From: Spitzer, Andras ([redacted])
Sent: Thursday, May 09, 2013 12:48 PM
To: [redacted] ([redacted])
Cc: [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted])
Subject: LDAP clients with overwhelming connection
Attachments: nscd_check.7z

Bob,

I'm finished with my analysis, I used the list you sent me two days ago to investigate the reason why a few UNIX servers still have a huge amount of LDAP connections per day even though NSCD is running and this also keep happening if we restart the NSCD daemon. Sorry for the lengthy e-mail, but I want to prove my point so you'll see the conclusion at the end.

First, let me group the servers based on OS, since the root cause is different for Linux and for Solaris. I'll start with the Linux because the problem is less complex here.

Linux

There is an SSL bug which cause the LDAP connection to be dropped by the LDAP server, the symptom is clearly visible in the LDAP server logs, for example :

[01/Apr/2013:00:00:01 -0400] - DISCONNECT - INFO - conn=2237010 reason="other" msg="Exception caught while polling client connection LDAPS.192.35.38.171.38080 -- javax.net.ssl.SSLException: Inbound closed before receiving peer's close_notify: possible truncation attack?"

I wrote about this a year ago in my e-mail "LDAP log processing script 0.1" :

"*DISCONNECT statistics

This section shows us the disconnect statistics. Whenever an LDAP connection disconnected, there will be a status code which will give us details about the circumstances. "unbind" means the client successfully and cleanly disconnected, both client and server agreed to disconnect. Anything else is something we should investigate. One thing I spotted here immediately is the second 23.2%, which is the "other" category and is having the message : "Exception caught while polling client connection <IP> -- javax.net.ssl.SSLException: Inbound closed before receiving peer's close_notify: possible truncation attack?"

This is probably a bug in DSP (Oracle support doc [ID 1450215.1]), it's an SSL handshake error, so the LDAP connection can't even be built. The client most likely will retry, but this is a waste of resource on both client and server sides, also it can cause minor glitches in the client side. As the document says this bug was fixed in DSP 6.3.1.1.1. Again, this is just the output of a sample log file, we'll have to analyze more logs before we consider taking actions. "

The Oracle support article states that this bug was fixed in DSP 6.3.1.1.1, which is the same as our version, so we have to ask Oracle for further support on this. The point is that the LDAP Proxy Server aborts the LDAP connection so the Linux server has to reconnect, there is not even a single UNBIND received by the server from the client, it will just DISCONNECT the client. That's why it has a huge amount of connects, Linux tries to reconnect.

Solaris

This one is even more interesting. The problem here comes from the fact that under certain circumstances one NSS query will use a single LDAP connection. It will CONNECT, BIND, SEARCH, UNBIND, DISCONNECT. This is very expensive as we know, because we prefer to have a single LDAP connection open and all the SEARCHes through that single LDAP connection. So why is this unusual behavior?

During my tests I realized that even on the same server, sometimes my NSS query does use a separate LDAP connection, sometimes it does not. For tests I used simple commands like 'ls –la' and 'id ag003902', any command will work if it triggers name resolution. I ran snoop to see the LDAP traffic, and I saw sometimes my query opens a new connection and tears it down after the result, sometimes it's using the currently opened connection by ldap_cachemgr. ldap_cachemgr is responsible for keeping a single LDAP connection open to all the configured LDAP servers, no user process should contact LDAP directly, that's what ldap_cachemgr and NSCD are for.

The system I tested on had the following passwd: order in /etc/nsswitch.conf :

```
passwd: files ldap lsass
```

Here is an **example when things work fine** :

nss_search() is the function in the C library (libc) which will parse /etc/nsswitch.conf and query the configured databases looking for the key you want to resolve
here you already see that nss_search() calls _nsc_search() which stands for Name Service Cache, so the use of NSCD is built into the name resolution library

```
26613/1@1:        -> libc:nss_search(0xff345700, 0xff24cab0, 0x5, 0xffbfeb60)
```

```
26613/1@1:          -> libc:_nsc_search(0xff345700, 0xff24cab0, 0x5, 0xffbfeb60)
...
```

as time goes on, we are still in nss_search(), and the function opens a door to the NSCD daemon. doors on Solaris are like IPC in generic UNIX,
it's a communication channel between local processes via special files in the file systems. The special file of NSCD is /var/run/name_service_door

```
26613/1:        open64("/var/run/name_service_door", O_RDONLY)  = 3
```

it issues the door commands, like getting info about the other side

```
26613/1:        door_info(3, 0xFF3458D0)                        = 0
26613/1:                target=405 proc=0x2DD6C data=0xDEADBEED
26613/1:                attributes=DOOR_UNREF
26613/1:                uniquifier=268
```

... also requesting a feature. *nss*_search here will ask NSCD to handle the request, either by returning data from cache or resolve and store the result while returning it as well

```
26613/1:        door_call(3, 0xFFBFE914)                        = 0
26613/1:                data_ptr=FF1B0000 data_size=255
26613/1:                desc_ptr=0x0 desc_num=0
26613/1:                rbuf=0xFF1B0000 rsize=16384
```

*_nsc*_search (=NSCD) returns with 0, so nss_search returns 0 as well, we found the entry we received it from NSCD

```
26613/1@1:          <- libc:_nsc_search() = 0
26613/1@1:        <- libc:nss_search() = 0
```

On the same machine, I realized that if the entry I'm requesting is either in 'files' or in 'ldap', I always get a response from NSCD. If not, well it's a different story.. let's see my test.

