From: Spitzer, Andras ([redacted])
Sent: Monday, June 17, 2013 11:54 AM
To: [redacted] ([redacted])
Cc: [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted])
Subject: NSSLTOF package

Bob, Jon, Nate,

You guys know how much time we spent on optimizing our LDAP infrastructure, both on client and server side, and **two of the major conclusions** were that

1. most **NSCDs will** eventually become unreliable/unresponsive/**crash** on both Linux and Solaris, and even when it keeps running still there will be NSS calls that remains uncached
2. likewise **'lsass' module made this** already troubled caching mechanism **even less efficient**, because it's not NSS2 compatible

I went through on many solutions out there, unscd (not native to Linux's libc so incompatibilities may occur), nslcd (will only keep connections open what NSCD already does – true, if it's running, still won't help caching entries), and none of them looked promising, except one that I found interesting by concept. This one was called NSSCACHE from Google, which means we are not the first enterprise company to face this type of problem. They pretty much came to the same conclusion, true though their scope was Linux only. They also summed up the reasons why they had to look for a solution, and even though Solaris is a tiny bit more reliable, overall I can say both Linux and Solaris suffers from NSS lookups in LDAP.

NSSCACHE doesn't run on Solaris for many reasons, still I loved the idea so I developed a Solaris 10+ SPARC version of the same concept, and I even designed it to be native to Solaris for simplicity. I wrote NSSLTOF from the scratch , also the approach is different in many ways, I only kept the basic concept, true though next to the basic concept I added two more 1. should run on all the different Solaris 10 update versions 2. should be simple enough for mass deployment. That's how **NSSLTOF was born** (sorry about the name).

The **concept was** the following :

1. let's create local cache files from LDAP using a tool
2. use those local cache files for NSS lookup

Using **this concept our NSS calls will never contact LDAP directly, so not even the likewise issue will impact our LDAP servers** anymore. The package consist of two parts, **one is the sync daemon (named nssltofd)**, **two is the NSS module (named issc) which is NSS2 compatible**, so NSCD can cache it's results. As a result, **this package will cut 98% of the LDAP traffic in a server, including Likewise enabled ones**. The package also contains documentation, man pages, etc.

The steps to install the package (roughly takes about 1 minute with the default settings) :

1. remote **install** the package using the URL **http://lonperf01.corporate.[redacted]p/ISSnssl2f.pkg** (the package will not overwrite, nor will modify any existing file, totally safe to install)
2. **enable t**he sync daemon with svcadm
3. **replace** the *ldap* keyword with *issc* at 'passwd:' and 'group:' lines in /etc/nsswitch.conf (only 'passwd' and 'group' are supported !)
4. run the **test** suite of : id, groups and su – with your admin account which originally came from LDAP, now will come from issc

+1 if, for any reason you want to rollback, just 1. disable the SMF service, switch the 'issc' keyword back to 'ldap' in nsswitch.conf, and run 'pkgrm ISSnssl2f' !

NOTE: There is one caveat, in Solaris 10U5 (Generic_127XXX-XX) and earlier versions the pam_unix_auth.so.1 PAM

module was not really 3rd party friendly, which means that if you install NSSLTOF please make sure that the "auth" line for "pam_unix_auth.so.1" is set to *sufficient*, instead of *binding*. If it's set to *binding*, even if the pam_radius.so.1 Succeed later in the PAM chain the authentication will be denied for users in the NSSLTOF cache file. The reason is that those old pam_unix_auth.so.1 modules mistakenly will try to authenticate against any 3rd party NSS module including issc, and obviously it'll receive Authentication failure (issc does not store passwords). If the control flag is binding, the Authentication failure will be the total result, even if a later module like pam_radius.so.1 Succeed. Changing it to sufficient will let the later module override the Authentication failure, so the users in issc are able to log in. Fortunately only less than 10% of our Solaris 10 servers are 10U5 or older.

Solaris 10U6 and later pam_unix_auth.so.1 simply ignore 3rd party NSS modules, which is the correct way to do.

