From: Spitzer, Andras ([redacted])
Sent: Friday, April 27, 2012 3:45 PM
To: [redacted]
Cc: [redacted]
Subject: Solaris 10, cron jobs exit with ts=13

Mark,

Please find below the result of my investigation.

Symptom :

We experienced unreasonable **cron job failures** on a Solaris host cmfalgadbd02, the unique identifier in this case was that the even though the jobs seemed to be failing, we believed that the jobs are 1. not even started by cron, most likely it dies before it starts 2. it exits **with the error code ts=13**, which means the job quit with UNIX signal #13, which is EPIPE (broken pipe error).

An example of the failure :

```
>  CMD: /util/dba/bin/alert_checker.sh > /dev/null 2>&1
>  oracle 3202 c Mon Apr 23 10:45:00 2012
<  oracle 3202 c Mon Apr 23 10:45:00 2012 ts=13
```

Also I have to emphasize that even though this looks similar to the skipped cron jobs on Linux we experienced lately, this is a totally different case.

Investigation :

Chapter #1

My first idea was obviously to collect all the data possible for this case, analyze data and connect the dots to get a full picture of this behavior. For data collection, my first move was to run truss on the cron daemon, and I wanted to compare a successful run with an unsuccessful run, this also can help me to see exactly what gets EPIPE during the job execution. Whenever I analyze a new unknown issue my **priority** next to understanding the logic behind the symptom **is to find a way to reproduce the issue**. If I can reproduce it, then I understand how does it work, and I also know how to fix it or prevent it in case the fix is out of my reach (for example software bug).

Let's see an unsuccessful truss output from the ts=13 job execution :

```
…
25764/1:        door_call(5, 0xFFBF1730)                        = 0
25764/1:                data_ptr=FFBF1FF0 data_size=8168
25764/1:                desc_ptr=0x0 desc_num=0
25764/1:                rbuf=0xFFBF1FF0 rsize=8192
25764/1:        write(6, " 005020104 B\0", 7)                   Err#32 EPIPE
25764/1:            Received signal #13, SIGPIPE [default]
8266/1:             Received signal #18, SIGCLD, in pollsys() [caught]
8266/1:               siginfo: SIGCLD CLD_KILLED pid=25764 status=0x000D
…
```

Pid 25764 is the child, the executed job, while 8266 is the main process, the cron daemon. As you see right before the write() it has a door_call. The door calls are specific to Solaris, and it allows a process to call a function in a different process via a special file in the filesystem, a door file. This used very frequently within Solaris, I will talk about this more later. So what we see here is that we have a write() command which tries to send 7 byte of data to file descriptor 6, which results an error, EPIPE. Also please take a look at the previous command just before the write, the door_call() function which is using file descriptor 5. This tells me that we are executing a function in the door server process, which may include the code which tries to write to file descriptor 6. In short, **file descriptor 5 may lead us to the process which has the code which tries to write to file descriptor 6**. I know this may sound complicated, but it is

☺.

Ok.. weird. Let's see what we have behind file descriptor 5 and 6, using pfiles :

```
cmfalgadbd02# pfiles 8266
8266:   /usr/sbin/cron
…
   4: S_IFIFO mode:0000 dev:358,0 ino:36404137 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
   5: S_IFDOOR mode:0444 dev:359,0 ino:53 uid:0 gid:0 size:0
      O_RDONLY FD_CLOEXEC  door to ldap_cachemgr[459]
      /var/run/ldap_cache_door
   6: S_IFSOCK mode:0666 dev:356,0 ino:50277 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
        SOCK_STREAM
        SO_SNDBUF(49152),SO_RCVBUF(49152),IP_NEXTHOP(0.0.192.0)
        sockname: AF_INET6 ::  port: 0
```

Hmm. File descriptor 6 is a socket, a SOCK_STREAM type socket which is a TCP socket, but as you can see, it's "empty", meaning it's not connected to anywhere. So the **job died because it tried to write to a socket which used to be an open TCP connection before, but it's not anymore**. Also file descriptor 5 points to the file '/var/run/ldap_cache_door', which is a door to process PID 459, ldap_cachemgr! This tells me that the write() which returns EPIPE most likely an LDAP related write(). Next move, we have to find out what was this TCP socket before, this will help us to understand the function of it. In order to find this out, I simply restarted cron daemon because my assumption was that cron works after a fresh restart, it starts to fail later on. I was right, after restarting cron, that's what I saw :

```
cmfalgadbd02# pfiles 6965
6965:   /usr/sbin/cron
…
   4: S_IFIFO mode:0000 dev:358,0 ino:57096173 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
   5: S_IFDOOR mode:0444 dev:359,0 ino:53 uid:0 gid:0 size:0
      O_RDONLY FD_CLOEXEC  door to ldap_cachemgr[459]
      /var/run/ldap_cache_door
   6: S_IFSOCK mode:0666 dev:356,0 ino:51617 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
        SOCK_STREAM
        SO_SNDBUF(49152),SO_RCVBUF(49640),IP_NEXTHOP(0.0.193.232)
        sockname: AF_INET6 ::ffff:3.154.182.215  port: 38422
        peername: AF_INET6 ::ffff:3.32.197.157  port: 389
```