When I request an ID which is not in 'files', nor in 'ldap', things turn weird (you can guess, this should go to 'lsass' then):

Same start, nss_search() and *_nsc*_search()

```
12669/1@1:        -> libc:nss_search(0xff345700, 0xff24cab0, 0x5, 0xffbfeb60)
12669/1@1:          -> libc:_nsc_search(0xff345700, 0xff24cab0, 0x5, 0xffbfeb60)
```

same as before, the function opens a door the NSCD

```
12669/1:        open64("/var/run/name_service_door", O_RDONLY)  = 3
```

same as before.. good so far,

```
12669/1:        door_info(3, 0xFF3458D0)                        = 0
12669/1:                target=405 proc=0x2DD6C data=0xDEADBEED
12669/1:                attributes=DOOR_UNREF
12669/1:                uniquifier=268
```

same as before.. still OK,

```
12669/1:        door_call(3, 0xFFBFE914)                        = 0
12669/1:                data_ptr=FF1B0000 data_size=255
12669/1:                desc_ptr=0x0 desc_num=0
12669/1:                rbuf=0xFF1B0000 rsize=16384
```

OPS. *_nsc*_search() returned with 5… We'll get to this soon..

```
12669/1@1:          <- libc:_nsc_search() = 5
```

What is happening? **My 'ls –la' command opens an LDAP connection?** This is ugly..

```
12669/1:        connect(5, 0xFFBFB2A0, 32, SOV_DEFAULT)         Err#150 EINPROGRESS
12669/1:                AF_INET6  name = ::ffff:3.34.169.233  port = 389
12669/1:                scope id = 0  source id = 0x0
12669/1:                flow class = 0x00  flow label = 0x00000
```

and nss_search returns with 1, which is "NSS_NOTFOUND". Fair enough, I tested on a random ID which was not in any of our databases.

```
12669/1@1:        <- libc:nss_search() = 1
```

So. Let's recap what just happened. When the account I tried to resolve was either in 'files' or in 'ldap', NSCD served me correctly, either with

return value 0 (NSS_SUCCESS) or with 1 (NSS_NOTFOUND). If I request to resolve an account which is non-existent, so it will go to 'lsass', NSCD returns me 5. Then, the application (ls, id, who, any UNIX command) will open (!) an LDAP connection to finish the name resolution!!

This is very ugly. Btw this is normal if we don't have NSCD running, because then the C library each and every time has to open an LDAP connection to query the server (and that's why NSCD is very important), but here we HAVE NSCD running ! So why it works sometimes and sometimes it doesn't?

Let's continue..

So what is return value 5?

```
NSS_TRYLOCAL = 5,
```

When NSCD can't find an entry it it's cache it will turn into the local nss_search() in libc, although under certain circumstances it will return NSS_TRYLOCAL, which will indicate to the caller's app to call nss_search() itself. That's why the app itself will open LDAP connections to query name services, when we receive 5 from NSCD. Why we received 5 from NSCD only when the resolution chain has to go through Likewise 'lsass' ?

Let's **dive deeper** :

lookup_int() is where it all roughly starts, which calls nss_psearch(), where we know that we don't have the requested data in cache, so we'll call nss_search() (this is NOT the same nss_search() as the function in the libc, this is just a wrapper which will call the libc nss_search()). Here comes the fun part, nss_search() calls _nscd_get_smf_state(), which will check

```
185     /*
186      * Foreign backend is not supported by nscd unless
187      * the backend supports the nss2 interface (global
188      * symbol _nss_<backname name>_version is present),
189      * tell the switch engine to return NSS_TRYLOCAL
190      * if needed via rc NSCD_SVC_STATE_FOREIGN_SRC.
191      */
192     if (srci >= _nscd_cfg_num_nsw_src)
193             return (NSCD_SVC_STATE_FOREIGN_SRC);
```

if the backend source is either 1) online, 2)unsupported or 3) foreign. Here is the list Solaris NSCD will consider as online :

```
nscd_cfg_id_t _nscd_cfg_nsw_src[] = {
118     {       0,      "files"                 },
119     {       1,      "ldap"                  },
120     {       2,      "nis"                   },
121     {       3,      "mdns"                  },
122     {       4,      "dns"                   },
123     {       5,      "compat"                },
124     {       6,      "user"                  },
125     {       7,      "ad"                    },
126     {       -1,     NULL                    }
127};
```

and as 'lsass' is not in the list, it will be considered as foreign backend source. And here is final piece, NSCD will force to return NSS_TRYLOCAL since 'lsass' is foreign and it is not an nss2 compatible backend (As you saw in the code description before) :

```
817             /* stop if the source is one that should be TRYLOCAL */
818             if (initf == nscd_initf &&      /* request is from the door */
819                 (smf_state == NSCD_SVC_STATE_UNSUPPORTED_SRC ||
820                 (smf_state == NSCD_SVC_STATE_FOREIGN_SRC &&
821                 s->be_version_p[n_src] == NULL) ||
822                 (params.privdb && try_local2(dbi, srci) == 1))) {
823                     _NSCD_LOG(NSCD_LOG_SWITCH_ENGINE, NSCD_LOG_LEVEL_DEBUG)
824                     (me, "returning TRYLOCAL ... \n");
825                     res = NSS_TRYLOCAL;
826                     goto free_nsw_state;
827             }
```