An **example** (the host is Solaris 10u6 or later):

Step #1, Install the package

```
cmfalgadetl03# pkgadd -d http://lonperf01.corporate.[redacted]p/ISSnssl2f.pkg
## Downloading...
.............25%.............50%.............75%.............100%
## Download Complete


The following packages are available:
  1  ISSnssl2f     NSSLTOF
                   (sparc) 0.5

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]: 1

Processing package instance <ISSnssl2f> from
<http://lonperf01.corporate.[redacted]p/ISSnssl2f.pkg>
…
Installation of <ISSnssl2f> was successful.
```

Step #2, enable the sync daemon

```
cmfalgadetl03# /usr/sbin/svcadm enable application/iss/nss-ldap-to-files
cmfalgadetl03# svcs application/iss/nss-ldap-to-files
STATE          STIME    FMRI
online         12:16:09 svc:/application/iss/nss-ldap-to-files:default
```

Step #3, Replace the 'ldap' keyword at passwd and group lines with 'issc' in /etc/nsswitch.conf

```
cmfalgadetl03# egrep "^(passwd|group)" /etc/nsswitch.conf
passwd:    files lsass issc
group:     files lsass issc
```

Step #4, Run the test commands using an account known to be an LDAP account (like our sysadmin accounts)

```
cmfalgadetl03# id ag003902
uid=11638(ag003902) gid=9999(superusr)
cmfalgadetl03# groups ag003902
superusr
cmfalgadetl03# ssh ag003902@localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
RSA key fingerprint is b0:47:99:d7:b8:86:ec:2b:56:f6:ea:35:25:69:69:a9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Password:
Last login: Sat Jun 15 14:10:18 2013 from 3.48.136.50
Oracle Corporation      SunOS 5.10      Generic Patch   January 2005
```

```
$ echo "hello world"
hello world
```

Bob, Nate, please try NSSLTOF and let me know if you like it, if you find this package a viable option, then I recommend installing it first on a few of the top LDAP Solaris 10 heavy hitters. If any of you guys would like to know more about the technical background of the package, or you just want to have a Q&A session, please let me know and I'll set up a call where we can discuss these questions. Also, if we like the concept, we could also evaluate deploying NSSCACHE for Linux, or at least on the LDAP heavy hitter hosts.

Regards,
sendai

PS: even though I have a lot more to write about this package, I wanted to keep it relatively short, here though I'll share **some techie details** in case you are interested.

How the sync daemon, nssltofd works

The daemon nssltofd will periodically sync objects from LDAP to local cache files. The cache files are named /etc/passwd.cache, /etc/group.cache and /etc/shadow.cache, today only *passwd* and *group* are supported, true though that should cover ~98% of a server's LDAP calls. The base directory of the package is /opt/ISSnssl2f, and there is a configuration file /opt/ISSnssl2f/etc/nssl2f.conf where you can tune the daemon, setting which database you want to sync and what type of sync you want, incremental, full, or mixed. The default values should be OK for us though, I set mixed (incremental updates every 6 hours, and a full update once a day), you'll also find man pages for the issc NSS module as issc(1), the nsltofd daemon as nssltofd(1M), and for the configuration file as nssl2f.conf(4). nsltofd is registered in the SMF framework, and is depending on ldap_cachemgr, so it won't start in single user mode.

The mixed update method works the best, since the incremental updates are using less resource on the LDAP server than the full update, though we need to have full updates to pick up the user deletes. I also added a fun feature called "randomWindow", which will delay the sync with a random amount of seconds set by this attribute. The idea behind that was I didn't want hundreds of servers sync at the same time cause a huge peaks at the LDAP server, using this attribute, even if we set it to a fairly low number (I recommend to set between 5 – 300 seconds), the individual servers will spread the sync start time as much that the LDAP servers won't have to deal with sync peaks. That was also my motivation of writing a daemon instead of running from cron, from cron they would run roughly the same time and I wanted to avoid that.

How issc, the NSS module works

The NSS cache files are having the exact same format as their original files, passwd.cache has the same format as passwd, etc. and 'issc' is a very lightweight NSS module, because the smart part was that I used the recursive dynamic symbol resolution to my advantage, so basicall issc is just a frontend module dynamically linked with nss_files, and when forward the constructor function to the original nss_files modules I modify the returning object by rewriting the filenames from their original name to original.cache, like rewriting passwd to passwd.cache, but everything else is done by the nss_files module still. There are many advantages to this since I don't have to worry about the nss_files internal logic, I don't even care about that, I just tell the module which files to use. This concept is similar to the object oriented concept of encapsulation, where you don't know neither you care how the object works, you just set the parameters and call the function to do the job. That's the concept I used here as well for issc, the package contains issc's source code in case you are interested, under the dir 'misc'.