Well. If the IP would not be familiar enough by itself, the port would definitely help us. **Port 389, LDAP**. The **IP address translates to dsproxy6cinci.corporate.[redacted]** which is one of our LDAP servers. This confirmed my theory that the write() code came from ldap_cachemgr! Until this point it was easy to get, I saw exactly what caused the EPIPE, I saw the forked cron job tries to write to LDAP, as the LDAP connection is not alive anymore, it received an EPIPE and quits. But this was not enough. First, I don't know how to prevent this, second, I still don't know how to reproduce it, third which is probably the most important, I see only the symptom, not the root cause. (These two are very different, even though lazy people tend to believe it's the same)

My next step was to **set up an infinite shell loop with 1 second sleep for capturing data** for

1. pfiles <cron PID>
   I wanted to see exactly **when will the connection disappear from file descriptor 6**, and my plan was to look for a consequence at the same time somewhere near the LDAP configuration
2. ldap_cachemgr –g
   This daemon is responsible for maintaining an LDAP cache/proxy within Solaris between the LDAP servers and any process which would like to use LDAP
3. 3. netstat –na | grep <38422>

I wanted to see at the TCP level as well as file descriptor 6 disappears, I should see the TCP connection to disappear as well here

After running this loop for a day, I was hoping to find some useful data. I started to see something interesting. Just **right before cron started to drop ts=13 in the log file, the primary LDAP server went to REMOVED** :

```
Server information:
  Previous refresh time: 2012/04/21 01:43:03
  Next refresh time:     2012/04/21 02:40:10
  server: 3.32.197.157, status: REMOVED
  server: 3.34.169.233, status: UP
Cache data information:
  Maximum cache entries:          256
  Number of cache entries:          0
```

and in the same time cron stopped logging ts=13 as the primary server came back from REMOVED to UP.. For the first blink you may say, “hey, this makes sense”, but for the second you should say “no, it does not”. For one, why would cron fail when we have an LDAP server UP yet? Next I decided to scan our environment for similar symptoms, and I had to realize that this is a very rare symptom, meaning I saw ts=13 error codes in cron logs in about 5 servers out of 500. Very rare, still more than one. So there is a reasonable logic in the background, I just have to find the triggers and the conditions. I started to think about what else tool I could use, DTrace immediately popped into my mind, although after a few tests I had to abandon it. Why? Because I had to enable many thousands of probes so I had to filter (predicate) based on PID, but the problem is that today DTrace can’t follow forks. There is an open RFE for this functionality, today it’s not implemented yet. And as I wanted to trace not only the cron daemon, but the executed jobs too, this follow fork feature became a requirement for me. I had a different idea, I can test. I can **block the primary LDAP server using ipf, and see if ts=13 starts showing up or not**.. ?

```
# svcadm enable ipfilter
```
We load the ipf kernel module

```
# echo “block out proto tcp from any to 3.32.197.157/32 port = ldap” | ipf –If –
```
We load the LDAP blocking rule to ipf’s inactive ruleset. If it works and looks good, we can simply swap between the active and inactive rulesets..

```
#ipf –s
```
Swap the active/inactive rulesets

And yes! **As I blocked the primary LDAP, the ldap_cachemgr after a few secs showed the server as REMOVED**, and ts=13 started to show up in the cron logs.. At this point I thought I know the trigger, looks so trivial. I was wrong. I configured ipf rules on a different host, blocked the primary LDAP server, ts=13 didn’t show up.. ok I have to continue. We got an ingredient for the sauce, but we don’t have the full recipe yet.

Chapter #2

While I was comparing servers where we don’t see these symptoms with the few where we do see them **I had to realize two things that are different**.

System with ts=13 symptom :

```
cmfalgadbd02# pgrep cron
6965
cmfalgadbd02# pldd 6965 | wc -l
      56
```

System without a ts=13 symptom :

```
sysas2t@cmfalgadbp12[$PWD]: pgrep cron
2251
sysas2t@cmfalgadbp12[$PWD]: pldd 2251 | wc -l
      17
```

What do we see here? **For some reason the cron experiencing the ts=13 error has way more dynamic libraries loaded**, compared to a cron which does not experience this problem.. Same binary, but when you start one of them using 3 times more dynamic libraries..