I spent a few hours troubleshooting this issue using DTrace, truss, and reading the kernel source code, when I found a non-documented awesome flag in /etc/nscd.conf we can use to turn super-verbose debugging on. According to the man page the debug-level can be set from 0-10, well that's far not true. If you want to turn on superb debug mode, then add these two lines into nscd.conf :

```
debug-components        2047
debug-level             32767
```

For detailed values of debug-level and debug-components, you can read the source code.

After spending hours on this issue, the super verbose mode confirmed what I just proved here, here is the log (btw it won't show you this with debug-level 10… poor me, I started with 10..) :

```
Wed May  8 13:48:57.8138--2--7943        nss_search:
                database = passwd, config = >>files ldap lsass<<
Wed May  8 13:48:57.8138--2--7943        nss_search:
                nsw source = files
Wed May  8 13:48:57.8138--2--7943        nss_search:
                looking up source = files, loop# = 0
Wed May  8 13:48:57.8139--2--7943        trace_result:
                NSS_NOTFOUND: database: passwd, operation: 5, source: files, erange= 0, herrno: Resolver
Error 0 (no error) (0)
Wed May  8 13:48:57.8139--2--7943        nss_search:
                nsw source = ldap
Wed May  8 13:48:57.8139--2--7943        nss_search:
                looking up source = ldap, loop# = 0
Wed May  8 13:48:57.8756--2--7943        trace_result:
                NSS_NOTFOUND: database: passwd, operation: 5, source: ldap, erange= 0, herrno: Resolver
Error 0 (no error) (0)
Wed May  8 13:48:57.8756--2--7943        nss_search:
                nsw source = lsass
Wed May  8 13:48:57.8756--2--7943        nss_search:
                returning TRYLOCAL ...
```

In short,

Likewise 'lsass' backend is not supported by NSCD on Solaris. As a result, whenever a query goes to 'lsass' in nsswitch.conf, NSCD will reject the request (with return value 5), so the app has to call manually nss_search() in libc, which will result an individual LDAP connection per query. Obviously another consequence is that the requested data will never get into the NSCD cache. This recommends to have 'lsass' as the last item in nsswitch.conf because this way if an entry can be satisfied with 'files' or 'ldap', it won't ever get to 'lsass' and it won't open a separate LDAP connection. If it does, NSCD will reject the query and the app has to handle the name resolution manually using the nss_search () in libc. Unfortunately if there is an non-resolvable UID in the filesystem which for whatever reason has to be resolved via NSS (Netbackup is an example which will read all the files anyway), it will trigger 'lsass' independently where we put as long as it's in the list.

Regards,
sendai

PS: I wrote a DTrace script so we can analyze any environment without reconfiguration or impact to the system. When you run the script, you have to attach it on NSCD, and it will print the queries and their result. This can show you easily which requests are receiving 5 from nss_search().

Please take a look at an example I will demonstrate on server cmfciohqapp02 :

We see that 'lsass' is after files, so anything can't be resolved via 'files' will be forced to do manual resolution via libc

```
cmfciohqapp02(ag003902): grep passwd /etc/nsswitch.conf
passwd:    files lsass ldap
```

get the PID of nscd

```
cmfciohqapp02(ag003902): svcs -p name-service-cache
STATE          STIME    FMRI
online         May_03   svc:/system/name-service-cache:default
               May_03       9310 nscd
```

and attach my script as root to that

```
cmfciohqapp02(ag003902): ./nscd_check.d 9310
```

Then, you'll see report as queries to NSCD arrive (lines beginning with # are my comments here and not part of the report) :

here I typed 'id zendai' in another window.. as there is no such user, it will go from 'files' to 'lsass', when NSCD will bounce 'id' back with return value 5 which will force 'id' to manually resolve the ID

```
2013 May  9 06:19:57 name to resolve : zendai in passwd
2013 May  9 06:19:57 name to resolve : zendai in passwd
```

here we go, NSCD even though received the request for the passwd database, it rejects it with 5

```
2013 May  9 06:19:57 not in cache, NSCD query for zendai @ passwd returned 5 from nss_search()
```

If you run the same experiment on a server without 'lsass' in /etc/nswitch.conf, NSCD will perform the lookup and will return 1 for NSS_NOTFOUND. In this case though, NSCD rejects to perform the lookup.

That's it !