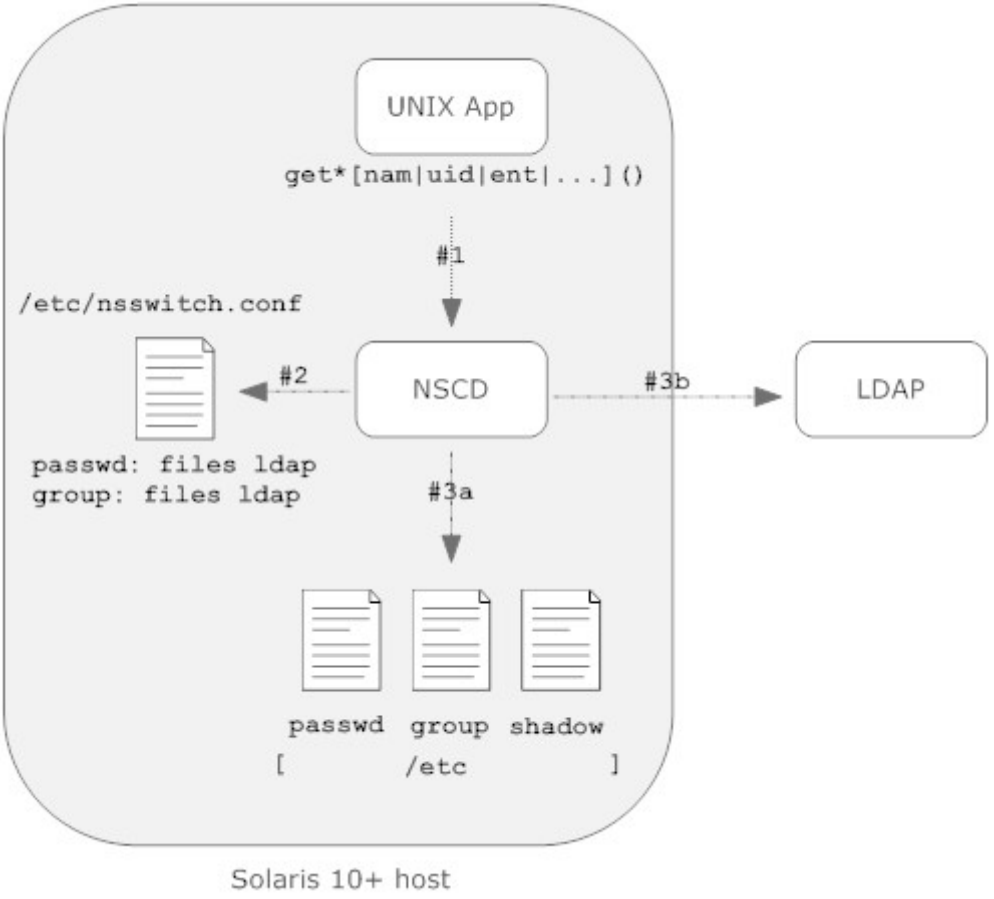
Also there is a log file /opt/ISSnssl2f/log/ltof.log where you can check the activity of the nssltofd daemon, also you don't have to worry about its size because it'll be auto-truncated based on the size limit set by the config file.

How we get the data from LDAP

nssltofd is using the native ldaplist command to query the LDAP. The great thing about that is nssltofd doesn't have to deal with the LDAP connection/configuration part, nssltofd doesn't have to know the credentials to access the LDAP server, neither it has to worry about redundancy, because ldaplist is using the internal ldap_cachemgr resulting a very clean and simple daemon, and simplification always means less possible point of failures and mistakes. The daemon though reads /var/ldap/ldap_client_cred to scan for custom attribute mappings. By default it'll use the ones defined