The second unusual thing I saw was this :

System ts=13 symptom :

```
   5: S_IFDOOR mode:0444 dev:359,0 ino:53 uid:0 gid:0 size:0
      O_RDONLY FD_CLOEXEC  door to ldap_cachemgr[459]
      /var/run/ldap_cache_door
   6: S_IFSOCK mode:0666 dev:356,0 ino:51617 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
        SOCK_STREAM
        SO_SNDBUF(49152),SO_RCVBUF(49152),IP_NEXTHOP(0.0.192.0)
        sockname: AF_INET6 ::  port: 0
```

System without ts=13 symptom :

It was missing these two file descriptors, the door to ldap_cachemgr and the direct TCP connection to the LDAP!

Why? Couldn't tell. So at this point I had the **following questions I had to answer** :

- now we know that the write() function which makes the cron job quit tries to send a message to LDAP. What message is this? (it seems too short for an LDAPMessage)
- we know that only the failing cron even though it's the same binary executable it has way more dynamic libraries loaded. Why it has more dynamic libraries loaded?
- we know that only the failing cron has an open file descriptor to '/var/run/ldap_cache_door'. Why only the failing cron has this open file descriptor?

At this point I had 20+ truss outputs, from cron job execution, cron startup, pam debug, nss debug, ldap_cachemgr debug outputs, etc. Next, I created a very long detailed debug output from the cron library calls, and wrote a small script which processed into a human-readonable format, so I can see exactly which library calls the failing write() function :

http://src.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/cmd/cron/cron.c#3313

```
  3313      r = pam_setcred(pamh, PAM_ESTABLISH_CRED);
```

The trace starts from the cron daemon itself, as cron calls pam_setcred.

```
2   22222/1@1:  -> libpam:pam_setcred(0x352e0, 0x1, 0x0, 0x15050)
4   22222/1@1:    -> libpam:run_stack(0x352e0, 0x1, 0x1, 0x10)
6   22222/1@1:      -> pam_unix_cred:pam_sm_setcred(0x352e0, 0x1, 0x0, 0x0)
8   22222/1@1:        -> libproject:getdefaultproj(0x39cd8, 0xffbfd168, 0xffbfc168,
0x1000)
10  22222/1@1:          -> libsecdb:getusernam(0x39cd8, 0xffbf80a4, 0xffbfc168, 0x1000)
12  22222/1@1:            -> libnsl:_getusernam(0x39cd8, 0xffbf7fd8, 0xffbf5fd8,
0x2000)
14  22222/1@1:              -> libc:nss_search(0xff126a38, 0xff0acd60, 0x4, 0xffbf5f40)
16  22222/1@1:                -> nss_ldap:getbyname(0x43c30, 0xffbf5f40, 0xff273700,
0xff142a00)
18  10342/1@1:                  -> nss_ldap:_nss_ldap_lookup(0x43cc0, 0xffbf5f40,
0xfeeef10c, 0xffbf5c70)
20  10342/1@1:                    -> libsldap:__ns_ldap_list(0xfeeef10c, 0xffbf5c70,
0xfeedb300, 0xfeeeec74)
22  10342/1@1:                      -> libsldap:init_search_state_machine(0x0, 0x0,
0x0, 0x0)
24  22222/1@1:                        -> libsldap:get_current_session(0x35510,
```

```
0xfee8d1bc, 0x3, 0x260)
26  22222/1@1:                          -> libsldap:__s_api_getConnection(0x0, 0x0,
0x0, 0xffbf51cc)
28  22222/1@1:                            -> libsldap:makeConnection(0xffbf5154, 0x0,
0x3ba98, 0xffbf51cc)
30  22222/1@1:                              -> libsldap:findConnection(0x0, 0x0,
0x3ba98, 0xffbf48d4)
32  22222/1@1:                                ->
libsldap:__s_api_requestServer(0xfeea706c, 0x0, 0xffbf4054, 0xffbf406c)
```

At this point the process can go two different way**, a good way and a bad way** … the bad way :

```
32  22222/1@1:                                -> libsldap:_DropConnection(0x400, 0x400,
0x1, 0x2f3a0a)
34  22222/1@1:                                  -> libldap:ldap_ld_free(0x39f08, 0x0,
0x0, 0x1)
36  22222/1@1:                                    ->
libldap:nsldapi_free_connection(0x39f08, 0x392f0, 0x0, 0x0)
38  22222/1@1:                                      ->
libldap:nsldapi_send_unbind(0x39f08, 0x3c4d8, 0x0, 0x0)
40  22222/1@1:                                        ->
libldap:nsldapi_ber_flush(0x39f08, 0x3c4d8, 0x3bf48, 0x1)
42  22222/1@1:                                          -> libldap:ber_flush(0x3c4d8,
0x3bf48, 0x1, 0x0)
44  22222/1@1:                                            -> libnspr4:pt_Send(0x30540,
0x3c01c, 0x7, 0x0)
46  22222/1@1:                                              -> libc:write(0x6, 0x3c01c,
0x7, 0x0)
48  22222/1@1:                                                -> libc:_write(0x6,
0x3c01c, 0x7, 0x0)
    22222/1:    write(6, " 005020104 B\0", 7)                   Err#32 EPIPE
```

