WRONLY|O_CREAT|O_APPEND|O_LARGEFILE, 0644) = 5
```

Скопируем недостающие файлы, удаляя при этом из конфигов ненужную информацию (например, лишних пользователей из /etc/ passwd).

Создадим в chroot-окружении /dev/null, необходимый для нормального функционирования nginx'а:

```
# mknod /chroot/nginx/dev/null c 1 3
```

Вот и все. Теперь запускать nginx в chroot можно следующим образом:

```
# chroot /chroot/nginx/ /usr/local/nginx/sbin/nginx
```

Заключение

Для применения strace есть некоторые ограничения. Во-первых, понятно, что не следует использовать этот инструмент в рабочем окружении (трассировка apache на высоконагруженном productionсервере будет большой ошибкой) — производительность приложения в режиме трассировки сильно снижается. Второе ограничение — это возможные проблемы с трассировкой 32-битных приложений на 64-битной системе. И, наконец, третье — некоторые проги падают при выполнении трассировки вследствие наличия либо багов, либо защиты от трассировок (в основном это касается, конечно, проприетарного софта).

Несмотря на широкие возможности, strace — не «серебряная пуля», он не сможет помочь найти причину абсолютно всех проблем. Однако это очень хороший инструмент, который обязательно нужно попробовать, прежде чем браться за gdb.

Немного истории

Статистика библиотечных вызовов OpenOffice Strace (сокращение от system trace) — это свободное ПО, распространяемое под BSD-подобной лицензией. Утилита была написана в 1991 году Полом Краненбургом для SunOS как аналог утилиты trace. На Linux ее портировал Бранко Ланкестер, который также реализовал поддержку в ядре. В 1992 году вышла версия 2.5 для SunOS, но версия для Linux все еще базировалась на версии 1.5. В 1993 году Рик Сладки объединил strace 2.5 для SunOS и второй релиз strace для Linux, добавив при этом много возможностей от truss из SVR4. В результате появилась strace, которая работала и на Linux, и на SunOS. В 1994 Рик портировал strace на SVR4 и Solaris, а в 1995 — на Irix. Сегодня strace поддерживается большим количеством людей, в списке разработчиков даже успел отметиться сам Линус. Последняя на момент написания статьи версия — 4.5.20 от 14 апреля 2010 года. strace сейчас достаточно активно развивается, в основном добавляется поддержка и фиксятся баги при работе на всяких экзотичных архитектурах.

Инструменты, подобные strace

DTrace — продукт Sun Microsystems, работает на Solaris, FreeBSD и Mac OS X (10.5 и старше). Есть тестовая версия порта для Linux. ktrace — работает на FreeBSD, OpenBSD, NetBSD и Mac OS X (до версии 10.5).

Inotify-tools

Кроме lncron есть еще полезная штука, использующая inotify — inotify-tools, включающая в себя inotifywait и inotifywatch, которые очень удобно использовать в скриптах. Inotifywait просто ждет указанных событий над указанными файлами и завершается с тем или иным кодом возврата. Немного модифицированный скрипт из man'а, хорошо иллюстрирующий предназначение inotifywait:

```
$ cat ~/script.sh
while inotifywait -e modify \
/var/log/apache2/error.log;
do
tail -1 /var/log/apache2/error.log | \
notify-send "Apache needs love!"
done
```

Inotifywatch просто собирает статистику по обращению к определенному файлу/каталогу в течение определенного времени или до прерывания и отображает ее в виде таблицы. Есть возможность сбора статистики только по определенным событиям, задания исключения файлов по маске и чтения списка объектов для мониторинга из файла.

Inotify: мониторинг событий

С помощью strace можно отследить, к каким файлам обращалось конкретное приложение. Но иногда возникает обратная задача — отследить обращения к определенному файлу и выполнить какието действия при этих обращениях. Тогда на помощь придет механизм inotify. Inotify — подсистема ядра, позволяющая отслеживать файловые операции. Технология проверена временем — она была включена еще в ядро 2.6.13 (июнь 2005). Inotify активно используется, например, десктопными поисковиками (вроде Beagle), а также такой полезной штукой, как incron.

Incron — аналог обычного cron с той лишь разницей, что выполнение команды происходит не по времени, а по наступлению указанного в задании события. После установки (incron есть в репозиториях большинства дистрибутивов) создается пустой файл /etc/incron.allow, в котором надо перечислить пользователей, которым разрешено использовать incron.
Создаются задания с помощью команды:

```
$ incrontab -e
```

Формат заданий:

```
<путь> <событие> <команда> (с разделением через пробел)
```

Самые интересные события

- IN_ACCESS — файл был прочитан
- IN_ATTRIB — изменились метаданные файла/каталога
- IN_MODIFY — файл был изменен
- IN_CREATE — файл или каталог был создан в отслеживаемой директории
- IN_DELETE — файл или каталог был удален в отслеживаемой директории
- IN_DELETE_SELF — отслеживаемый файл или каталог был удален
- IN_MOVE — файл был перемещен из отслеживаемого каталога или в него
- IN_ALL_EVENTS — все события

В описании команды можно использовать внутренние переменные. Самые полезные:

- $@ — полное имя отслеживаемого файла/каталога
- $# — относительное имя файла, вызвавшего событие (только при мониторинге каталога)
- $% — название события

Info

- Системные вызовы — это «интерфейс» между ядром и приложением. Ядра Linux ветки 2.6 имеют более 400 различных вызовов.
- Информацию о каждом системном вызове можно найти во втором разделе man. Например, про повсеместно встречающийся вызов open можно посмотреть так: «man 2 open».
- Для работы strace используется системный вызов ptrace.
- Для трассировки библиотечных вызовов есть отдельный инструмент — ltrace.

Links

- strace.sourceforge.net
- www.ltrace.org
- github.com/rvoicilas/inotify-tools

Сквозь тернии к файлам! Новые уязвимости доступа к файлам в PHP

03.07.2011

Оставить мнение

Чтобы оставить мнение, нужно залогиниться

Последние взломы

Закрепляемся в Active Directory. Как сохранить доступ при атаке на домен

Фундаментальные основы хакерства. Идентификация стартового кода и виртуальных функций приложений под Win64

Захват поддоменов. Как я захватил поддомены Microsoft и как работают такие атаки

Кот-призрак. Как эксплуатировать новую RCE-уязвимость в веб-сервере Apache Tomcat

Компьютерные трюки

Рулим дотфайлами. Как быстро повысить комфорт в любом Linux или WSL

Ни единого разрыва. Пишем на C# утилиту для мониторинга сети

Земля слушает. Принимаем и декодируем сигналы спутников Inmarsat и Iridium

NTFS изнутри. Как устроена файловая таблица MFT в Windows

Свежие новости

- 4 hour назад
  В 2019 году стоимость инструментов для фишинга выросла на 149%
- 5 hour назад
  Власти США предлагают 5 млн долларов за информацию о северокорейских хакерах
- 7 hour назад
  СМИ: эксплоит для 0-day уязвимости в Zoom продают за 500 000 долларов
- 20 hour назад
  Check Point: хакеры создают вредоносные «коронавирусные» приложения для Android
- 21 hour назад
  Данные 4 000 000 пользователей Quidd обнаружены в даркнете