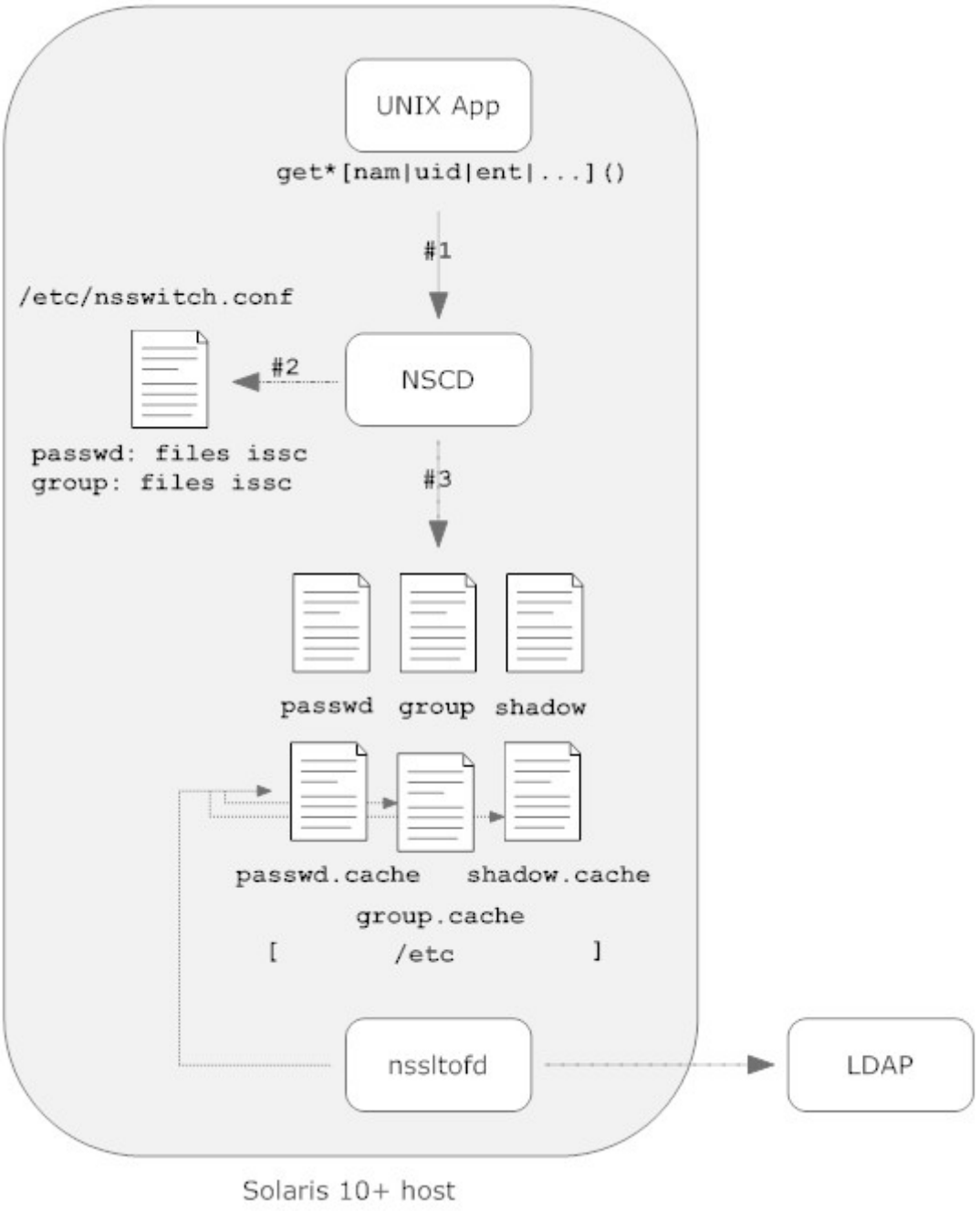
by the RFC, and it will override the defaults if there are custom definitions found natively in /var/ldap/ldap_client_cred.

Drew a picture to show how NSS lookup works with *files* and *ldap* :



The UNIX App tries to resolve a name, an uid, gid, username, groupname, etc. (#1) via the get* functions. The get* functions are all libc functions, and will connect to NSCD via door(), which after reading /etc/nsswitch.conf (#2) and found the entries *files ldap*, will contact the *files* (#3a) and *ldap* (#3b) for getting the data.

Now, here is a graph how NSS lookup works with *files* and *issc* :



As you can see here NSCD will never contact LDAP directly again, but after reading through the generic passwd, group, files local files, it will read the cache files as well, which are periodically updated by nssltofd. And the good part is that this is the flow even if NSCD dies or if it's unable to cache, as with Likewise. That's why this can cut the ~98% of the LDAP traffic on servers, including the ones with Likewise installed. Again, guys if any of you are interested in the details, I'm more than happy to set up a Q&A call about this.

Regards,
sendai