And the good way :

```
10342/1@1:                                ->
libsldap:__ns_ldap_freeParam(0xffbf4064, 0x2f, 0x2f, 0x2f3a0a)
10342/1@1:                                <- libsldap:__ns_ldap_freeParam() = 0
10342/1@1:                                ->
libsldap:__s_api_free_server_info(0xffbf4054, 0x31610, 0x2f, 0x2f3a0a)
10342/1@1:                                <- libsldap:__s_api_free_server_info() =
0xffbf4054
```

So the EPIPE which is causing us headache comes from the cron daemon which calls pam_setcred function (this is a PAM function), which calls nss_ldap library (NSS call) which calls libsldap/libldap library which then calls the write() which will return with EPIPE. Ok, so cron calls PAM, PAM calls NSS, NSS calls libldap, which fails at some point. This is a libldap library bug. If you read the functions' name and their code in src.opensolaris.org you will see that **the problem is that whenever the primary LDAP server goes down, the next time the cron tries to fork() the libsldap/libldap tries to cleanly disconnect from the LDAP server** (DropConnection, ldapi_send_unbind), but the connection is not alive anymore! So it goes to broken pipe, EPIPE!

That explains why ts=13 disappears when **1.** we still have the "empty socket" at file descriptor 6, but 2. the primary LDAP becomes UP again! This way the forked process won't try to unbind (even though the LDAP connection is dead long time ago!), so it won't break.

So this is obviously an libsldap/libldap bug, they handle this scenario badly, they should be prepared for a not-connected socket before they try to disconnect!

And we are not even finished yet..

Chapter #3

At this point we understand more from the puzzle, we know there is a libsldap/libldap bug which can be triggered if the primary LDAP connection goes down under certain circumstances. Let me **sum up quickly what is happening so far** :

1. cron daemon starts (for some reason with a lot of dynamic libraries)
2. cron daemon opens a door handler file descriptor to /var/run/ldap_cache_door (for example at file descriptor 5)
3. in the same time cron opens a persistent LDAP connection (for example at file descriptor 6)
4. after a while the connection which was opened at point 3 will tear down (most likely LDAP server initiated disconnect because of inactivity), file descriptor 6 will become a not-connected socket!
5. cron can fork (execute) jobs for as long as the connection we opened at 3. is still in open state, if that connection goes away and the LDAP server goes to REMOVED state, we start seeing ts=13 errors

My next goal was to find out why we see more libraries using pldd in the failing cron and /var/run/ldap_cache_door file descriptor using pfiles. This tells me that for some reason even the cron daemon is using LDAP via NSS services. **If I could tell why the failings crons keep open '/var/run/ldap_cache_door' in the first place** I could reconfigure so it would not use '/var/run/ldap_cache_door' and I would achieve that cron daemon would not have any LDAP connected socket which later on can cause us trouble under certain cirtumstances, when that connection tears down and the LDAP server gets unreachable even temporarily.

So how can I make cron not to use direct LDAP connection via NSS? This is not something you can configure, so I had to brainstorm with myself when cron can turn to NSS. NSS is used for name service lookups, userid, groupid translation. My thought was that cron is a very simple daemon, it loads the cron files from /var/spool/cron, then goes to sleep, until the first scheduled job's time come. So when I assume to have any NSS interaction? Only during the initial cron file loads from /var/spool/cron.

```
cmfstctdmetreo1(sysas2t): grep passwd /etc/nsswitch.conf
passwd:    files ldap lsass
cmfstctdmetreo1(sysas2t): pwd
/var/spool/cron/crontabs
cmfstctdmetreo1(sysas2t): ls -la
total 18
drwxr-xr-x   2 root     sys          512 Apr 26 20:33 .
drwxr-xr-x   4 root     sys          512 Mar  5  2008 ..
-rw-------   1 root     sys          190 Jul 13  2007 adm
-r--------   1 root     root         452 Jan 21  2005 lp
-rw-------   1 root     metreo       189 Sep 12  2011 metreo
-rw-------   1 root     root         939 Aug  5  2011 root
-rw-------   1 root     sys          308 Jul 13  2007 sys
-rw-------   1 root     superusr     104 Apr 26 08:41 sysas2t
-r--------   1 root     sys          404 Feb 29  2008 uucp
cmfstctdmetreo1(sysas2t): pfiles `pgrep cron` | grep -c "/var/run/ldap_cache_door"
0
cmfstctdmetreo1(sysas2t): pldd `pgrep cron` |wc -l
      18
```

Under /var/spool/cron/crontabs each file is a **cron file, the file names represents the user who should run the commands**. So the filename 'adm' will run as user 'adm', etc. The files you see in this example are all local files.. hence cron does not have to go to LDAP via NSS. You see this is a cron daemon with no door file descriptor to '/var/run/ldap_cache_door' and with the small amount of dynamic libraries loaded.. ! These are the symptoms for a correct cron which is not affected by the libsldap bug.. ! Now, **if I'm right, I have to create a bogus username** (which won't be local), **so cron will have to contact LDAP as well** during startup, as we have the 'ldap' keyword in nsswitch.conf and as 'files' won't find the bogus user it will go to ldap and lsass databases as well.

Let's try :

```
cmfstctdmetreo1(sysas2t): touch nosuchuser
cmfstctdmetreo1(sysas2t): svcadm restart cron
cmfstctdmetreo1(sysas2t): pfiles `pgrep cron` | grep -c "/var/run/ldap_cache_door"
1
```

```
cmfstctdmetreo1(sysas2t): pldd `pgrep cron` |wc -l
      56
```

!!!! The **assumption was correct**. I created an empty cron file (which contains no commands) with the name 'nosuchuser'. **When cron starts up, it will read this file and will query NSS for user info for 'nosuchuser'**, as it's not a local user (files) it will go to LDAP (ldap) and Likewise (lsass) as well. And this cron is connected to LDAP and it has a large amount of dynamic libraries loaded, hence … it is potentially can be affected by the libsldap bug! Why the failing cron loads more library you may ask? Because it has to load the extra 'ldap', and 'sasl' libraries for bogus (non-local) name lookups during startup.

Ok, so now I **found how can we identify if a cron daemon is potentially can be affected** by this bug or not, and I also described how can we make sure the cron can't be affected.. **we have to make sure all the files under /var/spool/cron are local username** files..

Now, let's see our specific case at cmfalgadbd02 :

```
cmfalgadbd02# ls -al /var/spool/cron/crontabs
total 36
drwxr-xr-x   2 root     sys          512 Apr 23 10:48 .
drwxr-xr-x   4 root     sys          512 Mar 17  2008 ..
-rw-------   1 root     sys          190 Jul 13  2007 adm
-r--------   1 root     root         452 Jan 21  2005 lp
-rw-------   1 root     superusr       0 Mar 30 09:24 mgn210
-rw-------   1 root     dba         7706 Apr 20 08:56 oracle
-rw-------   1 root     root         352 Mar 30 09:23 root
-rw-------   1 root     root         309 Apr 27  2010 root.backup.07052010
-rw-------   1 root     sys          308 Jul 13  2007 sys
-rw-------   1 root     superusr      54 Apr 23 10:48 sysas2t
-rw-------   1 root     superusr      91 Apr 23 09:26 sysrlpt
-r--------   1 root     sys          404 Feb 29  2008 uucp
```

Yey! There you go.. I see two files here which are not local users.. one is **mgn210** (0 size file, should be removed..) and **root.backup.07052010**. Now this is a very bad practice. I understand this tend to be a backup of the root cron, but cron is a dumb daemon, it has no idea it's just a backup file, hence it will try to process it.. so we immediately found 2 usernames which will be looked up by NSS during startup at LDAP because they are not local. Btw both files should be removed/moved, as the first is a 0 size file, the second is a backup file. So we have to remove/move these two files and restart cron to fix this..

Btw just to double confirm my theory, **here is the truss output when cron asks the LDAP whether it has info about the user 'root,backup.07062010'** :

```
264/1:          write(6, 0x0003C4E4, 210)                       = 210
264/1:             081CF020102 c81C9041E o u = p e o p l e , o u = c o r p u n i x
264/1:             , o = g e . c o m\n0102\n01030201\00201050101\0A0 <A31D04\v o b
264/1:             j e c t C l a s s040E g e p o s i x a c c o u n tA31B0403 u i d
264/1:            0414 r o o t . b a c k u p . 0 7 0 5 2 0 1 0 0 Z0402 c n0403 u i
264/1:             d04\t u i d n u m b e r04\t g i d n u m b e r0405 g e c o s04\v
264/1:             d e s c r i p t i o n0413 g e u n i x h o m e d i r e c t o r y
264/1:            0410 g e u n i x l o g i n s h e l l
```

Doesn't look good.. Anyway, we almost got all the pieces. If you go through these chapters, and you'll try to reproduce the issue, you will still fail ☺ Simply because we were missing one final piece :

If you **want to test it, reproduce it, please disable nscd**. If we have nscd running, it can save you from starting up cron in a failing state.

Chapter #4

Let's **sum up what we know so far**, let's put the puzzles together.

If you are one of the few who got to this part of the e-mail (I know it's long), you'll understand that this is not a simple bug, it has to have a certain environment where many small pieces has to be in a specific state so the bug can be hit. This explains why we see this bug very rarely hit in our environment. Btw, I remember we had similar issue with Netbackup long time ago (long running bpbkar process randomly received an EPIPE and quit), I can't say firmly we hit the same bug there, but it's very likely, I remember I conducted some investigation back then but I didn't go as deep as I did this time. Also, if you have nscd enabled, you'll reduce the overall chance of hitting the bug, and nscd should be enabled on all of our servers by default (nscd is very important). It is important to tell as well that **now it's proved the issue is not related to Likewise in any way**, this is a pure libsldap/libldap bug. Now as we got all the separate pieces, it's time to put them together using an example, which may help you to guide through what happens :

The system I will use to demonstrate this bug has never experienced the ts=13 ever before.

Preparation :

We have to **make sure nscd is disabled** :

```
sysas2t@gisadmin05[$PWD]: svcadm disable name-service-cache
sysas2t@gisadmin05[$PWD]: svcs name-service-cache
STATE          STIME    FMRI
disabled        9:52:54 svc:/system/name-service-cache:default
sysas2t@gisadmin05[$PWD]: pgrep nscd
sysas2t@gisadmin05[$PWD]:
```

We **confirm the cron daemon currently running has no open file descriptor to '/var/run/ldap_cache_door'**, and that **it has the lower amount of dynamic libraries loaded** :

```
sysas2t@gisadmin05[$PWD]: pfiles `pgrep cron` | grep -c "/var/run/ldap_cache_door"
0
sysas2t@gisadmin05[$PWD]: pldd `pgrep cron` |wc -l
     18
```

Looks good. Now we **confirm the host is LDAP enabled** :

```
sysas2t@gisadmin05[$PWD]: grep passwd /etc/nsswitch.conf
passwd:    files ldap
```

Correct. We **check the status of the LDAP servers configured** :

```
sysas2t@gisadmin05[$PWD]: /usr/lib/ldap/ldap_cachemgr -g | grep server:
  server: 3.32.197.157, status: UP
  server: 3.34.169.233, status: UP
```

Check. Now I **check if the ipfilter is enabled**, and **I add a rule to the inactive ruleset** (upper case I will indicate that) a rule which **blocks the first LDAP server**, we'll use this ruleset to simulate an LDAP server forced disconnect. The ruleset we add not is inactive, so it's not alive yet! We'll swap the active/inactive ruleset later on, when we want to activate the block, now we just add it :

```
sysas2t@gisadmin05[$PWD]: echo "block out proto tcp from any to 3.32.197.157/32 port =
389" | ipf -If -
sysas2t@gisadmin05[$PWD]: ipfstat -Iio
block out proto tcp from any to 3.32.197.157/32 port = ldap
empty list for inactive ipfilter(in)
```

Ruleset added to the inactive list.

Last but not least, I **created a simple cron job under my account which runs every minute** :

```
sysas2t@gisadmin05[$PWD]: crontab -l
* * * * *       /bin/true
```

I use the job to run /bin/true, because as I ran a lot of trace I wanted to have the least “noise” possible on the execution thread, and /bin/true is one of the simplest job ever (it exits immediately with 0, it’s a one liner exit(0) in C).

Step #1

We have to **create a file under /var/spool/cron/crontabs with the name which is definitely not a local user ID**. After restarting cron this will enforce cron to issue NSS query to ‘files’ first then ‘ldap’ as well :

```
sysas2t@gisadmin05[$PWD]: cd /var/spool/cron/crontabs
sysas2t@gisadmin05[$PWD]: ls -la
total 16
drwxr-xr-x   2 root     sys          512 Apr 27 09:54 .
drwxr-xr-x   4 root     sys          512 May 28  2008 ..
-rw-------   1 root     sys          190 Jul 13  2007 adm
-r--------   1 root     root         452 Jan 22  2005 lp
-rw-------   1 root     other        391 Mar 30  2010 root
-rw-------   1 root     sys          308 Jul 13  2007 sys
-rw-------   1 root     superusr      20 Apr 25 16:31 sysas2t
-r--------   1 root     sys          404 Feb 29  2008 uucp
sysas2t@gisadmin05[$PWD]: touch nosuchuser
sysas2t@gisadmin05[$PWD]: chmod go-r nosuchuser
sysas2t@gisadmin05[$PWD]: ls -la
total 16
drwxr-xr-x   2 root     sys          512 Apr 27 10:05 .
drwxr-xr-x   4 root     sys          512 May 28  2008 ..
-rw-------   1 root     sys          190 Jul 13  2007 adm
-r--------   1 root     root         452 Jan 22  2005 lp
-rw-------   1 root     superusr       0 Apr 27 10:05 nosuchuser
-rw-------   1 root     other        391 Mar 30  2010 root
-rw-------   1 root     sys          308 Jul 13  2007 sys
-rw-------   1 root     superusr      20 Apr 25 16:31 sysas2t
-r--------   1 root     sys          404 Feb 29  2008 uucp
```

Ok, we **created our empty file called ‘nosuchuser’**. This will force cron to query ‘files’ and ‘ldap’ for the user info of user ‘nosuchuser’. Let’s see if we are right :

```
sysas2t@gisadmin05[$PWD]: svcadm restart cron
sysas2t@gisadmin05[$PWD]: pfiles `pgrep cron` | grep -c "/var/run/ldap_cache_door"
1
sysas2t@gisadmin05[$PWD]: pldd `pgrep cron` |wc -l
      46
```

Yep! There you go. **cron now has an open file descriptor with /var/run/ldap_cache_door** (process ldap_cachemgr), and **it has more libraries loaded**. Now, let’s see our direct LDAP connection from cron :

```
sysas2t@gisadmin05[$PWD]: pfiles `pgrep cron`
14702:  /usr/sbin/cron
…
   6: S_IFSOCK mode:0666 dev:355,0 ino:56465 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
        SOCK_STREAM
        SO_SNDBUF(49152),SO_RCVBUF(49640),IP_NEXTHOP(0.0.193.232)
        sockname: AF_INET6 ::ffff:3.215.144.23  port: 41833
        peername: AF_INET6 ::ffff:3.32.197.157  port: 389
…

sysas2t@gisadmin05[$PWD]: netstat -na | grep 41833
3.215.144.23.41833   3.32.197.157.389     49640      0 49640      0 ESTABLISHED
```

Yep, **you can see we have a persistent LDAP connection from cron daemon to the LDAP server**.

my test **cron job still works like a charm**, no issues so far :

```
…
>  CMD: /bin/true
>  sysas2t 15016 c Fri Apr 27 10:16:00 2012
<  sysas2t 15016 c Fri Apr 27 10:16:02 2012
>  CMD: /bin/true
>  sysas2t 15028 c Fri Apr 27 10:17:00 2012
<  sysas2t 15028 c Fri Apr 27 10:17:01 2012
…
```

Step #2

Now even though cron has the open file descriptor 6 toward LDAP, remember it used during startup for NSS lookup, it will never use again, simply because the cron daemon does nothing else after startup but sleeps and executes jobs. Unless **we force the connection to go down**, which **will make the libsldap/libldap to try to disconnect** (unbind) whenever a job execution start, and bang **there we'll hit the EPIPE**. This is my theory. Let's force the cron's file descriptor 6 LDAP TCP connection to tear down with our ipf ruleset we prepared.. :

```
sysas2t@gisadmin05[$PWD]: ipf -s
Set 1 now inactive
sysas2t@gisadmin05[$PWD]: ipfstat -io
block out proto tcp from any to 3.32.197.157/32 port = ldap
empty list for ipfilter(in)
```

Ruleset swapped (you remember, we prepared the Inactive ruleset previously), now we have the **primary LDAP blocked**. Now, we have to wait a few secs here.. Let's check

```
sysas2t@gisadmin05[$PWD]: /usr/lib/ldap/ldap_cachemgr -g | grep server:
  server: 3.32.197.157, status: REMOVED
  server: 3.34.169.233, status: UP
```

Ok, **ldap_cachemgr already detected the block**, although the LDAP connection between cron and the LDAP server needs more time to tear down, **it's still ESTABLISHED** :

```
sysas2t@gisadmin05[$PWD]: netstat -na | grep 41833
3.215.144.23.41833   3.32.197.157.389     49640      6 49640      0 ESTABLISHED
```

Now **we just have to wait** a few more secs.. while we wait, this is a very interesting state now. Even though we have the primary LDAP connection blocked, the forked jobs still can run without a problem :

```
>  CMD: /bin/true
>  sysas2t 15222 c Fri Apr 27 10:25:00 2012
<  sysas2t 15222 c Fri Apr 27 10:25:01 2012
```

This is because even though they use NSS as well, they query for an available LDAP server and ldap_cachemgr will simply give them the second one, which is UP, 2.34.169.233 (you can see that if you enable debugging with /usr/lib/ldap/ldap_cachemgr –d6), so there is no issue whatsoever. What **we need is that the file descriptor 6** which **is** a direct connection to LDAP from the cron main process and which will never be used again by cron (remember? it was used only during startup to look up the bogus usernames in ldap) **to disconnect**…

<after ~8 minutes>

Finally, **we got the LDAP connection disconnected**… :

```
sysas2t@gisadmin05[$PWD]: pfiles `pgrep cron`
14702:  /usr/sbin/cron
…
  6: S_IFSOCK mode:0666 dev:355,0 ino:56465 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
        SOCK_STREAM
        SO_SNDBUF(49152),SO_RCVBUF(49152),IP_NEXTHOP(0.0.192.0)
```

```
        sockname: AF_INET6 ::  port: 0
…
sysas2t@gisadmin05[$PWD]: netstat -na | grep 41833
sysas2t@gisadmin05[$PWD]:
```

And from now, **we should see ts=13 to appear in the cron log** :

```
>  CMD: /bin/true
>  sysas2t 15369 c Fri Apr 27 10:30:00 2012
<  sysas2t 15369 c Fri Apr 27 10:30:01 2012
```

<Here we got the LDAP connection disconnected>

```
>  CMD: /bin/true
>  sysas2t 15380 c Fri Apr 27 10:31:00 2012
<  sysas2t 15380 c Fri Apr 27 10:31:00 2012 ts=13
>  CMD: /bin/true
>  sysas2t 15405 c Fri Apr 27 10:32:00 2012
<  sysas2t 15405 c Fri Apr 27 10:32:00 2012 ts=13
```

YES! There you go. **Bug confirmed**. As we have the LDAP connection disconnected, whenever cron tries to execute a job, at some point it will get to libsldap/libldap (via pam_setcred -> nss_ldap -> libsldap/libldap) which as it detected the LDAP TCP connection loss, it will try to cleanly disconnect (unbind) from it. At this point this is unreasonable, as the file descriptor 6, which used to be the TCP socket toward the LDAP is DOWN, so the unbind attempt receives an EPIPE, and the job execution will abort immediately, even before the job would have started.

Summary :

Under very specific circumstances, when :

1. we have LDAP configured in /etc/nsswitch.conf for ‘passwd’
2. we have files under /var/spool/cron/crontabs which names can’t be resolved locally (which will force cron to go to LDAP which will force cron to set up a persistent LDAP connection)
3. the persistent connection cron had set up during startup disconnects (this can take a while, we forced it with ipf)
4. the primary LDAP becomes unavailable for any reason, even temporarily

cron job execution fails with ts=13 what we’ll see in the cron log file. This is a libsdlap/libldap bug, as it tries to disconnect from the LDAP even though the TCP connection to LDAP is gone way before, resulting a broken pipe (EPIPE).

Workaround :

If we can ensure that cron will not use nss_ldap during startup, this bug won’t hit cron. We can ensure that by making sure all the filenames under /var/spool/cron/crontabs are local usernames. We also have to make sure nscd is running, as nscd can reduce the chance of hitting the bug, even if fail the previous condition.

You can confirm whether the running cron is a potential candidate for this bug to hit by running pfiles and if you have a line with ‘/var/run/ldap_cache_door’, there is a chance you’ll see this bug occasionally :

```
sysas2t@gisadmin05[$PWD]: pfiles `pgrep cron` | grep "ldap_cache_door"
      /var/run/ldap_cache_door
```

If you don’t see this line, your cron is safe from this bug.

Now, **finally, the actual fix for cmfalgadbd02** :

we **confirm the cron is a potential candidate for the bug** to hit :

```
cmfalgadbd02# pfiles `pgrep cron` | grep -c "ldap_cache_door"
1
```

Confirmed. Now **we cleanup /var/spool/cron/crontabs**, by getting rid of the two files that are not local (neither makes sense) :

```
cmfalgadbd02# rm mgn210
cmfalgadbd02# mv root.backup.07052010 ~sysas2t
cmfalgadbd02# ls -la
total 34
drwxr-xr-x   2 root     sys          512 Apr 27 04:48 .
drwxr-xr-x   4 root     sys          512 Mar 17  2008 ..
-rw-------   1 root     sys          190 Jul 13  2007 adm
-r--------   1 root     root         452 Jan 21  2005 lp
-rw-------   1 root     dba         7706 Apr 20 08:56 oracle
-rw-------   1 root     root         352 Mar 30 09:23 root
-rw-------   1 root     sys          308 Jul 13  2007 sys
-rw-------   1 root     superusr      54 Apr 23 10:48 sysas2t
-rw-------   1 root     superusr      91 Apr 23 09:26 sysrlpt
-r--------   1 root     sys          404 Feb 29  2008 uucp
```

Done. **Now we only have files which are local usernames**. Let's restart cron, and see if it fixes anything :

```
cmfalgadbd02# svcadm restart cron
cmfalgadbd02# pfiles `pgrep cron` | grep -c "ldap_cache_door"
0
```

Done. Our **cron did not go to 'ldap' nor to 'lsass' during startup** because all the filenames are local usernames, which was able to translate using 'files'. Just for the extra mile, **I enable the nscd** because it's important ☺ (this system had no nscd running for some reason..) :

```
cmfalgadbd02# svcs name-service-cache
STATE          STIME    FMRI
disabled       Apr_20   svc:/system/name-service-cache:default
cmfalgadbd02# svcadm enable name-service-cache
cmfalgadbd02# svcs name-service-cache
STATE          STIME    FMRI
online          4:52:31 svc:/system/name-service-cache:default
```

And we are done! This **system should not experience the ts=13 issue again**. (If it will I will bite my feet and continue the investigation, although I don't expect this to happen. This issue is complex enough already ☺)

Regards,
sendai